芯来科技 NUCLEI

Nuclei® Processor Integration Guide

Nucleisys

Contents

1 Copyright Notice

2 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.

List of Figures

List of Tables

3 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
1.3.0	2020/12/21	3, 8	1.Add mcu200t, ddr200t FPGA BOARD support
1.4.0	2021/10/09	3, 8	1.Add ku060 FPGA BOARD support
1.5.0	2022/01/25	10	1.Add “mux module” note in Logic Synthesis chapter 2.Add the tech folder as new package flow applied
1.6.0	2022/07/26	8	1.Add jtag_model description 2.Add sram_scan description 3.TB simulation use elf file directly
1.6.1	2022/08/05	11	1.FPGA mcs/bit no need to make install
1.7.0	2023/04/13	All	1.Use N900 release package as example
1.8.0	2023/12/18	8	1.Add more simulation commands to easy customers’ using
1.9.0	2024/08/13	6,9,10	1.Merge evalsoc doc into this doc. 2.Update run simulation command. 3.Use nxxx to replace n900
1.10.0	2024/08/15	4,9,10	1.Update the synthesis flow. 2.Add tech file description. 3.Add cpu_pma.csv description. 4.Add spyglass chapter to list the rules should be ignored.

4 Introduction of Release Package

4.1 Release Package

The Nuclei processor is released as a package, all the Nuclei processors (N/U/NX/UX/NA/NS/NI-100/200/300/600/900/1000 series) will keep consistent with the NXXX flow shown in this Integration guide.

Note: NXXX is the collective term for various CPU models from Nuclei System Technology.

For example:

- if user use N200 core, just replace the NXXX to N200 and nxxx to n200.
- if user use NS300 core, just replace the NXXX to NS300 and nxxx to ns300.
- if user use N900 core, just replace the NXXX to N900 and nxxx to n900.
- if user use UX900 core, just replace the NXXX to UX900 and nxxx to ux900.
- if user use NA900 core, just replace the NXXX to NA900 and nxxx to na900.

Using NXXX as example shown in *Release Packages* (page 6).

Table 4.1: Release Packages

Package Name	Direction
nxxx_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 NXXX Series Core can be licensed from Nuclei. After got the release package, user can use the following command to decompress.

```
tar -xzvf nxxx_rls_pkg.tar.gz
```

4.2 Files in Package

The files in the package are introduced as below.

```
nxxx_rls_pkg
    | env.csh     // csh environment script
    | env.sh      // bash environment script
    | libc.so.6   // The libc dependency file
    | nxxx.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 < Configure to Generate RTL > for more details


*After using ‘nuclei_gen’ to configure and generate the Core RTL, a new directory
‘nxxx’ will be generated, if using ‘nuclei_gen’ to reconfigure and generate a new
version of Core RTL, then the previous directory ‘nxxx’ will be moved to be
‘nxxx.bak’ and ‘nxxx’ is updated to the new generated one.*


    |nxxx
       |----bin             // Directory for scripts
       |----design          // Directory for RTL
          |----core         // Directory for Core
          |----soc          // Directory for subsystem in SoC
          |----tech         // Directory for synthesis tech library related
       |----riscv-tests     // Directory for assemble testcases
       |----tb              // Directory for Verilog TestBench
       |----nuclei-sdk      // Directory for generated nuclei-sdk
       |----sdk-import      // Directory for nuclei-sdk importing files
       |----vsim            // Directory for Simulation
                            // Please see more details of later Section
          |----bin          // Directory for functional scripts
          |----Makefile     // Makefile for simulation
          |----run          // Directory to run simulation
       |----fpga            // Directory for FPGA project
                            // Please see more details of later Section
       |----syn             // Directory for Synthesis project
                            // Please see more details of later Section
       |----cpufeature.mk   // makefile common arg file
       |----cpu_pma.csv     // pma(physical memory attribute) table
       |----Makefile        // makefile for initialing environment
```

Note: The above “nxxx_” is just a general prefix, for the specific core, user can set it in the nuclei_gen tool.

4.3 Naming Rule of Core

The source code of nxxx class have different prefix for the files and modules, for example, if it is NYYY Core, then the files and modules have the prefix “nyyy_”. The same naming rules applied to other Cores.

Nuclei NXXX series can support Cluster Configuration, NXXX series can support Cluster Configuration with SMP. The related naming rule can refer to related Databook.

4.4 Module Hierarchy of Core

Take nxxx as the example, as depicted in *Module Hierarchy of nxxx Core* (page 8), the key points are:

- nxxx_core_wrapper is the top module of the Core, which include several key sub-modules:
  - nxxx_core: The Core part.
  - nxxx_rst_ctrl: The module to sync external async reset signal to synced reset with "Asynchronously assert and synchronously de-assert" style.
  - nxxx_dbg_top: The module to handle the debug functionalities.
- nxxx_ucore is under Core hierarchy, it is the main part of Core.
- Besides the nxxx_ucore, there are several other sub-modules:
  - nxxx_clic_top: The private interrupt controller.
  - nxxx_tmr_top: The private timer unit.
  - nxxx_clk_ctrl: The clock control module.

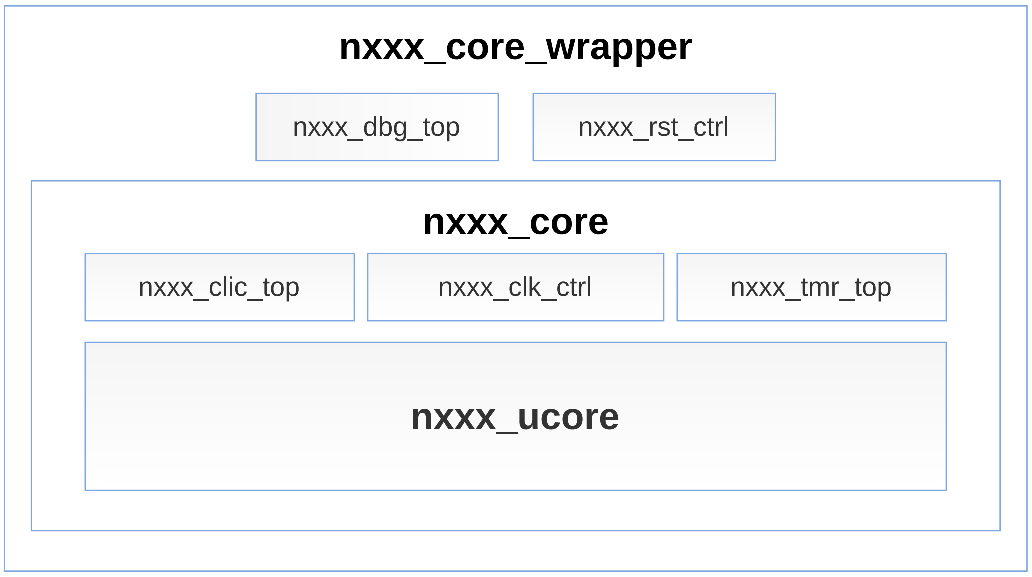


Fig. 4.1: Module Hierarchy of nxxx Core

NXXX Series Cluster Configuration module hierarchy is a little different from single core. Please refer related Top Diagram in the Databook.

5 Top Level Integration

Nuclei all series processor cores' delivered RTL have unitive hierarchy and top level.

This chapter takes NXXX as an example, other series processor cores follow the similar way.

5.1 Clocks

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

For the details of the NXXX Series Cores' clocks, please refer to Section "Clock Domains" of the document <Nuclei_XXX_Series_Databook.pdf>.

5.2 Interfaces

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

For the details of the NXXX Series Cores' interfaces, please refer to Chapter "Core Interfaces" of the document <Nuclei_XXX_Series_Databook.pdf>.

5.3 Memory Map

There are quite several interfaces and private peripherals for the N900 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 NXXX Series Cores' memory map, please refer to Section "Address Spaces of Interfaces and Private Peripherals" of the document <Nuclei_XXX_Series_Databook.pdf>.

6 SoC for Evaluation

In order to facilitate evaluation of Nuclei Processor Core, the prototype SoC (called EvalSoC) is provided for evaluation purpose.

6.1 Revision History

Rev.	Revision Date	Revised Section	Revised Content
1.0.0	2020/1/20	N/A	1.First version as the full English
1.1.0	2021/12/14	All Sections	1.Refine the content as the Eval_SoC adopts Nuclei SoC Subsystem
1.2.0	2022/07/06	All Sections	1.Eval_SoC is refined, peripherals are simplified
1.3.0	2024/05/10	All Sections	1.ROM is removed, UDMA and IOCP are added
1.3.1	2024/06/28	Section 11	1.Add register into misc device
1.3.2	2024/07/25	Section 7	1.description evalsoc memory and peripherals splitly. 2.add detail information for evalsoc memory and peripherals
1.3.3	2024/08/13	Section 3	1.Update Evalsoc diagram, add iram and dram.

6.2 Eval_SoC Background

To easy user to evaluate Nuclei Processor Core, the prototype SoC (called Eval_SoC) is provided for evaluation purpose. This Eval_SoC includes:

- Nuclei Processor Core,
- Memory resources for instruction and data.
  - On-Chip SRAMS.
  - External DDR.
- The SoC buses.
- The basic peripherals, such as UART, SPI.

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

Note:

- All modules including the peripherals in the Eval_SoC are for evaluation purpose and without Quality Guarantee/Supports, hence, it is not recommended to be used in any productions.
- If user really needs the SoC solution and peripherals IP with Quality Guarantee/Supports, Nuclei provides SoC Subsystem solution which can help user to quickly make the SoC product.

Nuclei can provide a concrete SoC Subsystem based on user's requirements, also we can provide:

- Soc RTL.
- Testbench for simulation.
- FPGA Bitstream.
- Scripts for synthesis.
- SDK with peripheral drivers and test cases.
- Documents.

For more details, please visit https://www.nucleisys.com/subsystem.php or contact Nuclei Support.

6.3 Eval_SoC Diagram

The Eval_SoC diagram is as depicted in *Eval_SoC Diagram* (page 11). There are Processor Core, Buses (System Bus and Peripheral Bus) and Peripherals in the SoC.

Note:

- Only 600/900/1000 Cores has the DDR, and the DDR uses the FPGA's.
- IOCP exits for 600/900/1000 Cluster Configurable.

For more information of the peripherals, please refer to *Peripherals of Eval_SoC* (page 15).

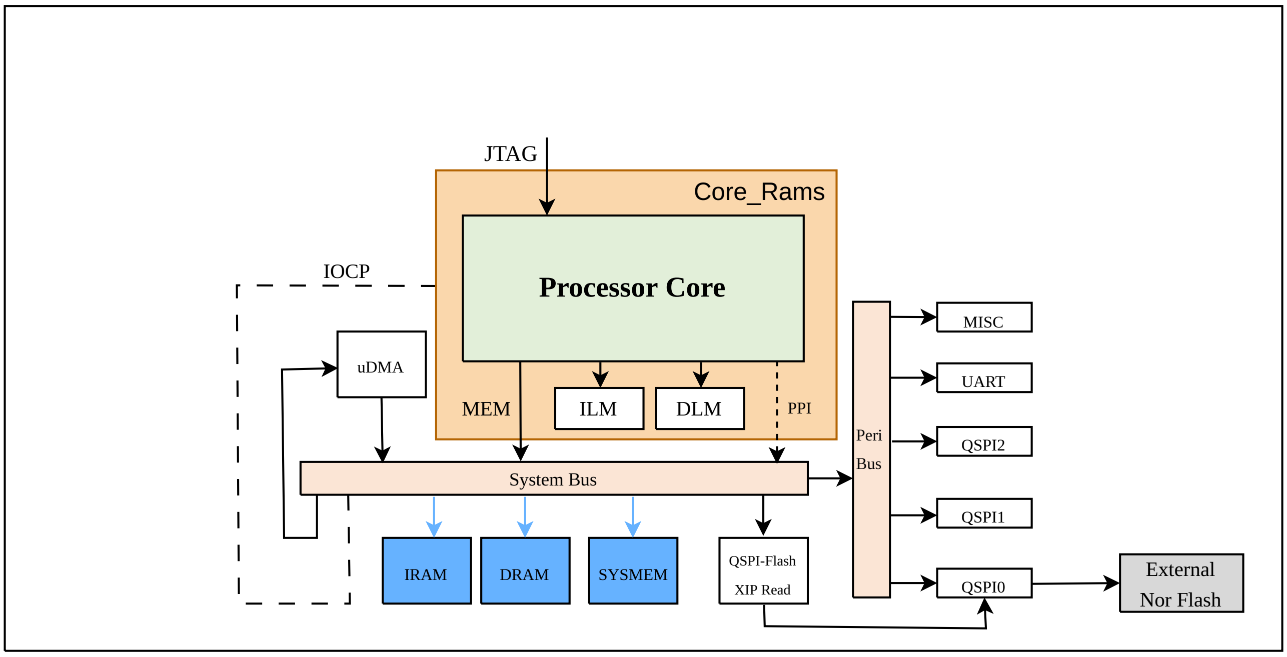


Fig. 6.1: Eval_SoC Diagram

6.4 Memory Resources of Eval_SoC

The Eval_SoC contained memory resources including on-chip SRAM resource, External Flash support and DDR (DDR only exists in FPGA Board of UX Core's Eval_SoC).

