HLSVerifier - A Verification System for High-level Synthesis

Marakkalage Dewmini Sudara
Processor Architecture Laboratory - EPFL
dewmini.marakkalage@epfl.ch

Abstract

HLSVerifier is a simple verification system for high-level synthesis (HLS) from C to VHDL. It supports the following three verification operations:

1. Verification of the input for HLS, which is a C function.
2. Verification of the output of HLS, which is a VHDL entity-architecture pair.
3. Co-simulation of the input C function and output VHDL entity.

1 Introduction

In this report, we present an overview of a simple verification system for HLS, which we call *HLSVerifier* and explain how to use it in practice. It is designed to perform three different types of verification for high-level synthesis that converts synthesizable C into VHDL. Namely, it supports the verification of the input for HLS (*cver*), the verification of the output of HLS (*vver*), and the co-verification of both the input and output of HLS (*cover*). A block diagram of the verification system depicting these operations and their relations are shown in Figure 1.

1.1 Verification of HLS Input (cver)

This verifies the C function, which is the Function Under Verification (FUV) that is provided as input to HLS by first running a provided C test-bench, and then comparing the results with provided **references**.

For this, HLSVerifier first compiles the C test-bench along with the input C function used in HLS, and runs it to generate outputs. It is assumed that this execution generates output data files for each output (or inout) parameter of the C function in a sub-directory called C_OUT. (See Preliminaries section for directory structure and naming convention).

1.2 Verification of HLS Output (vver)

This verifies the entity-architecture pair generated by HLS. For this, it takes a C test-bench as input and use it to generates a VHDL test-bench for the generated entity. It then simulates it by invoking ModelSim and compares the generated outputs with provided references. The VHDL test-bench is designed such that it generates the output files for each output (or inout) in a sub-directory called VHDL_OUT.

1.3 Verification of HLS Input and Output (cover)

This basically simulates both the C test-bench and the generated VHDL test-bench, and then compares the outputs of C simulation with those of the VHDL simulation.

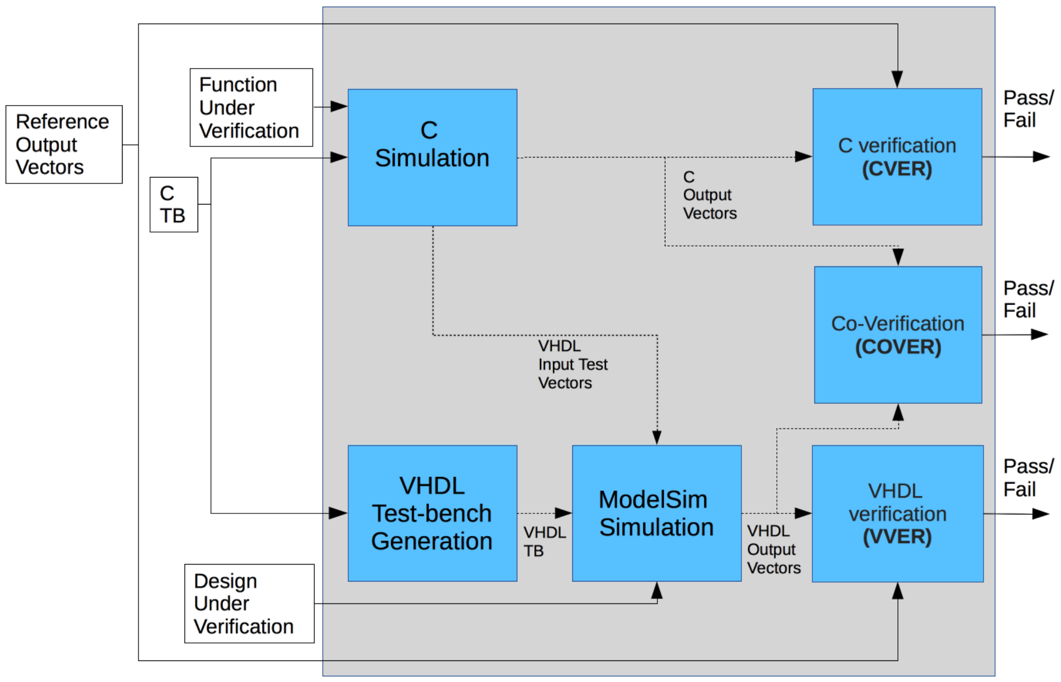

Figure 1: High-level block diagram

2 Preliminaries

Before we proceed, we note some prerequisites for running HLSVerifier, and introduce the naming convention it uses.

