by Dick Cappels

Minimum Mass Wireless Coupler

Dick's minimum mass wireless coupler allows you to connect the devices on your workbench to a PC. No electrical connections are required. You can use an AT90S2313 or an ATmega8 to control the system.

Let's examine a way to get an extremely short-range bidirectional radio frequency (RF) link for a microcontroller using the minimum number of controller resources and external components. Using an AT90S2313 or ATmega8, for example, my minimum mass RF coupler requires only the on-chip comparator, the 8-bit timer, and their associated interrupts to send and receive data at 1,200 bps. In the simplest implementation, the only necessary external components are a loop antenna, a resonating capacitor, and two resistors. If you use a 5.5-cm loop antenna, the reliable range is between 10 and 15 cm. The AT90S2313's driver requires only 129 words of code.

The coupler allows you to connect instruments on your workbench to your computer and to one another without an electrical connection. This is more than convenient; it gets around grounding problems and allows you to float instruments above or below ground.

To date, I've equipped a frequency meter, a 2 × 16 display, a scanning digital voltmeter, and an RS-232 adaptor with minimum mass RF couplers. The coupler also can be used for a variety of other bidirectional wireless identification and data communications tasks for which you might otherwise consider optical or RFID technologies. Although I've only tried this on AVR controllers, I suspect it would work well with other controller families (e.g., PIC).

SIMPLE CIRCUIT

To transmit, a resonant loop antenna converts signals from the controller's output pin to a time-varying magnetic field. When receiving, the same resonant loop antenna converts received time-varying magnetic fields to a voltage that's detected by the controller's on-chip analog comparator. Because the comparator shares a pin with a tristatable I/O port, the pin can be shared between the transmit and receive functions (see Figure 1).

A pair of resistors biases the circuit at half the supply voltage. This is the sweet spot of the comparator's Common mode input range, where the input offset voltage is minimum and results in the greatest receiver sensitivity. This is also nice for Transmit mode when sending the 181.818-kHz carrier because the zero to V_{CC} swing on the output pin results in symmetric drive to the coil. The values of the resistors set the maximum drive current in Transmit mode, but they have little effect on receiver performance.

I chose the 181.818-kHz carrier frequency because it's a sub-harmonic of the microcontroller's 4-MHz instruction rate and it's within a license-free band in the U.S. It's likely that this implementation is acceptable for license-free operating in most legal jurisdictions because of the low transmitter power and small antenna size. Check local regulations to be sure.

TRANSMISSION

The transmit routine shifts out the byte to be transmitted, one bit at a time, first shifting out a start bit (a one) then one data bit every 833 ms, which corresponds to 1,200 bps. Two stop bits (zeros) are then sent. If the bit being sent is a zero, the tristatable pin connected to the resonant loop antenna (pin 12 in Figure 1) remains

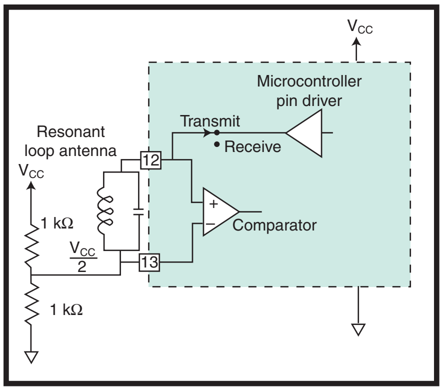

Figure 1—*Take a look at the tristateable I/O pin in the transmit/receiver circuit. Pin 12 switches between input and output states. Pin numbers relate to the AT90S2313 DIP package.*

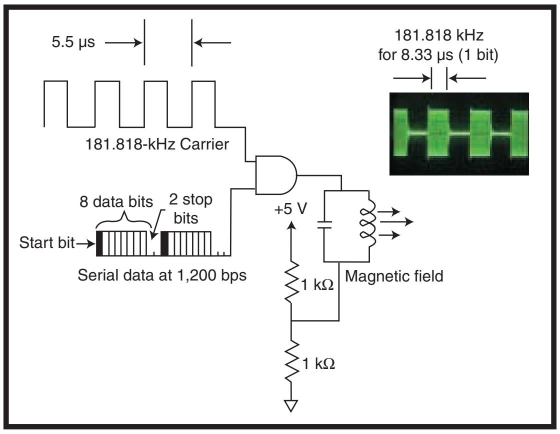

Figure 2—*The transmitter sends on/off-keyed serial data by generating a 181.818-kHz carrier for a logic 1 and no carrier for a logic 0. The timing corresponds to 1,200 bps.*

as an input during the bit period. If the bit being sent is a one, the tristatable pin connected to the resonant loop is set to an output and the controller generates 181.818-kHz square waves on the pin, thus transmitting a carrier, for one bit time (see Figure 2).

