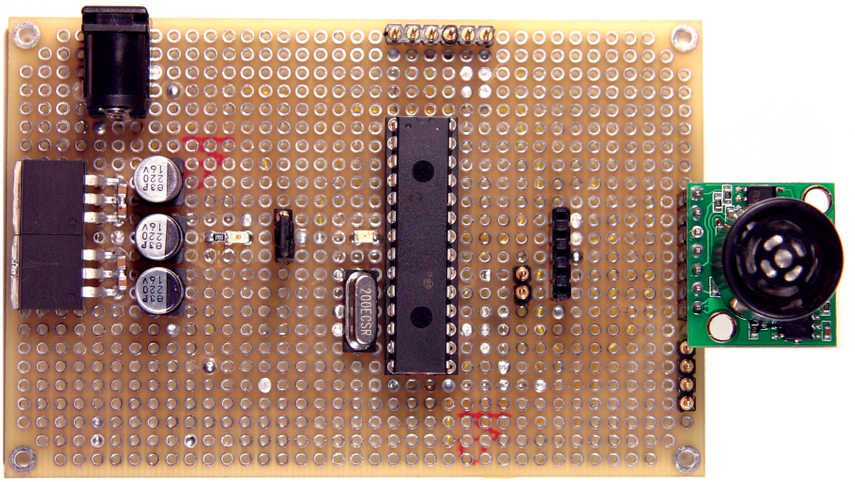
Building a Sonar System

I've always wanted to do an ultrasonic ranging project. So, guess what we'll be talking about and building up this month? Ultrasonic ranging is a great way to add eyes to your mechanical animal. I'll bet you didn't realize that there is an off-the-shelf ultrasonic ranging product out there that allows you to tune those "electromechanical eyes" to your robot's environment.



by Fred Eady

If you scan the pages of *SERVO* carefully, you'll come across a company called MaxBotix. They produce a line of ultrasonic ranging devices that take the hassle out of implementing a viable ultrasonic ranging application. Each of MaxBotix's LV-MaxSonar rangefinders has a differing beam pattern that results in a unique detection pattern. This allows you — the ultrasonic ranging system designer — to pick an LV-MaxSonar that is right for your application.

Ultrasonic rangefinders with wide beam widths are better suited for "eye" applications. An eye application may need to detect obstacles, avoid collisions, or sense the presence of a humanoid. Wide beams also are very good at detecting small objects due to their higher sensitivity.

MaxBotix offers LV-MaxSonar ultrasonic rangefinders that produce a narrow beam. These narrow beam rangefinders are good for ranging and room mapping. A narrow beam LV-MaxSonar rangefinder will do a better job at operating in cluttered and "high noise" environments as its beam is a bit less sensitive in the center. However, you can use a narrow beam ultrasonic rangefinder to do the work of a wide beam ultrasonic rangefinder if that's what is right for your application.

The LV-MaxSonar ultrasonic rangefinder line is composed of five models. As you can see in Figure 1, the EZ0 is the wide beam ultrasonic rangefinder model and is the most sensitive. The EZ1 emits a narrower beam than the EZ0, which makes it more suitable for sensing humans. The MaxBotix LV-MaxSonar-EZ2 produces a beam that is even narrower than the EZ1. The beam width narrows progressively from

FIGURE 1. This illustration makes it easy to comprehend the differing detection patterns of the line of MaxBotix LV-MaxSonar ultrasonic sensors. The really cool thing is that all of the LV-MaxSonar sensors have the same ranging data interface. This set of lobe shots demonstrates the ranging of various diameter dowels on a one foot grid.

LV-MaxSonar®-EZ beam patterns	EZ0™	EZ1™	EZ2™	EZ3™	EZ4™
Detection pattern to a 1/8 inch diameter dowel.					
Detection pattern to a 1/4 inch diameter dowel.					
Detection pattern to a 1 inch diameter dowel.					
Detection pattern to a 3 1/4 inch diameter dowel.					

–5V
•3.3V
V+ supply voltage.
(Distances overlaid on a 1 foot grid.)

