SONICS™

SystemC co-simulation for functional verification

Sonics, networks-on-chip & transaction level models

- Sonics is an IP provider of highly-configurable, high-performance Networks-on-Chip (NoC)

- SystemC IP models since 2005
  - *performance modeling and functional verification*
  - *TLM-2.0 models with OCP/AXI sockets since 2010*

- The primary reasons for this IP investment are two-fold:
  - *architectural executable for performance/complexity trade-offs*
  - *provide customers a fast model to validate performance on realistic traffic scenarios*
    - explore configuration and topologies against application goals (bandwidth/latency)
    - coupled with a powerful scenario traffic generation tool and analysis

SystemC TLM model as architectural executable

- New features and enhancements can be prototyped and measured for performance
  - *Transfer functions*
  - *Arbitration algorithms, QoS*
  - *Buffer sizing*
- Allows Sonics architects to foresee more scenarios
- Easier platform to test architectural features
  - *Less integration required than RTL*
  - *Especially with configurable hardware, prototyping can be done before all necessary "derivation" needed to integrate a system in RTL*
  - *Can operate against TLM-based behavioral models rather than a UVM testbench*

Cycle accurate TLM models

- In a NoC, micro-architecture subtleties can noticeably affect performance characteristics.
  - *unexpected "dead" cycles can be costly depending on use cases*
  - *breadth of configuration space requires self-checking*

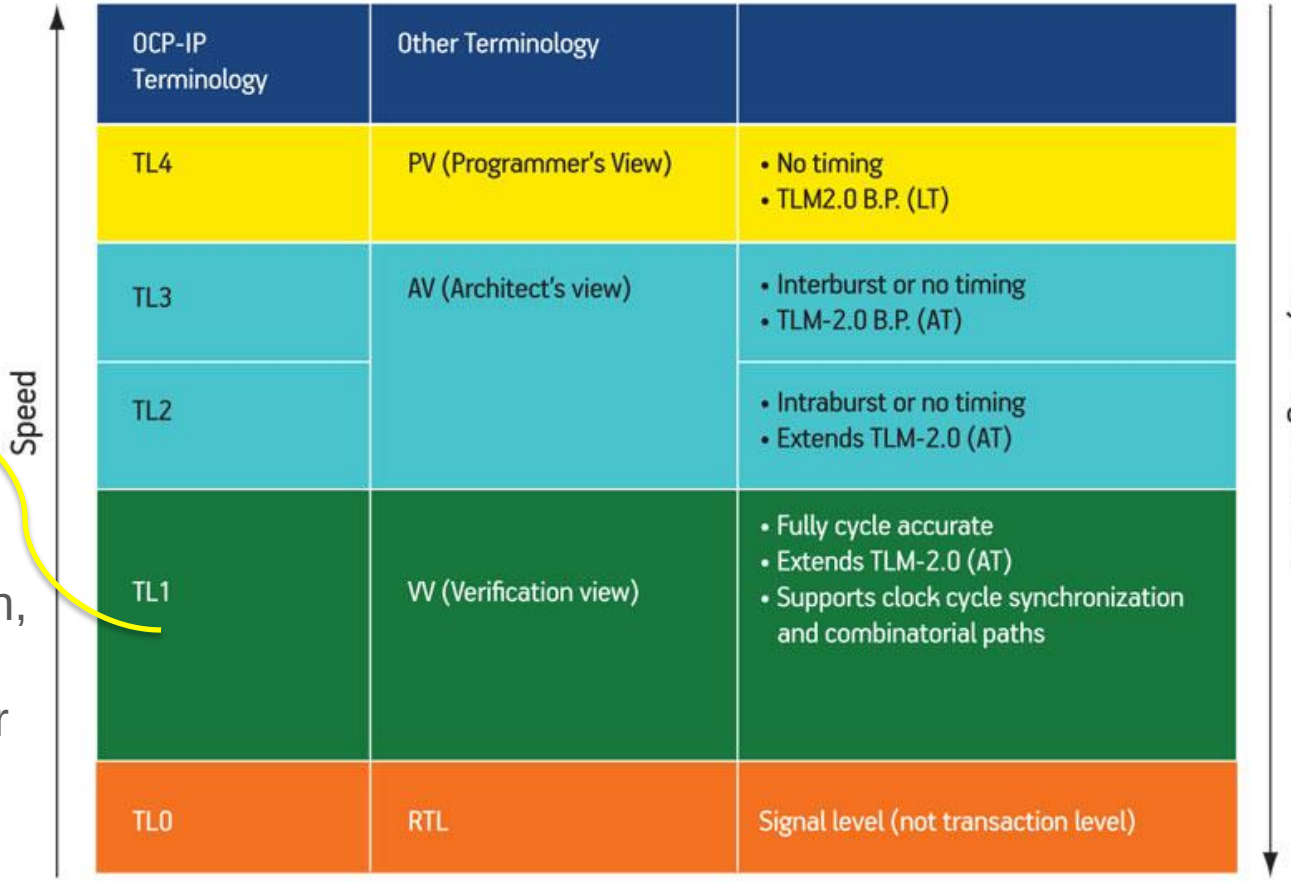
- While other cores in the SoC can be modeled approximately, the interconnect performance is too central to give up accuracy

- For that reason, all SystemC TLM models of Sonics IP are designed cycle-accurately with the intended RTL.