6.4.1 On-Chip SRAM Resource

The Eval_SoC contained on-chip SRAM, detailed as below.

- IRAM with features:
  - Base address can be configured.
  - Fixed Size. 32KB for 200/300, 64KB for 600/900/1000.
  - Mainly used to save the instructions, but also can be used to save the data.
  - Base address is allocated as shown in *Address Allocation of Eval_SoC Memory* (page 13).
  - The IRAM is connected to system bus, so the Core can access it with system bus interface.
- DRAM with features:
  - Base address can be configured.
  - Fixed Size. 32KB for 200/300, 64KB for 600/900/1000.
  - Mainly used to save the data.
  - Base address is allocated as shown in *Address Allocation of Eval_SoC Memory* (page 13).
  - The SRAM is connected to system bus, so the Core can access it with system bus interface.

6.4.2 External Flash Support

The Eval_SoC supports external SPI Nor Flash with QSPI0 interface, detailed as below.

- The QSPI0 support the eXecute-In-Place (XIP) mode, and it is the default mode after reset.
- With XIP mode, external Flash can be treated as a read-only memory directly accessible
  - Base and Size can be configured.
  - default address space is 0x2000_0000 ~ 0x3FFF_FFFF in this SoC.
- Hence, the instructions program can be saved at external Flash (will not be lost even after power down), and then after power-on reset, the Processor Core can fetch instruction directly from the external Flash with XIP mode through this QSPI master interface, and start to execute the programs.
- Please refer to *QSPI* (page 16) for more info of QSPI XIP mode.

6.4.3 SYSMEM

The Eval_SoC has system memory, detailed as below:

- Base address can be configured. Default is 0xa000_0000.
- Size can be configured.
  - Default is 256K for 200/200.
  - Default is 1.5G for 600/900/1000.
- It is sram for 200/300.
- It is ddr for 600/900/1000. The dde uses FPGA Board's DDR, can easy customer to boot Linux.
- The DDR only exists in FPGA Board (not in Simulation Environment).

6.5 Address Allocation of Eval_SoC

The address allocation of the Eval_SoC memory is as shown in *Address Allocation of Eval_SoC Memory* (page 13).

Table 6.2: Address Allocation of Eval_SoC Memory

Component	Address Spaces	Description
IRAM	Configurable Base address: CFG_ILM_BASE_ADDR Size: Fix 32KB for 200/300, 64KB for 600/900/1000.	IRAM address space.
DRAM	Configurable Base address: CFG_DLM_BASE_ADDR Size: Fix 32KB for 200/300, 64KB for 600/900/1000.	DRAM address space.
Off-Chip QSPI0 Flash Read	Configurable Base address: EVALSOC_FLASH_XIP_BASE_ADDR Size: 2^EVALSOC_FLASH_XIP_ADDR_WIDTH	QSPI0 with XiP mode read-only address space. Default: 0x2000_0000 ~ 0x3FFF_FFFF
SYSMEM	Configurable Base address: EVALSOC_SYSMEM_BASE_ADDR Size: 2^EVALSOC_SYSMEM_ADDR_WIDTH	System Memory address space. Note: for 600 & 900 series core, the default range is 0x8000_0000 ~ 0xffff_ffff; for 200 & 300, the default range is 0xa000_0000 ~ 0xa003_ffff.

The address allocation of the Eval_SoC peripherals is as shown in *Address Allocation of Eval_SoC Peripherals* (page 13).

Peripherals base address is configurable and use EVALSOC_PERIPS_BASE to config, default is 0x1000_0000.

The size is 1MB.

Table 6.3: Address Allocation of Eval_SoC Peripherals

Component	Address Spaces	Description
MISC	Configurable, Offset: 0x1_2000. Size: 4K	Misc Controller Note: This is only for 600 & 900 Default: 0x10012000
UART	Configurable Offset: 0x1_3000. Size: 4K	UART address space. Default: 0x10013000
QSPI0	Configurable Offset: 0x1_4000. Size: 4K	First QSPI address space. Default: 0x10014000
UDMA	Configurable Offset: 0x1_5000. Size: 4K	UDMA address space. Default: 0x10015000
QSPI1	Configurable Offset: 0x2_4000. Size: 4K	Second QSPI address space. Default: 0x10024000
QSPI2	Configurable Offset: 0x3_4000. Size: 4K	Third QSPI address space. Default: 0x10034000
IOCP	Configurable, Base: EVALSOC_IOCP_BASE Size: 2 ^ EVALSOC_IOCP_ADDR_WIDTH	Bus Fab to IOCP address space. Default: 0x5000_0000 ~ 0x5FFF_FFFF

Note:

- ILM & DLM Base Address and Size is configured by user in specific Core's nuclei_gen tool.
  - If Core config ILM, IRAM Base and Size are configured by Core
  - If Core don't config ILM, IRAM Base and Size are configured by Evalsoc
  - If Core config DLM, IRAM Base and Size are configured by Core
  - If Core don't config DLM, DRAM Base and Size are configured by Evalsoc
- Other Core's private peripherals like Debug, ECLIC, PLIC are all configurable in the specific Core's nuclei_gen tool, please refer the related Core's Databook.
- If CPU config or EvalSoC's address spaced is changed (different with default), the below files in NucleiSDK's should match the hardware address:
  - SoC/evalsoc/cpufeature.mk
  - SoC/evalsoc/Common/Include/cpufeature.h
  - SoC/evalsoc/evalsoc.json
  - SoC/evalsoc/Board/nuclei_fpga_eval/Source/GCC/evalsoc.memory
  - SoC/evalsoc/Board/nuclei_fpga_eval/openocd_evalsoc.cfg
- Peripherals base, XIP base address, XIP size, IOCP base and IOCP size are configurable in nuclei_gen.

6.6 Reset PC Address

After reset, the reset PC address of the Processor Core could be from external Flash (XIP mode), or from other memory.

- In FPGA board, the reset PC address is from external Flash (XIP)
  - In the SoC, "QSPI Flash XIP Read" bus slave port have been allocated with default address space of 0x2000_0000 ~ 0x3FFF_FFFF at system bus.
  - In the FPGA version of SoC RTL code, there is a macro check "ifdef FPGA_SOURCE", and in this case the input pin "reset_vector" of Processor Core is tied as value "QSPI Flash XIP Read",then after reset, the Core will start execution from "QSPI Flash XIP Read", i.e., the external Flash.
  - Please refer to *Eval_SoC Diagram* (page 11) for more information of the Eval_SoC diagram, and refer to *QSPI* (page 16) for more info of QSPI XIP mode.
- In SIMULATION Environment, the reset PC address is default from the ELF file's start address and user can change it to use "BOOT_ADDR" argument.

6.7 Ports of Eval_SoC

The top level ports of the Eval_SoC are as shown as followed table.

Table 6.4: Ports of Eval_SoC

Direction	Name	Description
Input	JTAG TCK	JTAG TCK Signal
Output	JTAG TDO	JTAG TDO Signal
Output	JTAG_DRV_TDO	JTAG TDO Output Enable
Input	JTAG TMS	JTAG TMS Signal
Input	JTAG TDI	JTAG TDI Signal
Output	JTAG_TMS_out	cJTAG TMS output signal
Output	JTAG_DRV_TMS	cJTAG TMS output enable signal
Output	JTAG_BK_TMS	cJTAG TMS bus keep control signal

continues on next page

Table 6.4 – continued from previous page

Direction	Name	Description
Bidir	QSPI DQ 3	Quad SPI Data3
Bidir	QSPI DQ 2	Quad SPI Data2
Bidir	QSPI DQ 1	Quad SPI Data1
Bidir	QSPI DQ 0	Quad SPI Data0
Output	QSPI CS	Quad SPI Chip Select
Output	QSPI SCK	Quad SPI Clock

6.8 Interrupts of Eval_SoC

The interrupts of Eval_SoC is managed by the interrupt controller of the core, the interrupts are directly connected to the external interrupt interface of the processor Core.

Table 6.5: Allocation of External Interrupts in Eval_SoC

IRQ_ID	ECLIC Interrupt ID	PLIC Interrupt ID	Source
0 & 32	19 & 51	1 & 33	uart
1 & 33	20 & 52	2 & 34	uDMA
2 & 34	21 & 53	3 & 35	qspi0
3 & 35	22 & 54	4 & 36	qspi1
4 & 36	23 & 55	5 & 37	qspi2
5 & 37	24 & 56	6 & 38	Counter0 irq
6 & 38	25 & 57	7 & 39	Counter1 irq

Note:

- IRQ_ID is the hardware interrupt wire connect number and ECLIC/PLIC Interrupt ID is software concept.
- To be compatible with interrupt source of the Core is less than 32, the external interrupt is connected to two IRQ inputs.

6.9 Peripherals of Eval_SoC

The Chapter will shortly introduce the peripherals used in the Eval_SoC.

6.9.1 MISC

This component is to control the input of Core.

Table 6.6: Register Description for MISC

Register	Offset	Description
Counter0 irq	0x0	bit31:0: set the interrupt0 initial counter value
Counter1 irq	0x4	bit31:0: set the interrupt1 initial counter value
IOCP_R	0x20	bit 3:0 to set IOCP arcache value; bit 31:4 to set IOCP read chanel's higher address [X:28].
IOCP_W	0x24	bit 3:0 to set IOCP awcache value; bit 31:4 to set IOCP write chanel's higher address [X:28].
VLM	0x28	bit4:0: set the vlm sram latency
Nmi	0x100	bit31:0: set the nmi initial counter value
Event	0x104	bit31:0: set the event initial counter value

continues on next page

Table 6.6 – continued from previous page

Register	Offset	Description
Debug	0x108	bit0: control dbg_sec_enable input bit1: control dbg_isolate input bit2: control dbg_override_dm_sleep input bit3: control dbg_stop_at_boot input bit4: control dbg_i_dbg_stop input
Version	0xf00	Indicate EvalSoC's version, it is fixed to 32'h00010100.

6.9.1.1 Interrupt Counter Register

In Misc component, we implement two interrupt counter.

When counter value decrease to 0, it will generate level interrupt. And interrupt will clear when software write 0xffff_ffff into counter.

6.9.1.2 IOCP Register

We connect the iocp port of cpu with one busfab slave port. But the slave port only allocate 256M region size.

IOCP_R and IOCP_W register can set the arcache/awcache and the high address value.

6.9.1.3 VLM Register

We connect the vlm port with vlm sram if VLM has config. And this register can config the latency for vlm sram.

6.9.1.4 Nmi Counter Register

In Misc component, we implement one nmi counter.

When counter value decrease to 0, it will generate nmi. And nmi will clear when software write 0xffff_ffff into counter.

6.9.1.5 Event Counter Register

In Misc component, we implement one event counter.

When counter value decrease to 0, it will generate event. And event will clear when software write 0xffff_ffff into counter.