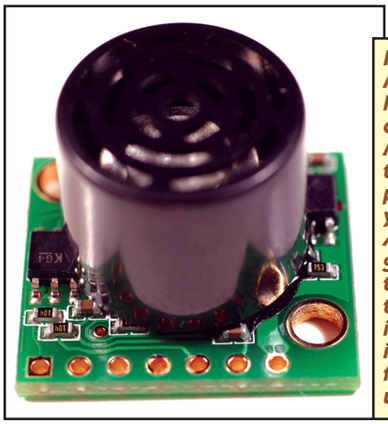
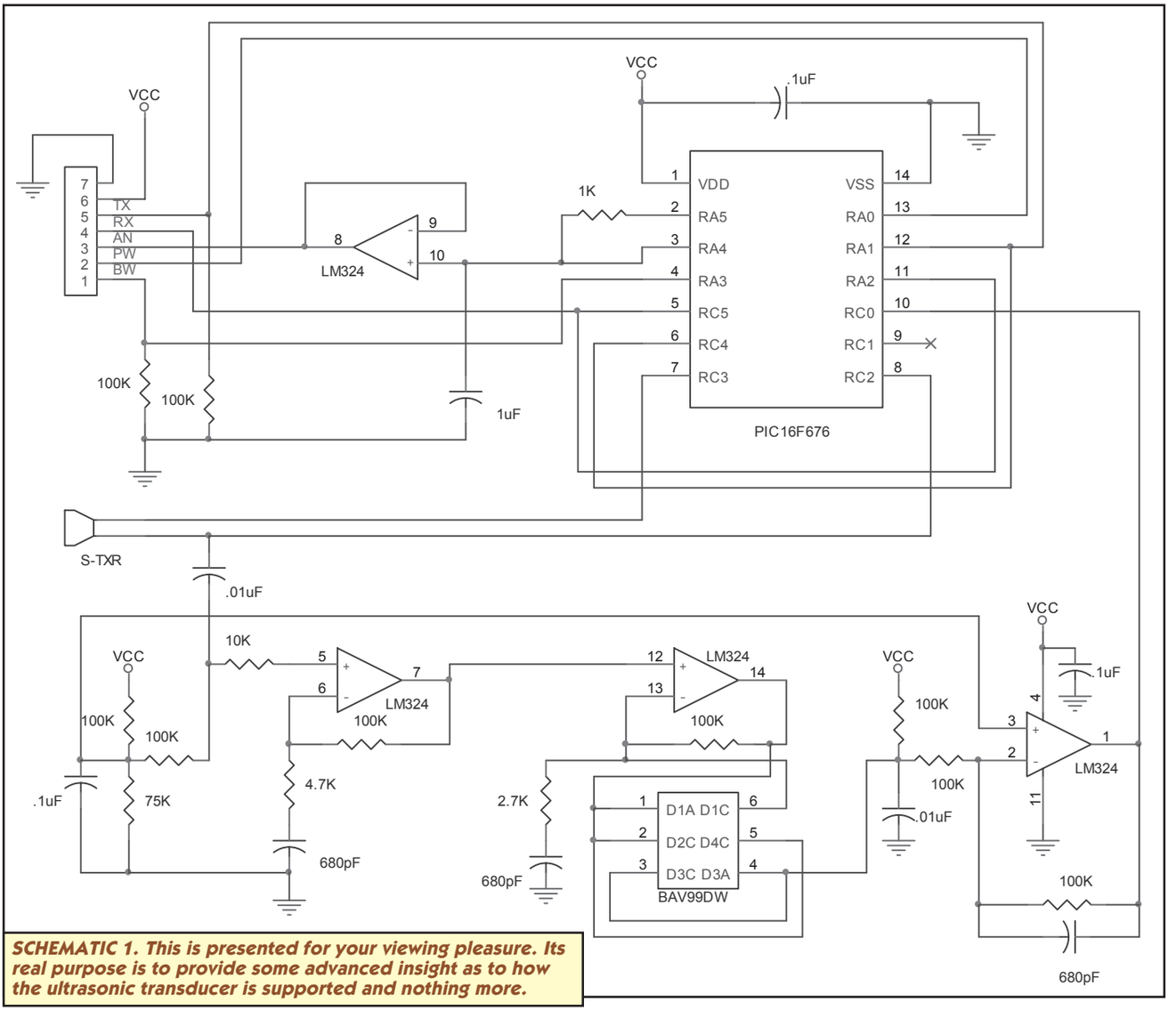


PHOTO 1. The LV-MaxSonar-EZ0 you see here is a combination of the MaxBotix MaxSonar-UT ultrasonic transducer and the proprietary circuitry you see in Schematic 1. Another reason for not getting too deep with the inner workings of the EZ0 is that there is no technical information available from MaxBotix for the ultrasonic transducer.



SCHEMATIC 1. This is presented for your viewing pleasure. Its real purpose is to provide some advanced insight as to how the ultrasonic transducer is supported and nothing more.

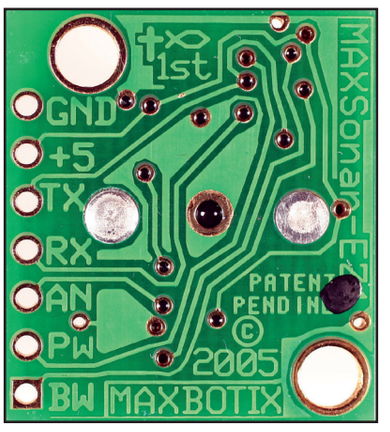
the LV-MaxSonar-EZ0 through to the LV-MaxSonar-EZ4.

I'm anxious to begin our MaxBotix sonar project. However, before we can run the range, we have to learn how to ride.

