Automated Generation of Built-In Self-Test and Measurement Circuitry for Mixed-Signal Circuits and Systems

George J. Starr, Jie Qin, Bradley F. Dutton, Charles E. Stroud, F. Foster Dai and
Victor P. Nelson

*Dept. of Electrical and Computer Engineering
Auburn University, Alabama
starrgj@auburn.edu*

Abstract

This paper presents a software based approach for automatic generation of digital circuitry for synthesis and incorporation in a mixed-signal circuit or system to provide Built-In Self-Test (BIST) and measurement of the analog circuitry. The measurements supported by the BIST circuitry include frequency response (both gain and phase), linearity and noise figure. The measurements provide analog functional testing as well as the basis for on-chip compensation to improve yield during manufacturing and performance during system operation.¹

1. Introduction

A number of mixed-signal Built-In Self-Test (BIST) approaches have been proposed to detect faults in analog circuitry [1][2]. Early approaches were primarily aimed at defect-oriented testing of the analog circuitry for manufacturing and system-level testing [2][3]. More recent approaches are target analog functional test and measurement, or specification-oriented testing [4]-[6]. Some of the common functional measurements include frequency response (both amplitude and phase measurements), noise figure (NF), and linearity, also known as third order intercept point (IP3) [2]. The early defect-oriented mixed-signal BIST approaches could be developed with parameterized VHDL or Verilog for easy integration into most circuits and systems, as was the case in [3]. While Fast Fourier Transform (FFT) approaches have been parameterized in hardware description languages, such as VHDL, as in the case of [4] and [5], the hardware overhead and power consumption typically associated with an FFTs prevent it from being an efficient BIST approach, unless the FFT is an inherent component of the system. Other recent specification-oriented test approaches, such as [6]-[9], are more difficult to parameterize due to the computational complexity of trigonometric functions such as sine, cosine, and tangent [8]. Approximation techniques for these functions severely affect the quality of the test signals produced as well as the measurements made by the BIST circuitry [7]. Furthermore, the same computational complexity emphasizes the need to provide high-level synthesizable VHDL or Verilog descriptions for easy integration in mixed-signal designs.

Rather than attempting to fully parameterize the hardware description language, we present software that automatically generates VHDL for the mixed-signal BIST approach based on the user-defined parameters for the mixed-signal system to be tested and/or measured. We begin with an overview of the architecture and operation of the mixed-signal BIST approach in Section 2 followed by a detailed description of the software developed for automatic generation of mixed-signal BIST VHDL in Section 3. Examples of user interaction with the

---

¹ This work was sponsored in part by the National Science Foundation Grant CNS-0708962.

software and experimental results for implementation in Field Programmable Gate Arrays (FPGAs) in mixed-signal systems are presented in Sections 4 and 5, respectively, before the paper is summarized and concludes in Section 6.

2. Background

The mixed-signal BIST architecture illustrated in Fig. 1 has been developed and is capable of accurate on-chip analog functional measurements of frequency response, NF, and IP3 [6][7]. The majority of the BIST circuitry resides in the digital portion of a mixed-signal system and includes a direct digital synthesizer (DDS) based test pattern generator (TPG), a multiplier/accumulator (MAC) based output response analyzer (ORA), and a test controller. The only test circuitry added to the analog domain is one or more analog loopback paths to facilitate direct measurement of the test signals by the ORA. The number and location of these loopback capabilities determines the accuracy and resolution of tests and measurements associated with a given analog circuit [6].

Figure 1. Mixed-signal BIST architecture

The DDS-based TPG consists of three numerically controlled oscillators (NCOs) and utilizes the existing digital-to-analog converter (DAC) from the mixed-signal system to complete the DDS. Fig. 2 shows a more detailed view of the NCO used in the TPG. The phase accumulator is used to generate the phase word sequence based on the frequency word f and the initial phase word θ . Then the NCO utilizes a look-up table (LUT) to convert the truncated phase word sequence to a digital sine wave sequence. The frequency of the output sinusoidal wave can be determined as

$$f' = \frac{f \cdot f_{clk}}{2^n} \quad (1)$$

where n is the word width of the phase accumulator. The digital sinusoidal wave serves two purposes. One purpose is to produce an analog stimulus to the analog device under test (DUT) through the DAC. The other purpose is to provide in-phase and out-of-phase test tones for the MAC-based ORA. The ORA consists of two sets of MACs, as illustrated in Fig. 1. Each MAC receives the output from the DUT $f(nT_{clk})$. The other two inputs, $f_1(nT_{clk})$ and $f_2(nT_{clk})$, are the two digital sinusoidal waves generated by the TPG and used by the MAC pair to perform selective spectrum analysis at the desired frequency. The MAC-based ORA facilitates selective spectrum analysis with low area overhead and power dissipation, making the BIST architecture illustrated in Fig. 1 a more efficient approach for on-chip analog functional measurements when compared to FFT-based approaches.

