Nuclei™

Processor Integration Guide

Copyright Notice

The product described herein is subject to continuous development and improvement; information herein is given by Nuclei in good faith but without warranties.

This document is intended only to assist the reader in the use of the product. Nuclei System Technology shall not be liable for any loss or damage arising from the use of any information in this document, or any incorrect use of the product.

Contact Information

Should you have any problems with the information contained herein or any suggestions, please contact Nuclei System Technology by email support@nucleisys.com, or visit “Nuclei User Center” website http://user.nucleisys.com for supports or online discussion.

Revision History

Rev.	Revision Date	Revised Section	Revised Content
1.0.0	2020/4/1	N/A	1. First version as the full English
1.1.0	202/7/20	5.5	1. Add JTAG_VPI Description
1.2.0	2020/8/28	1, 5	1. Update the RTL generation flow

Table of Contents

List of Tables

List of Figures

1. Introduction of Release Package

1.1. Release Package

The Nuclei processor is released as a package, all the Nuclei processors (N/NX/UX-200/600/600/900 series) will keep consistent with the n600 flow shown in this Integration guide.

Using n600 as example shown in Table 1-1.

Table 1-1 Release Packages

Package Name	Description
n600_rls_pkg.tar.gz	Including the Verilog RTL source codes, Core generation tool, Evaluation SoC, Simulation Environment, Logic Synthesis and FPGA project.

The release package of N600 Series Core can be licensed from Nuclei. After got the release package, user can use the following command to decompress.

```
tar -xzvf n600_rls_pkg.tar.gz
```

1.2. Files in Package

The files in the package are introduced as below.

```
n600_rls_pkg
    | env.csh        // csh environment script
    | env.sh         // bash environment script
    | libc.so.6      // The libc dependency file
    | n600.iplib     // The IP Library for Core RTL generation
    | private.pem    // Private Key for nuclei_gen (need to contact Nuclei)
    | nuclei_gen     // Core RTL configure and generation tool
                     // Please refer Chapter-4 for more details

After using 'nuclei_gen' to configure and generate the Core RTL, a new directory
```

'n600' will be generated, if using 'nuclei_gen' to reconfigure and generate a new version of Core RTL, then the previous directory 'n600' will be moved to be 'n600.bak' and 'n600' is updated to the new generated one.

```
|n600
    |----design      // Directory for RTL
        |----core           // Directory for Core
        |----soc            // Directory for bus-fab, subsystem,
                            // memory and peripherals in SoC
    |----riscv-tests // Directory for testcases
    |----tb                // Directory for Verilog TestBench
    |----vsim              // Directory for Simulation
                           // Please see more details from Chapter 5
        |----bin                // Directory for functional scripts
        |----Makefile           // Makefile for simulation
        |----run                // Directory to run
    |----fpga              // Directory for FPGA project
                           // Please see more details from Chapter 8
    |----syn              // Directory for Synthesis project
                          // Please see more details from Chapter 7
```

Note:

- The above "n600_" is just a general prefix, for the specific core, such as N607, will use the specific prefix "n607_".

1.3. Naming Rule of Core

The source code of N600 Series Cores have different prefix for the files and modules, for example, if it is N607 Core, then the files and modules have the prefix "n607_". The same naming rules applied to other Cores like N605, N608, etc.

1.4. Module Hierarchy of Core

Take N607 as the example, as depicted in Figure 1-1, the key points are:

- n607_core_wrapper is the top module of the Core, which include several key sub-modules:
  - n607_core: The Core part.
  - n607_rst_ctrl: The module to sync external async reset signal to synced reset with "Asynchronously assert and synchronously de-assert" style.
  - n607_dbg_top: The module to handle the debug functionalities.

- n607_ucore is under Core hierarchy, it is the main part of Core.
- Besides the n607_ucore, there are several other sub-modules:
  - n607_clic_top: The private interrupt controller.
  - n607_tmr_top: The private timer unit.
  - n607_clk_ctrl: The clock control module.

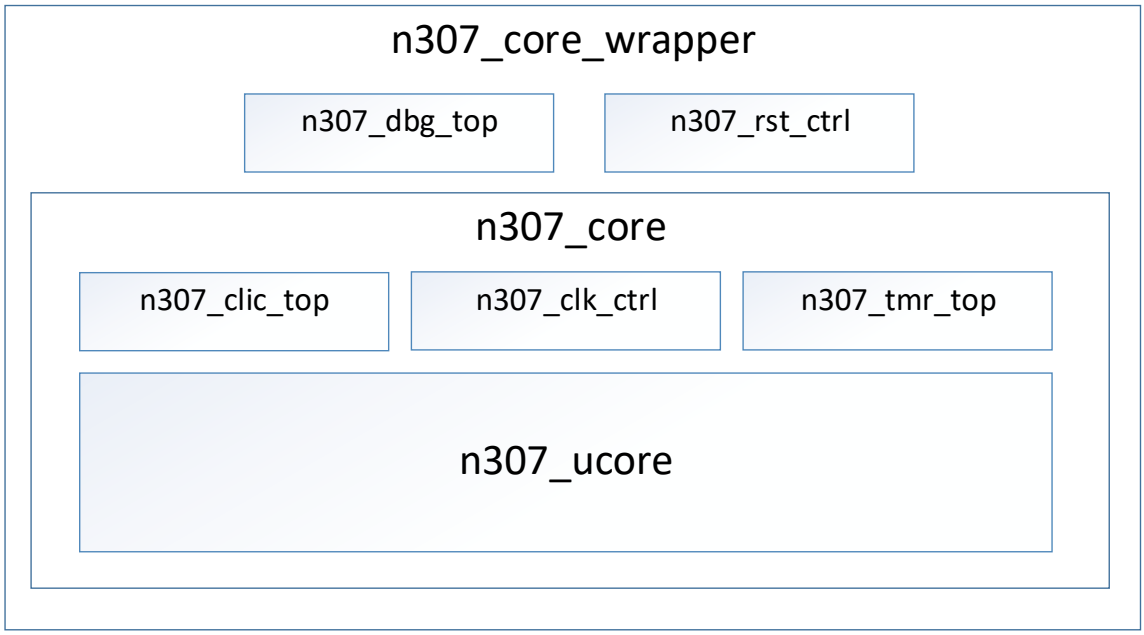


Figure 1-1 Module Hierarchy of N607 Core

2. Top Level Integration

2.1. Clocks

Clocks to the Core are the baseline of the top level integration.

For the details of the N600 Series Cores' clocks, please refer to Section "Clock Domains" of the document <Nuclei_N600_Databook.pdf>, which can be easily got from "Nuclei User Center" website http://user.nucleisys.com.

2.2. Interfaces

The interfaces of Core need to be carefully checked during the top level integration.

For the details of the N600 Series Cores' interfaces, please refer to Chapter "Core Interfaces" of the document <Nuclei_N600_Databook.pdf>, which can be easily got from "Nuclei User Center" website http://user.nucleisys.com.

2.3. Memory Map

There are quite several interfaces and private peripherals for the N600 Series Core, the address spaces of them are mostly configurable, hence the SoC integrator can determine the address memory map per the SoC requirements.

For the details of the N600 Series Cores' clocks, please refer to Section "Address Spaces of Interfaces and Private Peripherals" of the document <Nuclei_N600_Databook.pdf>, which can be easily got from "Nuclei User Center" website http://user.nucleisys.com.

3. SoC, FPGA, SDK and IDE for Evaluation

3.1. Nuclei Evaluation SoC (Hummingbird SoC)

To easy user to evaluate Nuclei Processor Core, the prototype SoC (called Hummingbird SoC) is provided for evaluation purpose. As depicted in Figure 3-1, this prototype SoC includes:

- Processor Core, it can be Nuclei N class, NX class or UX class Processor Core.
- On-Chip SRAMs for instruction and data.
- The SoC buses.
- The basic peripherals, such as UART, GPIO, SPI, I2C, etc.