Getting a clean carrier out of the microcontroller while timing one bit period requires the use of the 8-bit timer and its interrupt so the processor only spends time generating the carrier. Listing 1 is the pseudocode routine that generates the 181.818-kHz carrier. The carrier routine is called as a subroutine after setting the 8-bit timer to interrupt it after one bit time. The corresponding interrupt routine that's executed after the carrier is sent for one bit time is a short one (see Listing 2). The pop instructions reposition the stack pointer so that when the return from interrupt instruction is executed, it returns to the routine that called the 181.818-kHz carrier generation routine. A similar method is used when sending a logic 0, but a signal isn't generated on the output pin.

RECEPTION

The receiver is functionally similar to a sensitive frequency meter connected to a resonant loop antenna (see Figure 3). When the receive routine checks for a logic 1 (the presence of a carrier), comparator interrupts are enabled and an 8-bit timer is set to terminate the routine that checks for the presence of a carrier a quarter bit time later.

The comparator interrupt routine counts the number of interrupts during this quarter bit time. (Some counters can do this directly in hardware.) Thus, after a quarter bit time has passed, the register containing the bit count is a measure of the frequency of the signal received by the resonant loop antenna. If it corresponds to 181.818 kHz within preset limits, then the firmware considers the carrier to be present and assumes that the transmitter has sent a logic 1. If the comparator interrupt count is too low, the assumption is that there is no carrier present. If the count is too high, the assumption is that the interrupt source is noise or an interfering signal.

To decode the incoming data, the firmware looks for a one, which indicates a start bit. If a one is detected, it waits until the center of the following bit, and then samples each bit and shifts it into a byte-wide register until all 8 bits have been shifted in.

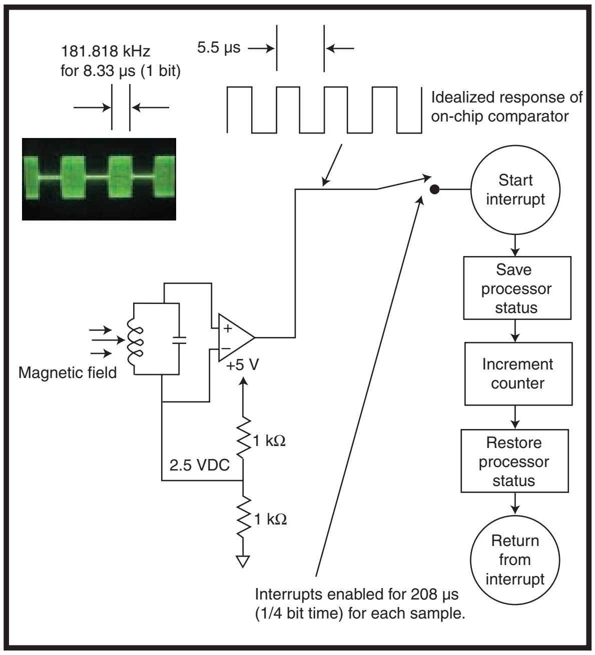

Figure 3—*The receiver counts the number of times the positive slope of the waveform across the resonant loop exceeds the comparator threshold within a quarter bit period during each sample. If the count is within the specified limits, it's determined that the carrier is present and it's interpreted as a logic 1.*

FIRMWARE

The firmware specific to the minimum mass wireless coupler is contained in an assembler include file, which includes detailed instructions and an example application. I've written include files for the AT90S2313 and the ATmega8 (both operating at 4 MHz). The include files only reference one register by name, RFChar, which is the register by which the incoming and outgoing byte are transferred. The rest are referenced directly by register name, so there won't be a "Register Already Defined" warning message during assembly that results from the use of some registers by both the main program and the include file.

The minimum mass wireless coupler routines save the status register and the contents of all working registers they use except RFChar. When returning to the calling program, the status register and all the working registers except RFChar are restored.

