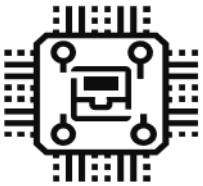


Nuclei SDK

Release 0.8.0

Nuclei

May 26, 2025

CONTENTS:

CHAPTER

ONE

OVERVIEW

1.1 Introduction

Note: Since 0.5.0 release of Nuclei SDK, we need to use Nuclei Studio >= 2023.10 or Nuclei Toolchain >=2023.10 to build and run it, see release *Changelog* (page 143).

Note: If you are looking for Nuclei N100 SDK for Nuclei 100 series CPU, please refer to https://doc.nucleisys.com/nuclei_n100_sdk

The **Nuclei Software Development Kit (SDK)** is an open-source software platform to speed up the software development of SoCs based on Nuclei Processor Cores.

This Nuclei SDK is built based on the NMSIS[1], user can access all the APIs provided by NMSIS[2] and also the APIs that provided by Nuclei SDK which mainly for on-board peripherals access such as GPIO, UART, SPI and I2C, etc.

Nuclei SDK provides a good start base for embedded developers which will help them simplify software development and improve time-to-market through well-designed software framework.

Note: To **get a pdf version** of this documentation, please click Nuclei SDK Document[3]

1.2 Design and Architecture

The Nuclei SDK general design and architecture are shown in the block diagram as below.

As *Nuclei SDK Design and Architecture Diagram* (page 2) shown, The Nuclei SDK provides the following features:

- Nuclei Core API service is built on top of NMSIS[4], so silicon vendors of Nuclei processors can easily port their SoCs to Nuclei SDK, and quickly evaluate software on their SoC.
- **NMSIS-NN** and **NMSIS-DSP** library can be also used in Nuclei SDK, and the prebuilt libraries are included in **NMSIS/Library** folder of Nuclei SDK.
- Mainly support two Nuclei Processor based SoCs, *Nuclei Eval SoC* (page 65) and *GD32VF103 SoC* (page 67)

[1] https://github.com/Nuclei-Software/NMSIS
[2] https://github.com/Nuclei-Software/NMSIS
[3] https://doc.nucleisys.com/nuclei_sdk/nucleisdk.pdf
[4] https://github.com/Nuclei-Software/NMSIS

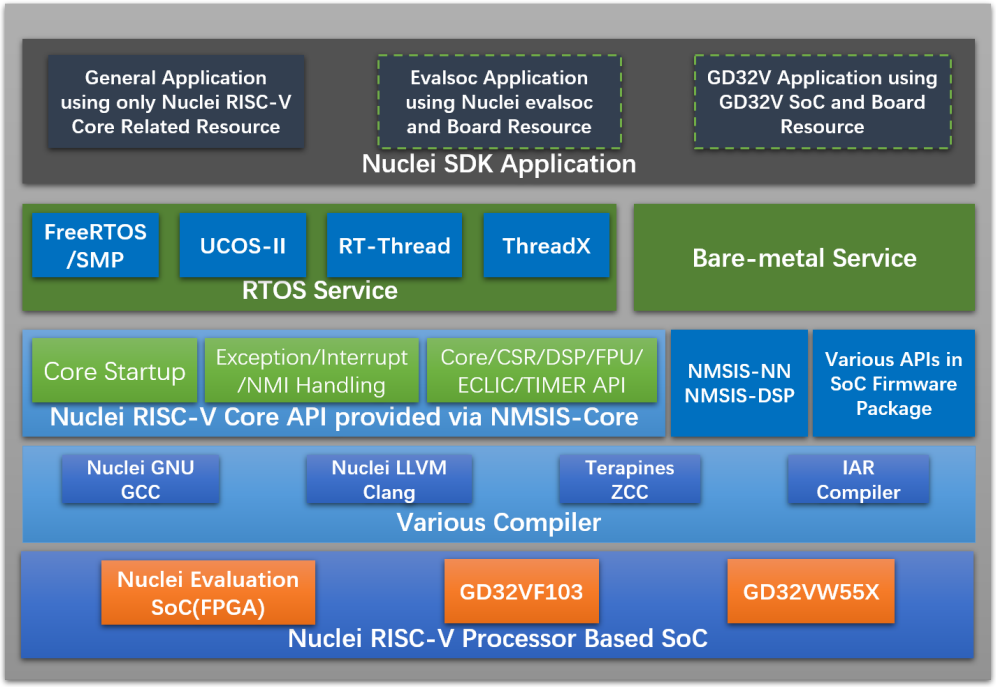


Fig. 1: Nuclei SDK Design and Architecture Diagram

- Provided realtime operation system service via *FreeRTOS* (page 87), *UCOSII* (page 88), *RT-Thread* (page 88) and *ThreadX* (page 89).
- Provided bare-metal service for embedded system software beginners and resource-limited use-cases.
- Currently Nuclei SDK doesn't define any common device APIs to access GPIO/I2C/SPI/UART devices, which still relies on the device/peripheral APIs from firmware libraries provided by various silicon vendors, such as current supported *GD32VF103 SoC* (page 67).
- Applications are logically separated into three parts:
  - **General applications for all Nuclei Processors**: In the Nuclei SDK software code, the applications provided are all general applications which can run on all Nuclei Processors, with basic UART service to provide `printf` function.
  - **Nuclei Eval SoC applications**: These applications are not included in the Nuclei SDK software code, and it is *maintained separately*, see application *Overview* (page 90), which will use resource from Nuclei Eval SoC and its evaluation boards to develop applications, which will not be compatible with different boards.
  - **GD32VF103 SoC applications**: These applications are not included in the Nuclei SDK software code, and it is *maintained separately*, which will use resource from GD32VF103 SoC and its evaluation boards to develop applications, which will not be compatible with different boards.

1.3 Get Started

Please refer to *Quick Startup* (page 5) to get started to take a try with Nuclei SDK.

1.4 Contributing

Contributing to Nuclei SDK is welcomed, if you have any issue or pull request want to open, you can take a look at *Contributing* (page 53) section.

1.5 Copyright

1.6 License

Nuclei SDK is an opensource project licensed by *Apache License 2.0* (page 177).

CHAPTER

TWO

QUICK STARTUP

2.1 Use Nuclei SDK in Nuclei Studio

Caution: If you are looking for Nuclei 100 series such as N100 support, you need to switch to **master_n100** or **nuclei_n100** branch of this repository to try it out.

If you are evaluating Nuclei CPU, in future released **nuclei_gen**, you will be able to use the generated Nuclei SDK, please see *Usage* (page 66).

For **Nuclei SDK 0.6.0** version and later ones, please use Nuclei Studio 2024.06[5] or Nuclei RISC-V Toolchain/OpenOCD/QEMU 2024.06.

From Nuclei Toolchain 2023.10[6], both gnu and llvm toolchain are provided, and toolchain prefix changed from `riscv-nuclei-elf-` to `riscv64-unknown-elf-`, and 0.5.0 SDK release will only support this `2023.10` or later toolchain.

If you want to learn about how to use Nuclei Tools(IDE,Toolchain,Qemu,OpenOCD,XlModel), please checkout https://doc.nucleisys.com/nuclei_tools.

If you want to report issues and see application note when using Nuclei Tools or Nuclei Studio, please checkout https://github.com/Nuclei-Software/nuclei-studio.

Now the nuclei-sdk **released** versions are deeply integrated with Nuclei Studio IDE via menu **RV-Tools -> NPK Package Management**, and you can directly create nuclei-sdk project in Nuclei Studio IDE Menu **File -> New Nuclei RISC-V C/C++ Project**.

You can download **Nuclei Studio IDE** from Nuclei Download Center[7], and follow Nuclei Studio and Nuclei Tools User Guide[8] to learn how to use it.

But if you want to use latest source code of Nuclei SDK, please follow the rest part of this guide to build and run using Nuclei SDK Build System in Makefile.

[5] https://github.com/Nuclei-Software/nuclei-studio/releases/tag/2024.06
[6] https://github.com/riscv-mcu/riscv-gnu-toolchain/releases/tag/nuclei-2023.10
[7] https://nucleisys.com/download.php
[8] https://doc.nucleisys.com/nuclei_tools/

2.2 Setup Tools and Environment

To start to use Nuclei SDK, you need to install the following tools:

From **2020.10** release version of Nuclei Studio, you can directly use the prebuilt tools provided in Nuclei Studio(**strongly suggested**), please following *Use Prebuilt Tools in Nuclei Studio* (page 6).

2.2.1 Use Prebuilt Tools in Nuclei Studio

Since **2020.10** release version of Nuclei Studio, you just need to download the **Nuclei Studio IDE** from Nuclei Download Center[9] for your development OS, and no need to do the following steps below, the prebuilt tools are already included.

For example:

- In Windows, if you have extracted the Nuclei Studio IDE to `D:\Software\NucleiStudio_IDE_202406`, then you can find the prebuilt tools in `D:\Software\NucleiStudio_IDE_202406\NucleiStudio\toolchain`.
- In Linux, if you have extracted the Nuclei Studio IDE to `/home/labdev/NucleiStudio_IDE_202406`, then you can find the prebuilt tools in `/home/labdev/NucleiStudio_IDE_202406/NucleiStudio/toolchain`.

You can also update tools located in the Nuclei Studio prebuilt tools `toolchain` by downloading newer version from Nuclei Tools[10] and replace it.

2.3 Get and Setup Nuclei SDK

The source code of Nuclei SDK is maintained in Github[11] and Gitee[12].

- We mainly maintained github version, and gitee version is mirrored, just for fast access in China.
- Check source code in Nuclei SDK in Github[13] or Nuclei SDK in Gitee[14] according to your network status.
- Stable version of Nuclei SDK is maintained in **master** version, if you want release version of **Nuclei SDK**, please check in Nuclei SDK Release in Github[15].

Here are the steps to clone the latest source code from Github:

- Make sure you have installed Git tool, see https://git-scm.com/download/
- Then open your terminal, and make sure git command can be accessed
- Run `git clone https://github.com/Nuclei-Software/nuclei-sdk nuclei-sdk` to clone source code into `nuclei-sdk` folder

  Note:

  - If you have no access to github.com, you can also use command `git clone https://gitee.com/Nuclei-Software/nuclei-sdk nuclei-sdk` to clone from gitee.
  - If you have no internet access, you can also use pre-downloaded `nuclei-sdk` code, and use it.

[9] https://nucleisys.com/download.php
[10] https://nucleisys.com/download.php
[11] https://github.com
[12] https://gitee.com
[13] https://github.com/Nuclei-Software/nuclei-sdk
[14] https://gitee.com/Nuclei-Software/nuclei-sdk
[15] https://github.com/Nuclei-Software/nuclei-sdk/releases

  - If the backup repo is not up to date, you can import github repo in gitee by yourself, see https://gitee.com/projects/import/url

- Create tool environment config file for Nuclei SDK

Note: If you want to use **Terapines ZCC** toolchain, you can download it from https://www.terapines.com/, or use **Nuclei Studio >= 2024.06**, a **Terapines ZCC Lite** version is integrated in **<NucleiStudio>/toolchain/zcc** folder, and you also need to add extra **PATH** into your environment, like this:

  - **Windows**: execute `set PATH=\path\to\zcc\bin;%PATH%` in windows cmd terminal before run Nuclei SDK
  - **Linux**: execute `set PATH=/path/to/zcc/bin:$PATH` in linux shell terminal before build Nuclei SDK

  - **Windows**

    If you want to use Nuclei SDK in **Windows Command Prompt** terminal, you need to create `setup_config.bat` in `nuclei-sdk` folder, and open this file your editor, and paste the following content, assuming you followed *Setup Tools and Environment* (page 6), and prebuilt tools located in `D:\Software\NucleiStudio_IDE_202406\NucleiStudio\toolchain`, otherwise please use your correct tool root path.

```
set NUCLEI_TOOL_ROOT=D:\Software\NucleiStudio_IDE_202406\NucleiStudio\
↪toolchain
```

    If you want to use Nuclei SDK in **Windows PowerShell** terminal, you need to create a `setup_config.ps1` in `nuclei-sdk` folder, and edit this file with content below if your prebuilt tools are located in `D:\Software\NucleiStudio_IDE_202406\NucleiStudio\toolchain`:

```
$NUCLEI_TOOL_ROOT="D:\Software\NucleiStudio_IDE_202406\NucleiStudio\
↪toolchain"
```

  - **Linux**

    Create `setup_config.sh` in `nuclei-sdk` folder, and open this file your editor, and paste the following content, assuming you followed *Setup Tools and Environment* (page 6) and prebuilt tools located in `/home/labdev/NucleiStudio_IDE_202406/NucleiStudio/toolchain`, otherwise please use your correct tool root path.

```
NUCLEI_TOOL_ROOT=/home/labdev/NucleiStudio_IDE_202406/NucleiStudio/toolchain
```

2.4 Build, Run and Debug Sample Application

Assume you have followed steps in *Get and Setup Nuclei SDK* (page 6) to clone source code and create files below:

- `setup_config.bat` for run in **Windows Command Prompt** terminal
- `setup_config.ps1` for run in **Windows PowerShell** terminal
- `setup_config.sh` for run in **Linux Bash** terminal

To build, run and debug application, you need to open command terminal in `nuclei-sdk` folder.

- For **Windows** users, you can open **Windows Command Prompt** terminal and cd to `nuclei-sdk` folder, then run the following commands to setup build environment for Nuclei SDK, the output will be similar as this screenshot *Setup Build Environment for Nuclei SDK in Windows Command Prompt* (page 8):

```
setup.bat
echo %PATH%
where riscv64-unknown-elf-gcc openocd make rm
make help
```

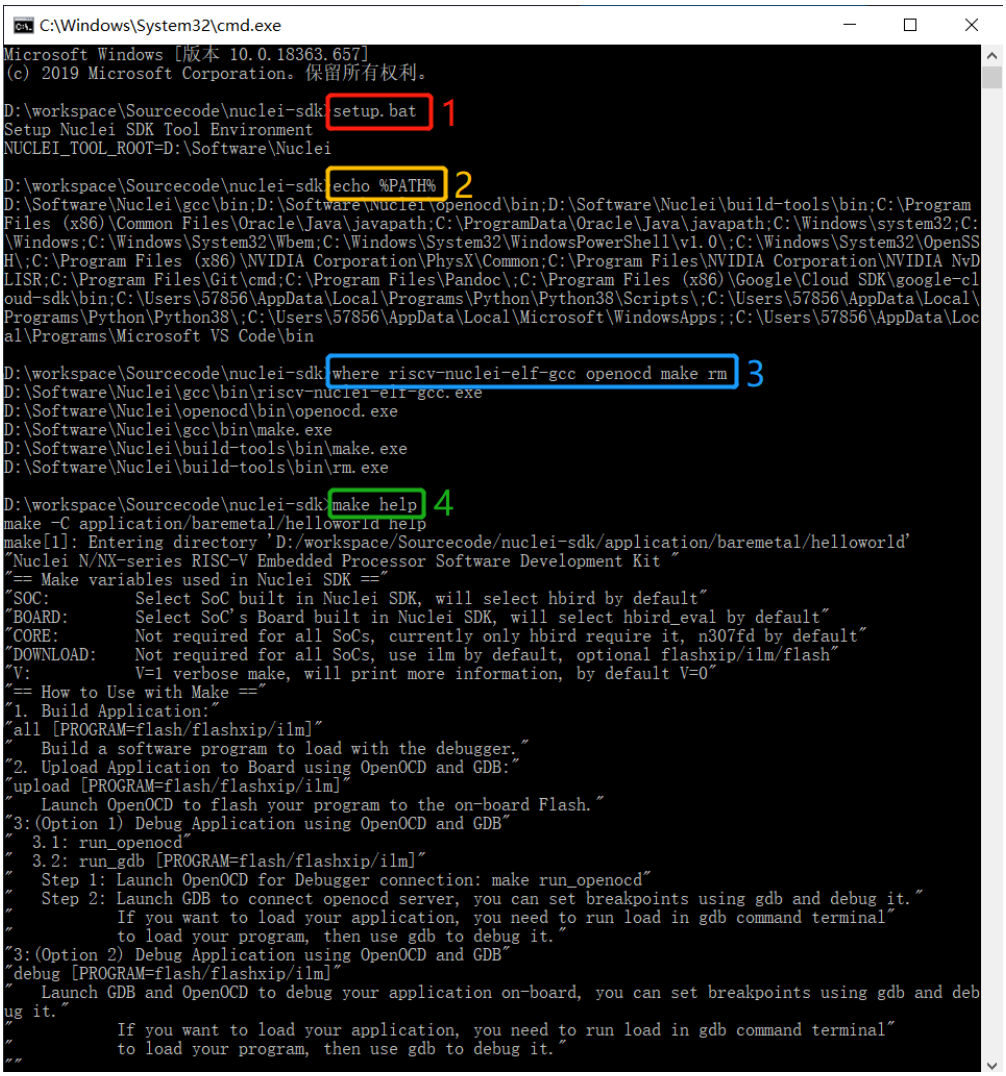


Fig. 1: Setup Build Environment for Nuclei SDK in Windows Command Prompt

- For **Linux** users, you can open **Linux Bash** terminal and cd to `nuclei-sdk` folder, then run the following commands to setup build environment for Nuclei SDK, the output will be similar as this screenshot *Setup Build Environment for Nuclei SDK in Linux Bash* (page 9):

```
source setup.sh
echo $PATH
which riscv64-unknown-elf-gcc openocd make rm
make help
```

Note:

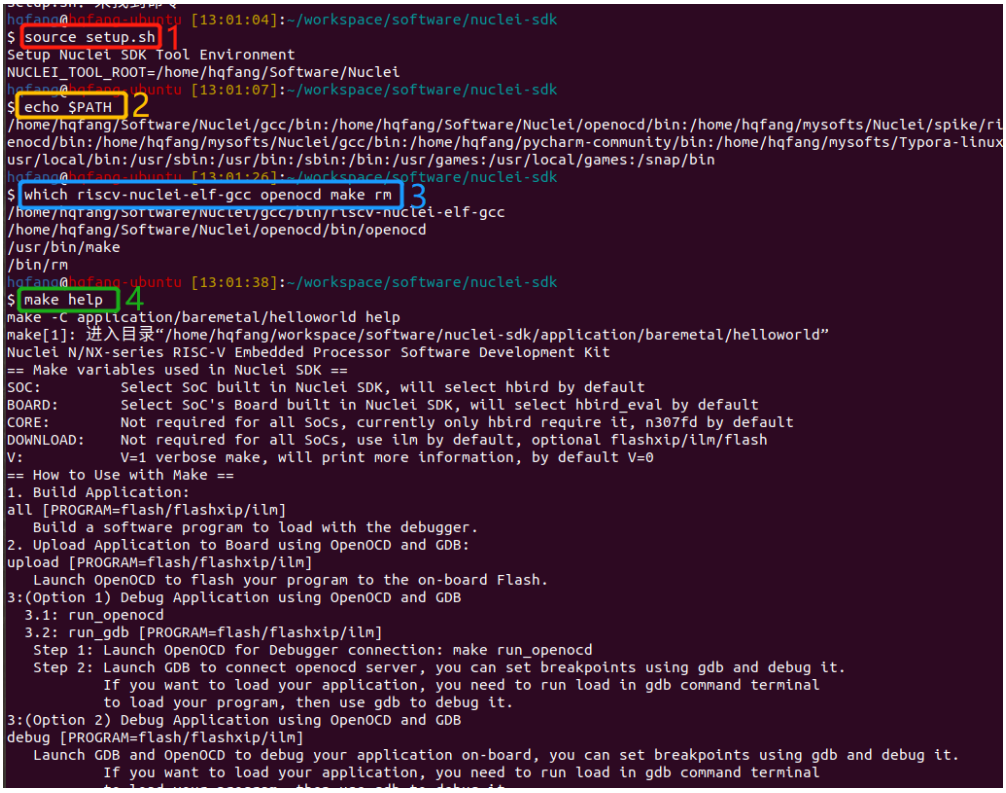


Fig. 2: Setup Build Environment for Nuclei SDK in Linux Bash

- Only first line `setup.bat` or `source setup.sh` are required before build, run or debug application. The `setup.bat` and `setup.sh` are just used to append Nuclei RISC-V GCC Toolchain, OpenOCD and Build-Tools binary paths into environment variable **PATH**
- line 2-4 are just used to check whether build environment is setup correctly, especially the **PATH** of Nuclei Tools are setup correctly, so we can use the `riscv64-unknown-elf-xxx`, `openocd`, `make` and `rm` tools
- If you know how to append Nuclei RISC-V GCC Toolchain, OpenOCD and Build-Tools binary paths to **PATH** variable in your OS environment, you can also put the downloaded Nuclei Tools as you like, and no need to run `setup.bat` or `source setup.sh`
- If you want to run in **Windows PowerShell**, please run `. .\setup.ps1` instead of `setup.bat`, and `setup_config.ps1` must be created as described in *Get and Setup Nuclei SDK* (page 6).

Here for a quick startup, this guide will take board *GD32VF103V RV-STAR Kit* (page 74) for example to demostrate how to setup hardware, build run and debug application in Windows.

The demo application, we will take `application/baremetal/helloworld` for example.

First of all, please reuse previously build environment command terminal.

Run `cd application/baremetal/helloworld` to cd the `helloworld` example folder.

2.4.1 Hardware Preparation

Please check *Board* (page 71) and find your board's page, and follow **Setup** section to setup your hardware, mainly **JTAG debugger driver setup and on-board connection setup**.

- Power on the *GD32VF103V RV-STAR Kit* (page 74) board, and use USB Type-C data cable to connect the board and your PC, make sure you have setup the JTAG driver correctly, and you can see JTAG port and serial port.
- Open a UART terminal tool such as TeraTerm in Windows[16] or Minicom in Linux[17], and minitor the serial port of the Board, the UART baudrate is *115200 bps*
- If you are building example for your own SoC and Board, please pass correct *SOC* (page 26) and *BOARD* (page 27) make variable. eg. If you SoC is `evalsoc` and Board is `nuclei_fpga_eval`, just pass `SOC=evalsoc BOARD=nuclei_fpga_eval` to make instead of the one mentioned below. If your default board for this `evalsoc` is `nuclei_fpga_eval`, then you don't need to pass `BOARD=nuclei_fpga_eval`.
- If you don't pass any SOC or BOARD via make, `evalsoc` and `nuclei_fpga_eval` are default SoC and Board.

If you just want to try on **Nuclei Evaluation SoC**, no need to pass **SOC** or **BOARD**, the default value is that, you just need to pass correct *CORE* (page 30), *ARCH_EXT* (page 31) and *DOWNLOAD* (page 29)

2.4.2 Build Application

We need to build application for this board *GD32VF103V RV-STAR Kit* (page 74) using this command line:

Note:

- If you want to run on Nuclei Evaluation SoC, see *Nuclei Eval SoC* (page 65), recommend to run *cpuinfo* (page 92)
- Since below steps are taking gd32vf103 SoC based board **gd32vf103v_rvstar** to do demostration, and when you pass `SOC=gd32vf103`, the default BOARD will be `gd32vf103v_rvstar`, so do you don't need to pass `BOARD=gd32vf103v_rvstar`
- You can check default SOC/BOARD/CORE information passed by using make target `info`, eg. `make SOC=gd32vf103 info`, for more information, please check *Makefile targets of make command* (page 25).

```
# clean application if build in other application before or build for other board
make SOC=gd32vf103 BOARD=gd32vf103v_rvstar clean
# first build choice: using full command line
make SOC=gd32vf103 BOARD=gd32vf103v_rvstar all
# second build choice: using simple command line, since when SOC=gd32vf103, default␣
↪BOARD is gd32vf103v_rvstar
make SOC=gd32vf103 all
```

Here is the sample output of this command:

```
# NOTICE: You can check this configuration whether it matched your desired configuration
Current Configuration: RISCV_ARCH=rv32imac RISCV_ABI=ilp32 SOC=gd32vf103␣
↪BOARD=gd32vf103v_rvstar CORE=n205 DOWNLOAD=flashxip
"Assembling : " ../../../SoC/gd32vf103/Common/Source/GCC/intexc_gd32vf103.S
"Assembling : " ../../../SoC/gd32vf103/Common/Source/GCC/startup_gd32vf103.S
"Compiling  : " ../../../SoC/gd32vf103/Board/gd32vf103v_rvstar/Source/gd32vf103v_rvstar.c
"Compiling  : " ../../../SoC/gd32vf103/Common/Source/Drivers/gd32vf103_adc.c
```

(continues on next page)

[16] http://ttssh2.osdn.jp/
[17] https://help.ubuntu.com/community/Minicom

(continued from previous page)

```
"Compiling  : " ../../../SoC/gd32vf103/Common/Source/Drivers/gd32vf103_bkp.c
"Compiling  : " ../../../SoC/gd32vf103/Common/Source/Drivers/gd32vf103_can.c
"Compiling  : " ../../../SoC/gd32vf103/Common/Source/Drivers/gd32vf103_crc.c
"Compiling  : " ../../../SoC/gd32vf103/Common/Source/Drivers/gd32vf103_dac.c
"Compiling  : " ../../../SoC/gd32vf103/Common/Source/Drivers/gd32vf103_dbg.c
"Compiling  : " ../../../SoC/gd32vf103/Common/Source/Drivers/gd32vf103_dma.c
"Compiling  : " ../../../SoC/gd32vf103/Common/Source/Drivers/gd32vf103_exmc.c
"Compiling  : " ../../../SoC/gd32vf103/Common/Source/Drivers/gd32vf103_exti.c
"Compiling  : " ../../../SoC/gd32vf103/Common/Source/Drivers/gd32vf103_fmc.c
"Compiling  : " ../../../SoC/gd32vf103/Common/Source/Drivers/gd32vf103_fwdgt.c
"Compiling  : " ../../../SoC/gd32vf103/Common/Source/Drivers/gd32vf103_gpio.c
"Compiling  : " ../../../SoC/gd32vf103/Common/Source/Drivers/gd32vf103_i2c.c
"Compiling  : " ../../../SoC/gd32vf103/Common/Source/Drivers/gd32vf103_pmu.c
"Compiling  : " ../../../SoC/gd32vf103/Common/Source/Drivers/gd32vf103_rcu.c
"Compiling  : " ../../../SoC/gd32vf103/Common/Source/Drivers/gd32vf103_rtc.c
"Compiling  : " ../../../SoC/gd32vf103/Common/Source/Drivers/gd32vf103_spi.c
"Compiling  : " ../../../SoC/gd32vf103/Common/Source/Drivers/gd32vf103_timer.c
"Compiling  : " ../../../SoC/gd32vf103/Common/Source/Drivers/gd32vf103_usart.c
"Compiling  : " ../../../SoC/gd32vf103/Common/Source/Drivers/gd32vf103_wwdgt.c
"Compiling  : " ../../../SoC/gd32vf103/Common/Source/Stubs/close.c
"Compiling  : " ../../../SoC/gd32vf103/Common/Source/Stubs/fstat.c
"Compiling  : " ../../../SoC/gd32vf103/Common/Source/Stubs/gettimeofday.c
"Compiling  : " ../../../SoC/gd32vf103/Common/Source/Stubs/isatty.c
"Compiling  : " ../../../SoC/gd32vf103/Common/Source/Stubs/lseek.c
"Compiling  : " ../../../SoC/gd32vf103/Common/Source/Stubs/read.c
"Compiling  : " ../../../SoC/gd32vf103/Common/Source/Stubs/sbrk.c
"Compiling  : " ../../../SoC/gd32vf103/Common/Source/Stubs/write.c
"Compiling  : " ../../../SoC/gd32vf103/Common/Source/gd32vf103_soc.c
"Compiling  : " ../../../SoC/gd32vf103/Common/Source/system_gd32vf103.c
"Compiling  : " hello_world.c
"Linking    : " hello_world.elf
text    data     bss     dec     hex filename
13022     112    2290   15424    3c40 hello_world.elf
```

As you can see, that when the application is built successfully, the elf will be generated and will also print the size information of the `hello_world.elf`.

Note:

- In order to make sure that there is no application build before, you can run `make SOC=gd32vf103 BOARD=gd32vf103v_rvstar clean` to clean previously built objects and build dependency files.
- About the make variable or option(**SOC**, **BOARD**) passed to make command, please refer to *Build System based on Makefile* (page 19).

2.4.3 Run Application

If the application is built successfully for this board *GD32VF103V RV-STAR Kit* (page 74), then you can run it using this command line:

```
make SOC=gd32vf103 BOARD=gd32vf103v_rvstar upload
```

Here is the sample output of this command:

```
"Download and run hello_world.elf"
riscv64-unknown-elf-gdb hello_world.elf -ex "set remotetimeout 240" \
        -ex "target remote | openocd -c \"gdb_port pipe; log_output openocd.log\" -f ../.
↪./../SoC/gd32vf103/Board/gd32vf103v_rvstar/openocd_gd32vf103.cfg" \
        --batch -ex "monitor halt" -ex "monitor halt" -ex "monitor flash protect 0 0␣
↪last off" -ex "load" -ex "monitor resume" -ex "monitor shutdown" -ex "quit"
D:\Software\Nuclei\gcc\bin\riscv64-unknown-elf-gdb.exe: warning: Couldn't determine a␣
↪path for the index cache directory.
Nuclei OpenOCD, 64-bit Open On-Chip Debugger 0.10.0+dev-00014-g0eae03214 (2019-12-12-
↪07:43)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
_start0800 () at ../../../SoC/gd32vf103/Common/Source/GCC/startup_gd32vf103.S:359
359         j 1b
cleared protection for sectors 0 through 127 on flash bank 0

Loading section .init, size 0x266 lma 0x8000000
Loading section .text, size 0x2e9c lma 0x8000280
Loading section .rodata, size 0x1f0 lma 0x8003120
Loading section .data, size 0x70 lma 0x8003310
Start address 0x800015c, load size 13154
Transfer rate: 7 KB/sec, 3288 bytes/write.
shutdown command invoked
A debugging session is active.

        Inferior 1 [Remote target] will be detached.

Quit anyway? (y or n) [answered Y; input not from terminal]
[Inferior 1 (Remote target) detached]
```

As you can see the application is uploaded successfully using `openocd` and `gdb`, then you can check the output in your UART terminal, see *Nuclei SDK Hello World Application UART Output* (page 13).

2.4.4 Debug Application

If the application is built successfully for this board *GD32VF103V RV-STAR Kit* (page 74), then you can debug it using this command line:

```
make SOC=gd32vf103 BOARD=gd32vf103v_rvstar debug
```

1. The program is not loaded automatically when you enter to debug state, just in case you want to debug the program running on the board.

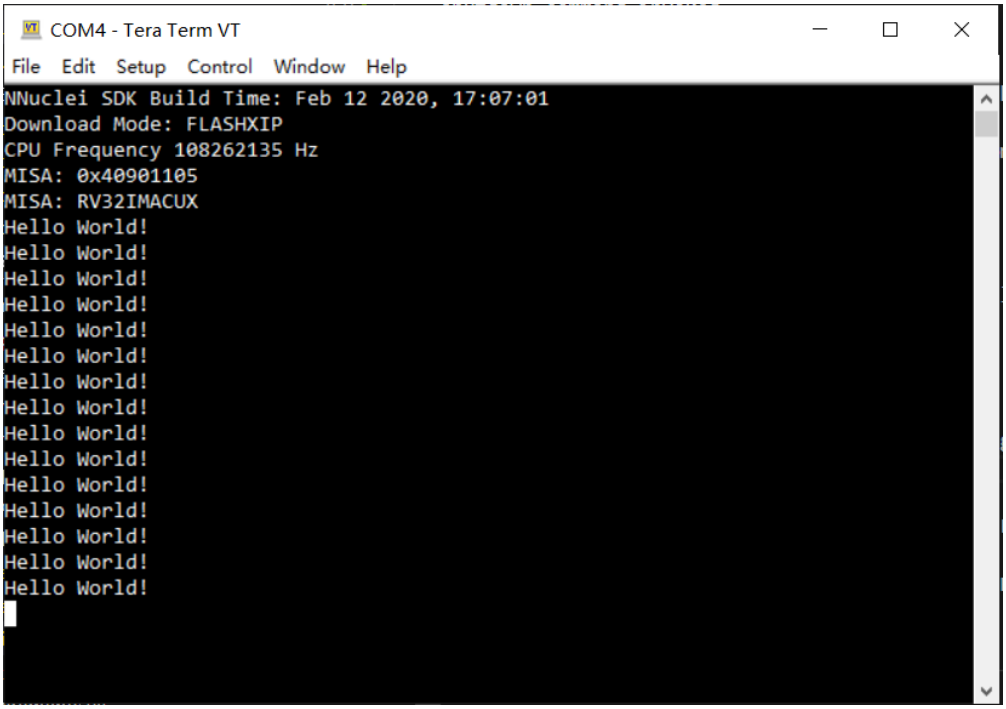


Fig. 3: Nuclei SDK Hello World Application UART Output

```
"Download and debug hello_world.elf"
riscv64-unknown-elf-gdb hello_world.elf -ex "set remotetimeout 240" \
        -ex "target remote | openocd -c \"gdb_port pipe; log_output openocd.log\" -
↪f ../../../SoC/gd32vf103/Board/gd32vf103v_rvstar/openocd_gd32vf103.cfg"
D:\Software\Nuclei\gcc\bin\riscv64-unknown-elf-gdb.exe: warning: Couldn't determine␣
↪a path for the index cache directory.
GNU gdb (GDB) 8.3.0.20190516-git
Copyright (C) 2019 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "--host=i686-w64-mingw32 --target=riscv64-unknown-elf".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.
--Type <RET> for more, q to quit, c to continue without paging--

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from hello_world.elf...
Remote debugging using | openocd -c \"gdb_port pipe; log_output openocd.log\" -f ../
↪../../SoC/gd32vf103/Board/gd32vf103v_rvstar/openocd_gd32vf103.cfg
Nuclei OpenOCD, 64-bit Open On-Chip Debugger 0.10.0+dev-00014-g0eae03214 (2019-12-
↪12-07:43)
```

(continues on next page)

(continued from previous page)

```
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
_start0800 () at ../../../SoC/gd32vf103/Common/Source/GCC/startup_gd32vf103.S:359
359         j 1b
```

2. If you want to load the built application, you can type `load` to load the application.

```
(gdb) load
Loading section .init, size 0x266 lma 0x8000000
Loading section .text, size 0x2e9c lma 0x8000280
Loading section .rodata, size 0x1f0 lma 0x8003120
Loading section .data, size 0x70 lma 0x8003310
Start address 0x800015c, load size 13154
Transfer rate: 7 KB/sec, 3288 bytes/write.
```

3. If you want to set a breakpoint at *main*, then you can type `b main` to set a breakpoint.

```
(gdb) b main
Breakpoint 1 at 0x8001b04: file hello_world.c, line 85.
```

4. If you want to set more breakpoints, you can do as you like.
5. Then you can type `c`, then the program will stop at **main**

```
(gdb) c
Continuing.
Note: automatically using hardware breakpoints for read-only addresses.

Breakpoint 1, main () at hello_world.c:85
85          srand(__get_rv_cycle()  | __get_rv_instret() | __RV_CSR_READ(CSR_
↪MCYCLE));
```

6. Then you can step it using `n` (short of next) or `s` (short of step)

```
(gdb) n
86          uint32_t rval = rand();
(gdb) n
87          rv_csr_t misa = __RV_CSR_READ(CSR_MISA);
(gdb) s
89          printf("MISA: 0x%lx\r\n", misa);
(gdb) n
90          print_misa();
(gdb) n
92          printf("Hello World!\r\n");
(gdb) n
93          printf("Hello World!\r\n");
```

7. If you want to quit debugging, then you can press `CTRL - c`, and type `q` to quit debugging.

```
(gdb) Quit
(gdb) q
A debugging session is active.
```

(continues on next page)

(continued from previous page)

```
        Inferior 1 [Remote target] will be detached.

Quit anyway? (y or n) y
Detaching from program: D:\workspace\Sourcecode\nuclei-sdk\application\baremetal\
↪helloworld\hello_world.elf, Remote target
Ending remote debugging.
[Inferior 1 (Remote target) detached]
```

Note:

- More about how to debug using gdb, you can refer to the GDB User Manual[18].
- If you want to debug using Nuclei Studio, you can open Nuclei Studio, and create a debug configuration, and choose the application elf, and download and debug in IDE.

2.5 Create helloworld Application

If you want to create your own `helloworld` application, it is also very easy.

There are several ways to achieve it, see as below:

- **Method 1:** You can find a most similar sample application folder and copy it, such as `application/baremetal/helloworld`, you can copy and rename it as `application/baremetal/hello`
  - Open the `Makefile` in `application/baremetal/hello`
    1. Change `TARGET = hello_world` to `TARGET = hello`
  - Open the `hello_world.c` in `application/baremetal/hello`, and replace the content using code below:

```c
// See LICENSE for license details.
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include "nuclei_sdk_soc.h"

int main(void)
{
    printf("Hello World from Nuclei RISC-V Processor!\r\n");
    return 0;
}
```

  - Save all the changes, and then you can follow the steps described in *Build, Run and Debug Sample Application* (page 7) to run or debug this new application.
- **Method 2:** You can also do it from scratch, with just create simple `Makefile` and `main.c`
  - Create new folder named `hello` in `application/baremetal`
  - Create two files named `Makefile` and `main.c`
  - Open `Makefile` and edit the content as below:

[18] https://www.gnu.org/software/gdb/documentation/

```
TARGET = hello

NUCLEI_SDK_ROOT = ../../..

SRCDIRS = .

INCDIRS = .

include $(NUCLEI_SDK_ROOT)/Build/Makefile.base
```

- Open `main.c` and edit the content as below:

```
// See LICENSE for license details.
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include "nuclei_sdk_soc.h"

int main(void)
{
    printf("Hello World from Nuclei RISC-V Processor!\r\n");
    return 0;
}
```

- Save all the changes, and then you can follow the steps described in *Build, Run and Debug Sample Application* (page 7) to run or debug this new application.