With this prototype SoC, user can run simulations, map it into the FPGA board, and run with real embedded application examples.

For the details of the Nuclei Evaluation SoC (Hummingbird SoC), please refer to the document <Nuclei_Eval_SoC_Intro.pdf>, which can be easily got from “Nuclei User Center” website http://user.nucleisys.com.

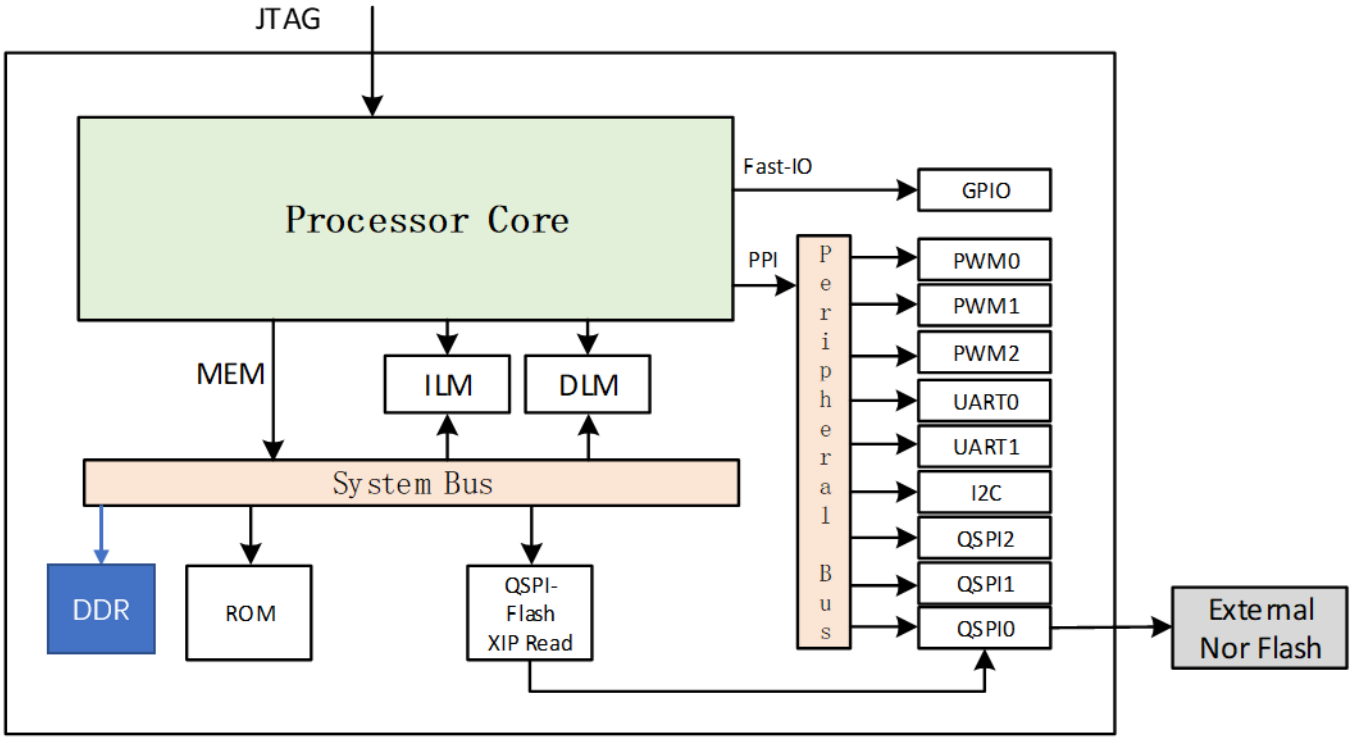


Figure 3-1 Nuclei Evaluation SoC (Hummingbird SoC)

Note: DDR will only exist in UX class cores.

3.2. FPGA Evaluation Board and JTAG Debugger

Nuclei have customized a FPGA evaluation board, called Hummingbird Evaluation Kit. This FPGA board can be used as the SoC prototype board directly:

- If the FPGA have been pre-burned (programmed) with "Nuclei evaluation SoC", this board can be worked as a SoC prototype directly. Since the board has been designed with buttons and extended ports names in line with the SoC GPIO pin name, the embedded software engineers can directly use this board without knowing any FPGA hardware knowledge.
- About how to generate the FPGA Bitstream (MCS) with pre-built FPGA project, please refer to Chapter 8.

Nuclei have customized a Debugger hardware (called Hummingbird Debugger Kit), which can be used to debug the RISC-V core in FPGA prototype or in real chip.

For the detailed introduction of the “Hummingbird Evaluation Kit” and “Hummingbird Debugger Kit”, please refer to the document <Nuclei_FPGA_DebugKit_Intro.pdf> which can be downloaded from “Development Boards” page of Nuclei website (http://www.nucleisys.com/developboard.php).

3.3. Software Development Kit (SDK)

Nuclei have created a “Nuclei Software Development Kit (Nuclei-SDK)” which is an open software platform to facilitate the software development for systems based on Nuclei Processor Cores. For more details about Nuclei-SDK, please see its online doc from http://doc.nucleisys.com/nuclei_sdk.

Based on the “Nuclei Evaluation SoC”, and with the demo software projects from Nuclei-SDK, user can quickly familiarize the software development for Nuclei Processor Cores.

3.4. Integrated Development Environment (IDE)

The SES (Segger Embedded Studio) is a professional and excellent IDE (Integrated Development Environment), which support the standard GCC toolchain, have the best-in-class debugging functionalities with famous Segger J-Link. It also supports to debug with the Hummingbird Debugger Kit.

Nuclei processor core can be fully supported by Embedded Studio and the J-Link.

For the quick-start introduction of SES for Nuclei Processor Cores, please refer to document <Nuclei_SES_IDE_QuickStart.pdf>, which can be easily got from “Nuclei User Center” website http://user.nucleisys.com.

4. Configure to Generate RTL

4.1. Use nuclei_gen Tool to generate RTL Codes

Since Nuclei N600 Series Core is fully configurable, Nuclei developed a tool called *nuclei_gen*. User can easily configure the Core according to their requirements at their field, and then generate the RTL code.

Under the n600_rls_pkg directory, there are files as below:

- nuclei_gen: The Core RTL configuration and generation tool.
- private.pem: The private Key to use nuclei_gen, need to contact Nuclei to get this.
- n600.iplib: The IP library for Core RTL generation.
- env.sh: Shell environment script.
- env.csh: Environment checking script .
- libc.so.6: The libc dependency file.

Note:

- Don't change the files "private.pem" and "libc.so.6", otherwise there might be errors when generating the RTL code.

