

Oscilloscope helps obtain Bode plots in non-50Ω environments

Antonio Eguizabal, Freescale Semiconductor Inc, Tempe, AZ

▶ A Bode plot can simplify characterization of an active or a passive network by showing frequency and phase representations of the network's transfer function, T . In its classic form, a Bode plot graphs frequency data on an X-axis logarithmic scale and amplitude and phase data in logarithmic or linear format on the Y-axis scale. However, most network analyzers' input ports typically present fixed, low impedances of either 50 or 75Ω that load any device under test that connects to the ports. To measure passive or active circuits in environments other than 50 or 75Ω, you can buffer the analyzer's inputs with amplifiers that present high input impedances to the device under test and low output impedances that match the network analyzer's inputs.

As an alternative to building or purchasing custom buffer amplifiers, you can use the near-ideal amplifiers in an analog oscilloscope that provides a vertical amplifier output on its rear panels—for example, the venerable Tektronix (www.tektronix.com) 465B. Its more commonly available cousin, the Tektronix 2465, provides a Channel 2 output on its rear panel. This Design Idea describes a proven measurement method that obtains magnitude and phase graphs of both active and passive devices. A Bode plot displays the magnitude $|T(j\omega)|$ as a function of angular frequency, $\omega = 2\pi f$.

Most measurements span a broad range of frequencies, and it is thus helpful to present the frequency data in logarithmic format ($\log f$) on the graph's abscissa (X axis) and amplitude

DI Inside

76 PRBS generator runs at 1.5 Gbps

80 Use SystemVerilog for coverage metrics

▶ What are your design problems and solutions? Publish them here and receive \$150! Send your Design Ideas to edndesignideas@reedbusiness.com.

data formatted as $20\log(|T(j\omega)|)$ on the ordinate (Y axis). Two graphs of magnitude and phase versus frequency thus present a compact representation of the network's electrical characteristics. Using the analyzer's controls, select the magnitude of S_{21} and phase of S_{21} as Y-axis displays in rectangular coordinates and select the $\log f$ display option for the X axis.

A Tektronix 465B or 2465 oscilloscope's vertical amplifier presents a 100-MHz bandwidth, a 1-MΩ input impedance, and a 50Ω output impedance. Connect the scope's low-impedance output to the network analyzer's Port 2 input. A 10× probe that connects to the oscilloscope can raise its effective input impedance to as high as 10 MΩ. Oscilloscopes other than those mentioned or stand-alone amplifiers can deliver wider bandwidths, higher dynamic-input-voltage range, and reduced phase error and group delay for more accurate measurements. Figure 1 illustrates the basic measurement configuration. Use coaxial cables with appropriate connectors to match the network analyzer's inputs. If the network analyzer requires dc bias for Port 1, use an external power supply.

For best results, calibrate the system as follows.

1. Perform the network analyzer's

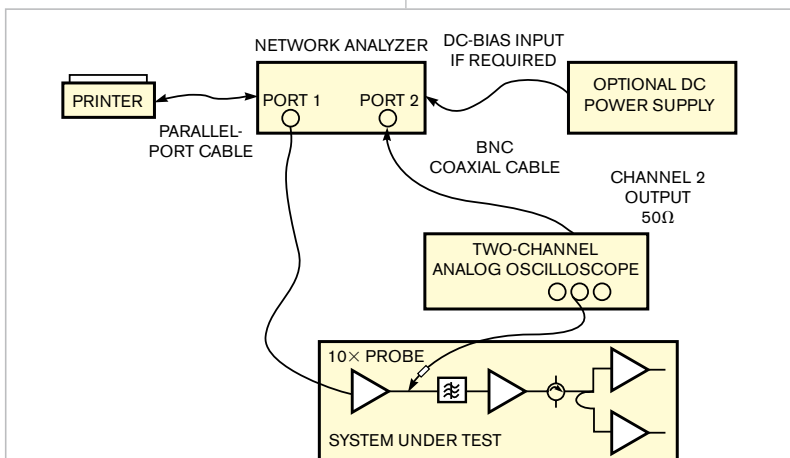


Figure 1 The basic test setup for generating Bode plots requires a network analyzer, an analog oscilloscope with one or more vertical outputs, an optional dc-bias power supply, and a printer.