6.9.2 QSPI

There are 3 independent QSPI in the SoC which can communicate with outside.

- One QSPI for Flash:
  - There is one QSPI master in the SoC which has dedicated SoC ports to communicate with external FLASH (e.g. Nor Flash with SPI interface).
  - This QSPI support the eXecute-In-Place (XIP) mode, and it is the default mode after reset. With XIP mode, external Flash can be treated as a read-only memory directly accessible. Hence, the instructions program can be saved at external Flash (will not be lost even after power down), and then after power-on reset, the Processor Core can fetch instruction directly from the external Flash with XIP mode through this QSPI master interface, and start to execute the programs.
- Another two QSPI:
  - There are two QSPI master (without XIP mode supported), one QSPI master has 4 CS (Chip Select) signals, another has 1 CS signal.

Note:

- This QSPI in the Eval_SoC is for evaluation purpose and without Quality Guarantee/Supports, hence, it is not recommended to be used in any productions.
- If user really needs the SoC solution or QSPI with Quality Guarantee/Supports, Nuclei provides SoC Subsystem which can help user to quickly make the SoC product. Please visit https://www.nucleisys.com/subsystem.php or contact Nuclei Support.

6.9.3 UART

There is one independent UART (Universal Asynchronous Receiver-Transmitter) in the SoC, which can communicate with outside.

Note:

- This UART in the Eval_SoC is for evaluation purpose and without Quality Guarantee/Supports, hence, it is not recommended to be used in any productions.
- If user really needs the SoC solution or UART with Quality Guarantee/Supports, Nuclei provides SoC Subsystem which can help user to quickly make the SoC product. Please visit https://www.nucleisys.com/subsystem.php or contact Nuclei Support.

6.9.4 uDMA

There is one independent uDMA ((Micro Direct Memory Access) in the SoC, which can communicate with outside.

Note:

- This uDMA in the Eval_SoC is for evaluation purpose and without Quality Guarantee/Supports, hence, it is not recommended to be used in any productions.
- If user really needs the SoC solution or uDMA with Quality Guarantee/Supports, Nuclei provides SoC Subsystem which can help user to quickly make the SoC product. Please visit https://www.nucleisys.com/subsystem.php or contact Nuclei Support.

7 FPGA, SDK and IDE for Evaluation

7.1 FPGA Evaluation Board and JTAG Debugger

Nuclei have customized 3 types FPGA evaluation boards, called MCU200T Evaluation Kit, DDR200T Evaluation Kit and KU060 Evaluation Kit (please go to Nuclei website for details of these 3 types of boards: https://www.nucleisys.com/developboard.php).

The FPGA boards can be used as the SoC prototype board directly:

- If the FPGA have been pre-burned (programmed) with "Nuclei EvalSoC", this board can be worked as a SoC prototype. The embedded software engineers can 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 *FPGA Prototyping* (page 44).

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

For the detailed introduction of the "Hummingbird Debugger Kit", please refer to "Development Boards" page of Nuclei website (http://www.nucleisys.com/developboard.php).

7.2 Software Development Kit (SDK)

Nuclei has 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 EvalSoC", and with the demo software projects from Nuclei SDK, user can quickly familiarize the software development for Nuclei Processor Cores.

Nuclei gen will generate the nuclei-sdk for user configuration.

In simulation env, user can run nuclei sdk test immediately, please see <Nuclei_Cpu_Case_Description.pdf> to get detail information.

7.3 Integrated Development Environment (IDE)

There are many choices for user to select an IDE to develop software based on Nuclei processor core. Here introduces Nuclei Studio first , then third-party IDEs.

7.3.1 Nuclei Studio

Nuclei Studio is an Eclipse based IDE developed and maintained by Nuclei. It has followed advantages for users who want to develop software on Nuclei cores:

- Free to download and use.
- Support all Nuclei series cores.
- Pre-integrated RISC-V GCC/GDB and Nuclei OpenOCD.
- Pre-integrated Nuclei SDK.
- Support Win10/Ubuntu/Redhat OS.
- One Click for project templates and configurations.

Please see the Nuclei Website https://www.nucleisys.com/download.php to get the install package and user guide. Please refer follow picture:

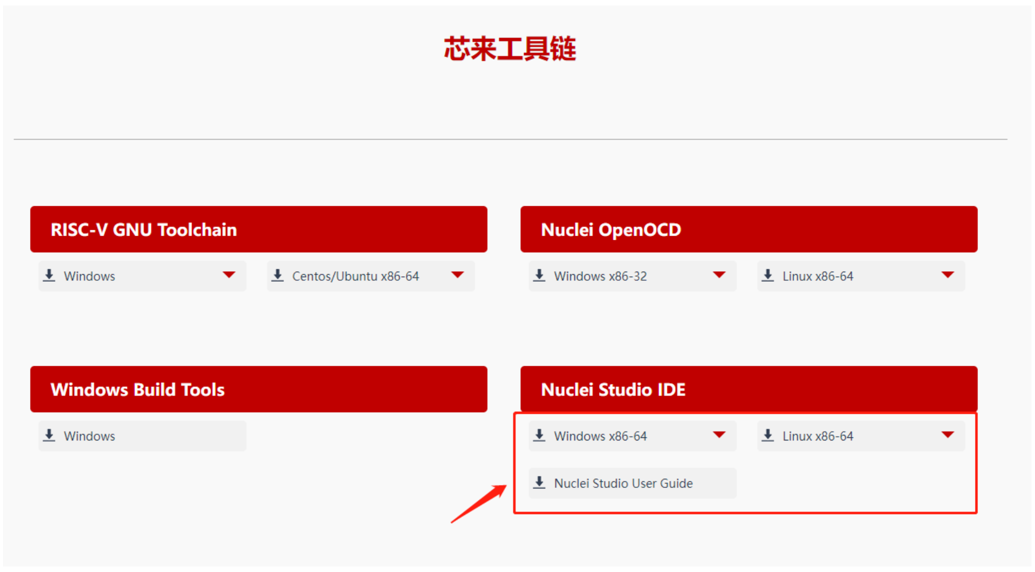


Fig. 7.1: Nuclei Studio IDE Install Package and User Guide

7.3.2 Third-Party IDEs

Nuclei collaborates with many famous IDEs vendors ,such as Segger, Lauterbach, IAR, COMPILER,etc. Their IDEs and relate tools can fully support Nuclei Core based chips and boards.

Please see the Nuclei Website https://www.nucleisys.com/download.php to get the details. For the quick-start of Segger Embedded Studio (SES) for Nuclei Core, there are two documents which can be downloaded from this page.

8 Configure to Generate RTL

8.1 Use nuclei_gen Tool to generate RTL Codes

Since Nuclei all Series Cores are fully configurable, Nuclei develops a tool called *nuclei_gen*. User can easily configures the Core according to their requirements at their field, and then generate the RTL code.

Take NXXX for example, under the nxxx_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.
- nxxx.iplib: The IP library for Core RTL generation.
- env.sh: Bash shell environment script.
- env.csh: csh shell environment 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.

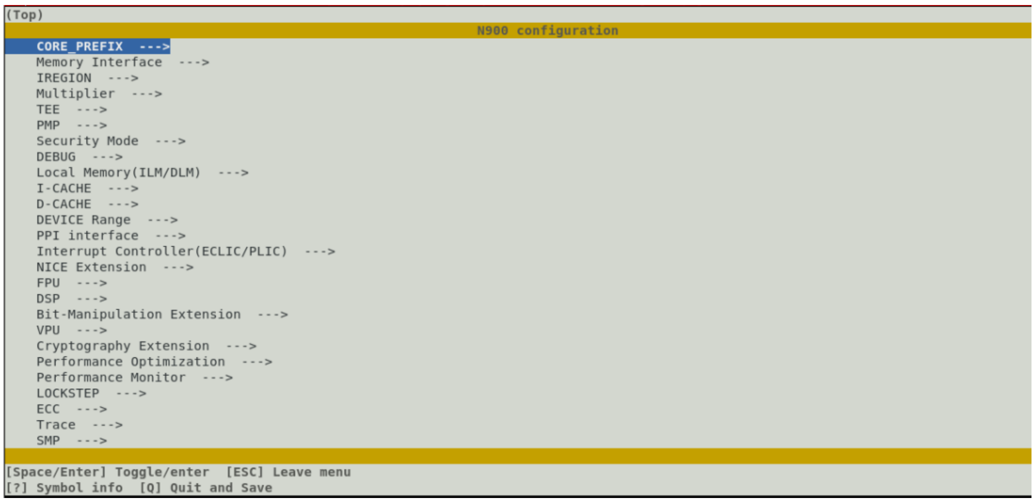
Before starting the nuclei_gen tool, there are several environment variables need to be set:

- 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 nxxx_rls_pkg
- PROJ_NAME: The Core's name.
- PROJ_GEN_ROOT: RTL source code directory, by default it is nxxx_rls_pkg/nxxx. If user wants to generate RTL code to other directory, please 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 *The user interface of core_gen tool* (page 21). The configurable options shown in the Window are also explained in document <Nuclei_XXX_Series_Processor_Core_Databook.pdf>.



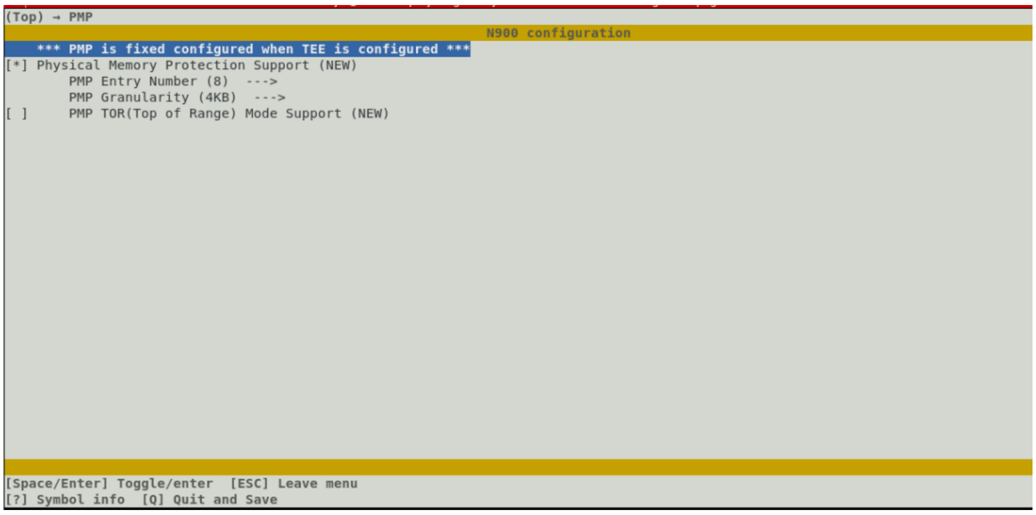
```
(Top)
                                        N900 configuration
    CORE_PREFIX  --->
    Memory Interface  --->
    IREGION  --->
    Multiplier  --->
    TEE  --->
    PMP  --->
    Security Mode  --->
    DEBUG  --->
    Local Memory(ILM/DLM)  --->
    I-CACHE  --->
    D-CACHE  --->
    DEVICE Range  --->
    PPI interface  --->
    Interrupt Controller(ECLIC/PLIC)  --->
    NICE Extension  --->
    FPU  --->
    DSP  --->
    Bit-Manipulation Extension  --->
    VPU  --->
    Cryptography Extension  --->
    Performance Optimization  --->
    Performance Monitor  --->
    LOCKSTEP  --->
    ECC  --->
    Trace  --->
    SMP  --->

[Space/Enter] Toggle/enter  [ESC] Leave menu
[?] Symbol info  [Q] Quit and Save
```

Fig. 8.1: The user interface of core_gen tool

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 *PMP Configuration sub-menu* (page 21).



