The HDL Test-bench User Manual

Bertrand Blanc

8th April 2002

Contents

1 Synopsis

`makehdltb` < **Esterel file** > < **ESI file** > < **options** >

< Esterel file >	< file.strl > | -strl < file > | -strl < file.strl > -main < main module >
< ESI file >	< file.esi > | -esi < file > | -esi < file.esi >
< options >	-o < output file > | --output < output file >
	< file.eso > | -eso < file > | -eso < file.eso >
	-v | --verbose
	-B < basename > | --basename < basename >
	-stdout
	-vhdl87 | -vhdl93 | -verilog | -s | --skeleton | -C | -cpp
	-dump "< white-separated-character dumped signals list >"
	--version
	--help
	--no-eso
	--no-comment
	--no-assert
	--string-size < string size >
	--integer-size < integer size >
	--no-user-type-assert
	--dc-version < 1 | 2 >
	--eso-format < coresim | signal_record >
	--evaluation-mode-allowed
	--automatic-assertions

Parameters order is not significant, options are optionals but Esterel file and ESI file are mandatory.

2 Description

We provide a tool, `makehdltb` , which is able to generate a test-bench into a specified HDL target language, taken as inputs an Esterel module and an ESI file describing a scenario. Moreover, some options are offered to specify either a way of HDL generation style, or the HDL language set, or other possibilities which will be told.

The application provides a test-bench, generated in a selected target language as VHDL, Verilog, C or C++, of an Esterel module applied on an ESI secenario. By default, the selected HDL target language is VHDL 87. Basically, `makehdltb` will generate a file which name is written by the separated underscore character concatenation of the Esterel module name, the ESI file name and the suffix `tb` representing the test-bench. Therefore, during the simulation an ESO file will be dynamically generated, showing the inputs and outputs set and the possible run-time error or violated assertions.

3 Inputs

< Esterel file >	Esterel input file describing input and output signals, sensors, relations. Such a file is indicated by his extension **.strl**. Nevertheless a possibility is offered, setting **-strl** keyword followed by the Esterel file ended or not by his extension.
< ESI file >	ESI input file describing a scenario applied on Esterel module inputs. Such a file is indicated by his extension **.esi**. Nevertheless a possibility is offered, setting **-esi** keyword followed by the ESI file ended or not by his extension.
< main module >	As an esterel file should contain more than one module, the option **-main** set the top level module with the string following the option. By default, the toplevel module is **MAIN**, basically called by Esterel Studio.

4 Options

-o, **--output**	followed by a file name identifier changes the default generated test-bench output file by this specified name
-B, **--basename**	followed by a base name identifier changes the default generated test-bench output file prefix by this specified base name. Include files in C and C++ target languages are changed into this base name
-eso	followed by a file name identifier changes the default run-time generated ESO and BLIF files by this specified basename.
-v, **--verbose**	writes in the standard output file steps reached by the compiler
-stdout	writes the generated test-bench on the screen instead of an output file set or not
-vhdl87	set the target language into the VHDL 87 mode. Basically VHDL 87 is the default mode
-vhdl93	set the target language into the VHDL 93 mode.
-C	set the target language into the C mode. An instrumented C scenario is generated
-cpp	set the target language into the C++ mode. An instrumented C++ scenario is generated
-verilog	set the target language into the Verilog mode
-s, **--skeleton**	set the generation style describing the generated test-bench in the ESI format. Thus, a not well indented ESI input file will be thereafter well formatted
-main	followed by a module name, changes the default main module name set on **MAIN**
-dump	adds mandatory outputs signals following the option. They will be written at run-time in a blif-formatted file
--version	writes the version number of the running `makehdltb`
--help	writes the version number and usage of the running as depicted by the synopsis section`makehdltb`
--no-eso	specifies no ESO file will be waited, generated during the run-time. This implies no added code is generated
--no-comment	the input scenario file is translated the best as possible including user comments. This option avoids these comments
--no-assert	assertions are now described in the extended ESI format. Then this option set off such statements including invariants
--string-size	followed by an integer set the string size
--integer-size	followed by an integer set the bit vector size representing integer type in Verilog
--no-user-type-assert	avoids assertions on user-defined types but allows assertion on pure signals and primitive types
--dc-version	followed by 1 or 2 represents the translation of Esterel valued signals into a selected target language. By default, the current version is set on 2
--eso-format	Outputs, ticks, ... are differently formatted according to the used ESO generator. Then, by default, the run-time generated ESO file is the most precise as possible. However, through this option followed by **coresim**, ESO file is formatted like ESO file generated by *libcoresim* and **signal_record** like ESO file generated by the Esterel Studio instrumented code.
--evaluation-mode-allowed	**evaluation_mode** commands are supported in the ESI input scenario file. Basically, these ESI commands are forbidden.
--automatic-assertions	ESO file should be considered as ESI file. Therefore, the comment emitted output are translatted into assertions ensuring, after tick event, outputs are well emitted

5 Outputs

As shown in the description section, by default, this compiler will produce a file containing the test-bench of an Esterel module — called MOD — applied on an ESI-format scenario — described in a file called scenario.esi — in VHDL target language.

