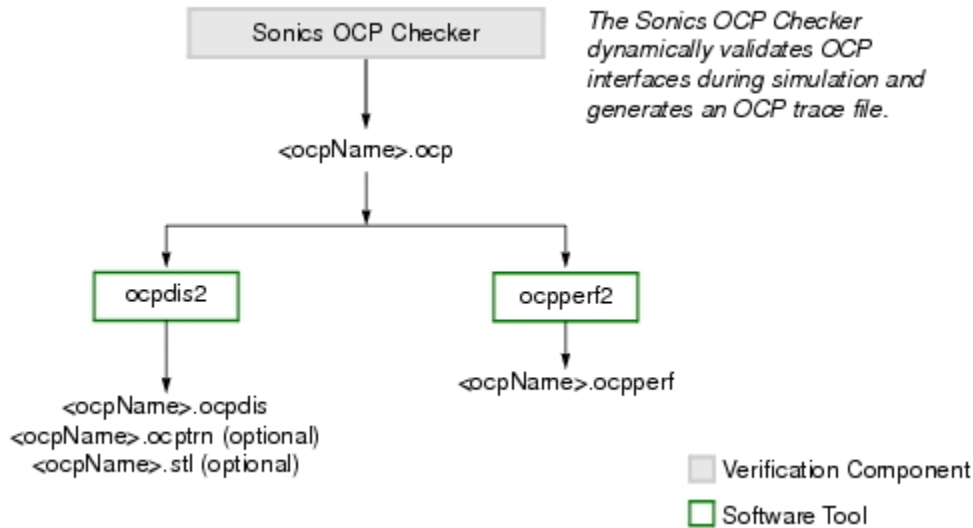


SOLV

OCP-IP released SOLV (Sonic's OCP Library for Verification) in 2008 November. SOLV consists of three Verification components – OCP Protocol Checker, OCPdis2 and OCPperf2. The tool flow for the SOLV package is shown in Figure 1.

Figure1 : SOLV tool flow



Open Core Protocol

The Open Core Protocol (OCP) is a public standard memory-mapped core interface. It is defined by collaboration between system houses, electronic-design-automation (EDA) vendors, and intellectual-property (IP) suppliers within the OCP International Partnership (www.ocpip.org). The motivations for defining the OCP are to enable better IP reuse and a wide spectrum of common tooling and experience.

OCP is a rich standardized interface and can support a wide range of cores. The interface is highly configurable and can adapt to the needs of various cores. The diverse nature of the interface can easily allow for the various combinations of cores used within the industry today.

The OCP Functional Verification Working Group (FVWG) has come up with a set of protocol rules for protocol compliance and the protocol rules have been documented in Chapter 18 of the OCP 3.0 specification. OCP Protocol Checker has been implemented to cover all the rules described in Chapter 18 of the OCP 3.0 specification. OCP Protocol Checker has been regressed across all Sonics IP's that support an OCP interface and has been used internally by Sonics for over 5 years. The checker has also been shipped to all Sonics customers with the verification testbench provided as part of Sonics IP deliverables. Hence the protocol checker has attained a

great level of maturity over the last 5 years and has become an integral verification component for Sonics and many of our customers.

Assertion Based Verification and SVA

Assertion Based Verification (ABV) Methodology has been adopted and used by the verification community extensively in the last decade. For SoC interfaces, ABV is a natural choice. With a defined protocol rule set, ABV can facilitate identifying bugs due to illegal behavior in the interface. System Verilog Assertion (SVA) language is a very robust language for ABV. SVA is widely accepted in the industry and the tool support for debugging assertions is very robust and mature. OCP Protocol Checker was developed using ABV Methodology and SVA language.

