Rev. 1.2

April 20th, 2016

Introduction

This document describes installation and usage of the C code header file generation script (HeaderGen) created by Renesas. The HeaderGen script is a Python based script that utilizes one external Python module for parsing Excel files. The purpose of the tool is to enable more configurable and consistent header file generation for our customers, as well as to provide some useful formatting options within the header files themselves.

Contents

[1. Installation 1](#_Toc442689055)

[2. Usage 2](#_Toc442689056)

[2.1 Configuration File 2](#_Toc442689057)

[2.2 Execution 3](#_Toc442689058)

[2.3 Output 3](#_Toc442689059)

[3. Revision Record 4](#_Toc442689060)

[4. Testing 5](#_Toc442689061)

# Installation

The following tools are needed for execution of the Script

|  |  |  |
| --- | --- | --- |
| **Tool** | **Version** | **Install instructions** |
| Python | v2.7.9 | Visit the Python Install site and download the latest v2.x.x Windows version:  https://www.python.org/downloads/ |
| openpyxl | v2.3.0 | (1) Download easy\_install from Windows from this link: <https://bootstrap.pypa.io/ez_setup.py>  (2) execute the file ez\_setup.py:   |  | | --- | | >python ez\_setup.py |   (3) Navigate to C:\PythonXX\Script and execute   |  | | --- | | > easy\_install openpyxl |   . |
| Excel | 2013 | Script requires use of the .xlsx Excel file format |

Table 1: Required SW Packages

# Usage

## Configuration File

The script takes as an input a text configuration file. Required fields in the file are:

|  |  |  |
| --- | --- | --- |
| **Field** | **Description** | **Example** |
| Input\_file\_name | Excel input file name and full or relative path | ./dr7f701310\_matrix\_smaller.xlsx |
| Tab\_name | Excel tab name with relevant data – typically “APB area2” | APB area2 |
| Output\_file\_location | Location where the resulting header files will be written with full or relative path.  **NOTE: this is currently unsupported** | ./ |
| Base\_type\_byte | Base type for byte variables | uint8 |
| Base\_type\_short | Base type for short (two byte) variables | uint16 |
| Base\_type\_long | Base type for long (four byte) type variables | uint32 |
| Base\_union\_name\_byte | Name for byte variable access within union of register access types | UINT8 |
| Base\_union\_name\_short | Name for short variable access within union of register access types | UINT16 |
| Base\_union\_name\_long | Name for long variable access within union of register access types | UINT32 |
| Base\_union\_name\_bits | Name for bit variable access within union of register access types. This will provide access to the bitfield structure. | BITS |
| [prefix] (sample text) | Anything following a line starting with [prefix] will be printed exactly at the start of the header file, after the inclusion guard #ifdef |  |
| [groups] (sample group) | By default, all register access structures, address binding pragmas, and access macros will be placed into the default output file. Adding a group to the configuration file will place any register whose name starts with the text following [groups] into a separate file.  More than one piece of text can follow a group name, as shown in the example. | [groups] ADC\_FILE = ADCD0, ADCD1  This will place all registers starting with either “ADC0” or “ADCD1” into an output file called dr7f701310\_ADC\_FILE.h, and cause them to skip the default header file.  [groups] DEFAULT =  This is the default group for all register information. It must be present in the configuration file. This will place all unmatched registers into an output file called dr7f701310\_DEFAULT.h. |
| [suffix] (sample text) | Anything following a line starting with [suffix] will be printed exactly at the end of the header file, before the inclusion guard #endif |  |
| [skip] (address) | Only hexadecimal addresses are allowed to follow the [skip] tag. Any register whose address overlaps with this address will be placed into a group labeled “skip” and generate into a \_SKIP.h file | [skip] DEADBEEF |
| use\_module\_names = <True/False> | Setting this argument to true will ignore all [groups] arguments and use the module column from the Excel file as register grouping names. | use\_module\_names = True |
| gen\_address\_macros = <True/False> | Setting this argument to true will generate macros for each register mapping their address to the register name followed by “\_ADDR” | gen\_address\_macros = True |

Table 2: configuration file entries

## Execution

The script can be executed by issuing this command from the command line:

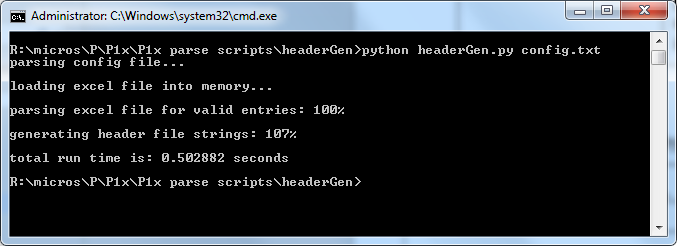
|  |
| --- |
| > python headerGen.py config.txt |



**Note:** this assumes (1) the python executable has been added to your system path, and (2) the headerGen,py script and config file are resident in your current directory.

The script will print several informative messages to the command line as it runs. Because the Excel file for a typical micro is very large (>50,000 lines), the script can take up to 10 minutes to execute. It will print percentage complete status messages during long running sections.

