Implementation of a High-Level Hardware Verification System using Truss

Myoung-Keun You, Joo-Hong Kim, Gi-Yong Song

School of Electrical and Computer Engineering
Chungbuk National University, Cheongju Chungbuk, 361-763, Korea
E-mail : mkyou77@chungbuk.ac.kr, goodwolf@white.chungbuk.ac.kr, gysong@chungbuk.ac.kr

Abstract : The implementation of a high-level hardware verification system using Truss is presented in this paper. Teal is a C++ class library for functional verification and enables functional verification by providing connections to HDL signals and allowing actions based on changes in the HDL simulation. Truss is an implementation of an open-source verification infrastructure based on layer approach through object-oriented programming techniques. The functional verification is performed on a simple device-under-test, the transposed FIR filter. The FIR filter which processes convolution sum is a typical operation being involved in various applications regarding DSP.

1. Introduction

The silicon revolution is in a constant state of evolution, and today the verification of a newly introduced hardware draws a lot of attention. Verification of a hardware module along with implementation is of a paramount importance in the design phase. A major development in the field of functional verification is the increasingly mainstream use of OOP(object-oriented programming) techniques. OOP is designed to manage complexity. Because of the flexibility inherent in using OOP, code is simpler to use and more adaptable [1-3].

HDLs do a good job of describing hardware concepts, but fall short when it comes to testing hardware. HDLs are limited in their ability to manage randomization. On the contrary, C++ supports the concept of interface versus implementation and operator overloading, templating, namespace, multiple inheritance, etc. So using C++ is acceptable to implement hardware verification environment.

In this paper, we implement a hardware verification system using Teal and Truss. Teal is C++ class library for functional verification, and provides C++-to-HDL interface. Truss provides the infrastructure layers above Teal and defines the sequencing of the various components of the system [1-2]. The functional verification is performed on the transposed FIR filter.

The rest of this paper is constructed as follows. Section 2 introduces Teal, Truss, and transposed FIR filter. Section 3 explains high-level hardware verification environment, and section 4 discusses simulation and validation. Finally section 5 is conclusion.

2. Preliminary

2. 1 Teal

C++ is not what most hardware engineers use for their HDL, so we need an interface layer to connect the HDL with C++. Being a C++ class library for functional verification, Teal [1-2] is such an interface. Teal provides the illusion that the verification system is in control of the DUT. In Teal, you write a verification_top() function, and create tests, generators, checkers, drivers, and monitors. Each of these objects appears to be running independently of the DUT, with each its own thread of execution. In reality, these threads execute in response to a wire or register change. Teal enables functional verification by providing connections to HDL signals and allowing actions based on changes in the HDL simulation. It encourages the development of independent generators, checkers, drivers, and monitors by providing management for user-created threads that execute concurrently with the HDL simulation.

The most important classes and namespaces of Teal are as follow;

- *reg class* which provides arbitrary-length, four-state(0, 1, X, Z) registers with corresponding operations
- *vreg class* which connects C++ code to HDL
- *memory namespace* which provides an abstract interface for reading and writing memory
- *vrandom class* which is a stable random-number generator
- *run_thread() function* which forks off a thread
- *at() function* which allows a thread to pause until any of the HDL signals has changed

2.2 Truss

Truss [1-2] is an implementation of an open-source verification infrastructure based on layered approach. The layered approach breaks each interface into three sublayers as shown in Fig. 1.

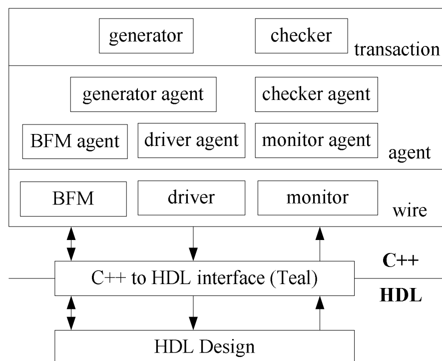

Fig. 1. Interface hierarchy structure

Truss uses the Teal library as a connection between C++ and the simulation. Teal provides the fundamental elements of a verification system. Truss, on the other hand, provides the infrastructure layers above Teal, adding a set of classes, templates, and conventions to facilitate the construction of an adaptable verification system. The top-level verification components are shown below in Fig. 2.

