Mixed-Signal Test Development using Open Standard Modeling and Description Languages

Ping Lu, Daniel Glaser, Guerkan Uygur,
Susanne Weichslgartner, Klaus Helmreich
Chair of Reliable Circuits and Systems
Friedrich-Alexander-University Erlangen-Nuremberg
Paul-Gordan-Str. 5, 91052 Erlangen, Germany
pinglu@lrs.eei.uni-erlangen.de

Armin Lechner
Konrad-Technologie GmbH
Fritz-Reichele-Ring. 5, 78315 Radolfzell
Dublin, Ohio 43017-6221
a.lechner@konrad-technologie.de [*]

ABSTRACT

A novel virtual platform is presented, providing CAD/CAT support for efficient test development and attempting to bridge the gap between design and test. The platform, which models and simulates the entire test environment, provides methodologies, model libraries and tool sets to enable design, debug and verification of all test relevant processes including fault analysis, test algorithm, load board and test program development concurrently with IC design and fabrication phase and later smoothly apply the results to various test systems. One major idea of our work is adopting the open standards approach to guarantee interoperability. In addition, modeling methodologies for virtual tester and virtual silicon are proposed to further enhance interoperability between virtual and real test. To give an insight on how such environment seamlessly integrates into the test development flow, an ADC test which is performed both on the virtual platform and the real tester is described. The simulation environment is built using SystemC/-AMS libraries.

1. INTRODUCTION

While SoC products and their intended test environments are growing more complex, the time and cost for developing Test Program Sets (TPSs: test programs, load board and associated documents) has become one of the major issues in design and manufacturing. Currently, the design and test automation support for analog/mixed-signal devices is still not as mature as for digital ones in the sense of engineering and performance. To overcome the situation, Virtual Test (VT [8][12][6]) techniques have been proposed and developed in the past decade to allow test engineers to develop, debug and verify TPSs in a simulation environment including models of Device Under Test and its target test system.

The VT techniques cover methodologies, model libraries and tool chains to collaborate between EDA tool and test system throughout the entire test development flow. An integral implementation of such method requires complete simulation models of DUT (Device Under Test), DIB (Device Interface Board), tester and even part of software - both test program and test system software (see Figure 1). Given the above, all aspects of a test can be designed and verified via simulation with the virtual test platform and proper tools. Obviously the modeling effort to build such environment is just as overwhelming as its benefits.

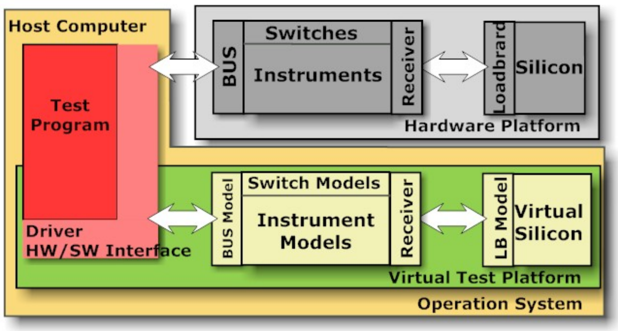

Figure 1: Overview of VT for test development

The VT approach has been around for well over a decade and many case studies could prove its value, however very few implementations are able to maximize its potential. The major reasons for this lack are missing tester-independence and -integration. In [9], we have analyzed the historical bottlenecks which prevented VT from use in day-to-day's practice and then proposed an applicable conceptual solution. This paper now presents in detail the implementation and utilization of an integrated Virtual Test Platform, consisting of methodologies, model libraries and seamless tool chains.

One essential characteristic of the solution is adopting the open standards approach to guarantee interoperability, namely using established test specifications and standards which are widely accepted by industry to define system interfaces, services, protocols and data formats. Standardized techniques enable data to be smoothly transfered from design to simulation and from simulation to real test. This has become increasing feasible with recent achievements in test technology.