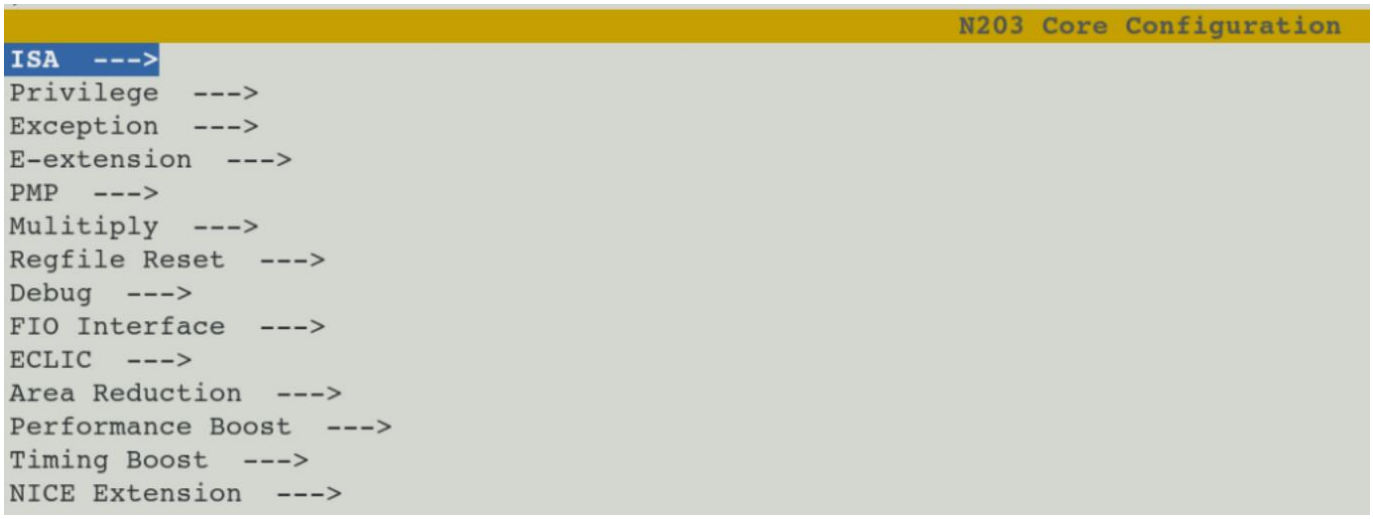
Before starting the nuclei_gen tool, there are several environment variables need to be set:

- bash environment: source env.sh
- csh environment: source env.csh

The above script will set the following environment variables:

- PROJ_SRC_ROOT: The directory of n600_rls_pkg
- PROJ_NAME: The Core's name.
- PROJ_GEN_ROOT: RTL source code directory, by default it is n600_rls_pkg/n600. If user wants to generate RTL code to other directory, user can change this variable.

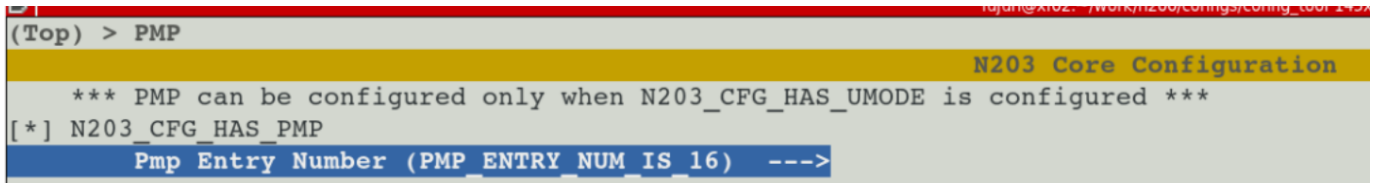
After setting environment correctly, user can directly execute "./nuclei_gen", it will launch the nuclei_gen tool, the pop Window is shown as in Figure 4-1. The configurable options shown in the Window are also explained in document <Nuclei_N600_Databook.pdf>, which can be easily got from "Nuclei User Center" website http://user.nucleisys.com.



In above figure, the special string post each option is explained as below:

- If there is " --->", then indicate there are sub-menu for this option, user can enter "Space" or "Enter" key, to enter sub-menu.
- If entered the sub-menu, user can enter the "<" key, to return to previous upper menu.

For example, if user enter "PMP" sub-menu, it is as shown in Figure 4-2.



In above figure, the special characters along with options are explained as below:

- [*] Indicating this option has been chose by user. If user enter the "Space" key, then

discard choosing this option.

- [ ] Indicating this option has not been chose by user. If user enter the “Space” key, then choose this option.
- -*- Indicating this option is fixed, i.e., not configurable.
- The value in () indicating the value of this configuration. If there is a (NEW), means it is default value, and if user configured different value, then this (NEW) will be disappeared.

Continue the above example, if enter “Pmp Entry Number” sub-menu, it is as shown in Figure 4-3. In this sub-menu, use “SPACE” key to choose the option you want.

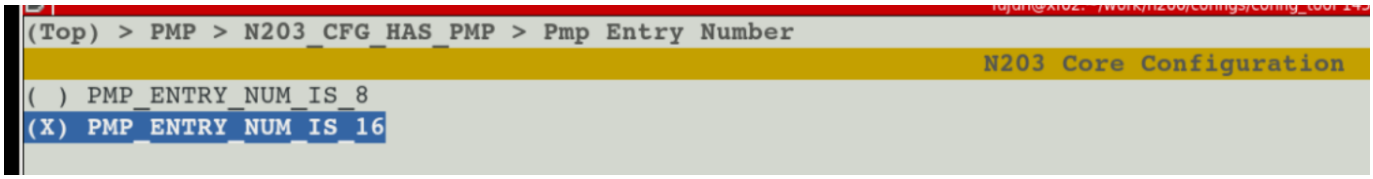


Figure 4-3 PMP Entry Number Menu (picture from N203 IPlib just as an example)

There might be some options need to be inputted with values. For example, the ECLIC Base Address as shown in Figure 4-4. In this option, enter the “Enter” or “Space” key, the configuration input window will be shown, as in Figure 4-5.

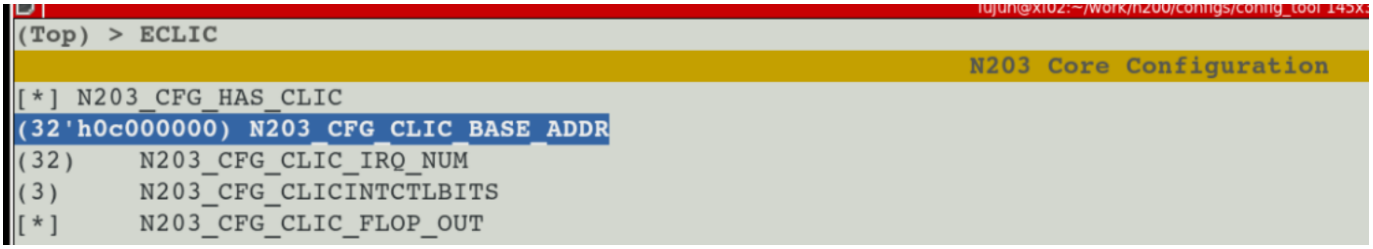


Figure 4-4 ECLIC Base Address Configuration (picture from N203 IPlib just as an example)

