

ATLAS2K – THE FRAMEWORK FOR PRECISE AND EXTENSIBLE SIGNAL DESCRIPTIONS IN MODERN ATS

Keith Ellis, APSYS Ltd, +44 (0)1438 821555, keith@apsys.co.uk
Dick Delaney, RAF Wyton +44 (0)1480 52451 5664, dickdelaney@hotmail.com
Chris Gorringe, THALES Inst. Ltd. +44 (0)1202 872800, chris@racalinst.co.uk

ABSTRACT

This paper presents a summary of the new Standard to be published by the IEEE, ATLAS2K, which is the next issue of the IEEE Std. 716-1995 C/ATLAS. This revision represents a radical departure from the previous issues, in that it no longer represents a language per se, but it provides a framework for component libraries of Signal descriptions that can be readily deployed in today's Automatic Test Systems (ATS) with other standards such as the IVI Signal Interface from the IVI Foundation [1].

Keywords: author guidelines, ATLAS2K, paper style, publication, requirements

1 INTRODUCTION

ATLAS (Abbreviated Test language for All Systems), or to give it its full title IEEE Std. 716-1995 C/ATLAS has long been recognized as the primary language developed specifically for documenting test specifications because:

- a) it employs test technology related keywords – it talks of Volts and Amps
- b) its semantics ensure that the resultant test specifications have a disciplined consistency - the measurement of capacitance *has to be* conducted using units of Farads - that may be checked by ATLAS compilers that generate meaningful and precise error messages
- c) it is an international standard, regulated by the IEEE and as such it is non proprietary.

Perhaps the greatest strength of ATLAS is that test specifications are written in a form that is UUT orientated, that is, they are independent of test equipment such that they are portable between different test platforms subject to the availability of suitable resources. This means that end users are not tied to one particular supplier and are free to migrate their Test Program sets (TPS) between different ATS to overcome obsolescence, thus maximizing the re-use on their original implementation.

Nevertheless, ATLAS has not been without its detractors and the three most commonly voiced criticisms are:

- a) the standard is reissued on approximately five yearly intervals, culminating in the most recent version of IEEE Std. 716-1995 C/ATLAS. This interval is too long, especially because its prime purpose is to ensure that ATLAS keeps up with the changes in technology and to correct perceived areas of weakness
- b) in between those releases, users have only limited facilities to extend the capabilities of the language to address emerging technologies and often adopt convenient test equipment related workarounds that tend to restrict the test platform independence that is a primary strength of ATLAS
- c) it is expensive, or rather the tools are perceived to be expensive, often because they are not compared on a like for like basis with complete development and documentation suites of software. As far as TPS development is concerned the argument is fallacious for it may be argued that a proficient ATLAS test programmer can produce an ATLAS test program for similar a cost to that for which a C programmer can produce an equivalent test program in C.

ATLAS2K addresses these criticisms, and others, while building upon the inherent strengths of ATLAS, its signal definitions, and aligning it with modern programming practices, most notably object-oriented languages and Component Object Model (COM) Technology. Allowing ATLAS2K to be used within any COM enabled development Environment (IDE); Enabling the ATP programmer to concentrate of his UUT signals rather than fighting his test language; Providing the ATE independence to ensure Test Program portability across equivalent ATEs; and finally but not least, Maintaining a strong Test Requirement language standard to allow UUT Test Requirements to be written in an stylized and formal English while maintaining formal signal integrity.

2 THE ATLAS2K LAYER MODEL

2.1 Overview

ATLAS2K progresses beyond the Test Requirement Specification position by focusing on the Signal description as its core. In this way ATLAS2K is able to:

- a) unambiguously define all signals and base those definitions upon a sound mathematical foundation such that a signal defined by Project A can be accurately recreated and used by some other user working on Project B using different test equipment
- b) allow new signals to be defined by providing an architecture that permits the combination of existing signals

- c) provide a means for collecting together and extending libraries of signals having similar application domains
- d) provide a means whereby signals can be used and controlled in any object orientated native language
- e) allow test programs to be written for any ATS that supports the same ATLAS2K signals.

ATLAS2K is applicable to a wider audience that traditionally would have looked to use IEEE Std. 716-1995. This allows test information to pass more freely between design, test and maintenance phases of a project, allowing the same information to be directly usable across formally disjoint project phases. This better use of information, ultimately leads to a improved life cycle cost.