In the conventional approach, the test engineer needs to perform lots of tedious and iterative work creating a test program that configures and controls instruments in order to bring signals to the DUT pins and capture responses. This strong coupling between test program and test resources is the major cause for both TPS cost and lack of test platform independence. The problem has been realized and explored by several test-related organizations (SCC20 [3], DoD [10], IVI [4]) over the last two decades. The resulting evolving advancements have led to plenty of significant IEEE proposals and industrial standards. Currently, the most acknowledged technologies are executable test specification and a compre-

[*]The reported work is performed as part of the project OKTOPUS (FKZ 01M3182), funded by the German Federal Ministry of Education and Research (BMBF) within the program IKT2020

hensive information framework [1] which defines descriptive languages for test program and test resource allowing information to be transfered from one life-cycle phase to another between components within a test system and between test system and outside world. Potentially, this also grants a new opportunity for Virtual Test to achieve a platform independent and yet easy integrateable implementation.

Another contribution is an efficient modeling strategy for Virtual Silicon and Virtual Tester. Due to the complexity of tester and their strong impact on analog test, it is essential to provide models of necessary accuracy with moderate computation cost and labour. Moreover, in order to smoothly transfer the test bench generated during simulation phase to the hardware platform, the virtual platform needs to provide the equivalent interfaces to test program as its hardware counterpart. In this paper, a unified solution based on SystemC/-AMS is adopted to describe the test relevant resources including hardware, software and interconnection supporting test and verification from system to electrical level.

In section two, a detailed architecture of the proposed Virtual Test Platform will be described. In section three, an experimental application based on a commercial ADC product is described first on Virtual Test Platform and afterwards in the real test environment. Test results will be compared in the fourth section, followed by conclusion and outlook in the last section.

2. IMPLEMENTATION OF VT PLATFORM

2.1 Architecture of Virtual Test Platform

A Virtual Test Platform shall always reflect the architecture of its host test system with acceptable tradeoff between fidelity and efficiency. In this case TPSs can be developed and debugged with their simulation models instead of interacting with real test HW. Figure 2 depicts the VT platform (right block) mimicking the entire tester environment (left block). The overall system features a layered architecture composed of six major layers – Test executive, Test application, Drive interface, Instruments, Device Interface Board and Device Under Test – and an information exchange framework that enables cooperation between components from different layers and cooperation between VT and tester environment. The major building blocks of software are:

The test application layer: Test application, usually written in programming language, details the test requirements on its intended target tester platform. Its essential tasks are stimulus creation, response capturing, post-processing, calibration, correctness determination, etc. Recent studies have proven that test requirements can be expressed in a descriptive way rather than programming approach. Later, such descriptive information can be understood and further executed by some intelligent service module through the dynamic allocation of the native test resources during run time. IEEE 1641 Signal & Test Definition (STD [7]) accommodates this objective. We implement this standard to formally describe and simulate test requirements and hence pave the way to interface simulation and real test.

The test executive layer: This is execution management software for running specific test steps/sequences. NI TestStand is used in our OKTOPUS test system. It is a powerful tool for organizing, controlling, executing test-flows and for

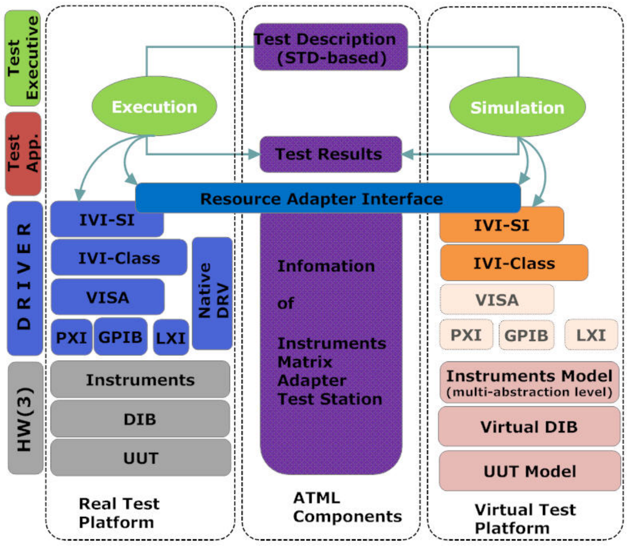

Figure 2: VT Platform Architecture Diagram

tracking DUT and reporting. Virtual test cases, when implemented with proper interfacing techniques, can also be managed within TestStand.

The driver interface layer: This is the layer between test program and hardware providing APIs to operate with instruments. For VT approach, it is beneficial to include the instrument driver into simulation model so that test program can interoperate with the model in a very equivalent manner to the real world.