OCP Protocol Checker

A significant amount of debug time is spent on the interfaces that are developed in-house for System On Chip (SoC) designs. With increasing complexity of the SoC's and derivative SoC designs it is imperative to use a standard interface and also use debug capabilities provided by the interface. OCP Protocol checker from Sonics can be easily plugged into multiple OCP interfaces used in the SoC testbench. The SOLV manual lists the steps required to connect the OCP protocol checker to the OCP interfaces in a given SoC design. Figure 2 is a simple OCP Protocol checker instantiation.

Figure2 : OCP Protocol Checker

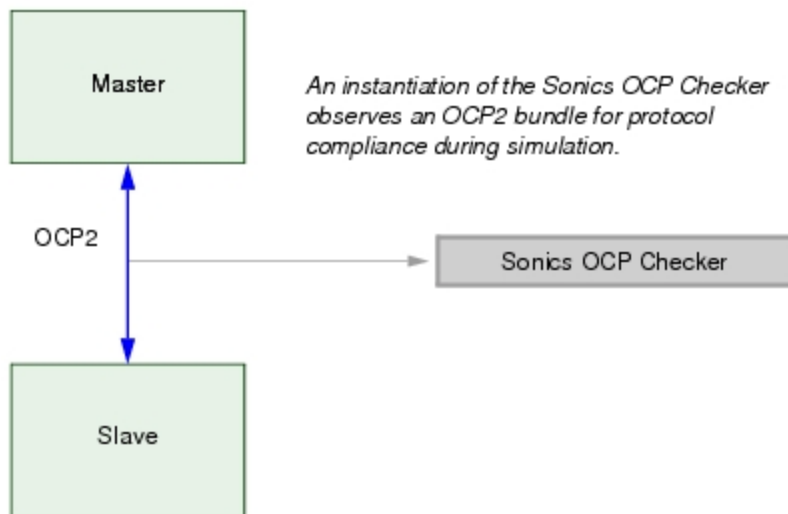


Figure 3 is an example instantiation of the Sonics OCP checker to the interface. Each checker instance will have instance parameters and protocol parameters. Example instance parameters are (a) number of cycles for the interface to be idle and (b) maximum number of outstanding requests in the interface. The SOLV documentation can be referred to for additional parameters that can be controlled by the user.

Figure 3: Instance and Protocol Parameter Map Example

```

ocp2_sva_checker #(
    //Instance Parameters
    .version ("<ocp-version-and-revision>"),
    .checkername ("<checker-name>"),
    .ocpcheck_enable (<0|1>),
    .name ("<checker-port>"),
    ...
    //Protocol Parameters.
    .broadcast_enable (1),
    .burst_aligned (1),
    ...
    .sdatainfo (0),
    .sdatathreadbusy (0),
    ...
}

```

The “Protocol Parameters” section needs to match the OCP configuration information of the interface. Parameters that are not used in the interface need to have default values. The next step is to instantiate the checker port for each interface in the testbench. An example of the OCP checker port map is available in figure 4.

Figure 4: *Example Port Map*

```

checker_port (
    .Clk_i (Clk),
    .MCmd_i (Mcmd),
    .MAddr_i (MAddr),
    ...
    ...
    .MBurstSeq_i (1'b0),
    .MBurstSingleReq_i (1'b0),
    ...
    ...
)

```

It is important to note that signals that are not used in the interface need to be tied off low. An example is MBurstSeq_i and MBurstSingleReq_i in Figure 4. The checker is supported with Incisive NCVERILOG, VCS and Modelsim simulators. The user needs to perform the above steps and OCP Protocol checker reports any mismatches in the protocol during the simulation run. An example protocol error mismatch is shown in Figure 5.

Figure 5: *Example error message from the OCP protocol checker.*

```

# ** Error: PROTOCOL: D096: SRespLast was asserted for a non-last response phase
# Time: 274434 ns Started: 274434 ns Scope: stim.McBpR_256DS.ocpmon.ocpmon2_sva.threaded_checks
[0].genblk11.ap_D096

```

The error message prints out a message explaining the error, time of the error during the simulation run and the hierarchical path of the OCP instantiation the error was reported from. The information provided in the error message is very useful for the engineer who is debugging the SoC allowing him to isolate the interface and hence provide a starting point to debug the issue.