Note:

- If your are looking for how to run for other boards, please ref to *Board* (page 71).
- Please refer to *Application Development* (page 49) and *Build System based on Makefile* (page 19) for more information.
- If you want to access SoC related APIs, please use `nuclei_sdk_soc.h` header file.
- If you want to access SoC and board related APIs, please use `nuclei_sdk_hal.h` header file.
- For simplified application development, you can use `nuclei_sdk_hal.h` directly.

2.6 Advanced Usage

For more advanced usage, please follow the items as below:

- Click *Design and Architecture* (page 59) to learn about Nuclei SDK Design and Architecture, Board and SoC support documentation.
- Click *Developer Guide* (page 19) to learn about Nuclei SDK Build System and Application Development.
- Click *Application* (page 90) to learn about each application usage and expected output.

Note:

- If you met some issues in using this guide, please check *FAQ* (page 171), if still not solved, please *Submit your issue* (page 57).
- If you are trying to **develop Nuclei SDK application in IDE**, now you have three choices:
  1. **Recommended**: Since Nuclei Studio 2020.08, Nuclei SDK will be deeply integrated with Nuclei Studio IDE, you can easily create a Nuclei SDK Project in Nuclei Studio through IDE Project Wizard, and easily configure selected Nuclei SDK project using SDK Configuration Tool, for more details, please click Nuclei Tools[19] to download Nuclei Studio IDE, and refer to the Nuclei Studio and Nuclei Tools User Guide[20] for how to use it.
  2. Now **Terapines ZCC Lite** is deeply integrated in **Nuclei Studio >= 2024.06**, so you just need to follow *Get and Setup Nuclei SDK* (page 6) to setup PATH for Terapines ZCC, and in Nuclei SDK, you can just pass **TOOCHAIN=terapines** during make to take a try with Terapines ZCC. From 0.7.0 release, you can create project in Nuclei Studio >= 2024.06 using Terapines ZCC, see Using Terapines ZCC Toolchain in Nuclei Studio[21].
  3. You can take a try using IAR workbench, we provided prebuilt projects directly in Nuclei SDK, just check the ideprojects/iar/README.md[22] to learn about it.
  4. You can take a try using Segger embedded studio, we provided prebuilt projects using Nuclei SDK release version, click Segger embedded studio projects for Nuclei SDK[23] to learn about it
  5. You can also take a try with the Cross-platform PlatformIO IDE, we provided our Nuclei platform and Nuclei SDK release version in PlatformIO, click Platform Nuclei in PlatformIO[24] to learn more about it, or you can visit Light on onboard LED of RVSTAR board using PlatformIO(Chinese)[25] to play with PlatformIO for Nuclei.
  6. You can also use source code in Nuclei SDK as base, and easily integrate with other IDE tools, such as ZStudio IDE[26], Compiler IDE[27] and others.

[19] https://nucleisys.com/download.php
[20] https://doc.nucleisys.com/nuclei_tools/
[21] https://1nfinite.ai/t/nuclei-studio-2024-06-ide-terapines-zcc/113
[22] https://github.com/Nuclei-Software/nuclei-sdk/blob/master/ideprojects/iar/README.md
[23] https://github.com/riscv-mcu/ses_nuclei_sdk_projects
[24] https://github.com/Nuclei-Software/platform-nuclei
[25] https://www.rvmcu.com/community-topic-id-310.html
[26] https://1nfinite.ai/t/zstudio-ide-risc-v/71
[27] https://www.compiler-dev.com/

CHAPTER

THREE

DEVELOPER GUIDE

3.1 Code Style

In Nuclei SDK, we use EditorConfig[28] to maintain our development coding styles and astyle[29] tool to format our source code.

- Our editorconfig file[30] is maintained in the root directory of Nuclei SDK, called `.editorconfig`.
- Our astyle option file is maintained in the root directory of Nuclei SDK, called `.astylerc`.

For example, if you want to format your applicaton code(*.c*/.h) located in `application/baremetal/demo_timer`, you can run the following command:

```
# make sure astyle is present in PATH
which astyle
# format code
astyle --options=.astylerc --recursive application/baremetal/demo_timer/*.c,*.h
```

You can install editorconfig plugins for your editor, see https://editorconfig.org/#download.

We use doxygen[31] to comment C/C++ source code.

3.2 Build System based on Makefile

Nuclei SDK's build system is based on Makefile, user can build, run ordebug application in Windows and Linux.

3.2.1 Makefile Structure

Nuclei SDK's Makefiles mainly placed in **<NUCLEI_SDK_ROOT>/Build** directory and an extra *Makefile* located in **<NUCLEI_SDK_ROOT>/Makefile**.

This extra **<NUCLEI_SDK_ROOT>/Makefile** introduce a new Make variable called **PROGRAM** to provide the ability to build or run application in **<NUCLEI_SDK_ROOT>**.

For example, if you want to *rebuild and upload* application **application/baremetal/timer_test**, you can run `make PROGRAM=application/baremetal/timer_test clean upload` to achieve it.

The **<NUCLEI_SDK_ROOT>/Build** directory content list as below:

[28] https://editorconfig.org/
[29] http://astyle.sourceforge.net/
[30] https://github.com/Nuclei-Software/nuclei-sdk/tree/master/.editorconfig
[31] http://www.doxygen.nl/manual/docblocks.html

```
gmsl/
toolchain/
Makefile.base
Makefile.conf
Makefile.core
Makefile.components
Makefile.files
Makefile.global  -> Created by user
Makefile.misc
Makefile.rtos
Makefile.rules
Makefile.soc
```

The file or directory is used explained as below:

Makefile.base

This **Makefile.base** file is used as Nuclei SDK build system entry file, application's Makefile need to include this file to use all the features of Nuclei SDK build system.

It will expose Make variables or options such as **BOARD** or **SOC** passed by `make` command, click *Makefile variables passed by make command* (page 26) to learn more.

This file will include optional *Makefile.global* (page 24) and *Makefile.local* (page 25) which allow user to set custom global Makefile configurations and local application Makefile configurations.

This file will include the following makefiles:

- *gmsl* (page 20): additional library functions provided via gmsl
- *toolchain* (page 21): additional library functions provided via gmsl
- *Makefile.misc* (page 21): misc functions and OS check helpers
- *Makefile.conf* (page 21): main Makefile configuration entry
- *Makefile.rules* (page 21): make rules of this build system

gmsl

The **gmsl** directory consist of the GNU Make Standard Library (GMSL)[32], which is an a library of functions to be used with GNU Make's $(call) that provides functionality not available in standard GNU Make.

We use this **gmsl** tool to make sure we help us achieve some linux command which is only supported in Linux.

[32] http://sourceforge.net/projects/gmsl/

toolchain

The **toolchain** directory contains different toolchain support makefiles, such as Nuclei GNU toolchain, Nuclei LLVM toolchain and Terapines toolchain, if you want to add a different toolchain support, you also need to add a new toolchain makefile in it, you can refer to existing ones.

Since different toolchain support is added, in application Makefile, if your toolchain options are not compatiable with others, to provide a compatiable application for different toolchain, we recommend you to add `toolchain_$(TOOLCHAIN).mk` file in your application folder, and in application Makefile include this file, you can refer to `application/baremetal/benchmark/coremark` to see example usage.

Makefile.misc

This **Makefile.misc** file mainly provide these functions:

- Define **get_csrcs**, **get_asmsrcs**, **get_cxxsrcs** and **check_item_exist** make functions
  - **get_csrcs**: Function to get `*.c or *.C` source files from a list of directories, no ability to do recursive match. e.g. `$(call get_csrcs, csrc csrc/abc)` will return c source files in `csrc` and `csrc/abc` directories.
  - **get_asmsrcs**: Function to get `*.s or *.S` source files from a list of directories, no ability to do recursive match. e.g. `$(call get_asmsrcs, asmsrc asmsrc/abc)` will return asm source files in `asmsrc` and `asmsrc/abc` directories.
  - **get_cxxsrcs**: Function to get `*.cpp or *.CPP` source files from a list of directories, no ability to do recursive match. e.g. `$(call get_cxxsrcs, cppsrc cppsrc/abc)` will return cpp source files in `cppsrc` and `cppsrc/abc` directories.
  - **check_item_exist**: Function to check if item existed in a set of items. e.g. `$(call check_item_exist, flash, flash ilm flashxip)` will check `flash` whether existed in `flash ilm flashxip`, if existed, return `flash`, otherwise return empty.
- Check and define OS related functions, and also a set of trace print functions.

Makefile.conf

This **Makefile.conf** file will define the following items:

- Toolchain related variables used during compiling
- Debug related variables
- Include *Makefile.files* (page 22) and *Makefile.rtos* (page 23)
- Collect all the C/C++/ASM compiling and link options

Makefile.rules

This **Makefile.rules** file will do the following things:

- Collect all the sources during compiling
- Define all the rules used for building, uploading and debugging
- Print help message for build system

Makefile.files

This **Makefile.files** file will do the following things:

- Define common C/C++/ASM source and include directories
- Define common C/C++/ASM macros

Makefile.soc

This **Makefile.soc** will include valid makefiles located in **<NUCLEI_SDK_ROOT>/SoC/<SOC>/build.mk** according to the *SOC* (page 26) makefile variable setting.

It will define the following items:

- **DOWNLOAD** and **CORE** variables
  - For *Nuclei Eval SoC* (page 65), we can support all the modes defined in *DOWNLOAD* (page 29), and **CORE** list defined in *Makefile.core* (page 24)
  - For *GD32VF103 SoC* (page 67), The **CORE** is fixed to N205, since it is a real SoC chip, and only **FlashXIP** download mode is supported
- Linker script used according to the **DOWNLOAD** mode settings
- OpenOCD debug configuration file used for the SoC and Board
- Some extra compiling or debugging options

A valid SoC should be organized like this, take `evalsoc` as example:

```
SoC/evalsoc
├── Board
│   └── nuclei_fpga_eval
│       ├── Include
│       │   ├── board_nuclei_fpga_eval.h
│       │   └── nuclei_sdk_hal.h
│       ├── Source
│       │   ├── IAR
│       │   └── GCC
│       └── openocd_evalsoc.cfg
├── build.mk
└── Common
    ├── Include
    │   ├── evalsoc.h
    │   ├── ... ...
    │   ├── evalsoc_uart.h
    │   ├── nuclei_sdk_soc.h
    │   └── system_evalsoc.h
    └── Source
        ├── Drivers
        │   ├── ... ...
        │   └── evalsoc_uart.c
        ├── GCC
        │   ├── intexc_evalsoc.S
        │   ├── intexc_evalsoc_s.S
        │   └── startup_evalsoc.S
        ├── IAR
```

(continues on next page)

(continued from previous page)

```
        │   ├── intexc_evalsoc.S
        │   ├── intexc_evalsoc_s.S
        │   └── startup_evalsoc.c
        ├── Stubs
        │   ├── newlib
        │   ├── libncrt
        │   └── iardlib
        ├── evalsoc_common.c
        └── system_evalsoc.c
```

Makefile.rtos

This **Makefile.rtos** will include **<NUCLEI_SDK_ROOT>/OS/<RTOS>/build.mk** according to our *RTOS* (page 37) variable.

A valid rtos should be organized like this, take `UCOSII` as example:

```
OS/UCOSII/
├── arch
├── build.mk
├── license.txt
├── readme.md
└── source
```

If no *RTOS* (page 37) is chosen, then RTOS code will not be included during compiling, user will develop baremetal application.

If **FreeRTOS**, **UCOSII** or **RTThread** RTOS is chosen, then FreeRTOS UCOSII, or RTThread source code will be included during compiling, and extra compiler option `-DRTOS_$(RTOS_UPPER)` will be passed, then user can develop RTOS application.

For example, if `FreeRTOS` is selected, then `-DRTOS_FREERTOS` compiler option will be passed.

Makefile.components

This **Makefile.components** will include `build.mk` Makefiles of selected components defined via makefile variable *MIDDLEWARE* (page 38), the Makefiles are placed in the sub-folders of **<NUCLEI_SDK_ROOT>/Components/**.

A valid middleware component should be organized like this, take `fatfs` as example :

```
Components/fatfs/
├── build.mk
├── documents
├── LICENSE.txt
└── source
```

For example, if there are two valid middleware components in **<NUCLEI_SDK_ROOT>/Components/**, called `fatfs` and `tjpgd`, and you want to use them in your application, then you can set `MIDDLEWARE` like this `MIDDLEWARE := fatfs tjpgd`, then the application will include these two middlewares into build process.

Makefile.core

This **Makefile.core** is used to define the RISC-V ARCH and ABI used during compiling of the CORE list supported.

If you want to add a new **CORE**, you need to add a new line before **SUPPORTED_CORES**, and append the new **CORE** to **SUPPORTED_CORES**.

For example, if you want to add a new **CORE** called **n308**, and the **n308**'s **ARCH** and **ABI** are `rv32imafdc` and `ilp32d`, then you can add a new line like this `N308_CORE_ARCH_ABI = rv32imafdc ilp32d`, and append **n308** to **SUPPORTED_CORES** like this `SUPPORTED_CORES = n201 n201e n203 n203e n308 nx600`

Note:

- The appended new **CORE** need to lower-case, e.g. *n308*
- The new defined variable **N308_CORE_ARCH_ABI** need to be all upper-case.

Makefile.global

This **Makefile.global** file is an optional file, and will not be tracked by git, user can create own **Makefile.global** in **<NUCLEI_SDK_ROOT>/Build** directory.

In this file, user can define custom **SOC**, **BOARD**, **DOWNLOAD** options to overwrite the default configuration.

For example, if you will use only the *GD32VF103V RV-STAR Kit* (page 74), you can create the **<NUCLEI_SDK_ROOT>/Build/Makefile.global** as below:

```
SOC ?= gd32vf103
BOARD ?= gd32vf103v_rvstar
DOWNLOAD ?= flashxip
```

Note:

- If you add above file, then you can build, run, debug application without passing **SOC**, **BOARD** and **DOWNLOAD** variables using make command for *GD32VF103V RV-STAR Kit* (page 74) board, e.g.
  - Build and run application for *GD32VF103V RV-STAR Kit* (page 74): `make run`
  - Debug application for *GD32VF103V RV-STAR Kit* (page 74): `make debug`
- The *GD32VF103V RV-STAR Kit* (page 74) only support `FlashXIP` download mode.
- If you create the **Makefile.global** like above sample code, you will also be able to use Nuclei SDK build system as usually, it will only change the default **SOC**, **BOARD** and **DOWNLOAD**, but you can still override the default variable using make command, such as `make SOC=evalsoc BOARD=nuclei_fpga_eval DOWNLOAD=ilm`

Makefile.local

As the *Makefile.global* (page 24) is used to override the default Makefile configurations, and the **Makefile.local** is used to override application level Makefile configurations, and also this file will not be tracked by git.

User can create `Makefile.local` file in any of the application folder, placed together with the application Makefile, for example, you can create `Makefile.local` in `application/baremetal/helloworld` to override default make configuration for this **helloworld** application.

If you want to change the default board for **helloworld** to use *GD32VF103V RV-STAR Kit* (page 74), you can create `application/baremetal/helloworld/Makefile.local` as below:

```
SOC ?= gd32vf103
BOARD ?= gd32vf103v_rvstar
DOWNLOAD ?= flashxip
```

Note:

- This local make configuration will override global and default make configuration.
- If you just want to change only some applications' makefile configuration, you can add and update `Makefile.local` for those applications.

3.2.2 Makefile targets of make command

Here is a list of the *Make targets supported by Nuclei SDK Build System* (page 25).

Table 1: Make targets supported by Nuclei SDK Build System

target	description
help	display help message of Nuclei SDK build system
info	display selected configuration information
showflags	display asm/c/cxx/ld flags and other info
showtoolver	display toolchain/qemu/openocd version
all	build application with selected configuration
clean	clean application with selected configuration
dasm	build and dissemble application with selected configuration
bin	build and generate application binary with selected configuration
upload	build and upload application with selected configuration
run_openocd	run openocd server with selected configuration, and wait for gdb at port specified by $(GDB_PORT)
run_gdb	build and start gdb process with selected configuration, and connect to localhost:$(GDB_PORT)
debug	build and debug application with selected configuration
run_qemu	run application on Nuclei Qemu Evalsoc[33] machine with selected configuration
run_xlspike	**internal used only**, run application on xlspike with selected configuration
run_xlmodel	run application on Nuclei Near Cycle Model[34] with selected configuration
size	show program size

Note:

[33] https://doc.nucleisys.com/nuclei_tools/qemu/intro.html
[34] https://doc.nucleisys.com/nuclei_tools/xlmodel/intro.html

- The selected configuration is controlled by *Makefile variables passed by make command* (page 26)
- For `run_openocd` and `run_gdb` target, if you want to change a new gdb port, you can pass the variable *GDB_PORT* (page 35)
- For `run_qemu`, only `SOC=evalsoc` supported, when do this target, you can pass `SIMU=qemu` to support auto-exit, project recompiling is required.
- For `run_xlspike`, only `SOC=evalsoc` supported, when do this target, you can pass `SIMU=xlspike` to support auto-exit, project recompiling is required.

3.2.3 Makefile variables passed by make command

In Nuclei SDK build system, we exposed the following Makefile variables which can be passed via make command.

- *SOC* (page 26)
- *BOARD* (page 27)
- *VARIANT* (page 28)
- *TOOLCHAIN* (page 28)
- *DOWNLOAD* (page 29)
- *CORE* (page 30)
- *ARCH_EXT* (page 31)
- *CPU_SERIES* (page 33)
- *SIMULATION* (page 34)
- *SEMIHOST* (page 33)
- *GDB_PORT* (page 35)
- *V* (page 36)
- *SILENT* (page 36)

Note:

- These variables can also be used and defined in application Makefile
- If you just want to fix your running board of your application, you can just define these variables in application Makefile, if defined, then you can simply use `make clean`, `make upload` or `make debug`, etc.

SOC

SOC variable is used to declare which SoC is used in application during compiling.

evalsoc is the default SoC, if no **SOC** passed or environment variable set, you can check default settings by run `make info`, it will will show default settings without any overriding make variable.

You can easily find the supported SoCs in the **<NUCLEI_SDK_ROOT>/SoC** directory.

Currently we support the following SoCs, see *Supported SoCs* (page 27).

Table 2: Supported SoCs

SOC	Reference
gd32vf103	*GD32VF103 SoC* (page 67)
gd32vw55x	*GD32VW55x SoC* (page 70)
evalsoc	*Nuclei Eval SoC* (page 65)

Note: If you are our SoC subsystem customer, in the SDK delivered to you, you can find your soc name in this **<NUCLEI_SDK_ROOT>/SoC** directory, take `gd32vf103` SoC as example, when `SOC=gd32vf103`, the SoC source code in **<NUCLEI_SDK_ROOT>/SoC/gd32vf103/Common** will be used.

This documentation just document the open source version of Nuclei SDK's supported SOC and Board.

BOARD

BOARD variable is used to declare which Board is used in application during compiling.

The **BOARD** variable should match the supported boards of chosen **SOC**. You can easily find the supported Boards in the **<NUCLEI_SDK_ROOT>/<SOC>/Board/** directory.

- *Supported Boards when SOC=gd32vf103* (page 27)
- *Supported Boards when SOC=evalsoc* (page 27)
- *Supported Boards when SOC=g32vw55x* (page 27)

Currently we support the following SoCs.

Table 3: Supported Boards when SOC=gd32vf103

BOARD	Reference
gd32vf103v_rvstar	*GD32VF103V RV-STAR Kit* (page 74)
gd32vf103c_dlink	*GD32VF103C DLink Debugger* (page 80)
gd32vf103v_eval	*GD32VF103V Evaluation Kit* (page 76)
gd32vf103c_longan_nano	*Sipeed Longan Nano* (page 77)
gd32vf103c_t_display	*Sipeed Longan Nano* (page 77)
gd32vw553h_eval	*GD32VW553H Evaluation Kit* (page 83)

Table 4: Supported Boards when SOC=evalsoc

BOARD	Reference
nuclei_fpga_eval	*Nuclei FPGA Evaluation Kit* (page 71)

Table 5: Supported Boards when SOC=g32vw55x

BOARD	Reference
gd32vw553h_eval	*GD32VW553H Evaluation Kit* (page 83)

Note:

- If you only specify **SOC** variable in make command, it will use default **BOARD** and **CORE** option defined in **<NUCLEI_SDK_ROOT>/SoC/<SOC>/build.mk**
- If you are our SoC subsystem customer, in the SDK delivered to you, you can check the board supported list in **<NUCLEI_SDK_ROOT>/<SOC>/Board/**, take `SOC=gd32vf103 BOARD=gd32vf103v_rvstar` as example, the board source code located **<NUCLEI_SDK_ROOT>/gd32vf103/Board/gd32vf103v_rvstar** will be used.

VARIANT

VARIANT variable is used to declare which variant of board is used in application during compiling.

It might only affect on only small piece of board, and this is SoC and Board dependent.

This variable only affect the selected board or soc, and it is target dependent.

TOOLCHAIN

Note: This variable is added in 0.5.0 release.

This variable is used to select different toolchain to compile application. Currently we support 3 toolchain in Nuclei SDK.

- **nuclei_gnu**: default, it will choose nuclei gnu toolchain, distributed with Nuclei Toolchain, see `Build/toolchain/nuclei_gnu.mk`.
- **nuclei_llvm**: supported, nuclei customized extensions not yet supported, distributed with Nuclei Toolchain, see `Build/toolchain/nuclei_llvm.mk`.
- **terapines**: supported, see `Build/toolchain/nuclei_gnu.mk`, and it depends on the toolchain vendor about the supported extensions, if you want to take a try with it, just visit https://www.terapines.com/ and request an terapines toolchain evaluation, or you can take a try with Nuclei Studio >= 2024.06.

To learn about how to use Nuclei RISC-V Toolchain, you can refer to https://doc.nucleisys.com/nuclei_tools/

If you want to add support for your own toolchain which is based on gcc/llvm, you can refer to above toolchain support makefile.

For **nuclei_gnu/nuclei_llvm** toolchain both newlib and libncrt library are supported, but **nuclei_llvm** toolchain multilib selection mechanism is not as good as gnu toolchain, you need to take care of the arch isa string order, please see `riscv64-unknown-unknown-elf-clang -v` output for supported multilib and its isa string order.

And **IAR compiler** support is also done in Nuclei SDK, you can take a try with it via ideprojects/iar[35] folder provided prebuilt ide projects.

If you want to use old Nuclei GNU Toolchain <= 2022.12 in Nuclei SDK 0.5.0, you need to pass extra `COMPILE_PREFIX=riscv-nuclei-elf-` when build any application, such as `make CORE=n300fd COMPILE_PREFIX=riscv-nuclei-elf- STDCLIB=libncrt_small clean all`, but this is not recommended, and will be deprecated in future any time.

From 0.8.0, **COMPILE_PREFIX** are supported by `nuclei_gnu` and `nuclei_llvm`, but for `nuclei_llvm`, `llvm-ar` and `llvm-size` are not set by this **COMPILE_PREFIX**.

[35] https://github.com/Nuclei-Software/nuclei-sdk/blob/master/ideprojects/iar/README.md

DOWNLOAD

DOWNLOAD variable is used to declare the download mode of the application, currently it has these modes supported as described in table *Supported download modes* (page 29)

Table 6: Supported download modes

DOWNLOAD	Description
ilm	Program will be downloaded into ilm/ram and run directly in ilm/ram, program will lost when poweroff
flash	Program will be downloaded into flash, when running, program will be copied to ilm/ram and run in ilm/ram
flashxip	Program will be downloaded into flash and run directly in flash
ddr	Program will be downloaded into ddr and run directly in ddr, program will lost when poweroff
sram	Program will be downloaded into sram and run directly in sram, program will lost when poweroff

Note:

- This variable now target dependent, and its meaning depending on how this variable is implemented in SoC's build.mk
- *GD32VF103 SoC* (page 67) only support **DOWNLOAD=flashxip**
- **flashxip** mode in *Nuclei Eval SoC* (page 65) is very slow due to the CORE frequency is very slow, and flash execution speed is slow
- **ddr** mode is introduced in release `0.2.5` of Nuclei SDK
- macro `DOWNLOAD_MODE` and `DOWNLOAD_MODE_STRING` will be defined in Makefile, eg. when `DOWNLOAD=flash`, macro will be defined as `-DDOWNLOAD_MODE=DOWNLOAD_MODE_FLASH`, and `-DDOWNLOAD_MODE_STRING=\"flash\"`, the `flash` will be in upper case, currently `DOWNLOAD_MODE_STRING` macro is used in `system_<Device>.c` when banner is print.
- This download mode is also used to clarify whether in the link script, your eclic vector table is placed in `.vtable_ilm` or `.vtable` section, eg. for evalsoc, when `DOWNLOAD=flash`, vector table is placed in `.vtable_ilm` section, and an extra macro called `VECTOR_TABLE_REMAPPED` will be passed in Makefile. When `VECTOR_TABLE_REMAPPED` is defined, it means vector table's LMA and VMA are different, it is remapped.
- From release `0.3.2`, this `DOWNLOAD_MODE` should not be used, and macros `DOWNLOAD_MODE_ILM`, `DOWNLOAD_MODE_FLASH`, `DOWNLOAD_MODE_FLASHXIP` and `DOWNLOAD_MODE_DDR` previously defined in `riscv_encoding.h` now are moved to `<Device.h>` such as `evalsoc.h`, and should be deprecated in future.

Now we are directly using `DOWNLOAD_MODE_STRING` to pass the download mode string, no longer need to define it in source code as before.

- From release `0.3.2`, you can define **DOWNLOAD** not just the download mode list above, you can use other download mode names specified by your customized SoC.
- For SRAM download mode, for 200/300, it don't has DDR, so sram is a external ram outside of cpu, for 600/900, it has DDR, so sram is the ddr ram

---

CORE

CORE variable is used to declare the Nuclei processor core of the application.

NOTICE: Nuclei 100 series such as N100 is not supported by normal Nuclei SDK, you need to switch to `develop_n100` branch to try it out.

Currently it has these cores supported as described in table table_dev_buildsystem_6.

When **CORE** is selected, the **ARCH**, **ABI** and **TUNE** (optional) are set, and it might affect the compiler options in combination with *ARCH_EXT* (page 31) depended on the implementation of SoC build.mk.

If you are not sure about which ARCH and extension and cpu feature your Nuclei CPU has, you can run *cpuinfo* (page 92) example to confirm it.

Note:

- `n205/n205e/n305/n307/n307fd` CORE are removed in Nuclei SDK 0.7.0
- `n200e/n202/n202e` CORE are added in Nuclei SDK 0.7.0
- In Nuclei SDK, this **CORE** variable is just a **shorthand** to find a suitable **ARCH**, **ABI** and **TUNE** for target SoC to pass to the compiler as described in above table. So for example, **CORE=n600fd** equals **CORE=u600fd**, **CORE=n900fd** equals **CORE=u900fd**
- Nuclei CPU product name such as N310, NA300, NA900, NI900, N308 is just a name, since the CPU itself is configurable, so the final **ARCH** and **ABI** is different according to your configuration, you should find a proper base **CORE** name according to your CPU RTL configuration, and if you have extra ISA not fit in this **CORE** name, you can pass it via *ARCH_EXT* (page 31), for example, if your CPU product is NA300, and **CPU_ISA** after RTL configuration is `rv32imafd_zca_zcb_zcf_zcmp_zcmt_zba_zbb_zbc_zbs_zfhmin_zicond_xxldspn3x`, then you can set **CORE=n300fd**, **ARCH_EXT** can be set to empty **ARCH_EXT=**, or **ARCH_EXT=_zca_zcb_zcf_zcmp_zcmt_zba_zbb_zbc_zbs_zfhmin_zicond_xxldspn3x**, or shorter **ARCH_EXT=_zca_zcb_zcf_zcmp_zcmt_zicond_xxldsp**, but a invalid **ARCH_EXT** could cause a library not match issue due to toolchain can only distributed with limited multilib which can be checked via `riscv64-unknown-elf-gcc -print-multi-lib`, so please take care.
- For other CPU features such as TEE, ECLIC, TIMER, CACHE, CCM, SMP and etc, you should modify the section **Processor and Core Peripheral Section** in your <`Device.h`>, such as `SoC/evalsoc/Common/Include/evalsoc.h`.

Take `SOC=evalsoc` as example.

- If **CORE=n205 ARCH_EXT=**, then `ARCH=rv32imac, ABI=ilp32 TUNE=nuclei-200-series`. riscv arch related compile and link options will be passed, for this case, it will be `-march=rv32imac -mabi=ilp32 -mtune=nuclei-200-series`.
- If **CORE=n205 ARCH_EXT=_zba_zbb_zbc_zbs**, it will be `-march=rv32imac_zba_zbb_zbc_zbs -mabi=ilp32 -mtune=nuclei-200-series`.

For riscv code model settings, the `RISCV_CMODEL` variable will be set to medlow for RV32 targets, otherwise it will be medany.

The some SoCs, the CORE is fixed, so the ARCH and ABI will be fixed, such as `gd32vf103` SoC, in build system, the CORE is fixed to n205, and ARCH=rv32imac, ABI=ilp32.

ARCH_EXT

ARCH_EXT variable is used to select extra RISC-V arch extensions supported by Nuclei RISC-V Processor, except the `iemafdc`.

Note: Nuclei Toolchain 2023.10[36] now bump gcc version from gcc 10 to gcc 13, which introduced incompatiable `-march` option, so `ARCH_EXT` usage is also incompatiable now.

About the incompatiable march option change, please see https://github.com/riscv-non-isa/riscv-toolchain-conventions/pull/26, which is already present in latest gcc and clang release.

About latest and full version of RISC-V Ratified ISA Spec, please click latest released spec here https://github.com/riscv/riscv-isa-manual/releases/, check the `unpriv-isa-asciidoc.pdf` and `priv-isa-asciidoc.pdf`.

About Nuclei RISC-V toolchain user guide, please check https://doc.nucleisys.com/nuclei_tools/toolchain/index.html

---

When using gcc 13 or clang 17 toolchain in 2023.10 or later toolchain release, you need to use it like this in 0.5.0 sdk release or later version.

Here are several examples when using **ARCH_EXT** for Nuclei RISC-V Processors:

Note: This **ARCH_EXT=** is only used in Nuclei SDK makefile based build system, not used in Nuclei Studio IDE, in Nuclei Studio IDE, you need to set the **Other extensions** in `Nuclei Settings` or `Project Properities -> Settings -> C/C++ Build -> Tool Settings -> Target Processor -> Other Extensions`, eg. If you pass **ARCH_EXT=_zba_zbb_zbc_zbs** using make, then you should set `_zba_zbb_zbc_zbs` in **Other extensions**.

- If you want to use just B 1.0 extension[37], you can pass **ARCH_EXT=_zba_zbb_zbc_zbs**
- If you want to use just Nuclei implemented P 0.5.4 extension[38] and N1/N2/N3 customized extension
  - Xxldsp: means P 0.5.4 + Nuclei default enabled additional 8 expd instructions for both RV32 and RV64, you can pass **ARCH_EXT=_xxldsp**
  - Xxldspn1x: means Xxldsp + Nuclei N1 additional instructions for RV32 only, you can pass **ARCH_EXT=_xxldspn1x**
  - Xxldspn2x: means Xxldspn1x + Nuclei N2 additional instructions for RV32 only, you can pass **ARCH_EXT=_xxldspn2x**
  - Xxldspn3x: means Xxldspn2x + Nuclei N3 additional instructions for RV32 only, you can pass **ARCH_EXT=_xxldspn3x**
- If you want to use K 1.0 extension[39], you can pass **ARCH_EXT=_zk_zks**
- If you want to use V 1.0 extension[40]
  - For rv32 without f/d extension, you can pass **ARCH_EXT=_zve32x**

[36] https://github.com/riscv-mcu/riscv-gnu-toolchain/releases/tag/nuclei-2023.10
[37] https://github.com/riscv/riscv-bitmanip/releases/tag/1.0.0
[38] https://github.com/riscv/riscv-p-spec/blob/33be869910077afd52653031f18a235b1f9d4442/P-ext-proposal.adoc
[39] https://github.com/riscv/riscv-crypto/releases/tag/v1.0.0
[40] https://github.com/riscv/riscv-v-spec/releases/tag/v1.0

  - For rv32 with f/d extension, you can pass **ARCH_EXT=_zve32f**
  - For rv64 without f/d extension, you can pass **ARCH_EXT=_zve64x**
  - For rv64 with f extension, you can pass **ARCH_EXT=_zve64f**
  - For rv64 with fd extension, you can pass **ARCH_EXT=v**
- If you want to use F16(zfh/zvfh) extension, you can follow below steps
  - For case without vector extension, you can add extra `_zfh` to **ARCH_EXT**, eg, **ARCH_EXT=_zfh**
  - For case with vector extension, you can add extra `_zfh_zvfh` to **ARCH_EXT**, eg, **ARCH_EXT=_zfh_zvfh**
  - And the prebuilt NMSIS DSP library also provide F16 support with prebuilt F16 library, you can check library name with `zfh`, such as `NMSIS/Library/DSP/GCC/libnmsis_dsp_rv32imafc_zfh_zvfh_zve32f.a`
  - Spec about zfh extension[41] and zvfh extension[42]
- If you want to use Zc 1.0 extension[43]
  - You can use it together with C extension, which means it should be concat with isa string like `rv32imafd_zca_zcb_zcf_zcmp_zcmt`
  - In Nuclei SDK, the isa string processing is done in build system
  - If you want to use with n300/n900, you can pass **ARCH_EXT=_zca_zcb_zcmp_zcmt**
  - If you want to use with n300f/n900f, you can pass **ARCH_EXT=_zca_zcb_zcf_zcmp_zcmt**
  - If you want to use with n300fd/n900fd, you can pass **ARCH_EXT=_zca_zcb_zcf_zcmp_zcmt**
  - If you want to use with n300fd/n900fd without zcmp/zcmt, you can pass **ARCH_EXT=_zca_zcb_zcf_zcd**
  - If you want to use with extra Nuclei Code Size Reduction extension called Xxlcz, you can add extra `_xxlcz` in **ARCH_EXT**, eg. for n300, you can pass **ARCH_EXT=_zca_zcb_zcmp_zcmt_xxlcz**
- When using customized extensions such as Xxldsp/Xxldspn1x/Xxldspn2x/Xxldspn3x/Xxlcz, the isa string must be placed after all `_z` started isa strings, here is an legal string such as `rv32imafd_zca_zcb_zcf_zcmp_zcmt_zba_zbb_zbc_zbs_zk_zks_xxlcz_xxldspn3x` for rv32 with imafd + Zc + B + K + Xxldspn3x + Xxlcz
- You need to handle this **ARCH_EXT** carefully, expecially using with demo_dsp demo since it will default search library match the whole arch name but you can pass *NMSIS_LIB_ARCH* (page 38) variable in Makefile to choose your desired library arch.
- LLVM Clang in Nuclei RISC-V Toolchain 2023.10 don't support `Xxldsp` and `Xxlcz` extension now, please take care.
- When using llvm clang compiler, the isa string order must be treat carefully, it is not handled very good when searching different multilib.
- You can check prebuilt multilib for gcc and clang using `riscv64-unknown-elf-gcc --print-multi-lib` and `riscv64-unknown-elf-clang --print-multi-lib`

Here below are for using gcc 10 toolchain, you can use it like this below in old nuclei sdk release before 0.5.0.

Currently, valid arch extension combination should match the order of `bpv`.

Here is a list of valid arch extensions:

[41] https://wiki.riscv.org/display/HOME/Recently+Ratified+Extensions
[42] https://github.com/riscv/riscv-v-spec/releases/tag/zvfh
[43] https://github.com/riscv/riscv-code-size-reduction/releases/tag/v1.0.4-3

- **ARCH_EXT=b**: RISC-V bitmanipulation extension.
- **ARCH_EXT=p**: RISC-V packed simd extension.
- **ARCH_EXT=v**: RISC-V vector extension.
- **ARCH_EXT=bp**: RISC-V bitmanipulation and packed simd extension.
- **ARCH_EXT=pv**: RISC-V packed simd and vector extension.
- **ARCH_EXT=bpv**: RISC-V bitmanipulation, packed simd and vector extension.

It is suggested to use this **ARCH_EXT** with other arch options like this, can be found in `SoC/evalsoc/build.mk`:

```
# Set RISCV_ARCH and RISCV_ABI
CORE_UPPER := $(call uc, $(CORE))
CORE_ARCH_ABI := $($(CORE_UPPER)_CORE_ARCH_ABI)
RISCV_ARCH ?= $(word 1, $(CORE_ARCH_ABI))$(ARCH_EXT)
RISCV_ABI ?= $(word 2, $(CORE_ARCH_ABI))
```

CPU_SERIES

Note:

- This variable is used to control different compiler options for different Nuclei CPU series such as 200/300/600/900/1000.
- If you are looking for Nuclei 100 series support, please refer to `develop_n100` branch of Nuclei SDK repository.