ATLAS2K achieves this by means of a layered architecture in which each layer is built upon the foundation of the lower layers. It is this layered approach that represents such a radical change from IEEE Std. 716-1995 C/ATLAS. For instance, ATLAS2K is no longer a purely textual language, but utilizes common interfacing methods to allow different user groups to take advantage of it core Signal definitions.

Fundamental to the ATLAS2K architecture is the goal to export to any run-time or development environment Signal Components, whose behavior is not only formally defined by the ATLAS2K standard, but where new Signal Components can be constructed to produce a more complex Signals and be equally valid across different platforms.. To support this goal ATLAS2K specifies a UUT Test Requirement language which uses these components and provides the actual formal definitions of each component for independent validation. Finally The ATLAS2K standard build on this footing to provide some useful domain specific features to show how the standard can be used to extend the Basic Components and to provide a migration path from previous ATLAS2K system.

The different aspects described above are obtained from the different ATLAS2K layers. Each layer performs a different function, and is targeted at a different user base, because the interaction between layers is defined by ATLAS2K the user does not have to concern themselves with all layers but only concentrate on the layer support they require.

The ATLAS2K standard is described in terms of the layer model shown in Figure 1.

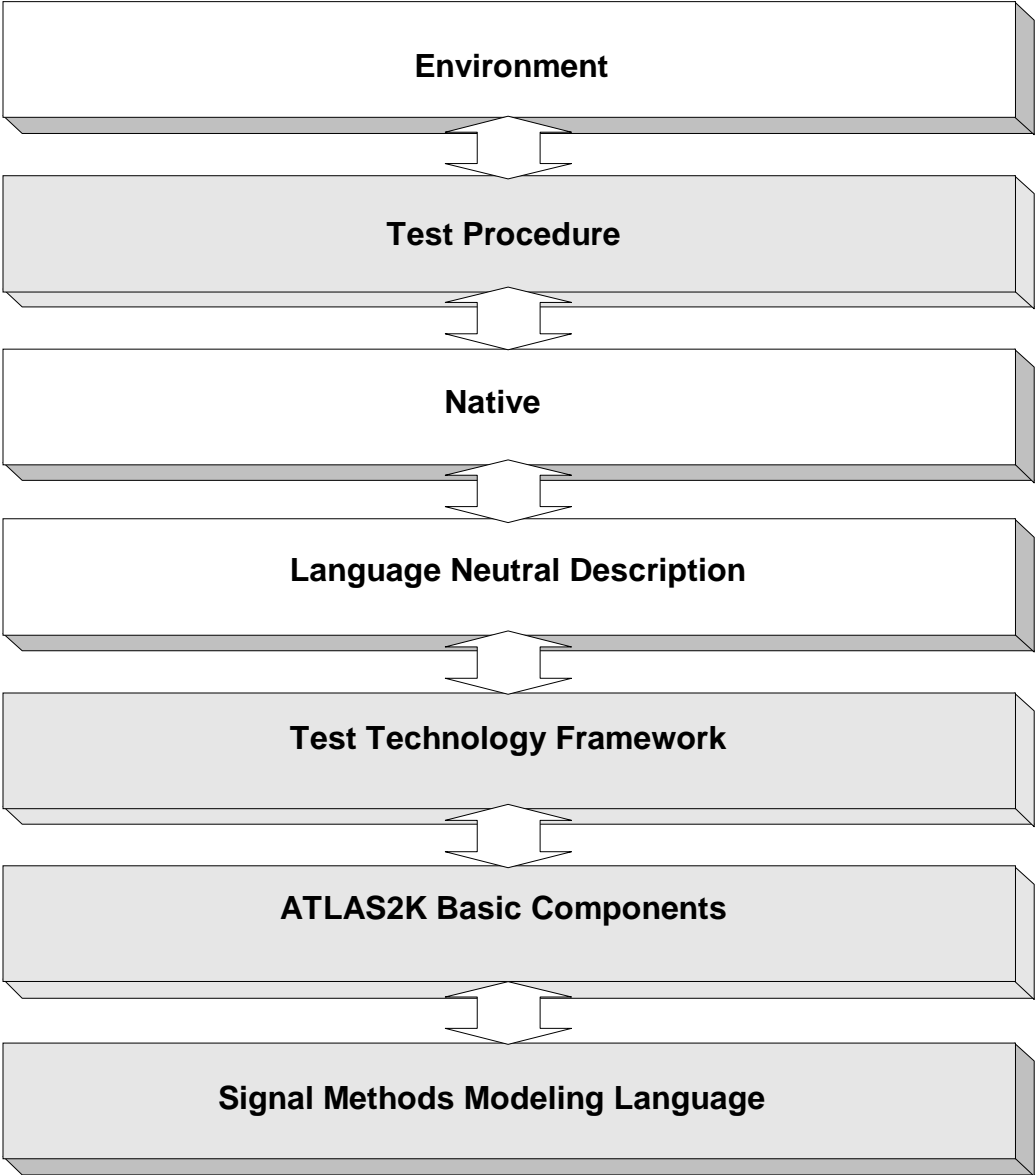


Figure 1. ATLAS2K Layer Diagram

The layers that are shaded in the Figure form part of the ATLAS2K Standard, the other layers are shown for completeness, as they are used by either the standard or an ATLAS2K implementation

2.2 Signal Methods Modeling Language Layer

The Signal Methods Modeling Language (SMML) provides a mechanism for describing the properties that support the description of the ATLAS2K Basic Signal Components. SMML provides for the structures to support signal component usage within a hierarchical signal classification system. By so doing, the signal models provide the basis for reuse that is essential to the evolution of ATLAS2K capabilities without a corresponding explosion in ATLAS Language nomenclature and complexity.

SMML provides the semantic behind the ATLAS2K signals definitions.

SMML allows the definition of signals, both analog and digital, as well as their functions or properties. It provides this capability by giving syntax and a number of predefined behaviors that can be combined as necessary to produce the desired signal definition. As such it acts as the tool box of fundamental Signal building blocks that can be put together in varying and a variety of different ways but at all stages it provides a formal definition of the signal being constructed.