Thus, this produced file will be called *MOD_scenario_tb.vhd*. At the run-time, during the simulation phasis, a trace will be written in an ESO, which is an ESI-like language, formatted file, showing inputs which had been set and outputs set after having called the reactive function. Then, adding to the informations contained in scenario.esi, this file — called by default *record_scenario.eso* — will contain the outputs set with their values, if signals are valued, and the possible violated assertions shown throw an explicit message.

Moreover, intern registers signals could be caught through *-dump* option. This state dump instrumentation will produce, at run-time, a blif-formatted trace which will be written in a file — called by default *dump_scenario.blif* —. This name can be changed by the user through the option *-eso* followed by an ESO file. So, the basename will be suffixed by *.blif*.

Nevertheless, options make `makehdltb` produce other output file. The following table describes the modified output file generated in function of the different options giving as inputs arguments. Then, no added code will be generated, making HDL test-bench file shorter and perhaps clearer.

Option	Description
-o --output	the default generated output file *MOD_scenario_tb.vhd* will be renamed into the name specified after this option
-B --basename	the default generated output file *MOD_scenario_tb.vhd* will be renamed into *<basename>_scenario_tb.vhd*
--no-eso	no ESO file will be generated during the simulation phasis
--no-comment	scenario comments will not be generated
--no-assert	assertions and invariants won't be generated
--no-user-type-assert	assertions and invariants based on user-defined type won't be generated. Then user is expected to avoid writing assertions including such types. So, user is able to write assertions based on pure signals or primitive valued signals, without writing in the target language the functions required by user-defined types

Moreover, select target langage option makes `makehdltb` produce other output file which default name are shown in the next table.

Target Language	Option	Default output name
VHDL	-vhdl	MOD_scenario_tb.vhd
Verilog	-verilog	MOD_scenario_tb.v
C	-C	MOD_scenario_tb.c
C++	-cpp	MOD_scenario_tb.cpp
ESI	-skeleton, -s	MOD_scenario.esi

Path files

Files giving as parameters to the test-bench application should be, of course, everywhere in the file system. In order to be clearer, test-bench generated file is produced into the current directory from where the application is run.

Assertions flag

In the front within generated scenario file a flag **AssertionFlag** allows user to set off assertions evaluation. By default, this flag set evaluation on.

6 Messages

`makehdltb` provides error messages when an error occurs such as, for example, if a signal is enabled in the scenario file though it is not declared in the Esterel module. Thus, the table below shows and comments messages which could be caught.

Message	Description
Error (in file < file >, line < line >) : pin < pin > does not exist in the interface < interface > (in file < Esterel file >).	`pin` wants to be used in the scenario `file` thought it was not declared in the `Esterel file`
Type error (in file < file >, line < line >, Manual 7.2.3): pin < pin > is declared in module < module > (file < Esterel file >) with the type < type > but used with an other one (set to "< setting >").	in the scenario file `file`, the signal `pin` wants to be set on the value `setting` which is not in the signal declared set
Syntax error (in file < file >, line < line >) : "< read >" is unexpected.	in file `file` a syntacticaly error occurs reading the token `read`
Emit error (in file < file >, line < line >, Manual 7.2.3) : "< pin > has been emitted yet.	emitting an even signal `pin` more than once is prohibited
Pin error (in file < file >) : "< pin >", HDL features not allow such a name.	some signals identifiers are prohibited in Esterel module, since they are reserved by HDL translation. Such names are the ones ended by `data` and `clk` and `rst`
Pin error (in file < file >, line < line >, Primer 4.2) : pin < pin > already exists.	a signal identifier cannot be declared more than one time
Not synthetisable (in file < file >, line < line >, < standard >): < read >.	the `read` code beyond is not synthetisable through `standard`
Assert error (in file < file >, line < line >) : < read >.	`read` throws this error, which has occured in an assertion or invariant
File error: "< file >" not accessible.	`file` cannot be opened
Main module not found (in file < file >): main module "< module >" is requested but not found	`module` not found
< severity level >: assertion violated in file < file >, line < line >	an assertion is violated at run-time
Syntax error (in file < file >, line < line >): valued signal < signal > assigned but evaluation mode pure set.	*evaluation mode pure* set implies that only statuses are expected
Syntax error (in file < file >, line < line >) set option to use the evaluation mode commands.	*evaluation mode* commands can be only used if **--eval-mode-selection-allowed** is set.

Thus, where do *standard*, *Manual* or *Primer* come from. These abbreviations are presented below, generally followed by the concern in-book section.