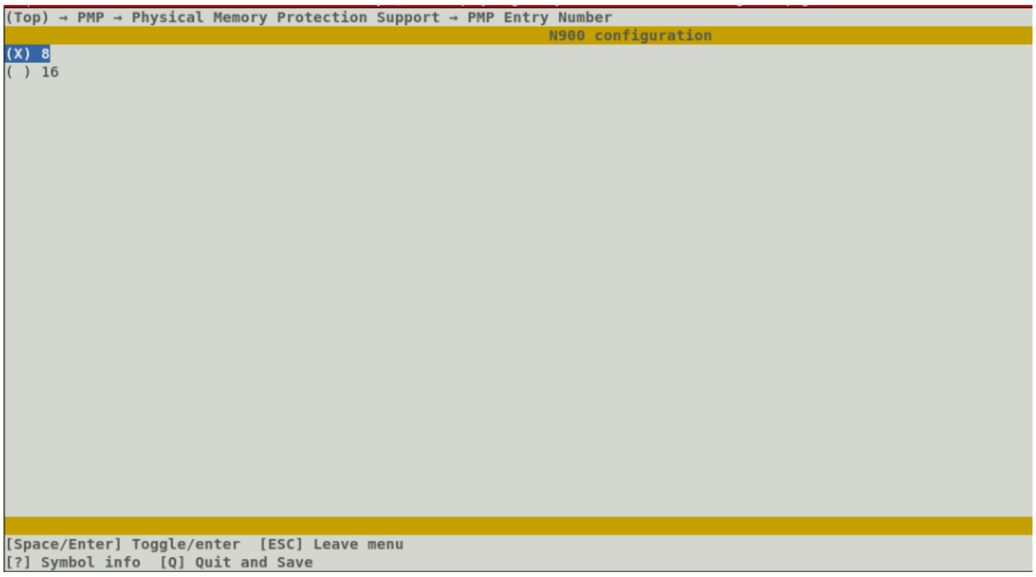
```
(Top) → PMP
                                        N900 configuration
    *** PMP is fixed configured when TEE is configured ***
[*] Physical Memory Protection Support (NEW)
        PMP Entry Number (8)  --->
        PMP Granularity (4KB)  --->
[ ]     PMP TOR(Top of Range) Mode Support (NEW)

[Space/Enter] Toggle/enter  [ESC] Leave menu
[?] Symbol info  [Q] Quit and Save
```

Fig. 8.2: PMP Configuration sub-menu

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.

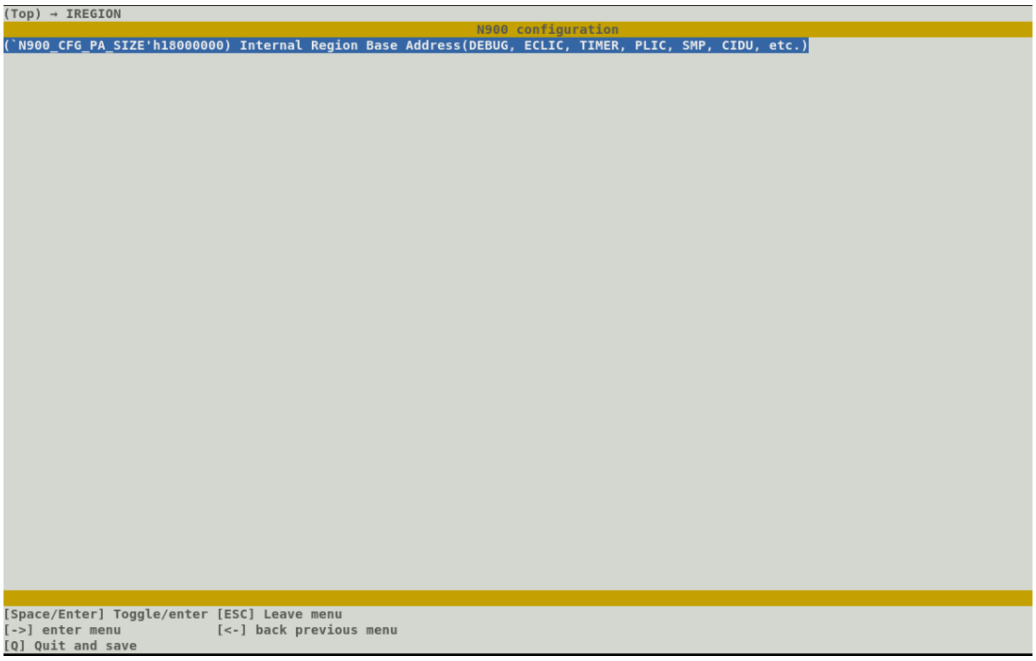
Continue the above example, if enter "Pmp Entry Number" sub-menu, it is as shown in *PMP Entry Number Menu* (page 22). In this sub-menu, use "SPACE" key to choose the option you want.



```
(Top) → PMP → Physical Memory Protection Support → PMP Entry Number
                                                N900 configuration
(X) 8
( ) 16

[Space/Enter] Toggle/enter  [ESC] Leave menu
[?] Symbol info  [Q] Quit and Save
```

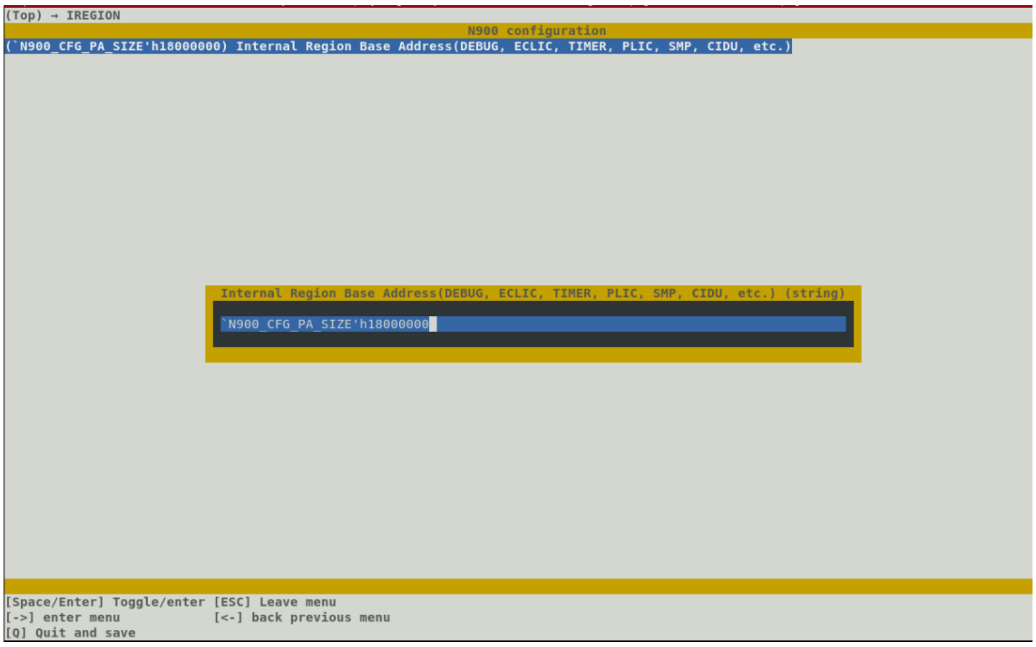
Fig. 8.3: PMP Entry Number Menu

There might be some options need to be inputted with values. For example, the IREGION Base Address as shown in *IREGION Base Address Configuration* (page 22). In this option, enter the "Enter" or "Space" key, the configuration input window will be shown, as in *Input the IREGION Base Address Value* (page 23).



```
(Top) → IREGION
                                                N900 configuration
(`N900_CFG_PA_SIZE'h18000000) Internal Region Base Address(DEBUG, ECLIC, TIMER, PLIC, SMP, CIDU, etc.)

[Space/Enter] Toggle/enter [ESC] Leave menu
[->] enter menu            [<-] back previous menu
[Q] Quit and save
```

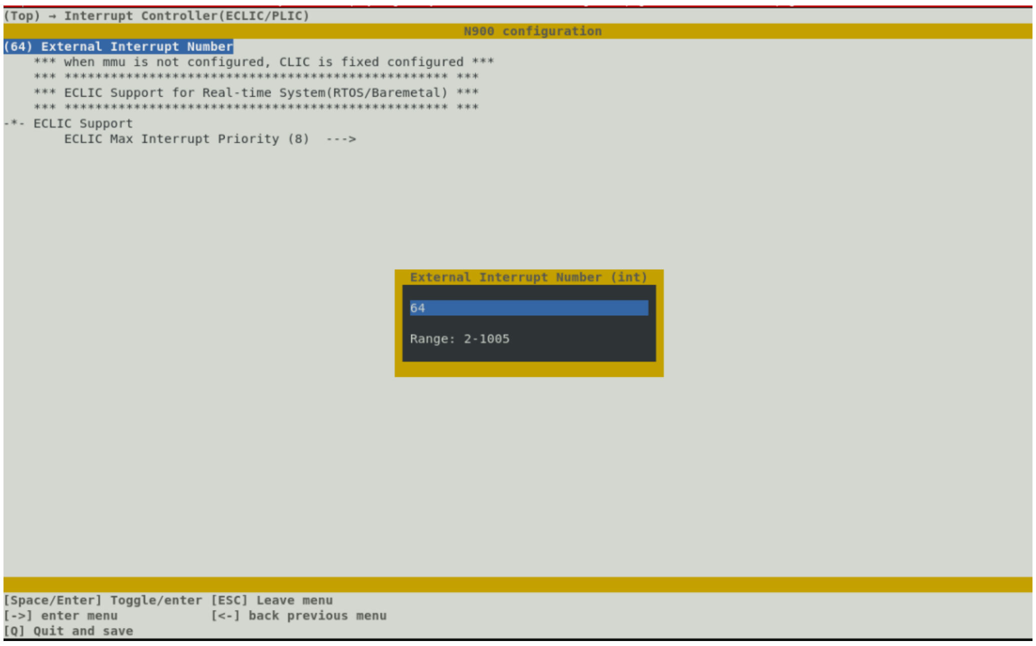
Fig. 8.4: IREGION Base Address Configuration



```
(Top) → IREGION
                                N900 configuration
(`N900_CFG_PA_SIZE'h18000000) Internal Region Base Address(DEBUG, ECLIC, TIMER, PLIC, SMP, CIDU, etc.)

    Internal Region Base Address(DEBUG, ECLIC, TIMER, PLIC, SMP, CIDU, etc.) (string)
    `N900_CFG_PA_SIZE'h18000000

[Space/Enter] Toggle/enter [ESC] Leave menu
[->] enter menu            [<-] back previous menu
[Q] Quit and save
```

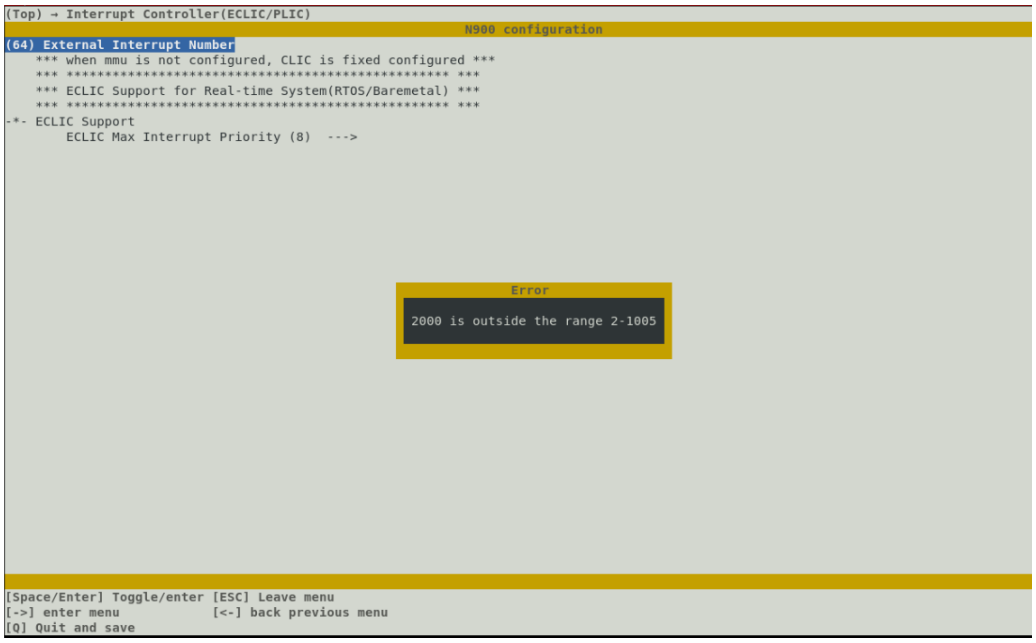
Fig. 8.5: Input the IREGION Base Address Value

