Using the NEMA™| SHADER-Edit

A Comprehensive Overview

Version 1.0

May 18, 2017

History

Version	Date	Description
1.0	10-5-2017	Initial Version

Disclaimer

Questions or comments may be directed to:
Think Silicon S.A
Suite B8
Patras Science Park
Rion Achaias 26504
Greece
web: http://www.think-silicon.com
email:info@think-silicon.com
Tel: +30 2610 911543
Fax: +30 2610 911544

Contents

List of Figures

1 Overview

NEMA™| SHADER-Edit is a graphical user interface (GUI) for the NEMA|tS Toolchain. It is an easy-to-use editor with an integrated compiler. The editor can be used for compiling vertex and fragment shaders written in OpenGL® Shading Language (GLSL). NEMA™| SHADER-Edit allows GLSL programmers to easily build and debug the input shaders without the need for a deeper knowledge of the internal operation of NEMA|tS toolchain. NEMA™| SHADER-Edit checks if a shader can be build for a specific target (e.g., NEMA|t, NEMA|S, or any Nema vertex processor) and informs the programmer in case of a conflict (e.g., not all input shaders can be built for NEMA|t due to its limited instruction set architecture). It also offers statistics for VLIW utilization, load/store instructions, and the number of required execution cycles.

The NEMA™| SHADER-Edit is user-friendly and allows ease generation of Nema executables. The editor also includes disassembly functionality that can be used for debugging purposes.

Given an input fragment or vertex shader, NEMA™| SHADER-Edit is able to generate and visualize the following information:

- SPIR-V IR under Vulcan Semantics
- LLVM IR
- C++ with Nema intrinsic functions
- Nema assembly
- ELF objects and executable
- Nema disasssembly
- Various code-level statistics

1.1 Limitations

NEMA™| SHADER-Edit as a standalone tool has no limitations, however it inherits the limitations posed by the current development phase of the NEMA|tS Toolchain.

- Only GLSL version 400 is supported
- Nema GLM is not yet supported
- Generation of binaries for compute, tessellation, and geometry shaders is not supported

1.2 Setup

NEMA™| SHADER-Edit is installed and tested in two Linux based platforms: Ubuntu 14.04 and 16.04. There is currently no version for Windows OS. After installation, NEMA™| SHADER-Edit will be located in the following path:

```
NemaTS/Toolchain/shader_editor/
```

The user can then run the executable:

```
./shader_editor
```

The interface of the NEMA™ | SHADER-Edit is shown in Figure 1:

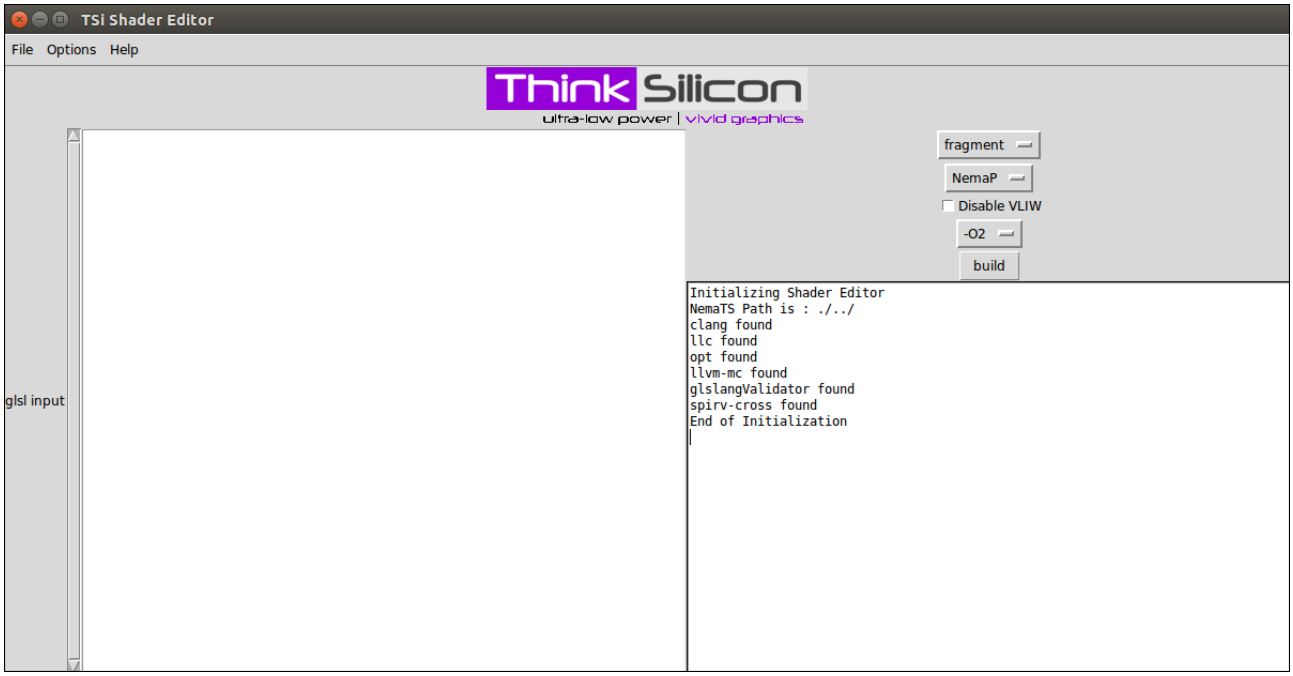

Figure 1: NEMA™ | SHADER-Edit GUI

In case that the toolchain path is different, the user can change the path by selecting in the Options menu the "`Set Toolchain Path`" option. Then the new relative or absolute path can be inserted in the template as shown in Figure 2:

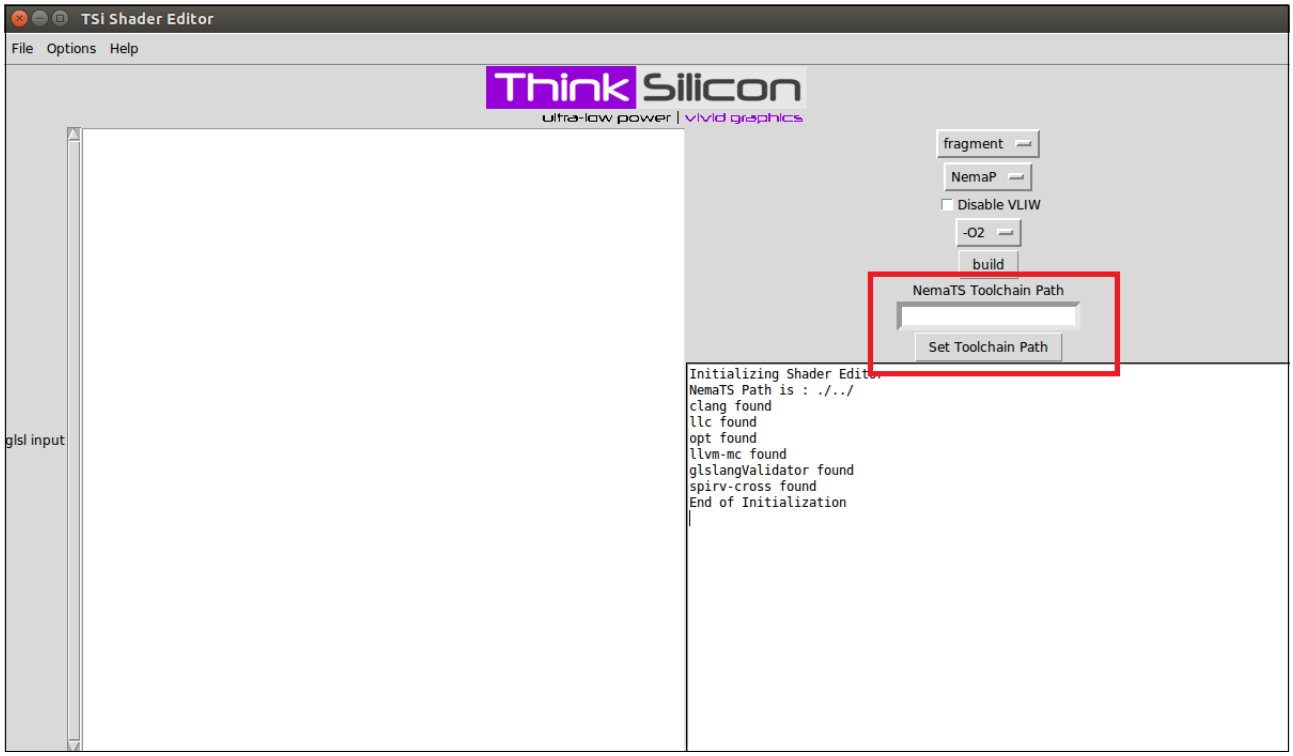

Figure 2: Set Toolchain Path

If the path is not correct, an error message will appear "`Abort setting Toolchain Path`".

2 Graphical Interface

2.1 User Interface Layout

The main interface of NEMA™ | SHADER-Edit consists of two main parts as illustrated in Figure 1. These sections are:

- Editor: This is the main area located on the left side of the GUI
- Debug Panel: This area located on the right side of the GUI shows the compiler settings and extra information about the shader file.

When the compilation of shader is successfully completed, dedicated pop-up windows will appear showing the Nema Assembly code and the Nema C++ code that can be used for debug purposes.

2.2 Menu Bar

Selecting the `File` option from the `Menu Bar` opens up the `File` menu. The listed options can be used to manage the source code files as well as to exit the GUI application. In particular, the following options are available:

- `New`: Creates a new text file in the editor
- `Open`: Opens existing files for editing
- `Save`: Saves a file
- `Save as`: Saves a file under a new name
- `Exit`: Quits the GUI application.

Selecting `Options` on the `Menu Bar` opens up the `Option` menu. The "`Set Toolchain Path`" option allows the user to change the Toolchain path as described in the previous section.

Finally, the `Help` menu offers the following options:

- `About`: View the Disclaimer and License information.
- `Version`: View basic information about NEMA™| SHADER-Edit release information such as the current version and the update history.

2.3 Edit and Compile a Shader Program

An example of a shader program is shown in Figure 3.