SMML is included in the ATLAS2K standard as an Annex because its workings are largely transparent to most users.

SMML is a language it has language words, syntax, semantics etc. and as such it has meaning only to the native SMML environment (Haskell) to allow ATLAS2K to make use of its Signal definition semantics but not restrict users to the Haskell environment. ATLAS2K standard defines a convenient set of atomic building blocks that can be accessed and used through the COM interface but which are fully described by SMML; the ATLAS2K Basic Components.

2.3 ATLAS2K Basic Components

The ATLAS2K Basic Components provide reusable, formally described fundamental signal classes. These fundamental signal classes define the lowest level of signal available to any ATLAS2K environment and they are used to define *all* higher-level signals. For each fundamental signal class there exists an SMML description.

The ATLAS2K Basic Components build on the static Signal definition provided by the SMML definitions and also provide an abstract run-time control architecture that defines dynamic Signal behavior as typically encountered in a UUT test program. The ATLAS2K Basic Components also introduce real world elements such as CNX fields, essentially to the correct execution of a test program but not strictly necessary for Signal definitions. The choice of ATLAS2K Basic Components, i.e. the atomic building blocks, is centered more towards real world entities rather than mathematical concepts, providing a more engineering reference.

The generic name for the fundamental signal classes is **SignalFunction**. The behavior of a **SignalFunction** is controlled through its **Attributes** and is observable by the **Values** it produces. **Attributes** represent inputs to the **SignalFunction**, while **Values** represent its outputs.

Interaction between **SignalFunctions** is achieved through messages, of which there are two types:

- a) continuous (**Signals**)
- b) discrete (**Events**).

There are many kinds of **SignalFunctions** including:

- a) **Sources**
- b) **MeasurementFunctions**
- c) **EventConditioners**
- d) **CNX**.

A typical **SignalFunction** may be represented pictorially as shown in Figure 2.

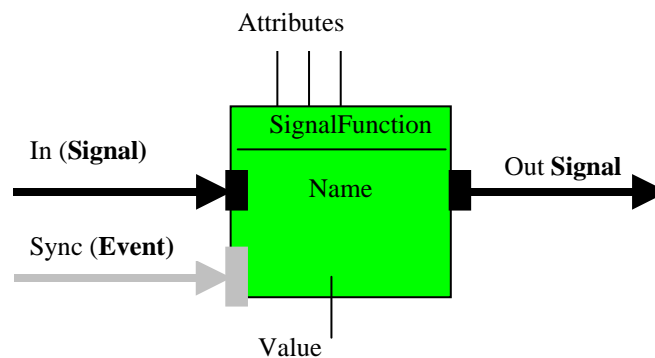


Figure 2. SignalFunction Diagram

SignalFunctions are designed to be grouped together in the form of a Signal Graph or net-list. This allows the user to customize any Signal they require by create new (and improved) Signals out of existing ATLAS2K BasicComponents.

