

# VHDL-AMS modeling and simulation of BPSK transceiver system

Pavel Nikitin, Erik Normark, Cherry Wakayama, and Richard Shi

University of Washington, Department of Electrical Engineering,  
Seattle, WA 98195-2500, USA  
Email: {nikitin, ecn1, wakayama, cjshi}@ee.washington.edu

*Abstract*—This paper describes a methodology for top-down design, modeling, and simulation of complete RF system using hardware description language VHDL-AMS.

As a demonstration example, we consider a simple BPSK system (transmitter, channel, and receiver) and show the details of VHDL-AMS implementation for each elementary block (mixer, oscillator, amplifier, channel, etc.). Using these behavioral blocks, we simulate our BPSK system and evaluate system performance in the presence of noise. We show that the results of VHDL-AMS simulations match both Agilent ADS results and theoretical calculations.

This tutorial-like paper together with the developed library of RF blocks is targeted towards engineers who work on behavioral modeling and simulation of complete RF systems using hardware description languages.

## I. INTRODUCTION

One great challenge in the design of radio frequency (RF) communication system-on-chips is verification and debugging of its functional performance at different detail levels: top-level, sub-system level, and circuit level. To be practical, modeling and simulation tools used for that must fit into existing RF, analog, and digital design flows. Flow standards strongly depend on electronic design automation (EDA) companies providing the tools for system level RF simulation (e.g. Agilent ADS or Cadence SPW), analog circuit simulation (e.g., Spectre RF, Hspice), and digital synthesis (VHDL- and Verilog-based).

A popular approach to modeling and simulation of complex mixed-signal (digital/analog/RF) systems is behavioral modeling, which is more time efficient than full circuit-level simulation and is invaluable for verification purposes. Behavioral modeling in modern electronic design flow is commonly performed with high-level hardware description languages (HDLs) [1]. Two most often used HDLs are VHDL-AMS [2] (an IEEE standard) and Verilog-A.

Major EDA companies already provide design environments that can incorporate designs written in various languages, including HDLs (e.g. Cadence AMS Designer or Mentor Graphics ADVance-MS). They also provide libraries of functional blocks that fit into their appropriate design flows and allow one to perform simulations at different levels of abstraction (behavioral system-level and transistor circuit-level) from the same environment. This is especially valuable when analog circuits are added to previously designed digital blocks.

So far there have been very few works published in

the area of HDL behavioral modeling and simulation of RF systems. Sida et al. [3] and Ravatin et al. [4] presented results of ADVance-MS simulations for commercial transceiver circuits. Knochel et al. [5] performed behavioral simulations for an 802.11a system in Cadence. The results presented in the above works are strongly dependent on EDA simulation environment, use proprietary block libraries, and thus cannot be easily shared or run on arbitrary HDL simulators. Fakhfakh et al. [6] and Milet-Lewis et al. [7] reported their recent work on open VHDL-AMS library of RF blocks, but their main focus were phase-locked loop and frequency synthesizer functionality issues.

In this paper, we concentrate on VHDL-AMS behavioral modeling and simulation of a complete RF system, a binary phase-shift keying (BPSK) transceiver, and implementation details for various blocks that it is comprised of. We present our library of simple RF blocks that can be run in any HDL simulator with proper language support functionality. Any additional level of detail can be further added to these blocks, down to the circuit level inclusively. For VHDL-AMS simulations, we use Mentor Graphics ADVance-MS (ADMS). We also model our system in Agilent ADS (HPTolemy) and compare the results both to VHDL-AMS and to theoretical calculations.

## II. TRANSCEIVER SYSTEM

Consider the following example BPSK transceiver system shown in Figure 1 that consists of a transmitter, an antennas/propagation channel, and a receiver. Binary phase shift keying uses binary polar signals to modulate the phase of the carrier by 180 degrees. Digital data is converted to an analog signal by passing through a 1-bit D/A converter. The analog signal is then up-converted, amplified, and transmitted into the noisy channel. Received signal is amplified, down-converted, low-pass filtered, and converted back into the digital domain.

The system uses a modulation frequency of 2.45 GHz and a data rate of 250 Mbps. The power amplifier (PA) and low-noise amplifier (LNA) each have a gain of +10 dBV, and the channel loss is assumed to be -20 dBV. The noise in the channel is varied from -16 dBV to -4 dBV so that  $E_b/N_0$  values range between -6 dBV and +6 dBV.