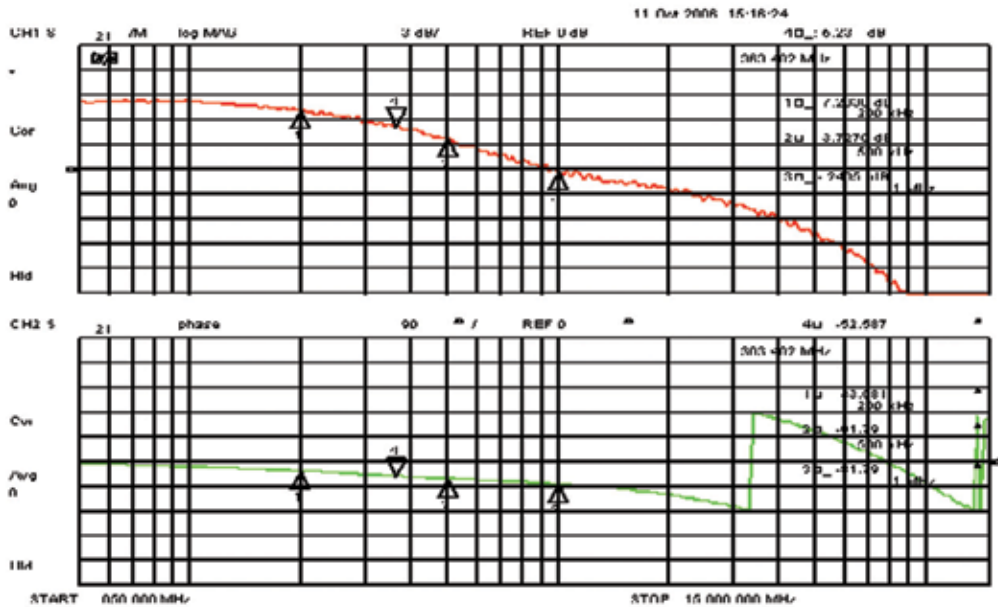


Figure 2 A vertical amplifier with 100-MHz bandwidth fairly accurately measures a device under test operating at 10 MHz, plotting the magnitude (top-trace) behavior and phase (bottom-trace) behavior of a typical device in log-frequency format spanning a 50-kHz to 15-MHz range. Markers show measured values at reference points.

two-port calibration procedure over the frequency range of interest.

2. Set the network analyzer to produce a dual display, with the magnitude of S21 on top and phase of S21 at the bottom of the display screen. Change the frequency-display mode from linear to log.

3. Set the oscilloscope for dc coupling and center its trace at midscreen. Select the required sweep rate and the triggering mode to ac and adjust the trigger level to produce a trace.

4. Connect the oscilloscope's Channel 2 input or probe to the network analyzer's Port 1 input and set the analyzer's controls to establish a reference line.

5. Adjust the vertical amplifier's

gain and attenuation (volts/division) controls until the analyzer displays random noise, which represents the lowest detectable signal.

6. Set the analyzer's gain-per-division scale to 3 dB/division, a convenient value for determining the frequencies at which the gain of the device under test decreases by 3 dB.


7. Adjust the network analyzer's source (output) power range in decibels referred to milliwatts and the oscilloscope's gain/attenuation settings in volts per division to obtain an optimum data display. If the device under test introduces appreciable gain or loss, adjust the analyzer's scale-reference control to recenter the displayed trace. **Figure 2** shows a Bode plot de-

rived from an active device that would not tolerate analyzer loads of less than 10-k Ω impedances.