- While modeling at that level presents its challenges, it also gives a unique opportunity to leverage the model for RTL verification.
  - *dolve some inherently difficult verification problems*
    - bubbles and QoS issues that a typical UVM scoreboard does not look for
    - clock gating, power-down state, auto wake-up

TLM Layers

- We are here! Several terms have been coined
  - *TL1 (ocp-ip)*
  - *CT (Accellera)*
    - cycle-timed
- With thoughtful design, most of the code can be shared from higher abstraction levels



OCP-IP Terminology	Other Terminology	
TL4	PV (Programmer's View)	• No timing • TLM2.0 B.P. (LT)
TL3	AV (Architect's view)	• Interburst or no timing • TLM-2.0 B.P. (AT)
TL2		• Intraburst or no timing • Extends TLM-2.0 (AT)
TL1	VV (Verification view)	• Fully cycle accurate • Extends TLM-2.0 (AT) • Supports clock cycle synchronization and combinatorial paths
TL0	RTL	Signal level (not transaction level)

Source: Tech Design Forums
http://www.techdesignforums.com/practice/technique/how-to-create-adaptors-between-modeling-abstraction-levels

SONICS

What is Cycle-accurate or Cycle-timed or TL1 or …

- It has one (or more) clock ports (sc_in_clk)
- It models the protocol of each interface at the “Phase” level
  - *What is a phase? (some examples)*
    - Request/data handshake/response for OCP
    - Address Read for AXI (AR* signals from ARVALID=1 to ARVALID&&ARREADY=1)
  - *Model every phase of every transfer of every transaction*
    - By protocol start and time of each phase are unambiguous
    - Models the start and end of each phase in the same cycle as it would occur in hardware

- What is TL0?
  - *A model with interfaces at the pin level*
  - *Each pin is represented by a systemc port (sc_port)*
  - *sc_port templated with a data type mappable in a simulator to a verilog signal*

Transaction recording

- IP modules record customized Performance Data
  - *protocol related phases (AXI 5 channels)*
  - *buffer occupancy*
  - *arbitration state*
- Used to gather performance data and find bottlenecks
- Using SCV transactions
  - *custom structure recorder with begin/end time*
    - begin_transaction/end_transaction
  - *custom back end recorder*
    - sqlite
    - text
    - Novas fsdb
    - sdi2 – visible in Simvision, during simulation. Live!

RTL/SystemC co-simulation with Incisive

- Sonics benefits from the multi-language capabilities of the Incisive Simulator
  - *Fully integrated SystemC/RTL*
  - *SystemC Debug facilities*

- Configurability puts focus on automatic generation
  - *SystemC models are self-configured at elaboration time*
  - *Automatic generation/connectivity with variable port lists*