You will see these messages when the command line script has completed:



## Output



Figure 1: header file contents and usage

# Revision Record

|  |  |  |  |
| --- | --- | --- | --- |
| Rev. | Date | Description | |
| Page | Summary |
| 1.0 | 2/8/2016 | All | Initial Draft |
| 1.1 | 2/25/2016 | Multiple | Adding support for multiple groups, marking output file directory config option as unsupported. |
| 1.2 | 4/20/2016 | Multiple | Adding support for register address macro generation as well as register grouping based on module name column from Excel |

# Testing

Due to the size of the input Excel file, it is not feasible for Renesas or a customer (user) of the script to test each of the registers generated. Therefore, we have identified a sub-set of registers that represent interesting register layout permutations for testing. As long as these registers have generated fine, we can assume that all other register, being of identical pattern to these registers, are fine as well.

The registers in the test suite are:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Number** | **Register size** | **Base access type** | **Bit access type** | **P1M example** |
| 1 | 8-bit | 8-bit | 8-bit | 15.3.10 – SCI30BRR |
|  | | | | |
| **NOTE:** SCI30BRR and SCI30MDDR are allocated to the same address, so the script will parse this incorrectly. | | | | |
|  | | | | |
| 2 | 8-bit | 8-bit | < 8-bit, single | 10.4.2 – CVMF |
|  | | | | |
|  | | | | |
| 3 | 8-bit | 8-bit | < 8-bit, multiple | 16.3.2.1 – RLN30LWBR |
|  | | | | |
|  | | | | |
| 3a | 8-bit | 8-bit, 1-bit | < 8-bit, single | 14.3.2 – CSIH0CTL0 |
|  | | | | |
|  | | | | |
| 4 | 16-bit | 16-bit | 16-bit | 13.3.10 – CSIG0TX0H |
|  | | | | |
|  | | | | |
| 5 | 16-bit | 16-bit | 8-bit | 16.3.3.18 – RLN30LUTDR |
|  | | | | |
|  | | | | |
| 6 | 16-bit | 16-bit | < 8-bit, single | 13.3.6 – CSIG0STCR0 |
|  | | | | |
|  | | | | |
| 7 | 16-bit | 16-bit | < 8-bit, multiple, cross byte boundaries | 13.3.4 – CSIG0CTL2 |
|  | | | | |
|  | | | | |
| 8 | 16-bit | 16-bit, 8-bit | 16-bit |  |
|  | | | | |
| 9 | 16-bit | 16-bit, 8-bit | 8-bit |  |
|  | | | | |
| 10 | 16-bit | 16-bit, 8-bit | < 8-bit, single |  |
|  | | | | |
| 11 | 16-bit | 16-bit, 8-bit | < 8-bit, multiple |  |
|  | | | | |
| 12 | 32-bit | 32-bit | 32-bit | 19.3.2 – RSENTT0TSC |
|  | | | | |
|  | | | | |
| 13 | 32-bit | 32-bit | 16-bit | 13.3.9 – CSIG0TX0W |
|  | | | | |
|  | | | | |
| 14 | 32-bit | 32-bit | 8-bit | 14.3.5 – CSIH0STR0 |
|  | | | | |
|  | | | | |
| 15 | 32-bit | 32-bit | < 8-bit, single | 7.9.2.1 – DMACTL |
|  | | | | |
|  | | | | |
| 15b | 32-bit | 32-bit | < 8-bit, single, all 32-bits | 7.9.2.13 – DTSPR0 |
|  | | | | |
| **Note:** Excel and .pdf UM don’t match exactly, Excel shows these as 2-bit entries, while UM shows them as single bit entries that are clearly paired via name. | | | | |
|  | | | | |
| 16 | 32-bit | 32-bit | < 8-bit, multiple | 7.9.2.9 – DM0CMV |
|  | | | | |
|  | | | | |
| 16b | 32-bit | 32-bit | < 8-bit, multiple, crossing byte boundaries | 11.3.5 – TSNREFD |
|  | | | | |
|  | | | | |
| 17 | 32-bit | 32-bit, 16-bit | 32-bit |  |
|  | | | | |
| 18 | 32-bit | 32-bit, 16-bit | 16-bit |  |
|  | | | | |
| 19 | 32-bit | 32-bit, 16-bit | 8-bit |  |
|  | | | | |
| 20 | 32-bit | 32-bit, 16-bit | < 8-bit, single |  |
|  | | | | |
| 21 | 32-bit | 32-bit, 16-bit | < 8-bit, multiple |  |
|  | | | | |
| 22 | 32-bit | 32-bit, 16-bit, 8-bit | 32-bit | 18.2.7.12 – FLXA0FRNMV1 |
|  | | | | |
|  | | | | |
| 23 | 32-bit | 32-bit, 16-bit, 8-bit | 16-bit | 18.2.6.2 – FLXA0FRSUCC2 |
|  | | | | |
|  | | | | |
| 24 | 32-bit | 32-bit, 16-bit, 8-bit | 8-bit | 18.2.3.1 – FLXA0FRLCK |
|  | | | | |
|  | | | | |
| 25 | 32-bit | 32-bit, 16-bit, 8-bit | < 8-bit, single | 18.2.2.1 – FLXA0FROC |
|  | | | | |
|  | | | | |
| 26 | 32-bit | 32-bit, 16-bit, 8-bit | < 8-bit, multiple, crossing byte boundaries | 18.2.5.1 – FLXA0FRT0C |
|  | | | | |
|  | | | | |
| 27 | Multiple | 32-bit, 16-bit, 8-bit |  | 17.3.3 – RSCAN0C0CTR |
|  | | | | |
| 28 | Multiple | 16-bit, 8-bit |  | 16.3.3.2 – RLN30LBRP01 |
|  | | | | |
| **Note:** The Excel file shows this split into two 8-bit named entries: LBRP0 and LBRP1, while the UM shows a single entry. | | | | |