



Shunt- TCP/IP data exchange between stand-alone SystemVerilog simulations and external applications

Table of Contents

1. Introduction	1
2. Goals and objectives	2
3. Overview	2
3.1. Directory structure	2
3.2. The Shunt hierarchy	3
3.3. Primitive Layer (PRIM)	3
3.4. Client-server layer (CS)	4
3.5. Handshake API layer (HS)	5
4. Getting Started	6
4.1. Library Installation	6
4.2. Library compilation	6
4.3. Compile and run C examples	6
4.4. Compile and run SV examples	7
5. Library version scheme and roadmap	7
6. Contact information	8

1. Introduction

The Shunt is Open Source Client/Server TCP/IP socket based communication library for SystemVerilog simulation.

- It aims to enable quick and easy development of communication between stand-alone SystemVerilog simulations and/or external applications
- It offers a common SystemVerilog/C API and supports all SystemVerilog data types.

The Shunt is available under a MIT License. It can be used without restriction in an open-source or commercial application.

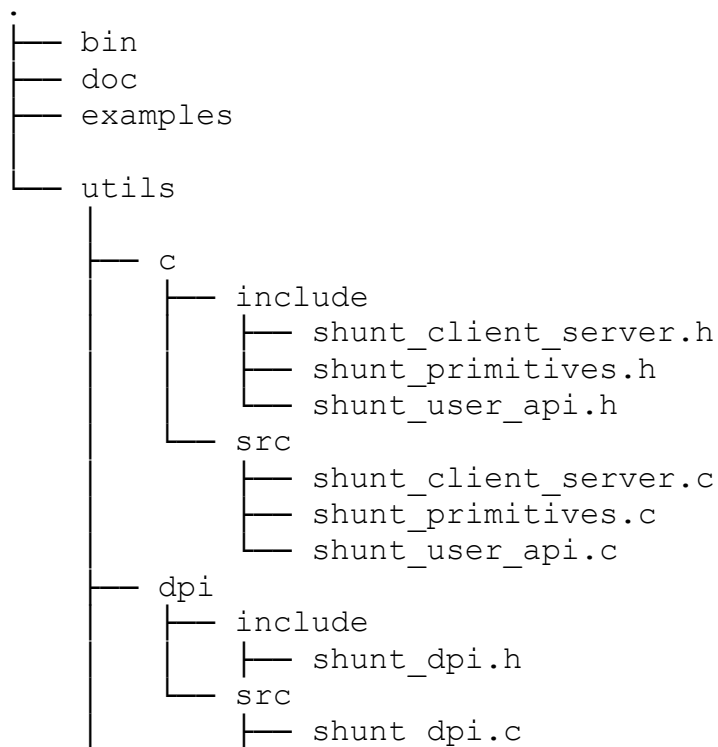
2. Goals and objectives

- **Portability.** The library supports all known SystemVerilog (SV) 2012 data types and data structures. It should provide consistent behavior for all major simulators.
- **Scalability and flexibility.** The library facilitates the development of communication between multiple stand-alone SV simulations and/or external applications.
- **Memory-Efficiency.** The library should not cause any memory leaks.
- **Simple API model.** The Shunt is aiming to minimize the up-front investment in time and effort by providing a simple, consistent API.
- **Extensibility.** The Shunt should enable a platform to build higher levels abstraction application and libraries.

3. Overview

3.1. Directory structure

The following diagram illustrates Shunt Directory structure:



```

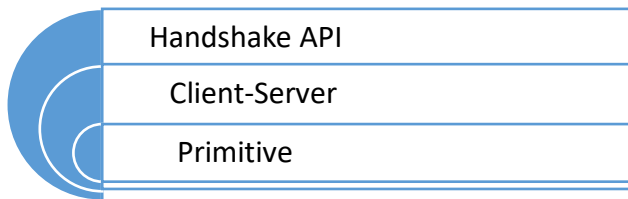
└─┬─ shunt_dpi_pkg.sv
   └─ shunt_hs_pkg.sv
└─ mkdir

```

Where is:

- The "utils" is a main Shunt library depository. It contains C functions (Reference to utils/c) and corresponding "SV" DPI wrappers (Reference to utils/dpi).
- The "doc" - Shunt Library documentation.
- The "bin" - C domain compiles results
- The "examples" - it contains "SV to SV" and "C to C" communication examples.