The MaxBotix LV-MaxSonar Ultrasonic Rangefinder

I seriously considered removing the MaxSonar-UT ultrasonic rangefinder from the EZ0 you see in Photo 1 so that you could see its circuitry. Rather than taking a chance on ruining my only one, I decided to show you what the circuitry looks like schematically. Take a look at Schematic 1. The EZ0 is under the control of a PIC16F676. Since we don't really know what the PIC is doing programmatically, we can only take a guess as to what the supporting circuitry is doing. If you take a look at the MaxBotix forum, you'll see that some students have attempted to simulate the circuit you see in Schematic 1. That's nice. However, MaxBotix wants to keep their secret formula for ultrasonic sensing under wraps. I respect that

PHOTO 2. It doesn't get any better than this. Each LV-MaxSonar-EZ0 I/O pin is clearly marked. The black dot is a color code that identifies this unit as an LV-MaxSonar-EZ0. The LV-MaxSonar-EZ4 has a yellow dot. I'm always interested in looking at printed circuit board land patterns as you never know what you may find. Can you find other "messages" in this image?



and that's all we'll say about the circuit shown in Schematic 1. After all, we're only interested in putting the EZ0 to work.

All of the ultrasonic engineering has been done for us by the MaxBotix engineers. All we really have to do to bring the EZ0 online is to apply some power and follow some very simple operational rules. The LV-MaxSonar series of ultrasonic rangefinders can be powered by voltages as low as 2.5 volts and as high as 5.5 volts. This power rail range allows the LV-MaxSonar ultrasonic rangefinder family to work with 3.3 volt systems, which are gaining popularity due to their lower power consumption characteristics.

These ultrasonic rangefinders draw approximately 3.0 ma of current when powered by a 5.0 volt power source. When powered by 3.0 volts, the rangefinders draw only 2.0 ma of current. That kind of current consumption allows the LV-MaxSonar ultrasonic rangefinders to easily operate in battery powered mobile systems.

As you can see in Schematic 1, the EZ0 (and all of the other LV-MaxSonar ultrasonic rangefinders) interfaces to the outside world with five I/O lines and two power connections. A physical look at the EZ0 interface can be seen in Photo 2. Let's walk through each line of the I/O interface.

Pin 1 is labeled "BW." This pin is used when multiple ultrasonic rangefinders need to be triggered. If triggering multiple rangefinders is not part of your application, you must tie the BW pin low or leave it open. Otherwise, holding the BW pin logically high will force the EZ0's TX pin to produce a pulse instead of serial data. The pulse is used to trigger other ultrasonic rangefinders in the rangefinder network. An initial seed pulse to the RX pin of the first LV-MaxSonar ultrasonic rangefinder in the chain is all that's needed to fire off the rest of the ultrasonic rangefinders behind it in the chain.

"PW" marks pin 2 of the I/O interface. When the EZ0 is ranging, the PW pin will emit a pulse that is relative to the distance to the target object. The ranging pulse is defined as 147 µs per inch.

If measuring pulse widths is not something your host microcontroller will do easily, you can opt to receive your ranging information from the EZ0's AN pin. However, your microcontroller will need to have an on-chip analog-to-digital (A-to-D) converter subsystem to capture the AN pin's output. As you've probably deduced, the AN I/O pin provides an analog voltage that is relative to the distance to the target object. The distance is calculated as Vcc/512 volts per inch. Doing the math, we can count on 9.7656 mV per inch from the AN pin. The Vcc/512 ratio works will with 10-bit A-to-D converters.

When the 10-bit A-to-D reference voltage is set to +5.12 volts, each A-to-D step (not including zero) is 4.8828 mV, which happens to be half of the EZ0's volts-per-inch figure of 9.7656 mV. If we put my HP-15C to work on the 3.3 volt A-to-D figures, we come up with 6.4453 mV per inch. The 3.3 volt ratio is not as pretty as the 5.12 volt ratio, but if that's what you have to run, you run it and work with the hand you're dealt. The LV-MaxSonar-EZ0 will supply precise A-to-D voltages. Your microcontroller must be able to handle the 3.3 volt A-to-D information accurately. The AN output voltages are buffered and represent the most recent ranging data.

You already have a clue as to the operation of the LV-MaxSonar-EZ0's RX pin. Recall that a pulse applied to the RX pin of a chained ultrasonic rangefinder will trigger a ranging operation. The RX pin is pulled logically high. In single ultrasonic rangefinder designs, ranging operations will be continuous if the RX pin is left open. The RX pin can also be held logically high if your host microcontroller needs to control the ranging process. Otherwise, if the EZ0's ultrasonic rangefinder RX pin is pulled logically low, ranging will cease. A low-to-high logical pulse with a duration of 20 µs or greater will trigger a ranging operation.