ATML Information sharing framework: It is the key interface between VT platform and real-world tester that permits interoperation – shown in the center of Figure 2. ATML (Automatic Test Mark-up Language, IEEE 1671 [1]) provides formal description and semantic meaning for test domain knowledge in order to share and propagate test related information within and between tester environments. Moreover, IEEE-1641 STD is already covered in ATML as the the means to describe the test requirements and the instruments' capabilities.

Tool sets: Several tools, either proprietary or non-proprietary, are brought together on the common framework to facilitate TPS development flow. In our implementation, for example, an EDA-like schematic entry, named "TPConstructor", was developed as the Virtual Test Platform to manage simulation models, construct tests, control simulation and interoperate with tester's software.

In summary, the VT platform achieves integration into equally ATML compatible test system by constructing, simulating STD-based test graphically based on SystemC, and then both the test setup and outcome are captured and exported to corresponding ATML file. The execution of the test will be managed by tester's native software. Further, through encompassing driver interface in the instrument model, test program based on instrument APIs can also be supported.

2.2 Modeling philosophy of test instruments

A test instrument is a physical device with accompanying driver resp. firmware that is used to accomplish a purpose (e.g. stimulus generation, measure, switching). The strategies for modeling instruments are:

HW/SW Coupling Unlike previous VT attempts, our instrument model does not only cover hardware behavior but also that of the driver (interfaces between HW and SW application). As shown in figure 2, the HW part of the model, which specifies its functional behavior, is the major entity for simulation purpose, whereas the SW part describes its associated driver behavior. This strategy provides a comprehensive way to characterize the behavior of an instrument model through driver APIs in a manner very equivalent manner to the real world. Furthermore, it helps to reduce the workload of porting the test program from the simulation environment to real execution. In fact, later we will show that using SystemC as modeling language, the effort is quite minimized.

Generic Behavioral Model In test systems, instruments can be categorized into several classes according to their roles (source, sink, condition, etc.) and functionalities. This fact permits us to sketch a common generic behavioral model for each class which can reflect instrument functionality parameterized as defined in the driver. Such generic models are described on high abstraction behavioral level without respect to different hardware implementations, yet can map to a variety of different instruments. This strategy reduces the modeling effort of instruments significantly with considerable enhancement in simulation efficiency.

Parametric Electrical Model (PEM) For some tests, instruments need to be described in more detail including timing, waveform levels and waveform shapes with distortion. This requires all aspects of distortions which are introduced along the signal path before instrument ports need to be considered.

Generally, instrument imperfections can be divided into two groups: Static nonlinearity (SNL), superposition and random/non-determinism usually can be described mathematically with explicit allocation to inputs and outputs. The second group consists of electrical parts that are coupling with external circuitry (e.g. impedance). This part must be modeled under consideration of conservation laws – using declarative equations rather than assignments. One difficulty is that, in most cases the PEM depends on the instruments configurations, roles and settings. Thus, we need to find a way to determine the correct PEM for the instrument under different configurations. Different instruments with even the same capability will have different performance (e.g. resolution, precision, etc).

Separate Descriptive Information Models In order to facilitate a generic instrument model to reflect a specific real world instrument followed by a relevant PEM (under the current setting), information of the particularity needs to be stored in a separate file for run-time query. Currently, the ATML Instrument Description file can fulfill the purpose – using STD to describe HW ability.

Instrument Bus Module Optionally, this can be inserted to the instrument model to describe bus occupation and access latency for data transfer between test program and instruments, thus providing simulation and debug capability for resource plan and test sequence scheduling.

The overall instrument model (see Figure 3), is assembled with all above described sub-models, either statically or dynamically according to the test intent. The implementation of instrument generic behavioral model utilizes a hierarchical architecture based on STD signal library. Some example code in SystemC and simulation result are shown in List 1 and Figure 4. One difficulty for model assem-