To minimize the phase shift that the oscilloscope's vertical amplifier introduces, choose an amplifier whose bandwidth greatly exceeds the operational bandwidth. In **Figure 2**, a vertical amplifier with 100-MHz bandwidth fairly accurately measures a device under test operating at 10 MHz. You can eliminate phase-shift and amplitude errors that the test fixture introduces by storing a reference trace and subtracting it from the active trace. Refer to the network analyzer's operating manual for details. **EDN**

PRBS generator runs at 1.5 Gbps

Lukasz Sliwczynski, AGH University of Science and Technology, Institute of Electronics, Krakow, Poland

 PRBS (pseudorandom-binary-sequence), or PN (pseudo-noise), generators find a broad range of applications in digital-data trans-

mission (**Reference 1**). These circuits often comprise simple shift registers with feedback that can serve as test sources for serial-data links. As their

name implies, the output sequence is not truly random and in fact repeats after $2^N - 1$ bits, where N denotes the shift register's length. Polynomial notation, in which the polynomial order corresponds to the shift register's length and, thus, the PRBS' period provides a convenient method of describing the sequence.

Communications-equipment tests use certain standard polynomials. For example, x^7+x^6+1 yields a PRBS period of 127 bits, $x^{23}+x^{18}+1$ yields a period of more than 8 million bits, and $x^{31}+x^{28}+1$ yields a period that's 256 times longer. A PRBS with a longer period generally produces a greater variety of data patterns that more thoroughly check the transmission system's performance.

A simple shift register with feedback from an intermediate stage can generate a PRBS. The flip-flops constituting the register must run at a speed equal to the transmission speed, which may pose a problem if you want to build a long-period PRBS generator that runs at a gigahertz clock rate. A high-speed serializer such as Texas Instruments' (www.ti.com) TLK2201B, which runs at data rates as high as 1.6 Gbps, offers one potential solution to the problem. However, instead of accepting a PRBS in its natural fully serial format, the serializer accepts only 10-bit portions at a time.

The circuit in **Figure 1** illustrates a 31st-order, parallel-PRBS generator that delivers 10-bit output segments and can easily adapt to other PRBS orders and output widths. To design the circuit, begin by drawing a diagram with 31 flip-flops arranged in rows containing nominally 10 flip-flops. In this instance, the design comprises four rows, with only one flip-flop in Row 1. **Figure 1** shows the timing relationships among the flip-flops and the numbering convention.

The resulting structure forms a parallel shift register, with the fourth row fed directly from the third row, the third fed from the second, and so on. Flip-flops 10 through 2 in Row 2 and flip-flop 1 in Row 1 receive their inputs from the feedback path. This arrangement ensures that flip-flops in consecutive rows always deliver their outputs 10 time instants apart, and the generator's clock thus runs at one-tenth the speed of an equivalent serial-shift-register PRBS implementation.

To determine the feedback signals, derive the equation that describes a standard—that is, serial—