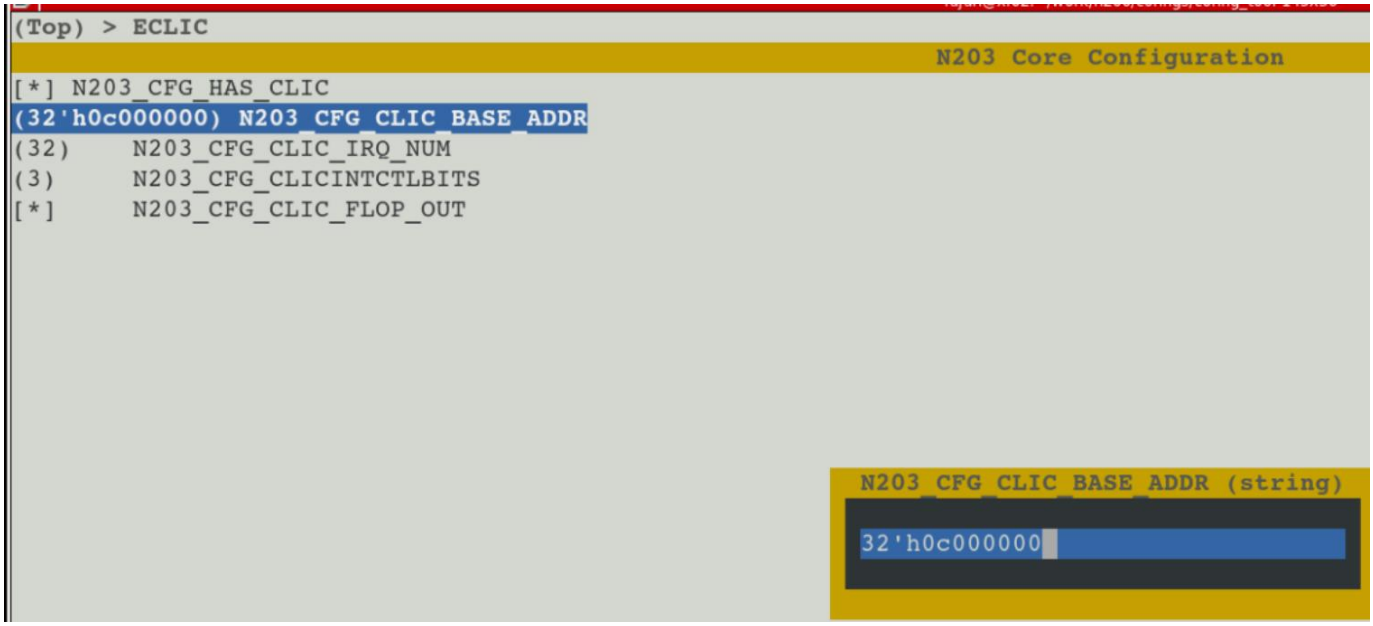


Figure 4-5 Input the ECLIC Base Address Value (picture from N203 IPlib just as an example)

There might be some options need to be inputted with values, but with constraints. For example, as shown in Figure 4-6, the range of interrupt number is constrained to 1~1005. If the inputted value is out of this range, it will be reported as "Error", as shown in Figure 4-7.

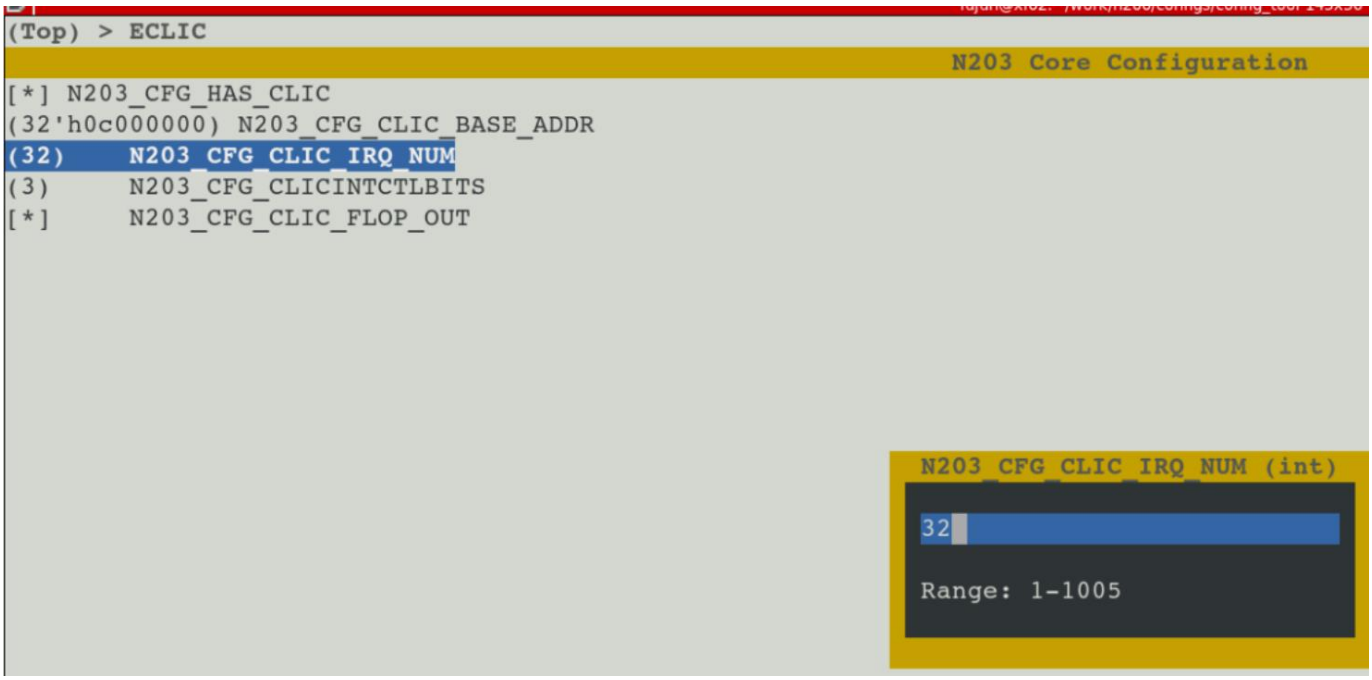


Figure 4-6 Input the ECLIC IRQ Number Value (picture from N203 IPlib just as an example)

Figure 4-7 The Configuration Error (picture from N203 IPlib just as an example)

After finished configuration, input the letter "q", save and exit. After exited, the nuclei_gen tool will start to generate the RTL codes. It will take several minutes to generate out the codes, user need to wait with patience. The generated code will be under directory of pointed by environment variable $PROJ_GEN_ROOT.

Note:

- The generated codes under $PROJ_GEN_ROOT contain lots of codes, including the Core's codes, and the SoC's codes. If user only needs the Core's codes, just check the code under directory of "core".
- There will be a file ".config" generated under current directory. When the nuclei_gen tool is re-opened next time, it will directly use the configuration from ".config". If this ".config" file is deleted, then the core_gen tool will use its inherent default configurations.

4.2. Check and Compile the Verilog RTL

If user wants check or compile the generated RTL code, the steps are detailed as below (take N607 as example).

// Note: Before operation, it is required to install the "RISC-V GNU Toolchain". The toolchain can be downloaded from Nuclei website (https://www.nucleisys.com/download.php).

// After the "RISC-V GNU Toolchain" package downloaded and decompressed, there will be a "bin" directory under GCC folder. User need to add this "bin" path into the Linux $PATH environment variable.

// Step 1: Use nuclei_gen to configure and generate the RTL code. Use the following commands:

```
cd n607_rls_pkg
source env.sh
./nuclei_gen
   // The detailed way to configured and generate code is described in Section 4.1.
   // The Core's RTL code is generated under n607_rls_pkg/n607
```

// Step 2: Compile the RTL, use the following commands:

```
cd n607_rls_pkg/n607/vsim

make install

make compile
    // Compile the RTL
```

// Step 3: Check the RTL codes, use the following commands:

```
make verilog
    // This command will open all of the Verilog codes, including the Testbench
and Verilog source codes (for entire SoC and Core)

make verilog_core
    // This command will open only the Core's Verilog RTL codes
```

5. Simulation with Simple Assembly Testcase

5.1. Overview of Self-Check Testcase

The “Self-Check Testcase” is a kind of assembly Testcase which can self-check if it is “passed” or “failed”. The Self-Check Testcase are under the following directory.

```
n600_rls_pkg
    |----n600
        |----riscv-tests
                |----isa_origs    // The directory for the source codes of
                                  // Self-Check Testcases.
```

The “Self-Check Testcase” will set some “expected value” at the check-point, if the “real result” is not as the expected, then it will jump to the label of TEST_FAIL, otherwise it will continue to run until it reach the final ending label of TEST_PASS.

For example, as shown in Figure 5-1, the Testcase (source code under isa_origs/rv64ui/add.S) is to test the “add” instruction to compute two operands’ addition (e.g., 0x00000003 and 0x00000007) , and then set its expected value (e.g., 0x0000000a). And then use the “compare” instruction to compare the “real result” is as expected or not, if not matched, then the test will jump to TEST_FAIL.

