Using VHDL-AMS to model complex heterogeneous systems, part 2

You can implement DSP-controller algorithms using the mixed-signal features of the VHDL-AMS modeling language.

By R Scott Cooper, Mentor Graphics Corp

Designing a motion-control system is difficult using conventional simulation tools. Tools fall into two basic categories: high-level, “math-based,” tools and low-level, “conservation-based,” tools. The EDA industry developed HDLs (hardware description languages), such as VHDL-AMS, to encompass both the high- and low-level modeling capabilities of these tools, as well as the gap that exists between them. The result is an environment with virtually unlimited modeling flexibility.

Part one of this series concentrated on the ability of VHDL-AMS to deal with analog behavioral modeling by developing a D-Q (direct-quadrature) motor model and an FOC (field-oriented-control) system in which to test it. Both the model and system were developed using only analog modeling techniques.

In a real motion-control system of this type, you would implement many of the analog functions digitally, using a microprocessor or DSP. With this reality in mind, this second part of the series demonstrates how to reimplement much of the analog FOC system using sequential algorithms. It then demonstrates how to test those algorithms in VHDL-AMS and C.

DSP-model development

As previously stated, the goal is to “digitize” the DSP-controller portion of the system shown in Figure 1. (The DSP Controller is enclosed in the dashed block.) To do so, you replace the continuous analog functions with their discrete equivalents while ensuring that you preserve the functions designed into the analog system. Referring again to Figure 1, you can describe the summing-junction, scaling, slip-calculation, and phase-transformation blocks all with straightforward algebraic mathematics. As you’ll soon see, you can readily adapt these blocks to sequential algorithmic descriptions.

The PI (proportional-integral) controller is a little different, in that its transfer function includes a pole and zero (see Equation 1). Because the z-domain is to discrete systems what the Laplace domain is to analog systems, and because you are planning to run the DSP at a constant sample rate, z-domain transformation techniques are a good choice for digitizing this block. These techniques are also extensible to more sophisticated filter/controller functions.

Convert PI controller to z-domain

To convert the PI controller into its z-domain equivalent form, you use the bilinear-transformation, or trapezoidal-integration, method. You can describe the transfer function for the PI controller as:

\[ \frac{K_p s + K_i}{s} \]

where \( K_p \) and \( K_i \) represent proportional and integrator gain coefficients, respectively.

The bilinear transform is shown as:

\[ \frac{2}{T} \left( \frac{1 - z^{-1}}{1 + z^{-1}} \right) \]

where \( T \) is the sampling period.
If you substitute Equation 2 in for each occurrence of s in Equation 1, and simplify the resulting equation, the z-domain transfer function becomes:

\[
\frac{(2K_p + TK_j) + (-2K_p + TK_j)z^{-1}}{2 - 2z^{-1}}.
\]  

This new controller implementation should be functionally equivalent to its continuous counterpart in the system, provided you choose an appropriate sampling period.

**Implement PI controller in VHDL-AMS**

VHDL-AMS provides a useful attribute for testing z-domain transfer functions. The so-called ‘ztf (z-transfer-function) attribute allows you to implement z-domain models directly and test them in either the time domain or the frequency domain. This ability is particularly useful, because you cannot simulate the model in the frequency domain once you have implemented it algorithmically and executed it with a sample clock.

After you verify the system using the ‘ztf attribute, you then transform the z-domain coefficients into actual software code and test these algorithms directly. Listing 1 gives the VHDL-AMS architecture code for this new controller:

As the listing shows, the numerator and denominator terms from Equation 3 are stored as real vector constants, num and den, in descending powers of z. You then use num and den as arguments to the ‘ztf attribute, along with the sample period, T. Although you use the ‘ztf attribute with z-domain transfer functions, the input and output voltages are analog quantities, (in this case, vin and vout), allowing you to directly interface the model with other analog models.

The next step is comparing the frequency response of this z-domain model to its analog equivalent. Further, because the z-domain model depends on the sample period T, you should test for the lowest frequency that still produces acceptable results. This trade-off is common in sampled systems: On one hand, you need a high enough sample frequency to achieve optimum accuracy, and on the other hand, you want to keep the sample frequency as low as possible to ease DSP computation requirements.

