

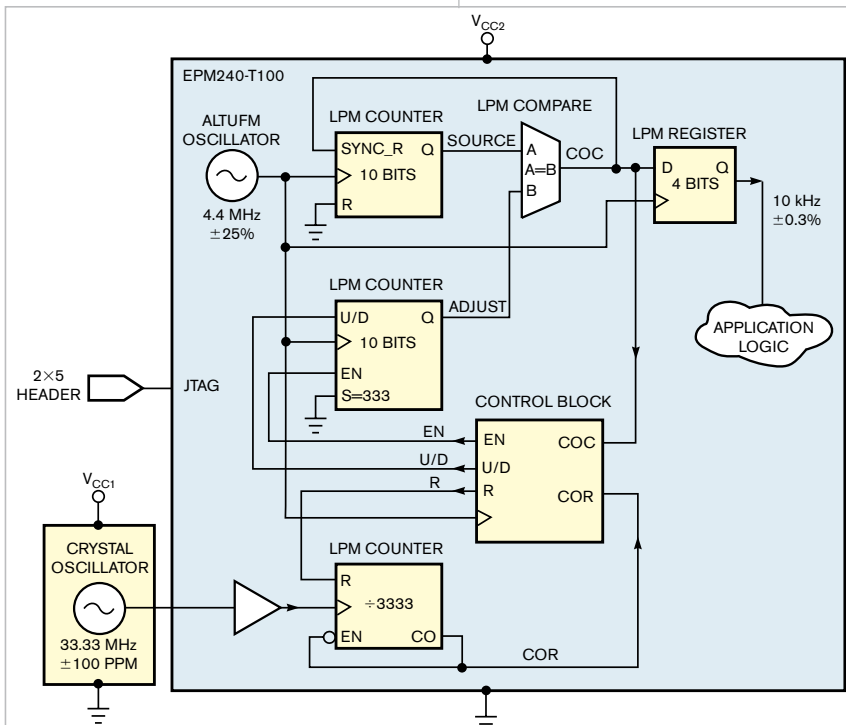
## CPLD's internal oscillator performs autocalibration

Rafael Camarota, Altera Corp, San Jose, CA

The MAX II CPLD family from Altera ([www.altera.com](http://www.altera.com)) features an internal oscillator that dissipates much lower power than do external oscillators. The internal oscillator has an accuracy of only  $\pm 25\%$ , sometimes limiting its usage. For example, many applications, such as an interval timer for data gathering and a real-time clock, require more accuracy— $\pm 0.1$  and  $\pm 0.001\%$ , respectively. A simple circuit uses an external crystal oscillator to calibrate a timer to better than  $\pm 0.3\%$  accuracy. The internal oscillator sustains the calibrated

output even after you shut down the external oscillator to save power. The circuit maintains this accuracy as long as the  $V_{CC}$  and temperature are stable. Whenever you enable the external oscillator, the circuit quickly recalibrates if necessary.

A remote industrial sensor should sample an event every second. To save power, a timer powers down most of the sensor circuit most of the time to increase battery life. The system powers up for a short sample; then, the system, except for the CPLD, powers down, which times the period to the



**Figure 1** This internal CPLD counter first synchronizes with an external clock to  $\pm 0.3\%$  accuracy and stays at that frequency until reset.

### DI Inside

**64** Swapping bits improves performance of FPGA-PWM counter

**66** Relays eliminate high-voltage noise

**70** VHDL program enables PCI-bus-arbiter core

► **What are your design problems and solutions? Publish them here and receive \$150! Send your Design Ideas to [edndesignideas@reedbusiness.com](mailto:edndesignideas@reedbusiness.com).**

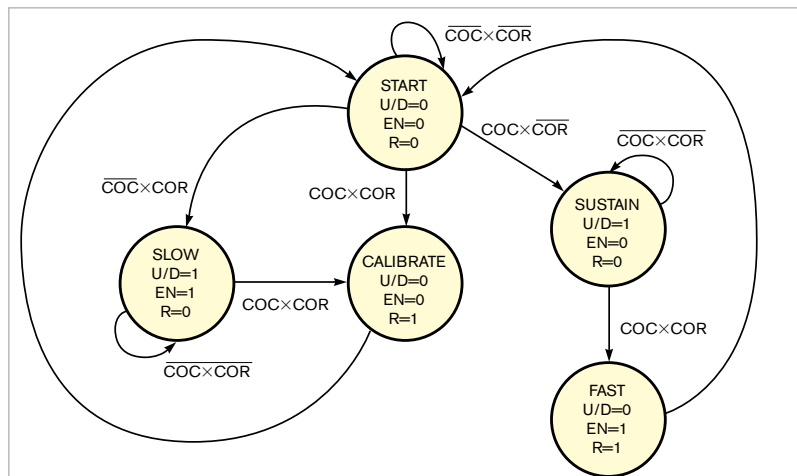
next power-up, sample, and calibration. Most of the components of a wireless receiver in a power-saving mode power down; however, the CPLD timer and wake-up mode stay on for monitoring and calibration.