Ocpdis2 and OcpPerf2

Along with the OCP protocol checker, the SOLV package also provides two programs that help with debug and performance: Ocpdis2 and OcpPerf2 (Figure 1). OCP Protocol Checker is capable of generating traces for each interface and the user has the capability to enable the generation of traces during setup time. The Ocpdis2 program could be run on the traces and the generated output can then be used to debug the transactions. An example disassembled file is shown in Figure 6.

Figure 6: *Sample Disassembled Trace File in Transfer Output Style*

```
##
## Generated by ocpdis2 1.7
##
# Legend for 'transfer' output style (sorted by request cycle)
# -----
##
# SimTime : Simulation Time | Time
# Cycle : Cycle Valid Time | Cycle
# Cmd : Master Command | MCmd
# BuS : Master Burst Sequence | MBurstSeq
# Burst : the running transfer count / Master Burst Length x Master Block Height / Master Block Stride | transfer count/
MBurstLength x MBlockHeight/MBlockStride
# L : Master Last Request | MReqLast
# RL : Master Row Last Request | MReqRowLast
# AS : Master Address Space | MAddrSpace
# Addr : Master Address | MAddr
# Data : Data | Data
# Resp : Slave Response | SResp
# SL : Slave Last Response | SRespLast
# SRL : Slave Row Last Response | SRespRowLast
SimTime Cycle Cmd BuS Burst L RL AS Addr Data Resp SL SRL
-----
790 79 WR BLCK 1/2x3/8 0 0 1 29929dcd4fbfe8 ad4fbfe8
800 80 WR BLCK 2/2x3/8 0 1 1 29929dcd4fbfec ad4fbfec
810 81 WR BLCK 3/2x3/8 0 0 1 29929dcd4fbff0 ad4fbff0
820 82 WR BLCK 4/2x3/8 0 1 1 29929dcd4fbff4 ad4fbff4
830 83 WR BLCK 5/2x3/8 0 0 1 29929dcd4fbff8 ad4fbff8
840 84 WR BLCK 6/2x3/8 1 1 1 29929dcd4fbffc ad4fbffc
850 85 RD BLCK 1/1x3/8 0 1 1 29929dcd4fbfe8 ad4fbfe8 DVA 0 1
860 86 RD BLCK 2/1x3/8 0 1 1 29929dcd4fbff0 ad4fbff0 DVA 0 1
860 86 RD BLCK 2/1x3/8 1 1 1 29929dcd4fbff8 ad4fbff8 DVA 1 1
```

The disassembled output is useful for an engineer debugging the SoC allowing him to identify the traffic pattern that resulted in the mismatch. The disassembler can produce the generated file in wire style, transfer style, transfer-data style and STL style. OcpPerf2 is used to measure Bandwidth, Latency, etc. SOLV documentation can be referred to for the available options which allow for measuring the performance on an OCP interface. A textual file similar to Figure 6 would be generated with the performance results for an OCP interface.

The article explained the steps involved in utilizing the SOLV software package to: instantiate the OCP Protocol Checker, debug OCP Interfaces, visualize traffic patterns and generate performance numbers. OCP Protocol Checker uses Assertion based verification methodology and also is supported with the three industry leading simulators. The package is very useful for SoC debug and compliments existing debug techniques in already available in the industry.



Ravi Venugopalan is the Verification Manager at Sonics Inc. A specialist in Functional Verification, Ravi has 10 years of experience in the field as an engineer and manager. While at Sonics, Ravi has been responsible for hiring, training and managing the verification group as well as managing the verification responsibilities across all Sonics' products. He is also involved with the management of Sonics offshore design center. Prior to joining Sonics, Ravi worked for IBM as a Verification Engineer