The following is an extract from the standard to illustrate a Ramp Signal definition:

Triangular Signal<type: Current || Voltage || Power >

Definition

A **Triangular Signal** is a periodic signal whose instantaneous value varies alternately and linearly between two specified values. Its parameters are defined by two amplitudes and the event times that bound each transition. All event times are referenced to **LocalZero** which is reset at the start of each cycle. The transition time from the initial value to the alternate shall be equal to the transition

time from alternate to initial. **Triangular Signal** is derived from the **Trapezoidal Signal** but with zero dwell time at each maximum.

Properties

Sync is input synchronization <Event>

Amp1 is Amplitude 1 < type >

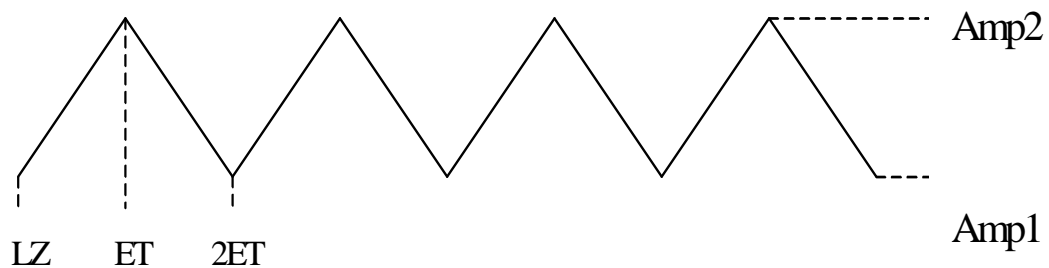
Amp2 is Amplitude 2 < type >

ET is the EventTime <time>

Out is Triangular Signal <Signal>

Description

The EventTime define the **event window** of a **linear** signal. The <slope> (**m**) of the **linear** signal is defined by the difference between the amplitudes divided by the **event window**. In a high to low transition the gradient is negative and in a low to high transition the gradient is positive. The <intercept> (**b**) is defined by Amplitude 1.



The ATLAS2K Basic Components provide a LND that allows an ATLAS2K implementation to provide common interfaces to each of the components. In this way programs or environment that utilize these LND can be directly transferred across platforms to other ATLAS2K implementations. This could include TPS portability or moving from a simulation environment analyzing TPS error detection to an ATE runtime system.

The ATLAS2K Basic Components are a finite number of building blocks, however it is recognized that in most cases it is more useful to reuse the same composite signals over and over again. To do this we don't want to add more and more Basic Components to the standard. To address this problem the standard introduces the notion of TTF. A library of pre-configured components built up from the ATLAS2K Basic Components, by defining how to create complex Signals from Basic Signals and then how to re-use these new Signals the ATLAS2K defines the way Signals can be extended without extending the standard. As an example of the notion the ATLAS2K standard defines the IEEE 716-1995 Signals in the first TTF, both showing how alternative domain TTF can be created and allowing for existing users to migrate to the ALTAS2K system in the future.

2.4 TTF Layer

The Test Technology Frameworks (TTF) provides libraries of reusable, formally described signals. The TTF components directly use the ATLAS2K Basic Components to construct a signal graph to describe the new Signal. The Basic Components rules on composition together with their SMML definitions provide the formal definitions to describe each TTF example.

The TTF may be used to accommodate a collection of domain specific signal definitions such as RF signals, data bus signals and so on or, as in the ATLAS2K standard, the TTF will contain signals that are equivalent to the functionality of the NOUNS described in IEEE Std.716-1995 C/ATLAS.

The signals of the TTF are complex signals comprising either two or more ATLAS2K Basic Components or some other combination of ATLAS2K Basic Components and other TTF signals.

The following example is provides the complete definition of an Amplitude Modulated Signal that is to be found in the ATLAS2K TTF.