There might be some options need to be inputted with values, but with constraints. For example, as shown in *Input the IRQ Number Value* (page 23), 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 *The Configuration Error* (page 24).



```
(Top) → Interrupt Controller(ECLIC/PLIC)
                                N900 configuration
(64) External Interrupt Number
    *** when mmu is not configured, CLIC is fixed configured ***
    *** *************************************************** ***
    *** ECLIC Support for Real-time System(RTOS/Baremetal) ***
    *** *************************************************** ***
-*- ECLIC Support
        ECLIC Max Interrupt Priority (8)  --->

                          External Interrupt Number (int)
                          64

                          Range: 2-1005

[Space/Enter] Toggle/enter [ESC] Leave menu
[->] enter menu            [<-] back previous menu
[Q] Quit and save
```

Fig. 8.6: Input the IRQ Number Value



```
(Top) → Interrupt Controller(ECLIC/PLIC)
                                   N900 configuration
(64) External Interrupt Number
    *** when mmu is not configured, CLIC is fixed configured ***
    *** ************************************************** ***
    *** ECLIC Support for Real-time System(RTOS/Baremetal) ***
    *** ************************************************** ***
-*- ECLIC Support
        ECLIC Max Interrupt Priority (8)  --->

                              Error
                     2000 is outside the range 2-1005

[Space/Enter] Toggle/enter [ESC] Leave menu
[->] enter menu            [<-] back previous menu
[Q] Quit and save
```

Fig. 8.7: The Configuration Error

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 needs 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 nuclei_gen tool will use its inherent default configurations.

If user wants to run simulation with the generated RTL code, It is better to keep default ILM and DLM base address and size setting. As when compile the testcases in later chapter, the default linker script file uses the default ILM and DLM base address and size. In other words, if user changes the base address or size of ILM and DLM in this step, please also change the linker script file correspondingly.

8.2 Check and Simulation the Verilog RTL

If user wants check or simulation the generated RTL code, the steps are detailed as below (take NXXX 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 nxxx_rls_pkg

```

(continues on next page)

(continued from previous page)

```
source env.sh

./nuclei_gen

    // The detailed way to configured and generate code is described in previous section.
    The Core’s RTL code is generated under nxxx_rls_pkg/nxxx


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

cd nxxx_rls_pkg/nxxx/vsim

make install

make compile  // Compile the RTL

make run_sdk TESTNAME=cpuinfo  // run the cpuinfo test, to print cpu hardware information.


// 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
```

9 Simulation with Simple Assembly Testcase

9.1 Overview of Self-Check Testcase

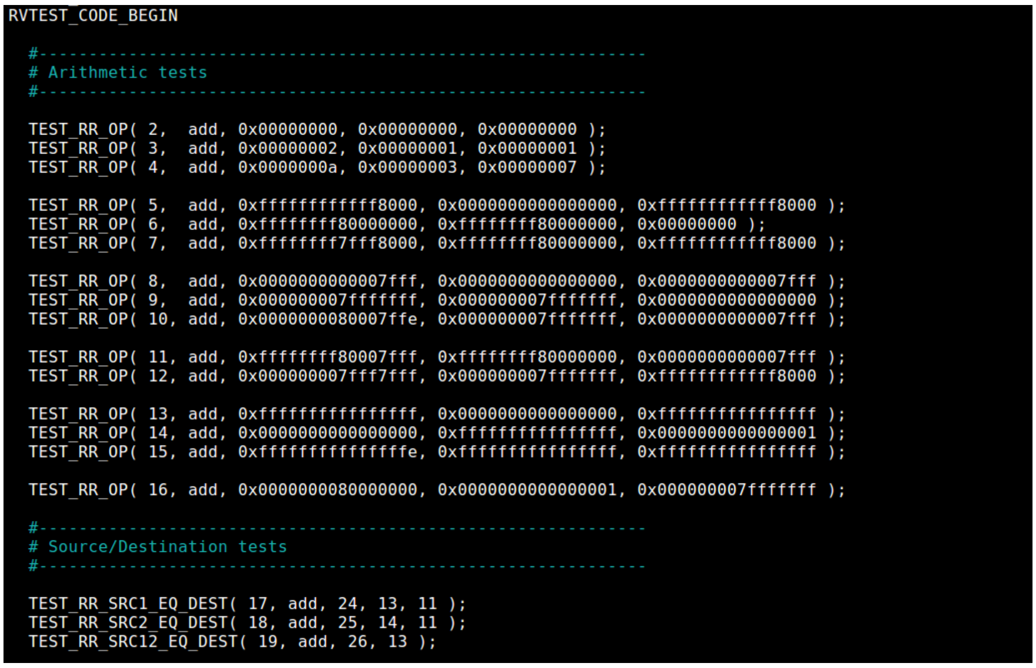
The "Self-Check Testcase" is a kind of assembly Testcase which can self-check if it is "passed" or "failed". The Self-Check Testcase are under the following directory.

```
nxxx_rls_pkg
    |----nxxx
       |----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 *The code segment of add.S test* (page 27), the Testcase (source code under isa_origs/rv32ui/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, 0xffffffffffffffff );

  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 );
```

Fig. 9.1: The code segment of add.S test

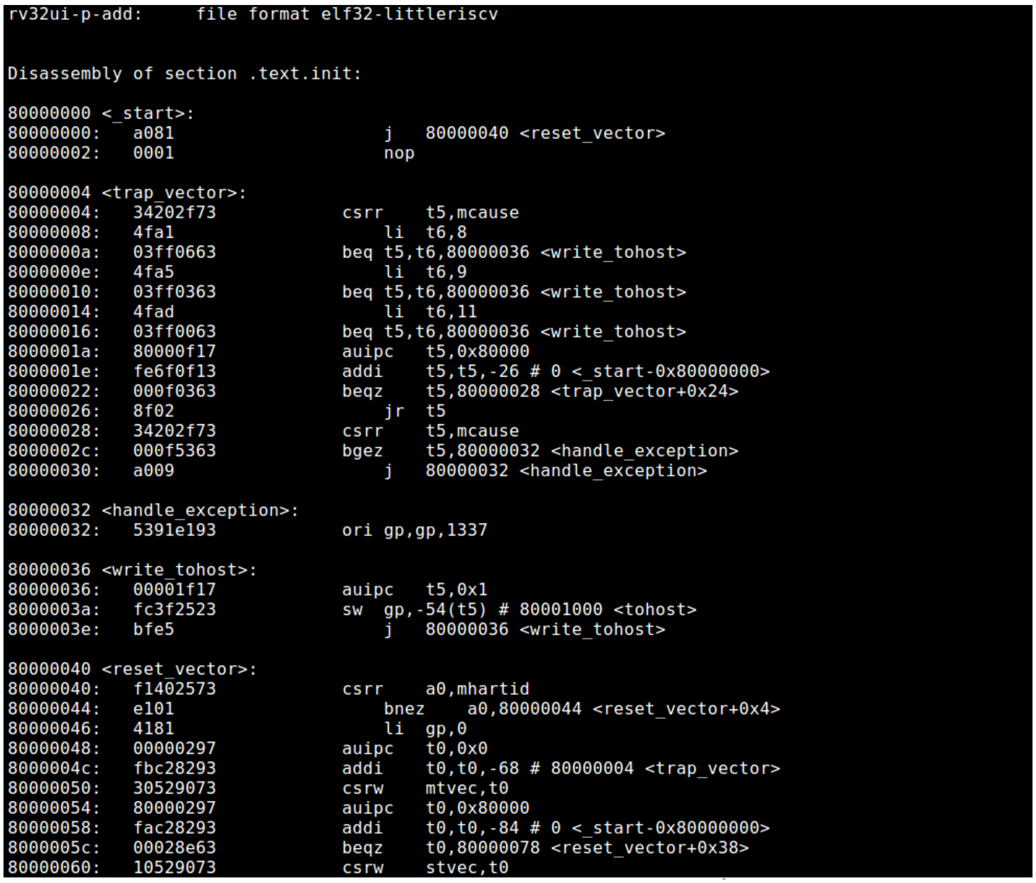
9.2 Testbench to Initialize Self-Check Testcase

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 *Check and Simulation the Verilog RTL* (page 24). The binary file (.verilog file) for each Testcase will be generated under riscv-tests/isa/generated directory, exampled as below.

```
nxxx
    |----riscv-tests
       |----isa
          |----generated
             |----rv32ui-p-addi         // The generated Elf file
             |----rv32ui-p-addi.dump    // The disassembly code
                    ...                 // more cases ...
```

The content of disassembly code (e.g., rv32ui-p-addi.dump) is as shown in *The content of rv32ui-p-addi.dump file* (page 28).



```
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
```

Fig. 9.2: The content of rv32ui-p-addi.dump file

9.3 Introduction of Testbench

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

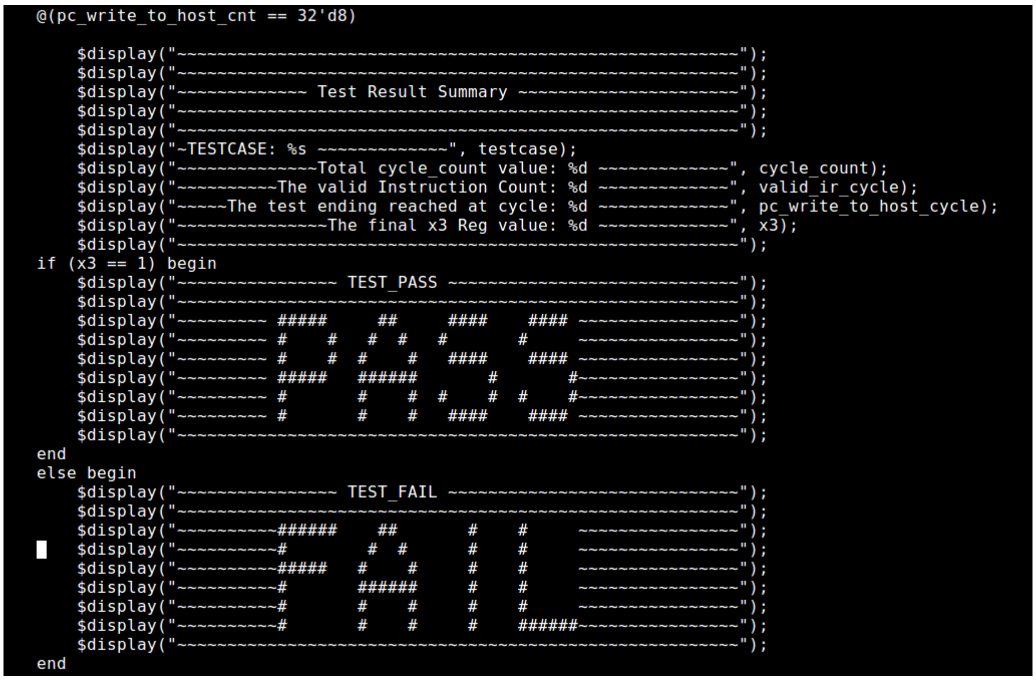
```
nxxx
    |----tb
       |----tb_*.v //Verilog TestBench source codes
```

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 elf2mem to read the elf file and then initialize the 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 *Print the PASS or FAIL in Testbench* (page 29)

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
```

Fig. 9.3: Print the PASS or FAIL in Testbench

9.4 Steps to Run Simulation

The steps to run simulation are as below (use nxxx for 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
needs to add this “bin” path into the Linux $PATH environment
variable.