This variable will be auto set if your CORE variable match the following rules:

- **200**: CORE start with *20*, the CPU_SERIES will be 200.
- **300**: CORE start with *30*, the CPU_SERIES will be 300.
- **600**: CORE start with *60*, the CPU_SERIES will be 600.
- **900**: CORE start with *90*, the CPU_SERIES will be 900.
- **1000**: CORE start with *100*, the CPU_SERIES will be 1000.
- **0**: CORE start with others, the CPU_SERIES will be 0.

It can also be defined in Makefile itself directly or passed via make command.

It will also define an macro called **CPU_SERIES**, eg. for CPU_SERIES=200, it will define macro CPU_SERIES=200.

This variable is currently used in benchmark cases, and require application Makefile changes.

SEMIHOST

If **SEMIHOST=1**, it means it will enable semihost support using openocd.

From 0.5.0, both newlib and libncrt support semihosting feature, and when using semihost, no need to implement the clib stub functions, which is done by newlib or libncrt semihosting library.

And for qemu 2023.10 verison, you can also use semihosting feature, simple usage is like below for qemu:

```
cd application/baremetal/helloworld
# clean project first
make SOC=evalsoc SEMIHOST=1 clean
make SOC=evalsoc SEMIHOST=1 all
# run on qemu, SEMIHOST=1 is required to pass when run qemu
make SOC=evalsoc SEMIHOST=1 run_qemu
```

When using semihosting feature with openocd, debug message will print via openocd console.

You need to use it like this(assume you are run on evalsoc, CORE=n300):

In terminal 1, open openocd and monitor the output:

```
cd application/baremetal/helloworld
make SOC=evalsoc CORE=n300 run_openocd
# when terminal 2 has download program and start to run, you will be able to see output␣
↪here
```

In terminal 2, gdb connect to the openocd exposed gdb port and load program, and run

```
# in normal shell terminal
cd application/baremetal/helloworld
make SOC=evalsoc CORE=n300 SEMIHOST=1 clean
make SOC=evalsoc CORE=n300 SEMIHOST=1 run_gdb

# now in gdb command terminal, run the following command
monitor reset halt
load
## when run continue, you will be able to see output in previous terminal 1 running␣
↪openocd
continue
```

SIMULATION

If **SIMULATION=1**, it means the program is optimized for hardware simulation environment.

Currently if **SIMULATION=1**, it will pass compile option **-DCFG_SIMULATION**, application can use this **CFG_SIMULATION** to optimize program for hardware simulation environment.

Note:

- Currently the benchmark applications in **application/baremetal/benchmark** used this optimization

---

GDB_PORT

Note:

- This new variable **GDB_PORT** is added in Nuclei SDK since version `0.2.4`

This variable is not used usually, by default the **GDB_PORT** variable is `3333`.

If you want to change a debug gdb port for openocd and gdb when run `run_openocd` and `run_gdb` target, you can pass a new port such as `3344` to this variable.

For example, if you want to debug application using run_openocd and run_gdb and specify a different port other than `3333`.

You can do it like this, take `nuclei_fpga_eval` board for example, such as port `3344`:

- Open openocd server: `make SOC=evalsoc BOARD=nuclei_fpga_eval CORE=n300f GDB_PORT=3344 run_openocd`
- connect gdb with openocd server: `make SOC=evalsoc BOARD=nuclei_fpga_eval CORE=n300f GDB_PORT=3344 run_gdb`

JTAGSN

Note:

- This new variable **JTAGSN** is added in `0.4.0` release

This variable is used specify jtag adapter serial number in openocd configuration, need to be supported in openocd configuration file and makefile, currently **evalsoc** is supported. It is used by openocd `adapter serial`.

Assume you have a jtag adapter, serial number is `FT6S9RD6`, and you want to download program through this jtag to a fpga with ux900 bitstream on it, you can do it like this.

For windows, you need to pass extra `A`, eg. `JTAGSN=FT6S9RD6A`

```
# cd to helloworld
cd application/baremetal/helloworld
# clean program
make SOC=evalsoc CORE=ux900 JTAGSN=FT6S9RD6 clean
# upload program
make SOC=evalsoc CORE=ux900 JTAGSN=FT6S9RD6 upload
```

BANNER

If **BANNER=0**, when program is rebuilt, then the banner message print in console will not be print, banner print is default enabled via `NUCLEI_BANNER=1` in `nuclei_sdk_hal.h`.

when `BANNER=0`, an macro `-DNUCLEI_BANNER=0` will be passed in Makefile.

The banner message looks like this:

```
Nuclei SDK Build Time: Jul 23 2021, 10:22:50
Download Mode: ILM
CPU Frequency 15999959 Hz
```

V

If **V=1**, it will display compiling message in verbose including compiling options.

By default, no compiling options will be displayed in make console message just to print less message and make the console message cleaner. If you want to see what compiling option is used, please pass **V=1** in your make command.

SILENT

If **SILENT=1**, it will not display any compiling messsage.

If you don't want to see any compiling message, you can pass **SILENT=1** in your make command.

3.2.4 Makefile variables used only in Application Makefile

The following variables should be used in application Makefile at your demand, e.g. `application/baremetal/demo_timer/Makefile`.

TARGET

This is a necessary variable which must be defined in application Makefile.

It is used to set the name of the application, it will affect the generated target filenames.

Warning:

- Please don't put any spaces in TARGET variable
- The variable shouldn't contain any space

```
# invalid case 1
TARGET ?= hello world
# invalid case 2
TARGET ?= helloworld # before this # there is a extra space
```

NUCLEI_SDK_ROOT

This is a necessary variable which must be defined in application Makefile.

It is used to set the path of Nuclei SDK Root, usually it should be set as relative path, but you can also set absolute path to point to Nuclei SDK.

RTOS

RTOS variable is used to choose which RTOS will be used in this application.

You can easily find the supported RTOSes in the **<NUCLEI_SDK_ROOT>/OS** directory.

- If **RTOS** is not defined, then baremetal service will be enabled with this application. See examples in `application/baremetal`.
- If **RTOS** is set the the following values, RTOS service will be enabled with this application.
  - `FreeRTOS`: *FreeRTOS* (page 87) service will be enabled, extra macro `RTOS_FREERTOS` will be defined, you can include FreeRTOS header files now, and use FreeRTOS API, for `FreeRTOS` application, you need to have an `FreeRTOSConfig.h` header file prepared in you application. See examples in `application/freertos`.
  - `UCOSII`: *UCOSII* (page 88) service will be enabled, extra macro `RTOS_UCOSII` will be defined, you can include UCOSII header files now, and use UCOSII API, for `UCOSII` application, you need to have `app_cfg.h`, `os_cfg.h` and `app_hooks.c` files prepared in you application. See examples in `application/ucosii`.
  - `RTThread`: *RT-Thread* (page 88) service will be enabled, extra macro `RTOS_RTTHREAD` will be defined, you can include RT-Thread header files now, and use RT-Thread API, for `RTThread` application, you need to have an `rtconfig.h` header file prepared in you application. See examples in `application/rtthread`.
  - `ThreadX`: *ThreadX* (page 89) service will be enabled, extra macro `RTOS_THREADX` will be defined, you can include ThreadX header files now, and use ThreadX API, for `ThreadX` application, you need to have an `tx_user.h` header file prepared in you application. See examples in `application/threadx`.

AUTOVEC

AUTOVEC variable is used to control whether to enable compiler auto vectorization feature.

By default, the compiler auto vectorization feature is controlled by the compiler options it passed.

When **AUTOVEC=0**, it will disable compiler auto vectorization feature as much as possible by passing extra compiler options, otherwise no extra compiler options will be passed.

- **nuclei_gnu**: `-fno-tree-vectorize -fno-tree-loop-vectorize -fno-tree-slp-vectorize`
- **nuclei_llvm/terapines**: `-fno-vectorize -fno-slp-vectorize`

MIDDLEWARE

MIDDLEWARE variable is used to select which middlewares should be used in this application.

You can easily find the available middleware components in the **<NUCLEI_SDK_ROOT>/Components** directory.

- If **MIDDLEWARE** is not defined, not leave empty, no middlware package will be selected.
- If **MIDDLEWARE** is defined with more than 1 string, such as `fatfs tjpgd`, then these two middlewares will be selected.

Currently we provide the following middlewares:

- **profiling**: This middleware is not expected to use in Makefile based build system, you need to use it in Nuclei Studio, it is used to provide code coverage via gcov and profiling via gprof, for details, please refer to the `README.md` in this folder.

NMSIS_LIB

NMSIS_LIB variable is used to select which NMSIS libraries should be used in this application.

Currently you can select the following libraries:

- **nmsis_dsp**: NMSIS DSP prebuilt library, see https://doc.nucleisys.com/nmsis/dsp/index.html.
- **nmsis_nn**: NMSIS NN prebuilt library, see https://doc.nucleisys.com/nmsis/nn/index.html.

NMSIS DSP and NN library source code can be found in https://github.com/Nuclei-Software/NMSIS.

You can select more than libraries of NMSIS. For example, if you want to use NMSIS NN library, NMSIS DSP library is also required. so you need to set **NMSIS_LIB** like this `NMSIS_LIB := nmsis_nn nmsis_dsp`

NMSIS_LIB_ARCH

This can be used to fix issue of prebuilt library for selected arch is not found during linking.

This variable is used to select real nmsis dsp/nn library arch used, if not set, it will use **RISCV_ARCH** passed.

The **NMSIS_LIB_ARCH** need to match the prebuilt libraries located in **NMSIS/Library/DSP/GCC** and **NMSIS/Library/NN/GCC**, eg. `NMSIS_LIB_ARCH := rv32imafc_zfh_zvfh_zve32f_zba_zbb_zbc_zbs_xxldspn1x` will select `libnmsis_dsp_rv32imafc_zfh_zvfh_zve32f_zba_zba.a` if `NMSIS_LIB := nmsis_dsp`

This is useful when you want to specify a different arch for library.

eg. When your cpu arch is `rv32imafdc_zba_zbb_zbc_zbs_zk_zks_xxldspn3x`, and you want to use `rv32imafdc_zba_zbb_zbc_zbs_xxldspn1x`, then you can set **NMSIS_LIB_ARCH=rv32imafdc_zba_zbb_zbc_zbs_xxldspn1x** in Makefile, otherwise it will use the real cpu arch passed by **CORE** and **ARCH_EXT** or directly via **RISCV_ARCH**.

STDCLIB

STDCLIB variable is used to select which standard c runtime library will be used. If not defined, the default value will be `newlib_nano`.

In Nuclei GNU Toolchain, we destributed newlib/newlib-nano/Nuclei c runtime library, so user can select different c runtime library according to their requirement.

Newlib is a simple ANSI C library, math library, available for both RV32 and RV64.

Nuclei C runtime library is a highly optimized c library designed for deeply embedded user cases, can provided smaller code size and highly optimized floating point support compared to Newlib.

From 0.5.0 release, to support both gcc and clang compiler, we decided not to use `--specs=` option to select system library, instead of that, we start to use `--nodefaultlibs` options, and link the required system libraries by the `STDCLIB` variable choice, so need to link desired libraries such as:

- `-lgcc`: a standard library (linked by default, excluded by -nodefaultlibs) that provides internal subroutines to overcome shortcomings of particular machines, see https://gcc.gnu.org/onlinedocs/gccint/Libgcc.html.
- `-lgcov`: a library used to test coverage program, known as `gcov/gprof`, see https://gcc.gnu.org/onlinedocs/gcc/Gcov.html
- `-lc/-lc_nano`: newlib c library or newlib nano c library, see https://sourceware.org/newlib/docs.html
- `-lm`: newlib math library, see https://sourceware.org/newlib/libm.html
- `-lstdc++`: gnu standard c++ library, see https://gcc.gnu.org/onlinedocs/libstdc++
- `-lsemihost`: riscv semihosting library which implement a set of standard I/O and file I/O operations, see https://github.com/riscv-mcu/riscv-newlib/tree/nuclei/newlib-4.3.0/libgloss/riscv
- `-lnosys`: a set of stub functions which implement a set of standard I/O operations but does nothing, and when link with it, it will throw link warning, see https://github.com/riscv-mcu/riscv-newlib/blob/nuclei/newlib-4.3.0/libgloss/libnosys
- `-lncrt_pico/-lncrt_nano/-lncrt_small/-lncrt_balanced/-lncrt_fast`: Nuclei libncrt library, it provides pico/nano/small/balanced/fast variant to provide standard c library, math library, and libgcc library features, and need to use together with `-lheapops_minimal/-lheapops_basic/-lheapops_realtime` heap operation API, and `-lfileops_uart/-lfileops_semi/-lfileops_rtt` file io operation API, when using this libncrt library, please don't link `-lgcc -lc_nano/-lc -lm -lsemihost -lnosys`, and it also can't link with `-lstdc++`
- Upgrading libncrt from Nuclei GNU Toolchain 2022.12 to Nuclei Toolchain 2023.10, please change it like this, take **libncrt_small** as example:
  - **asm/c/c++ options**: `--specs=libncrt_small.specs` -> `--specs=libncrt_small.specs` works for gcc, or `-isystem=/include/libncrt` works for both gcc and clang
  - **ld options**: `--specs=libncrt_small.specs` -> `--specs=libncrt_small.specs -lheapops_basic -lfileops_uart` works for gcc, `-nodefaultlibs -lncrt_small -lheapops_basic -lfileops_uart` works for both gcc and clang
  - We recommend you to use later version works for both gcc and clang, `-nodefaultlibs` is used to exclude startup crt, libgcc and c library in default gcc or clang, use the version specified by us to use libncrt.

Table 7: Available STDCLIB choices

STDCLIB	Description
newlib_full	Normal version of newlib, optimized for speed at cost of size. It provided full feature of newlib, with file io supported.
newlib_fast	Newlib nano version, with printf float and scanf float support.
newlib_small	Newlib nano version, with printf float support.
newlib_nano	Newlib nano version, without printf/scanf float support.
libncrt_fast	Nuclei C runtime library optimized for speed, full feature
libncrt_balanced	Nuclei C runtime library balanced at speed and code size, full feature
libncrt_small	Nuclei C runtime library optimized for code size, full feature
libncrt_nano	Nuclei C runtime library optimized for code size, without float/double support
libncrt_pico	Nuclei C runtime library optimized for code size, without long/long long/float/double support
nostd	no std c library will be used, and don't search the standard system directories for header files
nospec	no std c library will be used, not pass any –specs options

Note:

- For clang based compiler, if `-u _print_float` is not passed in linker options, it may fail during link process, so here we pass `-u _print_float` for newlib_nano, then it means for nuclei_llvm and terapines toolchain, `STDCLIB=newlib_nano` equals to `STDCLIB=newlib_small`
- Nuclei libncrt library couldn't be used with terapines toolchain, so you can't use any libncrt library when you are using terapines toolchain.
- About Newlib and Newlib nano difference, please check https://github.com/riscv-collab/riscv-newlib/blob/riscv-newlib-3.2.0/newlib/README
- About Nuclei C runtime library, it provided basic libgcc, c library and math library feature, but it didn't provided all the features that newlib can do, it is highly optimized for deeply embedded scenery, user no need to link with `-lm` when using libncrt library when math library is needed.
- Nuclei C runtime library is only available in Nuclei GNU Toolchain released after Nov 2021, about how to use this library, please follow doc located in `gcc\share\pdf`, changes need to be done in startup code, linker script, stub code, and compiler options, you can check commit history of nuclei sdk for support of libncrt.
- Nuclei C runtime library(libncrt) only support RV32 CPU target, so you cannot use it with RV64 CPU.
- Since there are different c runtime library can be chosen now, so developer need to provide different stub functions for different library, please check `SoC/evalsoc/Common/Source/Stubs/` and `SoC/evalsoc/build.mk` for example.

NCRTHEAP

Note:

- This variable is added in 0.5.0 release to support libncrt v3.0.0.

This variable is only valid when using libncrt c library >= v3.0.0, and you can choose different heapops when using libncrt c library to do heap related operations such as malloc or free.

- **basic**: default, this is previous release of libncrt c library used one. A low-overhead best-fit heap where allocation and deallocation have very little internal fragmentation
- **realtime**: A real-time heap where allocation and deallocation have O(1) performance
- **minimal**: An allocate-only heap where deallocation and reallocation are not implemented

For previous libncrt library, this heapops is default binded with libncrt library, so you can't choose different heap type, but now you can choose according to your requirements.

NCRTIO

Note:

- This variable is added in 0.5.0 release to support libncrt v3.0.0.

This variable is only valid when using libncrt c library >= v3.0.0, and you can choose different fileops when using libncrt c library to do basic input/output operations.

- **uart**: default, lower level input/output via uart, developer need to implement metal_tty_putc/getc
- **semi**: input/output via semihosting, if you pass **SEMIHOST=1** in make, it will default choose this one when using libncrt library.
- **rtt**: input/output via jlink rtt, require to use JLink tool.

SMP

SMP variable is used to control smp cpu core count, valid number must > 1.

When **SMP** variable is defined, extra gcc options for ld is passed `-Wl,--defsym=__SMP_CPU_CNT=$(SMP)`, and extra c macro `-DSMP_CPU_CNT=$(SMP)` is defined this is passed in each SoC's build.mk, such as `SoC/evalsoc/build.mk`.

When SMP variable is defined, extra openocd command `set SMP $(SMP)` will also be passed when run openocd upload or create a openocd server.

For SMP application, please check `application/baremetal/smphello`, if you want to implement a smp application, you need to reimplement `smp_main`, which all harts will run to this function instead of `main`, if you don't implement it, a weak `smp_main` in `startup_<Device>.S` will be used, and only boot hartid specified by **BOOT_HARTID** will enter to main, other harts will do wfi.

BOOT_HARTID

Note:

- This new variable **BOOT_HARTID** is added in `0.4.0` release

This variable is used to control the boot hartid in a multiple core system. If **SMP** variable is specified, it means this application is expected to be a smp application, otherwise it means this application is expected to be a amp application.

For amp application, only the boot hart specified by **BOOT_HARTID** will run, other harts will directly do wfi when startup, but for smp application, other hartid will do normal boot code instead of code/data/bss init, and do sync harts to make sure all harts boots.

For both amp and smp application, the program should execute on a share memory which all harts can access, not hart private memory such as ilm/dlm.

Currently **SMP** and **BOOT_HARTID** support all require SOC support code to implement it, currently evalsoc support it, currently qemu simulation didn't work for SMP/AMP use case.

Here is some basic usage for SMP and BOOT_HARTID on UX900 x4, run on external ddr.

```
# cd to helloworld
cd <Nuclei SDK>/application/baremetal/helloworld
# clean program
make SOC=evalsoc CORE=ux900 clean
# AMP: choose hart 1 as boot hartid, other harts spin
make SOC=evalsoc CORE=ux900 BOOT_HARTID=1 DOWNLOAD=ddr clean upload
cd <Nuclei SDK>/application/baremetal/smphello
# SMP: choose hart 2 as boot hartid
make SOC=evalsoc CORE=ux900 BOOT_HARTID=2 SMP=4 DOWNLOAD=ddr clean upload
```

HARTID_OFS

Note:

- This new variable is added in `0.5.0` release

This variable is used to set hartid offset relative to real hart index in a complex AMP SoC system.

eg.

In a SoC system, it has 2 CPU, CPU 0 has 2 smp core, CPU 1 has 1 core, and CPU 0 hartid is 0, 1, and CPU 1 hartid is 2, so for CPU 0, HARTID_OFS is 0, for CPU 1, HARTID_OFS is 2.

STACKSZ

STACKSZ variable is used to control the per core stack size reserved in linker script, this need to cooperate with link script file and linker options.

In link script file, `__STACK_SIZE` symbol need to use `PROVIDE` feature of ld to define a weak version, such as `PROVIDE(__STACK_SIZE = 2K);`, and gcc will pass ld options `-Wl,--defsym=__STACK_SIZE=$(STACKSZ)` to overwrite the default value if **STACKSZ** is defined.

STACKSZ variable must be a valid value accepted by ld, such as 0x2000, 2K, 4K, 8192.

For SMP version, stack size space need to reserve **STACKSZ** x SMP Core Count size.

You can refer to `SoC/evalsoc/Board/nuclei_fpga_eval/Source/GCC/gcc_evalsoc_ilm.ld` for smp version.

HEAPSZ

HEAPSZ variable is used to control the heap size reserved in linker script, this need to cooperate with link script file and linker options.

In link script file, `__HEAP_SIZE` symbol need to use `PROVIDE` feature of ld to define a weak version, such as `PROVIDE(__HEAP_SIZE = 2K);`, and gcc will pass ld options `-Wl,--defsym=__HEAP_SIZE=$(HEAPSZ)` to overwrite the default value if **HEAPSZ** is defined.

HEAPSZ variable must be a valid value accepted by ld, such as 0x2000, 2K, 4K, 8192.

RISCV_ARCH

RISCV_ARCH variable is used to control compiler option `-mcmodel=$(RISCV_ARCH)`.

It might override RISCV_ARCH defined in SoC build.mk, according to your build.mk implementation.

RISCV_ARCH might directly affect the gcc compiler option depended on the implementation of SoC build.mk.

Take `SOC=evalsoc` for example.

- **CORE=n300 RISCV_ARCH=rv32imafdc_zk_zks RISCV_ABI=ilp32d ARCH_EXT=_zba_zbb_zbc_zbs**, then final compiler options will be `-march=rv32imafdc_zk_zks -mabi=ilp32d -mtune=nuclei-300-series`. The **ARCH_EXT** is ignored.

RISCV_ABI

RISCV_ABI variable is used to control compiler option `-mcmodel=$(RISCV_ABI)`.

It might override RISCV_ABI defined in SoC build.mk, according to your build.mk implementation.

RISCV_CMODEL

RISCV_CMODEL is used to control compiler option `-mcmodel=$(RISCV_CMODEL)`.

For RV32, default value is `medlow`, otherwise `medany` for RV64.

You can set `RISCV_CMODEL` to override predefined value.

RISCV_TUNE

RISCV_TUNE is used to control compiler option `-mtune=$(RISCV_TUNE)`.

It is defined in SoC build.mk, you can override it if your implementation allow it.

APP_COMMON_FLAGS

Note:

- Added in 0.4.0 release.

This variable is used to define app common compiler flags to all c/asm/cpp compiler. You can pass it via make command to define extra flags to compile application.

APP_ASMFLAGS

This variable is similiar to **APP_COMMON_FLAGS** but used to pass extra app asm flags.

APP_CFLAGS

This variable is similiar to **APP_COMMON_FLAGS** but used to pass extra app c flags.

APP_CXXFLAGS

This variable is similiar to **APP_COMMON_FLAGS** but used to pass extra app cxx flags.

APP_LDFLAGS

This variable is similiar to **APP_COMMON_FLAGS** but used to pass extra app linker flags.

NOGC

NOGC variable is used to control whether to enable gc sections to reduce program code size or not, by default GC is enabled to reduce code size.

When GC is enabled, these options will be added:

- Adding to compiler options: `-ffunction-sections -fdata-sections`
- Adding to linker options: `-Wl,--gc-sections -Wl,--check-sections`

If you want to enable this GC feature, you can set **NOGC=0** (default), GC feature will remove sections for you, but sometimes it might remove sections that are useful, e.g. For Nuclei SDK test cases, we use ctest framework, and we need to set **NOGC=1** to disable GC feature.

When `NOGC=0``(default), extra compile options ``-ffunction-sections -fdata-sections`, and extra link options `-Wl,--gc-sections -Wl,--check-sections` will be passed.

RTTHREAD_MSH

RTTHREAD_MSH variable is valid only when **RTOS** is set to **RTThread**.

When **RTTHREAD_MSH** is set to **1**:

- The RTThread MSH component source code will be included
- The MSH thread will be enabled in the background
- Currently the msh getchar implementation is using a weak function implemented in `rt_hw_console_getchar` in `OS/RTTThread/libcpu/risc-v/nuclei/cpuport.c`

3.2.5 Build Related Makefile variables used only in Application Makefile

If you want to specify additional compiler flags, please follow this guidance to modify your application Makefile.

Nuclei SDK build system defined the following variables to control the build options or flags.

INCDIRS

This **INCDIRS** is used to pass C/CPP/ASM include directories.

e.g. To include current directory `.` and `inc` for C/CPP/ASM

```
INCDIRS = . inc
```

C_INCDIRS

This **C_INCDIRS** is used to pass C only include directories.

e.g. To include current directory `.` and `cinc` for C only

```
C_INCDIRS = . cinc
```

CXX_INCDIRS

This **CXX_INCDIRS** is used to pass CPP only include directories.

e.g. To include current directory `.` and `cppinc` for CPP only

```
CXX_INCDIRS = . cppinc
```

ASM_INCDIRS

This **ASM_INCDIRS** is used to pass ASM only include directories.

e.g. To include current directory `.` and `asminc` for ASM only

```
ASM_INCDIRS = . asminc
```

SRCDIRS

This **SRCDIRS** is used to set the source directories used to search the C/CPP/ASM source code files, it will not do recursively.

e.g. To search C/CPP/ASM source files in directory `.` and `src`

```
SRCDIRS = . src
```

C_SRCDIRS

This **C_SRCDIRS** is used to set the source directories used to search the C only source code files(`*.c, *.C`), it will not do recursively.

e.g. To search C only source files in directory `.` and `csrc`

```
C_SRCDIRS = . csrc
```

CXX_SRCDIRS

This **CXX_SRCDIRS** is used to set the source directories used to search the CPP only source code files(`*.cpp, *.CPP`), it will not do recursively.

e.g. To search CPP only source files in directory `.` and `cppsrc`

```
CXX_SRCDIRS = . cppsrc
```

ASM_SRCDIRS

This **ASM_SRCDIRS** is used to set the source directories used to search the ASM only source code files(`*.s, *.S`), it will not do recursively.

e.g. To search ASM only source files in directory `.` and `asmsrc`

```
ASM_SRCDIRS = . asmsrc
```

C_SRCS

If you just want to include a few of C source files in directories, you can use this **C_SRCS** variable, makefile wildcard pattern supported.

e.g. To include `main.c` and `src/hello.c`

```
C_SRCS = main.c src/hello.c
```

CXX_SRCS

If you just want to include a few of CPP source files in directories, you can use this **CXX_SRCS** variable, makefile wildcard pattern supported.

e.g. To include `main.cpp` and `src/hello.cpp`

```
CXX_SRCS = main.cpp src/hello.cpp
```

ASM_SRCS

If you just want to include a few of ASM source files in directories, you can use this **ASM_SRCS** variable, makefile wildcard pattern supported.

e.g. To include `asm.s` and `src/test.s`

```
ASM_SRCS = asm.s src/test.s
```

EXCLUDE_SRCS

If you just want to exclude a few of c/asm/cpp source files in directories, you can use this **EXCLUDE_SRCS** variable, makefile wildcard pattern supported.

e.g. To exclude `test2.c` and `src/test3.c`

```
EXCLUDE_SRCS = test2.c src/test3.c
```

COMMON_FLAGS

This **COMMON_FLAGS** variable is used to define common compiler flags to all c/asm/cpp compiler.

For example, you can add a newline `COMMON_FLAGS += -O3 -funroll-loops -fpeel-loops` in your application Makefile and these options will be passed to C/ASM/CPP compiler.

This variable should be defined in Makefile.

CFLAGS

Different to **COMMON_FLAGS**, this **CFLAGS** variable is used to define common compiler flags to C compiler only.

For example, you can add a newline `CFLAGS += -O3 -funroll-loops -fpeel-loops` in your application Makefile and these options will be passed to C compiler.

CXXFLAGS

Different to **COMMON_FLAGS**, this **CXXFLAGS** variable is used to define common compiler flags to cpp compiler only.

For example, you can add a newline `CXXFLAGS += -O3 -funroll-loops -fpeel-loops` in your application Makefile and these options will be passed to cpp compiler.

ASMFLAGS

Different to **COMMON_FLAGS**, this **ASMFLAGS** variable is used to define common compiler flags to asm compiler only.

For example, you can add a newline `ASMFLAGS += -O3 -funroll-loops -fpeel-loops` in your application Makefile and these options will be passed to asm compiler.

LDFLAGS

This **LDFLAGS** is used to pass extra linker flags, for example, if you want to use standard system libraries when linking, you can add a newline `LDFLAGS += -nodefaultlibs` in you application Makefile.

If you want to link with other libraries, please use `LDLIBS` and `LIBDIRS` variables.

LDLIBS

This **LDLIBS** variable is library flags or names given to compilers when they are supposed to invoke the linker.

Non-library linker flags, such as -L, should go in the **LIBDIRS** or **LDFLAGS** variable.

LIBDIRS

This **LIBDIRS** variable is used to store the library directories, which could be used together with **LDLIBS**.

For example, if you have a library located in **$(NUCLEI_SDK_ROOT)/Library/DSP/libnmsis_dsp_rv32imac.a**, and you want to link it, then you can define these lines:

```
LDLIBS = -lnmsis_dsp_rv32imac
LIBDIRS = $(NUCLEI_SDK_ROOT)/Library/DSP
```

LINKER_SCRIPT

This **LINKER_SCRIPT** variable could be used to set the link script of the application.

By default, there is no need to set this variable, since the build system will define a default linker script for application according to the build configuration. If you want to define your own linker script, you can set this variable.

For example, `LINKER_SCRIPT := gcc.ld`.

3.3 Application Development

3.3.1 Overview

Here will describe how to develop an Nuclei SDK application.

To develop a Nuclei SDK application from scratch, you can do the following steps:

1. Create a directory to place your application code.
2. Create **Makefile** in the new created directory, the minimal **Makefile** should look like this

```
TARGET = your_target_name

NUCLEI_SDK_ROOT = path/to/your_nuclei_sdk_root

SRCDIRS = .

INCDIRS = .

include $(NUCLEI_SDK_ROOT)/Build/Makefile.base
```

3. Copy or create your application code in new created directory.

Note:

- If you just want to SoC related resource, you can include header file `nuclei_sdk_soc.h` in your application code.

- If you just want to SoC and Board related resource, you can include header file `nuclei_sdk_hal.h` in your application code.
- For simplity, we recomment you to use `nuclei_sdk_hal.h` header file

4. Follow *Build System based on Makefile* (page 19) to change your application Makefile.

3.3.2 Add Extra Source Code

If you want to add extra source code, you can use these makefile variables:

To add all the source code in directories, recursive search is not supported.

- *SRCDIRS* (page 46): Add C/CPP/ASM source code located in the directories defined by this variable.
- *C_SRCDIRS* (page 46): Add C only source code located in the directories defined by this variable.
- *CXX_SRCDIRS* (page 47): Add CPP only source code located in the directories defined by this variable.
- *ASM_SRCDIRS* (page 47): Add ASM only source code located in the directories defined by this variable.

To add only selected c/cxx/asm source files

- *C_SRCS* (page 47): Add C only source code files defined by this variable.
- *CXX_SRCS* (page 47): Add CPP only source code files defined by this variable.
- *ASM_SRCS* (page 47): Add ASM only source code files defined by this variable.

To exclude some source files

- *EXCLUDE_SRCS* (page 48): Exclude source files defined by this variable.

3.3.3 Add Extra Include Directory

If you want to add extra include directories, you can use these makefile variables:

- *INCDIRS* (page 46): Include the directories defined by this variable for C/ASM/CPP code during compiling.
- *C_INCDIRS* (page 46): Include the directories defined by this variable for C only code during compiling.
- *CXX_INCDIRS* (page 46): Include the directories defined by this variable for CPP only code during compiling.
- *ASM_INCDIRS* (page 46): Include the directories defined by this variable for ASM only code during compiling.

3.3.4 Add Extra Build Options

If you want to add extra build options, you can use these makefile variables:

- *COMMON_FLAGS* (page 48): This will add compiling flags for C/CPP/ASM source code.
- *CFLAGS* (page 48): This will add compiling flags for C source code.
- *CXXFLAGS* (page 48): This will add compiling flags for CPP source code.
- *ASMFLAGS* (page 48): This will add compiling flags for ASM source code.
- *LDFLAGS* (page 48): This will add linker flags when linking.
- *LDLIBS* (page 49): This will add extra libraries need to be linked.
- *LIBDIRS* (page 49): This will add extra library directories to be searched by linker.

3.3.5 Optimize For Code Size

If you want to optimize your application for code size, you set `COMMON_FLAGS` in your application Makefile like this:

```
COMMON_FLAGS := -Os
```

If you want to optimize code size even more, you use this link time optimization(LTO) as below:

```
COMMON_FLAGS := -Os -flto
```

see *demo_eclic* (page 94) for example usage of optimize for code size.

For more details about gcc optimization, please refer to Options That Control Optimization in GCC[44].

3.3.6 Change Link Script

If you want to change the default link script defined by your make configuration(SOC, BOARD, DOWNLOAD). You can use *LINKER_SCRIPT* (page 49) variable to set your linker script.

The default linker script used for different boards can be found in *Board* (page 71).

3.3.7 Set Default Make Options

Set Default Global Make Options For Nuclei SDK

If you want to change the global Make options for the Nuclei SDK, you can add the *Makefile.global* (page 24).

Set Local Make Options For Your Application

If you want to change the application level Make options, you can add the *Makefile.local* (page 25).

3.4 Build Nuclei SDK Documentation

In Nuclei SDK, we use Sphinx and restructured text as documentation tool.

Here we only provide steps to build sphinx documentation in Linux environment.

3.4.1 Install Tools

To build this the documentation, you need to have these tools installed.

- Python3
- Python Pip tool

Then you can use the pip tool to install extra python packages required to build the documentation.

```
pip install -r doc/requirements.txt
```

[44] https://gcc.gnu.org/onlinedocs/gcc-9.2.0/gcc/Optimize-Options.html#Optimize-Options

3.4.2 Build The Documentation

Then you can build the documentation using the following command:

```
# cd to document folder
cd doc
# Build Sphinx documentation
make html
```

The documentation will be generated in *doc/build/html* folder.

You can open the *doc/build/html/index.html* in your browser to view the details.

CHAPTER

FOUR

CONTRIBUTING

Contributing to Nuclei SDK project is always welcome.

You can always do a lot of things to help Nuclei SDK project improve and grow stronger.

- *Port your Nuclei SoC into Nuclei SDK* (page 53)
- *Submit your issue* (page 57)
- *Submit your pull request* (page 57)

4.1 Port your Nuclei SoC into Nuclei SDK

Note: If you just want to do quick porting based on evalsoc implementation of Nuclei SDK to get quick ramp up, you can refer to Quick Porting to you SoC based on Evalsoc in Nuclei SDK[45]

If you want to port you Nuclei Processor Core based Board to Nuclei SDK, you need to follow these steps:

And the best example is our evalsoc support, although it may contains many unused features you may not want to use, but it is our evaluation SoC and will supply to provide best support for Nuclei RISC-V CPU features, if you are already using it, please keep in update of the evalsoc support changes in each release, you can track it by diff each release changes, and please also remember Nuclei SDK release may also bump with NMSIS release.

Assume your SoC name is `ncstar`, based on Nuclei core *n300f* (page 30), and **RISCV_ARCH** is `rv32imafc`, **RISCV_ABI** is `ilp32f`, and you made a new board called `ncstar_eval`, and this SoC only support **flashxip** *download* (page 29) mode.

Make sure the SoC name and Board name used in this Nuclei SDK is all in lowercase.

1. Create a folder named `ncstar` under **SoC** directory.
   - Create folder named `Board` and `Common` under `ncstar`
   - Create directory structure under `ncstar/Common` like below:

```
<ncstar/Common>
├── Include
│   ├── peripheral_or_device_headers.h
│   ├── ......
│   ├── ncstar.h
│   ├── nuclei_sdk_soc.h
│   └── system_ncstar.h
```

(continues on next page)

[45] https://doc.nucleisys.com/nuclei_studio_supply/28-quick_porting_from_evalsoc_to_customsoc_based_on_Nuclei_SDK/

(continued from previous page)

```
└── Source
    ├── Drivers
    │   ├── peripheral_or_device_sources.c
    │   └── ......
    ├── GCC
    │   ├── intexc_ncstar.S
    │   └── startup_ncstar.S
    ├── Stubs
    │   ├── newlib
    │   └── libncrt
    ├── ncstar_soc.c
    └── system_ncstar.c
```

Note:

- The directory structure is based on the NMSIS device template, please refer to https://doc.nucleisys.com/nmsis/core/core_templates.html
- The folder names must be exactly the same as the directory structure showed
- **peripheral_or_device_sources.c** means the SoC peripheral driver source code files, such as uart, gpio, i2c, spi driver sources, usually get from the SoC firmware library, it should be placed in **Drivers** folder.
- **peripheral_or_device_headers.h** means the SoC peripheral driver header files, such as uart, gpio, i2c, spi driver headers, usually get from the SoC firmware library, it should be placed in **Include** folder.
- The **Stubs** folder contains the stub code files for newlib c library and nuclei c runtime library porting code, take `SoC/evalsoc/Common/Stubs` as reference.
- The **GCC** folder contains *startup* and *exeception/interrupt* assemble code, if your board share the same linker script files, you can also put link script files here, the linker script files name rules can refer to previously supported *evalsoc* SoC.
- If you want to do IAR compiler support, you also need to implement IAR related stuff, which is located in folder called IAR.
- The **nuclei_sdk_soc.h** file is very important, it is a Nuclei SoC Header file used by common application which can run accoss different SoC, it should include the SoC device header file `ncstar.h`

- Create directory structure under `ncstar/Board` like below:

```
<ncstar/Board>
└── ncstar_eval
    ├── Include
    │   ├── ncstar_eval.h
    │   └── nuclei_sdk_hal.h
    ├── openocd_ncstar.cfg
    └── Source
        ├── GCC
        │   └── gcc_ncstar_flashxip.ld
        └── ncstar_eval.c
```

Note:

  - The **ncstar_eval** is the board folder name, if you have a new board, you can create a new folder in the same level
  - **Include** folder contains the board related header files
  - **Source** folder contains the board related source files
  - **GCC** folder is optional, if your linker script for the board is different to the SoC, you need to put your linker script here
  - **openocd_ncstar.cfg** file is the board related openocd debug configuration file
  - **ncstar_eval.h** file contains board related definition or APIs and also include the **SoC** header file, you can refer to previously supported board such as `nuclei_fpga_eval`
  - **nuclei_sdk_hal.h** is very important, it includes the **ncstar_eval.h** header file. This file is used in application as entry header file to access board and SoC resources.