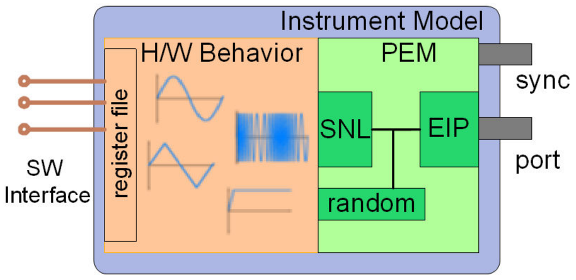

Figure 3: Overall Instrument Model

bly is to manage the data transfer and interaction among the models in terms of the internal coupling mechanism of the system during the automatic generation of test bench, i.e. dynamic model assembly. The overall test bench mixes models belonging to different domains (e.g. energy domain, signal flow, discrete event, etc) and described in different abstraction levels. A SystemC-based smart-router is developed to handle the problem and allocate appropriate channels to connect pairs of models.

```
BSC_SIM(AM) :
  public Modulator</*enum DY {NC, SYNC, GATE, SAG}*/ T>{
public:
  Ratio modIndex;
  Source<> *Carrier;

  BSC_SIM_CTOR(AM) : /*default values for parameters*/
    modIndex(0.3){...};

  void process () {
    dynamic_proc();  /*gate and sync process*/

    if (ACTIVE == sig.state()) {
      out.write(modulating(&(in.read()), NOW-delay));
    }
  }
  ...
  /* AM modulation routine e = (1 + modIndex m(t))*C(t). */
  double modulating(const double *x, double t) {
    if(!Carrier) return x[0];
    return (1 + modIndex*x[0])*Carrier->sample(t);
  }
};

/**
* FunctionGenerate SubClass : FgenModulateAM
* Description:  AM(t) = [M(t) + 1] * C(t) */
SC_MODULE(FgenModulateAM) :
  public IIviFgenModulateFM /*SW Driver Interface*/{
public:
  // HW interface
  sc_in<sc_logic> sync, gate;
  sca_sdf_in<double>  in;
  sca_sdf_in<double>  mod_in; /** external carrier in*/
  sca_sdf_out<double> out;

  /** Member Property */
  bool Enabled;
  /*Following attributes affect the behavior only when using Internal source*/
  double InternalDepth; /** units: % */
  double InternalFrequency; /** units: Hertz */
  IviFgenAMInternalWaveformEnum InternalWaveform;
  IviFgenAMSourceEnum Source;

  void * Carrier;
  ...
  SC_CTOR(IviFgenModulateFM) : sc_module(nm),
    am(new AM<SYNC_AND_GATE>("AM")), ...
  {   netlist();
    configure();  ... }

  void netlist();   // connect signal models
  void configure(); // translate parameters
  void map(Resource_t *rm); // map to hardware
private:
  IviFgenBase<1> *m_channel;
  boost::scoped_ptr<AM<SYNC_AND_GATE> > am;
  boost::scoped_ptr<Sinusoid<SYNC_AND_GATE> > sine;
  boost::scoped_ptr<Triangle<SYNC_AND_GATE> > triangle;
  boost::scoped_ptr<SquareWave<SYNC_AND_GATE> > square;
  ...
}
```

Listing 1: Function Generator: AM Module

2.3 Signal Path

The signal path from test system port to DUT will influence the signal integrity. Fur instance, our environment provides several parametric models of transmission line (e.g.

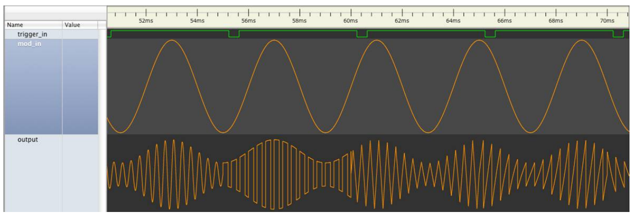

Figure 4: Simulation results of Function Generator: AM with different carrier

T, U, W types) for high frequency application, allowing to evaluate the signal deterioration along the path through simulation. The parameters for the model are calculated from measurement.

2.4 Modeling philosophy of DUT

The DUT models, either from designer during a top-down design flow or from sketch, have major influence on the test results, hence should be chosen carefully. There is no such model that can cover all the needs of VT especially when simulation cost is concerned. Therefore, the use of hybrid abstraction models, which only expose appropriate details, is widely acknowledged as DUT modeling policy in VT practice. It allows rapid exploration of more alternative test solutions through a massive virtual test. Furthermore, in this work, a Y-chart based approach [14] is developed to inspire test engineers to select appropriate abstraction models and assemble them together with proper interconnections for a particular test intent while taking into account DUT's structure and abstraction.