3.2. The Shunt hierarchy



The Shunt comprised of 3 layers of abstraction: primitive (PRIM), client-server (SC), and handshake API (HS). Each layer is self-sufficient. User can build an application on top of each layer.

3.3. Primitive Layer (PRIM)

The first layer contains basic TCP/IP socket initialization, some generic functions, and SV data exchange methods. Its main purpose is to enable a basic data transfer between initiator and target for all major SV data types. The PRIM operation doesn't require any memory allocation.

Initiator-Target PRIM exchange operation flow is shown below:

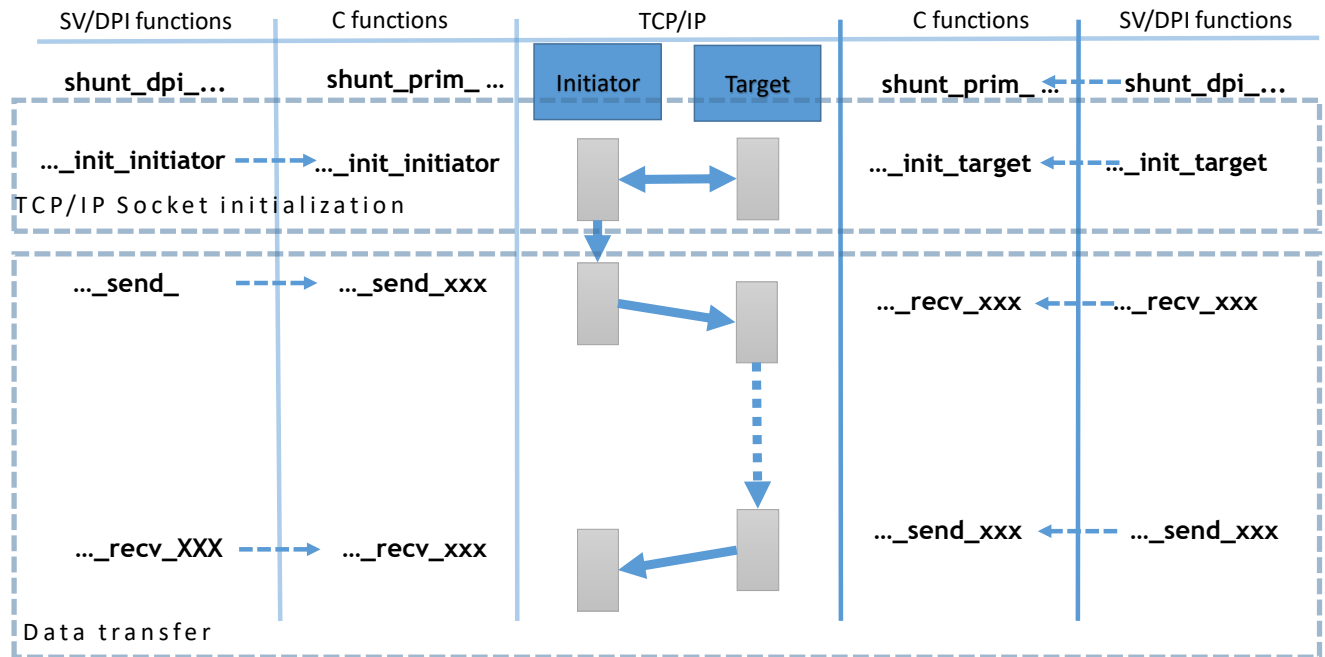


Figure 1 PRIM operation

"C to C" PRIM examples: [Shunt/examples/c/primitives](#)
 "SV to SV" PRIM examples: [Shunt/examples/sv/sv2c/initiator](#)
 (Ref. to xxx_loopback tasks)

3.4. Client-server layer (CS)

The second Layer offers data transfer for one-dimensional arrays.
 Also, it contains transaction header structures and methods that can be used to build up a client-server handshake protocol.
 User application is responsible for the proper memory garbage collection.