2. Create Makefile related to `ncstar` in *Nuclei SDK build system* (page 19)
   - Create **SoC/ncstar/build.mk**, the file content should be like this:

```
##### Put your SoC build configurations below #####

BOARD ?= ncstar_eval

# override DOWNLOAD and CORE variable for NCSTAR SoC
# even though it was set with a command argument
override CORE := n300f
override DOWNLOAD := flashxip

NUCLEI_SDK_SOC_BOARD := $(NUCLEI_SDK_SOC)/Board/$(BOARD)
NUCLEI_SDK_SOC_COMMON := $(NUCLEI_SDK_SOC)/Common

#no ilm on NCSTAR SoC
LINKER_SCRIPT ?= $(NUCLEI_SDK_SOC_BOARD)/Source/GCC/gcc_ncstar_flashxip.ld
OPENOCD_CFG ?= $(NUCLEI_SDK_SOC_BOARD)/openocd_ncstar.cfg

RISCV_ARCH ?= rv32imafc
RISCV_ABI ?= ilp32f

##### Put your Source code Management configurations below #####

INCDIRS += $(NUCLEI_SDK_SOC_COMMON)/Include

C_SRCDIRS += $(NUCLEI_SDK_SOC_COMMON)/Source \
             $(NUCLEI_SDK_SOC_COMMON)/Source/Drivers

ifneq ($(findstring libncrt,$(STDCLIB)),)
C_SRCDIRS += $(NUCLEI_SDK_SOC_COMMON)/Source/Stubs/libncrt
else ifneq ($(findstring newlib,$(STDCLIB)),)
C_SRCDIRS += $(NUCLEI_SDK_SOC_COMMON)/Source/Stubs/newlib
else
# no stubs will be used
endif
```

(continues on next page)

(continued from previous page)

```
ASM_SRCS += $(NUCLEI_SDK_SOC_COMMON)/Source/GCC/startup_ncstar.S \
            $(NUCLEI_SDK_SOC_COMMON)/Source/GCC/intexc_ncstar.S

# Add extra board related source files and header files
VALID_NUCLEI_SDK_SOC_BOARD := $(wildcard $(NUCLEI_SDK_SOC_BOARD))
ifneq ($(VALID_NUCLEI_SDK_SOC_BOARD),)
INCDIRS += $(VALID_NUCLEI_SDK_SOC_BOARD)/Include
C_SRCDIRS += $(VALID_NUCLEI_SDK_SOC_BOARD)/Source
endif
```

- If you need to place vector table in flash device, and copy it to ilm when startup, such as using `DOWNLOAD=flash` mode, then you need to define extra `VECTOR_TABLE_REMAPPED` macro in this `build.mk`, just take `SoC/evalsoc/build.mk` as reference.

```
## omit some code above
# Add extra cflags for SoC related
ifeq ($(DOWNLOAD), flash)
COMMON_FLAGS += -DVECTOR_TABLE_REMAPPED
endif
## omit some code below
RISCV_ARCH ?= rv32imafc
```

3. If you have setup the source code and build system correctly, then you can test your SoC using the common applications, e.g.

```
# Test helloworld application for ncstar_eval board
## cd to helloworld application directory
cd application/baremetal/helloworld
## clean and build helloworld application for ncstar_eval board
make SOC=ncstar BOARD=ncstar_eval clean all
## connect your board to PC and install jtag driver, open UART terminal
## set baudrate to 115200bps and then upload the built application
## to the ncstar_eval board using openocd, and you can check the
## run messsage in UART terminal
make SOC=ncstar BOARD=ncstar_eval upload
```

Note:

- You can always refer to previously supported SoCs for reference, such as the `evalsoc` and `gd32vf103` SoC, we suggest you follow the `evalsoc` implementation, since it is well maintained to support latest nuclei riscv cpu feature.
- The `evalsoc` SoC is a FPGA based evaluation platform, it have `ilm` and `dlm`, so it support many *download modes* (page 29)
- The `gd32vf103` SoC is a real silicon chip, it only have RAM and onchip flash, it only support FlashXIP mode.
- The **nuclei_sdk_soc.h** must be created in SoC include directory, it must include the device header file <device>.h and SoC firmware library header files.
- The **nuclei_sdk_hal.h** must be created in Board include directory, it must include **nuclei_sdk_soc.h** and board related header files.

4.2 Submit your issue

If you find any issue related to Nuclei SDK project, you can open an issue in https://github.com/Nuclei-Software/nuclei-sdk/issues

4.3 Submit your pull request

If you want to contribute your code to Nuclei SDK project, you can open an pull request in https://github.com/Nuclei-Software/nuclei-sdk/pulls

Regarding to code style, please refer to *Code Style* (page 19).

4.4 Git commit guide

If you want to contribute your code, make sure you follow the guidance of git commit, see here https://chris.beams.io/posts/git-commit/ for details

- Use the present tense ("Add feature" not "Added feature")
- Use the imperative mood ("Move cursor to…" not "Moves cursor to…")
- Limit the first line to 80 characters or less
- Refer github issues and pull requests liberally using #
- Write the commit message with an category name and colon:
  - soc: changes related to soc
  - board: changes related to board support packages
  - nmsis: changes related to NMSIS
  - build: changes releated to build system
  - library: changes related to libraries
  - rtos: changes related to rtoses
  - test: changes related to test cases
  - doc: changes related to documentation
  - ci: changes related to ci environment
  - application: changes related to applications
  - misc: changes not categorized
  - env: changes related to environment

CHAPTER

FIVE

DESIGN AND ARCHITECTURE

5.1 Overview

Nuclei SDK is developed based on **NMSIS**, all the SoCs supported in it are following the NMSIS-Core Device Templates Guidance[46].

So this Nuclei SDK can be treated as a software guide for how to use NMSIS.

The build system we use in Nuclei SDK is `Makefile`, it support both Windows and Linux, and when we develop Nuclei SDK build system, we keep it simple, so it make developer can easily port this Nuclei SDK software code to other IDEs.

Click *Overview* (page 1) to learn more about the Nuclei SDK project overview.

For example, we have ported Nuclei SDK to use Segger embedded Studio, IAR Workbench and PlatformIO.

5.1.1 Directory Structure

To learn deeper about Nuclei SDK project, the directory structure is a good start point.

Below, we will describe our design about the Nuclei SDK directory structure:

Here is the directory structure for this Nuclei SDK.

```
$NUCLEI_SDK_ROOT
├── application
│   ├── baremetal
│   ├── freertos
│   ├── ucosii
│   └── rtthread
├── Build
│   ├── gmsl
│   ├── toolchain
│   ├── Makefile.base
│   ├── Makefile.conf
│   ├── Makefile.core
│   ├── Makefile.components
│   ├── Makefile.files
│   ├── Makefile.global
│   ├── Makefile.misc
│   ├── Makefile.rtos
```

(continues on next page)

[46] https://doc.nucleisys.com/nmsis/core/core_templates.html

(continued from previous page)

```
    ├── Makefile.rules
    └── Makefile.soc
├── doc
    ├── build
    ├── source
    ├── Makefile
    └── requirements.txt
├── NMSIS
    ├── Core
    ├── DSP
    ├── NN
    └── Library
├── OS
    ├── FreeRTOS
    ├── UCOSII
    └── RTThread
├── SoC
    ├── gd32vf103
    └── evalsoc
├── test
    ├── core
    ├── ctest.h
    ├── LICENSE
    └── README.md
├── LICENSE
├── Makefile
├── NMSIS_VERSION
├── package.json
├── SConscript
├── README.md
├── setup.bat
└── setup.sh
```

- **application**

  This directory contains all the application softwares for this Nuclei SDK.

  The application code can be divided into mainly 4 parts, which are:

  - **Baremetal** applications, which will provide baremetal applications without any OS usage, these applications will be placed in *application/baremetal/* folder.
  - **FreeRTOS** applications, which will provide FreeRTOS applications using FreeRTOS RTOS, placed in *application/freertos/* folder.
  - **UCOSII** applications, which will provide UCOSII applications using UCOSII RTOS, placed in *application/ucosii/* folder.
  - **RTThread** applications, which will provide RT-Thread applications using RT-Thread RTOS, placed in *application/rtthread/* folder.

- **SoC**

  This directory contains all the supported SoCs for this Nuclei SDK, the directory name for SoC and its boards should always in lower case.

Here we mainly support Nuclei processor cores running in Nuclei FPGA evaluation board, the support package placed in *SoC/evalsoc/*.

In each SoC's include directory, *nuclei_sdk_soc.h* must be provided, and include the soc header file, for example, *SoC/evalsoc/Common/Include/nuclei_sdk_soc.h*.

In each SoC Board's include directory, *nuclei_sdk_hal.h* must be provided, and include the board header file, for example, *SoC/evalsoc/Board/nuclei_fpga_eval/Include/nuclei_sdk_hal.h*.

- **Build**

  This directory contains the key part of the build system based on Makefile for Nuclei SDK.

- **NMSIS**

  This directory contains the NMSIS header files, which is widely used in this Nuclei SDK, you can check the *NMSIS_VERSION* file to know the current *NMSIS* version used in **Nuclei-SDK**.

  We will also sync the changes in NMSIS project[47] when it provided a new release.

- **OS**

  This directory provided three RTOS package we suppported which are **FreeRTOS**, **UCOSII** and **RT-Thread**.

- **LICENSE**

  Nuclei SDK license file.

- **NMSIS_VERSION**

  NMSIS Version file. It will show current NMSIS version used in Nuclei SDK.

- **package.json**

  PlatformIO package json file for Nuclei SDK, used in Nuclei Platform for PlatformIO[48].

- **SConscript**

  RT-Thread package scons build script, used in RT-Thread package development[49].

- **Makefile**

  An external Makefile just for build, run, debug application without cd to any corresponding application directory, such as *application/baremetal/helloworld/*.

- **setup.sh**

  Nuclei SDK environment setup script for **Linux**. You need to create your own *setup_config.sh*.

```
# you can export this variable to Nuclei Studio's toolchain folder
NUCLEI_TOOL_ROOT=/path/to/your_tool_root
```

  In the **$NUCLEI_TOOL_ROOT** for **Linux**, you need to have Nuclei RISC-V GNU GCC toolchain and OpenOCD installed as below.

```
$NUCLEI_TOOL_ROOT
├── gcc
│   ├── bin
│   ├── include
│   ├── lib
│   ├── libexec
```

(continues on next page)

[47] https://github.com/Nuclei-Software/NMSIS
[48] https://platformio.org/platforms/nuclei/
[49] https://www.rt-thread.org/document/site/development-guide/package/package/

(continued from previous page)

```
│   ├── riscv64-unknown-elf
│   └── share
└── openocd
    ├── bin
    ├── contrib
    ├── distro-info
    ├── OpenULINK
    ├── scripts
    └── share
```

- **setup.bat**

  Nuclei SDK environment setup bat script for **Windows**. You need to create your own *setup_config.bat*.

```
set NUCLEI_TOOL_ROOT=\path\to\your_tool_root
```

  In the **%NUCLEI_TOOL_ROOT%** for **Windows**, you need to have Nuclei RISC-V GNU GCC toolchain, necessary Windows build tools and OpenOCD installed as below.

```
%NUCLEI_TOOL_ROOT%
├── build-tools
│   ├── bin
│   ├── gnu-mcu-eclipse
│   └── licenses
├── gcc
│   ├── bin
│   ├── include
│   ├── lib
│   ├── libexec
│   ├── riscv64-unknown-elf
│   └── share
└── openocd
    ├── bin
    ├── contrib
    ├── distro-info
    ├── OpenULINK
    ├── scripts
    └── share
```

5.1.2 Project Components

This Nuclei SDK project components is list as below:

- *Nuclei Processor* (page 63): How Nuclei Processor Core is used in Nuclei SDK
- *SoC* (page 65): How Nuclei processor code based SoC device is supported in Nuclei SDK
- *Board* (page 71): How Nuclei based SoC's Board is supported in Nuclei SDK
- *Peripheral* (page 85): How to use the peripheral driver in Nuclei SDK
- *RTOS* (page 86): What RTOSes are supported in Nuclei SDK
- *Application* (page 90): How to use pre-built applications in Nuclei SDK

5.2 Nuclei Processor

Nuclei processor core are following and compatible to RISC-V standard architecture, but there might be some additions and enhancements to the original standard spec.

Click Nuclei Spec[50] to learn more about Nuclei RISC-V Instruction Set Architecture.

5.2.1 Introduction

Nuclei provides the following RISC-V IP Products[51] for AIoT:

- **N100 series:** Designed for mixed digital and analog, IoT or other extremely low-power and small area scenarios, which is the perfect replacement of traditional 8051 cores, you need to use it with Nuclei N100 SDK[52] .
- **N200 series:** Designed for ultra-low power consumption and embedded scenarios, perfectly replaces the arm Cortex-M series cores.
- **N300 series:** Designed for extreme energy efficiency ratio, requiring DSP and FPU features, as IoT and industrial control scenarios.
- **600 series and 900 series:** Fully support Linux for high-performance edge computing and smart AIoT.
- **1000 series:** The UX1000 Series have three different variants: UX1030, UX1040 and UX1060. UX1030 is a 3-wide processor with good performance and smaller power & area; UX1040 is a 4-wide processor with better performance and balanced power & area; UX1060 is a 6-wide processor with even higher performance targeting high-end applications.

Note:

- **N100 series** is not supported by **NMSIS** and **Nuclei SDK**

5.2.2 NMSIS in Nuclei SDK

This Nuclei SDK is built based on the NMSIS[53] framework, user can access NMSIS Core API[54], NMSIS DSP API[55] and NMSIS NN API[56] provided by NMSIS[57].

These NMSIS APIs are mainly responsible for accessing Nuclei RISC-V Processor Core.

The prebuilt NMSIS-DSP and NMSIS-NN libraries are also provided in Nuclei SDK, see `NMSIS/Library/` folder.

Note:

- To support RT-Thread in Nuclei-SDK, we have to modify the **startup_<device>.S**, to use macro `RTOS_RTTHREAD` defined when using RT-Thread as below:

[50] https://doc.nucleisys.com/nuclei_spec/
[51] https://nucleisys.com/product.php
[52] https://doc.nucleisys.com/nuclei_n100_sdk
[53] https://github.com/Nuclei-Software/NMSIS
[54] https://doc.nucleisys.com/nmsis/core/api/index.html
[55] https://doc.nucleisys.com/nmsis/dsp/api/index.html
[56] https://doc.nucleisys.com/nmsis/nn/api/index.html
[57] https://github.com/Nuclei-Software/NMSIS

```
#ifdef RTOS_RTTHREAD
    // Call entry function when using RT-Thread
    call entry
#else
    call main
#endif
```

- In order to support RT-Thread initialization macros `INIT_XXX_EXPORT`, we also need to modify the link script files, add lines after `` *(.rodata .rodata.)`` as below:

```
. = ALIGN(4);
*(.rdata)
*(.rodata .rodata.*)
/* RT-Thread added lines begin */
/* section information for initial. */
. = ALIGN(4);
__rt_init_start = .;
KEEP(*(SORT(.rti_fn*)))
__rt_init_end = .;
/* section information for finsh shell */
. = ALIGN(4);
__fsymtab_start = .;
KEEP(*(FSymTab))
__fsymtab_end = .;
. = ALIGN(4);
__vsymtab_start = .;
KEEP(*(VSymTab))
__vsymtab_end = .;
/* RT-Thread added lines end */
*(.gnu.linkonce.r.*)
```

5.2.3 SoC Resource

Regarding the SoC Resource exclude the Nuclei RISC-V Processor Core, it mainly consists of different peripherals such UART, GPIO, I2C, SPI, CAN, PWM, DMA, USB and etc.

The APIs to access to the SoC resources are usually defined by the SoC Firmware Library Package provided by SoC Vendor.

In Nuclei SDK, currently we just required developer to provide the following common resources:

- A UART used to implement several stub functions for `printf` function
  - When using newlib library, please check stub functions list in `SoC/evalsoc/Common/Stubs/newlib`
  - When using libncrt library, please check stub functions list in `SoC/evalsoc/Common/Stubs/libncrt`
  - When using iar dlib library, please check stub functions list in `SoC/evalsoc/Common/Stubs/iardlib`
- Common initialization code defined in **System_<Device>.c/h** in each SoC support package in Nuclei SDK.
- Before enter to main function, these resources must be initialized:
  - The UART used to print must be initialized as `115200 bps, 8bit data, none parity check, 1 stop bit`

  - ECLIC MTH set to 0 using `ECLIC_SetMth`, means don't mask any interrupt
  - ECLIC NLBits set to `__ECLIC_INTCTLBITS`, means all the nlbits are for level
  - Global interrupt is disabled

Note:

- If you want to learn more about SoC, please click *SoC* (page 65)
- If you want to learn more about Board, please click *Board* (page 71)
- If you want to learn more about Peripheral, please click *Peripheral* (page 85)

5.3 SoC

5.3.1 Nuclei Demo SoC

Note: Since Hummingbird is already taken by the opensource Hummingbird E203 SoC, we just rename Hummingbird SoC in Nuclei SDK to Nuclei Demo SoC to make it more clear. For newer version of Nuclei CPU IP from 2022.07, which might has iregion feature, please use Eval SoC instead of Demo SoC.

Nuclei Demo SoC support in Nuclei SDK is removed in 0.5.0 release, please use *Nuclei Eval SoC* (page 65) now.

5.3.2 Nuclei Eval SoC

Note: Nuclei CPU IP now with iregion feature will use totally new evaluation SoC, this SoC is different from previous Demo SoC, please take care.

Nuclei DemoSoC is now removed in 0.5.0 release, and please use evalsoc now.

Nuclei Eval SoC is an evaluation FPGA SoC from Nuclei for customer to evaluate Nuclei RISC-V Process Core, and it is a successor for Demo SoC.

Overview

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

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, SPI etc.

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

If you want to learn more about this evaluation SoC, please get the <Nuclei_Eval_SoC_Intro.pdf> from Nuclei[58].

Supported Boards

In Nuclei SDK, we support the following boards based on **Nuclei Evaluation SoC**, see:

- *Nuclei FPGA Evaluation Kit* (page 71), default Board when this SoC selected.

Usage

Note: To ensure compatibility when using Nuclei EvalSoC(FPGA), please verify with our Application Engineer (AE) the specific CPU configuration to confirm if the EvalSoC's CPU possesses the features you intend to test. You can utilize the *cpuinfo* (page 92) application to determine the available CPU features on your system and cross-reference this information with the Nuclei ISA specifications.

Note: In latest CPU RTL generation flow, it will also generate an Nuclei SDK to match CPU and EvalSoC RTL configuration, please use the generated Nuclei SDK to evaluate your CPU and EvalSoC feature.

The generated Nuclei SDK by **nuclei_gen** will do the following tasks:

- Generate `SoC/evalsoc/cpufeature.mk`: which will define **CORE**, **ARCH_EXT**, **QEMU_SOCCFG** or **SIMULATION** default value.
- Generate `SoC/evalsoc/Common/Include/cpufeature.h`: which will define current cpu feature macros.
- Generate `SoC/evalsoc/evalsoc.json`: which will define current qemu soc configuration according to the evalsoc and cpu configuration.
- Generate `SoC/evalsoc/Board/nuclei_fpga_eval/Source/GCC/evalsoc.memory`: which will define the ilm/dlm/flash/ddr/sram base address and size.
- Modify `SoC/evalsoc/Board/nuclei_fpga_eval/openocd_evalsoc.cfg`: Mainly change `workmem_base/workmem_size/flashxip_base/xipnuspi_base` to adapt the evalsoc configuration.

If you want to use the generated Nuclei SDK by **nuclei_gen** In Nuclei Studio IDE, you need to zip it first, and then **import** it using `RV-Tools -> NPK Package Management` in Nuclei Studio IDE's menu, and when creating a IDE project using `New Nuclei RISC-V C/C++ Project`, please select the correct sdk and version which you can check it in the `<SDK>/npk.yml` file, and in the project example configuration wizard window, you should configure the **Nuclei RISC-V Core** and **ARCH Extensions**, **Nuclei Cache Extensions** according to your configured CPU ISA, and CPU feature defined in generated `cpufeature.h`.

WARNING: Currently you still need to modify IAR linker script(`*.icf`) by yourself, it is not automatically modified.

If you want to use this **Nuclei Evaluation SoC** in Nuclei SDK, you need to set the *SOC* (page 26) Makefile variable to `evalsoc`.

Note: IAR support is done by prebuilt IAR projects not through Makefile based build system, please check https://github.com/Nuclei-Software/nuclei-sdk/blob/master/ideprojects/iar/README.md for detailed usage.

---

[58] https://nucleisys.com/

Extra make variables supported only in this SoC and used internally only by Nuclei, not designed for widely used:

- **RUNMODE**: it is used internally by Nuclei, used to control ILM/DLM/ICache/DCache enable or disable via make variable, please check `SoC/evalsoc/runmode.mk` for details. It is not functional by default, unless you set a non-empty variable to this `RUNMODE` variable, it can be used with different **ILM_EN/DLM_EN/IC_EN/DC_EN/CCM_EN**.
- **L2_EN**: it is used internally by Nuclei, used to control L2 cache enable or disable, introduced in 0.6.0 release.
- **LDSPEC_EN**: it is used internally by Nuclei, used to control load speculative enable or disable, introduced in 0.6.0 release.
- **BPU_EN**: it is used internally by Nuclei, used to control branch prediction unit enable or disable, introduced in 0.6.0 release.
- **ECC_EN**: it is used internally by Nuclei, used to control (ilm/dlm/L1 I/Dcache)ecc unit enable or disable, introduced in 0.7.0 release.
- **XLCFG_xxx** make variables such as **XLCFG_CIDU**, **XLCFG_CCM**, **XLCFG_TEE** and **XLCFG_SMPU** which are used to overwrite default macros defined in `cpufeature.h` which will affect **XXX_PRESENT** macros in `evalsoc.h`, introduced in 0.7.0 release.
- **CODESIZE**: it is used to control whether remove all template routine code for interrupt and exception and banner print code to measure basic code size requirement for evalsoc when `CODESIZE=1`
- **SYSCLK**: it is used together with `CODESIZE=1` to overwrite default `SYSTEM_CLOCK` macro value for different bitstream, eg. `SYSCLK=50000000 CODESIZE=1`, it will set default SYSTEM_CLOCK to 50000000.
- **QEMU_MC_EXTOPT** is used to pass extra options to Nuclei Qemu `-M` machine options for evalsoc, please dont pass any extra `,` to this make variable, you can pass such as `QEMU_MC_EXTOPT=debug=1` but not pass `QEMU_MC_EXTOPT=,debug=1`
- **QEMU_CPU_EXTOPT** is used to pass extra options to Nuclei Qemu `-cpu` cpu options for evalsoc, please dont pass any extra `,` to this make variable, you can pass such as `QEMU_CPU_EXTOPT=vlen=512` but not pass `QEMU_CPU_EXTOPT=,vlen=512`

```
# Choose SoC to be evalsoc
# the following command will build application
# using default evalsoc SoC based board
# defined in Build System and application Makefile
make SOC=evalsoc info # you can check current working SDK configuration information
make SOC=evalsoc clean
make SOC=evalsoc all
```

5.3.3 GD32VF103 SoC

GD32VF103 SoC is the first general RISC-V MCU from GigaDevice Semiconductor[59] in the world which is based on Nuclei RISC-V Process Core.

If you want to learn more about it, please click https://www.gigadevice.com/products/microcontrollers/gd32/risc-v/

[59] https://www.gigadevice.com/

Overview

The GD32VF103 device is a 32-bit general-purpose micro controller based on the RISC-V core with best ratio in terms of processing power, reduced power consumption and peripheral set.

The RISC-V processor core is tightly coupled with an Enhancement Core-Local Interrupt Controller(ECLIC), SysTick timer and advanced debug support.

The GD32VF103 device incorporates the RISC-V 32-bit processor core operating at 108MHz frequency with Flash accesses zero wait states to obtain maximum efficiency.

It provides up to 128KB on-chip Flash memory and 32KB SRAM memory.

An extensive range of enhanced I/Os and peripherals connect to two APB buses.

The devices offer up to two 12-bit ADCs, up to two 12-bit DACs, up to four general 16-bit timers, two basic timers plus a PWM advanced timer, as well as standard and advanced communication interfaces: up to three SPIs, two I2Cs, three USARTs, two UARTs, two I2Ss, two CANs, an USBFS.

The SoC diagram can be checked as below *GD32VF103 SoC Diagram* (page 69)

Supported Boards

In Nuclei SDK, we support the following four boards based on **GD32VF103** SoC, see:

- *GD32VF103V RV-STAR Kit* (page 74), default Board when this SoC selected.
- *GD32VF103V Evaluation Kit* (page 76)
- *Sipeed Longan Nano* (page 77)
- *TTGO T-Display-GD32* (page 82)

Usage

If you want to use this **GD32VF103** SoC in Nuclei SDK, you need to set the *SOC* (page 26) Makefile variable to `gd32vf103`.

Extra make variables supported only in this SoC:

- **SYSCLK**: `108000000` by default, means 108MHz system clock will be selected during SystemInit function, it will define macro `SYSTEM_CLOCK=$(SYSCLK)` which is used in `system_gd32vf103.c`, such as SYSTEM_CLOCK=108000000.
- **CLKSRC**: hxtal by default, available choices are `hxtal` and `irc8m`, means select to use HXTAL PLL or IRC8M PLL, it will define macro `CLOCK_USING_$(CLKSRC)`, such as `CLOCK_USING_HXTAL`
- **USB_DRIVER**: none usb driver is selected by default. You can choose `device` or `host` or `both` to select device, host or both driver source code, and in application code, user need to provide usb host or device initialization code and header files.

```
# Choose SoC to be gd32vf103
# the following command will build application
# using default gd32vf103 SoC based board
# defined in Build System and application Makefile
make SOC=gd32vf103 clean
make SOC=gd32vf103 all
```

Note:

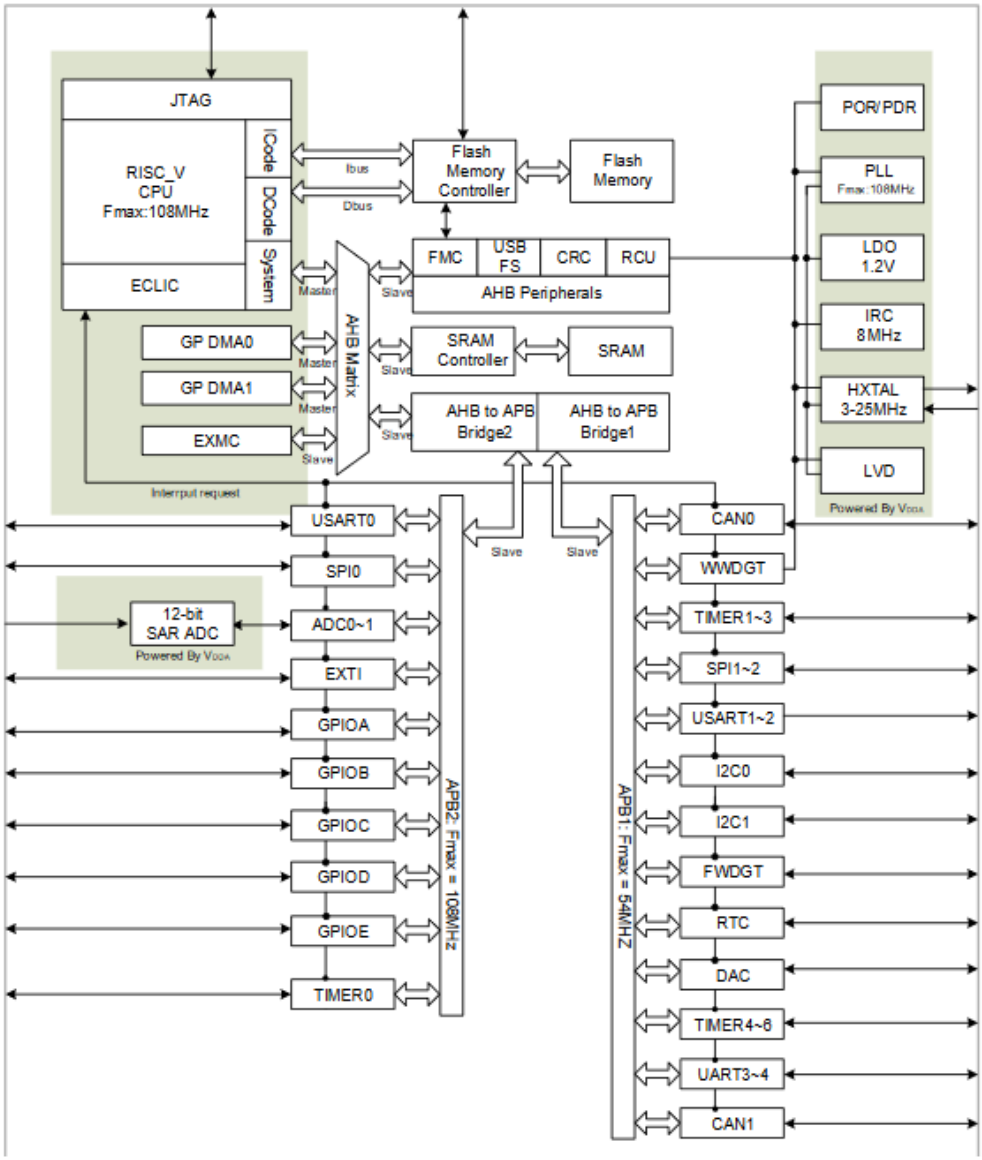


Fig. 1: GD32VF103 SoC Diagram

- Since this `gd32vf103` SoC is a real chip, it is using Nuclei RISC-V N205 core, so the **CORE** is fixed to `n205`
- **USB_DRV_SUPPORT** make variable is no longer available, please use **USB_DRIVER** variable to select different usb driver.
- You need to provide `usb_conf.h/usbd_conf.h/usbh_conf.h` file in you application code, if you want to use the usb driver of gd32vf103, see https://github.com/Nuclei-Software/nuclei-sdk/pull/54

5.3.4 GD32VW55x SoC

GD32VW55x SoC is an RISC-V WiFi/BLE MCU from GigaDevice Semiconductor[60] in the world which is based on Nuclei RISC-V N300 Processor.

If you want to learn more about it, please click https://www.gigadevice.com/about/news-and-event/news/gigadevice-launches-gd32vw553-series

Overview

The new GD32VW553 series integrates up to 4MB Flash, 320KB SRAM, and 32KB configurable Instruction Cache (I-Cache) to greatly improve CPU processing efficiency. The GD32VW553, delivering excellent wireless performance, is also equipped with rich universal wired interfaces, including three U(S)ART, two I2C, one SPI, one four-wire QSPI, and up to 29 programmable GPIO pins. Its built-in components include two 32-bit general-purpose timers, two 16-bit general-purpose timers, four 16-bit basic timers, one PWM advanced timer, and one 12-bit ADC. The power supply voltage ranges from 1.8 V to 3.6 V and it offers high temperature up to 105℃ to meet the application scenarios such as industrial control interconnection, lighting equipment, and socket panels.

Supported Boards

In Nuclei SDK, we support the following four boards based on **GD32VW55x** SoC, see:

- *GD32VW553H Evaluation Kit* (page 83), default Board when this SoC selected.

Usage

If you want to use this **GD32VW55x** SoC in Nuclei SDK, you need to set the *SOC* (page 26) Makefile variable to `gd32vw55x`.

Extra make variables supported only in this SoC(see `SoC/gd32vw55x/build.mk`):

- **SYSCLK**: `160000000` by default, means 160MHz system clock will be selected during `SystemInit` function, it will define macro `SYSTEM_CLOCK=$(SYSCLK)` which is used in `system_gd32vw55x.c`.
- **CLKSRC**: empty by default, available choices are `hxtal` and `irc16m`, means select to use HXTAL PLL or IRC16M PLL, it will define macro `CLOCK_USING_$(CLKSRC)`, such as `CLOCK_USING_HXTAL`

```
# Choose SoC to be gd32vw55x
# the following command will build application
# using default gd32vw55x SoC based board
# defined in Build System and application Makefile
make SOC=gd32vw55x clean
make SOC=gd32vw55x all
```

[60] https://www.gigadevice.com/

Note:

- Since this `gd32vw55x` SoC is a real chip, it is using Nuclei RISC-V N300 core, so the **CORE** is fixed to `n300fd`

5.4 Board

5.4.1 Nuclei FPGA Evaluation Kit

Overview

Nuclei have customized different FPGA evaluation boards (called Nuclei FPGA Evaluation Kit), which can be programmed with Nuclei Demo/Eval SoC FPGA bitstream.

- **Nuclei FPGA Evaluation Kit, 100T version**

  This **100T** version is a very early version which widely used since 2019, it has a Xilinx XC7A100T FPGA chip on the board.

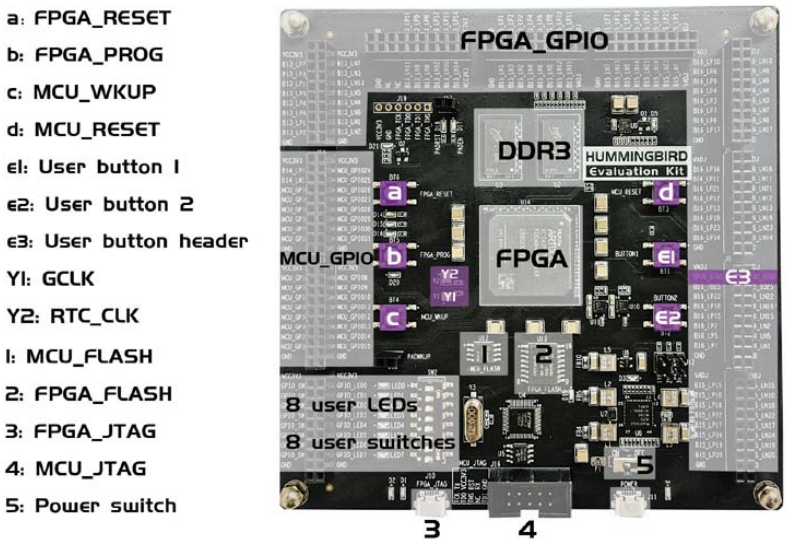


Fig. 2: Nuclei FPGA Evaluation Kit, 100T Version

- **Nuclei FPGA Evaluation Kit, DDR 200T version**

  This **DDR 200T** version is a latest version which provided since 2020.09, it has a Xilinx XC7A200T FPGA chip on the board, and the onboard DDR could be connected to Nuclei RISC-V Core.

  This board is a choice to replace the *100T version*, and it could be use to evaluate any Nuclei RISC-V core.

  We also use this version of board to evaluate Nuclei UX class core which can run Linux on it, it you want to run Linux on this board, please refer to Nuclei Linux SDK[61].

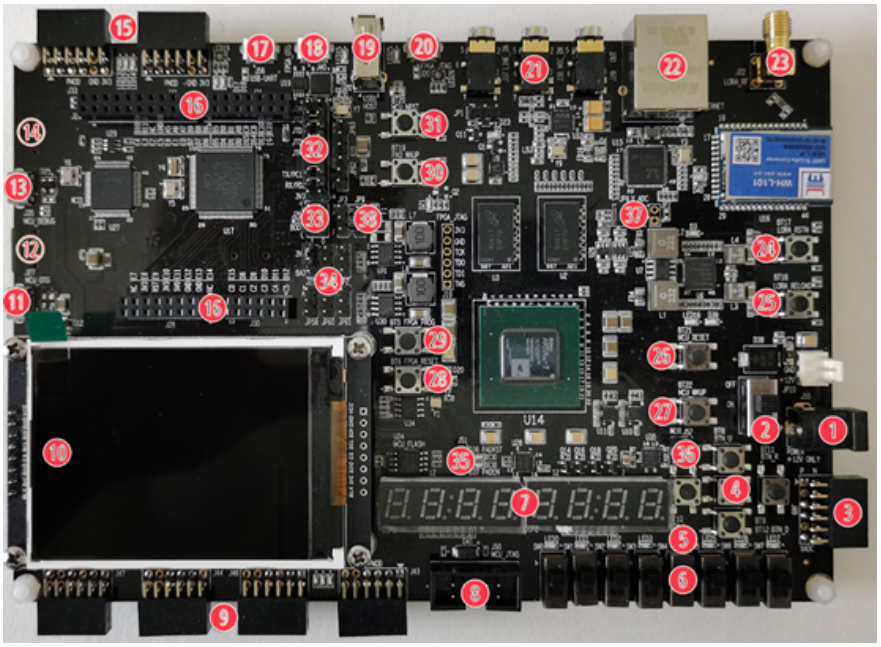


Fig. 3: Nuclei FPGA Evaluation Kit, DDR 200T Version

- **Nuclei FPGA Evaluation Kit, MCU 200T version**

  This **MCU 200T** version is a latest version which provided since 2020.09, it has a Xilinx XC7A200T FPGA chip on the board, but there is no DDR chip on the board.