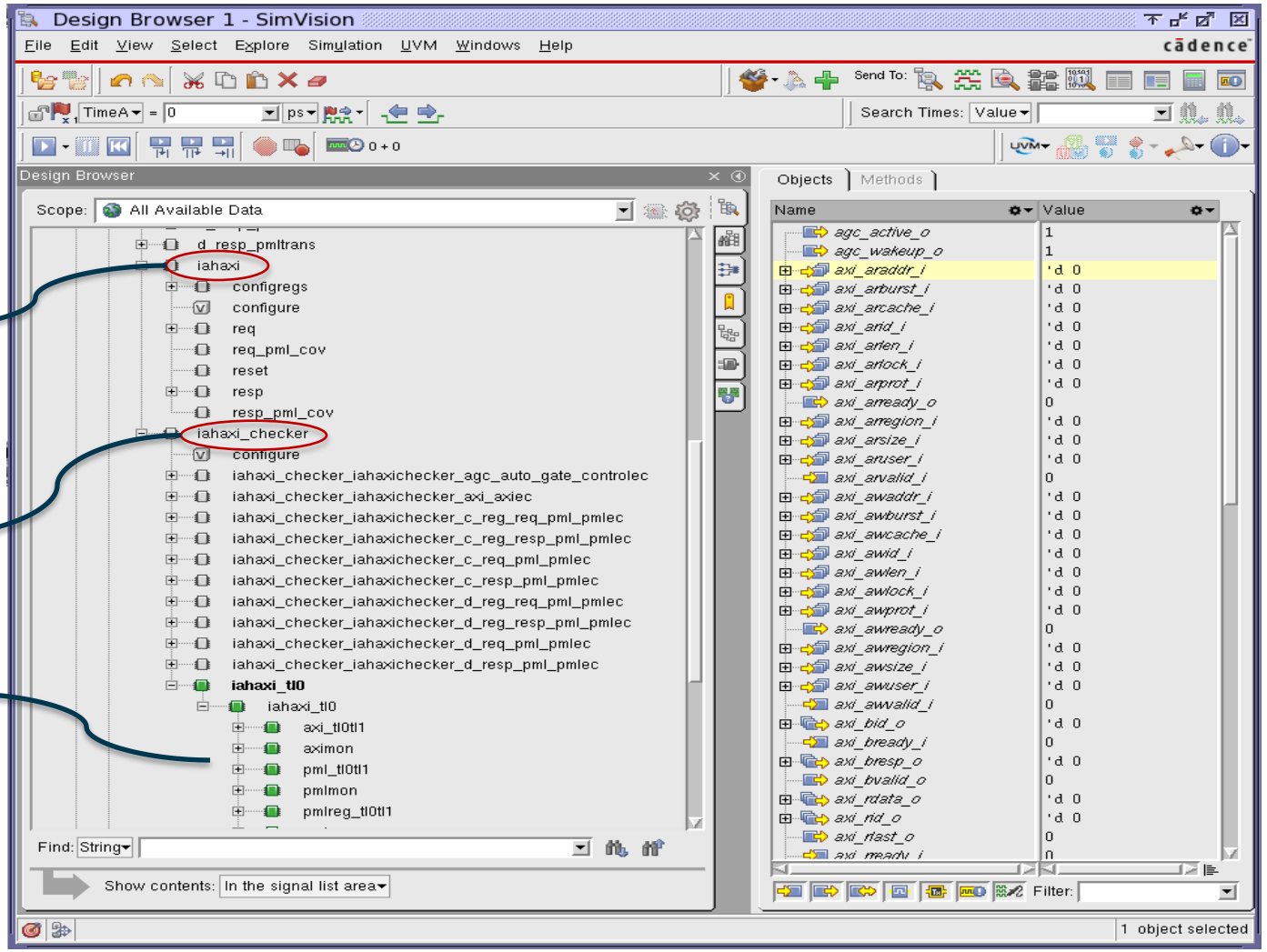
- Run-time checking with protocol-aware SVA equivalence checker (EC)