**Figure 1** shows a simple circuit with a crystal oscillator with typical  $\pm 100$ -ppm accuracy; an EPM240 CPLD with a  $\pm 25\%$ -accurate, 4.4-MHz internal oscillator; and an autocalibration circuit in the programmable-logic array that generates a  $\pm 0.3\%$ -accurate, 10-kHz clock. For simplicity, the figure omits the external oscillator's  $V_{CC1}$  power-down or enable circuit and the application logic using the 10-kHz clock. The 33.33-MHz clock drives a reference counter, which is a divide-by-3333 LPM (library-of-parameterized-macros) counter. You derive LPM blocks from Altera's Quartus II LPM. The COR (carry-out-reference) signal feeds back to the count-enable input such that the COR signal stays at one after reaching the 3333 count until you apply the reset signal. The divide-by-3333 counter generates a 0.1-msec reference period. The 4.4-MHz LPM oscillator drives all other clocks in the autocalibration circuit: the source

counter, a 10-bit counter with a power-up asynchronous reset; a synchronous reset; and a 10-bit output source. The 4.4-MHz clock also drives the 10-bit up/down-adjust counter that presets to 333 at power-up. It has an enable input, an up/down-control-input signal, and a 10-bit output adjustment. The adjust and source drive the inputs of the compare LPM that generates a one on the COC (carry-out-from-comparator) signal when adjust equals the source. The COC signal drives the synchronous input of the source counter, making it a free-running counter with a period equal to the adjustment signal. An LPM register converts the COC signal into a synchronous, 10-kHz pulse when you calibrate the system. The control logic block generates enable, up/down, and synchronous-reset output signals based on the COC and COR inputs.

**Figure 2** shows the operation of the control-block state machine. It also illustrates how the 10-kHz signal calibrates to the oscillator input. The system powers up in the start state, and the source and reference counters both start counting. Adjust starts at 333, the minimum count that the slowest variation of the LPM oscillator would require to generate a 10-kHz clock. The COR signal typically goes high before the COC signal. This action moves the state machine to the slow state, enabling the adjust counter in the up mode. It counts up from 333 until the source equals the reference, removing most of the difference between adjust and the value necessary to achieve calibration. Once the source equals the reference, the state machine moves to the calibrate state. Calibrate disables the adjust counter and resets the reference counter. The free-running source counter resets at the same time.

The COR signal will likely occur



**Figure 2** This state machine shows the transitions of the control block in Figure 1.

again before the COC signal and will repeat the last sequence. Eventually, the COC signal will happen before the COR signal, moving the state machine to the sustain state. In this state, the adjust counter is disabled. Once the COR signal goes high, the COC signal comes around again, making the COC and COR signals ones. The state machine then goes to the fast state. Fast enables the adjust counter in the down mode, resets the reference counter, and then goes to start. The free-running source counter resets at the same time.

When you calibrate the circuit, the COR and COC signals occur at the same time, and the state machine goes to the calibrate state. The adjust counter remains constant, the source counter resets, and the state machine moves to start. Meanwhile, the free-running source counter resets. The system stays in this calibrated loop with an occasional cycle through slow or fast to make minor adjustments to the adjust counter. If the external oscillator stops, the COR signal stays low, resulting in

the state machine's staying in the sustain state until the external clock starts again. In the sustain state, the 10-kHz output stays constant assuming no significant change in system temperature or  $V_{CC}$ .

The following equations set the reference-count, adjust, and source-counter bit width; the adjust-counter start value; and the output-frequency accuracy: Adjust and source bit width =  $\log_2(5,555,555/\text{output frequency})$  rounded up; adjust-start value =  $3,333,333/\text{output frequency}$ ; reference-counter period = external-oscillator frequency/output frequency; output-frequency error =  $\pm 1\%/(3,333,333/\text{output frequency})$ ; maximum output jitter =  $\pm 1/3,333,333$  sec; and maximum calibration time = output frequency  $\times 5$ .