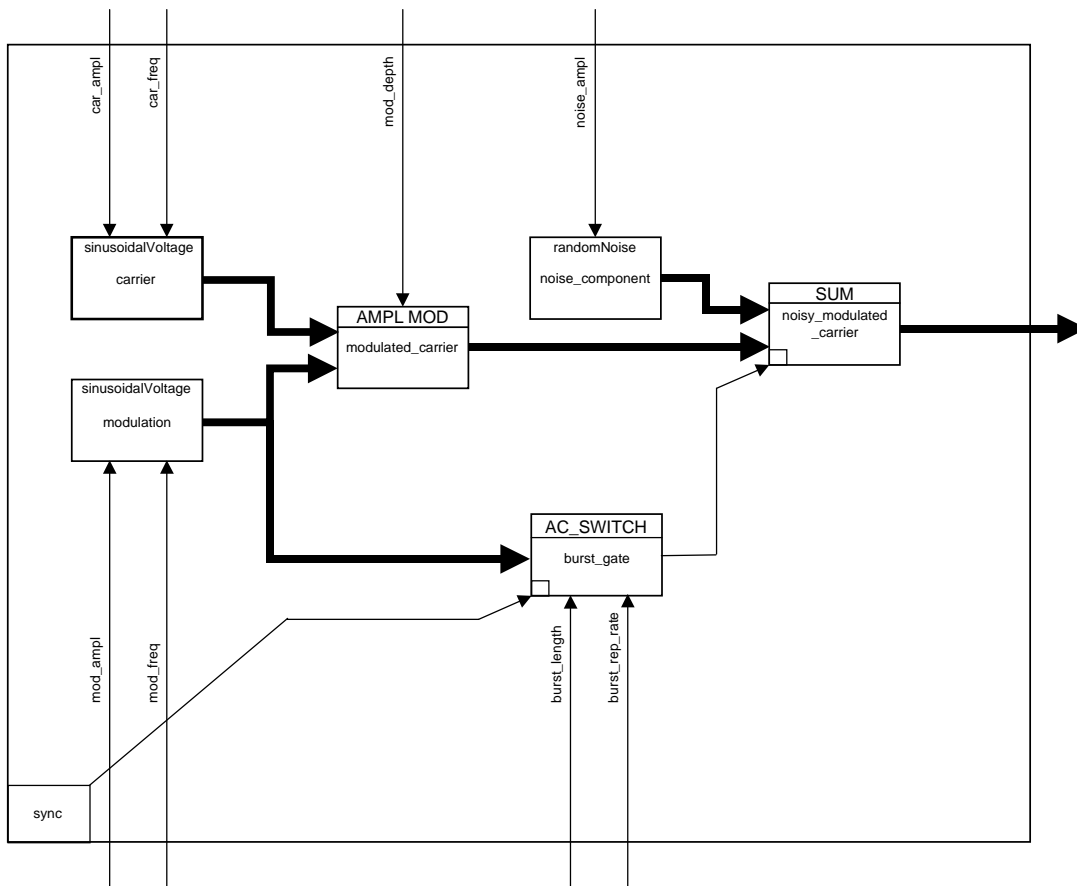
2.4.1 AM Signal

2.4.1.1 Interface

Description	Name	Type	Req/Opt	Default	Range
sync event	sync	event [In]	Optional	Enabled	
carrier amplitude	car_ampl	voltage	Required		
carrier frequency	car_freq	frequency	Required		
modulation amplitude	mod_ampl	voltage ratio	Required		
modulation frequency	mod_freq	frequency	Required		
depth of modulation	mod_depth	ratio	Optional	1 100%	
noise	noise_ampl	voltage	Optional	0	
burst length	burst_length	cycles	Optional	∞ (Note 1)	
burst repetition rate	burst_rep_rate	frequency	Optional	0 (Note 2)	
AM Signal	AM_SIGNAL	signal [Out]	Required		

Notes

- 1 In this context " ∞ " means that the signal is continuous.
- 2 In this context "0" means that the signal is continuous (i.e. there is no repeated burst).