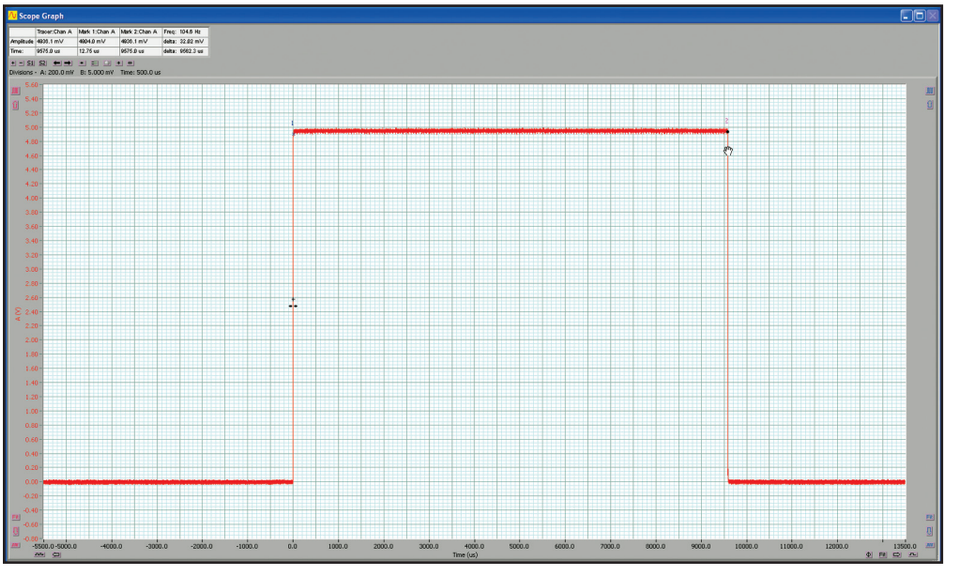
The EZ0 TX pin is very interesting. As long as the LV-MaxSonar-EZ0's BW pin is open or held low, the TX pin spouts asynchronous serial data in RS-232 format. Recall that when the BW pin is forced to a logical high, the TX pin will revert to sending pulses instead of RS-232 ranging data. Although the TX pin issues data in RS-232 format at zero-to-Vcc levels, you can hang the EZ0's TX pin on your laptop's serial port interface. The signal levels at the TX pin are logic levels and don't adhere to true positive and negative RS-232 voltage levels. So, to be politically correct, you'll need an RS-232 converter circuit or IC to interface the EZ0's TX signal to a true RS-232 port. You can take your chances with a direct interface between a PC serial port and the TX pin as long as you never connect the serial port's TX pin to the LV-MaxSonar-EZ0 I/O interface. As long as you're pushing properly polarized data into the PC's RX pin, there's a chance the serial interface will actually interpret the zero-to-Vcc logic transitions as if they were RS-232 signals. My Lenovo laptop has no problems with the LV-MaxSonar-EZ0 serial interface.

Once you've decided how to connect the EZ0's TX pin, your firmware should expect to see an ASCII "R" with three ASCII character digits following. The three ASCII digits will be your ranging data in inches. The maximum value of the ranging data will be 255 inches. A carriage return character (ASCII 13 or 0x0D) denotes the end of the ranging data stream. Speeds and feeds for the TX I/O pin's serial data are standard: 9600 bps, eight data bits, no parity, and one stop bit.

All of the methods of obtaining ranging data from the EZ0 can be used simultaneously. All we need to do is provide the necessary microcontroller interface to capture the ranging data from our desired ranging data portal. However, before we can start writing our I/O interface code, we need to understand the LV-MaxSonar-EZ0's timing and power-up specifications.

LV-MaxSonar Timing

The LV-MaxSonar-EZ0 needs 250 ms of idle time following power-up. After the 250 ms have passed, the EZ0



SCREENSHOT 1. This is a look at a LV-MaxSonar-EZ0 ranging pulse that is emitted from the ultrasonic rangefinder's PW pin. This pulse happens to be 9562.3 µS wide. With 147 µS representing one inch, this pulse equates to 65.0496 inches to the target, which happens to be the coffee tabletop to ceiling distance.

is ready to process input on its RX pin. Recall that if the RX pin is left open or forced logically high, the ranging process will begin and continue until the RX pin is pulled logically low. With that, let's run a scenario with the RX pin open at power-up plus 250 ms.

With the RX pin open or forced logically high following the setup time, the first ranging operation will be a 49 ms calibration cycle. The next ranging operation will be the first ranging operation that will report ranging data to the I/O interface. Thus, the very first ranging data will appear at the EZ0's data portal 98 ms past the 250 ms power-up setup period. All subsequent ranging operations will consume 49 ms each. What this all means is that the LV-MaxSonar-EZ0 can perform a ranging operation every 49 ms. If your application requires control of the ranging process, the EZ0 will scan the RX line at the end of every ranging cycle. This allows you to force the RX I/O pin logically low and take control of the scheduling of the subsequent ranging cycles.

Each ranging cycle is initiated by a logical high level at the RX I/O pin. Thirteen 42 kHz waves are transmitted at the beginning of a ranging cycle. After the 13 waves have been sent, the PW I/O pin is pulled to a logically high level. The PW pin will go logically low when a target object is detected. As you would expect, the maximum length of a PW pulse is 37.5 ms, or the equivalent of just over 255 inches. The maximum ranging distance of the LV-MaxSonar-EZ0 is 254 inches.

