Hardware Build System
User Manual

Revision 0.0

19 December 2025

Abstract

This document serves as the official user manual for Hardware Build System (HBS). HBS is a Tcl-based, minimal common abstraction approach for build system for hardware designs. The main goals of the system include simplicity, readability, a minimal number of dependencies, and ease of integration with the existing Electronic Design Automation (EDA) tools.

keywords: automation, hardware build system, hardware design, hardware synthesis, project maintenance, testbench runner, simulation, FPGA, ASIC, productivity

Contents

Participants

Michał Kruszewski, *Chair*, *Technical Editor*, mkru@protonmail.com

Glossary

Not all terms defined in the glossary list are used in the user manual. Some of them are formally defined because they are helpful when discussing, for example, core definition.

core

Tcl namespace in which `hbs::Register` proc is called.

core name

Name of the Tcl namespace in which `hbs::Register` is called. For example, if `hbs::Register` is called in namespace `lib::pkg::my-core`, then `my-core` is the core name.

core path

Tcl namespace for the core. For example, if `hbs::Register` is called in namespace `lib::pkg::my-core`, then `lib::pkg::my-core` is the core path.

dependency

A target on which at least one other target depends. The dependency is an argument for at least one `hbs::AddDep` proc call.

depender

A target depending on at least one another target. Within a depender body the `hbs::AddDep` proc is called at least once.

flow

An ordered set of actions taken by a tool to produce a result specified by a user.

hbs file

A file with `.hbs` extension containing valid Tcl code.

proc

A Tcl procedure.

stage

A piece of a tool flow with a clearly defined task and output. The number and types of stages depend on a tool. For example, the GHDL has analysis, elaboration and simulation stages.

target

A proc, which name does not start with the floor character (_), defined in core.

target name

The name of the target in the target path. For example, if the target path is `lib::pkg::my-core::my-target`, then the target name is `my-target`.

target path

The Tcl path for the target. For example, if proc `my-target` is defined in the core with the core path `lib::pkg::my-core`, then the target path is `lib::pkg::my-core::my-target`.

tool

A software capable of processing hardware description sources or output from another tool. Example tools are: GHDL, Verilator, yosys, Vivado, nvc, etc.

to run a target

To execute commands defined in the target.

1 Overview

Hardware Build System (HBS) is a build system for hardware design projects. A build system for hardware design collects and processes files required for FPGA programming, ASIC production, running functional simulation, or carrying out formal verification. The files required to obtain desired output usually include much more than just classic hardware description files such as VHDL or SystemVerilog sources. For example, any synthesis or place and route tool requires also design contraints, at least for pin location and timing closure analysis. Moreover, most of real projects are not implement from scratch and utilize third-party IP cores. Those IP cores might be provided in different formats. Sometimes, they might even be encrypted. HBS tries to support all files that might potentially be required to generate final result. HBS does not limit to managing only pure hardware description files.

HBS was created out of frustration with all existing build systems for hardware designs.

2 Installation

All installation methods require that `hbs` and `hbs.tcl` files are placed in the same directory. There are four preferred installation methods.

1. Copy `hbs` and `hbs.tcl` files to your project. This is preferred if you want to modify HBS source files change its default behavior. It is not advised to change the default behavior, but if you need, feel free to adjust the build system to your project needs.
2. Copy `hbs` and `hbs.tcl` files to one of the directories in the `$PATH` environment variable.
3. Clone the repository and add its path to the `$PATH` environment variable.
4. Clone the repository and add an alias to the `hbs` file in `.bashrc` file (or equivalent).

2.1 Dependencies

HBS has three dependencies, one mandatory and two optional.

1. `tclsh (version >= 8.5).`
2. `python3` - required for testbench target detection, automatic testbench running and dependency graph generation. If mentioned functionalites are not required, you can directly used `hbs.tcl` script instead of the `hbs` Python wrapper.
3. `graphviz` - required only if user wants to generate a dependency graph.

3 Internal architecture

3.1 General structure

3.2 Cores and cores detection

3.3 Targets and targets detection

3.4 Testbench targets

3.5 Running targets

3.6 Targets parameters

3.7 Target context

3.8 Target dependencies

3.9 EDA tool flow and stages

3.10 EDA tool commands custom arguments

3.11 HBS API extra symbols

3.12 Code generation

4 Command line interface commands

4.1 doc

4.2 dump-cores

4.3 graph

4.4 help

4.5 list-cores

4.6 list-targets

4.7 list-tb

4.8 run

4.9 test

4.10 version

4.11 whereis

5 Tcl tips

5.1 Support for arrow keys in `tclsh`

If you every tried to use `tclsh` to REPL (read-eval-print-loop), you probably realized that `tclsh` by default does not support arrow keys. You can’t fix a typo in a line without deleting some line content. There is also no command history support. However, this can be improved. The first solution is install the `rlwrap` programm and call `rlwrap tclsh` instead of `tclsh`. To make it shorter to type, you can define an alias for your shell of choice, for example, for bash alias `tclsh='rlwrap tclsh'`. The second option is to install the Tcl `tclreadline` package. This package often comes as OS distro package. For example, on Debian/Ubuntu, you can install it with `apt install tcl-tclreadline`. Once installed, create `.tclshrc` file in your home directory and add the following content:

```
package require tclreadline
tclreadline::Loop
```

Not only you will get support for arrow keys and command history, but also improved prompt.

5.2 Passing variadic arguments to proc

To pass variadic arguments to a proc, the last proc parameter must be called `args`. You can then easily iterate over the arguments using the `foreach` loop. The below example is taken directly from the hbs Tcl source code.

```
proc AddIgnoreRegex {args} {
  foreach reg $args {
    hbs::dbg "adding ignore regex $reg"
    lappend hbs::IgnoreRegexes $reg
  }
}
```