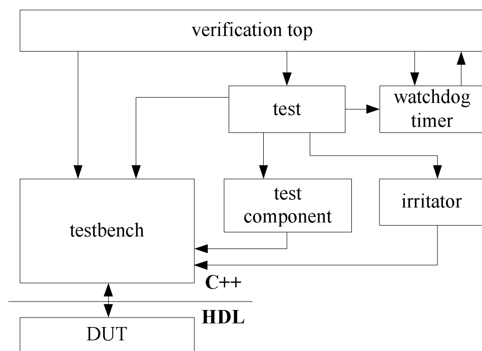

Fig. 2. Verification component hierarchy

The top-most C++ component is the verification_top() function, whose role is to create and sequence the other components through a standard test algorithm. The watchdog timer is a component created by verification_top(). This component's role is to shut down a simulation after a certain amount of time has elapsed, to make sure the simulation does not run forever. The testbench top-level component is the bridge between the C++ verification and the HDL DUT. As such, the testbench's role is to isolate the tests from having to know how C++ transactors, traffic generators and monitors interact with the DUT. The test executes a specific functionality of the DUT by using the testbench-created BFMs, monitors, and generators. The test component is an abstract base class whose role is to exercise some interface of the DUT. The test component describes the interface that all implementations should follow. The irritator is a background-traffic test component.

2.3 Transposed FIR Filter

Digital filters [4] are typically used to modify or alter the attributes of a signal in the time or frequency domain. An FIR with constant coefficients is a linear time-invariant (LTI) digital filter. The output sequence, $y(n)$, of an FIR of order or length L, to an input sequence, $x(n)$, is given by the finite version of convolution sum as follows;

$$y[n] = x[n] * f[n] = \sum_{k=0}^{L-1} x[k] f[n-k]$$

where $f[0] \neq 0$ through $f[L-1] \neq 0$ are the filter's L coefficients.

The FIR filter which processes convolution sum is a typical operation being involved in various applications regarding DSP. When implementing FIR filter, the transposed structure is preferred as the FIR filter with transposed structure does not need either an extra shift register for input sequence or an extra pipeline stage for adder of the products. The block diagram of the FIR filter in transposed structure is shown in Fig. 3.

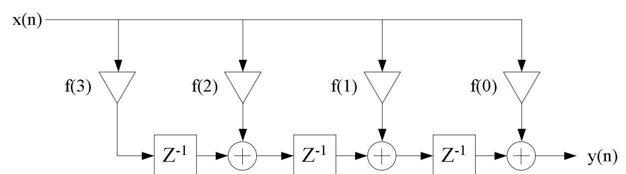

Fig. 3. Transposed FIR filter

3. Implementation of a High-Level Hardware Verification Environment using Teal and Truss

The functional verification on the DUT, the FIR filter, is performed by applying the DUT to Truss verification framework. The objects and connections of Truss based on the hierarchical structure of interface layer are shown in Fig. 4.

In Fig. 4, trans_fir::trans_fir_test class holds a *has-a* relationship with trans_fir::testbench and trans_fir::test_component class, having corresponding pointers to each class. Also, trans_fir::testbench class holds a *has-a* relationship with trans_fir::generator, trans_fir::driver, trans_fir::checker and trans_fir::monitor class, having corresponding pointers to each class. All of trans_fir::generator, trans_fir::driver, trans_fir::checker and trans_fir::monitor class are abstract base classes with each class having pure virtual functions, and the agents of these classes define the interfaces inheriting corresponding abstract base classes.

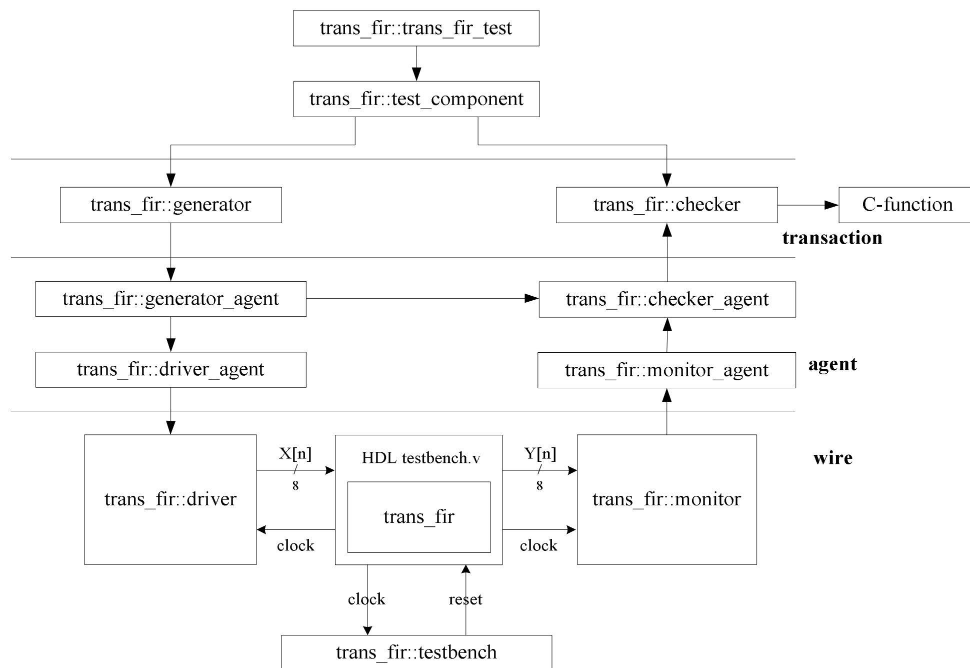

Fig. 4. The objects and connections of Truss

The operations performed on each layer are explained as follows;

