VeriGEN

Release v0.1

Oct 28, 2019

Introduction

VeriGEN is a simple, general-purpose text generation tool that reads any text source and processes the python script embedded within it. The output text is a mixture of original text and python script output.

The aim was to create a single-file module that can be easily incorporated in a project build system, for example, CMake.

CHAPTER 1

Project origins

I started this project once I observed quite a frequent need for having portions of source code generated in an automated way during the build process. In the meantime, I was working on open source NVDLA architecture. In their hardware project available on GitHub I found an interesting script called epython. It is exceptionally simple in implementation, but I noticed how powerful it can be when used wisely :).

Here are example use cases from real projects, where I partially managed to use or I have seen used automated code generation:

1. Given register specification in SystemRDL domain specific language generate:
    - RTL synthesisable register backend
    - RTL simulation test vectors
    - Register documentation
2. Given domain specific YAML specification with list of process variables:
    - Generate documentation describing each varibale, their limits, measurement unit, etc.
    - Generate C++ and Python wrapper around Redis database communication where process variables are actually stored
    - Generate XML or JSON description file that can be incorporated in third party tools

The RTL code generation is an especially interesting topic. There are already very good tools for interpreting SystemRDL specifications, like systemrdl-compiler.

CHAPTER 2

VeriGEN is not

- ... a runtime engine for dynamic content creation basing on templates. The embedded code execution **is not sandboxed**, which makes VeriGEN vulnerable when run on untrusted source files.
- ... an alternative to *Jinja2* or any similar template engine. Generating C-header files and some *restructured-Text* when building CMake project checked out from controlled repository is probably OK. Generating HTML content basing on user provided input is *not* OK.

2.1 Theory of operation

TODO

2.2 CMake example

TODO

2.3 Command line parameters

2.3.1 Help

This is the help output of the *verigen* tool. Fun fact: generated by *verigen* itself.

```
usage: verigen.py [-h] [-v] [--verbose <N>] [-o,--output <file>] [-l <lang>]
                  [--print-lang-specs] [-s <file>]
                  [input [input ...]]

VeriGEN, Versatile Text Generator, ver. 0.1
```

(continues on next page)

(continued from previous page)

```
positional arguments:
  input                 Input file. For standard input use '-'

optional arguments:
  -h, --help            show this help message and exit
  -v, --version         Show version and exit.
  --verbose <N>         Diagnostics verbosity, 0 = lowest, 9 = highest
  -o,--output <file>    Ouput file. Standard output if unspecified.
  -l <lang>, --lang <lang>
                        Select language of source file. If not specified, try
                        to guess from file extension.
  --print-lang-specs    Print predefined language specification in JSON format
                        and exit.
  -s <file>, --lang-spec <file>
                        Load language specification from JSON file.
```

2.4 Translator

class verigen.**Translator**(*output*, *language=None*, ***kwargs*)

This class represents top level translation engine that generates single output from multiple sources.

Parameters

- **output** (*str*) – output filename or standard output placeholder (-)
- **language** (*str, optional*) – preferred language (enforced on all source input)
- **matcher_cache** (MatcherCache, *keyword, optional*) – custom cache of language matchers

Raises TranslationError – when translator is unable to find valid matcher for given language

find_matcher(*language: str*)

Find syntax matcher for given language name or it's alias.

Parameters **language** (*str*) – language name

Raises TranslationError – when language is not supported

Returns valid matcher that can be passed to translation units.

Return type class:~Matcher

select_stream(*file*, **args*, ***kwargs*)

Selects proper stream depending on file type or name and additional hints. To be used with with clause.

If file represents file path, all positional and keyword parameters except dir are passed to standard open() function.

Parameters

- **file** (*str, IOBase*) – filename or existing stream
- **dir** (*str, keyword*) – stream direction hint with valid values: 'input' (default) or 'output'

Yields *tuple* – This function yields tuple of two values: 1. Stream object 2. Stream source name string for diagnostic purposes

translate_all(*src_list*)
Translates all sources from the specified list.

Parameters **src_list** (*list*) – List of filenames, or standard input placeholders (-)

Raises TranslationError – On any severe translation error. Note that embedded script errors are not considered as 'severe' error.

translate_stream(*in_s*, *out_s*)
Translate one open input stream to the open output stream

Parameters

- **in_s** (*tuple (IOBase, str)*) – input stream and corresponding name for diagnostic purposes
- **out_s** (*tuple (IOBase, str)*) – output stream and corresponding name for diagnostic purposes

Raises TranslationError – On any severe translation error. Note that embedded script errors are not considered as 'severe' error.