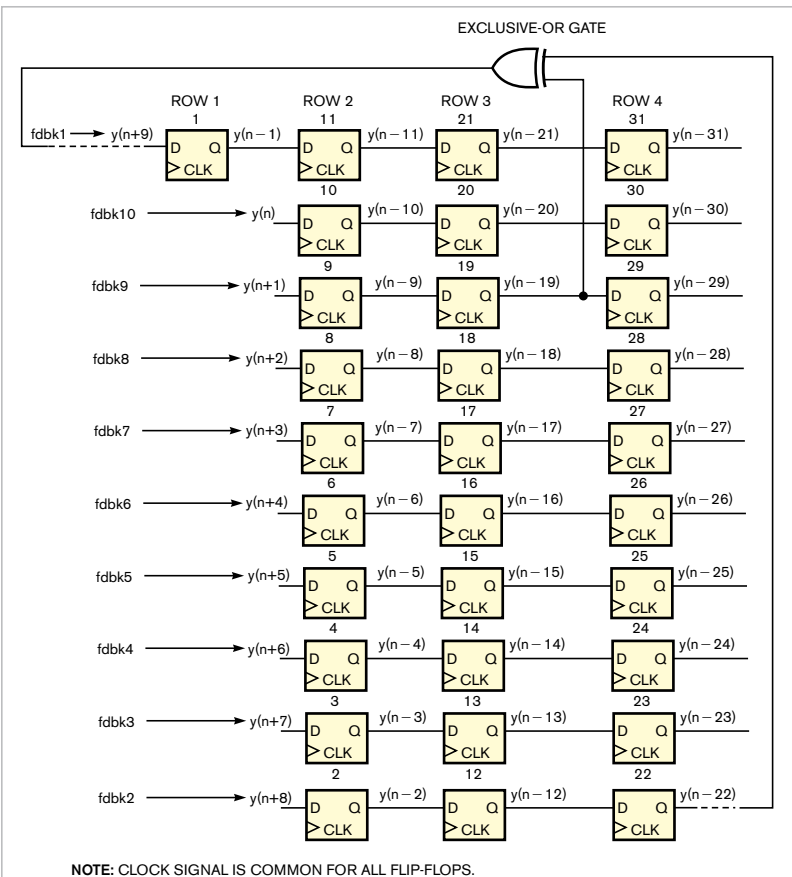


Figure 1 This circuit implements a 10-bit parallel-output PRBS generator defined by the polynomial equation $x^{31}+x^{28}+1$. To reduce clutter, the schematic shows only one of 10 exclusive-OR gates that generates the register's feedback signals. A common clock source (not shown) drives all 31 flip-flops' clock inputs.

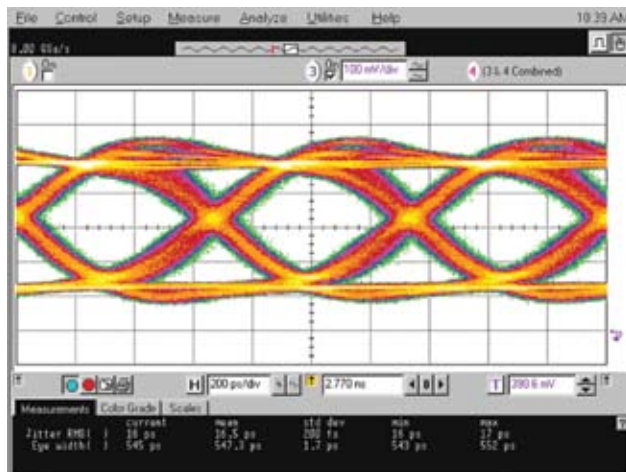


Figure 2 A pseudorandom sequence produces this eye diagram as measured at the output of a TLK2201B serializer that an FPGA-sequence generator drives.

PRBS generator's output, which, for a polynomial of $x^{31}+x^{28}+1$, yields: $y(n)=y(n-31) \text{ xor } y(n-28)$. Using that equation, you can derive the equations that describe feedback signals fdbk1 through fdbk10. That is, fdbk1: $y(n+9)=y(n-22) \text{ xor } y(n-19)$, fdbk2: $y(n+8)=y(n-23) \text{ xor } y(n-20)$, ... fdbk10: $y(n)=y(n-31) \text{ xor } y(n-28)$. For example, feedback signal fdbk1 derives from the output of a two-input exclusive-OR gate driven by the outputs of flip-flops 22 and 19.

Listing 1, which is available at the Web version of this Design Idea at

www.edn.com/070329di1, contains the VHDL code that implements the circuit of **Figure 1** in either a CPLD or an FPGA device. Lines 15 through 18 define the parallel-shift register, and lines 21 through 23 define the feedback circuit's construction. The circuit in this Design Idea fits into an XC3S50 Spartan 3 device from Xilinx (www.xilinx.com), runs at a 150-MHz clock rate, and drives a Texas Instruments TLK2201B serializer at 150 MHz through a 10-bit interface. Xilinx's ISE 7.1i software compiled the circuit's VHDL files. **Figure 2** dis-


plays an eye diagram for the serializer's output and confirms the circuit's operation at 1.5 Gbps. The compilation software predicts that the circuit should run at clock rates exceeding 300 MHz, but the TLK2201B limits operation to 150 MHz. **EDN**

REFERENCE

■ Miller, Andy, and Mike Gulotta, "PN generators using the SRL macro," Application Note APP211, Xilinx Inc, June 15, 2004, www.xilinx.com/bvdocs/appnotes/xapp211.pdf.

Use SystemVerilog for coverage metrics

Thomas L Anderson, Cadence Design Automation, San Jose, CA

 The design-and-verification industry is at the intersection of two important trends in the design and verification of SOC (system-on-chip) devices: the adoption of SystemVerilog HDVL (hardware-description and -verification language) and the increasingly critical role for coverage metrics. The interest in SystemVerilog is understandable; this IEEE-standard language has the features for RTL (register-transfer-level) design, high-level modeling, testbench creation, and assertion specification (**Reference 1**).