You can achieve accuracy better than  $\pm 0.3\%$  with a slower output frequency, but it cannot exceed the accuracy of the external oscillator. Therefore, you can build a real-time clock with a 0.01-second resolution and  $\pm 0.003\%$  accuracy. **EDN**

## Swapping bits improves performance of FPGA-PWM counter

Stefaan Vanheesbeke, Ledegem, Belgium



When you need some analog outputs and you have an FPGA in your system, you probably choose to use a PWM module and a simple low-

pass filter such as those in **Figure 1**. The output of the FPGA is typically a waveform with a fixed-frequency, variable-duty cycle, which a counter and a digital comparator generate (**Listing 1**).

Suppose that Enable is high, the counter counts up every clock cycle,

## LISTING 1 FPGA OUTPUT

```

module pwm(Clk, Reset, Enable, Value, Out);
parameter CountBits = 8;

input Clk, Reset;
input Enable;
output Out;
input [CountBits-1:0] Value;

reg [CountBits-1:0] Count;

assign Out = Count < Value;

always @(posedge Clk or posedge Reset)
  if (Reset)
    Count <= 0;
  else
    if (Enable)
      Count <= Count + 1;
endmodule

```

## LISTING 2 REWIRING MODIFICATION

```

module pwm(Clk, Reset, Enable, Value, Out);
parameter CountBits = 8;

input Clk, Reset;
input Enable;
output Out;
input [CountBits-1:0] Value;

reg [CountBits-1:0] Count;

reg [CountBits-1:0] Swapped;
integer k;
always @*
  for (k = 0; k < CountBits; k=k+1)
    Swapped[k] = Count[CountBits-1-k];

assign Out = Swapped < Value;

always @(posedge Clk or posedge Reset)
  if (Reset)
    Count <= 0;
  else
    if (Enable)
      Count <= Count + 1;
endmodule

```

and the frequency of the PWM output is the clock frequency divided by 2 count bits. You can use Enable to lower the output frequency by connecting it to a prescaler. Because the output frequency is fixed, the filter is easy to

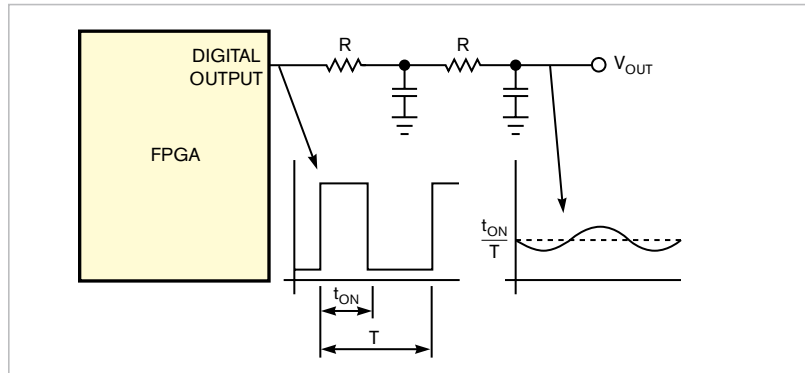


Figure 1 A simple lowpass filter changes the PWM digital output from an FPGA to an analog voltage level. The maximum ripple occurs at a 50% duty cycle.

## LISTING 3 SIMULATION RESULTS

```

Simulation results :

Testing 0 : 0000000000000000
Testing 1 : 0000000000000001
Testing 2 : 0000000100000001
Testing 3 : 0001000100000001
Testing 4 : 0001000100010001
Testing 5 : 0101000100010001
Testing 6 : 0101000101010001
Testing 7 : 0101010101010001
Testing 8 : 0101010101010101
Testing 9 : 1101010101010101
Testing 10 : 1101010111010101
Testing 11 : 110110111010101
Testing 12 : 110110111011101
Testing 13 : 111110111011101
Testing 14 : 111110111111101
Testing 15 : 111111111111101

```

calculate, because you know that the worst-case ripple happens at a duty cycle of 50%. The combination of the desired maximum ripple and settling time determines the filter type and RC (resistance/capacitance) values.

With a small change to the code in Listing 1, you can improve the performance of the PWM circuit. Whereas in the original system, the maximum ripple currents occur at a duty cycle of 50% and the minimum ripple currents

occur at the minimum duty cycle, the improved version shows a maximum ripple equal to the minimum of the standard version. The trick is to generate the highest frequency possible but keep the average duty cycle constant. The higher the frequency of the pulses on the output, the better the filter does its job.

The modification to Listing 1 consists of rewiring the binary comparator with all the bits swapped from left to right. The MSB (most significant bit) becomes the LSB (least significant bit), the LSB becomes the MSB, and so on (Listing 2). You do only a rewiring requiring no extra registers or logic.