The 37.5 ms pulse width will occur when no target objects are detected. Assuming we didn't detect a target object, we still have 11.5 ms of time left in the ranging cycle; 6.8 ms of the remaining ranging cycle time is used to adjust the analog voltage that will appear on the AN pin to the correct level. We still have 4.7 ms left. At this point, the EZ0 has presented its pulse width ranging data and its analog ranging data to the I/O interface. The RS-232 ranging data is all that's left to present. The serial ranging data is sent during the final 4.7 ms of the ranging cycle.

To guarantee successful ranging operations, all we have to do is make sure that there are no targets closer than seven inches to the ultrasonic rangefinder during its calibration time. Also, the EZ0 is not an outdoor cat. So, we must be sure to keep it out of harm's way as far as weather goes. We now have enough information to begin writing some interface code. I'm going to write the EZ0 driver in C using HI-TECH PICC-18 and I'll target the PIC18F2620. Before we start writing code and assembling hardware, we can use my CleverScope to check out the LV-MaxSonar-EZ0's pulse ranging data mechanism.

Ranging with a CleverScope

Let's use our human eyes to interpret the EZ0 ranging pulse I captured in Screenshot 1. The pulse width as measured by the CleverScope extents is 9562.3 µs. A bit of simple math will yield the distance from my coffee tabletop to the ceiling:

9562.3 µs/147 µs per inch = 65.0496 inches

This is an easy way to receive instant gratification from a ultrasonic rangefinder. However, to make the ranging information work for us, we must employ the resources of a microcontroller. My rangefinder support hardware is visually obtainable in Schematic 2. As you can see, I've tied the LV-MaxSonar ultrasonic rangefinder's PW output pin to the PIC18F2620's CCP1 capture pin.

Ranging with a PIC18F2620

We need to electronically measure the pulse width presented to the PIC18F2620's CCP1 capture input. The algorithm is simple and so is the code. We must set up the PIC18F2620 capture engine to trigger an interrupt on the rising edge of the PW ranging signal. Meanwhile, TIMER1 is running free with a period of 1 µs. Thus, a count is supplied to the CCP1 holding registers every microsecond. The 1 µs TIMER1 period is a direct result of us running the PIC18F2620 system clock at 4 MHz. I programmatically

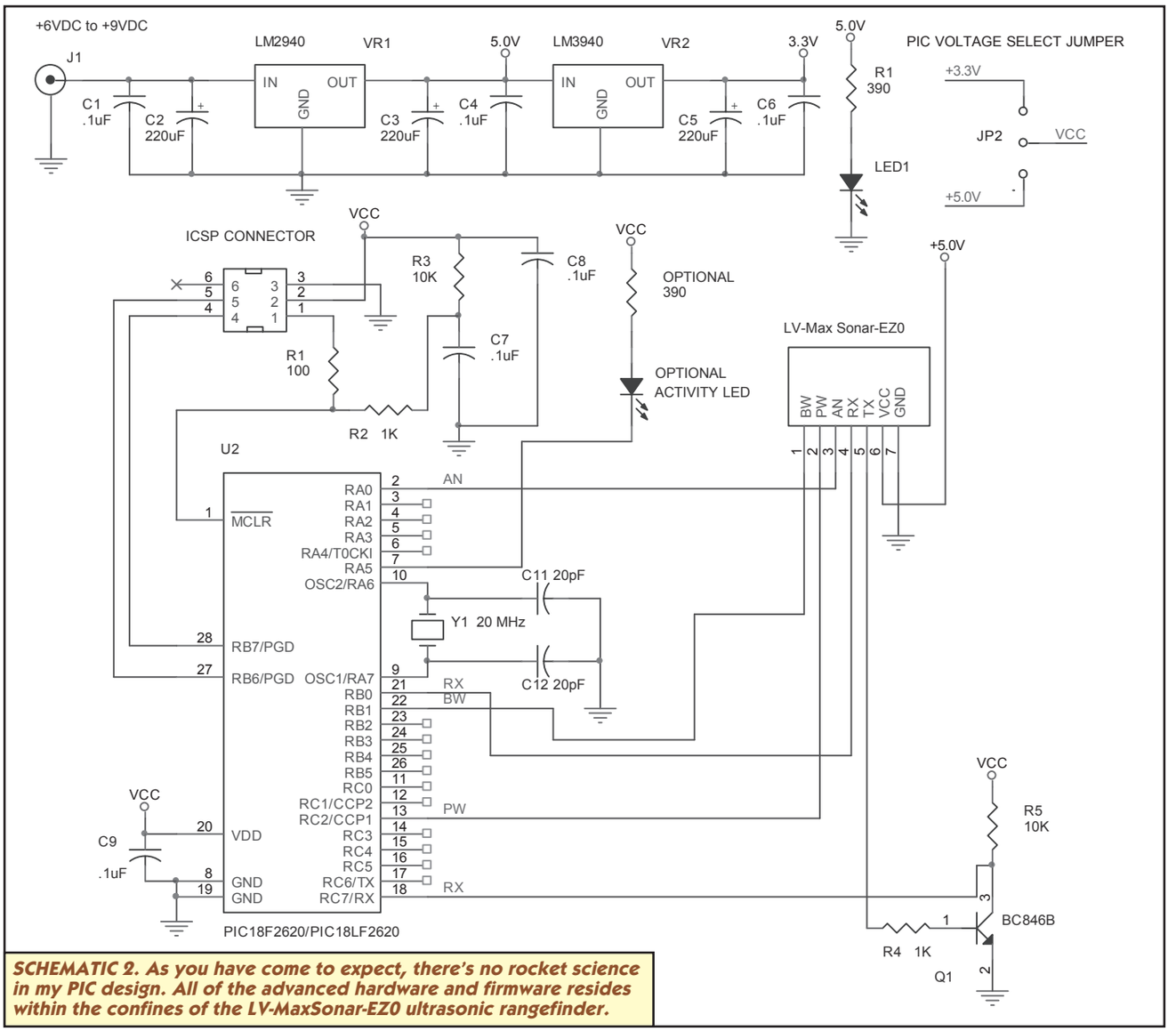
overrode the 20 MHz crystal you see in Schematic 2 with the PIC18F2620's internal oscillator.