The main program needs to take care of some housekeeping. It assigns both the RFChar variable and the I/O for the transmit pin, which could be separate from the receive pin if you want to use separate antennas or a more complicated circuit (e.g., one with a high-power output stage or receive preamp). The main program also handles the activity LED, the pins associated with analog

Listing 1—*To generate a symmetric square wave on the output pin, the pin is held high for the same amount of time that it's held low. To compensate for the two clock cycle jump instruction, the delay after setting the output pin low is two clock cycles shorter than the delay after setting the output high.*

```
forever:             ;Make 181.818-kHz square waves on output pin.
set output bit high
delay
set output bit low
delay
jump to forever
```

Listing 2—*The pop instructions remove the 2-byte interrupt return address from the stack. They're thrown away and aren't used. When the "return from interrupt" instruction is executed, the program flow returns to the routine that called routine that was interrupted by the interrupt timer. This is how the square wave generation loop is preempted.*

```
timer0service:
pop r18         ;Pop interrupt return address
pop r18
reti            ;Return from interrupt
```

comparator input 0, and a pin to switch power to the bias resistors.

There are four subroutines in the include file that can be called. `SendRFByte` sends the contents of `RFChar` via the minimum mass wireless coupler. The `ReceiveRFByte` subroutine waits for a start bit on RF channel for 63.75 bit times. If a start bit is found, the entire incoming byte is received and placed in `RFChar` and the carry bit is set. If a start bit isn't found, the routine returns with the carry bit cleared. This allows polling for incoming data without hanging up the receiver.

Listing 3 is an example. Because switching from transmitting to receiving requires a little time for the analog components to settle, some delay routines are necessary. The subroutines that generate these delays are `PostXmitDelay` and `PostRCVDelay`. The former is called when switching from transmitting data to receiving data. The latter is called when switching from receiving data to transmitting. Its purpose is to wait for the unit on the other end to become ready to receive.

Listing 4 is a short program that correctly uses all of the routines that can be called. I've used this program many times while testing pieces of the minimum mass wireless coupler. It receives a character over the coupler and echoes it back to the sender. While a character is transmitted or received, the pin associated with the activity LED goes high.

Listing 3—*The `ReceiveRFByte` routine returns after sampling the loop antenna output for 63.75 bit times and returns with the carry bit set only if a suspected valid character is received. This snippet waits until a valid character is received before continuing. It is analogous to monitoring the Data Received flag in a UART.*

```
getchar:
    rcall ReceiveRFByte   ;Wait for character to be received
    brcc getchar          ;If carry bit is clear, go back and
                          ;wait some more.
```

Listing 4—*This short program uses all of the routines in the include files that can be called. It retransmits each received character, which makes it a handy tool for some kinds of testing.*

```
EchoTest:
rcall   ReceiveRFByte  ;Wait for a character to be received.
brcc    EchoTest       ;If nothing is received, go back and check
again.
rcall   PostRCVDelay   ;Wait for far end transceiver to recover.
rcall   SendRFByte     ;Echo character to far end.
rcall   PostXmitDelay  ;Wait for local transceiver to recover.
rjmp    EchoTest       ;Go back to top and get another character.
```

ANTENNAS

Two of the four external components in the minimum configuration form the resonant loop antenna. The antenna's design dominates the transmitter's strength and the receiver's sensitivity.

A loop antenna can take many forms, although all the antennas I used incorporate an inductance of approximately 23 µH resonating with a 0.033-µF capacitor. The frequency meter and scanning voltmeter use a coil of 14 turns of #30 enameled copper wire wound on a 5.5-cm coil form (the bottom of a yogurt cup). It's then slipped off the form and held in the shape of a circular loop with tape. The antenna on the RS-232 base unit is rectangular because winding it around the perimeter of the circuit board gave the largest practical size and hence the maximum range (see Photo 1). In the 2 × 16 display, the RF coupler was retrofitted to an existing breadboard. Space was at a premium, so I wound the antenna with 29 turns of #30 wire on a 3-cm ferrite rod. The rod had a slightly shorter range than the circular and rectangular loop antennas, but it was worth it for the savings in circuit board space.

Loop antennas are directional. The best arrangement is to place them so that they are parallel. This gives the greatest error-free range. Antennas that are positioned at right angles to one another can't communicate because there isn't effective coupling between them in this orientation.

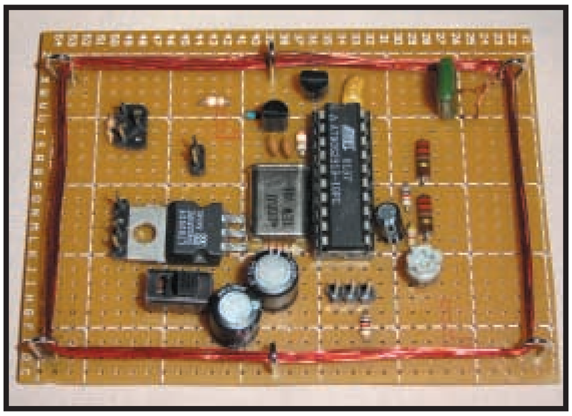
Photo 1—*The antenna for the base unit is the largest that fits neatly around the outer edge of the circuit board. Having the controller circuit inside the loop had little noticeable effect on the range.*

Magnetic antennas can be shielded from electrical interference. You can reduce the noise pickup by covering the antenna with grounded conductive foil. Also be careful not to create a shorted turn in parallel with the loop.