SystemVerilog also provides constructs for design-and-verification engineers to specify functional coverage points—conditions that designers must exercise for complete verification of the design. Designers increasingly use functional coverage to supplement traditional code coverage. The primary driver for this evolution is the widespread use of constrained-random-stimulus generation.

Traditional verification

plans typically include a list of design features or tests that verify features and test status. This approach has worked well with handwritten, directed tests because of the clear correspondence between features and tests. However, verification consists of writing and running each test in simulation, perhaps after turning on some code coverage to help identify features you may have missed in the plan.

Constrained-random-stimulus generation requires a different approach,

in which each automatically generated test can exercise many features and parts of the design. A modern verification plan lists features, functional coverage points for the features, and coverage status. You gauge verification closure by the number of coverage points you exercise rather than the number of tests you complete.

SystemVerilog provides all the features necessary to develop both handwritten tests and constrained-random testbenches and to track progress toward closure. Most simulators have built-in code coverage for the new design constructs that SystemVerilog introduces. Thus, code-coverage metrics are available for designs taking advantage of the language's advanced RTL features.

SystemVerilog provides several powerful specification methods for functional coverage. The first is cover property, which is part of the SVA (SystemVerilog Assertions) subset of the language. SVA's assertion features, including temporal sequences, are also available for functional coverage. For example, **Listing 1** ensures that the simulator exercises the two extremes—one and five cycles—of a request-ac-knowledge handshake. Both simulators and many formal-

LISTING 1 MINIMUM AND MAXIMUM RESPONSE

```
minimum_response: cover property (@(posedge clk)
    (req ##1 ack ));
maximum_response: cover property (@(posedge clk)
    (req ##5 ack ));
```

LISTING 2 PAYLOAD SIZES OF INCOMING PACKETS

```
covergroup payloads_seen (@(packet_received);
    coverpoint payload_size {
        bins empty = { 0 };
        bins minimum = { 1 };
        bins maximum = { 1023 };
        bins others = default; }
endgroup : payloads_seen
```

analysis tools support the cover-property construct. If formal analysis can prove that a coverage point is unreachable, a design bug may be blocking important functions from being exercised. If formal analysis instead provides a trace showing how to reach a coverage point, this trace can provide a good hint on how to write or generate a test.

Beyond individual coverage properties, you sometimes must track ranges of values. SystemVerilog provides the cover-group construct, which is not part of SVA, to perform this function. For example, **Listing 2** tracks the payload sizes of incoming packets on a network interface and ensures the coverage of corner cases of empty, minimum, and maximum payloads. SystemVerilog also provides the cross construct to measure cross-coverage between two coverage points. This feature allows

LISTING 3 ENUMERATED TYPE FOR FOUR PACKET CLASSES

```
enum { read, write, atomic, ctrl } packet_class
covergroup packets_seen @(packet_received);
  coverpoint payload_size {
    bins empty = { 0 };
    bins minimum = { 1 };
    bins maximum = { 1023 };
    bins others = default; }
  coverpoint packet_class;
  cross payload_size, packet_class;
endgroup : packets_seen
```

the tracking of combinations of coverage metrics. For example, **Listing 3** specifies an enumerated type for four packet classes for the network interface, adds a cover point to track the packet classes, and crosses the packet types with the payload sizes.

Ultimately, the SOC-tapeout decision must take into account all coverage metrics. Although functional coverage is the primary method, code coverage has value as a backup to identify areas of the design with no functional

coverage due to an incomplete verification plan. The project team needs to merge together code- and functional-coverage results to assess verification progress and help determine verification closure. Coverage is critical for modern, constrained-random verification. Without effective metrics, no reliable way exists to gauge status and manage progress. In addition to its other features and benefits, SystemVerilog provides support for functional coverage. By including coverage in the verification plan from the start of the project and taking advantage of SystemVerilog, the SOC team can employ a complete plan-to-closure methodology that greatly increases the chances for a successful product. **EDN**

REFERENCE

i www.systemverilog.org.