2.5 Modeling Language

Considering the aforementioned modeling strategy: HW SW coupling, hybrid abstraction modeling and open architecture, all these lead us to SystemC[5][11] and its extensions which are actively driving standardization processes (e.g. TLM) in the areas of SoC verification and test before 1st silicon. SystemC defines a modeling and simulation framework that provides one single language to describe and co-simulate systems from different disciplines with different models of computation and on different levels of abstraction.

In this paper, SystemC-AMS[13] – an extension of SystemC – is used to provide additional language constructs for modeling analog mixed signal behavior. The currently available version of SystemC-AMS is optimized for signal processing dominated applications and allows modeling of conservative linear (linear electrical networks) and non-conservative signal flow behavior with linear dynamic and non-linear static equations.

3. ADC CASE STUDY

3.1 Virtual Test Implementation

An existing mixed signal IC, AD1870, is used to perform VT in the above-mentioned environment. The whole test development flow is covered: from the very beginning of creating a hardware independent test description with a schematic entry tool (TPConstructor) over VT generation and execution unto finally propagating valuable results to the target test system (OKTOPUS TPS implementation) which will be shown later. The ADC1870 is a stereo, 16-bit oversampling ADC based on Σ-Δ technology. As widely known, simulation of such application is very CPU intensive, hence it's a challenging task to find efficient solutions. In this work, over 20 block models are associated to describe the ADC in different abstraction levels or views. Using the Y-chart based approach, the hierarchical combination of the models is evaluated before each test for yielding an appropriate accuracy and efficiency. Two tests cases are conducted and described in detail: Opens and Shorts Test (OAS) and dynamic performance test.

3.1.1 Opens and Shorts Test

OAS test, which belongs to DC parametric semiconductor validation, checks for faults in the protection diode circuitry of semiconductor chips to ensure that there is no short between each DUT pin and V_{dd}, V_{ss} or with each other.

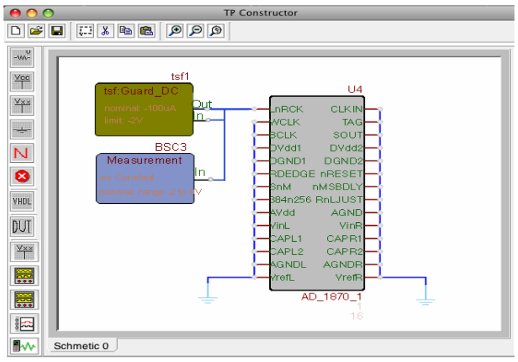

Figure 5: OAS Test Setup (TPConstructor)

Test Setup: The test setup for OAS is created with TP-Constructor in an EDA-like manner (see Figure. 5).

Here, the test is constructed in a signal-oriented way - specifying signals being applied or measured at the target DUT pins including complete information about signal type, signal role, signal attributes, relevant timing (delay, synchronization) and uncertainties. The descriptions are compliant to IEEE 1641 STD, which has defined over 60 most fundamental signal and measurement functions (BSCs) that might conceivable required on DUT pins, including sources, conditioners, events, measurements, digital and connections.

Beneficial from this approach, the test intent is quite instant at the schematic: -100μA current is drawn from one of the DUT pins (LnRCK) while all other pins are grounded; the voltage drop across LnRCK's internal diode is measured in the meantime. The source signal (Guard_DC) is built in a test signal library to provide reusability (see Figure 6). The underlining behavior is very clear: a current supply with over voltage protection.

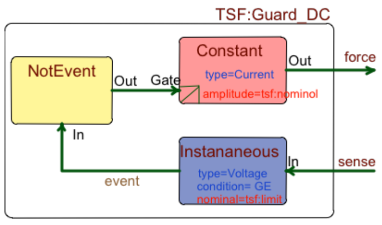

Figure 6: Internal Diagram of Guard DC

Once the schematic is captured, an ATML-compatible test description (see List 2) is generated accordingly. Later, the platform independent signal requirement is parsed to determine the suitable instrument.