The BPSK system shown above consists of several

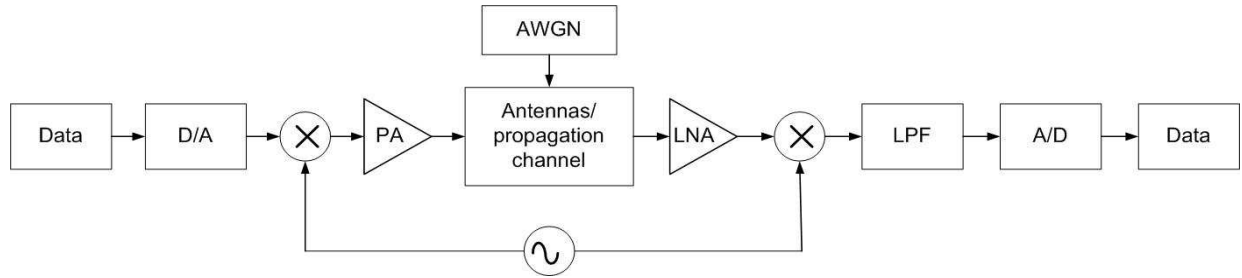


Fig. 1. BPSK system: transmitter, channel, and receiver.

blocks. VHDL-AMS implementations for all blocks are given in the Appendix while the blocks themselves are described in more details in the next section. To obtain an unbiased comparison between VHDL-AMS simulations and theoretical model, all blocks were assumed to be ideal and baseband pulse shaping and bandpass filtering were omitted.

### III. VHDL-AMS BEHAVIORAL BLOCKS

#### A. Modulator

To generate random stream of data for BPSK modulator, a non-periodic uniformly-distributed data source was created. The BPSK modulator itself consists of a 1-bit D/A converter, an oscillator, and a mixer. A D/A converter essentially shifts voltage levels of the source data and maps a logic '0' to -1V and a logic '1' to +1V. Multiplied by the oscillator frequency, this results in a BPSK-modulated signal. The modulator blocks are implemented in an ideal manner with no non-linearities or noise sources.

#### B. Amplifiers, Antennas/Propagation Channel

The power amplifier (PA) and the low-noise amplifier (LNA) are implemented as ideal pure gain blocks so that the functionality of the algorithm and relation to the theoretical system is quickly confirmed in simulation. The channel that contains both the effects of transmitting and receiving antennas and propagation effects is implemented as an ideal gain block with an additive white Gaussian noise (AWGN). Although other types of noise and interference are possible in a realistic RF environment, AWGN is most common and also allows one to compare BER simulation results to the theoretical calculations.

#### C. White Gaussian Noise Generator

Noise generator is necessary for calculating one of the most fundamental characteristics of a complete RF system: bit error rate (BER) vs. signal-to-noise ratio (SNR). Not all simulator platforms support functions for automatically generating white noise [5]. In our case, we used Box-Muller transformation to create a Gaussian distributed variable from two independent, uniformly distributed random variables. A periodic process computes

a new noise value at regular intervals and outputs that value scaled to the requested average power level.

Since a new noise value is produced at a given rate, one must make sure that in the transient simulation a new noise value is assigned to each original signal sample. For that, the sample rate of the original signal must be less than or equal to the rate at which noise is produced.

In our VHDL-AMS implementation of WGN generator, we found that with  $10^5$  samples, the mean and the variance of generated signal differ from ideal theoretical values by less than 3%, making this a very effective means of introducing AWGN into VHDL-AMS designs.

#### D. Demodulator

The BPSK demodulator consists of down-converter, oscillator, a low-pass filter (to eliminate aliases), and A/D converter (which works by sensing zero-crossings). Note that BPSK requires a coherent detection. For illustration simplicity, we use a transmitter's oscillator to provide a reference signal for the demodulator. In the next level of architectural complexity, a coherent detector needs to be added.

After demodulation, the signal is low-pass filtered and sampled at the middle of each bit period. Low-pass filter was implemented using Laplace transform package. BER was calculated by comparing demodulated signal to the original data stream.

### IV. SIMULATION RESULTS

Figure 2 shows three waveforms. Two top waveforms are transmitted and received digital data streams (taken on the input of D/A converter and the output of the A/D converter). The bottom waveform is the received analog signal (after filtering but before sampling) for  $E_b/N_o = 0$  dB. One can clearly see several bit errors (discrepancies between input and output digital data bits) for this particular time interval due to high noise level (SNR is 0dB) in the system.

The BER for ideal theoretical BPSK system is given by the following formula:

$$BER = 0.5 * \operatorname{erfc} \left( \sqrt{\frac{E_b}{N_o}} \right), \quad (1)$$

where  $E_b$  is energy per bit and  $N_o$  is noise power density.