Design Browser with mixed RTL/SystemC

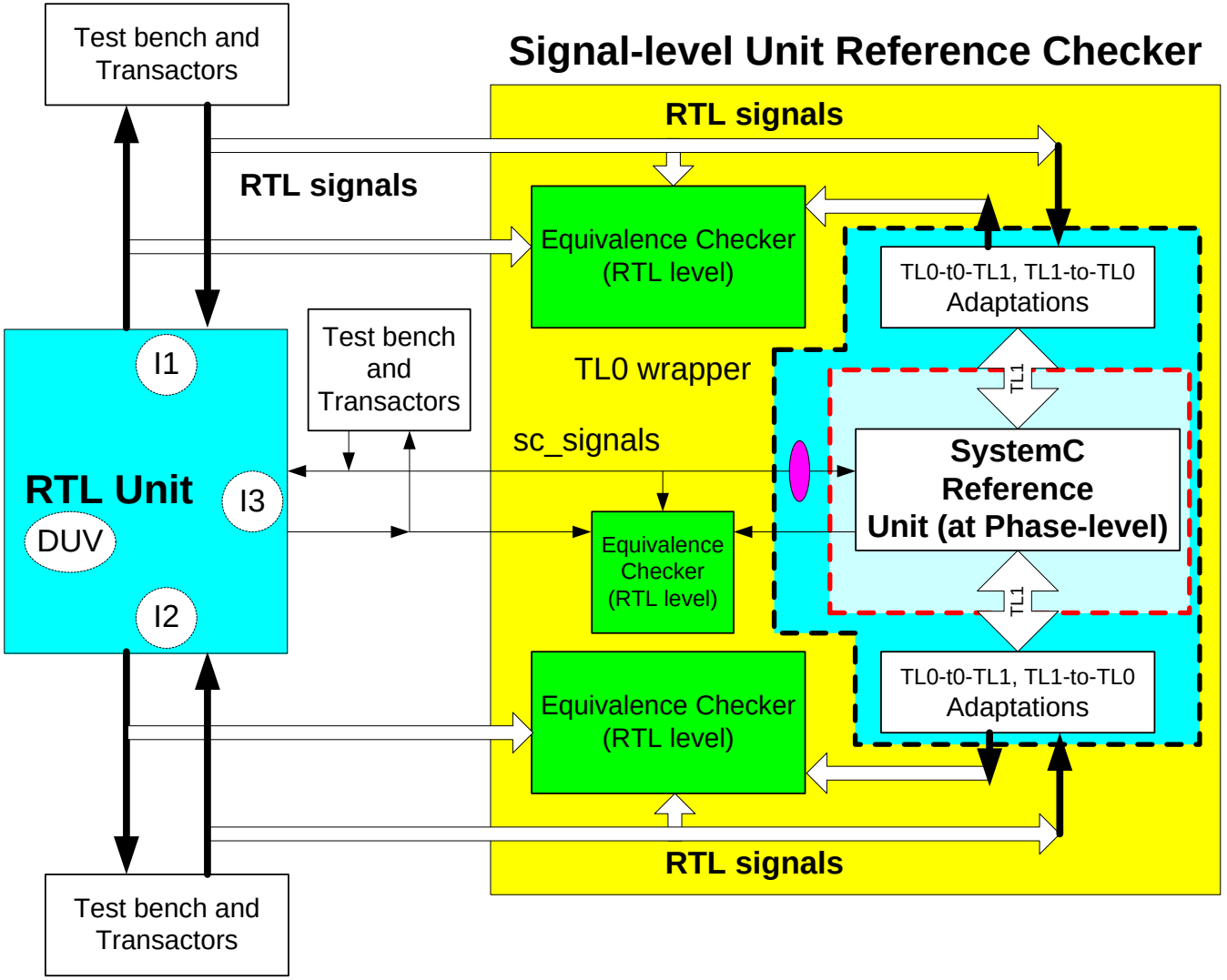
the DUT

the Checker

green: systemc



SystemC-based Unit Reference Checker and Unit Test Bench Environment



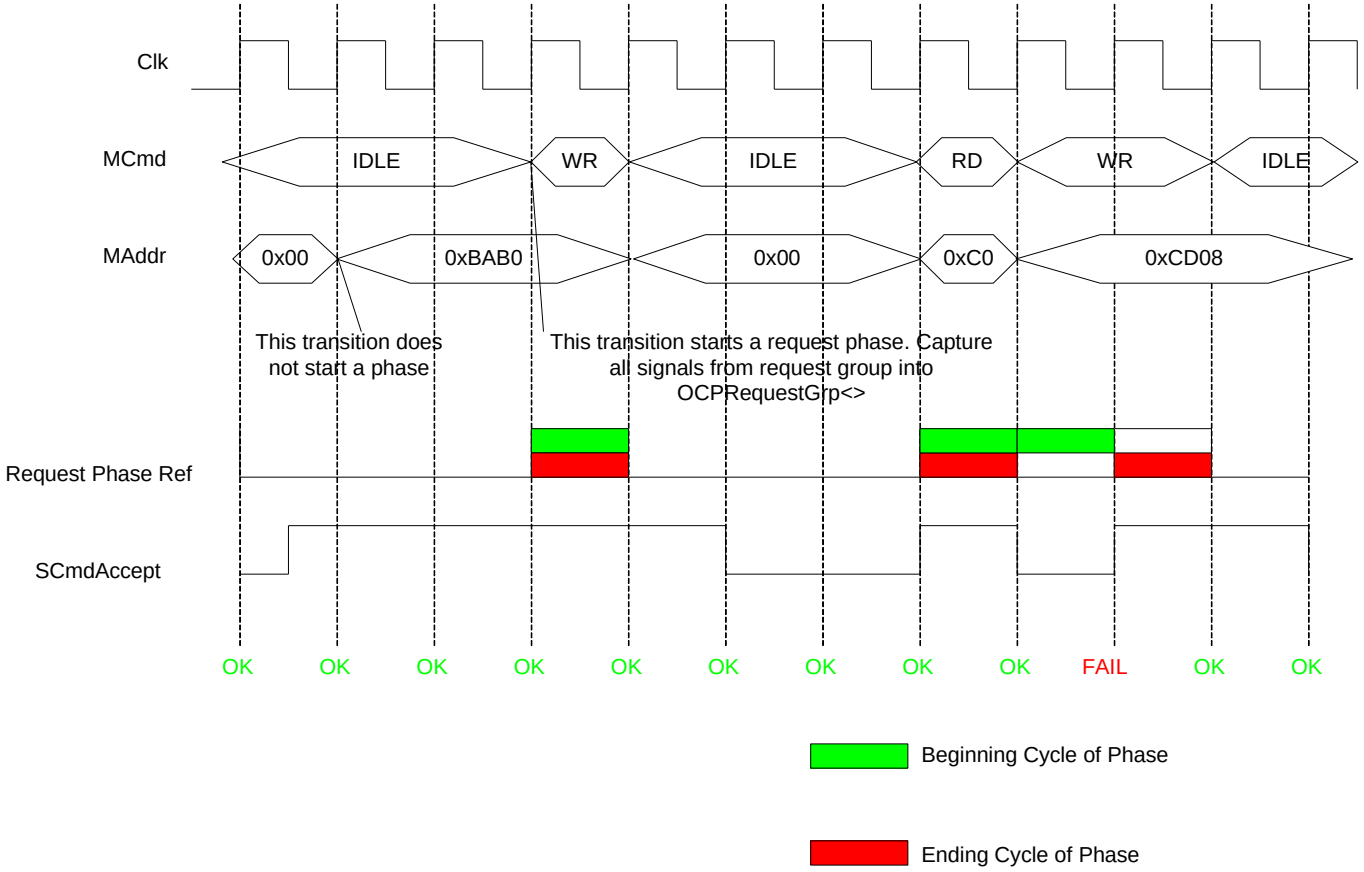
Co-simulation flow highlights

- EC checks all signals when relevant to protocol
  - *aware of each protocol phase*

- SystemC runs on ungated clock