When performing frequency response measurements, the top branch of MUX1 and MUX2 in Fig. 1 is activated and the DUT is driven by an in-phase signal with a frequency of f_1 . At the same time, the NCO3 generates a 90 degree out-of-phase signal at the frequency f_1 . In this way the ORA is able to measure the DUT output's amplitude and phase response at frequency f_1 . By sweeping the frequency f_1 over the bandwidth of interest, the DUT output's frequency response can be measured for both gain and phase [6].

When measuring NF, the top branch of MUX1 in Fig. 1 is activated and the DUT is driven by a signal with a frequency of f_1 . At the same time, NCO2 and NCO3 generate in-phase and out-of-phase signals, respectively, at frequency f . In this way the ORA is able to measure the amplitude and phase response of any noise at frequency f . By sweeping the frequency f over the bandwidth of interest, the noise can be measured, averaged, and compared to the DUT output response at frequency f_1 to calculate NF or the signal-to-noise ratio (SNR) [7].

Figure 2. Numerically controlled oscillator architecture

When performing linearity (IP3) measurement, the bottom branch of MUX1 in Fig. 1 is activated and the DUT is driven by a two-tone signal at frequencies f_1 and f_2 . Two accumulations are performed by the ORA to determine the spectrum at either the frequency pair f_1 and $2*f_1-f_2$ or f_2 and $2*f_2-f_1$. As a result, the magnitude of the fundamental and 3rd-order inter-modulation (IM3) terms can be obtained. Then the difference ΔP (in dB) between these two terms can be determined and the input referred IP3 (IIP3) can be calculated [6]-[9].

3. Generation of Hardware Description

Commonly in hardware description languages, many elements of a design can be parameterized to reduce the need to redevelop a hardware model after a minor design change to an existing, proven design. There are some elements which become increasingly difficult to parameterize as the level of complexity of the code base increases. This can result in a hardware description that is difficult to verify and maintain due to the vast number of conditional synthesis statements required to universalize a model. Alternately, if a large number of simplified models are created to account for every possible design consideration, then every model must be individually tailored to account for any future changes to the design.

To alleviate the problems associated with both of these common solutions, a different approach needs to be considered. For this, we propose a solution utilizing a custom written program, which can generate the correct hardware description given a set of system hardware parameters. In essence, this approach can utilize many software languages such as, but not limited to, C, C++, JAVA, or PERL. The single requirement is that this language must be able to create files. These files will eventually contain the final hardware model in a hardware description language such as VHDL or Verilog.

This is a common feature among modern programming languages that allows for a broad range of options for generating the hardware model. The design demonstrated in this paper utilizes C++ to generate VHDL models.

The mixed-signal BIST described above has many applications, but the components which comprise the hardware model remain relatively similar. The design will contain multiple NCOs, multiplexers, multipliers and adder elements as indicated by Fig. 1, but for any given application the size of the sine wave LUT may vary to accommodate the speed, size, and precision of the available analog to digital converter which will generate the analog waveform. In a similar fashion, different multiplier architectures such as array, Booth-encoded, or Wallace-tree architectures may be implemented to perform the multiplication. By exploiting the fact that the majority of these components remain unchanged, we can use exact models in every design using this mixed-signal BIST. But for those components to be altered based on specific system characteristics or other user parameters, the customized program can be used to insert, remove or modify portions of the hardware description to accommodate any changes.

This approach has other benefits beyond simplifying the design process and improving the uniformity and legibility of any generated hardware description. To generate an analog signal from digital elements, at least one NCO is implemented in the design. This is the foundation to generating sinusoidal waveforms, but depending on the system application, the speed and size of the waveform may require alteration. These sine wave values are loaded into LUTs in the descriptive model. But the calculation of these sinusoidal values are beyond the abilities most synthesis tools and hardware description languages such as VHDL or Verilog. Utilizing an external program allows use of prebuilt or customized libraries and functions of other language compilers to perform the complex mathematics. For example, we use the sine function included in C++ libraries. Then the values generated by this function can be converted into the desired hardware description. This method is not limited to simply using mathematical functions; any ability of that program language can be utilized.