  This board is a choice to replace the *100T version*, and it could be use to evaluate any Nuclei RISC-V core with don't use DDR.

There are also other fpga board we supported, such as KU060 and VCU118 board, please contact with our sales for details.

Click Nuclei FPGA Evaluation Kit Board Documents[62] to access the documents of these boards.

Setup

Follow the guide in Nuclei FPGA Evaluation Kit Board Documents[63] to setup the board, make sure the following items are set correctly:

- Use **Nuclei FPGA debugger** to connect the **MCU-JTAG** on board to your PC in order to download and debug programs and monitor the UART message.
- Power on the board using USB doggle(for 100T) or DC 12V Power(for MCU 200T or DDR 200T).
- The Nuclei FPGA SoC FPGA bitstream with Nuclei RISC-V evaluation core inside is programmed to FPGA on this board.
- Following steps in debugger kit manual[64] to setup JTAG drivers for your development environment

[61] https://github.com/Nuclei-Software/nuclei-linux-sdk
[62] https://nucleisys.com/developboard.php
[63] https://nucleisys.com/developboard.php
[64] https://www.nucleisys.com/theme/package/Nuclei_FPGA_DebugKit_Intro.pdf

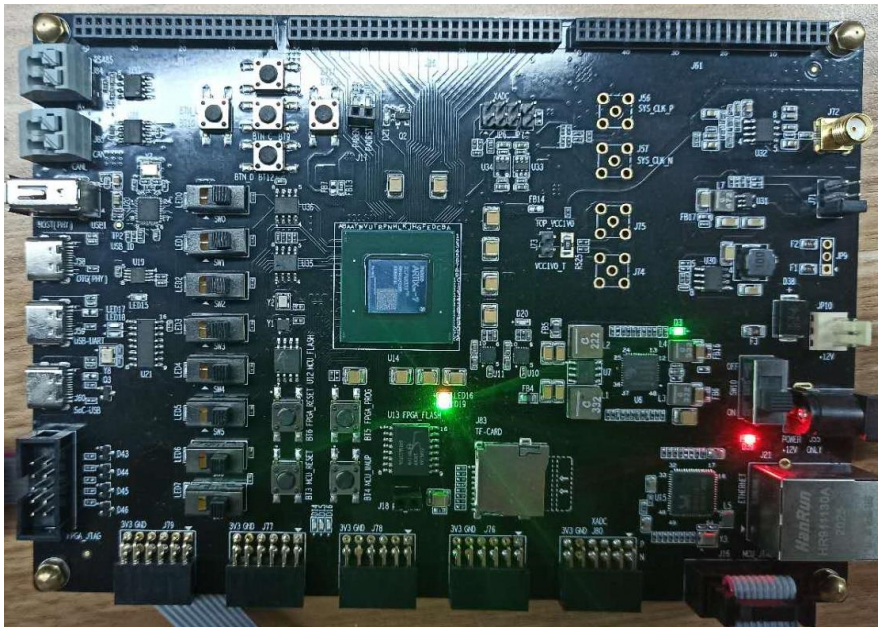


Fig. 4: Nuclei FPGA Evaluation Kit, MCU 200T Version

How to use

For **Nuclei FPGA Evaluation board**:

- `evalsoc` can run on this fpga board, please choose the correct SoC, `demosoc` support is removed in 0.5.0 release.
- **DOWNLOAD** support all the modes list in *DOWNLOAD* (page 29)
  - You can find default used linker scripts for different download modes in `SoC/evalsoc/Board/nuclei_fpga_eval/Source/GCC/`.
    - `gcc_evalsoc_ilm.ld`: Linker script file for `DOWNLOAD=ilm`
    - `gcc_evalsoc_flash.ld`: Linker script file for `DOWNLOAD=flash`
    - `gcc_evalsoc_flashxip.ld`: Linker script file for `DOWNLOAD=flashxip`
    - `gcc_evalsoc_sram.ld`: Linker script file for `DOWNLOAD=sram`
    - `gcc_evalsoc_ddr.ld`: Linker script file for `DOWNLOAD=ddr`. **Caution**: This download mode can be only used when DDR is connect to Nuclei RISC-V Core
  - If you want to specify your own modified linker script, you can follow steps described in *Change Link Script* (page 51)
  - If you want to change the base address or size of ILM, DLM, RAM, ROM or Flash of linker script file, you can adapt the Memory Section[65] in the linker script file it according to your SoC memory information.
- **CORE** support all the cores list in *CORE* (page 30)
- Its openocd configuration file can be found in `SoC/evalsoc/Board/nuclei_fpga_eval/openocd_evalsoc.cfg`

To run this application in Nuclei FPGA Evaluation board in Nuclei SDK, you just need to use this **SOC** and **BOARD** variables.

[65] https://sourceware.org/binutils/docs/ld/MEMORY.html

```
### For evalsoc
# Clean the application with DOWNLOAD=ilm CORE=n300f
make SOC=evalsoc BOARD=nuclei_fpga_eval DOWNLOAD=ilm CORE=n300f clean
# Build the application with DOWNLOAD=ilm CORE=n300f
make SOC=evalsoc BOARD=nuclei_fpga_eval DOWNLOAD=ilm CORE=n300f all
# Upload the application using openocd and gdb with DOWNLOAD=ilm CORE=n300f
make SOC=evalsoc BOARD=nuclei_fpga_eval DOWNLOAD=ilm CORE=n300f upload
# Debug the application using openocd and gdb with DOWNLOAD=ilm CORE=n300f
make SOC=evalsoc BOARD=nuclei_fpga_eval DOWNLOAD=ilm CORE=n300f debug
### For evalsoc
# Clean the application with DOWNLOAD=ilm CORE=n300f
make SOC=evalsoc BOARD=nuclei_fpga_eval DOWNLOAD=ilm CORE=n300f clean
# Upload the application using openocd and gdb with DOWNLOAD=ilm CORE=n300f
make SOC=evalsoc BOARD=nuclei_fpga_eval DOWNLOAD=ilm CORE=n300f upload
```

If you want to try other toolchain, such as nuclei llvm or terapines zcc, you can pass an extra *TOOLCHAIN* (page 28) make variable.

Note:

- `demosoc` support is removed, please use `evalsoc` now.
- You can change the value passed to **CORE** according to the Nuclei Demo SoC Evaluation Core the Nuclei FPGA SoC you have.
- You can also change the value passed to **DOWNLOAD** to run program in different modes.
- The FreeRTOS and UCOSII demos maybe not working in `flashxip` download mode in Nuclei FPGA board due to program running in Flash is really too slow. If you want to try these demos, please use `ilm` or `flash` download mode.

5.4.2 GD32VF103V RV-STAR Kit

Overview

This GD32VF103V RV-STAR Kit is an arduino compatiable board from Nuclei using GD32VF103VBT6 as main MCU.

Click GD32VF103V RV-STAR Development Kit[66] to access the documents of this board.

Click online RV-STAR Development Board Overview[67] to get basic information of this board.

[66] https://nucleisys.com/developboard.php
[67] https://doc.nucleisys.com/nuclei_board_labs/hw/hw.html#rv-star

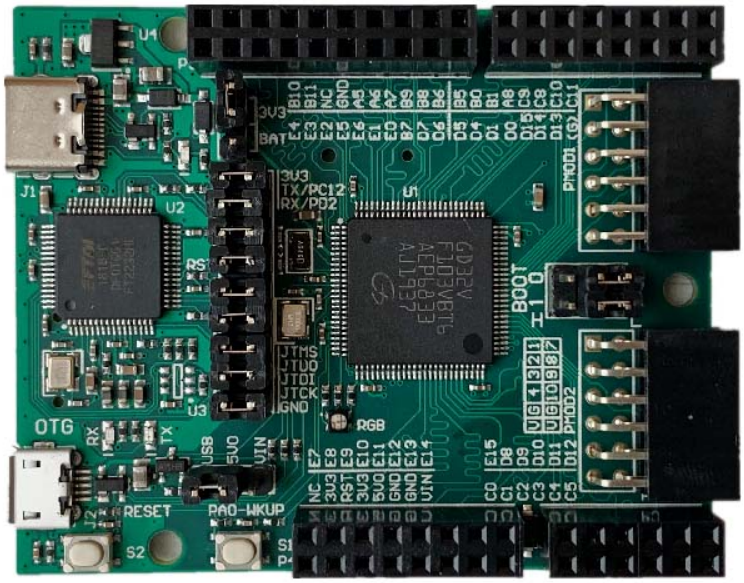


Fig. 5: GD32VF103V RV-STAR Board

Setup

Follow the guide in GD32VF103V RV-STAR Development Kit[68] to setup the board, make sure the following items are set correctly:

- Connect the USB Type-C port on board to your PC in order to download and debug programs and monitor the UART message.
- Following steps in RV-STAR user manual[69] to setup JTAG drivers for your development environment

How to use

For **GD32VF103V RV-STAR** board, the **DOWNLOAD** and **CORE** variables are fixed to `flashxip` and `n205`.

- You can find its linker script in `SoC/gd32vf103/Board/gd32vf103v_rvstar/Source/GCC/`
  - `gcc_gd32vf103_flashxip.ld`: Linker script file for `DOWNLOAD=flashxip`
- If you want to specify your own modified linker script, you can follow steps described in *Change Link Script* (page 51)
- You can find its openocd configuration file in `SoC/gd32vf103/Board/gd32vf103v_rvstar/openocd_gd32vf103.cfg`

To run this application in GD32VF103V RV-STAR board in Nuclei SDK, you just need to use this **SOC** and **BOARD** variables.

[68] https://nucleisys.com/developboard.php
[69] https://doc.nucleisys.com/nuclei_board_labs/hw/hw.html#on-board-debugger-driver

```
# Clean the application
make SOC=gd32vf103 BOARD=gd32vf103v_rvstar clean
# Build the application
make SOC=gd32vf103 BOARD=gd32vf103v_rvstar all
# Upload the application using openocd and gdb
make SOC=gd32vf103 BOARD=gd32vf103v_rvstar upload
# Debug the application using openocd and gdb
make SOC=gd32vf103 BOARD=gd32vf103v_rvstar debug
```

5.4.3 GD32VF103V Evaluation Kit

Overview

This GD32VF103V Evaluation Kit is an evaluation board from gigadevice using GD32VF103VBT6 as main MCU.

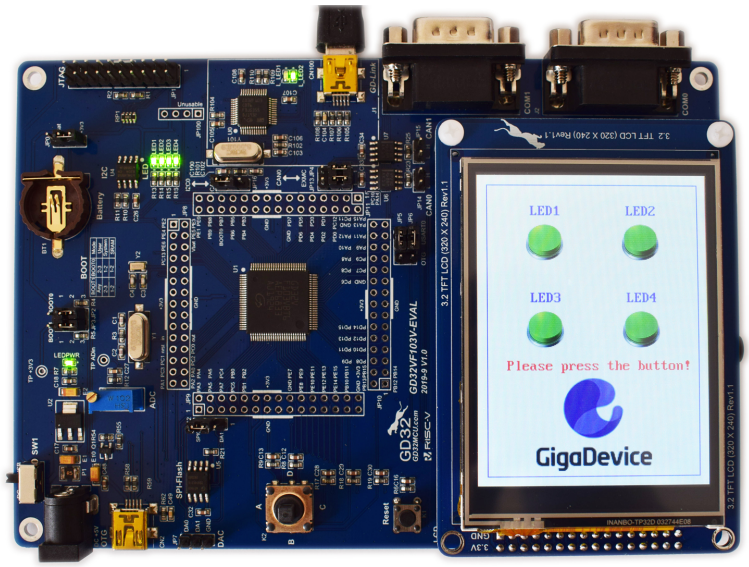


Fig. 6: GD32VF103V-EVAL Board

If you want to learn about this board, please click GD32VF103V EVAL Board Documents[70].

Setup

Follow the guide in GD32VF103V EVAL Board Documents[71] to setup the board, make sure the following items are set correctly:

- Connect the GD-Link on board to your PC in order to download and debug programs.
- Select the correct boot mode and then power on, the LEDPWR will turn on, which indicates the power supply is ready
- Connect the `COM0` to your PC
- Following steps in board user manual to setup JTAG drivers for your development environment

[70] https://github.com/riscv-mcu/GD32VF103_Demo_Suites/tree/master/GD32VF103V_EVAL_Demo_Suites/Docs
[71] https://github.com/riscv-mcu/GD32VF103_Demo_Suites/tree/master/GD32VF103V_EVAL_Demo_Suites/Docs

How to use

For **GD32VF103V-EVAL** board, the **DOWNLOAD** and **CORE** variables are fixed to `flashxip` and `n205`.

- You can find its linker script in `SoC/gd32vf103/Board/gd32vf103v_eval/Source/GCC/`
  - `gcc_gd32vf103_flashxip.ld`: Linker script file for `DOWNLOAD=flashxip`
- If you want to specify your own modified linker script, you can follow steps described in *Change Link Script* (page 51)
- You can find its openocd configuration file in `SoC/gd32vf103/Board/gd32vf103v_eval/openocd_gd32vf103.cfg`

To run this application in GD32VF103V-EVAL board in Nuclei SDK, you just need to use this **SOC** and **BOARD** variables.

```
# Clean the application
make SOC=gd32vf103 BOARD=gd32vf103v_eval clean
# Build the application
make SOC=gd32vf103 BOARD=gd32vf103v_eval all
# Upload the application using openocd and gdb
make SOC=gd32vf103 BOARD=gd32vf103v_eval upload
# Debug the application using openocd and gdb
make SOC=gd32vf103 BOARD=gd32vf103v_eval debug
```

5.4.4 Sipeed Longan Nano

Overview

The Sipeed Longan Nano is a board made by Sipeed using a GD32VF103CBT6 as main MCU. Is is similar to the well known STM32-based **Blue Pill** board.

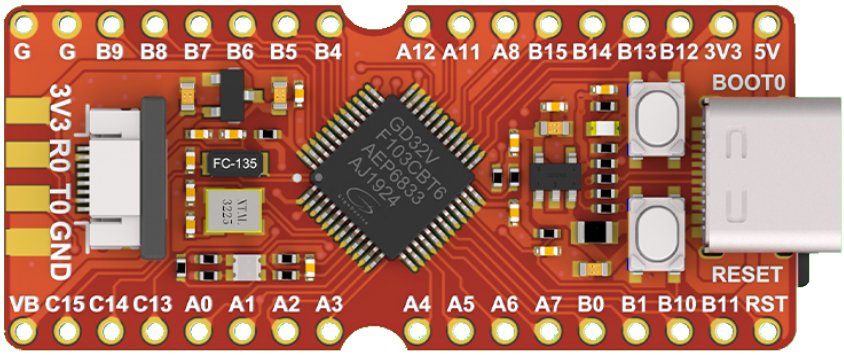


Fig. 7: Sipeed Longan Nano Board.

Versions

There are two versions of this board available.

- GD32VF103C**B**T6 with 128k Flash / 32k RAM
- GD32VF103C**8**T6 with 64k Flash / 20k RAM. This is sometimes called the **lite** version.

If you want to buy one, carefully take a look at the description because sometimes they are offered with the GD32VF103CB controller, but they only contain the GD32VF103C8 controller.

Pinout

The pinout of Sipeed Logan Nano is shown in the following picture

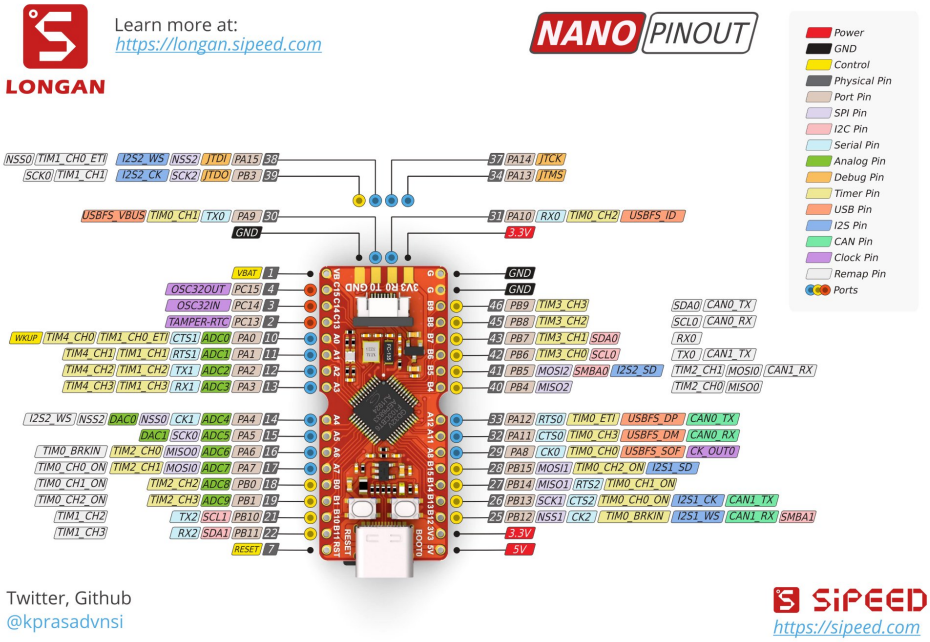


Fig. 8: Sipeed Longan Nano Pinout.

Schematic

Resources

Click Sipeed Longan Nano Documentation[72] to get all information about this board from Sipeed website.

[72] https://longan.sipeed.com/en/

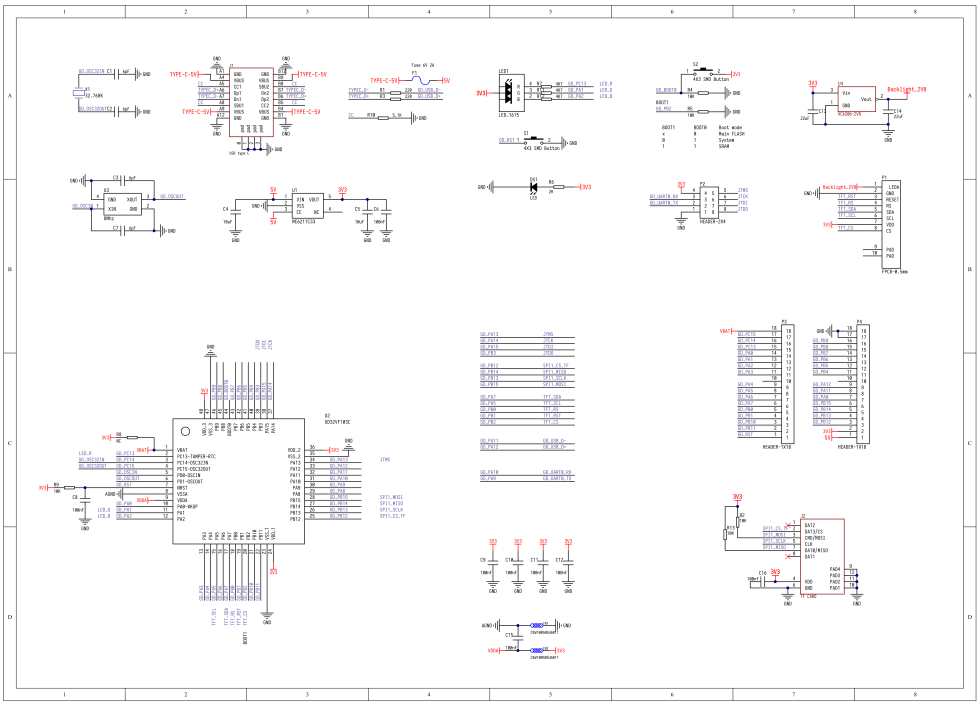


Fig. 9: Sipeed Longan Nano Schematic.

Setup

To setup the board, make sure the following items are set correctly:

- Power up the board by either the USB-C port **or** the by the debugger.
- The default serial port is USART0, whitch is also available at the debug header. See *Sipeed Longan Nano Pinout.* (page 78)

How to use

For **Sipeed Longan Nano** board, the **DOWNLOAD** and **CORE** variables are fixed to `flashxip` and `n205`. The **VARIANT** variable can be used for choosing a board variant.

- You can find its linker scripts in `SoC/gd32vf103/Board/gd32vf103c_longan_nano/Source/GCC/`
  - `gcc_gd32vf103xb_flashxip.ld`: Linker script file for `DOWNLOAD=flashxip` and 128k flash, this is the default.
  - `gcc_gd32vf103x8_flashxip.ld`: Linker script file for `DOWNLOAD=flashxip` and 64k flash, the **lite** version, you can pass extra `VARIANT=lite` via make command to select this linker script.
- If you want to specify your own modified linker script, you can follow steps described in *Change Link Script* (page 51)
- You can find its openocd configuration file in `SoC/gd32vf103/Board/gd32vf103c_longan_nano/openocd_gd32vf103.cfg`

To run this application in Sipeed Longan Nano board in Nuclei SDK, you just need to use this **SOC** and **BOARD** variables.

```
# Clean the application
make SOC=gd32vf103 BOARD=gd32vf103c_longan_nano clean
# Build the application
make SOC=gd32vf103 BOARD=gd32vf103c_longan_nano all
# Upload the application using openocd and gdb
make SOC=gd32vf103 BOARD=gd32vf103c_longan_nano upload
# Debug the application using openocd and gdb
make SOC=gd32vf103 BOARD=gd32vf103c_longan_nano debug
```

To build for the **lite** variant you also need to set the **VARIANT** variable.

```
# Build the application
make SOC=gd32vf103 BOARD=gd32vf103c_longan_nano VARIANT=lite all
```

Extensions

There are three extensions on the board:

- On the back of the circuit board there is a socket for a micro SD card.
  - The SD-card is connected to SPI1.
- On the front there is a socket for a small LCD which is offered by some sellers.
  - The LCD is connected to SPI0.
  - The controller on the LCD is similar to Sitronix' ST7735.
- One RGB-LED
  - The red LED is controlled via PC13. This LED can be addressed by LED3 or LEDR.
  - The green LED is controlled via PA1. This LED can be addressed by LED1 or LEDG.
  - The blue LED is controlled via PA2 This LED can be addressed by LED2 or LEDB.

There are two buttons on the board. One is the reset button and the other is to activate the internal bootloader. Unfortunately, none of these buttons can be used as user inputs.

5.4.5 GD32VF103C DLink Debugger

Overview

This GD32VF103C DLink Debugger is used to debug Nuclei RISC-V CPU from Nuclei using **GD32VF103CVBT6** as main MCU.

Click https://github.com/nuclei-Software/nuclei-dlink to learn more about Nuclei DLink project.

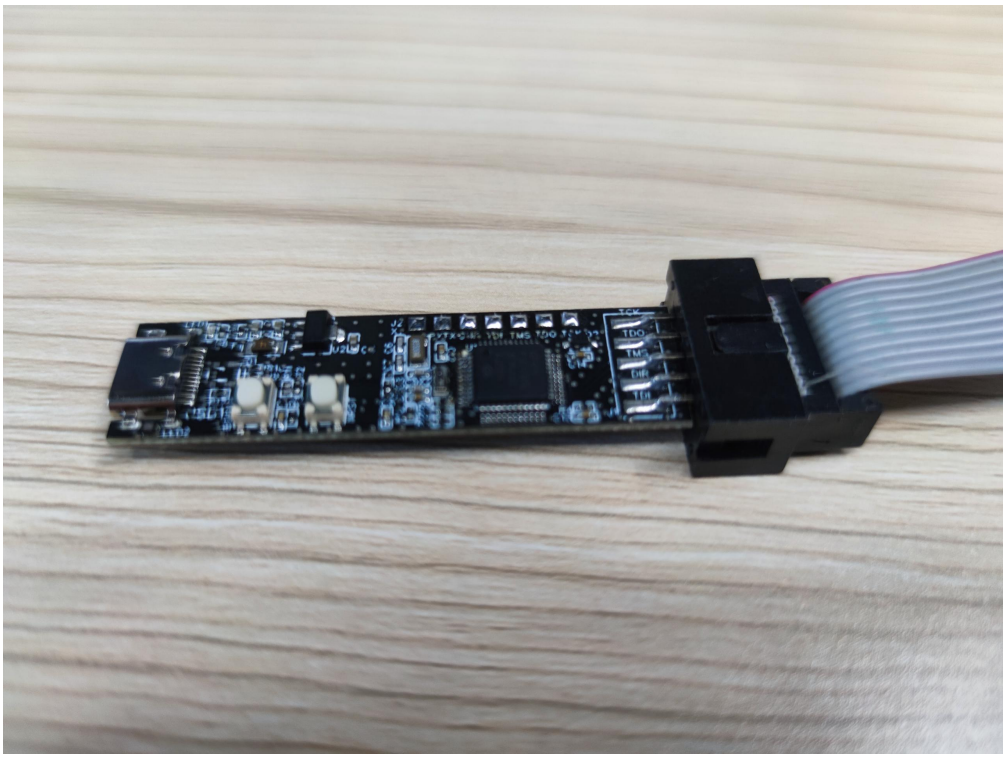


Fig. 10: GD32VF103C DLink Debugger Board

Setup

How to use

For **GD32VF103C DLink Debugger** board, the **DOWNLOAD** and **CORE** variables are fixed to `flashxip` and `n205`.

- You can find its linker script in `SoC/gd32vf103/Board/gd32vf103c_dlink/Source/GCC/`
  - `gcc_gd32vf103_flashxip.ld`: Linker script file for `DOWNLOAD=flashxip`
- If you want to specify your own modified linker script, you can follow steps described in *Change Link Script* (page 51)
- You can find its openocd configuration file in `SoC/gd32vf103/Board/gd32vf103c_dlink/openocd_gd32vf103.cfg`

To run this application in GD32VF103C DLink Debugger board in Nuclei SDK, you just need to use this **SOC** and **BOARD** variables.

```
# Clean the application
make SOC=gd32vf103 BOARD=gd32vf103c_dlink clean
# Build the application
make SOC=gd32vf103 BOARD=gd32vf103c_dlink all
# Upload the application using openocd and gdb
make SOC=gd32vf103 BOARD=gd32vf103c_dlink upload
# Debug the application using openocd and gdb
make SOC=gd32vf103 BOARD=gd32vf103c_dlink debug
```

5.4.6 TTGO T-Display-GD32

Overview

The TTGO T-Display-GD32[73] is a minimal board from LilyGo using the GD32VF103CBT6 as main MCU.

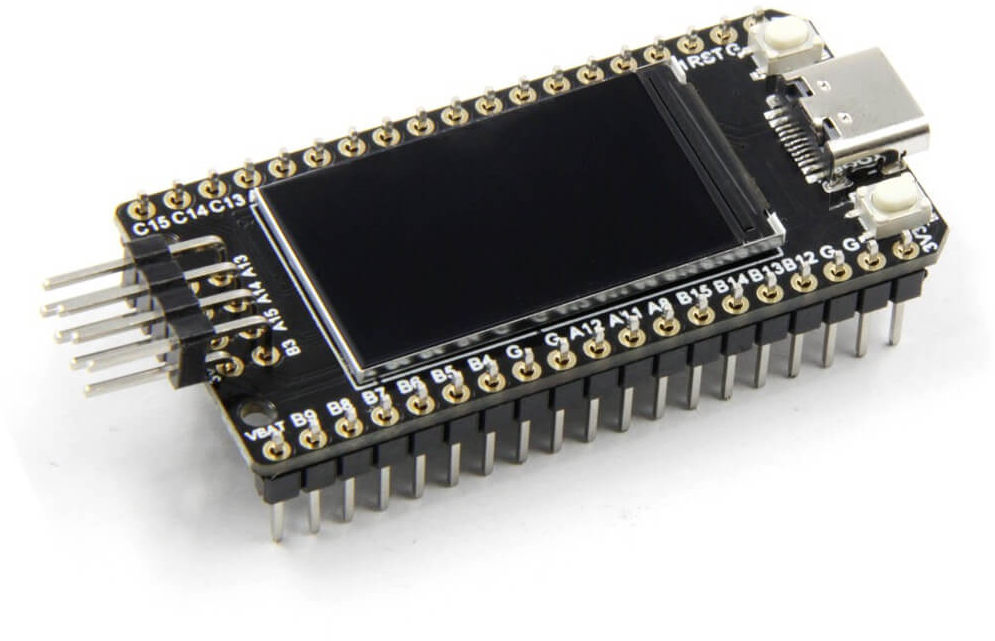


Fig. 11: TTGO T-Display-GD32 Board

Setup

Wire your JTAG debugger as following. Below table assumes the Sipeed USB-JTAG/TTL RISC-V Debugger. With other brands the pin namings should be the same. You also need to power up the board via USB.

Debugger	TTGO T-Display-GD32
GND	GND
RXD	PA9
TXD	PA10
NC	
GND	GND (optional)
TDI	PA15
RST	RST
TMS	PA13
TDO	PB3
TCK	PA14

[73] http://www.lilygo.cn/prod_view.aspx?TypeId=50033&Id=1251&FId=t3:50033:3

How to use

For **TTGO T-Display-GD32** board, the **DOWNLOAD** and **CORE** variables are fixed to `flashxip` and `n205`.

- You can find its linker script in `SoC/gd32vf103/Board/gd32vf103c_t_display/Source/GCC/gcc_gd32vf103_flashxip.ld`
- If you want to specify your own modified linker script, you can follow steps described in *Change Link Script* (page 51)
- You can find its openocd configuration file in `SoC/gd32vf103/Board/gd32vf103c_t_display/openocd_gd32vf103.cfg`

To run this application in TTGO T-Display-GD32 board in Nuclei SDK, you just need to use this **SOC** and **BOARD** variables.

```
# Clean the application
make SOC=gd32vf103 BOARD=gd32vf103c_t_display clean
# Build the application
make SOC=gd32vf103 BOARD=gd32vf103c_t_display all
# Upload the application using openocd and gdb
make SOC=gd32vf103 BOARD=gd32vf103c_t_display upload
# Debug the application using openocd and gdb
make SOC=gd32vf103 BOARD=gd32vf103c_t_display debug
```

5.4.7 GD32VW553H Evaluation Kit

Overview

This GD32VW553H Evaluation Kit is an evaluation board from gigadevice using GD32VW553HM as main MCU.

If you want to learn about this board, please click GD32VW553H EVAL Board Documents[74].

Setup

Follow the guide in GD32VW553H EVAL Board Documents[75] to setup the board, make sure the following items are set correctly:

- Connect the **GD-Link** on board to your PC in order to download and debug programs.
- Connect the `USART` to your PC as UART communication.
- Following steps in board user manual to setup JTAG drivers for your development environment

[74] https://www.gd32mcu.com/en/download/8?kw=GD32VW5
[75] https://www.gd32mcu.com/en/download/8?kw=GD32VW5

Fig. 12: GD32VW553H EVAL Board

How to use

For **GD32VW553H-EVAL** board:

- **DOWNLOAD**: `flashxip` by default, and you can also choose `sram` download mode
  - You can find its linker script in `SoC/gd32vw55x/Board/gd32vw553h_eval/Source/GCC/`
  - `gcc_gd32vw55x_flashxip.ld`: Linker script file for `DOWNLOAD=flashxip`
  - `gcc_gd32vw55x_sram.ld`: Linker script file for `DOWNLOAD=sram`
  - If you want to specify your own modified linker script, you can follow steps described in *Change Link Script* (page 51)
- **CORE**: `n300fd` by default, this by default is `rv32imafdc` arch, but you can also choose `n300` or `n300f`
- **ARCH_EXT**: `_zba_zbb_zbc_zbs_xxldspn1x` by default, you can pass less extensions such as `_zba_zbb_zbc_zbs`
- You can find its openocd configuration file in `SoC/gd32vw55x/Board/gd32vw553h_eval/openocd_gd32vw55x.cfg`

To run this application in GD32VW553H-EVAL board in Nuclei SDK, you just need to use this **SOC** and **BOARD** variables.

```
# Clean the application
make SOC=gd32vw55x BOARD=gd32vw553h_eval clean
# Build the application
make SOC=gd32vw55x BOARD=gd32vw553h_eval all
# Upload the application using openocd and gdb
make SOC=gd32vw55x BOARD=gd32vw553h_eval upload
# Debug the application using openocd and gdb
make SOC=gd32vw55x BOARD=gd32vw553h_eval debug
```

5.5 Peripheral

5.5.1 Overview

Regarding the peripheral support(such as UART, GPIO, SPI, I2C and etc.) in Nuclei SDK, we didn't define a device or peripheral layer for different SoCs, so the peripheral drivers are directly tighted with each SoC, and if developer want to use the drivers, they can directly use the driver API defined in each SoC.

Considering this peripheral driver difference in each SoC, if you want to write portable code in Nuclei SDK, you can use include the `nuclei_sdk_soc.h`, then you can write application which only use the resources of Nuclei Core.

If you want to use all the board resources, you can include the `nuclei_sdk_hal.h`, then you can write application for your own board, but the application can only run in the board you provided.

5.5.2 Usage

If you want to learn about what peripheral driver you can use, you can check the `nuclei_sdk_soc.h` of each SoC, and `nuclei_sdk_hal.h` of each board.

For SoC firmware library APIs:

- You can find the **GD32VF103 SoC firmware library APIs** in `SoC/gd32vf103/Common/Include`
- You can find the **GD32VW55x SoC firmware library APIs** in `SoC/gd32vw55x/Common/Include`
- You can find the **Nuclei Eval SoC firmware library APIs** in `SoC/evalsoc/Common/Include`

If you just want to use SoC firmware library API, you just need to include `nuclei_sdk_soc.h`, then you can use the all the APIs in that SoC include directory.

Note: For GD32VF103 SoC, if you want to use the USB driver API, then you need to add `USB_DRIVER = both` in your application to enable both host and device driver.

For Board related APIs:

- You can find the **GD32VF103 EVAL Board related APIs** in `SoC/gd32vf103/Board/gd32vf103v_eval/Include`
- You can find the **GD32VF103 RV-STAR Board related APIs** in `SoC/gd32vf103/Board/gd32vf103v_rvstar/Include`
- You can find the **Sipeed Longan Nano Board related APIs** in `SoC/gd32vf103/Board/gd32vf103c_longan_nano/Include`
- You can find the **Nuclei FPGA Evaluation Board related APIs** in `SoC/evalsoc/Board/nuclei_fpga_eval/Include`
- You can find the **TTGO T-Display-GD32 related APIs** in `SoC/gd32vf103/Board/gd32vf103c_t_display/Include`

If you just want to use all the APIs of Board and SoC, you just need to include `nuclei_sdk_hal.h`, then you can use the all the APIs in that Board and SoC include directory.

5.6 RTOS

5.6.1 Overview

In Nuclei SDK, we have support four most-used RTOSes in the world, **FreeRTOS**, **UCOSII**, **ThreadX** and **RT-Thread from China**.

Our RTOS port require Nuclei ECLIC interrupt controller, please make sure your Nuclei CPU is configured with ECLIC present.

If you want to use RTOS in your application, you can choose one of the supported RTOSes.

Note: When you want to develop RTOS application in Nuclei SDK, please don't reconfigure `SysTimer` and `SysTimer Software Interrupt`, since it is already used by RTOS portable code.

5.6.2 FreeRTOS

FreeRTOS[76] is a market-leading real-time operating system (RTOS) for microcontrollers and small microprocessors.

In our FreeRTOS portable code, we are using `SysTimer Interrupt` as RTOS SysTick Interrupt, and using `SysTimer Software Interrupt` to do task switch.

These two interrupts are kept as lowest level, and `SysTimer Interrupt` is initialized as non-vector interrupt, and `SysTimer Software Interrupt` is initialized as vector interrupt and interrupt handler implemented using asm code.

In our FreeRTOS porting, we also allow FreeRTOS configuration variable `configMAX_SYSCALL_INTERRUPT_PRIORITY` which can be find in https://www.freertos.org/a00110.html.

The `configMAX_SYSCALL_INTERRUPT_PRIORITY` should be set to be a absolute interrupt level range from 1 to (2^lvlbits-1) while `lvlbits = min(nlbits, CLICINTCTLBITS)`.

If you set configMAX_SYSCALL_INTERRUPT_PRIORITY to value above the accepted value range, it will use the max value.

If you want to learn about how to use FreeRTOS APIs, you need to go to its website to learn the FreeRTOS documentation in its website.

In Nuclei SDK, if you want to use **FreeRTOS** in your application, you need to add `RTOS = FreeRTOS` in your application Makefile.

And in your application code, you need to do the following things:

- Add FreeRTOS configuration file -> `FreeRTOSConfig.h`
- Include FreeRTOS header files

Now we also support FreeRTOS SMP version, about SMP version, please refer to https://www.freertos.org/symmetric-multiprocessing-introduction.html , and we also provide freertos smpdemo example in our SDK, you can find it in `application\freertos\smpdemo`.

To use FreeRTOS SMP version for 2 Core SMP CPU, you need to add `SMP = 2` in your application Makefile. And also you need to make sure your application code is placed and run on shared memory which can be accessed by both CPUs. When `SMP=2` is specified, it will define extra requried macro called `configNUMBER_OF_CORES`, for details, please check `OS/FreeRTOS/build.mk`.