// Step 1: Generate the tests.

cd nxxx_rls_pkg/nxxx/vsim
    // Enter into the vsim directory.

make clean
    // Clean up the directory.

make install
    // Use this command to generate the Testbench and compile the riscv-tests


// 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
```

(continues on next page)

(continued from previous page)

```
    // 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”.

// Step 5: If want to run dhrystone/coremark, use the following commands.

make run_sdk TESTNAME=dhrystone
    // This command will run the dhrystone program in nuclei-sdk.

make run_sdk TESTNAME=coremark
    // This command will run the coremark program in nuclei-sdk.
```

Note:

- make help will list all commands supported.
- In Step1, make install will check if the riscv gcc toolchain is installed or not, then compile the risv-test or not. And if there is multi version riscv gcc toolchain including gcc 13, it will use gcc 13.
- In Step2, make compile needs the environment has VCS tool, we have verified vcs2019-06-SP1 and later version. If your VCS version is older, please contact Nuclei Support.
- In Step 3, our EvalSoC default set the reset_vector of core to 0x1000 , it is the ROM of SoC, the Rom Code will jump to 0x8000_0000 (default ILM base address), if the ILM Base is changed or user want to set the reset_vector to other address, please use BOOT_ADDR argument just like:

  make run_test TESTNAME=rv32ui-p-add BOOT_ADDR=a0000000

  it means the reset vector is 0xa000_00000, the value of BOOT_ADDR is always hex number without “0x” prefix.
- In Step5, if the dhrystone or coremark result is not as expected, please contact Nuclei Support.

9.5 Introduction of JTAG_VPI

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

```
nxxx
    |----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:

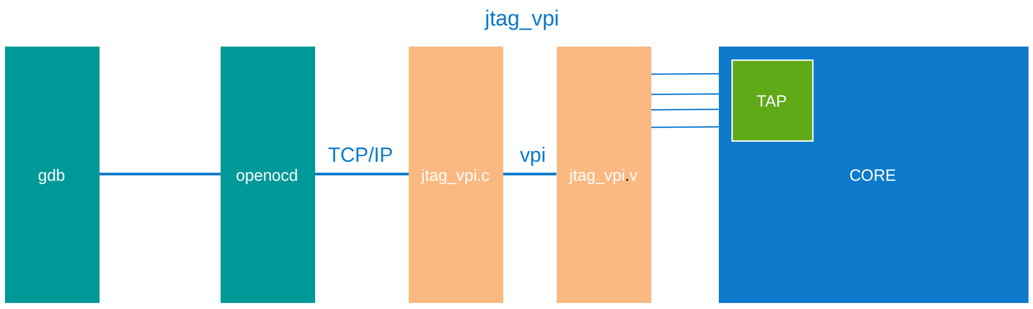


Fig. 9.4: JTAG_VPI Connection Diagram

Here is a demo flow for reference:

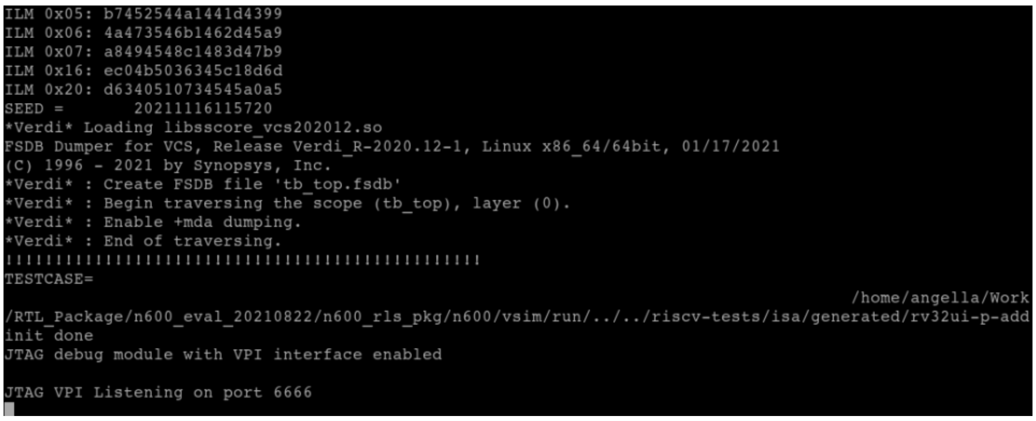
It may take not short time for JTAG connection and debug, to avoid the simulation ends before JTAG connection, it is better to run a longer case or with '*while*' inside.

Open the 1st Terminal:

here uses nxxx as reference:

```
cd nxxx_rls_pkg/nxxx/vsim
    // Enter into the vsim directory.

make clean
    // should make clean here

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

make run_sdk TESTNAME=helloworld JTAGVPI=1 JTAGPORT=6666
   //run simulation and JTAGVPI task
```

Then it can get outputs as below:



```
ILM 0x05: b7452544a1441d4399
ILM 0x06: 4a473546b1462d45a9
ILM 0x07: a8494548c1483d47b9
ILM 0x16: ec04b5036345c18d6d
ILM 0x20: d6340510734545a0a5
SEED =       20211116115720
*Verdi* Loading libsscore_vcs202012.so
FSDB Dumper for VCS, Release Verdi_R-2020.12-1, Linux x86_64/64bit, 01/17/2021
(C) 1996 - 2021 by Synopsys, Inc.
*Verdi* : Create FSDB file 'tb_top.fsdb'
*Verdi* : Begin traversing the scope (tb_top), layer (0).
*Verdi* : Enable +mda dumping.
*Verdi* : End of traversing.
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
TESTCASE=
                                                                   /home/angella/Work
/RTL_Package/n600_eval_20210822/n600_rls_pkg/n600/vsim/run/../../riscv-tests/isa/generated/rv32ui-p-add
init done
JTAG debug module with VPI interface enabled

JTAG VPI Listening on port 6666
```

Fig. 9.5: JTAG_VPI Waiting for OpenOCD Connection

Note:

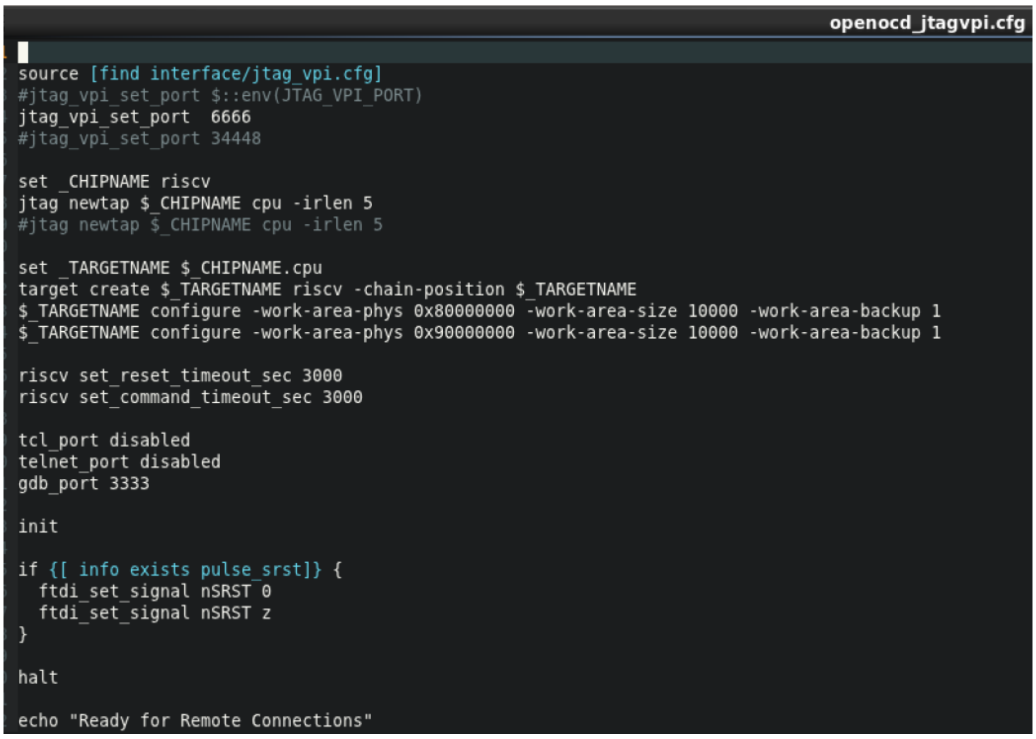
- If the result in your environment is not the same, maybe the simulation ends before the JTAG

task begins, please change to another complicated case or use the case in next Section.

- '+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 *nxxx/tb/jtag_vpi/openocd_jtagvpi.cfg* as below: *(here uses nxxx as reference)*



```
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"
```

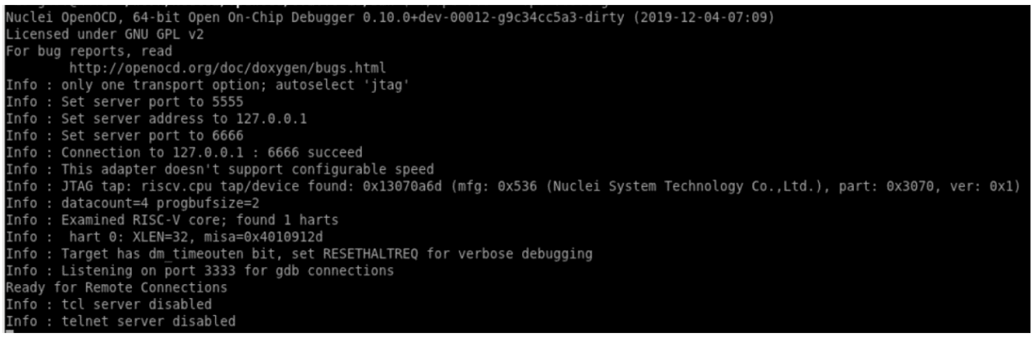
Fig. 9.6: OpenOCD Configuration File

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
```

Fig. 9.7: OpenOCD Waiting for GDB connection

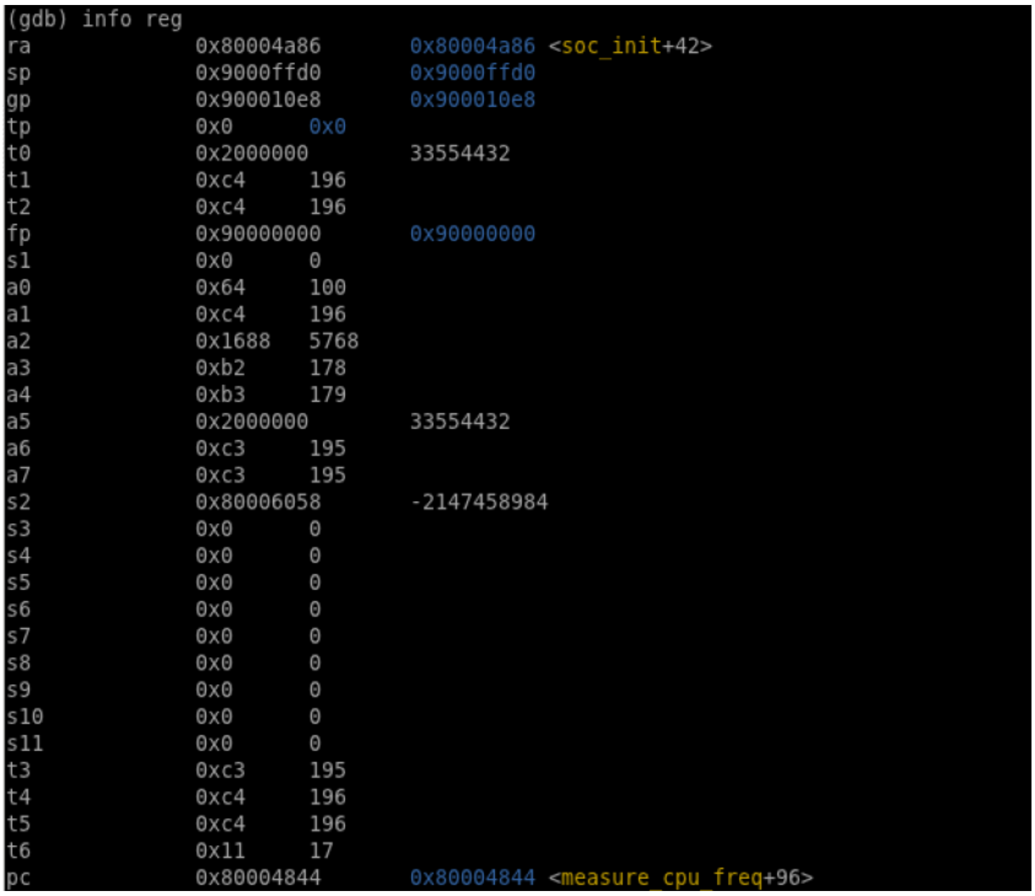
Open the 3rd new Terminal: RISC-V GDB should be ready here, if not, please go back to *Check and Simulation the Verilog RTL* (page 24) and install the RISC-V GNU Toolchain firstly. And more GDB commands can be found in *nxxx/tb/jtag_vpi/README.md*.