RS-232 BASE UNIT

The base unit, which is the larger enclosure in Photo 2, allows my computer to communicate with coupler-equipped devices using a terminal emulator program. Consequently, the base unit is basically an RS-232 interface and minimum mass wireless coupler. The RS-232 interface to the computer is 9,600 bps, 1 stop bit, no parity. That's plenty fast to keep up with the 1,200-bps maximum data rate from the coupler.

The antenna is made of 12 turns of #30 enameled wire threaded through metal guide loops arranged around the perimeter of the board (see Photo 1). This antenna's area is about 160% that of the prototype 5.5-cm loop, so it

Photo 2—*A battery-operated frequency meter (left) only needs to be near the base unit in order for them to communicate. The glowing yellow and orange LEDs indicate that data are being transferred between the two.*

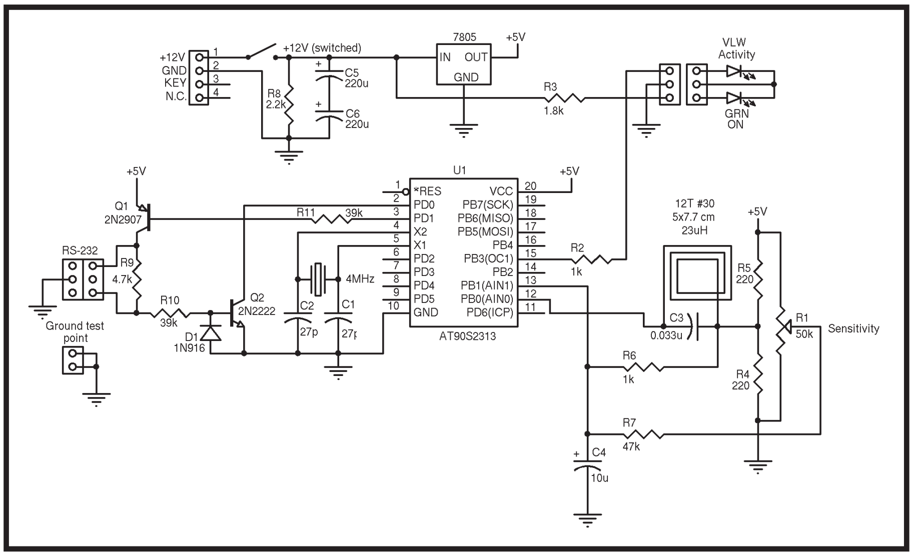

Figure 4—*The base unit includes an activity LED that glows when data is being transmitted or received. A weak pull-up resistor inside the controller pulls up the 2N2222's collector.*

has a little more range for transmitting and receiving.

To further increase the transmitter's range, I used 220-Ω bias resistors (see Figure 4). This sets the maximum peak current from the microcontroller's output pin at approximately 23 mA, which is safely below the maximum current specified on the microcontroller's datasheet. Because the antenna is resonant, the peak antenna current is greater than the microcontroller's drive current. It's approximately 70 mA peak-to-peak in this case.

The base unit uses a trick to increase the receiver's sensitivity. I added an adjustment to compensate for some of the comparator's offset. The smaller the offset voltage, the smaller the signal needed from the antenna to trip the comparator. Adjusting R1 causes a voltage drop across R6 of up to ±60 mV. This effectively compensates for offset voltages across the comparator input up to the maximum shown in the controller's datasheet.

Increasing the sensitivity can be a good thing, but too much can be a problem. When the offset is adjusted to nearly zero, noise from the microcontroller causes the receiver to repeatedly detect false signals. I've also noted that the base unit picks up interference when I put it too close to the backlight power supply in my notebook computer. Not only does the RS-232 output send out garbage, the noise keeps the desired signals from being properly decoded. The solution is to back off on the sensitivity adjustment to the point that the receiver is stable.