Instrument Assembling: In this work, instruments in the OKTOPUS system are described in terms of the signal capabilities that they support. List 3 illustrates the information of NI PXI-4130 module which is employed in this test. Its associated model consists of two parts: function behavior (Guard_DC) and pin error.

```
<td:Parameter ID="sch01_para1">
  <!--Draw -100uA at AD1870 LnRCK with -2V protection-->
  <td:Value>
    <c:Datum xsi.type="td:StdSignalStimulus">
      <std:Signal>
        <mtsf:Guarded_DC name="tsf1" nominal="-100uA" limit="-2V"/>
        <std:TwoWire name="tsf1con1" in="tsf1" hi="LnRCK" lo="GND"/>
      </std:Signal>
    </c:Datum>
  </td:Value>
</td:Parameter>
<td:TestResult>
  <!-- Measure DC Voltage at DUT(1870) pin LnRCK -->
  <td:ValueDescription
      xsi:type="td:StdSignalMeasurement">
    <std:Signal>
      <std:TwoWire name="bsc3con2" hi="LnRCK" lo="GND"/>
      <std:Measure name="bsc3" As="Constant" In="bsc3con2"
        attribute="Amplitude" amplitude="range␣-2␣to␣0V"/>
    </std:Signal>
  </td:ValueDescription>
</td:TestResult
```

Listing 2: ATML Test Description (partial)

```
<InstrumentDescription name="NI␣PXI-4130">
  <hw:Resource name="IVIDCPwrBase">
    <hw:Description>The IviDCPwrBase capability group
    supports the most basic DC power supply capabilities.
    </hw:Description>
    <hw:Interface>
      <c:Port name="Ch1Hi"/>
      <c:Port name="Ch1HiSense"/>
      <c:Port name="Ch1Lo"/>
      <c:Port name="Ch1LoSense"/>
    </hw:Interface>
    <hw:Capabilities>
      <hc:Capability name="DCOut1">
        <hc:Interface>
          <c:Ports><c:Port name="DCOut1"/></c:Ports>
        </hc:Interface>
        <OneOf>
          <!--Voltage Output-->
          <std:Constant type="Voltage"amplitude=
          "range␣-6␣to␣6V␣resol␣0.01mV␣errlmt␣0.034%"/>
          <!--Current Output-->
          <std:Constant name="Current" amplitude=
          "range␣-200␣to␣200uA␣resol␣10nA␣errlmt␣0.03%"/>
          <!--Current Output with Over Voltage Protection-->
          <mtsf:Guard_DC name="guard_dc" nominal=
          "range␣-200uA␣to␣200uA␣errlmt␣0.03%+0.1uA"
          limit="range␣-6␣to␣6V␣errlmt␣0.03%+1.5mV"/>
          ...
        </OneOf>
      </hc:Capability>
    </hw:Capabilities>
    <el:ErrorModels>
      <el:ErrorModels id="Ch1Hi.slewrate" name="slewrate" parameter="0.08V/us"/>
      <el:ErrorModels id="Ch1Hi.output_impedence" name= "output_impedence"
        parameter="C␣10nF␣R␣0Ohm" In="Volt1␣Curr1"/>
      <el:ErrorModels id="Ch1Hi.TL" name="TL_T" parameter="Z␣50␣Ohm␣Td␣0.6ns␣"/>
      ...
    </el:ErrorModels>
  </hw:Resource>
</InstrumentDescription>
```

Listing 3: NI PXI-4130 Description File (partial)

Furthermore, PEM provides more accuracy. It gives guidance for prediction and evaluation of instrument performance. In this test, slew rate and output impedance (described in linear network domain) are used.

The overall instrument model is automatically assembled with these parts and appropriate interface – once adequate information is available. In addition, an IVI-Class compliant instruments library is built through the reuse of BSC signals and PEMs to form functional behavior and additional IVI wrappers. This allows the test program developed traditionally being debugged on VT platform through direct call to the instruments.

DUT Model: An AD1870 IBIS [2] model is used to efficiently characterize the ground clamp I-V behavior of protection diodes, i.e. a lookup table with appropriate equivalent circuit. The LUT modeling approach here provides a rapid simulation solution with adequate accuracy. In addition, other DC parameter tests such as input voltage threshold test (VIL, VIH), input leakage test (IIL, IIH), output voltage level test (VOL, VOH) can employ the same modeling approach.