▪ The wire layer

The wire layer classes have simple procedural interface. Their role is to take method and procedure calls and execute the protocol. This layer is responsible for the mapping between a method call and wire-change sequence. The interlock between HDL simulator and C++ code is implemented through Teal's vreg object. The vreg class makes C++ code accessible to wires and registers of DUT through VPI(Verilog procedural interface) [5-6]. The trans_fir::driver, trans_fir::monitor, and trans_fir:: testbench objects have corresponding vreg objects in order to access testbench.v of HDL. Trans_fir::testbench initializes verification process by sending reset signal to testbench.v. Trans_fir::driver sends test_vector which is generated in trans_fir::generator to DUT. Trans_fir::monitor reads output from DUT, and sends them again to trans_fir::checker.

▪ The agent layer

The agent layer acts as a go-between for two well-defined components. The classes in this layer add some sort of queue, for data or control actions, depending on what the transaction layer generate or check. Inheriting abstract base class, each agent defines corresponding interface, and Truss channel connects all agents. Test_vector is transferred from trans_fir::generator_agent to trans_fir::driver_agent and trans_fir::checker. The outputs from DUT are transferred from trans_fir::monitor_agent to trans_fir:: checker_agent.

▪ The transaction layer

The transaction layer uses other layer to exercise interface or feature of DUT and validate the response. The driving part is generator and response validating part is checker. Trans_fir::generator generates test_vector which will be sent to HDL simulator using random() function, and transfers the test_vector to trans_fir::driver_agent through trans_fir::generator_agent. Trans_fir::checker receives both test_vector and output from DUT through two channels connected to trans_fir::checker_agent. After comparing the output from HDL simulator to the output from C-simulator of the DUT, the verification result on the operation of DUT is reported to log file.

4. Simulation and Validation

Steps of implementation of hardware verification system using Teal, Truss, and HDL simulator are roughly three as follow;

- Step 1: design the DUT in Verilog-HDL and simulate it using simple test vector.
- Step 2: code all components of verification system provided by Truss infrastructure.
- Step 3: verify the functional behavior of the DUT using Teal and Truss library.

Creating all objects of verification system's components and connecting each object through channels are performed in step 2. An implemented hardware verification system using Truss infrastructure has two big benefits. One is verification of the DUT with other test vector can be easily achieved as only changing constraints of test vector in generator module. The other is the verification system can be easily adjusted to other verification project because of the characteristic of OOP.

To simplify the result observation, we set coefficients of 4-stage transposed FIR filter as 0x1, 0x2, 0x3, and 0x4, and specify the range constraint of test vector between 0x0 and 0x5. The log file which reports the result of a functional verification is shown in Fig. 5 when trans_fir::generator generated test vector as 0x04, 0x0, 0x1, and 0x4. Messages of log file are reported by trans_fir::checker after comparing output from HDL simulator to the output from C-simulator of the DUT.

5. Conclusions

This paper describes an implementation of a high-level hardware verification system using Truss. Hardware modules are designed with Verilog HDL in order to make C++ code accessible to wires and registers of DUT through VPI. Also, based on inheritance, we build verification components provided by Truss infrastructure for functional verification. A log file is reported on the result of functional verification.

Hardware verification system implemented in this paper can be reused in another verification project due to the characteristic of OOP.

References

[1] Mike Mintz, Robert Ekendahl, Hardware Verification with C++ : A Practitioner's Handbook, Springer, 2006.

[2] The Teal User's Manual, http://www.trusster.com

[3] Andreas Meyer, Principles of Functional Verification, Newnes, 2003.

[4] U.Meyer-Baese, Digital Signal Processing with Field Programmable Gate Arrays, Springer, 2001.

[5] Swapnajit Mitra, Principles of Verilog PLI, Kluwer Academic Publishers, 1999

[6] Stuart Sutherland, The Verilog PLI Handbook : A Tutorial and Reference Manual on the Verilog Programming Language Interface, Kluwer Academic Publishers, 2002.