At the label of TEST_PASS, the test will set the value of general register X3 to 1; while at the label of TEST_FAIL, the test will set the value of general register X3 to “not 1”. Hence, the testbench can monitor the final X3 value to check the Testcase is passed or failed.

```
RVTEST_CODE_BEGIN

  #-------------------------------------------------------------
  # Arithmetic tests
  #-------------------------------------------------------------

  TEST_RR_OP( 2,  add, 0x00000000, 0x00000000, 0x00000000 );
  TEST_RR_OP( 3,  add, 0x00000002, 0x00000001, 0x00000001 );
  TEST_RR_OP( 4,  add, 0x0000000a, 0x00000003, 0x00000007 );

  TEST_RR_OP( 5,  add, 0xffffffffffff8000, 0x0000000000000000, 0xffffffffffff8000 );
  TEST_RR_OP( 6,  add, 0xffffffff80000000, 0xffffffff80000000, 0x00000000 );
  TEST_RR_OP( 7,  add, 0xffffffff7fff8000, 0xffffffff80000000, 0xffffffffffff8000 );

  TEST_RR_OP( 8,  add, 0x0000000000007fff, 0x0000000000000000, 0x0000000000007fff );
  TEST_RR_OP( 9,  add, 0x000000007fffffff, 0x000000007fffffff, 0x0000000000000000 );
  TEST_RR_OP( 10, add, 0x0000000080007ffe, 0x000000007fffffff, 0x0000000000007fff );

  TEST_RR_OP( 11, add, 0xffffffff80007fff, 0xffffffff80000000, 0x0000000000007fff );
  TEST_RR_OP( 12, add, 0x000000007fff7fff, 0x000000007fffffff, 0xffffffffffff8000 );

  TEST_RR_OP( 13, add, 0xffffffffffffffff, 0x0000000000000000, 0xffffffffffffffff );
  TEST_RR_OP( 14, add, 0x0000000000000000, 0xffffffffffffffff, 0x0000000000000001 );
  TEST_RR_OP( 15, add, 0xfffffffffffffffe, 0xffffffffffffffff, 0xfffffffffffffffd );

  TEST_RR_OP( 16, add, 0x0000000080000000, 0x0000000000000001, 0x000000007fffffff );

  #-------------------------------------------------------------
  # Source/Destination tests
  #-------------------------------------------------------------

  TEST_RR_SRC1_EQ_DEST( 17, add, 24, 13, 11 );
  TEST_RR_SRC2_EQ_DEST( 18, add, 25, 14, 11 );
  TEST_RR_SRC12_EQ_DEST( 19, add, 26, 13 );
```

Figure 5-1 The code segment of add.S test

5.2. Testbench to Initialize Self-Check Testcase

In order to have the Self-Check Testcase simulated in the Verilog Testbench, it is needed to convert the Testcase into the binary file with the format which can be initialized by Verilog Testbench.

After the “make install” command as described in Section 4.2. The binary file (.verilog file) for each Testcase will be generated under riscv-tests/isa/generated directory, exampled as below.

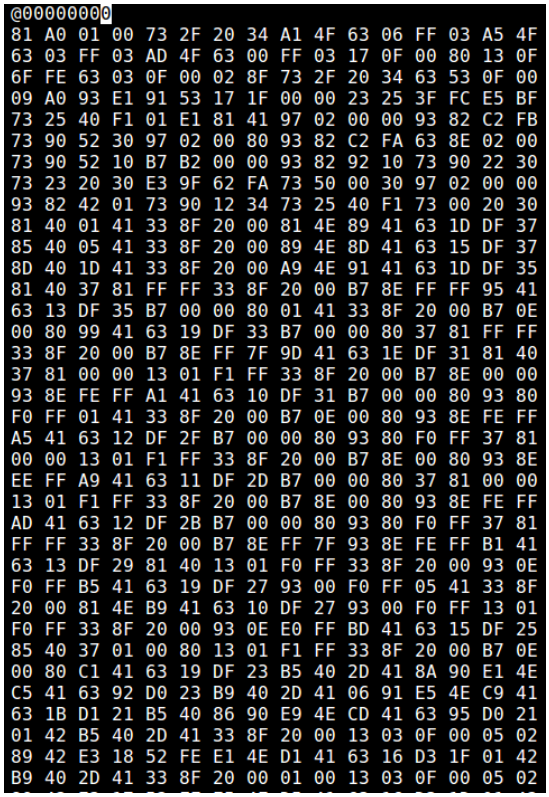
```
n600
    |----riscv-tests
        |----isa
            |----generated
                |----rv32ui-p-addi          // The generated Elf file
                |----rv32ui-p-addi.dump     // The disassembly code
                |----rv32ui-p-addi.verilog  // The format which can be read by
                                            // Verilog's readmemh function in
                                            // Testbench.

                    ……
```

The content of disassembly code (e.g., rv32ui-p-addi.dump) is as shown in Figure 5-2.

```
rv32ui-p-add:     file format elf32-littleriscv


Disassembly of section .text.init:

80000000 <_start>:
80000000:   a081                    j   80000040 <reset_vector>
80000002:   0001                    nop

80000004 <trap_vector>:
80000004:   34202f73            csrr    t5,mcause
80000008:   4fa1                    li  t6,8
8000000a:   03ff0663            beq t5,t6,80000036 <write_tohost>
8000000e:   4fa5                    li  t6,9
80000010:   03ff0363            beq t5,t6,80000036 <write_tohost>
80000014:   4fad                    li  t6,11
80000016:   03ff0063            beq t5,t6,80000036 <write_tohost>
8000001a:   80000f17            auipc   t5,0x80000
8000001e:   fe6f0f13            addi    t5,t5,-26 # 0 <_start-0x80000000>
80000022:   000f0363            beqz    t5,80000028 <trap_vector+0x24>
80000026:   8f02                    jr  t5
80000028:   34202f73            csrr    t5,mcause
8000002c:   000f5363            bgez    t5,80000032 <handle_exception>
80000030:   a009                    j   80000032 <handle_exception>

80000032 <handle_exception>:
80000032:   5391e193            ori gp,gp,1337

80000036 <write_tohost>:
80000036:   00001f17            auipc   t5,0x1
8000003a:   fc3f2523            sw  gp,-54(t5) # 80001000 <tohost>
8000003e:   bfe5                    j   80000036 <write_tohost>

80000040 <reset_vector>:
80000040:   f1402573            csrr    a0,mhartid
80000044:   e101                    bnez    a0,80000044 <reset_vector+0x4>
80000046:   4181                    li  gp,0
80000048:   00000297            auipc   t0,0x0
8000004c:   fbc28293            addi    t0,t0,-68 # 80000004 <trap_vector>
80000050:   30529073            csrw    mtvec,t0
80000054:   80000297            auipc   t0,0x80000
80000058:   fac28293            addi    t0,t0,-84 # 0 <_start-0x80000000>
8000005c:   00028e63            beqz    t0,80000078 <reset_vector+0x38>
80000060:   10529073            csrw    stvec,t0
```