  - *allows to check RTL fine-grained clock gating*
  - *coarse-grained clock request signal is checked at each cycle by an EC*

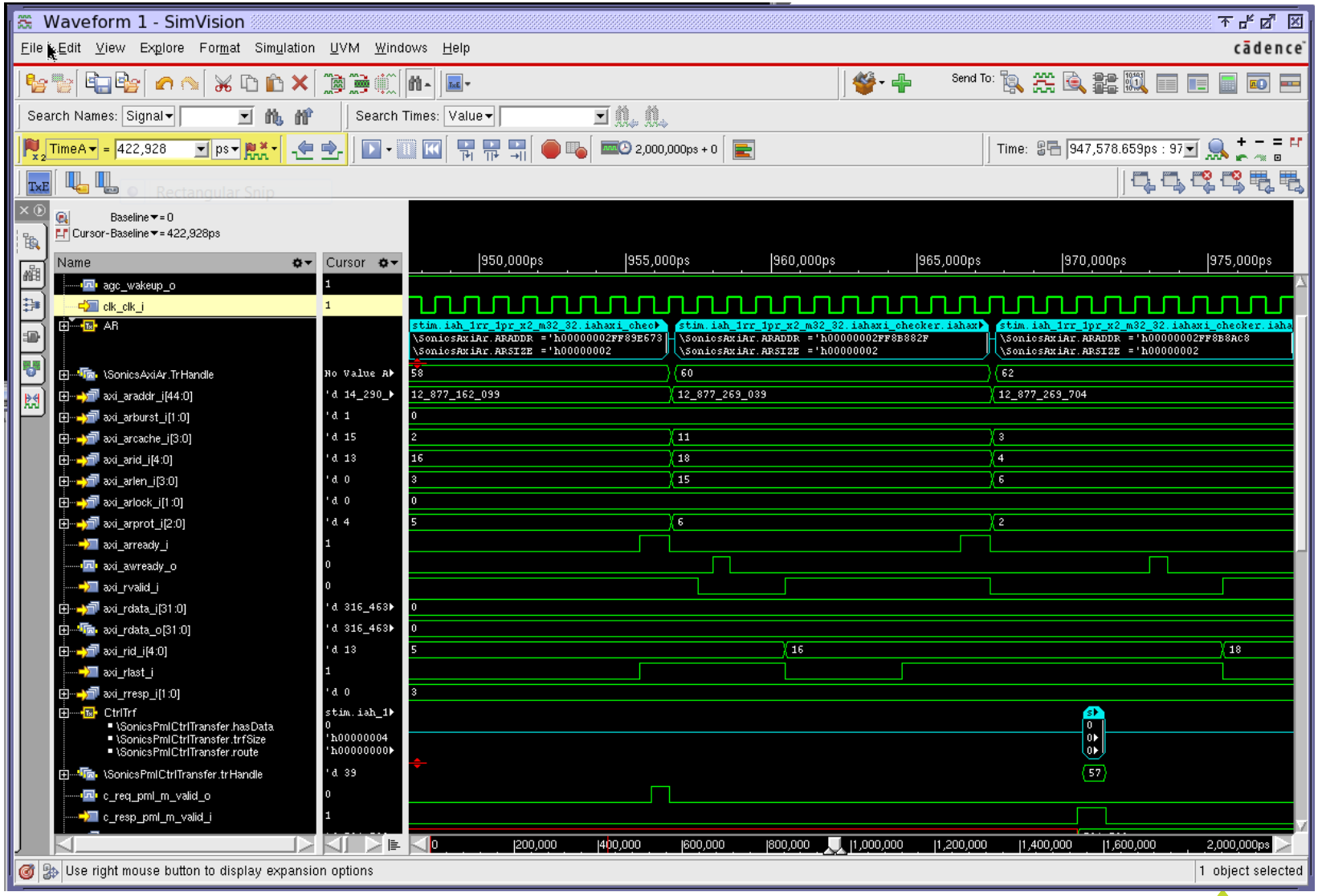
- Each unit (component) of the IP is verified against its model
  - *fully cycle accurate across range of configuration options*
  - *units are assembled structurally into a SystemC model of the full IP*
  - *NoC model cycle accurate by construction*