Figure 2 illustrates the frequency response of the analog PI Controller, as well as responses of the digitized controller, at sample periods of 10, 1, and 0.1 msec. You would expect the PI Controller to exhibit high gains for low frequencies, due to the integrator, and then level out to the proportional gain value at higher frequencies. The proportional gain value is set at 750 (57.5 dB). Based on overall system requirements, you want the usable bandwidth of this controller to be around 200 Hz.

As the figure shows, all of the PI-controller responses look very good for low frequencies (the waveforms stack right on top of each other). As the frequency is increases, however, the digitization effects become apparent.

With a 100-Hz sample frequency (T=10 msec), the controller gain drops to −3 dB at about 45 Hz, which is insufficient. With a 1-KHz sample frequency (T=1 msec), the −3-dB point doesn’t occur until about 450 Hz, which meets the bandwidth requirements.

**Implement DSP controller digitally**

Because you have verified that the z-domain version of the PI Controller matches the analog version, you can now transform the z-domain transfer function into sequential statements. You accomplish this task by solving the z-domain transfer-function equation for the output value at the current sample clock. This implementation is referred to as a difference equation.

You can now describe the remaining blocks sequentially. These blocks are fairly straightforward to implement, primarily because they are algebraic in nature. Listing 2 gives a portion
of the VHDL-AMS code.

The model listing shows that all of the DSP-controller code has been implemented in a process. In VHDL-AMS, a process contains a collection of statements that are sequentially executed between the “begin” and “end process” keywords. Multiple processes are executed concurrently with respect to each other.

The signals that follow the keyword “process” in a process statement are referred to as the “sensitivity list.” These signals determine when the process will be executed. In this example, when a transition occurs on the system clock, smp_clk, all of the statements within the process are sequentially executed, after which the process suspends. The process remains in this suspended state until an event once again occurs on smp_clk.

There are two digital assignment statements in VHDL-AMS: variable assignments and signal assignments. Although both of these statements assign digital values at discrete points in time, variable assignments are local to the process in which they are declared, and immediately overwrite the previous value of the variable when assigned. Signal assignments are global to the given architecture, and schedule a new value to be applied to the signal when the process suspends. You make variable assignments using the := operator, and signal assignments using the <= operator. As you have already seen, you specify simultaneous statements using the == operator.

As shown, each individual system block is implemented with one or more assignment statements. As you update variables throughout the controller, the new values propagate down through the code until the two- to three-phase transform section. At this point, the variables are assigned to signals, which simultaneous statements then include.

The ‘ramp attribute provides a convenient means for converting from digital signals to analog quantities in VHDL-AMS. It provides a linear transition (100 nsec, in this case), between discrete level jumps to avoid discontinuities that such jumps would otherwise cause. You use another attribute, ‘above, to convert analog quantities to digital signals, by detecting analog threshold crossings. Together, these attributes provide for seamless mixed-signal modeling.

With this new model in hand, you can compare shaft-velocity waveforms between the analog and discrete DSP-controller implementations. Further, you can test the discrete controller at sample periods of 10, 1, and 0.1 msec. Figure 3 gives the results of these tests.

The digitized controller appears to accurately track the analog controller, provided the sample period is 1 msec or smaller. (This article forgoes the quantitative error analysis that you would typically perform to determine the ideal sample period and instead settles for a generalized visual comparison of the waveforms.)

### Implement a DSP controller in C

Now, consider one final step. You will eventually run this code on a processor, and it will most likely be authored in a formal programming language. It would be beneficial to test the actual code in the authoring language, so as to catch any translation errors between the VHDL-AMS model and the actual system algorithms.

Although C is not intrinsic to the VHDL-AMS modeling language, function calls are. With this in mind, you can now code the algorithms in C, and call this code as a “function” to the VHDL-AMS model. This, and other types of “multilingual” simulation are possible with the appropriate simulation environment. Listing 3 gives a portion of the C function.

Listing 3 shows that the DSP-controller algorithms are basically the same as those given in Listing 2, with slight syntax changes for the C programming language. The C function accepts three values from the calling VHDL-AMS model, performs the sequential operations, and returns four values back to the calling model.

Listing 4 gives a portion of the VHDL-AMS model that calls this C function. The C func-
tion is called FOC_C, and data passes to and from it as real vectors. The v_in_sig vector contains three elements that are passed to the function, and the v_out_sig vector contains four elements that are returned to the calling model.