Figure 5-2 The content of rv32ui-p-addi.dump file

The content of binary code (e.g., rv32ui-p-addi.verilog) is as shown in Figure 5-3, which can be read by Verilog's readmemh function in Verilog Testbench.



```
@00000000
81 A0 01 00 73 2F 20 34 A1 4F 63 06 FF 03 A5 4F
63 03 FF 03 AD 4F 63 00 FF 03 17 0F 00 80 13 0F
6F FE 63 03 0F 00 02 8F 73 2F 20 34 63 53 0F 00
09 A0 93 E1 91 53 17 1F 00 00 23 25 3F FC E5 BF
73 25 40 F1 01 E1 81 41 97 02 00 00 93 82 C2 FB
73 90 52 30 97 02 00 80 93 82 C2 FA 63 8E 02 00
73 90 52 10 B7 B2 00 00 93 82 92 10 73 90 22 30
73 23 20 30 E3 9F 62 FA 73 50 00 30 97 02 00 00
93 82 42 01 73 90 12 34 73 25 40 F1 73 00 20 30
81 40 01 41 33 8F 20 00 81 4E 89 41 63 1D DF 37
85 40 05 41 33 8F 20 00 89 4E 8D 41 63 15 DF 37
8D 40 1D 41 33 8F 20 00 A9 4E 91 41 63 1D DF 35
81 40 37 81 FF FF 33 8F 20 00 B7 8E FF FF 95 41
63 13 DF 35 B7 00 00 80 01 41 33 8F 20 00 B7 0E
00 80 99 41 63 19 DF 33 B7 00 00 80 37 81 FF FF
33 8F 20 00 B7 8E FF 7F 9D 41 63 1E DF 31 81 40
37 81 00 00 13 01 F1 FF 33 8F 20 00 B7 8E 00 00
93 8E FE FF A1 41 63 10 DF 31 B7 00 00 80 93 80
F0 FF 01 41 33 8F 20 00 B7 0E 00 80 93 8E FE FF
A5 41 63 12 DF 2F B7 00 00 80 93 80 F0 FF 37 81
00 00 13 01 F1 FF 33 8F 20 00 B7 8E 00 80 93 8E
EE FF A9 41 63 11 DF 2D B7 00 00 80 37 81 00 00
13 01 F1 FF 33 8F 20 00 B7 8E 00 80 93 8E FE FF
AD 41 63 12 DF 2B B7 00 00 80 93 80 F0 FF 37 81
FF FF 33 8F 20 00 B7 8E FF 7F 93 8E FE FF B1 41
63 13 DF 29 81 40 13 01 F0 FF 33 8F 20 00 93 0E
F0 FF B5 41 63 19 DF 27 93 00 F0 FF 05 41 33 8F
20 00 81 4E B9 41 63 10 DF 27 93 00 F0 FF 13 01
F0 FF 33 8F 20 00 93 0E E0 FF BD 41 63 15 DF 25
85 40 37 01 00 80 13 01 F1 FF 33 8F 20 00 B7 0E
00 80 C1 41 63 19 DF 23 B5 40 2D 41 8A 90 E1 4E
C5 41 63 92 D0 23 B9 40 2D 41 06 91 E5 4E C9 41
63 1B D1 21 B5 40 86 90 E9 4E CD 41 63 95 D0 21
01 42 B5 40 2D 41 33 8F 20 00 13 03 0F 00 05 02
89 42 E3 18 52 FE E1 4E D1 41 63 16 D3 1F 01 42
B9 40 2D 41 33 8F 20 00 01 00 13 03 0F 00 05 02
```

Figure 5-3 The content of rv32ui-p-addi.verilog file

5.3. Introduction of Testbench

The Verilog Testbench source codes are under the “tb” directory as below.

```
n600
    |----tb
        |----tb_*.v          //Verilog TestBench source codes
```

The functionality of Testbench is briefly introduced as below:

- Instantiated DUT.
- Generate the clock and reset.
- According to the run options, to recognize the Testcase name, and then use readmemh function to read the .verilog file (e.g., rv32ui-p-addi.verilog), and then initialize the instruction memory in SoC.
- At the end of the simulation, check the value of X3, if the X3 value is 1, then the test is

passed, print the “PASS” on the terminal, otherwise it is failed and printed as “FAIL”, as shown in Figure 5-4.

Note:

- User can also integrate these tb_*.v files into their SoC environment, such that in the user’s SoC, the Testcase can also be as the sanity Testcases.
- However, these above Testcases are very basic tests, which cannot guarantee the full coverage. If users have modified the Core’s RTL code, should not assume the functional correctness can be verified by running the above Testcases.

```
@(pc_write_to_host_cnt == 32'd8)

    $display("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");
    $display("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");
    $display("~~~~~~~~~~~~~~ Test Result Summary ~~~~~~~~~~~~~~~~~~~~~");
    $display("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");
    $display("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");
    $display("~TESTCASE: %s ~~~~~~~~~~~~~~", testcase);
    $display("~~~~~~~~~~~~~~~Total cycle_count value: %d ~~~~~~~~~~~~~~", cycle_count);
    $display("~~~~~~~~~~~The valid Instruction Count: %d ~~~~~~~~~~~~~~", valid_ir_cycle);
    $display("~~~~~The test ending reached at cycle: %d ~~~~~~~~~~~~~~", pc_write_to_host_cycle);
    $display("~~~~~~~~~~~~~~~~~The final x3 Reg value: %d ~~~~~~~~~~~~~~", x3);
    $display("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");
if (x3 == 1) begin
    $display("~~~~~~~~~~~~~~~~~ TEST_PASS ~~~~~~~~~~~~~~~~~~~~~~~~~~~~");
    $display("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");
    $display("~~~~~~~~~ #####     ##     ####    #### ~~~~~~~~~~~~~~~~");
    $display("~~~~~~~~~ #    #   #  #   #       #     ~~~~~~~~~~~~~~~~");
    $display("~~~~~~~~~ #    #  #    #   ####    #### ~~~~~~~~~~~~~~~~");
    $display("~~~~~~~~~ #####   ######       #       #~~~~~~~~~~~~~~~~");
    $display("~~~~~~~~~ #       #    #  #    #  #    #~~~~~~~~~~~~~~~~");
    $display("~~~~~~~~~ #       #    #   ####    #### ~~~~~~~~~~~~~~~~");
    $display("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");
end
else begin
    $display("~~~~~~~~~~~~~~~~~ TEST_FAIL ~~~~~~~~~~~~~~~~~~~~~~~~~~~~");
    $display("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");
    $display("~~~~~~~~~~######    ##       #    #     ~~~~~~~~~~~~~~~~");
    $display("~~~~~~~~~~#       #  #      #    #     ~~~~~~~~~~~~~~~~");
    $display("~~~~~~~~~~#####   #    #     #    #     ~~~~~~~~~~~~~~~~");
    $display("~~~~~~~~~~#       ######     #    #     ~~~~~~~~~~~~~~~~");
    $display("~~~~~~~~~~#       #    #     #    #     ~~~~~~~~~~~~~~~~");
    $display("~~~~~~~~~~#       #    #     #    ######~~~~~~~~~~~~~~~~");
    $display("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");
end
```

Figure 5-4 Print the PASS or FAIL in Testbench

5.4. Steps to Run Simulation

The steps to run simulation are as below:

```
// Note: Before operation, it is required to install the “RISC-V GNU Toolchain”.
```

The toolchain can be downloaded from Nuclei website (https://www.nucleisys.com/download.php).

// After the "RISC-V GNU Toolchain" package downloaded and decompressed, there will be a "bin" directory under GCC folder. User need to add this "bin" path into the Linux $PATH environment variable.

// Step 1: Generate the tests.

```
cd n607_rls_pkg/n607/vsim
        // Enter into the vsim directory.

make clean
        // Clean up the directory.

make install
        // Use this command to generate the tests and Testbench.
```

// Step 2: Compile RTL.

```
make compile
        // Compile the Verilog source codes.
```

// Step 3: If want to run one single testcase, use the following commands.

```
make run_test TESTNAME=rv32ui-p-add
        // This command will run the simulation for one Testcase "rv32ui-p-add"
        // from riscv-tests/isa/generated directory.

make wave TESTNAME=rv32ui-p-add
        // This command will check the generated waveform.
```

// Step 4: If want to run the regression, use the following commands.

```
make regress_run
        // This command will run the regression for all the tests from
        // riscv-tests/isa/generated directory.

make regress_collect
        // This command will collect the simulation result for regression. It will
print a summary result, with each line for each testcase. For each line, if the
Testcase is passed then marked as "PASS", otherwise as "FAIL".
```

5.5. Introduction of JTAG_VPI

The Verilog Testbench source codes are under the "tb" directory as below.

```
n600
    |----tb
        |----jtag_vpi       //jtag vpi source codes
```

The JTAG_VPI module is used to test the DEBUG module in simulation, which can simulate the GDB feature without needing the FPGA environment, the waveform can also be dumped.