Once the program has interpreted any user parameters into the appropriate hardware design decisions, the results can be saved as flags in the program with each flag corresponding to a user input design choice. After all decisions have been evaluated by the program, it can generate the hardware model. Each programming language has its own method for creating and writing files. In general the procedure is to first create a file, then line by line insert the contents into the file, followed by closing the file. Following this guideline for the C++ language, we must first create the blank result files (*.vhd) for a VHDL model. In C++ this is easily performed using the *createfile* function. Then using functions such as *writefile* or *fprintf* the contents can be written into the files. The exact contents are determined by the flags which have been set and formatted to appear as VHDL statements. In conjunction with these hardware descriptions, additional required files can be created such as any project information needed for synthesizing these models or constraints files for configuring the layout procedure.

At this point we have properly formatted files containing the desired hardware description and constraint information. Many synthesis tool sets can then automatically be called to synthesize, layout, route, generating timing information, and even to generate a configuration file for FPGAs. It is in FPGAs where this method is useful as rapid prototyping of multiple design iterations is a proven cost effective approach to test designs prior to implementation in an application specific integrated circuit (ASIC).

4. Device and Parameter Selection and Program Interface

Most hardware models have some level of parameterization and the presented mixed-signal BIST design is no exception. As presented, there is a level of parameterization that will result in code that is too complex to easily optimize. Therefore using the presented software-based method, it is possible to create the mixed-signal BIST model without the complexity of utilizing many *generate* statements.

The mixed-signal BIST model can be separated into NCOs, digital multiplexers, adder units, multipliers, and accumulators for the TPG and ORA as show in Fig 1. Additional components can be added to allow for automated testing and recording of the results. These components represent a test controller containing basic logic elements and RAMs for storing the resulting measurement data. The possible addition of a communication interface for controlling testing and reporting results can be added as well. As there are a large number of communication interfaces possible, using *generate* statements to insert a particular interface and associated connections to the hardware model becomes a more challenging task which this method alleviates.

The NCO is created with three main components: accumulator, phase truncator, and sinusoidal LUT. The most critical part of the NCO is the sinusoidal LUT. This LUT contains the phase location of the sinusoidal waveform at regular increments. This converts a linearly incrementing number on the input of the LUT into trigonometric increments on the output. This LUT can be connected to a wide variety of DACs which may operate with signed or unsigned numbers. Furthermore, the number of bits in the LUT output word will vary with the resolution of the DAC. Rather than truncating input bits to the DAC, the LUT can be customized for maximum efficiency. Given how many bits the phase accumulator will input into the LUT, how many bits the input word to the DAC will require, and whether the DAC operates using signed or unsigned numbers we can generate the LUT information utilizing the equation

$$phase = A \cdot 2^n \cdot \sin\left(\frac{2 \cdot \pi \cdot f'}{2^m}\right) \quad (2)$$

to determine the value of the LUT at each location. In this equation, f' represent the value seen on the input of the LUT, m represents the number of bits entering the LUT, and n represents the number of bits entering the DAC. Finally, the value A represents the relative amplitude from 0 to 1. To minimize the space required to generate a sinusoidal LUT, only one quarter of the waveform needs to be encoded into the LUT as it repeats after changing direction every quarter wave. Therefore, the two most significant frequency word bits may be truncated before the LUT. These bits are instead used to decide in which direction and from which starting location the LUT is addressed. By plugging the above formula into the standard sine function in the C++ math library, we generate a unique value for every possible input combination. With each possible outcome described in VHDL using *case-when* statements, we regenerate a discrete sine wave from LUTs. The additional logic required to access elements of this array is fixed and can be simply added by the software to the VHDL file which contains the LUT hardware description.

The final TPG component is the phase accumulator designed to take an increment value, called the frequency word, and add it to the accumulator each clock cycle. The value of the frequency word determines the frequency of the resulting sine wave; and the value in the accumulator determines the current location in the phase of the

sinusoidal waveform. Since these elements are fixed in definition, but not in size, they can be simply defined using parameterized VHDL. Leaving them as parameterizable components allows the developer some level of simple customization after the model has been generated. Alternatively, the program can be configured to generate a fixed phase accumulator, which can reduce the area occupied by the accumulator.