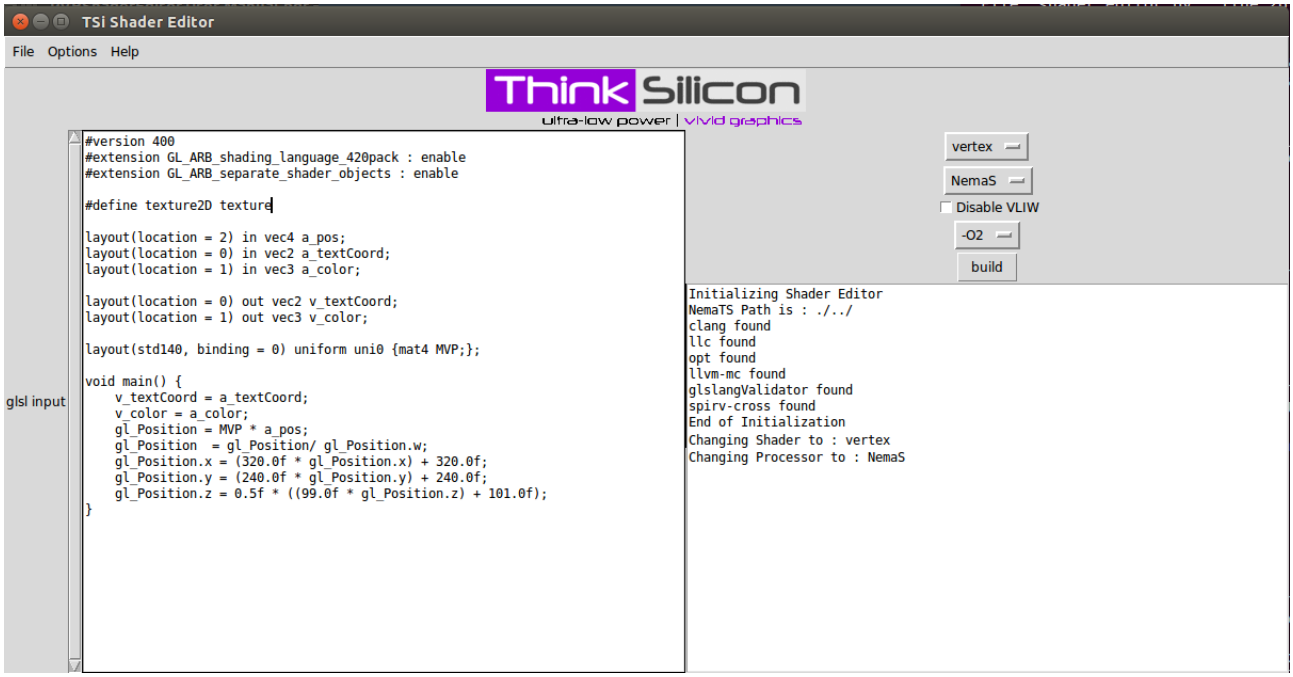

Figure 3: Editing a Shader program

The following commands must be included at the beginning of each program:

```
#version 400
#extension_GL_ARB_shading_language_420pack : enable
#extension_GL_ARB_seperate_shader_objects : enable
#define texture2D texture
```

As noted, NEMA™| SHADER-Edit currently supports Vertex and Fragment shaders. The Nema target and the shader type must be selected before compiling the program. The user can set up the configuration constraints from the drop-down menus on the right side of the GUI as shown in Figure 3. Depending on the selected Nema the following configuration options are available:

- NEMA|P: Fragment shaders only
- NEMA|T: Fragment shaders only
- NEMA|S: Fragment and Vertex shaders

If the user selects a shader type that is not supported by the selected Nema target, an error will appear.

The other configuration constraints are to enable/disable the VLIW packaging of the instructions and to set the optimization level (by default is set to 2 (`-O2`) as shown in Figure 3).

When the shader program is ready and the configuration is setup properly, the user can compile the program by pressing the `build` button as shown in Figure 4