Below is the connection diagram:

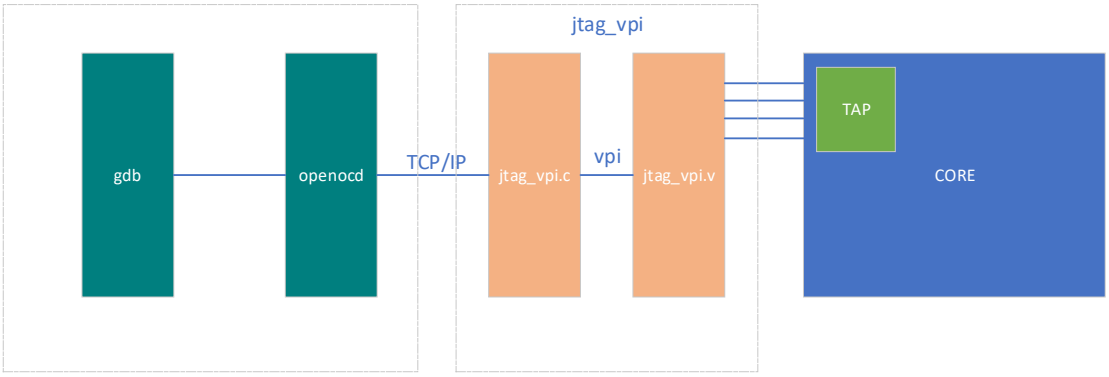


Figure 5-5 JTAG_VPI Connection Diagram

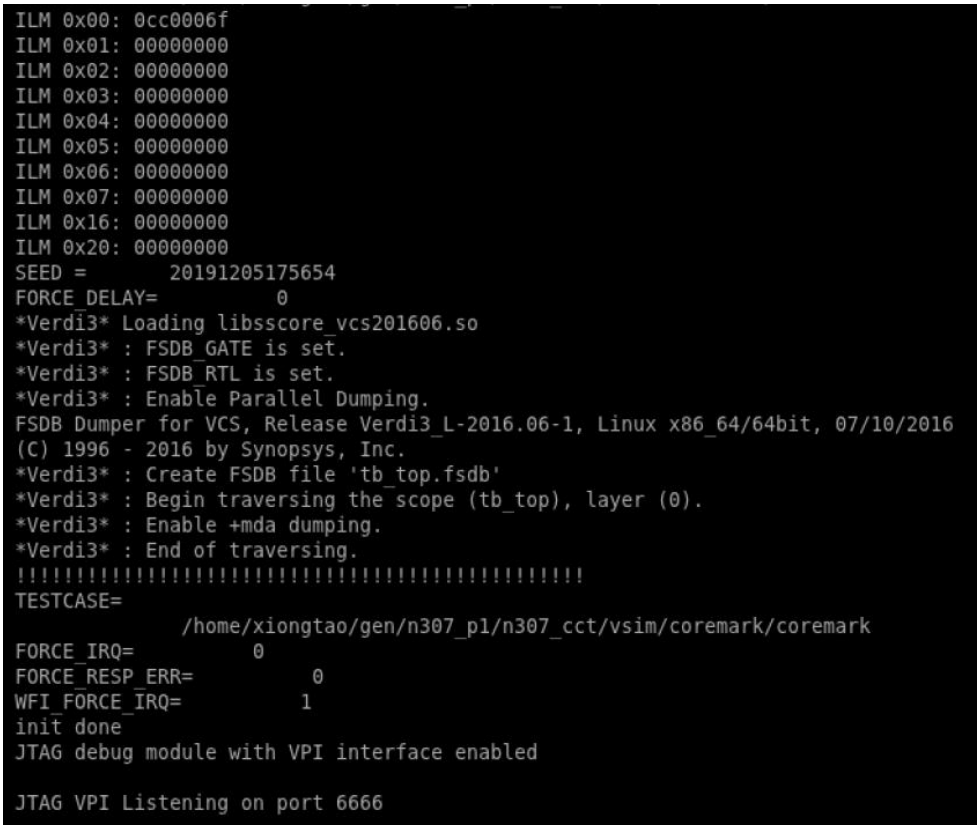
Here is a demo flow for reference:

It may take not short time for JTAG connection and debug, to avoid the simulation ends before JTAG connection, it is better to run a longer case or with '*while*' inside.

Go to n607/vsim dir and: *(here using n607 as reference)*

make clean
make install
make run_test TESTCASE=$PWD/testcase JTAGVPI=1 JTAGPORT=6666

Then you can get outputs as below in **the 1st terminal**:



Note:

'+jtagvpi' and '+jtag_port=JTAGPORT' are added in the options when compiling for simulation, so 'JTAGVPI=1' and 'JTAGPORT=*xxx*' are needed to be specified in the *make* command. JTAGPORT is the port to be connected with *openocd*, this same port is also needed to be specified in the *openocd_jtagvpi.cfg*. But this port may be already used by others, if so JTAG will choose other port automatically.

When simulation, after reset, JTAG is waiting for connection with *openocd*, but before this step, the port for both JTAG_VPI and GDB should be set in the *n607/tb/jtag_vpi/openocd_jtagvpi.cfg* as below: *(here using n607 as reference)*

openocd_jtagvpi.cfg

```
source [find interface/jtag_vpi.cfg]
#jtag_vpi_set_port $::env(JTAG_VPI_PORT)
jtag_vpi_set_port  6666
#jtag_vpi_set_port 34448

set _CHIPNAME riscv
jtag newtap $_CHIPNAME cpu -irlen 5
#jtag newtap $_CHIPNAME cpu -irlen 5

set _TARGETNAME $_CHIPNAME.cpu
target create $_TARGETNAME riscv -chain-position $_TARGETNAME
$_TARGETNAME configure -work-area-phys 0x80000000 -work-area-size 10000 -work-area-backup 1
$_TARGETNAME configure -work-area-phys 0x90000000 -work-area-size 10000 -work-area-backup 1

riscv set_reset_timeout_sec 3000
riscv set_command_timeout_sec 3000

tcl_port disabled
telnet_port disabled
gdb_port 3333

init

if {[ info exists pulse_srst]} {
  ftdi_set_signal nSRST 0
  ftdi_set_signal nSRST z
}

halt

echo "Ready for Remote Connections"
```

Open the 2nd new Terminal:

openocd -f path-to/openocd_jtagvpi.cfg

(openocd can be downloaded from https://www.nucleisys.com/download.php)

Then there will be output as below to wait for GDB connection:

```
Nuclei OpenOCD, 64-bit Open On-Chip Debugger 0.10.0+dev-00012-g9c34cc5a3-dirty (2019-12-04-07:09)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
Info : only one transport option; autoselect 'jtag'
Info : Set server port to 5555
Info : Set server address to 127.0.0.1
Info : Set server port to 6666
Info : Connection to 127.0.0.1 : 6666 succeed
Info : This adapter doesn't support configurable speed
Info : JTAG tap: riscv.cpu tap/device found: 0x13070a6d (mfg: 0x536 (Nuclei System Technology Co.,Ltd.), part: 0x3070, ver: 0x1)
Info : datacount=4 progbufsize=2
Info : Examined RISC-V core; found 1 harts
Info :  hart 0: XLEN=32, misa=0x4010912d
Info : Target has dm_timeouten bit, set RESETHALTREQ for verbose debugging
Info : Listening on port 3333 for gdb connections
Ready for Remote Connections
Info : tcl server disabled
Info : telnet server disabled
```