The ORA accumulators are generated in a fashion similar to those used in the TPG. The size of the accumulator can be specified in parameterized VHDL or as a fixed value. Unlike the accumulators, the adders units and multiplexers in both the TPG and ORA may need to be a fixed size for a given design. Their input width is determined by the size of the DAC and ADC, which also determines the NCO output word size. Because the program collects this data to generate the sine LUTs in the NCOs, it can also calculate and generate VHDL models for adders and multiplexers.

The final required component in the model is the multiplier unit. In many FPGAs there are digital signal processing cores that contain multiplier units. However, in an ASIC, these components do not exist and must be generated (usually at a gate level for complex or high performance designs). Therefore, in addition to automated configuration of the NCO, the program offers configuration of the multiplier by selecting the type of multiplier core to be used: DSP based multiplier for FPGA design, array multiplier, Booth-encoded multiplier, or Wallace-tree multiplier. But selecting these multipliers is only half the challenge as the size of the multipliers is also determined by the size of the sine waves that they are multiplying. Once again rather than generate a large array of conditional *generate* statements in VHDL, simple mathematics in C can be utilized to calculate and minimize the combination of adders and logic gates which form the multiplier. After the calculation is complete, VHDL can be automatically generated to implement the multipliers at the user's request.

This concludes the basic elements that make up the foundation of the mixed-signal BIST core, but realistically a model must contain more elements to make it functional. These additional elements may include a test controller, clock interface, debug interface, or communication port. The test controller typically consists of registers and counters tied to control and calibration signals. The size of each signal can be fixed or parameterized as the model they connect to is fixed in size for each signal. The contents of the test controller however are specific to the system in which the mixed-signal BIST is to be implemented. Therefore, if the testing procedure is known in advance, it can be written into the C++ program. If it is not known, the C++ program can generate a shell wrapper model to which a custom controller may be connected.

The clock interface is an important feature of the mixed-signal BIST. It provides a consistent timing source so that the NCOs can generate waveforms at precise frequencies. It also allows the test controller to determine the time to perform each test has elapsed and perform any calculations to derive frequency response or the linearity of the DUT. The clock signal may be generated from a number of sources. If it comes from an external source, it can be an external crystal oscillator. Alternatively a faster source can be used if it is derived from an internal oscillator, such as a ring oscillator. Given these options and the desired operating frequency, a VHDL model can be created within the C++ program to implement the oscillator connection or implement the ring oscillator design using buffers and inverters to achieve the desired frequency. Realistically, the placement of these elements within the device will have a major impact on the timing considerations which cannot be predicted until the developer places the components and routes the interconnections.

The mixed-signal BIST is designed for two types of applications. The first is autonomous control and calibration of the DUT. The second is user controlled calibration and testing of the DUT. In the first situation there is no need to implement a user interface as the device is to run autonomously under the control of the test controller. If an optional user interface is required, then the communication channel, registers and RAMs must be added to capture the test data. To accommodate this, the program is designed to place RAMs at key locations in the design so that every sinusoidal signal can be captured and transmitted over the user interface. The program incorporates designs for three types of communication interface: Boundary Scan, UART, and SPI.

After each option has been selected the developer only needs to choose the build option. This instructs the program to consider each option selected and test for incompatible selections. The program then generates the VHDL files for the complete model in addition to any constraint files needed. To allow a developer the ability to modify each option in the design process a simple DOS based application was developed. As illustrated in Fig. 3, this application allows the user to select one option at a time and modify it based on a list of supported options. When the program is terminated, a save file is created to record the developers options so that when the program restarts the same options are preselected.

Figure 3. Program to generate VHDL model for designer requirements.

5. Experimental Results

The C++ program was verified by generating several mixed-signal BIST designs which were then synthesized and downloaded in different Xilinx FPGAs. One such design for a mixed-signal system with 12-bit ADC and DAC connected to a Spartan-3 S200 FPGA was generated with the parameters shown in Fig. 3. By executing the program with the options shown, a VHDL project including VHDL files, constraints files for pin assignments, and a project file for Xilinx ISE 10.1 were automatically created in the program's working directory. The design after synthesis, placement, and routing is shown in Fig. 4. This design was used to take measurements of the mixed-signal system shown in Fig. 5, which contains a 12-bit signed DAC, tunable lowpass filter, tunable common-collector amplifier (for introducing non-linearity), an analog 2:1

multiplexor, and 12-bit signed ADC arranged as in Fig. 1. The lowpass filter in series with the common-collector amplifier is collectively referred to as the tunable DUT.