Signal-assignment statements within the model convert returned values into signals. The signals are again used in simultaneous statements as shown in Listing 2. Figure 4 gives simulation waveforms comparing shaft velocities for the VHDL-AMS and C implementations with a 1-msec sample period. As the figure shows, the waveforms are virtually identical.

Author’s biography
Scott Cooper is a technical marketing engineer at Mentor Graphics Corporation. He has been involved in simulation technology for more than 12 years and is the author of The Designer’s Guide to Analog & Mixed-Signal Modeling. He is also a contributing writer to The System Designer’s Guide to VHDL-AMS. He holds a MS in systems management from the University of Southern California and a BS in electrical engineering technology from Metropolitan State College of Denver.

For more information
All the simulations in this article were performed using the SystemVision Analysis Platform. If you would like to simulate the designs discussed in these articles, or your own VHDL-AMS designs, you can download free SystemVision software as well as VHDL-AMS and application-based tutorials at www.mentor.com/system/.

Listing 1—VHDL-AMS discrete PI Controller

```vhdl
architecture behavioral of zPI is
  quantity vin across input;
  quantity vout across iout through output;
  constant T : real := 1.0/Fsmp; -- Sample period
  constant n0 : real := 2.0*Kp + T*Ki; -- z0 num
  constant n1 : real := T*Ki - 2.0*Kp; -- z-1 num
  constant d0 : real := 2.0; -- z0 den
  constant d1 : real := -2.0; -- z-1 den
  constant num : real_vector := (n0, n1);
  constant den : real_vector := (d0, d1);
begin
  -- ztf
  vout := vin'ztf(num, den, T);
end behavioral;
```

Listing 2—Discrete DSP controller in VHDL-AMS

```vhdl
proc : process (smp_clk)
begin
if smp_clk'event and smp_clk = '1' then
  -- Close loop
  w_err := w_cmd - fb_sf*wr;
```

-- PI Controller (difference equation and storage)
zo_PI_dly1 := ieqs;  -- store previous output value
z_PI_new := (Kp + T*Ki/2.0)*w_err +
               (-Kp + T*Ki/2.0)*zi_PI_dly1 + zo_PI_dly1;
zi_pi_dly1 := w_err;  -- store previous input value

-- ...Other blocks not shown...

-- Two to three phase transform
vsa_sig <= vsq;
vsb_sig <= -0.5*vsq - (sqrt(3.0)/2.0)*vsd;
vs_c_sig <= -vsa - vsb;
end if;
end process;

-- Simultaneous equations
vsa == vsa_sig'ramp(100.0e-9);
vsb == vsb_sig'ramp(100.0e-9);
 vsc == vsc_sig'ramp(100.0e-9);
theta ==  theta_sig'ramp(100.0e-9);
end architecture bhv;

Listing 3—C function (partial listing)

/* Get input data from VHDL-AMS model */
  w_cmd = in_vector[0];
  wr = in_vector[1];
  ieds = in_vector[2];
...

/* Close loop */
  w_err = w_cmd - fb_sf*wr;
...

/* Perform 2 to 3 phase conversion */
  vsa = vsq;
  vsb = -0.5*vsq - (sqrt(3.0)/2.0)*vsd;
  vsc = -vsq + 0.5*vsq + (sqrt(3.0)/2.0)*vsd;

/* Put data in output vector to return to model */
  out_vector[0] = vsa;
  out_vector[1] = vsb;
  out_vector[2] = vsc;
  out_vector[3] = theta;

Listing 4—VHDL-AMS model that calls C function

...
v_in_sig(0) := w_cmd;
v_in_sig(1) := wr;
v_in_sig(2) := ieds;

-- Call the C function
v_out_sig := FOC_C(v_in_sig);

-- Assign returned values to individual signals
va_sig <= v_out_sig(0);
vb_sig <= v_out_sig(1);
vc_sig <= v_out_sig(2);
theta_sig <= v_out_sig(3);
...

---

Figure 1—You can digitize the DSP controller portion of this induction-motor-drive system using VHDL-AMS.
Figure 2—The low-frequency responses of both the analog and digital PI controllers look good. As the frequency increases, however, the digitization effects become apparent.

Figure 3—The digitized controller accurately tracks the analog controller, provided the sample period is 1 msec or smaller.
Figure 4—Simulation waveforms comparing shaft velocities for the VHDL-AMS and C implementations with a 1-msec sample period are virtually identical.