Note:

- You can check the `application\freertos\` for freertos application reference
- From Nuclei SDK 0.6.0, we introduced FreeRTOS SMP support, both Nuclei RV32 and RV64 processors are supported.
- Current version of FreeRTOS used in Nuclei SDK is `V11.1.0`
- If you want to change the OS ticks per seconds, you can change the `configTICK_RATE_HZ` defined in `FreeRTOSConfig.h`

More information about FreeRTOS get started, please click https://www.freertos.org/FreeRTOS-quick-start-guide.html

[76] https://www.freertos.org/

5.6.3 UCOSII

UCOSII[77] a priority-based preemptive real-time kernel for microprocessors, written mostly in the programming language C. It is intended for use in embedded systems.

In our UCOSII portable code, we are using `SysTimer Interrupt` as RTOS SysTick Interrupt, and using `SysTimer Software Interrupt` to do task switch.

If you want to learn about `UCOSII`, please click https://www.micrium.com/books/ucosii/

We are using the opensource version of UC-OS2 source code from https://github.com/SiliconLabs/uC-OS2, with optimized code for Nuclei RISC-V processors.

In Nuclei SDK, if you want to use **UCOSII** in your application, you need to add `RTOS = UCOSII` in your application Makefile.

And in your application code, you need to do the following things:

- Add UCOSII application configuration header file -> `app_cfg.h` and `os_cfg.h`
- Add application hook source file -> `app_hooks.c`
- Include UCOSII header files

Note:

- You can check the `application\ucosii\` for ucosii application reference
- The UCOS-II application configuration template files can also be found in https://github.com/SiliconLabs/uC-OS2/tree/master/Cfg/Template
- Current version of UCOSII used in Nuclei SDK is `V2.93.00`
- If you want to change the OS ticks per seconds, you can change the `OS_TICKS_PER_SEC` defined in `os_cfg.h`

Warning:

- For Nuclei SDK release > v0.2.2, the UCOSII source code is replaced using the version from https://github.com/SiliconLabs/uC-OS2/, and application development for UCOSII is also changed, the `app_cfg.h`, `os_cfg.h` and `app_hooks.c` files are required in application source code.

5.6.4 RT-Thread

RT-Thread (page 88) was born in 2006, it is an open source, neutral, and community-based real-time operating system (RTOS).

RT-Thread is mainly written in C language, easy to understand and easy to port(can be quickly port to a wide range of mainstream MCUs and module chips).

It applies object-oriented programming methods to real-time system design, making the code elegant, structured, modular, and very tailorable.

In our support for RT-Thread, we get the source code of RT-Thread from a project called RT-Thread Nano[78], which only provide kernel code of RT-Thread, which is easy to be integrated with Nuclei SDK.

[77] https://www.micrium.com/
[78] https://github.com/RT-Thread/rtthread-nano

In our RT-Thread portable code, we are using `SysTimer Interrupt` as RTOS SysTick Interrupt, and using `SysTimer Software Interrupt` to do task switch.

And also the `rt_hw_board_init` function is implemented in our portable code.

If you want to learn about `RT-Thread`, please click:

- For Chinese version, click https://www.rt-thread.org/document/site/
- For English version, click https://github.com/RT-Thread/rt-thread#documentation

In Nuclei SDK, if you want to use **RT-Thread** in your application, you need to add `RTOS = RTThread` in your application Makefile.

And in your application code, you need to do the following things:

- Add RT-Thread application configuration header file -> `rtconfig.h`
- Include RT-Thread header files
- If you want to enable RT-Thread MSH feature, just add `RTTHREAD_MSH := 1` in your application Makefile.

Note:

- You can check the `application\rtthread\` for rtthread application reference
- In RT-Thread, the `main` function is created as a RT-Thread thread, so you don't need to do any OS initialization work, it is done before `main`
- We also provide good support directly through RT-Thread official repo, you can check Nuclei processor support for RT-Thread in RT-Thread BSP For Nuclei[79].

5.6.5 ThreadX

Eclipse ThreadX[80] offers a vendor-neutral, open source, safety certified OS for real-time applications, all under a permissive license. It stands alone as the first and only RTOS with this unique blend of attributes to meet a wide range of needs that will benefit industry adopters, developers and end users alike.

Microsoft has contributed the Azure RTOS technology to the Eclipse Foundation. With the Eclipse Foundation as its new home, Azure RTOS now becomes Eclipse ThreadX – an advanced embedded development suite including a small but powerful operating system that provides reliable, ultra-fast performance for resource-constrained devices.

ThreadX is IEC 61508, IEC 62304, ISO 26262, and EN 50128 conformance certified by SGS-TÜV Saar. ThreadX has also achieved EAL4+ Common Criteria security certification. These certifications are a big differentiator, and are unprecedented in the industry. They are a game changer, as there are currently no open source RTOS's which have them.

In our ThreadX portable code, we are using `SysTimer Interrupt` as RTOS SysTick Interrupt, and using `SysTimer Software Interrupt` to do task switch.

If you want to learn about `Eclipse ThreadX`, please click:

- For introduction of Eclipse ThreadX, click https://eclipse-foundation.blog/2023/11/21/introducing-eclipse-threadx/
- For ThreadX documentation, click https://github.com/eclipse-threadx/rtos-docs/blob/main/rtos-docs/threadx/index.md

[79] https://github.com/RT-Thread/rt-thread/tree/master/bsp/nuclei/
[80] https://github.com/eclipse-threadx/threadx

In Nuclei SDK, if you want to use **ThreadX** in your application, you need to add `RTOS = ThreadX` in your application Makefile.

And in your application code, you need to do the following things:

- Add ThreadX application configuration header file -> `tx_user.h`
- Include ThreadX header files

Note:

- You can check the `application\threadx\` for threadx application reference
- Currently we only support single core version, the SMP version is not yet supported.

5.7 Application

5.7.1 Overview

In Nuclei SDK, we just provided applications which can run in different boards without any changes in code to demonstrate the baremetal service, freertos service and ucosii service features.

The provided applications can be divided into three categories:

- Bare-metal applications: Located in `application/baremetal`
- FreeRTOS applications: Located in `application/freertos`
- UCOSII applications: Located in `application/ucosii`
- RTThread applications: Located in `application/rtthread`
- ThreadX applications: Located in `application/threadx`

If you want to find more examples, please visit the following links:

- Nuclei Board Labs: https://github.com/Nuclei-Software/nuclei-board-labs
- Nuclei Tensorflow Lite Micro AI Demo: https://github.com/Nuclei-Software/npk-tflm
- Nuclei Tinymaix TinyAI Demo: https://github.com/Nuclei-Software/npk-tinymaix
- NMSIS DSP Examples: https://doc.nucleisys.com/nmsis/dsp/get_started.html#how-to-run
- NMSIS NN Examples: https://doc.nucleisys.com/nmsis/nn/get_started.html#how-to-run
- NMSIS Crypto(MbedTLS) Examples: https://github.com/Nuclei-Software/mbedtls/blob/nuclei/v3.3.0/accelerator/README.md

And we can also provide more examples to test cpu features, please contact with our AE for help.

If you want to develop your own application in Nuclei SDK, please click *Application Development* (page 49) to learn more about it.

The following applications are running using RV-STAR board or Nuclei Eval SoC.

Note:

- Since 0.7.0 introduced support for CLINT and PLIC interrupt mode, if you are working in such interrupt mode or don't have ECLIC module, then all RTOSes will not able to run in your environment, due to RTOS port require ECLIC interrupt.

- Most of the application demostrated below using `SOC=gd32vf103`, you can easily change it to other SoC such as evalsoc by change it to `SOC=evalsoc`
- Some applications may not be able to be run on your SoC using Nuclei CPU due to lack of cpu feature required to run on it.
- Almost all the applications required Nuclei CPU configured with ECLIC and System Timer hardware feature.
- Almost all the application required UART to print message, so you need to implement an UART drivers and clib stub functions, if you use *SEMIHOST* (page 33) to print message, it is not required.

5.7.2 Bare-metal applications

helloworld

This helloworld application[81] is used to print hello world, and also will check this RISC-V CSR **MISA** register value.

How to run this application:

```
# Assume that you can set up the Tools and Nuclei SDK environment
# cd to the helloworld directory
cd application/baremetal/helloworld
# Clean the application first
make SOC=gd32vf103 clean
# Build and upload the application
make SOC=gd32vf103 upload
```

Expected output as below:

```
Nuclei SDK Build Time: Feb 21 2020, 12:24:22
Download Mode: FLASHXIP
CPU Frequency 109323529 Hz
MISA: 0x40901105
MISA: RV32IMACUX
0: Hello World From Nuclei RISC-V Processor!
1: Hello World From Nuclei RISC-V Processor!
2: Hello World From Nuclei RISC-V Processor!
3: Hello World From Nuclei RISC-V Processor!
4: Hello World From Nuclei RISC-V Processor!
5: Hello World From Nuclei RISC-V Processor!
6: Hello World From Nuclei RISC-V Processor!
7: Hello World From Nuclei RISC-V Processor!
8: Hello World From Nuclei RISC-V Processor!
9: Hello World From Nuclei RISC-V Processor!
10: Hello World From Nuclei RISC-V Processor!
11: Hello World From Nuclei RISC-V Processor!
12: Hello World From Nuclei RISC-V Processor!
13: Hello World From Nuclei RISC-V Processor!
14: Hello World From Nuclei RISC-V Processor!
15: Hello World From Nuclei RISC-V Processor!
16: Hello World From Nuclei RISC-V Processor!
17: Hello World From Nuclei RISC-V Processor!
```

(continues on next page)

[81] https://github.com/Nuclei-Software/nuclei-sdk/tree/master/application/baremetal/helloworld

(continued from previous page)

```
18: Hello World From Nuclei RISC-V Processor!
19: Hello World From Nuclei RISC-V Processor!
```

cpuinfo

This cpuinfo application[82] is used to print the Nuclei RISC-V CPU information to help you to know what CPU features are present in this processor.

You can also use openocd to probe the cpu feature, see https://doc.nucleisys.com/nuclei_tools/openocd/intro.html#nuclei-customized-features

How to run this application:

```
# Assume that you can set up the Tools and Nuclei SDK environment
# cd to the helloworld directory
cd application/baremetal/cpuinfo
# Assume to run on UX900 SMPx8 CPU
# Clean the application first
make SOC=evalsoc DOWNLOAD=sram clean
# Build and upload the application
make SOC=evalsoc DOWNLOAD=sram upload
```

Expected output as below:

```
Nuclei SDK Build Time: May 28 2024, 13:36:12
Download Mode: SRAM
CPU Frequency 50322800 Hz
CPU HartID: 0

-----Nuclei RISC-V CPU Configuration Information-----
        MARCHID: 0x900
        MIMPID: 0x30900
            ISA: RV64 A B C D F I M S U Zc Xxlcz
            MCFG: ECLIC PLIC ICACHE DCACHE SMP ZC_XLCZ_EXT IREGION No-Safety-Mechanism␣
↪DLEN=VLEN/2
        ICACHE: 64 KB(set=512,way=2,lsize=64,ecc=0)
        DCACHE: 64 KB(set=512,way=2,lsize=64,ecc=0)
            TLB: MainTLB(set=256,way=4,entry=1,ecc=0) ITLB(entry=16) DTLB(entry=16)
        IREGION: 0x18000000 128 MB
                Unit        Size        Address
                INFO        64KB        0x18000000
                DEBUG       64KB        0x18010000
                ECLIC       64KB        0x18020000
                TIMER       64KB        0x18030000
                SMP         64KB        0x18040000
                CIDU        64KB        0x18050000
                PLIC        64MB        0x1c000000
        SMP_CFG: CC_PRESENT=1 SMP_CORE_NUM=7 IOCP_NUM=0 PMON_NUM=4
        ECLIC: VERSION=0x0 NUM_INTERRUPT=71 CLICINTCTLBITS=3 MTH=0 NLBITS=3
        L2CACHE: 2 MB(set=2048,way=16,lsize=64,ecc=0)
    INFO-Detail:
```

(continues on next page)

[82] https://github.com/Nuclei-Software/nuclei-sdk/tree/master/application/baremetal/cpuinfo

(continued from previous page)

```
                mpasize : 32
-----End of Nuclei CPU INFO-----
```

demo_timer

This demo_timer application[83] is used to demonstrate how to use the CORE TIMER API including the Timer Interrupt and Timer Software Interrupt in ECLIC interrupt mode.

- Both interrupts are registered as non-vector interrupt.
- First the timer interrupt will run for 5 times
- Then the software timer interrupt will start to run for 5 times

How to run this application:

```
# Assume that you can set up the Tools and Nuclei SDK environment
# cd to the demo_timer directory
cd application/baremetal/demo_timer
# Clean the application first
make SOC=gd32vf103 clean
# Build and upload the application
make SOC=gd32vf103 upload
```

Expected output as below:

```
Nuclei SDK Build Time: Feb 21 2020, 12:52:37
Download Mode: FLASHXIP
CPU Frequency 108794117 Hz
init timer and start
MTimer IRQ handler 1
MTimer IRQ handler 2
MTimer IRQ handler 3
MTimer IRQ handler 4
MTimer IRQ handler 5
MTimer SW IRQ handler 1
MTimer SW IRQ handler 2
MTimer SW IRQ handler 3
MTimer SW IRQ handler 4
MTimer SW IRQ handler 5
MTimer msip and mtip interrupt test finish and pass
```

[83] https://github.com/Nuclei-Software/nuclei-sdk/tree/master/application/baremetal/demo_timer

demo_clint_timer

This demo_clint_timer application[84] is used to demonstrate how to use the CORE TIMER API including the Timer Interrupt and Timer Software Interrupt in CLINT interrupt mode.

- Interrupt is set to working in CLINT interrupt mode
- Both interrupts are registered as core interrupt.
- First the timer interrupt will run for 5 times
- Then the software timer interrupt will start to run for 5 times
- **NOTE**: not able to working in qemu, and only works for evalsoc

How to run this application:

```
# Assume that you can set up the Tools and Nuclei SDK environment
# cd to the demo_timer directory
cd application/baremetal/demo_clint_timer
# Clean the application first
make SOC=evalsoc clean
# Build and upload the application
make SOC=evalsoc upload
```

Expected output as below:

```
Nuclei SDK Build Time: Jul 25 2024, 10:39:39
Download Mode: ILM
CPU Frequency 16000614 Hz
CPU HartID: 0
init timer and start
SysTimer IRQ handler 1
SysTimer IRQ handler 2
SysTimer IRQ handler 3
SysTimer IRQ handler 4
SysTimer IRQ handler 5
SysTimer SW IRQ handler 1
SysTimer SW IRQ handler 2
SysTimer SW IRQ handler 3
SysTimer SW IRQ handler 4
SysTimer SW IRQ handler 5
SysTimer MTIP and MSIP CLINT interrupt test finish and pass
```

demo_eclic

This demo_eclic application[85] is used to demonstrate how to use the ECLIC API and Interrupt is working in ECLIC interrupt mode.

Note: In this application's Makefile, we provided comments in Makefile about optimize for code size.

If you want to optimize this application for code size, you can set the `COMMON_FLAGS` variable to the following values, we recommend to use `-Os -flto`.

[84] https://github.com/Nuclei-Software/nuclei-sdk/tree/master/application/baremetal/demo_clint_timer
[85] https://github.com/Nuclei-Software/nuclei-sdk/tree/master/application/baremetal/demo_eclic

Table 1: Code size optimization for demo_eclic on RV-STAR target

COMMON_FLAGS	text(bytes)	data(bytes)	bss(bytes)	total(bytes)
	13724	112	2266	16102
-flto	13598	112	2266	15976
-Os	9690	112	2264	12066
-Os -flto	9132	112	2264	11508
-Os -msave-restore -fno-unroll-loops	9714	112	2264	12090
-Os -msave-restore -fno-unroll-loops -flto	9204	112	2264	11580

- The timer interrupt and timer software interrupt are used
- The timer interrupt is registered as non-vector interrupt
- The timer software interrupt is registered as vector interrupt, and we enable its preemptive feature by using `SAVE_IRQ_CSR_CONTEXT` and `RESTORE_IRQ_CSR_CONTEXT` in timer software interrupt handler
- The timer interrupt is triggered periodically
- The timer software interrupt is triggered in timer interrupt handler using `SysTimer_SetSWIRQ` function
- In the application code, there is a macro called `SWIRQ_INTLEVEL_HIGHER` to control the timer software interrupt working feature:
  - If **SWIRQ_INTLEVEL_HIGHER=1**, the timer software interrupt level is higher than timer interrupt level, so when timer software interrupt is triggered, then timer software interrupt will be processed immediately, and timer interrupt will be preempted by timer software interrupt.
  - If **SWIRQ_INTLEVEL_HIGHER=0**, the timer software interrupt level is lower than timer interrupt level, so when timer software interrupt is triggered, then timer software interrupt will be processed after timer interrupt, and timer interrupt will not be preempted by timer software interrupt.

How to run this application:

```
# Assume that you can set up the Tools and Nuclei SDK environment
# cd to the demo_eclic directory
cd application/baremetal/demo_eclic
# Change macro SWIRQ_INTLEVEL_HIGHER value in demo_eclic.c
# to see different working mode of this demo
# Clean the application first
make SOC=gd32vf103 clean
# Build and upload the application
make SOC=gd32vf103 upload
```

Expected output(SWIRQ_INTLEVEL_HIGHER=1) as below:

```
Nuclei SDK Build Time: Feb 21 2020, 16:35:58
Download Mode: FLASHXIP
CPU Frequency 108794117 Hz
Initialize timer and start timer interrupt periodically
--------------------
[IN TIMER INTERRUPT]timer interrupt hit 0 times
[IN TIMER INTERRUPT]trigger software interrupt
[IN TIMER INTERRUPT]software interrupt will run during timer interrupt
[IN SOFTWARE INTERRUPT]software interrupt hit 0 times
```

(continues on next page)

(continued from previous page)

```
[IN SOFTWARE INTERRUPT]software interrupt end
[IN TIMER INTERRUPT]timer interrupt end
------------------
[IN TIMER INTERRUPT]timer interrupt hit 1 times
[IN TIMER INTERRUPT]trigger software interrupt
[IN TIMER INTERRUPT]software interrupt will run during timer interrupt
[IN SOFTWARE INTERRUPT]software interrupt hit 1 times
[IN SOFTWARE INTERRUPT]software interrupt end
[IN TIMER INTERRUPT]timer interrupt end
------------------
[IN TIMER INTERRUPT]timer interrupt hit 2 times
[IN TIMER INTERRUPT]trigger software interrupt
[IN TIMER INTERRUPT]software interrupt will run during timer interrupt
[IN SOFTWARE INTERRUPT]software interrupt hit 2 times
[IN SOFTWARE INTERRUPT]software interrupt end
[IN TIMER INTERRUPT]timer interrupt end
------------------
[IN TIMER INTERRUPT]timer interrupt hit 3 times
[IN TIMER INTERRUPT]trigger software interrupt
[IN TIMER INTERRUPT]software interrupt will run during timer interrupt
[IN SOFTWARE INTERRUPT]software interrupt hit 3 times
[IN SOFTWARE INTERRUPT]software interrupt end
[IN TIMER INTERRUPT]timer interrupt end
```

Expected output(SWIRQ_INTLEVEL_HIGHER=0) as below:

```
Nuclei SDK Build Time: Feb 21 2020, 16:35:58
Download Mode: FLASHXIP
CPU Frequency 108794117 Hz
Initialize timer and start timer interrupt periodically
------------------
[IN TIMER INTERRUPT]timer interrupt hit 0 times
[IN TIMER INTERRUPT]trigger software interrupt
[IN TIMER INTERRUPT]software interrupt will run when timer interrupt finished
[IN TIMER INTERRUPT]timer interrupt end
[IN SOFTWARE INTERRUPT]software interrupt hit 0 times
[IN SOFTWARE INTERRUPT]software interrupt end
------------------
[IN TIMER INTERRUPT]timer interrupt hit 1 times
[IN TIMER INTERRUPT]trigger software interrupt
[IN TIMER INTERRUPT]software interrupt will run when timer interrupt finished
[IN TIMER INTERRUPT]timer interrupt end
[IN SOFTWARE INTERRUPT]software interrupt hit 1 times
[IN SOFTWARE INTERRUPT]software interrupt end
------------------
[IN TIMER INTERRUPT]timer interrupt hit 2 times
[IN TIMER INTERRUPT]trigger software interrupt
[IN TIMER INTERRUPT]software interrupt will run when timer interrupt finished
[IN TIMER INTERRUPT]timer interrupt end
[IN SOFTWARE INTERRUPT]software interrupt hit 2 times
[IN SOFTWARE INTERRUPT]software interrupt end
------------------
```

(continues on next page)

(continued from previous page)

```
[IN TIMER INTERRUPT]timer interrupt hit 3 times
[IN TIMER INTERRUPT]trigger software interrupt
[IN TIMER INTERRUPT]software interrupt will run when timer interrupt finished
[IN TIMER INTERRUPT]timer interrupt end
[IN SOFTWARE INTERRUPT]software interrupt hit 3 times
[IN SOFTWARE INTERRUPT]software interrupt end
```

demo_plic

This demo_plic application[86] is used to demonstrate how to use the PLIC API and Interrupt is working in CLINT/PLIC interrupt mode.

Note: This demo only works on evalsoc, and require PLIC module present.

- This demo will show how to use plic external interrupt
- This demo use uart rx interrupt
- **NOTE**: not able to working in qemu

How to run this application:

```
# Assume that you can set up the Tools and Nuclei SDK environment
# cd to the demo_plic directory
cd application/baremetal/demo_plic
# For this case, if your bit has PLIC, and you are not using sdk generated by nuclei_gen
# XLCFG_PLIC=1 will define CFG_HAS_PLIC macro
make SOC=evalsoc XLCFG_PLIC=1 clean
# Build and upload the application
make SOC=evalsoc XLCFG_PLIC=1 upload
```

```
Nuclei SDK Build Time: Jul 23 2024, 17:49:27
Download Mode: ILM
CPU Frequency 50000000 Hz
CPU HartID: 0
You can press any key now to trigger uart receive interrupt
Enter uart0 interrupt, you just typed: 1
Enter uart0 interrupt, you just typed: 2
```

demo_dsp

This demo_dsp application[87] is used to demonstrate how to NMSIS-DSP API.

- Mainly show how we can use NMSIS DSP library and header files.
- It mainly demo the `riscv_conv_xx` functions and its reference functions
- By default, the application will use prebuilt NMSIS-DSP library match riscv isa arch defined by *CORE* (page 30) and *ARCH_EXT* (page 31)

[86] https://github.com/Nuclei-Software/nuclei-sdk/tree/master/application/baremetal/demo_plic
[87] https://github.com/Nuclei-Software/nuclei-sdk/tree/master/application/baremetal/demo_dsp

- If your selected **CORE** and **ARCH_EXT** don't have a prebuilt NMSIS DSP library, you can use *NMSIS_LIB_ARCH* (page 38) make variable to select another most suitable prebuilt NMSIS DSP or NN library.

eg. If you build with `make CORE=n900f ARCH_EXT=_zca_zcb_zcf_zcmp_zcmt_xxldsp clean all`, you will get a link error like this `riscv64-unknown-elf/bin/ld: cannot find -lnmsis_dsp_rv32imaf_zca_zcb_zcf_zcmp_zcmt_xxldsp: No such file or directory`, this is caused by no prebuilt libraries are built with `Zc*` extension. But you can build it by modify this example's Makefile by adding `NMSIS_LIB_ARCH := rv32imafc_xxldsp` newline.

Take care `Zc*` is not fully compatiable with `C` extension, especially when `D` extension present, see latest RISC-V ISA manual `riscv-unprivileged.pdf` which can found in https://github.com/riscv/riscv-isa-manual/releases .

And if you want to modify and build your own NMSIS DSP and NN library and see other DSP and NN examples and test cases, you can checkout the following links:

- https://github.com/Nuclei-Software/NMSIS
- https://doc.nucleisys.com/nmsis/dsp/index.html
- https://doc.nucleisys.com/nmsis/nn/index.html

Note:

- For other Nuclei Processor Core based SoC, please check whether it has DSP feature enabled to decide which kind of **NMSIS-DSP** library to use.
- Even our NMSIS-DSP library with DSP disabled are also optimized, so it can also provide good performance in some functions.

How to run this application:

```
# Assume that you can set up the Tools and Nuclei SDK environment
# cd to the demo_dsp directory
cd application/baremetal/demo_dsp
# Clean the application first
make SOC=gd32vf103 clean
# Build and upload the application
make SOC=gd32vf103 upload
```

Expected output as below:

```
Nuclei SDK Build Time: Jun 18 2020, 17:43:31
Download Mode: FLASHXIP
CPU Frequency 108270000 Hz
CSV, riscv_conv_q31, 1225418
CSV, ref_conv_q31, 2666240
SUCCESS, riscv_conv_q31
CSV, riscv_conv_q15, 289940
CSV, ref_conv_q15, 311158
SUCCESS, riscv_conv_q15
CSV, riscv_conv_q7, 463
CSV, ref_conv_q7, 846
SUCCESS, riscv_conv_q7
CSV, riscv_conv_fast_q15, 106293
CSV, ref_conv_fast_q15, 247938
SUCCESS, riscv_conv_fast_q15
CSV, riscv_conv_fast_q31, 490539
```

(continues on next page)

(continued from previous page)

```
CSV, ref_conv_fast_q31, 2215917
SUCCESS, riscv_conv_fast_q31
CSV, riscv_conv_opt_q15, 217250
CSV, ref_conv_opt_q15, 311162
SUCCESS, riscv_conv_opt_q15
CSV, riscv_conv_opt_q7, 714
CSV, ref_conv_opt_q7, 842
SUCCESS, riscv_conv_opt_q7
CSV, riscv_conv_fast_opt_q15, 137252
CSV, ref_conv_fast_opt_q15, 249958
SUCCESS, riscv_conv_fast_opt_q15
all test are passed. Well done!
```

lowpower

This lowpower application[88] is used to demonstrate how to use low-power feature of RISC-V processor.

Timer interrupt is setup before enter to wfi mode, and global interrupt will be disabled, so interrupt handler will not be entered, and will directly resume to next pc of wfi.

How to run this application:

```
# Assume that you can set up the Tools and Nuclei SDK environment
# Assume your processor has enabled low-power feature
# cd to the low-power directory
cd application/baremetal/lowpower
# Clean the application first
make SOC=evalsoc BOARD=nuclei_fpga_eval DOWNLOAD=ilm CORE=n300 clean
# Build and upload the application
make SOC=evalsoc BOARD=nuclei_fpga_eval DOWNLOAD=ilm CORE=n300 upload
```

Expected output as below:

```
Nuclei SDK Build Time: Jun  9 2022, 11:23:14
Download Mode: ILM
CPU Frequency 15996354 Hz
CSV, WFI Start/End, 178264/178289
CSV, WFI Cost, 25
```

smphello

This smphello application[89] is used to demonstrate how to use baremetal SMP feature.

This demo requests the SMP cores share the same RAM and ROM, for example, in current evalsoc system, ilm/dlm are private resource for cpu, only the DDR/SRAM memory are shared resource for all the cpu.

And *RVA* atomic extension is required to run this application, this extension is used to do spinlock in this example.

Note:

[88] https://github.com/Nuclei-Software/nuclei-sdk/tree/master/application/baremetal/lowpower
[89] https://github.com/Nuclei-Software/nuclei-sdk/tree/master/application/baremetal/smphello

- It doesn't work with gd32vf103 processor.
- **MUST** Need to enable I/D Cache in <Device.h> if I/D Cache present in CPU.
- It needs at least a 2-Core SMP CPU

- Each hart must wait until all the harts stop printing(or just stay in `while (1);` after its job has finished), because the `_postmain_fini` will print some dummy '0', which has no lock-protecting to UART causing `corrupted-printing`
- `spinlock lock` should be `volatile`, or else the compiler maybe optimize out the `spinlock_unlock` if more than one pair of `spinlock_lock` `spinlock_unlock` used in one function/branch, causing the lock unreleased

How to run this application:

```
# Assume that you can set up the Tools and Nuclei SDK environment
# Use Nuclei UX900 SMP 2 Core RISC-V processor as example
# application needs to run in ddr memory not in ilm memory
# cd to the smphello directory
cd application/baremetal/smphello
# Clean the application first
make SOC=evalsoc BOARD=nuclei_fpga_eval SMP=2 CORE=ux900 clean
# Build and upload the application
make SOC=evalsoc BOARD=nuclei_fpga_eval SMP=2 CORE=ux900 upload
```

Expected output as below:

```
Nuclei SDK Build Time: May 30 2022, 15:38:00
Download Mode: SRAM
CPU Frequency 15998648 Hz
Hello world from hart 0
Hello world from hart 1
All harts boot successfully!
```

demo_nice

Note:

- It doesn't work with gd32vf103 processor.
- Need nice feature enabled, and Nuclei NICE hardware demo integrated such as evalsoc

---

This demo_nice application[90] is used to demonstrate how to Nuclei NICE feature.

NICE is short for Nuclei Instruction Co-unit Extension, which is used to support extensive customization and specialization.

NICE allows customers to create user-defined instructions, enabling the integrations of custom hardware co-units that improve domain-specific performance while reducing power consumption.

For more about **NICE** feature, please click Nuclei User Extended Introduction[91].

- Mainly show how to use NICE intrinsic function with compiler.

[90] https://github.com/Nuclei-Software/nuclei-sdk/tree/master/application/baremetal/demo_nice
[91] https://doc.nucleisys.com/nuclei_spec/isa/nice.html

- It only works with Nuclei RISC-V Processor with the hardware NICE demo integrated.

How to run this application:

```
# Assume that you can set up the Tools and Nuclei SDK environment
# Use Nuclei UX900 RISC-V processor as example, hardware NICE demo integrated
# cd to the demo_dsp directory
cd application/baremetal/demo_nice
# Clean the application first
make SOC=evalsoc BOARD=nuclei_fpga_eval CORE=ux900 clean
# Build and upload the application
make SOC=evalsoc BOARD=nuclei_fpga_eval CORE=ux900 upload
```

Expected output as below:

```
Nuclei SDK Build Time: May 28 2024, 13:32:18
Download Mode: ILM
CPU Frequency 49999631 Hz
CPU HartID: 0

Nuclei Nice Acceleration Demonstration
Warning: This demo required CPU to implement Nuclei provided NICE Demo instructions.
        Otherwise this example will trap to cpu core exception!

1. Print input matrix array
the element of array is :
        10      30      90
        20      40      80
        30      90      120

2. Do reference matrix column sum and row sum
3. Do nice matrix column sum and row sum
4. Compare reference and nice result
5) Reference result:
the sum of each row is :
                130     140     240
the sum of each col is :
                60      160     290
6) Nice result:
the sum of each row is :
                130     140     240
the sum of each col is :
                60      160     290
7) Compare reference vs nice: PASS
8. Performance summary
        normal:
            instret: 502, cycle: 502
        nice  :
            instret: 177, cycle: 177
```

demo_vnice

Note:

- It only works with Nuclei EvalSoC with Vector NICE demo instructions enabled.
- Need vector nice feature enabled, and Nuclei NICE hardware demo integrated such as evalsoc

This demo_vnice application[92] is used to demonstrate how to Nuclei Vector NICE feature.

NICE is short for Nuclei Instruction Co-unit Extension, which is used to support extensive customization and specialization.

How to run this application:

```
# Assume that you can set up the Tools and Nuclei SDK environment
# Use Nuclei UX900 + Vector Nice RISC-V processor as example, hardware NICE demo␣
↪integrated
# cd to the demo_dsp directory
cd application/baremetal/demo_vnice
# Clean the application first
make SOC=evalsoc clean
# Build and upload the application
make SOC=evalsoc upload
```

Expected output as below:

```
Nuclei SDK Build Time: May 28 2024, 13:31:08
Download Mode: ILM
CPU Frequency 1000000716 Hz
CPU HartID: 0
1. Set array_normal_in1 array_normal_in1 array_vnice_in1 array_vnice_in2
2. Do reference vector complex mul, store, load
3. Do vector nice complex mul, store, load
4. Compare reference and vnice result
PASS
5. Performance summary
        normal:
            instret: 22546, cycle: 22546
        vnice  :
            instret: 1010, cycle: 1010
```

coremark

This coremark benchmark application[93] is used to run EEMBC CoreMark Software.

EEMBC CoreMark Software is a product of EEMBC and is provided under the terms of the CoreMark License that is distributed with the official EEMBC COREMARK Software release. If you received this EEMBC CoreMark Software without the accompanying CoreMark License, you must discontinue use and download the official release from www.coremark.org.

In Nuclei SDK, we provided code and Makefile for this `coremark` application. You can also optimize the `COMMON_FLAGS` defined in coremark application Makefile to get different score number.

[92] https://github.com/Nuclei-Software/nuclei-sdk/tree/master/application/baremetal/demo_vnice
[93] https://github.com/Nuclei-Software/nuclei-sdk/tree/master/application/baremetal/benchmark/coremark

- By default, this application runs for 800 iterations, you can also change this in Makefile. e.g. Change this `-DITERATIONS=800` to value such as `-DITERATIONS=5000`
- macro **PERFORMANCE_RUN=1** is defined
- **STDCLIB ?= newlib_small** is added in its Makefile to enable float value print
- For different Nuclei CPU series, the benchmark options are different, currently you can pass `CPU_SERIES=900` to select benchmark options for 900 series, otherwise the benchmark options for 200/300/600/900 will be selected which is also the default value.

Note:

- Since for each SoC platforms, the CPU frequency is different, so user need to change the `ITERATIONS` defined in Makefile to proper value to let the coremark run at least 10 seconds
- For example, for the `gd32vf103` based boards supported in Nuclei SDK, we suggest to change `-DITERATIONS=800` to `-DITERATIONS=5000`

How to run this application:

```
# Assume that you can set up the Tools and Nuclei SDK environment
# cd to the coremark directory
cd application/baremetal/benchmark/coremark
# Clean the application first
make SOC=evalsoc BOARD=nuclei_fpga_eval DOWNLOAD=ilm CORE=n300fd ARCH_EXT=_zba_zbb_zbc_
↪zbs_zicond clean
# Build and upload the application
make SOC=evalsoc BOARD=nuclei_fpga_eval DOWNLOAD=ilm CORE=n300fd ARCH_EXT=_zba_zbb_zbc_
↪zbs_zicond upload
```

Expected output as below:

```
Nuclei SDK Build Time: May  6 2025, 16:33:08
Download Mode: ILM
CPU Frequency 16003563 Hz
CPU HartID: 0
Start to run coremark for 800 iterations
2K performance run parameters for coremark.
CoreMark Size    : 666
Total ticks      : 207671961
Total time (secs): 12.976789
Iterations/Sec   : 61.648533
Iterations       : 800
Compiler version : GCC14.2.1 20240816
Compiler flags   : -Ofast -fno-code-hoisting -fno-common -finline-functions -falign-
↪functions=6 -falign-jumps=6 -falign-loops=4 -finline-limit=2001
Memory location  : STACK
seedcrc          : 0xe9f5
[0]crclist       : 0xe714
[0]crcmatrix     : 0x1fd7
[0]crcstate      : 0x8e3a
[0]crcfinal      : 0xcc42
Correct operation validated. See readme.txt for run and reporting rules.
```

(continues on next page)

(continued from previous page)

```
CoreMark 1.0 : 61.648533 / GCC14.2.1 20240816 -Ofast -fno-code-hoisting -fno-common -
↪finline-functions -falign-functions=6 -falign-jumps=6 -faligns

     (Iterations is: 800
     (total_ticks is: 207671961
 (*) Assume the core running at 1 MHz
     So the CoreMark/MHz can be calculated by:
     (Iterations*1000000/total_ticks) = 3.852229 CoreMark/MHz


CSV, Benchmark, Iterations, Cycles, CoreMark/MHz
CSV, CoreMark, 800, 207671961, 3.852
IPC = Instret/Cycle = 184031355/207671961 = 0.886
```

dhrystone

This dhrystone benchmark application[94] is used to run DHRYSTONE Benchmark Software, whose version is v2.1.

The Dhrystone benchmark program has become a popular benchmark for CPU/compiler performance measurement, in particular in the area of minicomputers, workstations, PC's and microprocesors.

- It apparently satisfies a need for an easy-to-use integer benchmark;
- it gives a first performance indication which is more meaningful than MIPS numbers which, in their literal meaning (million instructions per second), cannot be used across different instruction sets (e.g. RISC vs. CISC).
- With the increasing use of the benchmark, it seems necessary to reconsider the benchmark and to check whether it can still fulfill this function.

In Nuclei SDK, we provided code and Makefile for this `dhrystone` application. You can also optimize the `COMMON_FLAGS` defined in dhrystone application Makefile to get different score number.