```
riscv-nuclei-elf-gdb

(gdb) set remotetimeout 1999999999
```

(continues on next page)

(continued from previous page)

```
    // set the timeout value

(gdb) target remote :3333
    // then GDB connection

(gdb) set disassemble-next-line on
    // enable auto display the disassemble

(gdb) info reg
    // then use GDB commands to start debugging
```



```
(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>
```

Fig. 9.8: GDB shows all GPRs

9.6 Introduction of JTAG_Model

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

```
nxxx
    |----tb
       |----jtag_model   // jtag model source codes
```

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

Below is the connection diagram:

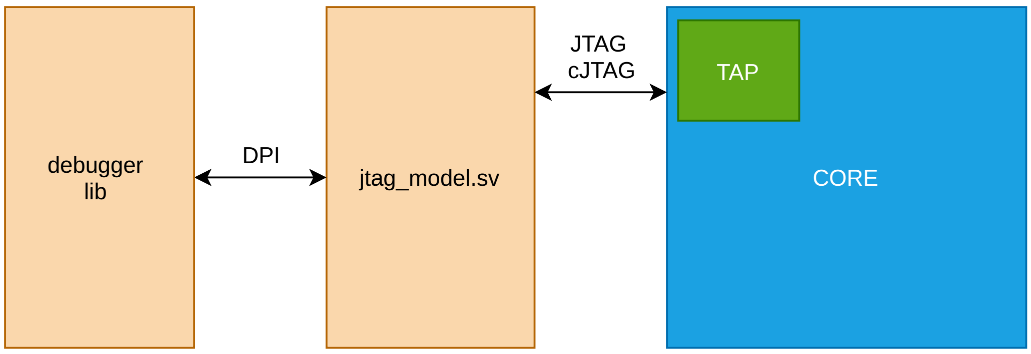


Fig. 9.9: JTAG_Model Connection Diagram

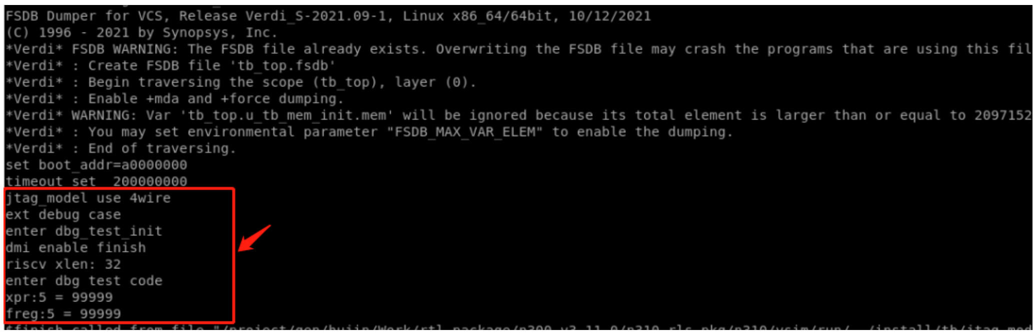
Here is a demo flow for reference:

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.

Here uses nxxx as reference:

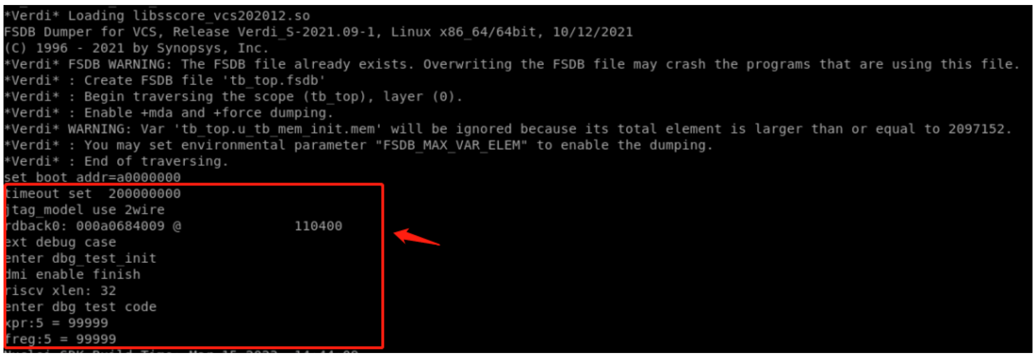
```
cd nxxx_rls_pkg/nxxx/vsim
    // Enter into the vsim directory.

make clean
    // should make clean here

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

make run_sdk TESTNAME=helloworld JTAG_MODEL=1 JTAG_TEST=demo
   //run simulation and JTAG_Model task with JTAG interface

make run_sdk TESTNAME=helloworld JTAG_MODEL_2WIRE=1 JTAG_TEST=demo
   //run simulation and JTAG_Model task with cJTAG interface
```

Then it can get outputs as below:



```
FSDB Dumper for VCS, Release Verdi_S-2021.09-1, Linux x86_64/64bit, 10/12/2021
(C) 1996 - 2021 by Synopsys, Inc.
*Verdi* FSDB WARNING: The FSDB file already exists. Overwriting the FSDB file may crash the programs that are using this fil
*Verdi* : Create FSDB file 'tb_top.fsdb'
*Verdi* : Begin traversing the scope (tb_top), layer (0).
*Verdi* : Enable +mda and +force dumping.
*Verdi* WARNING: Var 'tb_top.u_tb_mem_init.mem' will be ignored because its total element is larger than or equal to 2097152
*Verdi* : You may set environmental parameter "FSDB_MAX_VAR_ELEM" to enable the dumping.
*Verdi* : End of traversing.
set boot_addr=a0000000
timeout set  200000000
jtag_model use 4wire
ext debug case
enter dbg_test_init
dmi enable finish
riscv xlen: 32
enter dbg test code
xpr:5 = 99999
freg:5 = 99999
```

Fig. 9.10: Outputs of JTAG_Model Task 4 Wire (JTAG)



```
*Verdi* Loading libsscore_vcs202012.so
FSDB Dumper for VCS, Release Verdi_S-2021.09-1, Linux x86_64/64bit, 10/12/2021
(C) 1996 - 2021 by Synopsys, Inc.
*Verdi* FSDB WARNING: The FSDB file already exists. Overwriting the FSDB file may crash the programs that are using this file.
*Verdi* : Create FSDB file 'tb_top.fsdb'
*Verdi* : Begin traversing the scope (tb_top), layer (0).
*Verdi* : Enable +mda and +force dumping.
*Verdi* WARNING: Var 'tb_top.u_tb_mem_init.mem' will be ignored because its total element is larger than or equal to 2097152.
*Verdi* : You may set environmental parameter "FSDB_MAX_VAR_ELEM" to enable the dumping.
*Verdi* : End of traversing.
set boot addr=a0000000
timeout set  200000000
jtag_model use 2wire
rdback0: 000a0684009 @              110400
ext debug case
enter dbg_test_init
dmi enable finish
riscv xlen: 32
enter dbg test code
xpr:5 = 99999
freg:5 = 99999
```

Fig. 9.11: Outputs of JTAG_Model Task 2 Wire (cJTAG)

Note:

- If the result in your environment is not the same, maybe the simulation ends before the JTAG task begins, please change to another complicated case or use the case in next Section.
- User can edit jtag_model/debugger_test/demo/dbg_test_code.cpp to do more test, then should use make to update the debugger.so.

User can see <Nuclei_Jtag_Model_User_Guide.pdf> to get more detail information about jtag_model.

User can see <Nuclei_Cpu_Case_Description.pdf> to get jtag model test and run command.

9.7 Introduction of SRAM_Scan

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

```
nxxx
    |----tb
       |----tb_sram_scan.v   // sram scan source codes
```

The SRAM_Scan module is used to test all SRAM of the Core/Cluster, if user has replace the SRAM, then can integrate this module to the SoC and run the task to check the SRAM replacement is complete before running software.

Here uses nxxx as reference:

```
cd nxxx_rls_pkg/nxxx/vsim
    // Enter into the vsim directory.

make clean
    // should make clean here

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

make run_test TESTNAME=tb_sram_scan
   // Run SRAM_Scan task
```

When it shows PASS, it is all OK. When it shows FAIL, user should check more details.

Note:

- It should not run any other case when do sram scan task.

10 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.

Note: Nuclei-SDK is auto generated in the package after nuclei_gen.

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

Note: Here we use NXXX Core as example case.

```
// Step 1: install tb

cd nxxx_rls_pkg/nxxx/vsim

make install

// Step 2: run nuclei-sdk dhrystone test

make run_sdk TESTNAME=dhrystone

// Step 3: open waveform

make wave_sdk TESTNAME=dhrystone
```

If you want to run elf immediately, use below command to run