When the rising edge of the PW ranging signal triggers an interrupt, we immediately read the value of the CCP1 holding registers and preserve that value in the pulsestart variable. The next step we perform is to configure the CCP1 input to trigger on a falling edge. Meanwhile, the TIMER1 clock is still counting. When the falling edge occurs, another interrupt is generated and we preserve the contents of the CCP1 holding registers in the pulseend variable.

The pulse width is the difference in the value of the pulseend variable and the pulsestart variable. Since the pulsewidth value is in microseconds, we can easily determine the target distance in inches by dividing the pulsewidth value by 147. Here's a look at the PW pulse capture interrupt handler code:

```
void interrupt MEASURE(void)
{
    if(CCP1IE && CCP1IF)
    {
        //interrupt caused by rising edge of PW
        if(flags.rising_edge)
        {
            //save start count
            pulsestart = make16(CCPR1H,CCPR1L);

            CCP1CON = 0b00000100;
            //capture falling edge
            flags.rising_edge = 0;
            //setup for falling edge
        }
        else
        //interrupt caused by falling edge of PW
        {
            //save end count
            pulseend = make16(CCPR1H,CCPR1L);
```



SCHEMATIC 2. As you have come to expect, there's no rocket science in my PIC design. All of the advanced hardware and firmware resides within the confines of the LV-MaxSonar-EZ0 ultrasonic rangefinder.

```
            CCP1CON = 0b00000101;
            //capture rising edge
            flags.rising_edge = 1;
            //setup for rising edge
            flags.captured = 1;
            //signal captured pulse
        }
            CCP1IF = 0; //clear the CCP1 interrupt flag
    }
}
```

The interrupt handler is steered by the flags.rising_edge flag, which is a logical one for capturing the rising edge of the PW ranging pulse and a logical zero for capturing the falling edge of the PW ranging pulse. The flags.captured bit signals the main body code that a valid set of pulse width values has been captured. Here's how the flag bits were realized in code:

```
typedef struct {
    charrising_edge:1;
    charcaptured:1;
} FFlags;

FFlags flags;
```

I created a structure of type FFlags, which consists of two bits, rising_edge, and captured. The instantiated structure flags are based on the structure FFlags. FFlags is reusable. For instance, I could instantiate a structure of bits called pwbits in this way:

```
FFlags pwbits;
```

The bits contained within the structure pwbits are referenced in the following manner:

```
pwbits.rising_edge = 1;
pwbits.captured = 0;
```

Although the bits rising_edge and captured are common to both structures, they are separate entities and can be used together in the same body of code. Defining flag bits this way is just a fancy (and easier) way of doing this:

```
char flags;
#define rising_edge      0x01    //rising_edge bit
#define captured         0x02    //captured bit

//rising_edge = 0
#define clr_rising_edge  flags &= ~rising_edge

//rising_edge = 1
#define set_rising_edge  flags |= rising_edge

//captured = 0
#define clr_captured  flags &= ~captured

//captured = 1
#define set_captured  flags |= captured
```

Let's take a look at the main body code that summons the services of the capture interrupt handler we call MEASURE. Recall that the ultrasonic rangefinder needs 250 ms of time to get its act together after power-up. Also recall that the rangefinder's first ranging cycle is a calibration run. So, to make sure the EZ0 ultrasonic range finder is ready for work, we allow it to run free for a while:

```
//*********************************************************
//* MAIN SERVICE LOOP
//*********************************************************
void main(void)
{
    init();
    //allow TIME to calibrate
    RX = 1;

    for(temp16=0;temp16<0xFF;++temp16) {
        NOP();
    }
    RX = 0;
```