2.1 Tool Dependency

To use *cver* and *cover* of HLSVerifier, GNU C compiler is required. To use *vver* and *cover*, ModelSim is required. It is also necessary for the binaries of these tools (gcc and vsim) to be in the executable path when HLSVerifier is run.

2.2 Directory Structure

HLSVerifier assumes the following directory structure.

```
PROJECT_ROOT
    |__ C_SRC
    |__ VHDL_SRC
    |__ REF_OUT
    |__ HLS_VERIFY
```

The verification environment *must* be invoked within the `HLS_VERIFY` sub-directory, and once invoked, it might create additional directories named `C_OUT`, `INPUT_VECTORS`, and `VHDL_OUT`. The roles of these directories are as follows:

C_SRC	All C source files (both the FUV and the test-bench) are placed in this directory.
VHDL_SRC	All VHDL source files are placed in this directory.
INPUT_VECTORS	Input data files (.dat) for VHDL simulation are placed in this directory.
REF_OUT	The golden references for all outputs are placed in this directory.
HLS_VERIFY	This is the working directory of HLS verifier
C_OUT	The generated output files (.dat) of C test-bench are placed in this directory.
VHDL_OUT	The generated output files (.dat) of VHDL test-bench are placed in this directory.

2.3 IO Definition Style for C

HLSVerifier also assumes that the C Test-bench clearly identifies the inputs, outputs, and inouts of the C function that is being used as the input for HLS.

Moreover, it assumes that the types are properly defined. If a function parameter is used as input, it should be of a type that starts with the prefix `in_`. Likewise, if it is used as an output or if it is used as both input and output, it must have type prefixed with `out_` or `inout_` respectively. These argument declarations should be defined in the header file (.h file) of the FUV.

For example, consider a function magic, that takes three arguments `foo`, `bar`, and `foobar`. Suppose that foo is an input, bar is an output, and foobar is used as both input and output. Then the respective function declaration should be as follows:

```
    void magic(in_type1_t foo[4], out_type2_t bar[4], inout_type3_t *foobar);
```

2.4 Data File Format

HLSVerifier assumes that the input and output data files follows the following format. To elaborate, a data for a single test is wrapped inside opening and closing *runtime* tags. Within each test, there are multiple test vectors, and each such test vector is wrapped inside opening and closing *transaction* tags. Data is written in the hexadecimal format. Please note that the golden reference output vectors should also be in this format.

```
[[[runtime]]]
[[transaction]] 0
0x5
0x5
0x5
0x5
[[/transaction]]
[[transaction]] 1
0x5
0x5
0x5
0x5
[[/transaction]]
[[[/runtime]]]
```

2.5 Naming Convention

HLSVerifier assumes/uses following naming convention.

2.5.1 Port Names of HLS Output

Any HLS output entity must have the following 5 ports.

Signal	Direction	Description
clk	in	Clock signal
rst	in	Reset signal
start	in	Start command for DUV
done	out	High when DUV operation is done
idle	out	High when DUV is idle

Additionally, for each parameter in the C function that was input to HLS, there is a set of associated signals depending on its type. For a function parameter named xyz, consider the following ports for different types.