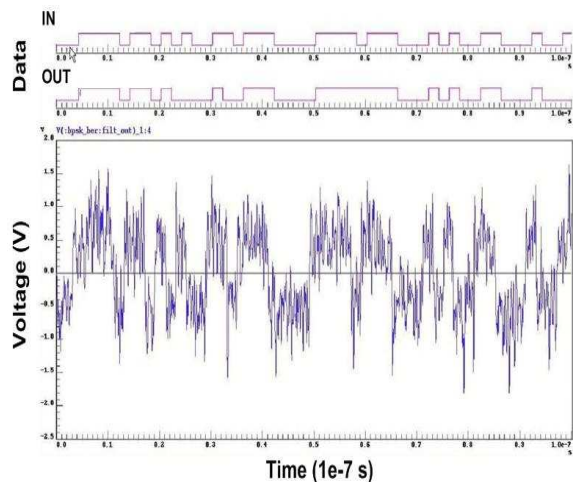


Fig. 2. ADMS simulated waveforms for ideal BPSK system ( $E_b/N_o = 0$  dB): input data, output data, and received baseband analog signal.

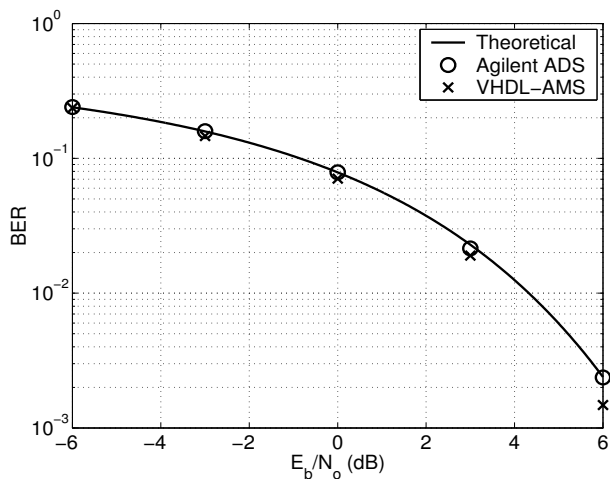


Fig. 3. Bit error rate for ideal BPSK system: comparison between theoretical formula, Agilent ADS simulations, and VHDL-AMS simulations (ADMS).

We have also implemented and simulated our BPSK system in Agilent ADS (HPTolemy). Figure 3 shows that results of VHDL-AMS simulations with fixed data sequence length ( $10^5$  bits) are very close (within 10%) to both Agilent ADS results and theoretical results from Formula 1. Note that for low BER (high  $E_b/N_o$  values), increasing the data sequence length would yield a better graph agreement.

## V. DISCUSSION

Noise modeling is a difficult issue in behavioral simulations and ultimately depends on the capabilities and language support of HDL simulators. The WGN generation method described here is suitable if simulation sampling rate can be fixed, as was the case here. Variable rate noise generation in VHDL-AMS requires the use of the "procedural" simultaneous statement, which is currently

not supported by ADMS.

In the example presented here, we intentionally kept all blocks straightforward and simple to demonstrate clearly all steps involved into the process of behavioral modeling and simulation of complete RF system. One can further reduce the level of abstraction and add a separate coherent detector as well as introduce more accurate and detailed models for PA, LNA, antennas/propagation channel, etc. Some blocks, such as mixer or oscillator, can even be implemented at a circuit level. However, in that case the system performance becomes dependent on many variables and the BER analysis becomes inherently more complex.

The simulation of our example system was carried out in ADVance MS simulator, but any HDL simulator with proper language support functionality can be used for that purpose.

## VI. CONCLUSIONS

In this paper, we described a methodology for modeling and simulation of complete RF system using VHDL-AMS. As a demonstration example, we considered a simple BPSK system and developed a library of behavioral level blocks for it. We simulated and evaluated system performance in a noisy environment and found that VHDL-AMS simulation results agree well with theoretical calculations and Agilent ADS simulations.

We target this paper towards the audience use of VHDL-AMS and thus describe in details our library of behavioral level VHDL-AMS RF blocks. We hope that this paper will help VHDL-AMS designers to better understand the process of HDL modeling and simulation for RF systems.