Open the 3rd new Terminal:

riscv-nuclei-elf-gdb testcase

then GDB connection:
(gdb) target remote :3333

Then start debugging:
Such as: '*info reg*' for checking the GPRs, '*x oxaddr*' for checking the memory value, etc.

```
(gdb) info reg
ra             0x80004a86       0x80004a86 <soc_init+42>
sp             0x9000ffd0       0x9000ffd0
gp             0x900010e8       0x900010e8
tp             0x0      0x0
t0             0x2000000        33554432
t1             0xc4     196
t2             0xc4     196
fp             0x90000000       0x90000000
s1             0x0      0
a0             0x64     100
a1             0xc4     196
a2             0x1688   5768
a3             0xb2     178
a4             0xb3     179
a5             0x2000000        33554432
a6             0xc3     195
a7             0xc3     195
s2             0x80006058       -2147458984
s3             0x0      0
s4             0x0      0
s5             0x0      0
s6             0x0      0
s7             0x0      0
s8             0x0      0
s9             0x0      0
s10            0x0      0
s11            0x0      0
t3             0xc3     195
t4             0xc4     196
t5             0xc4     196
t6             0x11     17
pc             0x80004844       0x80004844 <measure_cpu_freq+96>
```

6. Simulation with Comprehensive C Program

If user wants to run simulation with comprehensive C program, then the "Nuclei-SDK" is needed. For more details about Nuclei-SDK, please see its online doc from http://doc.nucleisys.com/nuclei_sdk.

Take "dhrystone" from Nuclei-SDK as example, user can use the following steps to make it running under simulation environment.

Note: Here we use N607 Core as example case, so use the CORE=n607.

```
// Step 1: Enter into nuclei-sdk
   cd nuclei-sdk

// Step 2: Generate the binary files with "Makefile" command.
make dasm PROGRAM=baremetal/benchmark/dhrystone CORE=n607 DOWNLOAD=ilm SIMULATION=1

// Step 3: Enter n607_rls_pkg
   cd n607_rls_pkg

// Step 4, copy the "dhrystone directory updated by Step 2" to n607_rls_pkg/n600/vsim directory
cp nuclei-sdk/application/baremetal/benchmark/dhrystone  n607_rls_pkg/n607/vsim -rf

// Step 5: Run simulation under vsim directory
    cd n607_rls_pkg/n607/vsim
    make run_test TESTCASE=$PWD/dhrystone/dhrystone
```

7. Logic Synthesis

7.1. Logic Synthesis for Verilog RTL

The release package have included an example synthesis project, the steps to run are as below (take N607 as example).

```
// Step 1: Enter into n607/syn

cd n607_rls_pkg/n607/syn


// Step 2: Modify the Makefile under n607/syn directory, to set up the Libaray path,
  Design path, frequency, etc.


// Step 3: Generate the synthesis scripts

make install


// Step 4: run synthesis

make syn
        // The generated synthesis result and reports will be under the directory
        // of syn_<CORE>_config_<freq>_<lib>/reports
```

7.2. Notes for Attentions

The example synthesis project above is just for reference, if the user want to get more precise result, it is suggested with following notes:

- It is strongly recommended to use the "**Flatten**" synthesis mode to flatten the hierarchy during synthesis optimization, to achieve better result of timing and areas.
- The "clock gating module" in the Core source files, need to be replaced to the real "clock gating cell" from the ASIC process library used by user.
  - Take N607 as example, the "clock gating module" is module "n607_clkgate", which can be searched under "n607_rls_pkg/n607/design/core" directory.

- jtag_TMS is used as clock when switching between 4-wire and 2-wire JTAG modes. It is recommended to set the frequency of jtag_TMS to half of jtag_TCK.
- A generated clock, u_n607_core_wrapper/u_n607_dbg_top/u_n607_dbg_2jtag/u_n607_dbg_apu/tck_s, which is a divide-by-3 clock of jtag_TCK and duty cycle is 1/3, should be created. Following is an example.

```
create_generated_clock -name tck_div3 -add -edges {1 3 7} \
-source jtag_TCK -master_clock jtag_TCK [get_pins  \
u_n607_core_wrapper/u_n607_dbg_top/u_n607_dbg_2jtag/u_n607_dbg_apu
/tck_s_reg/Q]
```

8. FPGA Prototyping

8.1. Files in FPGA Project

The files in the FPGA project are introduced as below.

```
N600_rls_pkg
    |n600
        |----fpga                // Directory for the FPGA project
            |----xxx200t            // Directory for Hummingbird Evaluation Kit
                |----constrs         // Directory for the .xdc constraint files
                |----Makefile        // Makefile
                |----script          // Directory for TCL script
                |----src             // Directory for Verilog code
                    |----system.org  // The top file of FPGA Verilog
            |----Makefile            // Top Makefile
```

There are several key notes in FPGA projects:

- FPGA Project will use Makefile to add a Macro “FPGA_SOURCE” in the Core’s defines.v file, as depicted in Figure 8-1. This will make sure the FPGA project is using the RTL as FPGA version (FPGA_SOURCE Macro included).
- In the top level file “system.org”, there are SoC top level module (n600_soc_top) instantiated. Besides, there are just the Xilinx I/O Pads instantiated.
- In the top level file “system.org”, the Xilinx MMCM (kind of PLL to generate clock) is instantiated. The FPGA project use the MMCM outputted clock for the SoC main system clock, and directly use the external input clock from the FPGA board (Hummingbird Evaluation Kit) as the real-time clock (32.768KHz).
- The JTAG Pads of SoC are constrained by nuclei-master.xdc, and map them to the pins of MCU_JTAG connecter on FPGA board (Hummingbird Evaluation Kit).

```
20 install:
21     mkdir -p ${PWD}/install
22     cp ${PWD}/../rtl/${CORE} ${INSTALL_RTL} -rf
23     cp ${PWD}/artydevkit/src/system.org ${INSTALL_RTL}/system.v -rf
24     sed -i 's/n200/${CORE}/g' ${INSTALL_RTL}/system.v
25     sed -i '1i\`define FPGA_SOURCE\'  ${INSTALL_RTL}/core/${CORE}_defines.v
```

Figure 8-1 FPGA_SOURCE Macro added in to Core’s defines.v

8.2. Generate Bitstream (MCS format)

In Section 3.1, it introduced the Nuclei Evaluation SoC, the SoC can be generated as FPGA Bitstream, and program into FPGA board (Hummingbird Evaluation Kit), such that, the FPGA board can be worked as a prototype board.

The steps to generate the Bistream for FPGA board are as below (take N607 as example):

```
// Step 1: Generate the FPGA version RTL codes.

cd n607_rls_pkg/n607/fpga
        // Enter into n607_rls_pkg/n607/fpga directory

make install (CORE=n607 FPGA_NAME=xxx200t) (default)
        // Use this command to specify the Core name (there will be default value
for each package) and the Board name xxx200t (for Hummingbird Evaluation Kit), to
generate RTL codes and Vivado scripts into the "install" directory.


// Step 2: Generate the Bitstream (MCS format).

make mcs
        // The generated MCS format Bitstream will be under
        // n607_rls_pkg/n607/fpga/wuhan200t/obj/system.mcs
```

8.3. Program Bitstream (MCS format) into FPGA

About how to program the Bitstream (MCS format) into the FPGA board (Hummingbird Evaluation Kit), please refer to the document <Nuclei_FPGA_DebugKit_Intro.pdf> which can be downloaded from "Development Boards" page of Nuclei website (http://www.nucleisys.com/developboard.php).