Type	Signal	Direction	Description
Input xyz	xyz_din	in	Data Bus
Input *xyz	xyz_din	in	Data Bus
Input xyz[arr_length]	xyz_din	in	Data Bus from memory
	xyz_address	out	Memory address (array index)
	xyz_ce	out	Chip select signal
Output *xyz	xyz_dout	out	Data Bus
	xyz_ce	out	Chip select signal
	xyz_we	out	Write enable signal
Output xyz[arr_length]	xyz_dout	out	Data bus to the memory
	xyz_address	out	Memory address (array index)
	xyz_ce	out	Chip select signal
	xyz_we	out	Write enable signal
In-Out *xyz	xyz_din	in	Data Bus in
	xyz_dout	out	Data bus out
	xyz_ce	out	Chip select signal
	xyz_we	out	Write enable signal
In-Out xyz[arr_length]	xyz_din	in	Data Bus from memory
	xyz_dout	out	Data bus to the memory
	xyz_address	out	Memory address (array index)
	xyz_ce	out	Chip select signal
	xyz_we	out	Write enable signal

If the function is not a void type but has a return value, then there will be another signal in the entity interface as shown below. The name of the signal in the interface will always be hls_return_dout.

Signal	Direction	Description
hls_return_dout	out	Return data
hls_return_ce	out	Chip select signal
hls_return_we	out	Write enable signal

2.5.2 File Names

The reference files must be named as follows.

Reference files:	output_⟨c_function_parameter_name⟩.dat	E.g.: output_bar.dat, output_foobar.dat

The C output data files, VHDL input vector files and VHDL output data files are also named in a fashion similar to the reference files. However, to distinguish these files, the reference files, C output files, VHDL input vector files, and VHDL output files are placed in directories REF_OUT, C_OUT, INPUT_VECTORS ,and VHDL_OUT respectively.

2.6 Supported C Data Types

Data Type	Data Width (bits)
char	8
signed char	8
unsigned char	8
short	16
short int	16
signed short	16
signed short int	16
unsigned short	16
unsigned short int	16
int	32
unsigned int	32
signed int	32
long	32
unsigned long	32
signed long	32
long int	32
unsigned long int	32
signed long int	32
unsigned	32
signed	32
long long	64
unsigned long long	64
signed long long	64
long long int	64
unsigned long long int	64
signed long long int	64
float	32
double	64
long double	128

For integer type data, an exact comparison will be performed whereas for the float type data, the difference between two values should be lower than a threshold value for the comparison to be successful.

3 Using HLSVerifier

As introduced earlier, HLSVerifier supports three operations, and the usage is described below:

3.1 Verifying the input to HLS (cver)

To verify the C function that is used as the input in HLS, a C test-bench must be provided by the user. The command for invoking cver is given below:

```
$ hlsverifier cver path_of_c_test_bench path_of_c_FUV FUV_name
E.g.,: $ hlsverifier cver ../C_SRC/array_RAM_test.c ../C_SRC/array_RAM.c array_RAM
```

If the outputs of C and the reference output are same, it will print that the output comparison is successful.

3.2 Verifying the output of HLS (vver)

To verify the output entity-architecture pair of HLS (DUV), the user has to provide the VHDL files output by HLS in the VHDL_SRC sub-directory, and also provide a C test-bench and the top-level entity name of the DUV. The command for invoking vver is as follows:

```
$ hlsverifier vver path_of_c_test_bench top_level_entitiy_name_of_duv
E.g.,: $ hlsverifier vver ../C_SRC/array_RAM_test.c array_RAM
```

Note that top level entity name o DUV should be same as the name of FUV. If the outputs of VHDL and the reference output are same, it will print that the output comparison is successful.

3.3 Co-simulation of HLS Input and HLS Output (cover)

To simulate the C and VHDL designs in parallel and compare their outputs, the cover command is used. The syntax for this command is as follows:

```
$ hlsverifier cover path_of_c_test_bench path_of_c_fuv top_level_entitiy_name_of_duv
E.g.,: $ hlsverifier cover ../C_SRC/array_RAM_test.c ../C_SRC/array_RAM.c array_RAM
```

If the outputs of C and VHDL designs are same, it will print that the output comparison is successful.

4 Acknowledgement

We note that the VHDL codes for file IO are taken from the Vivado-HLS auto-generated test-benches. The input and output files encodes data using the same format as Vivado-HLS (I.e., using `runtime` and `transaction` tags). HLSVerifier also uses the boost regex library for using regular expressions. Further, to compile and execute the vhdl test-bench, we use ModelSim.