- **STDCLIB ?= newlib_small** is added in its Makefile to enable float value print
- You can change `Number_Of_Runs` in `dhry_1.c` to increate or decrease number of iterations

How to run this application:

```
# Assume that you can set up the Tools and Nuclei SDK environment
# cd to the dhrystone directory
cd application/baremetal/benchmark/dhrystone
# Clean the application first
make SOC=evalsoc BOARD=nuclei_fpga_eval DOWNLOAD=ilm CORE=n300fd ARCH_EXT=_zba_zbb_zbc_
↪zbs clean
# Build and upload the application
make SOC=evalsoc BOARD=nuclei_fpga_eval DOWNLOAD=ilm CORE=n300fd ARCH_EXT=_zba_zbb_zbc_
↪zbs upload
```

Expected output as below:

[94] https://github.com/Nuclei-Software/nuclei-sdk/tree/master/application/baremetal/benchmark/dhrystone

```
Nuclei SDK Build Time: May  6 2025, 16:29:34
Download Mode: ILM
CPU Frequency 16004874 Hz
CPU HartID: 0

Dhrystone Benchmark, Version 2.1 (Language: C)

Program compiled without 'register' attribute

Please give the number of runs through the benchmark:
Execution starts, 500000 runs through Dhrystone
Execution ends

Final values of the variables used in the benchmark:

Int_Glob:            5
        should be:   5
Bool_Glob:           1
        should be:   1
Ch_1_Glob:           A
        should be:   A
Ch_2_Glob:           B
        should be:   B
Arr_1_Glob[8]:       7
        should be:   7
Arr_2_Glob[8][7]:    500010
        should be:   Number_Of_Runs + 10
Ptr_Glob->
  Ptr_Comp:          -1879032528
        should be:   (implementation-dependent)
  Discr:             0
        should be:   0
  Enum_Comp:         2
        should be:   2
  Int_Comp:          17
        should be:   17
  Str_Comp:          DHRYSTONE PROGRAM, SOME STRING
        should be:   DHRYSTONE PROGRAM, SOME STRING
Next_Ptr_Glob->
  Ptr_Comp:          -1879032528
        should be:   (implementation-dependent), same as above
  Discr:             0
        should be:   0
  Enum_Comp:         1
        should be:   1
  Int_Comp:          18
        should be:   18
  Str_Comp:          DHRYSTONE PROGRAM, SOME STRING
        should be:   DHRYSTONE PROGRAM, SOME STRING
Int_1_Loc:           5
        should be:   5
Int_2_Loc:           13
        should be:   13
```

(continues on next page)

(continued from previous page)

```
Int_3_Loc:            7
        should be:    7
Enum_Loc:             1
        should be:    1
Str_1_Loc:            DHRYSTONE PROGRAM, 1'ST STRING
        should be:    DHRYSTONE PROGRAM, 1'ST STRING
Str_2_Loc:            DHRYSTONE PROGRAM, 2'ND STRING
        should be:    DHRYSTONE PROGRAM, 2'ND STRING

 (*) User_Cycle for total run through Dhrystone with loops 500000:
151000097
       So the DMIPS/MHz can be calculated by:
       1000000/(User_Cycle/Number_Of_Runs)/1757 = 1.884608 DMIPS/MHz


CSV, Benchmark, Iterations, Cycles, DMIPS/MHz
CSV, Dhrystone, 500000, 151000097, 1.884
IPC = Instret/Cycle = 145000036/151000097 = 0.960
```

dhrystone_v2.2

This dhrystone_v2.2 benchmark application[95] is used to run DHRYSTONE Benchmark Software, whose version is v2.2.

In Nuclei SDK, we provided code and Makefile for this `dhrystone` application. You can also optimize the `COMMON_FLAGS` defined in dhrystone application Makefile to get different score number.

- **STDCLIB ?= newlib_small** is added in its Makefile to enable float value print
- `Number_Of_Runs` will increase itself if running time is too small

How to run this application:

```
# Assume that you can set up the Tools and Nuclei SDK environment
# cd to the dhrystone_v2.2 directory
cd application/baremetal/benchmark/dhrystone_v2.2
# Clean the application first
make SOC=evalsoc BOARD=nuclei_fpga_eval DOWNLOAD=ilm CORE=n300fd ARCH_EXT=_zba_zbb_zbc_
↪zbs clean
# Build and upload the application
make SOC=evalsoc BOARD=nuclei_fpga_eval DOWNLOAD=ilm CORE=n300fd ARCH_EXT=_zba_zbb_zbc_
↪zbs upload
```

Expected output as below:

```
Nuclei SDK Build Time: May  6 2025, 16:22:34
Download Mode: ILM
CPU Frequency 16006184 Hz
CPU HartID: 0

Dhrystone Benchmark, Version C, Version 2.2
Program compiled without 'register' attribute
```

(continues on next page)

[95] https://github.com/Nuclei-Software/nuclei-sdk/tree/master/application/baremetal/benchmark/dhrystone_v2.2

(continued from previous page)

```
Using time(), HZ=1

Trying 50000 runs through Dhrystone:
Measured time too small to obtain meaningful results

Trying 500000 runs through Dhrystone:
Final values of the variables used in the benchmark:

Int_Glob:            5
        should be:   5
Bool_Glob:           1
        should be:   1
Ch_1_Glob:           A
        should be:   A
Ch_2_Glob:           B
        should be:   B
Arr_1_Glob[8]:       7
        should be:   7
Arr_2_Glob[8][7]:    550010
        should be:   Number_Of_Runs + 10
Ptr_Glob->
  Ptr_Comp:          -1879032368
        should be:   (implementation-dependent)
  Discr:             0
        should be:   0
  Enum_Comp:         2
        should be:   2
  Int_Comp:          17
        should be:   17
  Str_Comp:          DHRYSTONE PROGRAM, SOME STRING
        should be:   DHRYSTONE PROGRAM, SOME STRING
Next_Ptr_Glob->
  Ptr_Comp:          -1879032368
        should be:   (implementation-dependent), same as above
  Discr:             0
        should be:   0
  Enum_Comp:         1
        should be:   1
  Int_Comp:          18
        should be:   18
  Str_Comp:          DHRYSTONE PROGRAM, SOME STRING
        should be:   DHRYSTONE PROGRAM, SOME STRING
Int_1_Loc:           5
        should be:   5
Int_2_Loc:           13
        should be:   13
Int_3_Loc:           7
        should be:   7
Enum_Loc:            1
        should be:   1
Str_1_Loc:           DHRYSTONE PROGRAM, 1'ST STRING
        should be:   DHRYSTONE PROGRAM, 1'ST STRING
```

(continues on next page)

(continued from previous page)

```
Str_2_Loc:           DHRYSTONE PROGRAM, 2'ND STRING
        should be:   DHRYSTONE PROGRAM, 2'ND STRING

Microseconds for one run through Dhrystone:       14.0
Dhrystones per Second:                           71429

 (*) User_Cycle for total run through Dhrystone with loops 500000:
128000082
       So the DMIPS/MHz can be calculated by:
       1000000/(User_Cycle/Number_Of_Runs)/1757 = 2.223248 DMIPS/MHz


CSV, Benchmark, Iterations, Cycles, DMIPS/MHz
CSV, Dhrystone_v2.2, 500000, 128000082, 2.223
IPC = Instret/Cycle = 117500053/128000082 = 0.917
```

whetstone

This whetstone benchmark application[96] is used to run C/C++ Whetstone Benchmark Software (Single or Double Precision), whose version is roy@roylongbottom.org.uk, 6 November 1996.

The Fortran Whetstone programs were the first general purpose benchmarks that set industry standards of computer system performance. Whetstone programs also addressed the question of the efficiency of different programming languages, an important issue not covered by more contemporary standard benchmarks.

In Nuclei SDK, we provided code and Makefile for this `whetstone` application. You can also optimize the `COMMON_FLAGS` defined in whetstone application Makefile to get different score number.

- **STDCLIB ?= newlib_small** is added in its Makefile to enable float value print
- Extra **LDFLAGS := -lm** is added in its Makefile to include the math library

How to run this application:

```
# Assume that you can set up the Tools and Nuclei SDK environment
# cd to the whetstone directory
cd application/baremetal/benchmark/whetstone
# Clean the application first
make SOC=evalsoc BOARD=nuclei_fpga_eval DOWNLOAD=ilm CORE=n300fd clean
# Build and upload the application
make SOC=evalsoc BOARD=nuclei_fpga_eval DOWNLOAD=ilm CORE=n300fd upload
```

Expected output as below:

```
Nuclei SDK Build Time: May  6 2025, 16:31:23
Download Mode: ILM
CPU Frequency 15984885 Hz
CPU HartID: 0

########################################
Double Precision C Whetstone Benchmark Opt 3 32 Bit
Calibrate
```

(continues on next page)

[96] https://github.com/Nuclei-Software/nuclei-sdk/tree/master/application/baremetal/benchmark/whetstone

(continued from previous page)

```
       0.37 Seconds          1   Passes (x 100)

Use 8  passes (x 100)


          Double Precision C/C++ Whetstone Benchmark
Loop content                  Result              MFLOPS      MOPS   Seconds

N1 floating point -1.12441415430187974        12.486              0.012
N2 floating point -1.12239951147853168        16.874              0.064
N3 if then else    1.00000000000000000                   0.000    0.000
N4 fixed point    12.00000000000000000                 120.022    0.021
N5 sin,cos etc.    0.49907428465337039                   0.402    1.654
N6 floating point  0.99999988495142078         9.600              0.449
N7 assignments     3.00000000000000000                  71.982    0.021
N8 exp,sqrt etc.   0.75095530233199781                   0.423    0.704

MWIPS                                            27.355              2.925


MWIPS/MHz                                         1.711              2.925


CSV, Benchmark, MWIPS/MHz
CSV, Whetstone, 1.711
IPC = Instret/Cycle = 35858111/49362436 = 0.726
```

whetstone_v1.2

This whetstone_v1.2 benchmark application[97] is used to run C Converted Whetstone Single or Double Precision Benchmark Version 1.2 22 March 1998, which has different algorithm to this version whetstone benchmark application[98] (they are incomparable).

In Nuclei SDK, we provided code and Makefile for this `whetstone_v1.2` application. You can also optimize the `COMMON_FLAGS` defined in whetstone application Makefile to get different score number.

- **STDCLIB ?= newlib_small** is added in its Makefile to enable float value print
- Extra **LDFLAGS := -lm** is added in its Makefile to include the math library

How to run this application:

```
# Assume that you can set up the Tools and Nuclei SDK environment
# cd to the whetstone_v1.2 directory
cd application/baremetal/benchmark/whetstone_v1.2
# Clean the application first
make SOC=evalsoc BOARD=nuclei_fpga_eval DOWNLOAD=ilm CORE=n300fd clean
# Build and upload the application
make SOC=evalsoc BOARD=nuclei_fpga_eval DOWNLOAD=ilm CORE=n300fd upload
```

Expected output as below:

[97] https://github.com/Nuclei-Software/nuclei-sdk/tree/master/application/baremetal/benchmark/whetstone_v1.2
[98] https://github.com/Nuclei-Software/nuclei-sdk/tree/master/application/baremetal/benchmark/whetstone

```
Nuclei SDK Build Time: May  6 2025, 16:11:48
Download Mode: ILM
CPU Frequency 15984885 Hz
CPU HartID: 0

#########################################
Single Precision C Whetstone Benchmark Version 1.2      22 March 1998
Nuclei SDK Build Time: May  6 2025, 16:13:56
Download Mode: ILM
CPU Frequency 16004874 Hz
CPU HartID: 0

#########################################
Double Precision C Whetstone Benchmark Version 1.2      22 March 1998

Loops: 50000, Iterations: 1, Duration: 70 sec.
C Converted Double Precision Whetstones: 71.4 MIPS

CSV, Benchmark, MWIPS/MHz
CSV, Whetstone_v1.2, 4.462
IPC = Instret/Cycle = 704074177/1133874283 = 0.620
```

demo_smode_eclic

This demo_smode_eclic application[99] is used to demostrate how to use the ECLIC API and Interrupt in supervisor mode with TEE.

Note:

- It doesn't work with gd32vf103 processor.
- It needs Nuclei CPU configured with TEE feature and S-Mode ECLIC
- In this application's Makefile, we provided comments in Makefile about optimization for code size, please refer to chapter *demo_eclic* (page 94) for details.
- Need to enable TEE in <Device.h> if TEE present in CPU.

- The timer interrupt and timer software interrupt are used
- The timer interrupt is registered as non-vector interrupt
- The timer software interrupt is registered as vector interrupt, and we enable its preemptive feature by using `SAVE_IRQ_CSR_CONTEXT_S` and `RESTORE_IRQ_CSR_CONTEXT_S` in timer software interrupt handler
- The timer interrupt is triggered periodically
- The timer software interrupt is triggered in timer interrupt handler using `SysTimer_SetHartSWIRQ` function
- Interrupts occur in supervisor mode to which it drops from machine mode, and you can observe the difference from *demo_eclic* (page 94) by console output
- In the application code, there is a macro called `SWIRQ_INTLEVEL_HIGHER` to control the timer software interrupt working feature:

[99] https://github.com/Nuclei-Software/nuclei-sdk/tree/master/application/baremetal/demo_smode_eclic

- If **SWIRQ_INTLEVEL_HIGHER=1**, the timer software interrupt level is higher than timer interrupt level, so when timer software interrupt is triggered, then timer software interrupt will be processed immediately, and timer interrupt will be preempted by timer software interrupt.
- If **SWIRQ_INTLEVEL_HIGHER=0**, the timer software interrupt level is lower than timer interrupt level, so when timer software interrupt is triggered, then timer software interrupt will be processed after timer interrupt, and timer interrupt will not be preempted by timer software interrupt.

How to run this application:

```
# Assume that you can set up the Tools and Nuclei SDK environment
# cd to the demo_smode_eclic directory
cd application/baremetal/demo_smode_eclic
# MUST: Your CPU configuration must has TEE configured
# Since Nuclei SDK 0.7.0, if you are sure CFG_HAS_TEE is not defined in cpufeature.h,␣
↪but you have TEE
# you can pass extra make variable XLCFG_TEE=1 during make command to tell sdk
# the TEE present, it will define CFG_HAS_TEE
# Change macro SWIRQ_INTLEVEL_HIGHER value in demo_smode_eclic.c
# to see different working mode of this demo
# Clean the application first
make SOC=evalsoc BOARD=nuclei_fpga_eval DOWNLOAD=ilm CORE=n300 clean
# Build and upload the application
make SOC=evalsoc BOARD=nuclei_fpga_eval DOWNLOAD=ilm CORE=n300 upload
```

Expected output(SWIRQ_INTLEVEL_HIGHER=1) as below:

```
Nuclei SDK Build Time: Aug  5 2022, 15:05:52
Download Mode: ILM
CPU Frequency 15989145 Hz
Current sp is 0x9000ffa0, so it is in Machine Mode!
Drop to S-Mode now
[IN S-MODE ENTRY POINT] Hello Supervisor Mode!!!
Current sp is 0x90000f40, so it is in Supervisor Mode!
Initialize timer and start timer interrupt periodically
Current sp is 0x90000d80, so it is in Supervisor Mode!
--------------------
[IN S-MODE TIMER INTERRUPT]timer interrupt hit 0 times
[IN S-MODE TIMER INTERRUPT]trigger software interrupt
[IN S-MODE TIMER INTERRUPT]software interrupt will run during timer interrupt
[IN S-MODE SOFTWARE INTERRUPT]software interrupt hit 0 times
Current sp is 0x90000d10, so it is in Supervisor Mode!
[IN S-MODE SOFTWARE INTERRUPT]software interrupt end
[IN S-MODE TIMER INTERRUPT]timer interrupt end
Current sp is 0x90000d80, so it is in Supervisor Mode!
--------------------
[IN S-MODE TIMER INTERRUPT]timer interrupt hit 1 times
[IN S-MODE TIMER INTERRUPT]trigger software interrupt
[IN S-MODE TIMER INTERRUPT]software interrupt will run during timer interrupt
[IN S-MODE SOFTWARE INTERRUPT]software interrupt hit 1 times
Current sp is 0x90000d10, so it is in Supervisor Mode!
[IN S-MODE SOFTWARE INTERRUPT]software interrupt end
[IN S-MODE TIMER INTERRUPT]timer interrupt end
Current sp is 0x90000d80, so it is in Supervisor Mode!
--------------------
```

(continues on next page)

(continued from previous page)

```
[IN S-MODE TIMER INTERRUPT]timer interrupt hit 2 times
[IN S-MODE TIMER INTERRUPT]trigger software interrupt
[IN S-MODE TIMER INTERRUPT]software interrupt will run during timer interrupt
[IN S-MODE SOFTWARE INTERRUPT]software interrupt hit 2 times
Current sp is 0x90000d10, so it is in Supervisor Mode!
[IN S-MODE SOFTWARE INTERRUPT]software interrupt end
[IN S-MODE TIMER INTERRUPT]timer interrupt end
Current sp is 0x90000d80, so it is in Supervisor Mode!
------------------
[IN S-MODE TIMER INTERRUPT]timer interrupt hit 3 times
[IN S-MODE TIMER INTERRUPT]trigger software interrupt
[IN S-MODE TIMER INTERRUPT]software interrupt will run during timer interrupt
[IN S-MODE SOFTWARE INTERRUPT]software interrupt hit 3 times
Current sp is 0x90000d10, so it is in Supervisor Mode!
[IN S-MODE SOFTWARE INTERRUPT]software interrupt end
[IN S-MODE TIMER INTERRUPT]timer interrupt end
```

Expected output(SWIRQ_INTLEVEL_HIGHER=0) as below:

```
Nuclei SDK Build Time: Aug  5 2022, 15:09:46
Download Mode: ILM
CPU Frequency 15989145 Hz
Current sp is 0x9000ffa0, so it is in Machine Mode!
Drop to S-Mode now
[IN S-MODE ENTRY POINT] Hello Supervisor Mode!!!
Current sp is 0x90000f50, so it is in Supervisor Mode!
Initialize timer and start timer interrupt periodically
Current sp is 0x90000d90, so it is in Supervisor Mode!
------------------
[IN S-MODE TIMER INTERRUPT]timer interrupt hit 0 times
[IN S-MODE TIMER INTERRUPT]trigger software interrupt
[IN S-MODE TIMER INTERRUPT]software interrupt will run when timer interrupt finished
[IN S-MODE TIMER INTERRUPT]timer interrupt end
[IN S-MODE SOFTWARE INTERRUPT]software interrupt hit 0 times
Current sp is 0x90000ee0, so it is in Supervisor Mode!
[IN S-MODE SOFTWARE INTERRUPT]software interrupt end
Current sp is 0x90000d90, so it is in Supervisor Mode!
------------------
[IN S-MODE TIMER INTERRUPT]timer interrupt hit 1 times
[IN S-MODE TIMER INTERRUPT]trigger software interrupt
[IN S-MODE TIMER INTERRUPT]software interrupt will run when timer interrupt finished
[IN S-MODE TIMER INTERRUPT]timer interrupt end
[IN S-MODE SOFTWARE INTERRUPT]software interrupt hit 1 times
Current sp is 0x90000ee0, so it is in Supervisor Mode!
[IN S-MODE SOFTWARE INTERRUPT]software interrupt end
Current sp is 0x90000d90, so it is in Supervisor Mode!
------------------
[IN S-MODE TIMER INTERRUPT]timer interrupt hit 2 times
[IN S-MODE TIMER INTERRUPT]trigger software interrupt
[IN S-MODE TIMER INTERRUPT]software interrupt will run when timer interrupt finished
[IN S-MODE TIMER INTERRUPT]timer interrupt end
[IN S-MODE SOFTWARE INTERRUPT]software interrupt hit 2 times
```

(continues on next page)

(continued from previous page)

```
Current sp is 0x90000ee0, so it is in Supervisor Mode!
[IN S-MODE SOFTWARE INTERRUPT]software interrupt end
Current sp is 0x90000d90, so it is in Supervisor Mode!
--------------------
[IN S-MODE TIMER INTERRUPT]timer interrupt hit 3 times
[IN S-MODE TIMER INTERRUPT]trigger software interrupt
[IN S-MODE TIMER INTERRUPT]software interrupt will run when timer interrupt finished
[IN S-MODE TIMER INTERRUPT]timer interrupt end
[IN S-MODE SOFTWARE INTERRUPT]software interrupt hit 3 times
Current sp is 0x90000ee0, so it is in Supervisor Mode!
[IN S-MODE SOFTWARE INTERRUPT]software interrupt end
```

demo_smode_plic

This demo_smode_plic application[100] is a bare metal program demonstrating the PLIC (Platform Level Interrupt Controller) functionality in RISC-V processor's S-Mode (Supervisor Mode). The program shows how to switch from M-Mode to S-Mode and handle UART interrupts in S-Mode.

Note:

- Ensure hardware supports required processor features
- It needs Nuclei CPU configured with PLIC, S-Mode and PMP
- Proper definitions in <Device>.h
- Need to enable PLIC in <Device.h> if PLIC present in CPU
  - `__PMP_PRESENT=1` and `__PLIC_PRESENT=1`

This demo will demostrate the following features:

- Demonstrates M-Mode to S-Mode transition
- Configures PMP to allow S-Mode access to all address spaces
- Registers and handles UART interrupts in S-Mode
- Supports UART receive interrupt handling

How to run this application:

```
# Assume that you can set up the Tools and Nuclei SDK environment
# cd to the demo_smode_plic directory
cd application/baremetal/demo_smode_plic
# MUST: Your CPU configuration must has PLIC/PMP/SMode configured
# Since Nuclei SDK 0.7.0, if you are sure CFG_HAS_PLIC is not defined in cpufeature.h,␣
↪but you have PLIC
# you can pass extra make variable XLCFG_PLIC=1 during make command to tell sdk
# the PLIC present, it will define CFG_HAS_PLIC
# Clean the application first
make SOC=evalsoc CORE=n900 clean
# Build and upload the application
make SOC=evalsoc CORE=n900 upload
```

[100] https://github.com/Nuclei-Software/nuclei-sdk/tree/master/application/baremetal/demo_smode_plic

```
Nuclei SDK Build Time: Apr 28 2025, 15:06:30
Download Mode: ILM
CPU Frequency 50002329 Hz
CPU HartID: 0
Current sp is 0x9000ff80, so it is in Machine Mode!
Drop to S-Mode now
[IN S-MODE ENTRY POINT] Hello Supervisor Mode!!!
Current sp is 0x900010c0, so it is in Supervisor Mode!
You can press any key now to trigger uart receive interrupt
Enter uart0 interrupt, you just typed: 1
Enter uart0 interrupt, you just typed: 2
```

demo_sstc

This demo_sstc application[101] is used to demostrate how to use the ECLIC API and Interrupt in supervisor mode with TEE and SSTC.

This demo is similar with *demo_smode_eclic* (page 110)

Note:

- It doesn't work with gd32vf103 processor.
- It needs Nuclei CPU configured with TEE feature and S-Mode ECLIC and SSTC feature

---

How to run this application:

```
# Assume that you can set up the Tools and Nuclei SDK environment
# cd to the demo_sstc directory
cd application/baremetal/demo_sstc
# MUST: Your CPU configuration must has TEE and SSTC configured
# Assume you are using n300
# Clean the application first
make SOC=evalsoc CORE=n300 clean
# Build and upload the application
make SOC=evalsoc CORE=n300 upload
```

```
Nuclei SDK Build Time: Feb 21 2025, 11:12:45
Download Mode: ILM
CPU Frequency 15987179 Hz
CPU HartID: 0
Current sp is 0x9000ff70, so it is in Machine Mode!
Drop to S-Mode now
[IN S-MODE ENTRY POINT] Hello Supervisor Mode!!!
Current sp is 0x90001040, so it is in Supervisor Mode!
Initialize timer and start timer interrupt periodically
Current sp is 0x90000f50, so it is in Supervisor Mode!
-------------------
[IN S-MODE TIMER INTERRUPT]timer interrupt hit 0 times
[IN S-MODE TIMER INTERRUPT]trigger software interrupt
```

(continues on next page)

[101] https://github.com/Nuclei-Software/nuclei-sdk/tree/master/application/baremetal/demo_sstc

(continued from previous page)

```
[IN S-MODE TIMER INTERRUPT]software interrupt will run during timer interrupt
[IN S-MODE SOFTWARE INTERRUPT]software interrupt hit 0 times
Current sp is 0x90000e10, so it is in Supervisor Mode!
[IN S-MODE SOFTWARE INTERRUPT]software interrupt end
[IN S-MODE TIMER INTERRUPT]timer interrupt end
Current sp is 0x90000f50, so it is in Supervisor Mode!
-------------------
[IN S-MODE TIMER INTERRUPT]timer interrupt hit 1 times
[IN S-MODE TIMER INTERRUPT]trigger software interrupt
[IN S-MODE TIMER INTERRUPT]software interrupt will run during timer interrupt
[IN S-MODE SOFTWARE INTERRUPT]software interrupt hit 1 times
Current sp is 0x90000e10, so it is in Supervisor Mode!
[IN S-MODE SOFTWARE INTERRUPT]software interrupt end
[IN S-MODE TIMER INTERRUPT]timer interrupt end
Current sp is 0x90000f50, so it is in Supervisor Mode!
```

demo_spmp

This demo_spmp application[102] is removed from `0.8.0` release since the `sPMP` hardware feature is upgraded to `SMPU` in nowadays Nuclei RISC-V CPU, please refer to *demo_smpu* (page 115).

demo_smpu

`SMPU` is upgraded from `sPMP` to enable S-mode OS to limit the physical addresses accessible by U-mode software on a hart. This demo_smpu application[103] is used to demonstrate how to grant physical memory privileges(read, write, execute) on each physical memory region by supervisor-mode control CSRs.

Note:

- It doesn't work with gd32vf103 processor.
- It needs Nuclei CPU configured with TEE, PMP, SMPU feature
- Need to enable PMP in <Device.h> if PMP present in CPU.
- Need to enable TEE in <Device.h> if TEE present in CPU.
- Need to enable SMPU in <Device.h> if smpu present in CPU.

- The demo_smpu application[104] has many common design with demo_spmp application[105], and you should first pay attention to Encoding of Permissions and Context Switching Optimization when changed to smpu
- Unlike sPMP, `__set_SMPUSWITCHx` should be called to activate the entries
- `smpu_violation_fault_handler` is registered, which is to execute when smpu violation exception occurs
- The SMPU is checked before the PMA checks and PMP checks
- There're three config structures, `pmp_config` inits that M-mode grants full permission of the whole address range on S and U mode; `smpu_config_x` sets protected executable address range as 2^12 bytes; `smpu_config_rw`

[102] https://github.com/Nuclei-Software/nuclei-sdk/tree/master/application/baremetal/demo_spmp
[103] https://github.com/Nuclei-Software/nuclei-sdk/tree/master/application/baremetal/demo_smpu
[104] https://github.com/Nuclei-Software/nuclei-sdk/tree/master/application/baremetal/demo_smpu
[105] https://github.com/Nuclei-Software/nuclei-sdk/tree/master/application/baremetal/demo_spmp

sets protected data range as 2^12 bytes, and you can change the `protection`, `order`, `base_addr` according to your memory assignments

- SMPU has three kinds of rules: U-mode-only, S-mode-only, and Shared-Region rules. The S bit marks a rule as S-mode-only when set and U-mode-only when unset
- `protection` of smpu_config_x and smpu_config_rw should be assigned according to `2.4. Encoding of Permissions` of Ssmpu spec
- Exception delegation from default M mode to S mode is also provided in this demo, when it violates smpu check. When exception occurs, the print info including `scause`, `sepc` can be observed by serial console, which explains the exception cause of smpu permission violation, and shows which asm instruction triggers the violation
- In the application code, there is a macro called `TRIGGER_SMPU_VIOLATION_MODE` to control the smpu working feature:
  - If **TRIGGER_SMPU_VIOLATION_MODE=INSTRUCTION_SMPU_EXCEPTION**, the unallowed memory is to execute, which triggers `Instruction SMPU fault`, whose scause.EXCCODE = 12
  - If **TRIGGER_SMPU_VIOLATION_MODE=LOAD_SMPU_EXCEPTION**, the unallowed memory is to read, which triggers `Load SMPU fault`, whose scause.EXCODE = 13
  - If **TRIGGER_SMPU_VIOLATION_MODE=STORE_SMPU_EXCEPTION**, the unallowed memory is to write, which triggers `Store/AMO SMPU fault`, whose scause.EXCODE = 15
  - If **TRIGGER_SMPU_VIOLATION_MODE=EXECUTE_SHARED_DATA_REGION_EXCEPTION**, the shared R/W data region is to execute, which triggers `Instruction SMPU fault`
  - If **TRIGGER_SMPU_VIOLATION_MODE=WRITE_READONLY_SHARED_DATA_EXCEPTION**, the shared Read-only data region is to write, which triggers `Store/AMO SMPU fault`
  - If **TRIGGER_SMPU_VIOLATION_MODE=SHARE_CODE_DATA_REGION**, the shared code region is to execute, and the shared R/W data region is to read and write, both of which is allowed
  - If **TRIGGER_SMPU_VIOLATION_MODE=RUN_WITH_ENTRY_INACTIVE**, the code region and data reigon is set to inaccessible, but disable corresponpding entries, so the rules doesn't take effect and execution and read/write succeed

How to run this application:

Expected output(TRIGGER_SMPU_VIOLATION_MODE=INSTRUCTION_SMPU_EXCEPTION) as below:

```
Nuclei SDK Build Time: Jun 18 2024, 18:36:40
Download Mode: ILM
CPU Frequency 16058613 Hz
CPU HartID: 0
------smpu demo with trigger condition 0------
Get pmp entry: index 0, prot_out: 0x9f, addr_out: 0x0, order_out: 32
Get smpu entry: index 0, prot_out: 0x9b, addr_out: 0x80004000, order_out: 12
Get smpu entry: index 1, prot_out: 0x9b, addr_out: 0x90000000, order_out: 12
Attempting to fetch instruction from protected address 0x0x80004000
Instruction SMPU fault occurs, cause: 0x1000000c, epc: 0x80004000
```

Expected output(TRIGGER_SMPU_VIOLATION_MODE=LOAD_SMPU_EXCEPTION) as below:

```
Nuclei SDK Build Time: Jun 18 2024, 18:39:13
Download Mode: ILM
CPU Frequency 16068116 Hz
CPU HartID: 0
------smpu demo with trigger condition 1------
```

(continues on next page)

(continued from previous page)

```
Get pmp entry: index 0, prot_out: 0x9f, addr_out: 0x0, order_out: 32
Get smpu entry: index 0, prot_out: 0x9c, addr_out: 0x80004000, order_out: 12
Get smpu entry: index 1, prot_out: 0x9c, addr_out: 0x90000000, order_out: 12
Attempting to fetch instruction from protected address 0x0x80004000
----protected_execute succeed!----
Attempting to read protected_data[0] at 0x90000000
Load SMPU fault occurs, cause: 0x1000000d, epc: 0x8000608c
```

Expected output(TRIGGER_SMPU_VIOLATION_MODE=STORE_SMPU_EXCEPTION) as below:

```
Nuclei SDK Build Time: Jun 18 2024, 18:40:00
Download Mode: ILM
CPU Frequency 16057630 Hz
CPU HartID: 0
------smpu demo with trigger condition 2------
Get pmp entry: index 0, prot_out: 0x9f, addr_out: 0x0, order_out: 32
Get smpu entry: index 0, prot_out: 0x9c, addr_out: 0x80004000, order_out: 12
Get smpu entry: index 1, prot_out: 0x99, addr_out: 0x90000000, order_out: 12
Attempting to fetch instruction from protected address 0x0x80004000
----protected_execute succeed!----
Attempting to read protected_data[0] at 0x90000000
protected_data[0]: 0xAA succeed
Attempting to write protected_data[0] at 0x90000000
Store/AMO SMPU fault occurs, cause: 0x1000000f, epc: 0x800060b2
```

Expected output(TRIGGER_SMPU_VIOLATION_MODE=EXECUTE_SHARED_DATA_REGION_EXCEPTION) as below:

```
Nuclei SDK Build Time: Jun 18 2024, 18:40:39
Download Mode: ILM
CPU Frequency 16057630 Hz
CPU HartID: 0
------smpu demo with trigger condition 3------
Get pmp entry: index 0, prot_out: 0x9f, addr_out: 0x0, order_out: 32
Get smpu entry: index 0, prot_out: 0x1e, addr_out: 0x80004000, order_out: 12
Get smpu entry: index 1, prot_out: 0x1e, addr_out: 0x90000000, order_out: 12
Attempting to fetch instruction from protected address 0x0x80004000
Instruction SMPU fault occurs, cause: 0x1000000c, epc: 0x80004000
```

Expected output(TRIGGER_SMPU_VIOLATION_MODE=WRITE_READONLY_SHARED_DATA_EXCEPTION) as below:

```
Nuclei SDK Build Time: Jun 18 2024, 18:41:17
Download Mode: ILM
CPU Frequency 16057630 Hz
CPU HartID: 0
------smpu demo with trigger condition 4------
Get pmp entry: index 0, prot_out: 0x9f, addr_out: 0x0, order_out: 32
Get smpu entry: index 0, prot_out: 0x9a, addr_out: 0x80004000, order_out: 12
Get smpu entry: index 1, prot_out: 0x9f, addr_out: 0x90000000, order_out: 12
Attempting to fetch instruction from protected address 0x0x80004000
----protected_execute succeed!----
Attempting to read protected_data[0] at 0x90000000
```

(continues on next page)

(continued from previous page)

```
protected_data[0]: 0xAA succeed
Attempting to write protected_data[0] at 0x90000000
Store/AMO SMPU fault occurs, cause: 0x1000000f, epc: 0x800060b2
```

Expected output(TRIGGER_SMPU_VIOLATION_MODE=SHARE_CODE_DATA_REGION) as below:

```
Nuclei SDK Build Time: Jun 18 2024, 18:41:46
Download Mode: ILM
CPU Frequency 16068116 Hz
CPU HartID: 0
------smpu demo with trigger condition 5------
Get pmp entry: index 0, prot_out: 0x9f, addr_out: 0x0, order_out: 32
Get smpu entry: index 0, prot_out: 0x9a, addr_out: 0x80004000, order_out: 12
Get smpu entry: index 1, prot_out: 0x1e, addr_out: 0x90000000, order_out: 12
Attempting to fetch instruction from protected address 0x0x80004000
----protected_execute succeed!----
Attempting to read protected_data[0] at 0x90000000
protected_data[0]: 0xAA succeed
Attempting to write protected_data[0] at 0x90000000
Won't run here if violates rules check!
```

(Default)Expected output(TRIGGER_SMPU_VIOLATION_MODE=RUN_WITH_ENTRY_INACTIVE) as below:

```
Nuclei SDK Build Time: Jun 18 2024, 18:42:19
Download Mode: ILM
CPU Frequency 16057630 Hz
CPU HartID: 0
------smpu demo with trigger condition 6------
Get pmp entry: index 0, prot_out: 0x9f, addr_out: 0x0, order_out: 32
Get smpu entry: index 0, prot_out: 0x18, addr_out: 0x80004000, order_out: 12
Get smpu entry: index 1, prot_out: 0x18, addr_out: 0x90000000, order_out: 12
Attempting to fetch instruction from protected address 0x0x80004000
----protected_execute succeed!----
Attempting to read protected_data[0] at 0x90000000
protected_data[0]: 0xAA succeed
Attempting to write protected_data[0] at 0x90000000
Won't run here if violates rules check!
```

demo_profiling

This demo_profiling application[106] is used to demonstrate how to use gprof or gcov in Nuclei Studio.

This application itself is modified based on an opensource aes application, we add gprof and gcov collection code to `main.c`, it will dump gprof and gcov data in console when main part code is executed.