The base unit, which operates continuously for days or weeks at a time from an AC power adaptor, doesn't have an accessible power switch. Resetting it manually would be inconvenient, so the on-chip watchdog timer is used to catch hang-ups in the firmware. The main routine in the firmware checks the RS-232 interface to see if there is anything to send out via the coupler. It then checks the coupler to see if there is incoming data to receive and send out the RS-232 interface. Following this, the watchdog is reset. If for any reason the controller doesn't return from one of the two communications tasks, the watchdog will reset the controller.

WIRELESS FREQUENCY METER

A simple one-chip frequency meter was the first instrument I updated with my coupler. The frequency meter is the smaller enclosure in Photo 2. It's self-contained (including batteries). The only external electrical connection is a pair of wires to connect to the signal being measured. To use it, I merely attach the input leads to the signal I want to measure, place the meter next to the base unit, and switch it on. The activity LEDs blink on both enclosures as the frequency meter sends its command menu. When the menu prints on my computer screen, the system is ready for taking measurements.

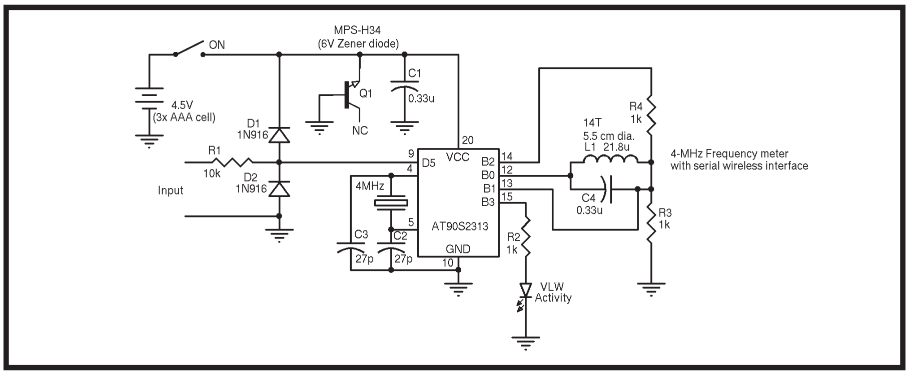

Figure 5—*The only external wires for this frequency meter are for the signals to be measured.*

After the frequency meter sends its command menu, it disconnects from the RF channel. In other words, it doesn't respond to commands or send data until it receives an ASCII ctrl+A character, which is its reconnection command. A ctrl+B character from the keyboard disconnects the meter from the channel again. Other devices use different control codes. For example, the scanning voltmeter uses ctrl+C to connect and ctrl+D to disconnect. This discipline allows multiple coupler-equipped devices to share the same base unit without interfering with each other.

The circuit in Figure 5 uses the simplest possible implementation of the minimum mass wireless coupler. As long as the input signal swings through the chip's logic switching thresholds, it will be counted. As the battery voltage drops lower and lower, the 1N916 protection diodes will keep the input pin on the controller from exceeding the maximum current shown on the datasheet. A further protection is the upside-down transistor connected as a low-current Zener diode, just in case a large signal is applied to the input leads when the power is off.

A battery saver shuts down the controller and power to the bias resistors if the circuit doesn't receive a command for a long period of time. This feature has saved me from having to replace the batteries countless times already.

EASY START

Files available on the 'Net provide code to get you started with the AT90S2313 or ATmega8. With these, you can probably whip up an implementation for a different controller without too much trouble. What the minimum mass wireless coupler has going for it are its simplicity and low cost. The only unusual part needed is the antenna coil, which is easy to make and easily adapted to various form factors. You can download AVRStudio include files for the AT90S2313 and ATmega8, source code for the RS-232 base unit, the frequency meter, and the scanning voltmeter along with the schematic of the voltmeter. ■

Dick Cappels enjoys tinkering with and writing about analog circuits and microcontrollers. He has worked with video circuits, computer displays, color management, and user I/O. You can reach Dick at projects@cappels.org.

PROJECT FILES

To download the code, go to ftp.circuitcellar.com/pub/Circuit_Cellar/2005/175.

SOURCE

AT90S2313 and ATmega8 MCUs
Atmel Corp.
www.atmel.com

RESOURCES

Atmel Corp., "8-bit AVR Microcontroller with 2K Bytes of In-System Programmable Flash: AT90S2313," 0839I, 2002.

———"8-bit AVR Microcontroller with 8K Bytes of In-System Programmable Flash: ATmega8," rev. 2486DAVR, 2002.

Code of Federal Regulations, Title 47, Volume 1, Section 15.217, "Operation in the band 160– 190 kHz," U.S. Government Printing Office.