2.4.1.2 Model

Type	Name/Signal	Terminal	Value	
sinusoidalVoltage	carrier	carrier amplitude	car_ampl	
		carrier frequency	car_freq	
sinusoidalVoltage	modulation	modulation amplitude	mod_ampl	
		modulation frequency	mod_freq	
AMPL MOD	modulated_carrier	carrier		
		modulation		
		depth of modulation	mod_depth	
randomNoise	noise_component	noise amplitude	noise_ampl	

Type	Name/Signal	Terminal	Value	
AC SWITCH	burst_gate	burst length	burst_length	
		burst repetition rate	burst_rep_rate	
		modulation		
		sync	sync	
SUM	AM_SIGNAL	modulated_carrier		
		noise_component		
		sync	burst_gate	

The TTF is the extensibility mechanism that allows users to create additional signal class definitions and is much more powerful than the EXTEND mechanism of IEEE Std.716-1995 C/ATLAS.

2.5 Language Neutral Description (LND)

The Language Neutral Description (LND) is a type library that provides a method for specifying all of the ATLAS2K Basic and TTF components and is written using an Interface Definition Language (IDL). The type libraries ensure that all implementation of the ATLAS2K signal definitions adhere to the same interface. The LND represents the contract between supplier and user, where the user has the knowledge that this is the same interface offered across implementations.

The ATLAS2K Standard provides, for each Basic Component and TTF component, a full LND that defines that component, itemizing its name, property name, allowed values, enumerate value types and so on. The LND removes any possibility of different behavior due to language dependent features.

2.6 Native Code Layer

The Native code layer is shown for completeness. The user of the ATLAS2K Standard will have tools (graphical development tools, compilers, and execution tools) that will provide the means to describe and execute the signal descriptions that are written in the carrier language for the test program development. The resultant native code may be the output of the compiler tools that are available in the carrier environment, such as C++, JAVA and so on. The ATLAS2K standard includes a clause that identifies the requirements of the Carrier Language

This layer provides reusable libraries for test program generation.

Why is the Native Code Layer mentioned if its not used by the standard? Because it's the Layer that most users will interact with, i.e. Test Programmers will not need to write test programs in

ATLAS2K but can use ATLAS2K signals and features in their known programming environments.

What about if I only want to capture UUT Test Requirements (and am not interested in Basic or other languages)? The ATLAS2K standard introduces the Test procedure layer. This layer recognizes the advantages of capturing UUT Test Requirements in an English style notation.

2.7 Test Procedure Layer

The Test Procedure layer defines the stylized English syntax and the underlying semantics required to define a signal in an “all-textual” test requirement. It will therefore be used like the previously published ATLAS Standards to describe signals and their control parameters and the operations to be conducted upon them.

Contained within this layer are the traditional ATLAS single action verbs such as SETUP, CONNECT, ARM and FETCH. The multi-action verbs are defined in terms of their single-action counterparts.

This layer not only defines the stylized English but also how it maps onto a Native Code Layer utilizing the TTF and ATLAS2K Basic Components. This means that writing any Test procedure using these layers definitions will result in the same program regardless of the implementers or ATE system

2.8 Environment Layer

The environment layer is described as being a user’s choice of graphical interface. It is not part of the standard, but it is shown for completeness.

Commercial Off-The-Shelf (COTS) software development environments such as Visual Basic and Visual C++ may be used for building TPSs, and Test Procedures can be developed to include ATLAS2K robust Signal Descriptions, while control and flow structures are provided by the native code development environment.

ATLAS2K will allow different end user application to be developed that can simultaneously co-exist, and provide the end user with benefits of choice and have a pick and mix of features rather than locking them into one system. This is directly attributable to the common interfaces and semantics provided by the ATLAS2K standard.

3 CONCLUSIONS

ATLAS2K is the culmination of a radical review of the ATLAS test programming language.

It provides the capability to describe and operate on signals. This capability coexists with the user's choice of operating environment including the choice of Carrier Language. Thus ATLAS2K exists as signal operations embedded in Visual C++, JAVA or any other object-oriented environment of the user's choice.

The strengths of the ATLAS2K signal definitions is so great that they can be deployed in other ATS architecture standards such as the IVI signal Interface. In time it is anticipated that systems designers from a variety of electronics industries will find advantage in using the signal definitions of ATLAS2K.

The legacy link to previously published ATLAS716 Standards is preserved in that the user can describe signal operations using keywords. These keywords now have formal definitions tied to the TTF and ATLAS2K Basic Components layer and are implemented using the ODL and Native Code layers. Furthermore, the parameters of the signals themselves also have a rigorous formal behavioral description within the SMML layer.

3.1 References

[1] IVI Foundation Website, <http://www.ivifoundation.org>