2.5 Translation unit

class verigen.**TranslationUnit**(*matcher: verigen.Matcher*, *src*, *dest*, ***kwargs*)
This class represents translation engine invoked for single input file.

Parameters

- **matcher** (verigen.Matcher) – language matcher object
- **src** (*tuple(IOBase, str)*) – input stream and corresponding name for diagnostics
- **dest** (*tuple(IOBase, str)*) – output stream and corresponding name for diagnostics

STATE_GENERATED = 2
GENERATED state means that translation unit is passing through previously generated code. This state occurs when in-place translation is done multiple times

STATE_SCRIPT = 1
SCRIPT state means that current line is collected into script bucket and executed as soon as last script in the current block is detected.

STATE_VERBATIM = 0
VERBATIM state means that current line is copied **as is** to the output stream

issue_msg(*level*, **args*, ***kwargs*)
Issue diagnostics message

Parameters

- **level** (*int*) – severity level
- **args** (*list*) – additional parameters passed to diag()
- **line_no** (*str, keyword, optional*) – line number coordinate; if not specified currently translated line number is used

translate()
Process through translation of entire content in the input stream.

2.6 Language matching

2.6.1 Language matcher

class verigen.**Matcher**(*language: str*, ***kwargs*)

This class represents language specific syntax matching.

match(*text: str*)

Match provided text against rules of this matcher object. As a match result, *dict()* object is returned with predefined keys:

- **type - match type; one of: script, generated,** verbatim
- **scope - scope of the successfully matched line; for script** result it can be either common or local. For generated result it can be either begin or end. verbatim output does not produce any scope.
- indent - optional hint about indentation of the output text
- text - content of the script or verbatim text depending on type.

Parameters **text** (*str Line of text*) –

Returns

Return type dict Dictionary with match result.

supports_filename(*fname: str*)

Checks if this matcher can potentially support file name, basing on its extension.

Parameters **fname** (*str*) – File name or path.

Returns

Return type True if specified file may be supported by this matcher

supports_language(*language: str*)

Checks if this matcher supports specified language. Language is case insensitive and can have name aliases like C++ and CPP.

Parameters **language** (*str*) – Language name or it's alias (case insensitive)

Returns True if language is supported by this matcher

Return type bool

2.6.2 Matcher cache

class verigen.**MatcherCache**

Collection of language matchers initialized with predefined list of matchers.

find(*language: str*)

Find matcher that supports specified language

Parameters **language** (*str*) – language or alias name, case insensitive

Returns Instance of matcher object valid for specified language. None if language is not supported

Return type class:~Matcher

find_by_file (*fname: str*)
Find matcher by file extension.

Parameters **fname** (*str*) – File name or path

Returns Instance of matcher object valid for specified extension. None if extension is not supported.

Return type class:~Matcher

2.7 Miscellaneous

Documentatiton of miscellaneous functions present in verigen module.

2.7.1 Diagnostics

verigen.**diag** (*lvl*, **args*, ***kwargs*)
Print diagnostics at specified verbosity (or severity) level.

The output diagnostics tries to resemble (more or less) the GCC output.

Severity levels

- FATAL - fatal error, immediate exit
- ERROR - translation or embedded script error
- WARNING - translation warnings that require user attention
- INFO - translation process info
- DIAG - extra diagnostics for errors and warnings
- TRACE - for debugging only

Parameters

- **lvl** (*int*) – verbosity or severity level.
- **args** (*list*) – extra parameters passed as is to print function
- **file** (*str, keyword, optional*) – related file name
- **line_no** (*int, keyword, optional*) – related line coordinate

verigen.**keep_short** (*string*)
Make string shorter. Strip any newlines. Used by debug diagnostics

Parameters **string** (*str*) – Input string.

Returns Input or it's shorter version.

Return type str

2.7.2 Embedded code execution

verigen.**execute_embedded** (*cmd*, *globals=None*, *locals=None*, *description='source string'*)
This function executes specified command cmd and returns content of the standard output.

Parameters

- **cmd** (*str, required*) – script to execute
- **globals** (*list, optional*) – list of global variables passed to `exec`
- **locals** (*list, optional*) – list of local variables passed to `exec`
- **description** (*str, optional*) – description of executed code

Raises `EmbeddedScriptError` – when in-text script is ill-formed or cannot execute from other reasons

Index