```
// Step 1: install tb

cd nxxx_rls_pkg/nxxx/vsim

make install

// Step 2: run your test

make run_test TESTCASE=your_elf

// Step 3: open waveform

make wave TESTNAME=dhrystone
```

11 Spyglass

Core RTL code has been full checked by spyglass tools.

11.1 RTL Lint

User can ignore below spyglass lint:

- FlopEConst
- W34: Macro ‘xxx’ is never used
- W111: Not all elements of array ‘xxx’ are ‘read’.
- W120: Variable ‘xxx’ declared but not used.
- W146: Explicit named association is recommended in instance references.
- W175: Parameter ‘xxx’ declared but not used.
- W240: Input “xxx” declared but not read.
- W287b: Instance output port “xxx” is not connected.
- W310: Converting integer to reg.
- W313: Converting integer to single bit.
- W341: Constant xxx is extended by 0.
- W415a: Signal “xxx” is being assinged multiple times (assignment within same for-loop) in same always block.
- W446: Output port ‘xxx’ is being read inside module.
- W527: Potential dangling ‘else’ statement of ‘if’ statement.
- W528: Variable “xxx” set but not read.
- W563: Use of unary reduction operator “x” on a 1-bit expression.
- Notab: Use spaces rather than tabs for indentation.
- ExprParen: Use parenthesis in complex expressions.
- RegOutputs: Port ‘xxx’ is not driven by a register.
- Indent: Statement xxx.
- ClkName: Clock signal name ‘xxx’ should conform to the regular expression.
- ConstName: Constant ‘xxx’ does not follow recommended naming convention
- InstName: Instance name ‘xxx’ does not follow naming convention.
- SigName: Signal xxx does not follow naming convention.

- VarName: Variable xxx does not follow naming convention.
- ParamName: Parameter name xxx does not follow naming convention
- PortComment: Port declaration does not have comments.
- SignalComment: Signal declaration does not have comments.
- VariableComment: Variable declaration does not have comments.
- LineLength: Line length xxx is more than yyy characters.
- STARC05-1.1.1.1: Filename 'xxx' does not indicate module and should be 'yyy'
- STARC05-1.1.4.6a: Port 'xxx' is 'tied-high'.
- STARC05-2.3.1.3: Insert delay in flip-flop data path
- STARC05-2.3.5.1: Flip-flop 'xxx' has fixed input value 'y'
- STARC05-2.10.5.1: Logic 'xxx' with operand bit-width greater than 8 is common with conditional expression.
  - xxx_gnrl.v
- STARC05-2.10.6.6:n The expression 'xxx' has more than one arithmetic operators
- STARC05-3.1.3.1: Order of specification of port 'xxx' not consistent with port declaratio.
- ConstDrivenNet-ML: Wire xxx is assigned a constant value 1'b0/1'b1.
- UnloadedInPort-ML: Detected unloaded(unconnected)input port "xxx".
- UnloadedNet-ML: Detected unloaded(unconnected) net.
- UnloadedOutTerm-ML: etected unloaded(unconnected) output terminal.
- UndrivenNUnloaded-ML: Detected undriven and unloaded(unconnected) net "xxx".
- UnInitParam-ML: Instance xxx instantiated but parameter not specified
- NoConstSourceInAlways-ML: Signal 'xxx' is assigned a constant value 'yyy' in implicit always block.
- NoExprInPort-ML: Expressions "xxx" used in port connection.
- ConstantInput-ML: Input 'xxx' to Instance 'yyy' is assigned a constant.
- ParamOverrideMismatch-ML: Mismatch in number of parameter over-rides and number of parameters.
- GenvarUsage-ML: Genvar variable 'i' declared but not used.
- SelfAssignment-ML: Same operand 'xxx' used on both sides of an assignment.
- OneModule-ML: File 'xxx' contains more than one module
- ModuleName-ML: Module 'xxx' does not follow the recommended naming convention
- HangingFlopOutput-ML: Flop output "xxx" connected to module output "yyy" is unused.

11.2 CDC Lint

User can ignore below spyglass cdc lint for some module or signals:

- Ac_cdc01a
  - Fast(core_clk_aon) to slow(jtag_tck) clock
- Ac_datahold01a
  - Destination flop "***.u_dbg_top.u_dbg.***"
- Ac_datahold01a
  - Designation flot "***.dtm_tap_aon_top.***"
- Ac_glitch03
  - Destination flop "***.dtm_tap_aon_top.***"

- Clock_glitch05
  - Asynchronous source '***.reset_bypass'
  - Asynchronous source '***.clkgate_bypass'
- Clock_sync06
  - Primary output signal "ram_sync_reset_n"
- Reset_check12
  - flop "***" is not asynchronously asserted
- Ac_conv03
  - synchronizers ***.dtm_tap_aon_top.***
- Ac_conv04
  - At least two bits (0, 1) of source bus "***.u_dbg_top.u_dbg.***"
- Ac_datahold01a
  - Synchronized crossing: destination flop "***.u_dbg_top.***"
- Reset_sync04
  - Asynchronous reset signal '***.por_reset_n' is synchronized at least twice

12 Logic Synthesis

12.1 Logic Synthesis for Verilog RTL

We have provided a set of reference synthesis flow. This flow use one open source technology library.

Note: User can replace the open source technology library to real asic technology library to synthesis.

We provide three synthesis flow:

- Mode1: Core logic synthesis. Apply on single core.
- Mode2: Core logic with sram synthesis. Apply on single core.
- Mode3: Hierarchical synthesis. Apply on smp core
  - Synthesis nxxx_core_rams module with sram first.
  - Synthesis nxxx_cluster_top module last.

12.2 Tool required

This flow require three tools:

- Synopsys Design Compiler tool.
  - We recommend using a tool version no lower than S-2021.06-SP1.
  - Flow mode1/mode2/mode3 required.
- Synopsys Library Compiler tool.
  - We recommend using a tool version no lower than R-2020.09.
  - Flow mode2/mode3 required.
- Synopsys PrimeTime tool.
  - We recommend using a tool version no lower than 2020.09-SP5-1.
  - Flow mode3 required.

12.3 Flow Mode1

Run the flow steps:

```
// Step1: load the dc tool into your PATH
export PATH={your_dc_bin_path}:$PATH

// Step2: Enter into nxxx/syn

cd nxxx_rls_pkg/nxxx/syn

// Step3: generate the synthesis flow

make gen

// Step4: Enter into generated demo dir
cd demo_logic_only

// Step5: Run synthesis
make dc
```

For more detail information, you can see syn/README.txt to get more details.

12.4 Flow Mode2

Run the flow steps:

```
// Step1: load the dc tool and library compiler into your PATH
export PATH={your_dc_bin_path}:{your_lc_bin_path}:$PATH

// Step2: Enter into nxxx/syn

cd nxxx_rls_pkg/nxxx/syn

// Step3: generate the synthesis flow

make gen SYN_WITH_SRAM=1

// Step4: Enter into generated demo dir
cd demo_syn_with_sram

// Step5: Run synthesis
make dc
```

For more detail information, you can see syn/README.txt to get more details.

12.5 Flow Mode3

Run the flow steps:

```
// Step1: load the dc tool, library compiler tool and primetime tool into your PATH
export PATH={your_dc_bin_path}:{your_lc_bin_path}:{your_pt_bin_path}:$PATH

// Step2: Enter into nxxx/syn

cd nxxx_rls_pkg/nxxx/syn
```

(continues on next page)

(continued from previous page)

```

// Step3: generate the synthesis flow

make gen SYN_WITH_SRAM=1

// Step4: Enter into generated demo dir
cd demo_hier_syn_logic_with_sram

// Step5: Run synthesis
make dc
```

For more detail information, you can see syn/README.txt to get more details.

12.6 Tech Module Replace

Core will use some special stdcell module, and these module files are put in design/tech dir.

Take nxxx as example:

Table 12.1: Technology module

Tech_file	Tech_mode	Description	ASIC Replace
nxxx_hand_buf.v	nxxx_hand_buf	Buffer function. Using to keep signal.	Recommend replace
nxxx_hand_mux_bit.v	nxxx_hand_mux_bit	Mux function. Using to jtag clock mux.	Must replace.
nxxx_gnrl_tech_clkgate.v	nxxx_gnrl_tech_clkgate	Clocking gating function.	Must replace.
nxxx_gnrl_tech_sync.v	nxxx_gnrl_tech_sync	Sync function. Using to sync some input signal. Using to asynchronous reset, synchronous release.	Optional replace.
nxxx_gnrl_tech_sram.v	nxxx_gnrl_tech_sram	Sram function.	Must replace.

12.7 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 design source files, need to be replaced to the real "clock gating cell" from the ASIC process library. And please keep the comment unchanged during replacing.
  - Take nxxx as example, the "clock gating module" is module "nxxx_gnrl_tech_clkgate" in , which is in file

    nxxx_rls_pkg/nxxx/design/tech/nxxx_gnrl_tech_clkgate.v.
- The "mux module" in the design source files, need to be replaced to the real "mux cell" from the ASIC process library (this module is for clock source switch, user can chose specific cell if the ASIC process library has). And please keep the comment unchanged during replacing.
  - Take nxxx as example, the "mux module" is module "nxxx_hand_mux_bit", which is in file

    nxxx_rls_pkg/nxxx/design/tech/nxxx_hand_mux_bit.v
- The "hand buf module" in the design source files, need to be replaced to the real "hand buf cell" from the ASIC process library. And please keep the comment unchanged during replacing.
  - Take nxxx as example, the "hand buf module" is module "nxxx_hand_buf", which is in file

nxxx_rls_pkg/nxxx/design/tech/nxxx_hand_buf.v

- The "sync module" in the design source files, can be replaced to the real "sync cell" from the ASIC process library. And please keep the comment unchanged during replacing.
  - Take nxxx as example, the "sync module" is module "nxxx_gnrl_tech_sync", which is in file

    nxxx_rls_pkg/nxxx/design/tech/nxxx_gnrl_tech_sync.v
- The "sram module" in the design source files, need to be replaced to the real "sram" from the ASIC process library. And please keep the comment unchanged during replacing.
  - Take nxxx as example, The "sram" is module "nxxx_gnrl_tech_ram", which is in file

    nxxx_rls_pkg/nxxx/design/tech/nxxx_gnrl_tech_ram.v
  - We provide one sram wrapper file in file

    nxxx_rls_pkg/nxxx/syn/mem/wrapper/spram_macros_wrapper.v

    User can replace the module name with prefix "NUCLEI_SPRAM_" to real ASIC sram.

13

FPGA Prototyping

13.1 Files in FPGA Project

The files in the FPGA project are introduced as below.

```
nxxx_rls_pkg/nxxx
  |----fpga                     // Directory for the FPGA project
    |----boards                 // Directory for the FPGA boards
      |----share                // Directory for MCU200T,DDR200T and KU060 Common Kit
        |----xdc                            // Directory for the .xdc constraint files
        |----script                         // Directory for TCL script
        |----src                            // Directory for Verilog code for fpga top
      |----mcu200t                  // Directory for MCU200T Evaluation Kit
        |----nuclei-master.xdc  // The main .xdc constraint files
        |----xdc                                // Directory for the .xdc constraint files
        |----script                             // Directory for TCL script
        |----src                                // Directory for Verilog code for fpga top
      |----ddr200t                              // Directory for DDR200T Evaluation Kit
        |----nuclei-master.xdc  // The main .xdc constraint files
        |----xdc                                // Directory for the .xdc constraint files
        |----script                             // Directory for TCL script
        |----src                                // Directory for Verilog code for fpga top
      |----ku060                                // Directory for KU060 Evaluation Kit
        |----nuclei-master.xdc  // The main .xdc constraint files
        |----xdc                                // Directory for the .xdc constraint files
        |----script                             // Directory for TCL script
        |----src                                // Directory for Verilog code for fpga top
      |----yyy                                  // Directory for yyy Evaluation Kit
    |----Makefile       // Makefile
  |----Makefile         // Top Makefile
  |----common.mk        // Common.mk
```

Note: FPGA Flow only support vivado tool.

We suggest the version of vivado is no lower than 2018.

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. This will make sure the FPGA project is using the RTL as FPGA version (FPGA_SOURCE Macro included).
- In the top level file “nxxx_system.v”, there are SoC top level module (nxxx_soc_top) instantiated. Besides, there are just the Xilinx I/O Pads instantiated.

- In the top level file "nxxx_system.v", 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 (mcu200t, ddr200t, ku060) 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 (mcu200t, ddr200t, ku060).

13.2 Generate Bitstream (MCS format)

In *SoC for Evaluation* (page 10), it introduces the Nuclei Evaluation SoC, the SoC can be generated as FPGA Bitstream, and program into FPGA board (mcu200t, ddr200t, ku060), 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 NXXX as example):

```
// Enter into fpga directory
cd nxxx_rls_pkg/nxxx/fpga

// MCU200T Evaluation Kit (mcu200t):
make BOARD_NAME=mcu200t mcs

// DDR200T Evaluation Kit (ddr200t):
make BOARD_NAME=ddr200t mcs

// KU060 Evaluation Kit (ku060):
make BOARD_NAME=ku060 mcs


// The generated MCS format Bitstream will be under
// nxxx_rls_pkg/nxxx/fpga/gen/$FPGA_NAME/obj/system.mcs
```

13.3 Program Bitstream (MCS format) into FPGA

About how to program the Bitstream (MCS format) into the FPGA board, 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).

14 Appendix

- **Nuclei RISC-V IP Products**: https://www.nucleisys.com/product.php
- **Nuclei Spec Documentation**: https://nucleisys.com/download.php#spec
- **Nuclei RISCV Tools and Documents**: https://nucleisys.com/download.php
- **Nuclei Prebuilt Toolchain and IDE**: https://nucleisys.com/download.php#tools
- **NMSIS**: https://github.com/Nuclei-Software/NMSIS
- **Nuclei SDK**: https://github.com/Nuclei-Software/nuclei-sdk
- **Nuclei Linux SDK**: https://github.com/Nuclei-Software/nuclei-linux-sdk
- **Nuclei Software Organization in Github**: https://github.com/Nuclei-Software/
- **RISC-V MCU Organization in Github**: https://github.com/riscv-mcu/
- **RISC-V MCU Community Website**: https://www.rvmcu.com/
- **Nuclei riscv-openocd**: https://github.com/riscv-mcu/riscv-openocd
- **Nuclei riscv-gnu-toolchain**: https://github.com/riscv-mcu/riscv-gnu-toolchain