Abbreviation	Manual
Primer	The Esterel v5 Language Primer Version v5_91
Manual	The Esterel v5_91 System Manual
IEEE VHDL	IEEE Standard VHDL Language Reference Manual
VHDL Synt.	IEEE P1076.6/D2.0 Draft Standard For VHDL Register Transfer Level Synthesis
IEEE Verilog	IEEE P1364 Draft Verilog Hardware Description Language
Verilog Synt.	IEEE P1364.1/D1.6 Draft Standard For Verilog Register Transfer Level Synthesis

7 Extended ESI

Usual ESI language was extended in order to add some other statements such as, in particulary, assertions and invariants. Invariants are considered as assertions which are to be evaluated every *tick*. Then this section will present the expressions and their meaning in assertions. This statement could begin either by `% assert` or `% invariant`, ended by the current end-line. It will be commented adding before the beginning comment character as usually %.

Invariants could be described in the Esterel file throw the **relation** constructions. However it could not be describe as an extended ESI language feature.

Severity level in assertion

Moreover, a severity level could be set to adopt a behavioural strategy when an assertion failure happens. The following table summarizes these levels and their meanings.

Level	Description
note	it could be considered as a breakpoint to ensure the point of the scenario where this assertion failed, is well reached. Therefore, the control flow can normally carry on.
warning	the control flow can in fact go on, however the meaning is a little bit different than the one represented by `note`. It's not considered as an error but an advertising.
error	the simulation completes, showing that an error occured.
failure	the simulation completes, an error occured meaning that the global behaviour could crash.

But, how should I set these severity level? The following table presents it.

Starting keyword	Severity level
! assert	error
! assert note	note
! assert warning	warning
! assert error	error
! assert failure	failure
! note	note
! warning	warning
! error	error
! failure	failure
! invariant	error
! invariant note	note
! invariant warning	warning
! invariant error	error

Expressions

The tables, referenced 1 and 2 highlight operators allow by types and their precedence. Starting from the top, each group of operators has precedence over the next. Operators within a particular group have the same level of precedence when used within an expression. Several operators could be written either in a C-like style, considered as alias, or in an Esterel-like style which is best considered since we are in an Esterel world. Nevertheless C-like notation has been conserved in order to let C-friendly users in their well-known logical environment.

Keyword	Associativity	Type	Alias	Description
()		high priority		parenthesis
$?_{\vert 1}$		valuated signals		value accession
$?_{\vert 1}$		sensors		value accession
$present_{\vert 1}$		signals		status
not		logical	!	negation
$-_{\vert 1}$		arithmetical		unary −
/		arithmetical		division
mod		arithmetical	%	modulo
*	left	arithmetical		product
$+_{\vert 2}$	left	arithmetical		binary +
$-_{\vert 2}$	left	arithmetical		binary −
=		logical	==	equality
<>		logical	!=	inequality
<		logical		strict less
>		logical		strict greater
<=		logical		less or equal
>=		logical		greater or equal
and	left	logical	&&	and
and then	left	logical		and conditional
nand	left	logical	!&&	not and
or	left	logical	$\vert\vert$	or
or else	left	logical		or conditional
xor	left	logical	^	exclusive or
nor	left	logical	!$\vert\vert$	not or
xnor	left	logical	!^	not exclusive or
implies		logical	=>	logical implication

Table 1: Operators

Example

This part explains throw an approach based on examples how could we use *assertions* and *invariants*.Here a piece of an Esterel interface followed by a piece of an ESI scenario. Notice that assertions and invariants are to be declared before inputs and sensors settings.

```
module TEST;
type data;

input I_PURE;
input I_INT : integer;
input I_DATA : data;
output O_PURE;
output O_INT : integer;
output O_DATA : data;
sensor S_INT : integer;
```

Type	Operator	Description
integer	$=, <>, <, <=, >, >=$ $+, -, *, /, mod$ $-_{\|1}$	relationship operators arithmetical operators unary $-$
string	$=, <>, <, <=, >, >=$ $+$	relationship operators concatenation operator
boolean	$=, <>, <, <=, >, >=$ and, and then, or, or else xor, nand, nor, xnor, implies not	relationship operators logical operators unary logical operator
pure signal	and, and then, or, or else xor, nand, nor, xnor, implies not	logical operators unary logical operator
user-defined type	$=, <>, <, <=, >, >=$	relationship operators implemented by user in the target language by overloading operators