Figure 4. Spartan-3 S200 FPGA layout (left) containing analog BIST structure. Mixed-signal test board (right-top) with Digilent S-3 Board (bottom).

The characteristics of three paths can be measured by controlling MUX3 and MUX4 as shown in Fig. 1. Only the path through the DAC and MUX3, or the path through the DAC, DUT, and MUX3 can be measured with external equipment. Frequency response and IP3 measurements were taken for each of the external paths using the BIST circuitry implemented in the Spartan-3 FPGA. These measurements were retrieved via Boundary Scan and compared to a simultaneous measurement made with an Agilent E4446A spectrum analyzer (SA) and the results of SPICE simulation of the tunable DUT. The results of simultaneous IP3 measurements of the tunable DUT with BIST circuitry and the external SA are shown in Fig. 5 for seven samples taken across the tunable range of the DUT. Note that the ΔP measured with the BIST circuitry is typically less than the ΔP value measured with external equipment. This is because any non-linearity introduced by the ADC is not measured by the external SA since the measurement is before conversion to the digital domain. Using the communication interface discussed in Section 4, we can monitor sinusoidal waveforms generated by the FPGA - a screen capture of which is shown in Fig. 6.

Figure 5. IP3 measurements of tunable DUT by BIST circuitry compared with external spectrum analyzer.

Figure 6. Waveforms generated by TPG and captured in RAMs include 500kHz and 510kHz sinusoids and resulting modulated waveforms.

6. Conclusions

Analog functional test and measurement using BIST techniques is complex and can require considerable knowledge and effort by designers for successful implementation in a mixed-signal design. In an attempt to universalize the digital portion of BIST hardware for easier implementation in a mixed-signal system, we find that parameterized hardware descriptions are not always the best solution to the problem. For complex designs which contain many elements with many possible configurations, parameterization and generate statements lead to overcomplicated and difficult to maintain models. To simplify the hardware description model, hardware elements can be conditionally generated using software. In addition, included trigonometric library functions can be used to generate some essential components of the BIST architecture. The final result is a program capable of automatically generating a custom mixed-signal BIST model that can be used in any FPGA or custom ASIC design.

References

- [1] M. Burns and G. Roberts, *Introduction to Mixed-Signal IC Test and Measurement*, Oxford Univ. Press, 2000.
- [2] L-T Wang, C. Stroud and N. Touba, *System-on-Chip Test Architectures*, Morgan Kaufmann, 2007.
- [3] C. Stroud, J. Morton, A. Islam and H. Alassaly, "A Mixed-Signal Built-In Self-Test Approach for Analog Circuits," Proc. IEEE Southwest Symp. on Mixed-Signal Design, pp. 196-201, 2003.
- [4] J. Emmert, J. Cheatham, B. Jagannathan and S. Umarani, "An FFT Approximation Technique Suitable for On-Chip Generation and Analysis of Sinusoidal Signals", Proc. IEEE Int. Symp. on Defect and Fault Tolerance in VLSI Systems, pp. 361-368, 2003.
- [5] J. Emmert, J. Cheatham, B. Jagannathan and S. Umarani, "A Monolithic Spectral BIST Technique for Control or Test of Analog or Mixed-Signal Circuits", Proc. IEEE Int. Symp. on Defect and Fault Tolerance in VLSI Systems, pp. 303-310, 2003.
- [6] F. Dai, C. Stroud and D. Yang, "Automatic Linearity and Frequency Response Tests with Built-in Pattern Generator and Analyzer," IEEE Trans. on VLSI Systems., vol. 14, no. 6, pp. 561-572, 2006.
- [7] J. Qin, C. Stroud and F. Dai, "FPGA-Based Analog Functional Measurements for Adaptive Control in Mixed-Signal Systems," IEEE Trans. on Industrial Electronics, vol. 54, no. 4, pp. 1885-1897, 2007.
- [8] J. Qin, C. Stoud and F. Dai, "Phase Delay Measurement and Calibration in Built-In Analog Functional Testing", Proc. IEEE Southeastern Symp. on System Theory, pp. 145-149, 2007.
- [9] J. Qin, C. Stoud and F. Dai, "Test Time of Multiplier/Accumulator Based Output Response Analyzer in Built-In Analog Functional Testing," Proc. IEEE Southeastern Symp. on System Theory, pp. 363-367, 2009.