Design – phase detection



- Protocol knowledge
  - *Understand what starts/ends a phase*
    - MCmd
    - SCmdAccept
    - SResp
    - MRespAccept

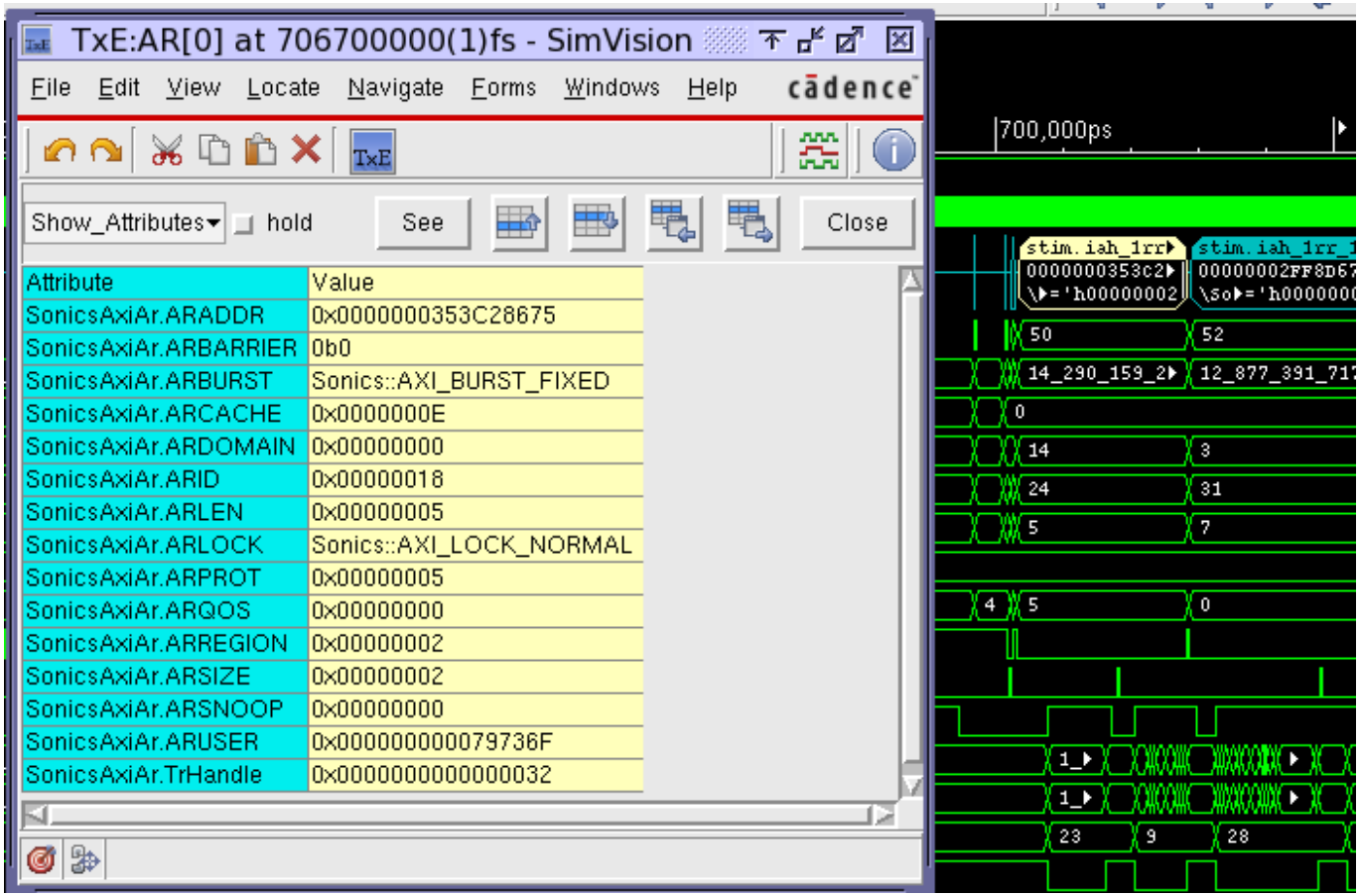
Mixed Waveform: signals + transactions



Transaction Debug

- Transactions visible alongside verilog
- Transaction ID stored on each record
  - *break out attributes on the waveform*
- Greatly simplifies debug
  - *especially when IP/protocols support out of order responses*
  - *can record parent/child relationship*
    - visible with Transaction Explorer