## REFERENCES

- [1] A. Doboli and R. Vemuri, "Behavioral modeling for high-level synthesis of analog and mixed-signal systems from VHDL-AMS," *IEEE Transactions on CAD of Integrated Circuits and Systems*, vol. 11, pp. 1504–1520, November 2003.
- [2] E. Christen and K. Bakalar, "VHDL-AMS – a hardware description language for analog and mixed-signal applications," *IEEE Transactions on Circuits and Systems*, vol. 46, pp. 1263–1272, October 1999.
- [3] M. Sida, R. Ahola, and D. Wallner, "Bluetooth transceiver design and simulation with VHDL-AMS," *IEEE Circuits and Devices Magazine*, vol. 19, pp. 11–14, March 2003.
- [4] J. Ravatin, J. Oudinot, S. Scotti, A. Le-Clercq, and J. Lebrun., "Full transceiver circuit simulation using VHDL-AMS," *Journal of Microwave Engineering*, pp. 29–33, May 2002.
- [5] U. Knochel, J. Hartung, and R. Kakerow, "Verification of the RF subsystem within wireless LAN system level simulation," *Design, Automation and Test in Europe Conference*, pp. 286–291, 2003.
- [6] A. Fakhfakh, H. Levi, N. Milet-Lewis, and Y. Danto, "Behavioral modeling of analogue and mixed integrated systems with VHDL-AMS for RF applications," *Proceedings of Symposium on Integrated Circuits and Systems Design*, pp. 308–313, September 2002.
- [7] N. Milet-Lewis, G. Monnerie, A. Fakhfakh, D. Geoffroy, Y. Herve, H. Levi, and J. Charlot, "A VHDL-AMS library of RF blocks models," *Proceedings of the IEEE International Workshop on Behavioral Modeling and Simulation*, pp. 12–14, October 2001.

## APPENDIX: VHDL-AMS BLOCKS

Architectural descriptions for all blocks are shown below. Because of space limitation, entity declarations are omitted.

### Random data generator

```
ARCHITECTURE behav OF RandData IS
  signal bit_out : bit := '0';
BEGIN -- behav
  bits_out_flop : process (clk)
    variable s1 : positive := seed1;
    variable s2 : positive := seed2;
    variable x : real;
  BEGIN -- process
    if (clk'event and clk = '1') then
      UNIFORM(s1,s2,x);
      if (x > 0.5) then bit_out <= '1';
      else bit_out <= '0';
      end if;
    end if;
  END process;
  output <= bit_out;
END behav;
```

### D/A converter

```
ARCHITECTURE behav OF DA_converter IS
  quantity V across I through
    OUT to electrical_ground;
BEGIN
  if (input='0')
    use V == -1.0;
  else V == 1.0;
  end use;
END;
```

### Oscillator

```
ARCHITECTURE behav OF Oscillator IS
  quantity V across I through N1 to N2;
BEGIN
  V==10**(A/20.0)*sin(2.0*math_pi*f*now);
END;
```

### Mixer

```
ARCHITECTURE behav OF Mixer IS
  quantity V1 across I1 through
    IN1 to electrical_ground;
  quantity V2 across I2 through
    IN2 to electrical_ground;
  quantity Vout across Iout through
    OUT to electrical_ground;
BEGIN
  V1==V1*R; V2==V2*R; Vout==V1*V2;
END;
```

### Amplifier

```
ARCHITECTURE behav OF Amplifier IS
  quantity V1 across I1 through
    IN to electrical_ground;
  quantity V2 across I2 through
    OUT to electrical_ground;
BEGIN
  V1==R*I1; V2==10**(Gain/20.0)*V1;
END;
```

### Antennas/propagation channel

```
ARCHITECTURE behav OF AntPropChannel IS
  quantity V1 across I1 through
    IN to electrical_ground;
  quantity V2 across I2 through
    OUT to electrical_ground;
  quantity AWGN across NOISE
    to electrical_ground;
BEGIN
  V1==I1*R; V2==V1*10**(Att/20.0)+AWGN;
END;
```

### Additive white gaussian noise block

```
ARCHITECTURE behav OF AWGN IS
  quantity Vo across Io through
    OUT to electrical_ground;
  signal noise_s : real := 0.0;
BEGIN -- behav
  noise_calc : process (noise_s)
    variable s1 : positive := seed1;
    variable s2 : positive := seed2;
    variable x1,x2 : real;
  BEGIN -- process
    UNIFORM(s1,s2,x1); UNIFORM(s1,s2,x2);
    noise_s <= SQRT(-2.0*LOG(x1))*
      COS(2.0*MATH_PI*x2) after rate;
  END process noise_calc;
  Vo == 10.0**(level/20.0)*noise_s;
END;
```

### Lowpass filter

```
ARCHITECTURE behav OF LowpassFilter IS
  quantity Vin across Iin through
    IN to electrical_ground;
  quantity Vout across Iout through
    OUT to electrical_ground;
  constant num: real_vector:=(1.0,0.0);
  constant den: real_vector:=
    (1.0,1.061e-10,5.629e-21,1.493e-31);
BEGIN
  Vout==Vin'LTF(num,den)*10**(Gain/20.0);
  Iout==Iin;
END;
```

### A/D converter

```
ARCHITECTURE behav OF AD_converter IS
  quantity A across I through
    IN to electrical_ground;
  signal D : bit := '0';
BEGIN -- behav
  data_out : process (A'above(level))
  BEGIN -- process
    if A'above(level) then D <= '1';
    else D <= '0'; end if;
  END process;
  D_out <= D; A == I*Ro;
END;
```