In the end, a SystemC-AMS test bench is generated automatically from the schematic setup (see List 4).

```
test_result_t test_run_1(test_parameters_t *params) {
  ...
  tsf1 = RM.Require("Guard_DC", "tsf1");
  tsf1->nominal = "-100uA";
  tsf1->limit = "-2V";

  bsc3 = RM.Require("Measure", "bsc3");
  bsc3->As=RM.Require("Constant","as");
  bsc3->attribute="Amplitude";
  bsc3->amplitude="range␣-2␣to␣0V";
  /** netlist */

  conn = new SmartRoute("router",params->traceoption);
  conn(tsf1->out, adc1870_ibis_LnRCK_GND_clamp->LnRCK, params->trace["LnRCK"]);
  conn(tsf1->in, adc1870_ibis_LnRCK_GND_clamp->LnRCK,..);
  conn(bsc3->in, adc1870_ibis_LnRCK_GND_clamp->LnRCK,..);
  ...
}
```

Listing 4: Test Bench in SystemC/-AMS (partial)

3.1.2 *Performance Test: Dynamic Specification*

Dynamic specifications of ADCs are very important in high-speed applications and are very sensitive to the instrument performance. Most dynamic specifications can be retrieved by driving an ADC with a full-scale sinusoidal input within interested signal band.

Test Setup: The test setup for ADC1870 dynamic performance test is shown in Figure 7. The primitive components on the schematic are peripheral circuit on the load board to enhance ADC performance. As they're not configurable through tests therefore they are omitted when creating test description file, however are payed with well respect in simulation. On the other hand, the signal models which imply later implementation with right instruments are reflected in both files.

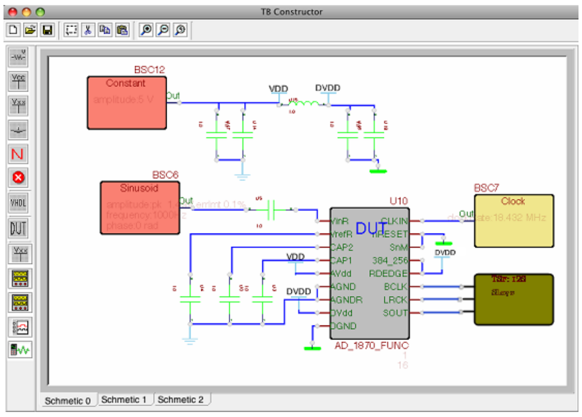

Figure 7: ADC Dynamic Test Setup

DUT Models: Unlike conventional analog simulator, SystemC-AMS provides an open framework allowing user to differentiate analog block (e.g. conservative/non-conservative, linear/nonlinear, etc) and select most appropriate modeling technique and simulation algorithms. Here we use synchronous data flow (SDF) to character ADC dynamic parameters via time domain simulation. The choice is appropriate because most of analog components of Σ-Δ modulator are clocked and hence the terminals between integrator blocks and filters may be considered to be non-conservative and the direction is mostly strait forward. Especially, SystemC-AMS is optimized for multi-rate SDF application

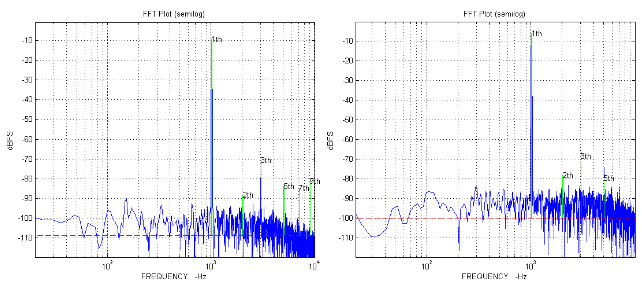

Figure 8: FFT for 1 kHz signal

which are very well suited for this oversample modulator. Computation time compared between SystemC-AMS SDF, Matlab and VHDL-AMS with same amount of samples (resp. $2s$, $3.7s$, > 15min at 8k samples), proves its efficiency.

3.2 Hardware Implementation

To achieve the selected tests (OAS, dynamic performance test) of the ADC, following NI PXI(e) Instruments have been used in our OKTOPUS system:

8130 Real-Time Controller
6652 Synchronization and CLK
2535 Switch-Matrix
4130 Device Power Supply (SMU)
5421 Arbitrary Waveform Generator
7842 Protocol Capture (FPGA)

One thing worth noting is FPGA card that can be configured for multi-function. In order to make it addressable from specific signal requirement - in our case I^{2}S protocol - we must describe its intend usage in its information file.