Transaction Explorer



Incisive specific setup

- "global" functions can be inserted by inheriting *ncsc_global_functions*
  - *set time resolution*
    - *Important to have systemc resolution matching the simulator*
  - *set up tracing*
    - *dlopen libtbsc.so*
    - Call *tb_tr_sdi_init()*

```
#ifdef NC_SYSTEMC
namespace Sonics {

// this is a hook provided by Cadence to do global set up in our co-sim library
// I don't know of a similar hook in MTI
class NcscGlobalDefs : public ncsc_global_functions
{
public:
    void startup() {

        // by default the systemc resolution is 1 ps. The following retrieves the
        // time precision (timescale X/X -> 2nd argument) from the VPI and uses it
        // to force the systemc resolution to match the verilog co-simulation
        if ( ncsc_in_simulator() ) {
            int resExponent = vpi_get( vpiTimePrecision , 0 );
            double resolution = pow( 10, resExponent );
            sc_set_time_resolution( resolution, SC_SEC );
        }

        // the following interprets custom arguments passed to us through
        // the -systemc_args option of irun
        char** argv = const_cast<char**>( sc_argv() );
        for ( int i=0; i < sc_argc(); ++i ) {
            // enable performance analysys, using the SDI entry point for
            // Cadence
            if ( strcmp( argv[i], "--perfmon" ) == 0 ) {
                const std::string entryPoint = "tb_tr_sdi_init";
                const std::string entryPointLib = "libtbsc.so";
                const std::string libPath = entryPointLib;
                SonicsPI2::getSonicsPIPlatform()->setEntrypoint(entryPoint);
                SonicsPI2::getSonicsPIPlatform()->setEntrypointLibrary(entryPointLib);
                SonicsPI2::getSonicsPIPlatform()->setLibraryName(libPath);
                SonicsPI2::setFormat("custom");
                SonicsPI2::enableSonicsPI();
            }
        }
    }
};
}

Sonics::NcscGlobalDefs g_SonicsNcscGlobalDefs;
#endif
```

Port Connection

- Binding based on port name
  - *matched during ncelab*
  - *need to bind ports against pre-compiled model*

- 2 approaches
  - *Generate ports dynamically in wrapper*
    - map ports by name with a base proxy class
    - let pre-compiled model interact with port through proxy interface

  - *Use ports with length context at construction time*
    - data type for vectors: sc_uint_base for vectors <= 32 bits, sc_bv_base for larger
    - sc_length_context determines width of the next signal

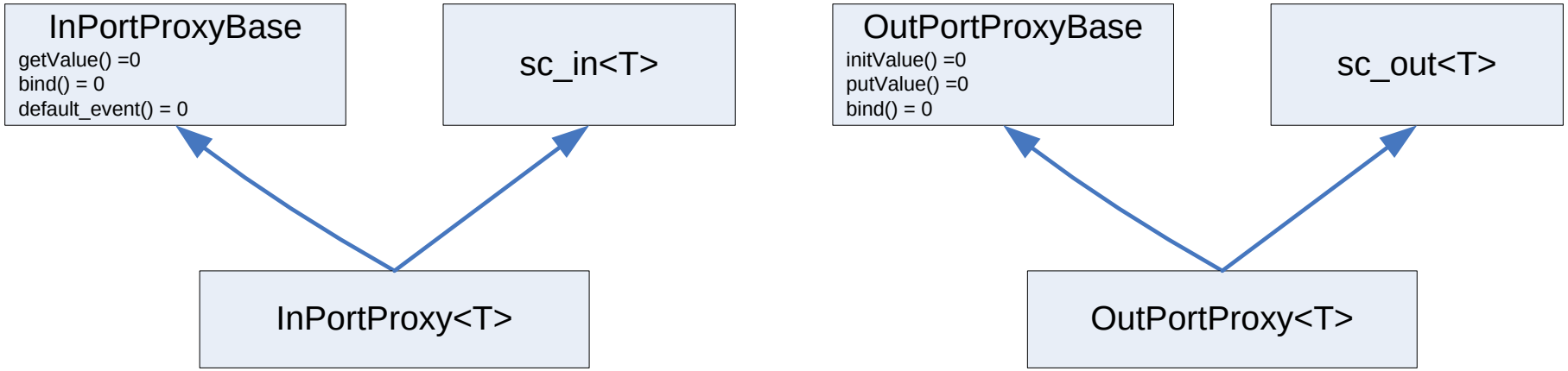
```
sc_length_context context( sc_length_param( 40 ) );
sc_in<sc_bv_base> port;  // 40 bit input port
```