After we're sure the LV-MaxSonar rangefinder is ready to go, we turn our attention to the PIC18F2620 and set up the PIC's capture subsystem:

```
    CCP1CON = 0b00000101;        //capture rising edge
    CCP1IE = 1;          //enable capture interrupt
    CCP1IF = 0;          //clear capture interrupt flag
    reset_TIMER1();      //initialize TIMER1 and start it
```

We are now ready to let the main loop take control:

```
do{
    enable_GLOBALint;    //enable interrupts
    RX = 1;              //kick off a ranging cycle

    do{
        NOP();           //wait for ranging to complete
    } while(!flags.captured);
```

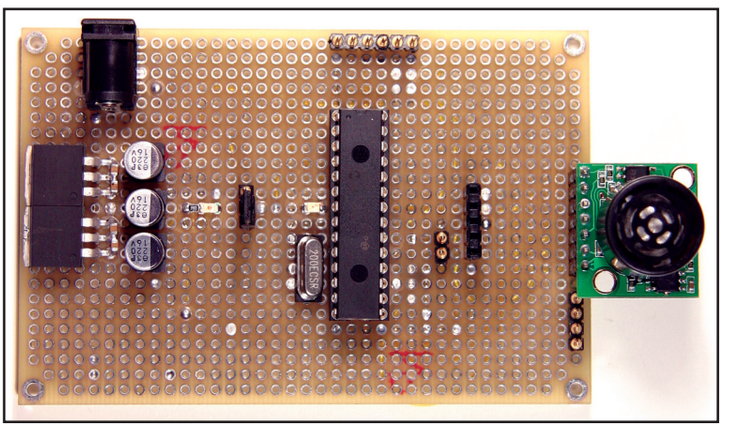
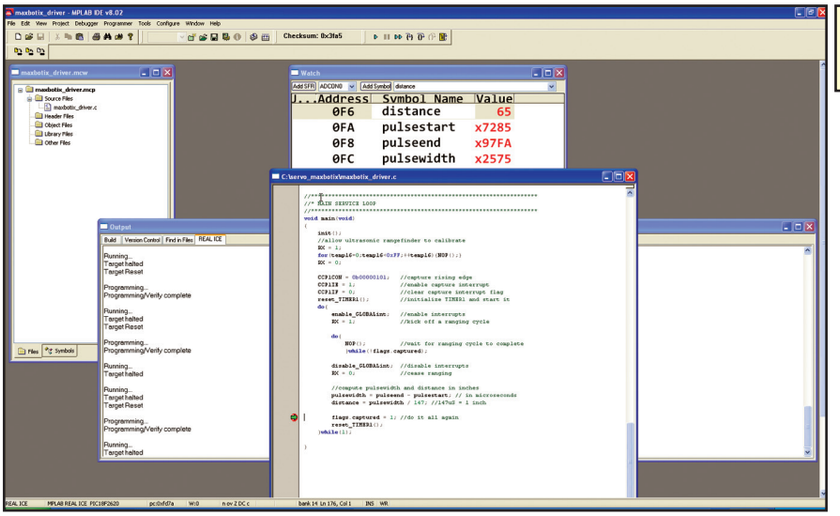


PHOTO 3. Nothing fancy here except the LV-MaxSonar-EZ0 ultrasonic rangefinder. From left to right we have a dual-rail +3.3/+5.0 volt regulated power supply, the PIC18F2620, and the EZ0 ultrasonic rangefinder.

Building a Sonar System



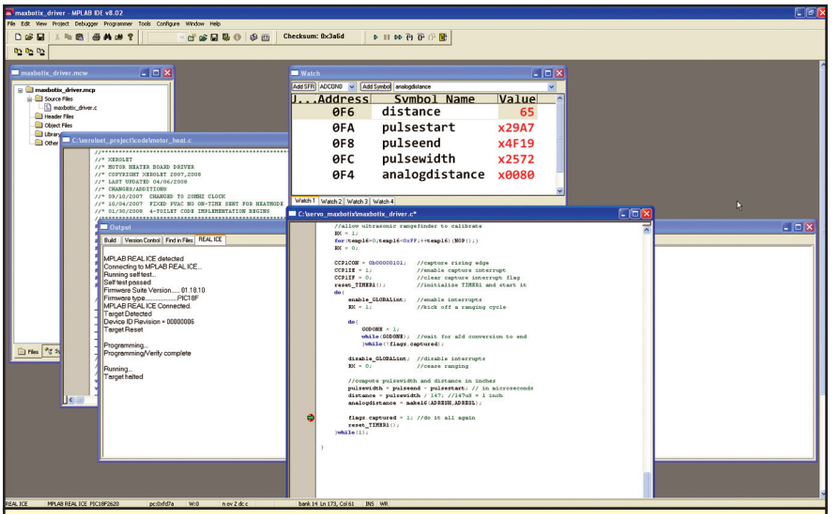
SCREENSHOT 2. The proof is in the pudding. The distance value matches our CleverScope observation.

```
 disable_GLOBALint;        //disable
                           //interrupts
 RX = 0;                   //cease ranging

 //compute pulsewidth and distance
 //in inches
 pulsewidth = pulseend - pulsestart;
// in ms
 distance = pulsewidth / 147;
//147us = 1"

 flags.captured = 1;       //do it all
                           //again
 reset_TIMER1();
} while(1);
```

The result of running the PW code through a ranging cycle is captured in Screenshot 2. Enough said.