For dynamic performance, the ADC serial output stream into the PXI-7842 which implement I^{2}S-to-parallel converter and buffer. The samples are later transfered through the PXI bus to the real time controller and then to the host. On the host, performance is analyzed within Matlab. The FFT could also be implemented into the FPGA card after well verified in Virtual Test, in order to be more sufficient for production test.

4. RESULTS

It's clear that VT benefits both the designer and test engineer by enhancing verification/test debug efficiency and fault analysis through full access/control of the DUT model. Another major advantage is that, with accurate instrument models, VT allows verification of instrument performance and prediction of test results including influence from target HW. In order to show the feature, we performed dynamic parameter test with a legacy instrument (Philips PM 5193) as stimulus and without a bandpass filter. The same test is run in our VTP, with the models exactly describing PM 5193's performance including noise floor, nonlinearity and jitter. Results (Figure 8) show that through VT the instrument influence to the test results can be predicted efficiently. This feature allows test engineer evaluates instruments for high performance tests.

5. CONCLUSION AND OUTLOOK

Conclusion: VT is a powerful Concurrent Engineering technique in TPS development flow. In this work, an Open Standards Architectured-VTP was proposed in order to permit VT techniques deeply integrated into the TPS development flow. A pilot VTP is established, consisting of an extendable library of virtual tester, virtual silicon and EDA-like front end. The objective is, through our platform independent OSA-VTP, to bring simulation and test environment together in an interoperable manner, allowing the test development process going along efficiently in VTP and transfer the results to the real test system. Modeling strategies for instruments and DUTs are discussed with respecting to the trade-offs between modeling effort, simulation accuracy and efficiency. An ADC test is performed to evaluate proposed VTP. Results achieved are promising.

Outlook: At present the test programs in VTP and target test environment are still slightly different, though the syntax for configuring tester resources is close enough; further exploration is still required for better test program reusability. While the modern test standard and architecture is still evolving, some components are not final yet, it is beneficial to keep a close watch on them and replace the non-standard parts gradually. For even more accurate simulation results, it is necessary to take also the switch matrix and the interconnect into account. The model of signal paths in the system will be further developed for high performance tests. This was not the scope of this paper but will be presented in future work.

6. REFERENCES

[1] ATML http://grouper.ieee.org/groups/scc20/tii/.
[2] IBIS home http://www.eigroup.org/ibis/.
[3] IEEE SCC20 http://grouper.ieee.org/groups/scc20/.
[4] IVI foundation http://www.ivifoundation.org/.
[5] Open SystemC initiative http://www.systemc.org/.
[6] K. Einwich, G. Krampl, R. Hoppenstock, P. Koutsandreas, S. Sattler, and S. Munich. A Multi-Level Modeling Approach rendering Virtual Test Engineering (VTE) Economically Viable for Highly Complex Telecom Circuits. In *Proceedings user forum DATE*, pages 9–12, 1999.
[7] C. Gorringe, T. Lopes, and D. Pleasant. ATML capabilities explained. *Autotestcon, 2007 IEEE*, pages 178–189, September 2007.
[8] K. Helmreich and G. Reinwardt. Virtual test of noise and jitter parameters. *Test Conference, 1996. Proceedings., International*, pages 461–470, 1996.
[9] P. Lu, D. Glaser, G. Uygur, and K. Helmreich. A Novel Approach to Entirely Integrate Virtual Test into Test Development Flow. *Design, Automation and Test in Europe, 2009, Proceedings of*, April 2009.
[10] M. Malesich. Advances in DoD's ATS Framework. *Aerospace and Electronic Systems Magazine, IEEE*, 23(2):11–16, February 2008.
[11] G. Martin. SystemC and the future of design languages: opportunities for users and research. *Integrated Circuits and Systems Design, 2003. SBCCI 2003. Proceedings.*, pages 61–62, 2003.
[12] M. Miegler and W. Wolz. Development of test programs in a virtual test environment. *VLSI Test Symposium, 1996., Proceedings*, pages 99–103, 1996.
[13] A. Vachoux, C. Grimm, and K. Einwich. SystemC-AMS requirements, design objectives and rationale. *Design, Automation and Test in Europe Conference and Exhibition, 2003*, pages 388–393, 2003.
[14] R. Waxman. *High-level System Modeling: Specification and Design Methodologies.* Springer, 1996.