SONICS

Design – TL0 ports

Use a port proxy

Generic model interacts with API of proxy base class



Generated SystemC Modules

iahaxi_tl0_iahaxitl0.v

```
//
// Copyright (c) 1999-2015 Sonics, Inc.(R).  All
// contains confidential and trade secret materia
// made, use, reproduce, display or perform (publ
// derivative works based on, offer for sale, sel
// disclose, license, sublicense, dispose of and
// solely in accordance with your license agreeme
// you have not agreed to all of the terms and co
// Agreement, you should immediately return this
// to Sonics, Inc..  This RTL or portions thereof
// and foreign patent and patent applications.
//
//
// Release 2.3.0
//
`ifdef INCA
module iahaxi_tl0_iahaxitl0(clk_clk_i, sys_reset_
                agc_wakeup_o, axi_awvalid_i, axi_
                axi_awready_o, axi_awid_i, axi_aw
                axi_awsize_i, axi_awburst_i, axi_
                // some ports deleted for display
                d_reg_resp_pml_m_valid_o, d_reg_r
    (* integer foreign = "SystemC"; *);

    input             clk_clk_i;
    input             sys_reset_ni;
    output            agc_wakeup_o;
    input             axi_awvalid_i;
    input   [44:0]    axi_awaddr_i;
    input   [2:0]     axi_awprot_i;
    output            axi_awready_o;
    input   [4:0]     axi_awid_i;
    input   [1:0]     axi_awregion_i;
    input   [3:0]     axi_awlen_i;
    input   [2:0]     axi_awsize_i;
    input   [1:0]     axi_awburst_i;
    input   [1:0]     axi_awlock_i;
    input   [3:0]     axi_awcache_i;
    output            d_reg_resp_pml_m_valid_o;
    output  [10:0]    d_reg_resp_pml_p_payload_o;
endmodule
`endif
```

u--**-XEmacs: iahaxi_tl0_iahaxitl0.v (Verilo

iahaxi_tl0_iahaxitl0.cc

```
#include "iahaxi_tl0_iahaxitl0.h"
#include "IChipDM.h"
#include "ScTl0Ports.h"

extern sc_module* sonics_make_instance_iahaxi_tl0_iahaxitl0( const std::string& );

iahaxi_tl0_iahaxitl0::iahaxi_tl0_iahaxitl0( sc_module_name name ) :
    sc_module( name )
{
    // create ports and pre-set bindings for iahaxi_tl0_iahaxitl0_tl0
    std::map< std::string, Sonics:: InPortProxyBase* >& inPortBundleMap = Sonics::BundleSignalFactory
        < Sonics:: InPortProxyBase >::s_externalMap;
    std::map< std::string, Sonics::OutPortProxyBase* >& outPortBundleMap = Sonics::BundleSignalFactory
        < Sonics::OutPortProxyBase >::s_externalMap;
    {
        Sonics::InPortProxy< bool >* p_sys_reset_ni = new Sonics::InPortProxy< bool >( "sys_reset_ni" );
        inPortBundleMap["sys_reset_ni"] = p_sys_reset_ni;
        sys_reset_ni = p_sys_reset_ni;
    }
    {
        Sonics::InPortProxy< bool >* p_clk_clk_i = new Sonics::InPortProxy< bool >( "clk_clk_i" );
        inPortBundleMap["clk_clk_i"] = p_clk_clk_i;
        clk_clk_i = p_clk_clk_i;
    }
    {
        Sonics::InPortProxy< bool >* p_axi_awvalid_i = new Sonics::InPortProxy< bool >( "axi_awvalid_i" );
        inPortBundleMap["axi_awvalid_i"] = p_axi_awvalid_i;
        axi_awvalid_i = p_axi_awvalid_i;
    }
    {
        Sonics::InPortProxy< sc_uint< 45 > >* p_axi_awaddr_i = new Sonics::InPortProxy< sc_uint< 45 > >( "axi_awaddr_i" );
        inPortBundleMap["axi_awaddr_i"] = p_axi_awaddr_i;
        axi_awaddr_i = p_axi_awaddr_i;
    }

    // signals deleted for brevity
    //
    iahaxi_tl0_iahaxitl0_tl0.reset( sonics_make_instance_iahaxi_tl0_iahaxitl0( basename() ) );
    inPortBundleMap.clear();
    outPortBundleMap.clear();
    end_module();
}


#ifdef NCSC
NCSC_MODULE_EXPORT(iahaxi_tl0_iahaxitl0);
#endif
#ifdef MTI_SYSTEMC
SC_MODULE_EXPORT(iahaxi_tl0_iahaxitl0);
#endif
```

u--**-XEmacs: iahaxi_tl0_iahaxitl0.cc (C++ Font Abbrev)----L14--C8--All----------

Conclusion

- SystemC Model usable in functional verification
  - *Amortize investment in the model*
  - *Facilitates debug*
- Use incisive features
  - *Elaboration time port binding*
  - *SDI transaction recording*
- Caveat
  - *SystemC distribution is not binary compatible with OSCI reference simulator*
  - *Forces to compile all libraries specifically for Cadence*