Red Hat_9 - VMware Workstation ACE Edition

File Edit View VM Team ACE Windows Help

Home | Red Hat_9

```
[190 ns][watchdog][watchdog][FILE: ./truss_watchdog.cpp][line: 65][INFO]  Using Timeout of 10000000
[190 ns][test_component][test_component][FILE: /usr/local/hardware_verification_with_cpp/truss/inc/truss_test_component.h][li
ne: 64][DEBUG]start_() for test_component  begin
[190 ns][test_component][test_component][FILE: ./test_component.cpp][line: 100][DEBUG]Starting components.
[190 ns][trans_fir_driver_0][test_component][FILE: ./driver_agent.cpp][line: 60][DEBUG] start
[190 ns][Teal::synch][FILE: ./teal_synch.cpp][line: 518][INFO]Thread trans_fir_driver_0 created. ID is 0xb2f7fbb0 result 0x0
[190 ns][trans_fir_driver_0][test_component][FILE: ./driver.cpp][line: 57][DEBUG] start
[190 ns][Teal::synch][FILE: ./teal_synch.cpp][line: 518][INFO]Thread trans_fir_checker_0 created. ID is 0xb257ebb0 result 0x0
[190 ns][test_component][test_component][FILE: ./test_component.cpp][line: 75][DEBUG] randomize
[190 ns][test_component][test_component][FILE: ./test_component.cpp][line: 109][DEBUG]trans_fir::test_component::generate num
ber of inputs is 4
[190 ns][test_component][test_component][FILE: /usr/local/hardware_verification_with_cpp/truss/inc/truss_test_component.h][li
ne: 68][DEBUG]start_() for test_component  end
[190 ns][Teal::synch][test_component][FILE: ./teal_synch.cpp][line: 274][INFO]thread_completed: Thread test_component
[200 ns][trans_fir_checker_0][trans_fir_checker_0][FILE: ./checker.cpp][line: 60][INFO] GENERATOR : send value  == 0x4
[200 ns][trans_fir_checker_0][trans_fir_checker_0][FILE: ./checker.cpp][line: 64][INFO] EXPECTED: value 0x4 == 0x4
[220 ns][trans_fir_checker_0][trans_fir_checker_0][FILE: ./checker.cpp][line: 60][INFO] GENERATOR : send value  == 0x0
[220 ns][trans_fir_checker_0][trans_fir_checker_0][FILE: ./checker.cpp][line: 64][INFO] EXPECTED: value 0x8 == 0x8
[240 ns][trans_fir_checker_0][trans_fir_checker_0][FILE: ./checker.cpp][line: 60][INFO] GENERATOR : send value  == 0x1
[240 ns][trans_fir_checker_0][trans_fir_checker_0][FILE: ./checker.cpp][line: 64][INFO] EXPECTED: value 0xd == 0xd
[260 ns][trans_fir_checker_0][trans_fir_checker_0][FILE: ./checker.cpp][line: 60][INFO] GENERATOR : send value  == 0x4
[260 ns][trans_fir_checker_0][trans_fir_checker_0][FILE: ./checker.cpp][line: 64][INFO] EXPECTED: value 0x16 == 0x16
[280 ns][trans_fir_checker_0][trans_fir_checker_0][FILE: ./checker.cpp][line: 60][INFO] GENERATOR : send value  == 0x0
[280 ns][trans_fir_checker_0][trans_fir_checker_0][FILE: ./checker.cpp][line: 64][INFO] EXPECTED: value 0xb == 0xb
[300 ns][trans_fir_checker_0][trans_fir_checker_0][FILE: ./checker.cpp][line: 60][INFO] GENERATOR : send value  == 0x0
[300 ns][trans_fir_checker_0][trans_fir_checker_0][FILE: ./checker.cpp][line: 64][INFO] EXPECTED: value 0x10 == 0x10
[320 ns][trans_fir_checker_0][trans_fir_checker_0][FILE: ./checker.cpp][line: 60][INFO] GENERATOR : send value  == 0x0
[320 ns][trans_fir_checker_0][trans_fir_checker_0][FILE: ./checker.cpp][line: 64][INFO] EXPECTED: value 0x10 == 0x10
[320 ns][verification_top:][verification_top][FILE: ./truss_verification_top.cpp][line: 171][DEBUG]About to report
[320 ns][testbench][verification_top][FILE: ./testbench.cpp][line: 121][DEBUG]Final Report:  report
[320 ns][test_component][verification_top][FILE: /usr/local/hardware_verification_with_cpp/truss/inc/truss_test_component.h][
line: 57][DEBUG]Final Report:  Completed as expected.
[320 ns][Shutdown][verification_top][FILE: ./truss_verification_top.cpp][line: 78][INFO]Test trans_fir_test Passed.
```

Fig. 5. Part of log file for functional verification