SCREENSHOT 3. The analogdistance value is actually the number of 4.858 mV steps measured by the PIC18F2620's analog-to-digital converter subsystem. To get the distance, we must multiply the number of steps and the voltage per step values and divide the product by the AN scaling factor.

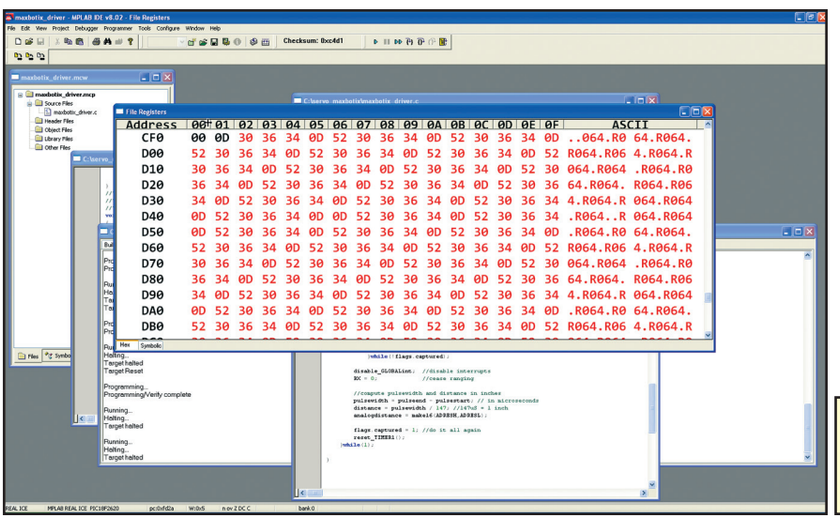
Ranging with an Analog-to-Digital Converter

I put my Meterman DM73B multimeter leads across the +5 volt power supply of the rangefinder support hardware you see in Photo 3. My +5 volt power supply is actually producing +4.97 VDC. Using the AN portal scaling factor (Vcc/512) and my actual power supply voltage (4.97 volts) yields a conversion factor of 9.707 mV per inch. With an A-to-D reference voltage equal to the power supply voltage, the PIC18F2620's A-to-D step voltage works out to 4.858 mV per step.

To get the ranging information from the EZ0's AN output pin into the PIC18F2620, I attached it to the PIC18F2620's RA0 analog input. To obtain the target distance, I sprinkled in a bit of A-to-D code onto our PW code:

```
do{
        GODONE = 1;        //trigger an AD
                           //conversion
        while(GODONE);     //wait for the
                           //conversion to
                           //end
        } while(!flags.captured);
```

Rather than just spin around doing



SCREENSHOT 4. It doesn't get much easier than this. All I had to do was add a transistor and some PIC18F2620 EUSART code to get this result.

nothing while waiting for the pulse width distance figure to be computed, I replaced the NOP (No Operation) instruction with an A-to-D conversion trigger. A ranging cycle with the new A-to-D code resulted in the analogdistance value you see in Screenshot 3. To convert the analogdistance voltage value to inches, we do the following:

1) Convert the analogdistance raw value to volts: 0x80 * 4.858 mV = 0.6218 volts.

2) Convert volts to inches: 0.6218 volts/9.707 mV = 64.062 inches.

If we consider the PW distance golden, the A-to-D distance value is well within tolerances, considering I'm feeding the PIC18F2620's A-to-D converter with a piece of wirewrap wire hung out in the wind. Let's see what the TX output has to say.

Automatic Ranging with the TX Output

Transistor Q1 in Schematic 2 is acting as an inverter. If we are to make any sense of the TX ASCII output, we must invert the TX serial data before presenting it to the PIC18F2620 RX input. The PIC18F2620 doesn't have a native method of inverting the data that is coming into its EUSART. So, Q1 acting as a logic inverter is a necessary hardware addition. From the looks of Screenshot 4, it appears 64 inches is the consensus distance determination.

Home on the Range

I've had a great time experimenting with my set of LV-MaxSonar ultrasonic rangefinders. I'll post all of the PIC18F2620 LV-MaxSonar ultrasonic rangefinder driver code we talked about plus the PIC18F2620 RS-232 driver code on the *SERVO* website (**www.servomagazine.com**) so that you can have just as much fun with your LV-MaxSonar ultrasonic rangefinder as I had with mine. See you next time! **SV**

Fred Eady can be reached via email at fred@edtp.com.

Resources

MaxBotix — **www.maxbotix.com**
LV-MaxSonar Ultrasonic Rangefinders

HI-TECH Software — **www.htsoft.com**
HI-TECH PICC-18 C Compiler

Microchip — **www.microchip.com**
PIC18F2620