Table 2: Operators by type

```
sensor S_DATA : data;
loop
  await case [ I_PURE ] do
    emit O_PURE
  end await
end loop
||
loop
  await case [ I_INT ] do
    emit O_INT ( ? S_INT )
  end await
end loop
||
loop
  await case [ I_DATA ] do
    emit O_DATA ( ? S_DATA )
  end await
end loop
end module
```

```
! warning !(present O_PURE and not (present O_INT or present O_DATA))
I_PURE    S_INT ("2")   S_DATA ("22;22");
! note !((?O_INT <> ?I_INT) and (?O_INT = ?S_INT) and not (present O_PURE
    or present O_DATA))
I_INT = "99"    S_INT ("3")   S_DATA ("33;33");
! failure !(present O_DATA and (?O_DATA = ?S_DATA) and not (present O_PURE
    or present O_INT))
I_DATA = "99;99"    S_INT ("7")   S_DATA ("77;77");

! invariant error (present I_PURE implies O_PURE)
```

or (**present** I_INT **implies** ?O_INT = ?S_INT)
or (**present** I_DATA **implies** ?O_DATA = ?S_DATA)
; % This tick is mandatory to mandatory take care of this invariant

New commands

New case-insensitive commands were added in order to provide, in a first hand, a more readable scenario file. Thereby, standard ESI usual commands were aliased such the table below lets see.

Esi command	Extended ESI alias	Description
.	!quit !exit !bye	program exit requested

8 User-defined types

User-defined types require declarations, functions or procedures implemented by user in the target langage. Then, some features are expected being followed in order to produce a well-compiled source generated code.

To illustrate this per target langage, we are aimed to set an Esterel module BAR in file FILE.strl, an ESI scenario in SCENARIO.esi and an user-defined type TYPE.

VHDL

Packages depicted in the following table are expected.

Package	Description
BAR_data_type_pkg	contains user-defined data types and their definition
BAR_data_pkg	contains constants and data types assign procedures
BAR_data_sim_pkg	contains simulation functions such as *check_TYPE*, *text_to_TYPE*, *TYPE_to_text* ones, and relationship operators overloads

Verilog

Files depicted in the following table are expected.

File	Description
BAR_data_type_pkg.v	contains user-defined data types and their definition using `` `define ``
BAR_data_pkg.v	contains constants and data types assign procedures
BAR_data_sim_pkg.v	contains simulation functions such as *check_TYPE*, *text_to_TYPE*, *TYPE_to_text* ones

C

Files depicted in the following table are expected.

File	Description
FILE.h	contains user-defined data types and their definition. This file contains simulation functions such as *_check_TYPE*, *_text_to_TYPE*,*_TYPE_to_text* ones
SignalRecord.c SignalRecord.h	mandatory code used by instrumented code

Then you can at least compile:

- *prompt* > esterel -Lc:"-signal_record" FILE.strl
- *prompt* > makehdltb FILE.strl SCENARIO.esi -C
- *prompt* > cc -DSIGNAL_RECORD FILE.c BAR_SCENARIO_tb.c SignalRecord.c -o FILE.exe

C++

Files depicted in the following table are expected.

File	Description
FILE_data.h	contains user-defined data types and their definition. This file contains simulation functions such as *_check_TYPE*, *_text_to_TYPE*,*_TYPE_to_text* ones
SignalRecord.c SignalRecord.h	mandatory code used by instrumented code

Then you can at least compile:

- *prompt* > esterel -Lcpp:"-signal_record" FILE.strl
- *prompt* > makehdltb FILE.strl SCENARIO.esi -cpp
- *prompt* > gcc -c SignalRecord.c
- *prompt* > cxx -DSIGNAL_RECORD FILE.cpp BAR_SCENARIO_tb.cpp SignalRecord.o -o FILE.exe

9 Performance results

In this section we present results obtained with the following input files:

- an Esterel module declaring 98 mixed inputs, outputs and sensors, valuated or not. As valuated signals and sensors are translated into HDL target language by two signals, declared signals reach a number of 182
- an ESI scenario containing 65,000 statements and 12,000 reaction function calls. This file size is about 1,7Mb big

based uppon the following system features:

- Linux RedHat 6.2 operating system

- Intel PIII 1GHz processor
- 2GB RAM

The following table shows results, using no swap memory, generating as output file, a test-bench written in the specified target language.

	VHDL (no option)	VHDL (options) --no-comment --no-eso	Verilog (no option)	Verilog (options) --no-comment --no-eso	ESI (no option)
user time	15.54	7.00	14.49	7.40	3.63
system time	10.59	3.21	10.31	3.50	0.81
elapsed time	0:26.94	0:10.29	0:28.50	0:11.32	0:04.66
CPU (%)	96	99	97	96	95
size (MB)	153.76	29.61	132.06	32.53	1.82

10 Incompatibilities with standard ESI

Almost all the standard ESI language is supported except the three following points:

- extended ESI commands must begin with '!', even thougt we are not obliged, in standard ESI, to do so
- commands not currently supported are skipped

11 Bug report

If you find a bug or if you have some other requirements, ideas, etc, I would appreciate that you send me an e-mail to report this. My e-mail address is

Bertrand.BLANC@esterel-technologies.com.