Listing 3 shows the pulse trains that a 4-bit PWM emits. In Listing 3, you see that at 50% duty cycle (Value=8, second column), the frequency is maximum and equal to the clock frequency divided by two. At the first point at which some ripple shows up (Value=1, second column), there is exactly the same ripple as in the conventional PWM system—that is, the pulse train is the same. EDN

## Relays eliminate high-voltage noise

Jui-I Tsai, Woei-Wu Pai, Feng-Chang Hsu, Po-Jui Chen, Ching-Cheng Teng, and Tai-Shan Liao, National Applied Research Laboratories, Hsinchu, Taiwan

Most laboratories and industrial environments have many kinds of electrical-noise sources at all frequencies from heavy machinery, in-

struments, power supplies, and TV stations. Engineers have used many simple devices and techniques to handle this noise. These techniques include the use

of proper grounding methods, shielded and twisted wires, signal averaging, differential-input-voltage amplifiers, and filters. Although these methods can control and reduce the noise in most measurements, some techniques just prevent noise from entering the system, whereas others remove only extraneous noise from the signal. These methods usually find use only in low-voltage sys-

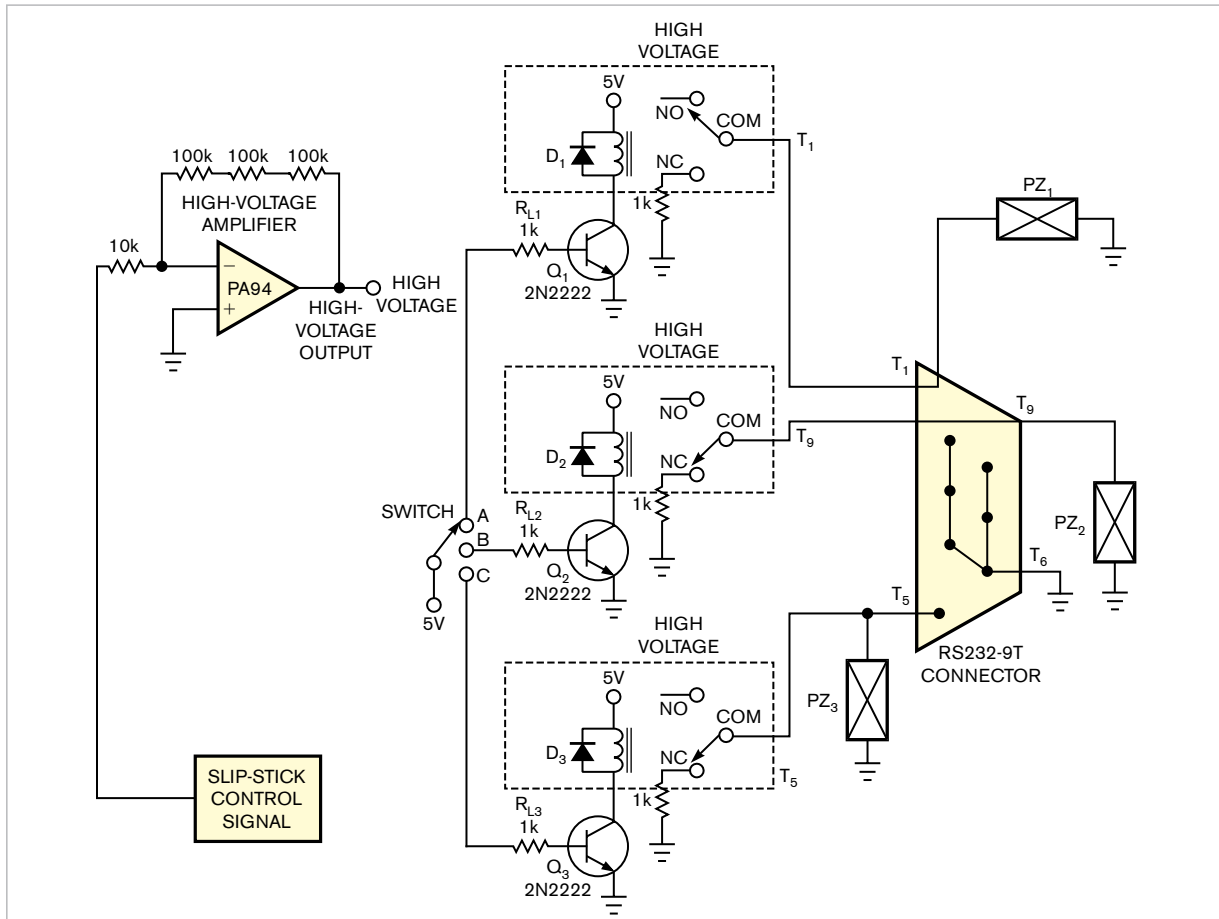


Figure 1 This simple circuit connects unenergized inputs to ground through a resistor.

tems; they do not address high-voltage-induced noise. This Design Idea offers a practical approach to reducing high-voltage-induced noise. The floating input of a scanning electron microscope has high impedance, and it acts as an antenna, picking up noise signals. The microscope's actuators need a high-voltage signal to drive their piezoelectric slip-stick stack motors. The motion mechanism requires a ramping waveform spanning to 800V p-p. The mechanism requires multiple channels because there are three degrees of tip motion. Some microscopes incorporate optical-path-adjustment microsliders for atomic-force microscopy; those scopes need even more channels.