A typical CS layer operation is presented below:

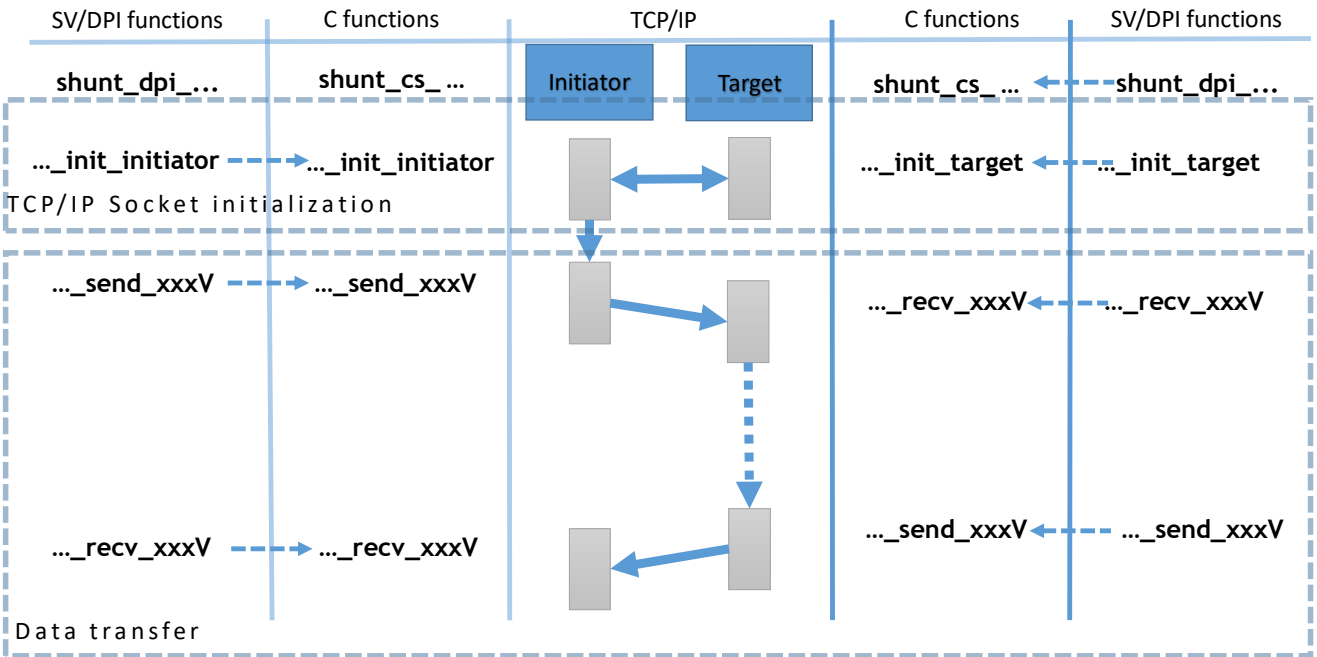


Figure 2 CS Operation

"C to C" SC examples: [Shunt/examples/client_server](#)

"SV to SV" SC examples: [Shunt/examples/sv/sv2c/initiator](#)
(Ref. to xxxV_loopback tasks)

3.5. Handshake API layer (HS)

This layer provides framework for TCP/IP handshake communication.

Users can setup their own packet or utilize/extend the predefined header/data payload pair.

The HS supports dynamic (vector) and static (array) data structures for all major SV data types.

The user application is responsible for the proper memory garbage collection

An example of Client-Server handshake is presented below:

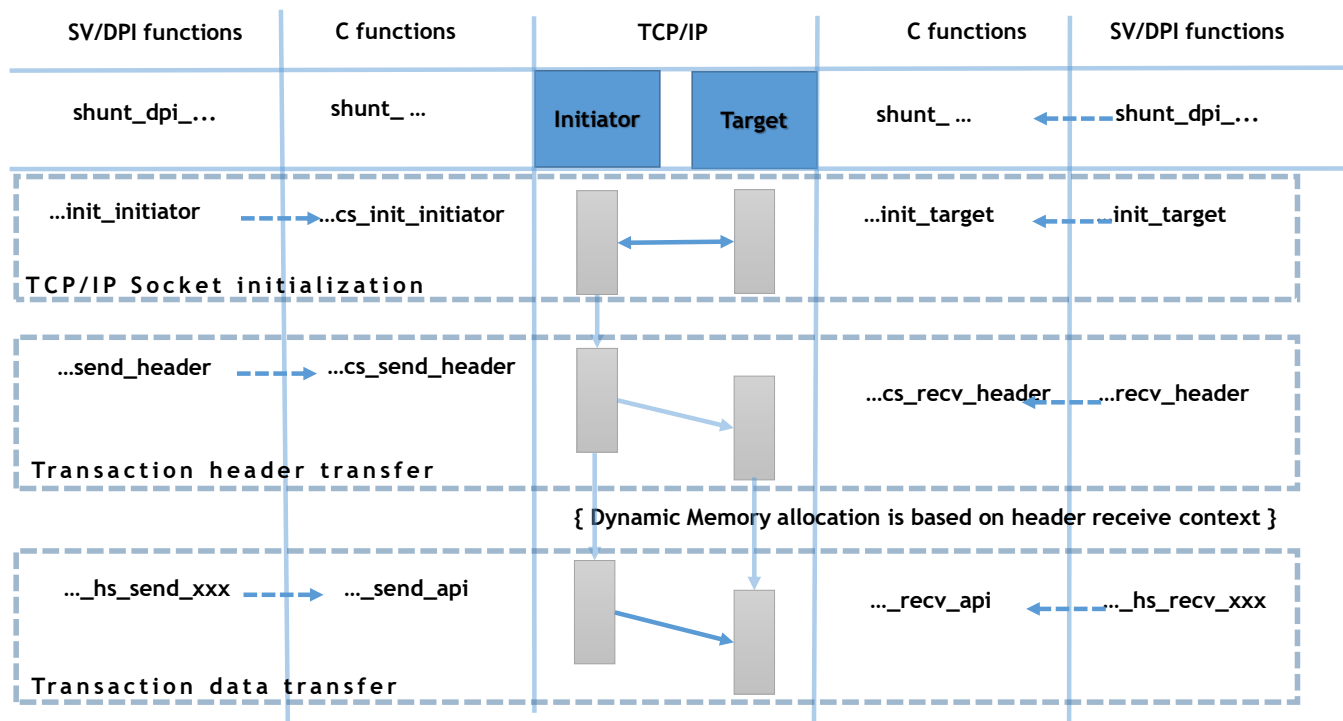


Figure 3 HS Operation

"C to C" HS examples: [Shunt/examples/c/user_api/](#)
"SV to SV" HS examples: [Shunt/examples/SV/handshake](#)

4. Getting Started

4.1. Library Installation

Download Shunt from <https://github.com/xver/Shunt>
Setup following variables:

- "SHUNT_HOME" to the SHUNT home directory.
- "SHUNT_SVDPI" to the location of svdpi.h file

4.2. Library compilation

- Go to "\$SHUNT_HOME/utils/makedir/utils/makedir"
- Run "make clean;make all"
- Compile result will be placed under \${SHUNT_HOME}/bin as a "libutils.so"

4.3. Compile and run C examples

- Go to example makedir root directory.
(Example: \$SHUNT_HOME/examples/c/primitives/makedir)

- To compile the library and launch test run "make all"

4.4. Compile and run SV examples

The Shunt includes a complete Makefile structure for the C portion of the library, but ONLY Makefile target place holders for SV domain.

- Edit "Makefile" initiator/target "compile_sv" under initiator/target local makedir
(Example: \$SHUNT_HOME/examples/sv/sv2c/initiator/makedir)
- Edit "run" file placeholder. Setup run command for appropriate source.
- Go to example makedir root directory run "make all" and ./run
(Example: \$SHUNT_HOME/examples/sv/sv2cmakedir)

5. Library version scheme and roadmap

The Shunt is adopting semantic versioning scheme as:

- It starts at 1.0.0
- Bug fixes, Patch release - the last number increment, e.g. 1.0.1.
- Minor release - the middle number increment, e.g. 1.1.0.
- Major release - the first number increment, e.g. 2.0.0.

Minor releases:

- Add makefile targets for all major SV simulators
- SV Queue and Associative array support

Major releases:

- Add UVM and TLM 2.0 support.
- TCP/IP client-server optimization

6. Contact information

If you have any questions please contact us at icshunt.help@gmail.com