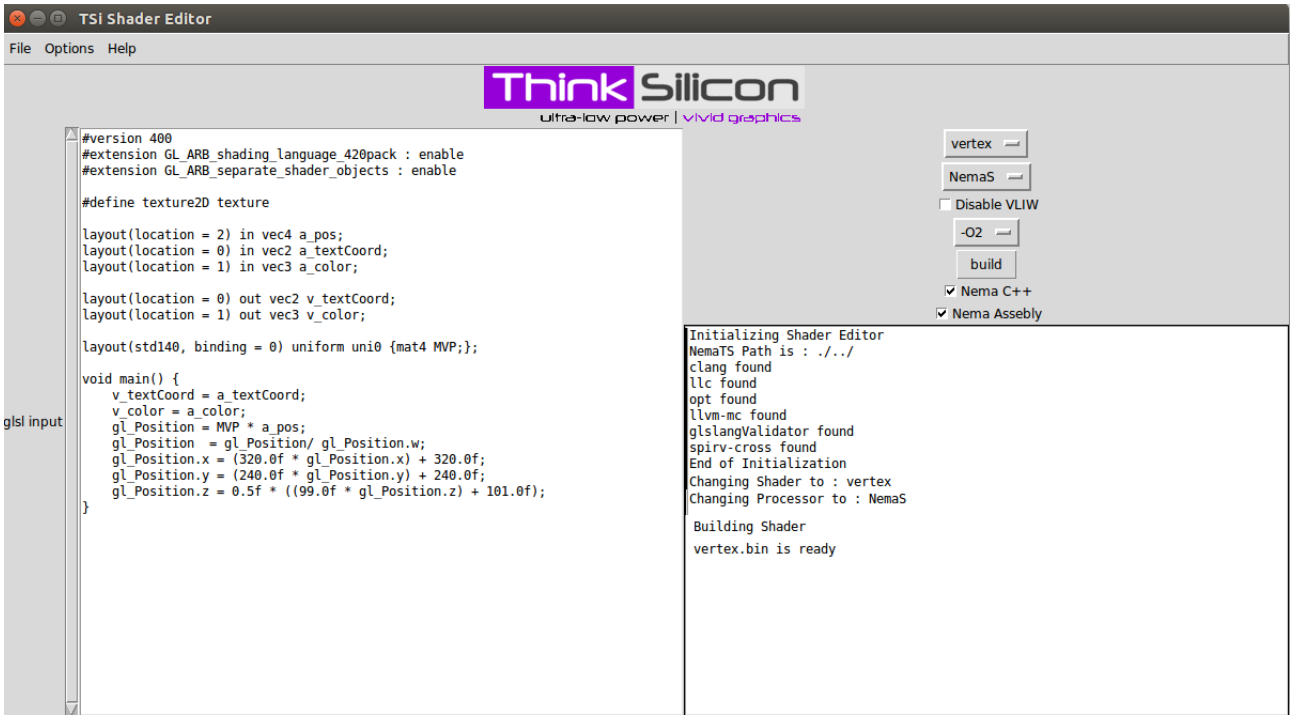

Figure 4: Building a Shader program

A message "`Building Shader` " will appear during the compilation process. When the shader is successfully built, an executable file vertex.bin or fragment.bin (depending on the shader type) will be produced. An appropriate message will appear when the compilation process is successfully complete ("`vertex.bin is ready`" or "`fragment.bin is ready`". Figure 4 shows an example of building a Vertex shader programm for NEMA|s.

After successfully building a shader program, two extra options are available, the Nema C++ and the Nema Assembly as shown in Figure 4. By ticking these options pop-up windows that show the C++ and the Assembly code will show. This information can be used for debbuging purposes. Figure 5 depicts generated Nema C++ and Figure 6 depicts the corresponding assembly code.

Generated Nema C++ from SPIR-V

```
// This C++ Nema shader is autogenerated by spirv_cross.
#include "glm/glm.hpp"
#include "glsl_defines.h"
#include "glsl_lib.h"
#include "glsl_enum.h"
#include "nema_programHW_intern.h"
#include "array"
using namespace glm;

//struct Shader



vec2 a_textCoord; //StageInput0
vec3 a_color; //StageInput1
vec4 a_pos; //StageInput2

void main_shader(uint32_t rast_indx)
{
    //declare variables
    struct gl_PerVertex
    {
        vec4 gl_Position;
        float gl_PointSize;
```

Figure 5: Generated Nema C++

Generated Nema Assembly

```
        .text
        .abicalls
        .option pic0
        .section .mdebug.abi32,"",@progbits
        .nan    legacy
        .file   "tmpxyz.ll"
        .text
        .globl  _Z11main_shaderj
        .align  2
        .type   _Z11main_shaderj,@function
        .set    nomicronemats
        .set    nonemats16
        .ent    _Z11main_shaderj
_Z11main_shaderj:                       # @_Z11main_shaderj
        .frame  $29,40,$31
        .mask   0x40ff0000,-4
        .fmask  0x00000000,0
        .set    noreorder
        .set    nomacro
        .set    noat
# BB#0:                                 # %entry
        addiu   $29, $29, 0xffffffd8
        sw      $30, 0x24($29)          # 4-byte Folded Spill
        sw      $23, 0x20($29)          # 4-byte Folded Spill
```

Figure 6: Nema Assembly