Traditionally, each channel needs a high-voltage amplifier. So, two degrees of tip motion need two high-voltage amplifiers, three degrees need three amplifiers, and so on. High-voltage am-

plifiers are expensive and need considerable space on the PCB (printed-circuit board), however. Therefore, controlling multiple degrees of tip motion using only one high-voltage amplifier that switches among multiple channels saves cost and space. The pins of high-voltage connectors have enough space between them to avoid disturb-


### THE FLOATING INPUT OF A SCANNING ELECTRON MICROSCOPE HAS HIGH IMPEDANCE, AND IT ACTS AS AN ANTENNA, PICKING UP NOISE SIGNALS.

ing adjacent signals. But high-voltage connectors are expensive and too large to easily arrange. So, the best choice is to use a commercial RS-232-standard, nine-pin/25-pin connector (Figure 1). The pins of most commercial RS-232 connectors are close enough together to easily pick up induced high-voltage signals. You can solve this problem by connecting a low impedance to the floating pins of the RS-232 connector.

In this circuit, three piezoelectric motors, PZ<sub>1</sub>, PZ<sub>2</sub>, and PZ<sub>3</sub>, connect to the T<sub>1</sub>, T<sub>5</sub>, and T<sub>9</sub> pins of the RS-232-9T connector. The circuit has three relays that switch the high-voltage input to the piezoelectric motors. The normally open node of the relays connects to the high-voltage-amplifier output. The normally closed nodes of the relays connect to three 1-kΩ resistors to bypass high-voltage-induced noise to ground. **EDN**

## VHDL program enables PCI-bus-arbiter core

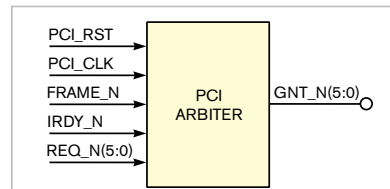
Antonio Di Rocco, Selex Communications, Chieti, Italy

 This Design Idea describes a VHDL implementation of a PCI 2.2-bus arbiter (**Figure 1**). Any PCI system may have one or more PCI-master devices. Most devices can behave as target hosts, but one must be a PCI-bus initiator, or master. Normally, only microprocessors or high-level DSPs perform both PCI master and target modes, and they may include a PCI arbiter. **Listing 1**, a simple VHDL program, is available at [www.edn.com/070913di](http://www.edn.com/070913di). It performs an arbitration function by enabling access to the PCI bus depending on the predetermined priorities of each PCI device. The PCI-arbiter core interfaces with 33- and 66-MHz PCI systems, supports as many as six PCI-bus masters, supports “bus parking,” enables a pure rotational-arbitration scheme, supports bus latency and broken masters, and is a synthesizable VHDL source with-

out FPGA- or PLD-library intellectual property.

The PCI bus supports more than one master device. If only one master requests the bus, that master immediately gets the grant. If several devices simultaneously require the use of the PCI bus to perform a data transfer, they assert their request signal, REQ\_N, to the arbiter. The one with highest priority gets the GNT\_N grant. After that, the one with the second highest priority has the highest priority, and so on. The PCI\_RST assertion resets the arbiter’s priority-shift register to device 0.

The PCI bus has no pullups on the AD bus and C/BE lines. To avoid having these signals float for a long period, PCI designs must implement bus parking, meaning that a master device drives the AD bus and C/BE lines during bus-idle states. The arbiter selects



**Figure 1** This PLD/FPGA-based PCI-bus arbiter grants bus requests based on a simple rotational-priority scheme.

which master will be park master. The arbiter asserts GNT\_N of the park master, even though the park master did not assert REQ\_N. The constant “Bus\_parker” in the VHDL code defines the park master. After a device has access to the PCI bus, this device must start the bus access within 16 PCI clock cycles. If this start-up does not happen, the device loses the bus grant, and the device with the next highest priority gets the bus. To check bus latency, the arbiter must check the signals FRAME\_N and IRDY\_N. The PCI-arbiter core fits into any PLD or FPGA and consumes few resources. **EDN**