Note:

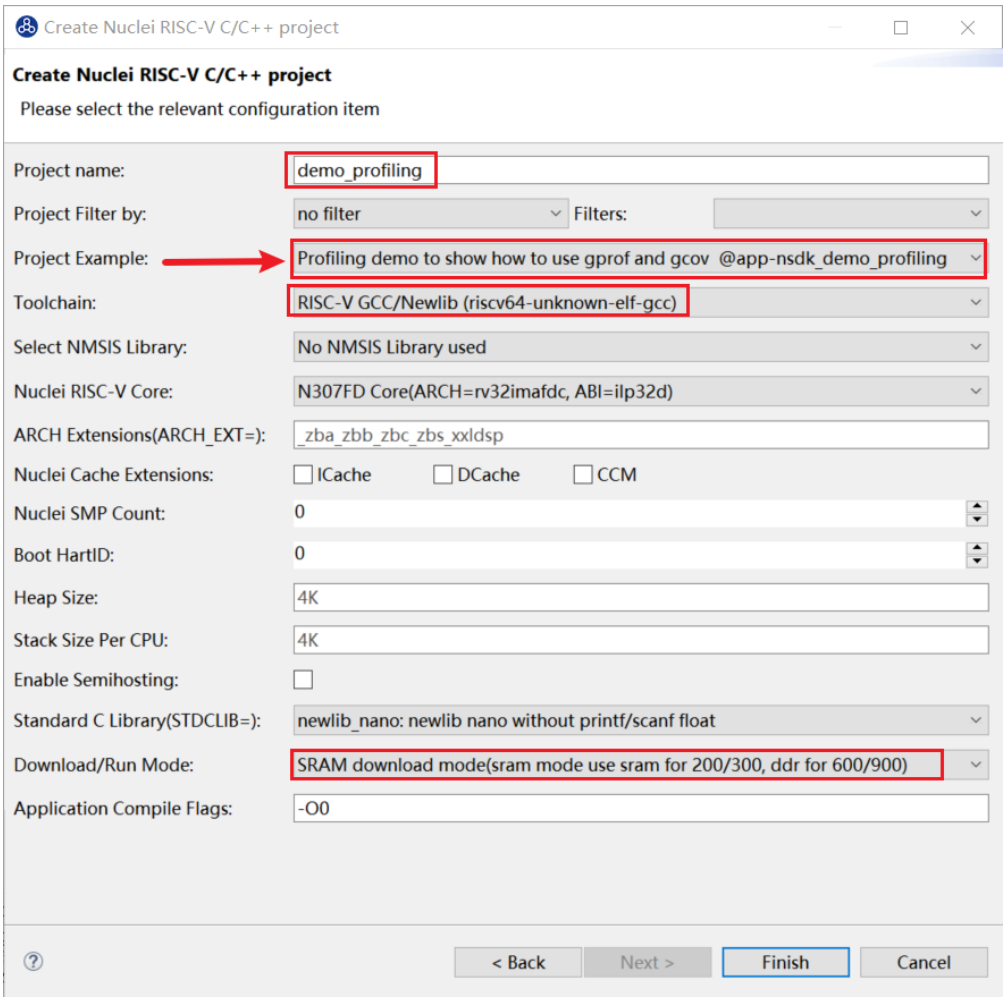
- Introduced in Nuclei SDK 0.5.1, worked with Nuclei Studio >= 2024.02

[106] https://github.com/Nuclei-Software/nuclei-sdk/tree/master/application/baremetal/demo_profiling

- Using gprof or gcov introduces instrument code into the original program, necessitating additional memory to store the collected data. This results in a slight increase in the program's memory footprint compared to its uninstrumented counterpart.
- It cannot be directly used in command line mode, you should use it in Nuclei Studio.
- Please check `README.md` about gcov and gprof support in https://github.com/Nuclei-Software/nuclei-sdk/tree/master/Components/profiling

---

Import or download Nuclei SDK 0.5.1 or later release NPK in Nuclei Studio, and then create a project called `demo_profiling` based on `app-nsdk_demo_profiling` using `Create Nuclei RISC-V C/C++ Project` Wizard as below:

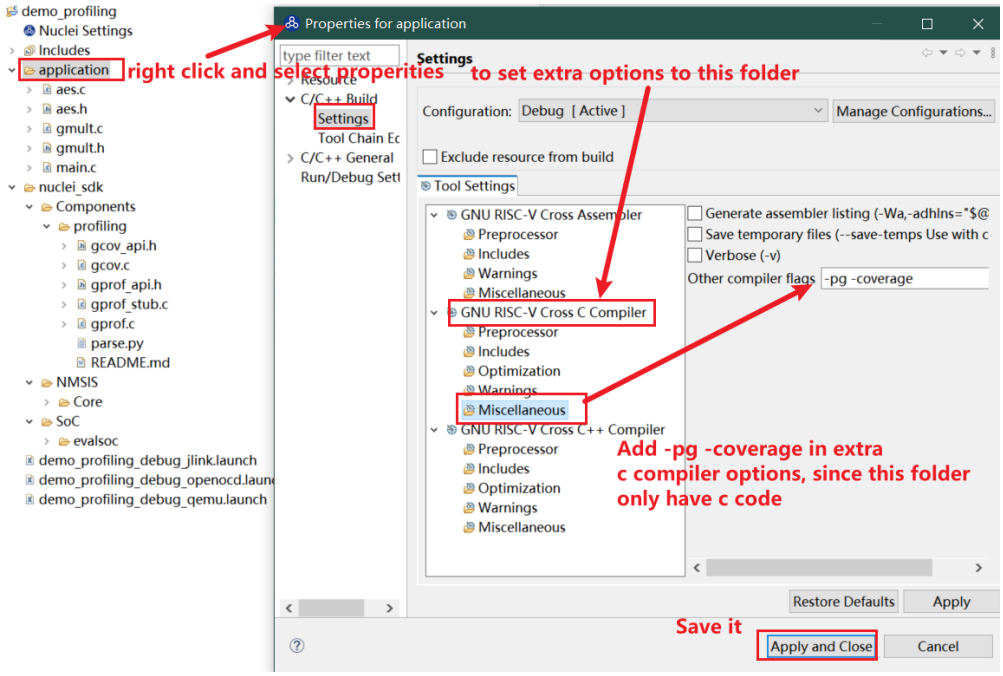


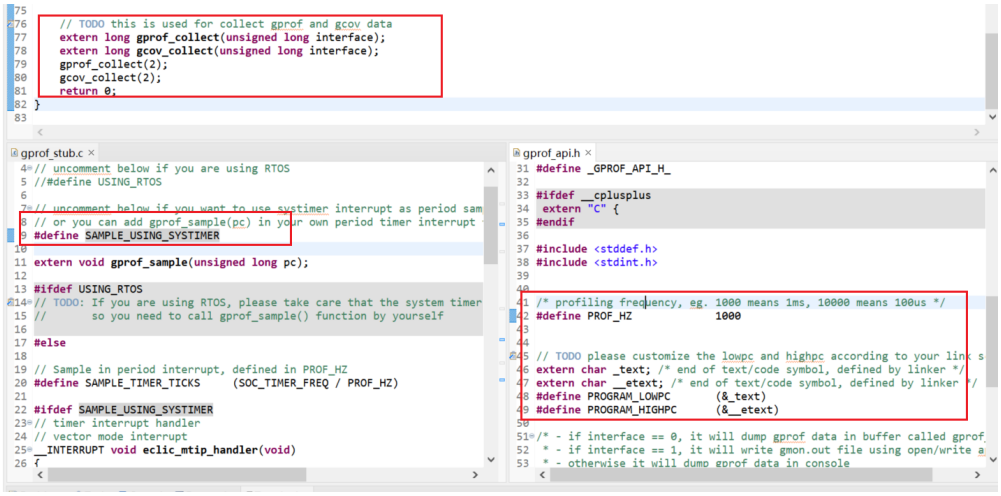
And when example is created, assume you want to profiling the `application` folder, since it is the core algorithm of this example, then you just need to do the following steps:

- Right click on the `application` folder, and click `Properities`, and add extra options in `C/C++ Build -> Settings -> GNU RISC-V Cross C Compiler -> Miscellaneous -> Other compiler flags`. - If you want to do gprof, you need to add `-pg` option. - If you want to do gcov, you need to add `-coverage` option.
- Open `main.c`, and find `TODO` item, and comment `gprof_collect(2);` or `gcov_collect(2);` based on gprof or gcov you want to collect.
- If you want to collect gprof data, you also need to modify `nuclei_sdk/Components/profiling/`

`gprof_stub.c`, if you code already has a 1ms period timer interrupt, you should copy code in `eclic_mtip_handler` to do executing sampling, otherwise you can uncomment `#define SAMPLE_USING_SYSTIMER`

Here I want to collect both gprof and gcov, so I modify it like below:





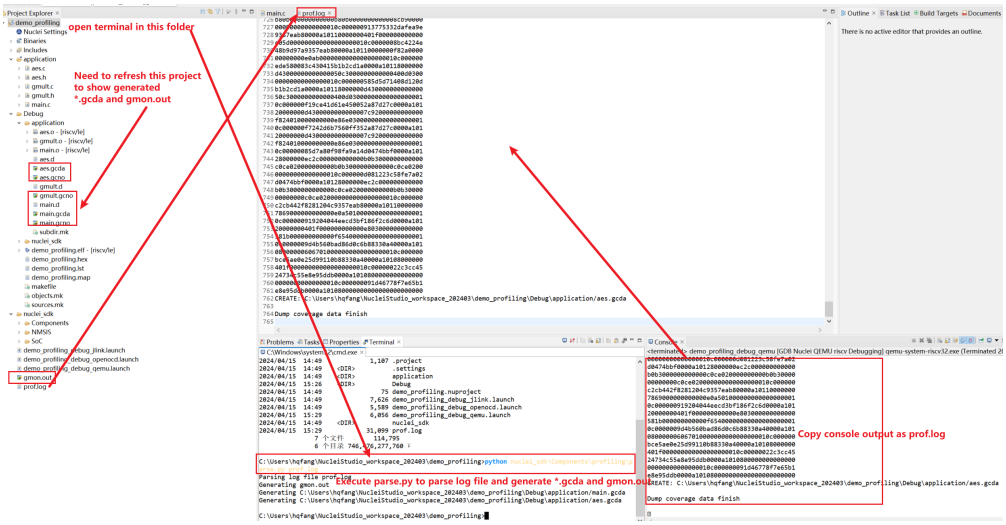
And then compile this example code, and run it using hardware or qemu, qemu is just function model, so it didn't provide correct timing information.

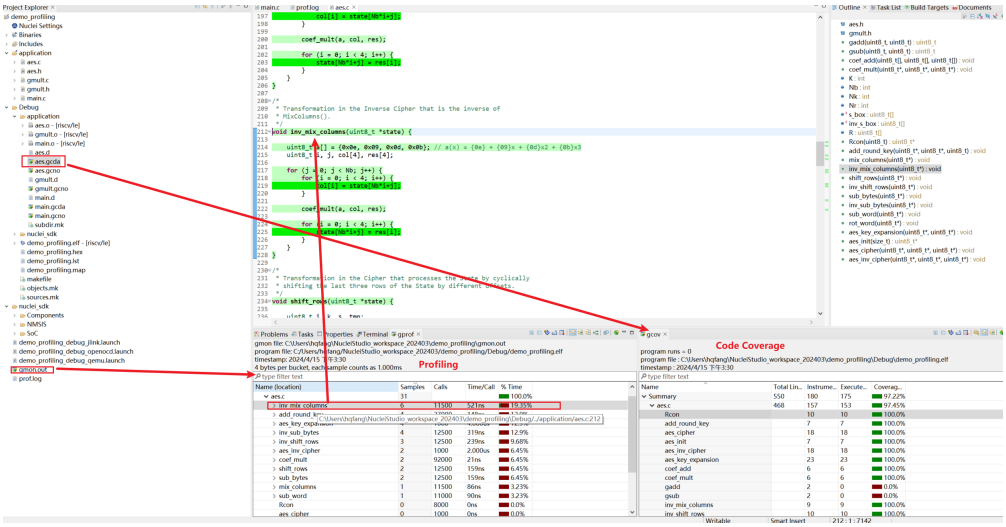
When program runs, it will dump gprof and gcov data in console, and you can copy all the output as a file called `prof.log`, and use `gprof_parse.py` to parse the data, and generate a gcov and gprof binary files.

Then you can double click `gmon.out` and `aes.gcda` to check the gprof and gcov view in Nuclei Studio like below:

About GProf view, please click https://help.eclipse.org/latest/topic/org.eclipse.linuxtools.gprof.docs/Linux_Tools_Project/GProf/User_Guide/GProf-View.html to learn more.

About Gcov view, please click https://help.eclipse.org/latest/topic/org.eclipse.linuxtools.gcov.docs/Linux_Tools_


open terminal in this folder
Need to refresh this project to show generated *.gcda and gmon.out
Copy console output as prof.log
Execute parse.py to parse log file and generate *.gcda and gmon.out


Profiling
Code Coverage

Project/GCov/User_Guide/Gcov-main-view.html to learn more.

demo_pmp

This demo_pmp application[107] is used to demonstrate how to grant physical memory privileges (read, write, execute) on each physical memory region by machine mode control CSRs.

Note:

- It doesn't work with gd32vf103 processor.
- It needs Nuclei CPU configured with PMP feature
- Need to enable PMP in <Device.h> if PMP present in CPU.

- `pmp_violation_fault_handler` is registered, which is to execute when pmp violation exception occurs
- There're two config structures, `pmp_config_x` sets protected executable address range as 2^12 bytes; `pmp_config_rw` sets protected readable/writable address range as 2^12 bytes, and you can change the `protection`, `order`, `base_addr` according to your memory assignments
- When exception occurs, the print info including `mcause`, `mepc` can be observed by serial console, which explains the exception cause of PMP permission violation, and shows which asm instruction triggers the violation
- In the application code, there is a macro called `TRIGGER_PMP_VIOLATION_MODE` to control the PMP working feature:
  - If **TRIGGER_PMP_VIOLATION_MODE=INSTRUCTION_FETCH_EXCEPTION**, the unallowed memory is to execute, which triggers `Instruction access fault`, whose mcause.EXCCODE = 1 and mdcause = 1
  - If **TRIGGER_PMP_VIOLATION_MODE=LOAD_EXCEPTION**, the unallowed memory is to read, which triggers `Load access fault`, whose mcause.EXCODE = 5 and mdcause = 1
  - If **TRIGGER_PMP_VIOLATION_MODE=STORE_EXCEPTION**, the unallowed memory is to write, which triggers `Store/AMO access fault`, whose mcause.EXCODE = 7 and mdcause = 1
  - If **TRIGGER_PMP_VIOLATION_MODE=RUN_WITH_NO_PMP_CHECK**, no violation occurs

How to run this application:

Expected output(TRIGGER_PMP_VIOLATION_MODE=INSTRUCTION_FETCH_EXCEPTION) as below:

```
Nuclei SDK Build Time: Aug 15 2022, 15:45:57
Download Mode: ILM
CPU Frequency 16006184 Hz
------PMP demo with trigger condition 0------
Get pmp entry: index 0, prot_out: 0x9b, addr_out: 0x80004000, order_out: 12
Get pmp entry: index 1, prot_out: 0x9b, addr_out: 0x90000000, order_out: 12
Attempting to fetch instruction from protected address
Instruction access fault occurs, cause: 0x30000001, epc: 0x80004000
```

From disassembly code, MEPC refers to

```
80004000:   90002537                lui     a0,0x90002
```

Expected output(TRIGGER_PMP_VIOLATION_MODE=LOAD_EXCEPTION) as below:

[107] https://github.com/Nuclei-Software/nuclei-sdk/tree/master/application/baremetal/demo_pmp

```
Nuclei SDK Build Time: Aug 15 2022, 15:45:57
Download Mode: ILM
CPU Frequency 16006184 Hz
------PMP demo with trigger condition 1------
Get pmp entry: index 0, prot_out: 0x9f, addr_out: 0x80004000, order_out: 12
Get pmp entry: index 1, prot_out: 0x9a, addr_out: 0x90000000, order_out: 12
Attempting to fetch instruction from protected address
----protected_execute succeed!----
Attempting to read protected_data[0]
Load access fault occurs, cause: 0x30000005, epc: 0x80004022
```

From disassembly code, MEPC refers to

```
80004022:   00044583                lbu     a1,0(s0) # 90000000 <_sp+0xffff0000>
```

Expected output(TRIGGER_PMP_VIOLATION_MODE=STORE_EXCEPTION) as below:

```
Nuclei SDK Build Time: Aug 15 2022, 15:45:57
Download Mode: ILM
CPU Frequency 15998320 Hz
------PMP demo with trigger condition 2------
Get pmp entry: index 0, prot_out: 0x9f, addr_out: 0x80004000, order_out: 12
Get pmp entry: index 1, prot_out: 0x99, addr_out: 0x90000000, order_out: 12
Attempting to fetch instruction from protected address
----protected_execute succeed!----
Attempting to read protected_data[0]
protected_data[0]: 0xAA succeed
Attempting to write protected_data[0]
Store/AMO access fault occurs, cause: 0x30000007, epc: 0x80004044
```

From disassembly code, MEPC refers to

```
80004044:   00f40023                sb      a5,0(s0)
```

(Default)Expected output(TRIGGER_PMP_VIOLATION_MODE=RUN_WITH_NO_PMP_CHECK) as below:

```
Nuclei SDK Build Time: Aug 15 2022, 15:45:57
Download Mode: ILM
CPU Frequency 16006184 Hz
------PMP demo with trigger condition 3------
Get pmp entry: index 0, prot_out: 0x1f, addr_out: 0x80004000, order_out: 12
Get pmp entry: index 1, prot_out: 0x1b, addr_out: 0x90000000, order_out: 12
Attempting to fetch instruction from protected address
----protected_execute succeed!----
Attempting to read protected_data[0]
protected_data[0]: 0xAA succeed
Attempting to write protected_data[0]
Won't run here if violates L R\W\X permission check!
```

demo_cidu

This demo_cidu application[108] is used to demonstrate External Interrupt Distribution (external interrupt broadcast/first come first claim), Inter Core interrupt and Semaphore of Cluster Interrupt Distribution Unit (CIDU).

This demo requests the SMP cores share the same RAM and ROM, for example, in current evalsoc/demosoc system, ilm/dlm are private resource for cpu, only the DDR/SRAM memory are shared resource for all the cpu.

Note:

- It doesn't work with gd32vf103 processor.
- It needs Nuclei SMP CPU configured with CIDU feature
- It needs Nuclei EvalSoC's uart and its interrupt, if you want to port it, you need to port uart driver of your SoC
- Need to enable CIDU in <Device.h> if CIDU present in cluster.
- Multicore SoC is needed.

- `UART0` receive is used as external interrupt, registered as `eclic_uart0_int_handler`, which is the best choice for evalsoc/demosoc and is easy to trigger by writing the serial terminal
- `UART0` receive interrupt can be broadcast to all the cores or some, and also first coming first claim mode will ensure only the first responding core handle the interrupt service routine(ISR)
- Inter core interrupt shows likes this: core3 sends interrupt to core2, core2 sends interrupt to core1, core1 sends interrupt to core0, and core0 sends interrupt to core3, registered as `eclic_inter_core_int_handler`, supposing the SoC is four cores, and etc.
- To demonstrate it will handle properly if multiple cores send interrupt to one core simultaneously, besides core2, core0 also sends interrupt to core1, supposing the SoC is four core
- To protect `UART0` resource when multicores want to access it(call `printf`), semaphore is configured, which needs to be acquired successfully before accessing `UART0`, and release it after job done
- `ENABLE_FIRST_COME_FIRST_CLAIM_MODE` is defined by default, you can comment it to just use broadcast mode

How to run this application:

```
# Assume that you can set up the Tools and Nuclei SDK environment
# Use Nuclei UX900 SMP 2/4/8(4/8 is better) Core RISC-V processor as example
# application needs to run in ddr memory not in ilm memory
# cd to the demo_cidu directory
cd application/baremetal/demo_cidu
# Since Nuclei SDK 0.7.0, if you are sure CFG_HAS_IDU is not defined in cpufeature.h,␣
↪but you have CIDU
# you can pass extra make variable XLCFG_CIDU=1 during make command to tell sdk
# the cidu present, it will define CFG_HAS_IDU
# Clean the application first
make SOC=evalsoc BOARD=nuclei_fpga_eval SMP=4 CORE=ux900 clean
# Build and upload the application
make SOC=evalsoc BOARD=nuclei_fpga_eval SMP=4 CORE=ux900 upload
```

Expected output(inter core interrupt) as below:

[108] https://github.com/Nuclei-Software/nuclei-sdk/tree/master/application/baremetal/demo_cidu

```
Nuclei SDK Build Time: Feb 10 2023, 18:39:17
Download Mode: SRAM
CPU Frequency 100602675 Hz
CPU HartID: 0
Core 3 has received interrupt from core 0
Core 1 has received interrupt from core 0
Core 2 has received interrupt from core 3
Core 1 has received interrupt from core 2
Core 0 has received interrupt from core 1
```

From output, each core sends interrupt in order, and core 1 has received interrupts from both core 0 and core 2.

Expected output(write anything to the serial terminal, enable first come first claim mode) as below:

```
Nuclei SDK Build Time: Feb 10 2023, 18:44:45
Download Mode: SRAM
CPU Frequency 100612833 Hz
CPU HartID: 0
Core 3 has received interrupt from core 0
Core 1 has received interrupt from core 0
Core 2 has received interrupt from core 3
Core 1 has received interrupt from core 2
Core 0 has received interrupt from core 1
Core 2 enters uart0_receive_handler
Core 1 enters uart0_receive_handler
Core 2 wants to process rx input
Core 2 processed input:d
Core 3 enters uart0_receive_handler
Core 0 enters uart0_receive_handler
Core 3 wants to process rx input
Core 3 enters uart0_receive_handler
Core 1 enters uart0_receive_handler
Core 3 wants to process rx input
Core 3 processed input:q
Core 0 enters uart0_receive_handler
Core 2 enters uart0_receive_handler
Core 0 wants to process rx input
Core 0 enters uart0_receive_handler
Core 1 enters uart0_receive_handler
Core 0 wants to process rx input
Core 0 processed input:s
Core 3 enters uart0_receive_handler
Core 2 enters uart0_receive_handler
Core 3 wants to process rx input
Core 1 enters uart0_receive_handler
Core 2 enters uart0_receive_handler
Core 0 enters uart0_receive_handler
Core 1 wants to process rx input
Core 1 processed input:g
Core 3 enters uart0_receive_handler
Core 3 wants to process rx input
```

From output, though setting interrupt broadcasted to all(all the core enters the ISR), while only one core (the first one) can claim the the interrupt(first come first claim) then process the uart0 input, others quit when find interrupt has been

claimed.

Expected output(write anything to the serial terminal, disable first come first claim mode) as below:

```
Nuclei SDK Build Time: Feb 10 2023, 18:48:47
Download Mode: SRAM
CPU Frequency 100602675 Hz
CPU HartID: 0
Core 3 has received interrupt from core 0
Core 1 has received interrupt from core 0
Core 2 has received interrupt from core 3
Core 1 has received interrupt from core 2
Core 0 has received interrupt from core 1
Core 2 enters uart0_receive_handler
Core 0 enters uart0_receive_handler
Core 2 wants to process rx input
Core 2 processed input:q
Core 0 wants to process rx input
Core 1 enters uart0_receive_handler
Core 1 wants to process rx input
Core 3 enters uart0_receive_handler
Core 3 wants to process rx input
Core 3 enters uart0_receive_handler
Core 0 enters uart0_receive_handler
Core 1 enters uart0_receive_handler
Core 2 enters uart0_receive_handler
Core 0 wants to process rx input
Core 0 processed input:w
Core 1 wants to process rx input
Core 3 wants to process rx input
Core 2 wants to process rx input
Core 2 enters uart0_receive_handler
Core 0 enters uart0_receive_handler
Core 1 enters uart0_receive_handler
Core 1 wants to process rx input
Core 1 processed input:e
Core 0 wants to process rx input
Core 2 wants to process rx input
Core 3 enters uart0_receive_handler
Core 3 wants to process rx input
Core 3 enters uart0_receive_handler
Core 1 enters uart0_receive_handler
Core 3 wants to process rx input
Core 3 processed input:r
Core 0 enters uart0_receive_handler
Core 1 wants to process rx input
Core 0 wants to process rx input
Core 2 enters uart0_receive_handler
Core 2 wants to process rx input
```

From output, all the core enters the ISR(means broadcasted), while only one core can process the uart0 input(semaphore used), when semaphore released, other core wants to handle the ISR job(means claim mode disabled), but process nothing (keyboard input has been received and rx interrupt pending cleared) because it has been processed.

demo_cache

Note:

- It doesn't work with gd32vf103 processor.
- It needs Nuclei CPU configured with CCM feature

This demo_cache application[109] is used to demonstrate how to understand cache mechanism.

This demo requests DCache, ICache and CCM(Cache Control and Maintenance), and needs to run in DDR/SRAM memory, because cache will bypass when run in ilm, data in dlm(private resource for cpu).

Note:

- Need to enable DCache, ICACHE, CCM in <Device.h> if present in CPU.

- An arrary(`ROW_SIZE` * `COL_SIZE`) called `array_test` is created to access its first element `array_test[0][0]`
- Firstly, enable and invalidate all DCache, update `array_test` by writing a consant, the cache miss happens and will update `array_test`'s mapping value in DCache, read out `array_test[0][0]`; then disable the Dcache, init array_test in the ddr memory to different constant, read out `array_test[0][0]`; after that, enable the DCache flushes DCache to ddr memory, read out `array_test[0][0]`, and compare these `array_test[0][0]` value
- Again disable the Dcache, init array_test in the ddr memory, read out `array_test[0][0]`; then enable the DCache, read out `array_test[0][0]`, and compare with the one before
- **For further understanding**, if the CPU has configured `HPM` (Hardware Performance Monitor), observe the cache miss count by recording the cache miss of updating array_test with DCache invalid, then compared to updating array_test with keeping DCache valid; also, compare the cache miss count of updating array_test row by row with column by column
- `BIG_ROW_SIZE` can be defined to make the array size `2048*64` bytes, which is big to see the cache miss gap(performance gap) between updating `array_test` row by row and column by column
- In our evalsoc/demosoc, cache line size is 64 bytes generally, so `array_test`'s `COL_SIZE` is 64 bytes for calculating the cache miss manually and easily
- When `HPM` used, because there's global variables in `HPM_START` and `HPM_END` , **these will bring 3 cache miss itself** (not considering cached)
- You can manage ICache apis like DCache, which skipped in this demo for less similar code
- Different compile optimization level such as -O2/-O0 effects cache miss

Note:

- There's `printf` hidden in `HPM_END`, if there is another HPM_END before it, the `printf` will bring some cache miss

How to run this application:

[109] https://github.com/Nuclei-Software/nuclei-sdk/tree/master/application/baremetal/demo_cache

```
# Assume that you can set up the Tools and Nuclei SDK environment
# Use Nuclei UX900 Core RISC-V processor as example
# application needs to run in ddr memory not in ilm memory
# cd to the demo_cache directory
cd application/baremetal/demo_cache
# Since Nuclei SDK 0.7.0, if you are sure CFG_HAS_IOCC is not defined in cpufeature.h,␣
↪but you have CCM
# you can pass extra make variable XLCFG_CCM=1 during make command to tell sdk
# the ccm present, it will define CFG_HAS_IOCC
# Clean the application first
make SOC=evalsoc BOARD=nuclei_fpga_eval CORE=ux900 DOWNLOAD=sram CCM_EN=1 clean
# Build and upload the application
make SOC=evalsoc BOARD=nuclei_fpga_eval CORE=ux900 DOWNLOAD=sram CCM_EN=1 upload
```

Expected output(DISABLE_NMSIS_HPM defined) as below:

```
Nuclei SDK Build Time: Feb 14 2023, 18:14:18
Download Mode: SRAM
CPU Frequency 100605952 Hz
CPU HartID: 0
DCache Linesize is 64 bytes, ways is 2, setperway is 512, total size is 65536 bytes

array_test 10 * 64 bytes

------Update array in memory------

------Update array to all 0xab in cache: array_update_by_row------

-------Keep DCache valid, do array_update_by_row again-------

-------Invalidate all the Dcache-------

------Update array to all 0xab in cache: array_update_by_col ------
Read out array_test[0][0] 0xab in cache, then disable DCache

------Init array in memory to all 0x34------
Read out array_test[0][0] 0x34 in memory, then enable Dcache
After cache flushed to memory, array_test[0][0] in memory is 0xab

------Again init array in memory to all 0x34, then enable DCache------
Read out array_test[0][0] 0x34 in memory
Read out array_test[0][0] 0xab in cache, when mapped value in memory has changed
```

From output, `array_test` is updated in memory to all 0xab, and **cached in DCache** when miss happens, then disable DCache, init array_test just in memory to all 0x34, **after cache flushed to memory**, `array_test` in memory is all 0xab same with `array_test` in DCache. **Disable DCache and init array_test again**, `array_test` now (all 0x34) differs with cached array_test (all 0xab) after DCache enabled.

Expected output(DISABLE_NMSIS_HPM undefined) as below:

```
Nuclei SDK Build Time: Dec 27 2024, 11:07:56
Download Mode: DDR
CPU Frequency 50002001 Hz
CPU HartID: 0
```

(continues on next page)

(continued from previous page)

```
Benchmark initialized
DCache Linesize is 64 bytes, ways is 2, setperway is 512, total size is 65536 bytes

array_test 10 * 64 bytes

------Update array in memory------
High performance monitor initialized

------Update array to all 0xab in cache: array_update_by_row------
CSV, array_update_by_row_cycle, 15544
HPM4:0xf0000021, array_update_by_row_dcache_miss, 21
HPM3:0xf0000011, array_update_by_row_icache_miss, 60

-------Keep DCache valid, do array_update_by_row again-------
CSV, array_update_by_row_cycle, 15164
HPM4:0xf0000021, array_update_by_row_dcache_miss, 3
HPM3:0xf0000011, array_update_by_row_icache_miss, 26

-------Invalidate all the Dcache-------

------Update array to all 0xab in cache: array_update_by_col ------
CSV, array_update_by_col_cycle, 16194
HPM4:0xf0000021, array_update_by_col_dcache_miss, 22
Read out array_test[0][0] 0xab in cache, then disable DCache

------Init array in memory to all 0x34------
Read out array_test[0][0] 0x34 in memory, then enable Dcache
After cache flushed to memory, array_test[0][0] in memory is 0xab

------Again init array in memory to all 0x34, then enable DCache------
Read out array_test[0][0] 0x34 in memory
Read out array_test[0][0] 0xab in cache, when mapped value in memory has changed
HPM4:0xf0000021, dcachemiss_readonebyte, 4
```

From output, `HPM` is enabled, cache miss is counted and `array_test` size is 10 * 64 bytes. **At first, DCache is invalid**, the first time `array_test` update by row has 10 miss(HPM4 shows more, because HPM4 and other execution it wraps bring some); **Keep DCache valid**, update array_test by row again, cache miss decreases rapidly, which means `array_test` has already cached; **Then invalidate all the Dcache**, array_test update by col seems has the same cache miss as update by row.

Expected output(BIG_ROW_SIZE defined, DISABLE_NMSIS_HPM undefined) as below:

```
Nuclei SDK Build Time: Dec 27 2024, 11:07:28
Download Mode: DDR
CPU Frequency 50002001 Hz
CPU HartID: 0
Benchmark initialized
DCache Linesize is 64 bytes, ways is 2, setperway is 512, total size is 65536 bytes

array_test 2048 * 64 bytes

------Update array in memory------
High performance monitor initialized
```

(continues on next page)

(continued from previous page)

```
------Update array to all 0xab in cache: array_update_by_row------
CSV, array_update_by_row_cycle, 3166169
HPM4:0xf0000021, array_update_by_row_dcache_miss, 2076
HPM3:0xf0000011, array_update_by_row_icache_miss, 58

-------Keep DCache valid, do array_update_by_row again-------
CSV, array_update_by_row_cycle, 3195588
HPM4:0xf0000021, array_update_by_row_dcache_miss, 2058
HPM3:0xf0000011, array_update_by_row_icache_miss, 27

-------Invalidate all the Dcache-------

------Update array to all 0xab in cache: array_update_by_col ------
CSV, array_update_by_col_cycle, 5091193
HPM4:0xf0000021, array_update_by_col_dcache_miss, 130975
Read out array_test[0][0] 0xab in cache, then disable DCache

------Init array in memory to all 0x34------
Read out array_test[0][0] 0x34 in memory, then enable Dcache
After cache flushed to memory, array_test[0][0] in memory is 0xab

------Again init array in memory to all 0x34, then enable DCache------
Read out array_test[0][0] 0x34 in memory
Read out array_test[0][0] 0xab in cache, when mapped value in memory has changed
HPM4:0xf0000021, dcachemiss_readonebyte, 4
```

From output, `array_test` size is enlarged to `2048 * 64` bytes, which is **two times the size of DCache (1024 * 64 bytes)**. Cache miss brought by `HPM` itself ignored, array update by col has **63 times cache miss(130975) as the array update by row has(2076)**. That's because when first byte access brings one cache misse, **one cache line(64 bytes in this demo) is fetched to cache**, and it works best if other 63 cached bytes can be accessed before getting dirty as soon as possible, as update by row does, so the cache miss equals nearly to `ROW_SIZE`, while when updated by col, every byte in `ROW_SIZE` * `COL_SIZE` will cause a cache miss! which is cache-unfriendly. What's more, considering `array_test` size is two times the size of DCache, the cached data has been kicked out when `do array_update_by_row again`, so the cache miss is nearly the same as the first time.

demo_stack_check

Note:

- It doesn't work with gd32vf103 processor.
- It needs Nuclei CPU configured with stack check feature

This demo_stack_check application[110] is used to demonstrate how to check stack overflow and underflow and track the `sp`.

For now, this demo needs to run on **only 300 Series v4.2.0 or later**, which supports this `Stack Check` function.

Note:

[110] https://github.com/Nuclei-Software/nuclei-sdk/tree/master/application/baremetal/demo_stack_check

- The Stack Check can work as expected only when the stack downwardly grows.

- `STACK_TOP`, `STACK_BOTTOM`, `STACK_SIZE` refers to stack's high/low address and size in bytes, which gets from the linker script
- `stack_corrupt_exception_handler` is registered as exception handler to process stack overflow and underflow
- A simple recursive function of calculating factorial is reformed, which will consume stack more or less by the `n` input, thus may cause overflow; a trick is used to cause underflow that when it iterates over, decrease the stack base value to make the underflow condition on purpose
- The `sp` has grown downwardly 0x50 bytes in the exception entry saving context, in this demo, add `sp` by 0x50 is the `sp` value that triggered overflow/underflow
- When it comes into exception and handle it over, the flow doesn't stop in it as usual, and `pc` continues to execute, which is on purpose to show `overflow`, `underflow` and `track sp` mode in one-time run
- In `sp track mode`, logging is enabled in `factorial`, to show the `sp` value change; and the BOUND won't track sp(won't change) if sp is bigger in the second run

---

Note:

- Must set the BOUND and BASE before setting the check mode
- Reserve 0x200 bytes for exception stack push/pop

How to run this application:

```
# Assume that you can set up the Tools and Nuclei SDK environment
# Use Nuclei n300 Core RISC-V processor as example
# cd to the demo_stack_check directory
cd application/baremetal/demo_stack_check
# Clean the application first
make SOC=evalsoc BOARD=nuclei_fpga_eval DOWNLOAD=ddr CORE=n300 clean
# Build and upload the application
make SOC=evalsoc BOARD=nuclei_fpga_eval DOWNLOAD=ddr CORE=n300 upload
```

Expected output as below:

```
Nuclei SDK Build Time: Oct 18 2023, 18:45:02
Download Mode: ILM
CPU Frequency 15996682 Hz
CPU HartID: 0
Stack's top high address: 0x90010000, stack's bottom low address: 0x9000fa00, stack␣
↪size: 0x600

--------OVERFLOW CHECK MODE--------
BOUND register set to: 0x9000fa00
BASE register set to: 0x90010000
Stack overflow fault occurs at iteration 84, cause: 0x30000018, epc: 0x80000e90, sp:␣
↪0x9000f990

--------UNDERFLOW CHECK MODE--------
BASE register set to: 0x9000fd00
Stack underflow fault occurs at iteration 1, cause: 0x30000019, epc: 0x80000fd0, sp:␣
```

(continues on next page)

(continued from previous page)

```
↪0x9000fd00
BASE register set to: 0x90010000

--------TRACK SP MODE--------
BOUND register set to: 0x90010000
Iterations: 1, stack bound: 0x9000fdc0
Iterations: 2, stack bound: 0x9000fd70
Iterations: 3, stack bound: 0x9000fd20
Iterations: 4, stack bound: 0x9000fcd0
Iterations: 5, stack bound: 0x9000fc80
Iterations: 6, stack bound: 0x9000fc30
Iterations: 7, stack bound: 0x9000fbe0
Iterations: 8, stack bound: 0x9000fb90
Iterations: 9, stack bound: 0x9000fb40
Iterations: 10, stack bound: 0x9000faf0
Iterations: 11, stack bound: 0x9000faa0
Iterations: 12, stack bound: 0x9000fa50
Iterations: 13, stack bound: 0x9000fa00
Iterations: 14, stack bound: 0x9000f9b0
Iterations: 15, stack bound: 0x9000f960
Iterations: 16, stack bound: 0x9000f910
Iterations: 17, stack bound: 0x9000f8c0
Iterations: 18, stack bound: 0x9000f870
Calculate factorial over, the max stack used downwards to: 0x9000f820

Rerun it. The BOUND won't track sp if sp is bigger:
Iterations: 1, stack bound: 0x9000f820
Iterations: 2, stack bound: 0x9000f820
Iterations: 3, stack bound: 0x9000f820
Iterations: 4, stack bound: 0x9000f820
Iterations: 5, stack bound: 0x9000f820

Stack check demo over!
```

demo_pma

This demo_pma application[111] is used to demonstrate how to set memory region to different attribute(`Device/Non-Cacheable/Cacheable`)

Note:

- PMA are split into three attributes: `Device/Cacheable/Non-Cacheable`, which correspondingly the whole memory region are split into
- Take care to set the region type and address range, which maybe causing function or performance issue!
- `NMSIS/Core/Include/core_feature_pma.h` provides apis like `PMA_DisableHwXXRegion`, `PMA_EnableHwXXRegion` to disable/enable hardware-defined regions, but please take care to use it, because maybe the region you disable will go to `Device` (maybe covered by another bigger-range `Device` region!), then instruction fetch exception happens!

---

[111] https://github.com/Nuclei-Software/nuclei-sdk/tree/master/application/baremetal/demo_pma

- Observe cycles taken when executing same task array_read_by_row``(read from ``array_test, update into array_test_r) by changing the same memory region to Non-Cacheable/Cacheable
- Struct PMA_CONFIG is used to assign PMA, which consists of region_type region_base region_size region_enable
- region_type could be PMA_REGION_TYPE_SECSHARE, PMA_REGION_TYPE_NC, PMA_REGION_TYPE_DEV, PMA_REGION_TYPE_CA
- region_base is base physical address, which needs to be 4K byte aligned
- region_size needs to be 4K byte aligned; the region_base should be integer multiples of region_size
- region_enable enable(1) or disable(0) the region, could be PMA_REGION_ENA, PMA_REGION_DIS
- After pma_cfg is assigned, and give the entry_idx, call PMA_SetRegion to take effect
- The entry_idx (0-n) depends on number of paired mattri(n)_mask and mattri(n)_base, refer to Nuclei ISA specifications for max region entries
- The api will do aligning by 4KB(because region granularity is 4KB) to region_base and region_size forcely
- The regions can be overlapped as the priority: Non-Cacheable > Cacheable > Device, , but especially be careful not to overlap software's instruction/data sections by Device, or overlap Device(like uart) memory by Cacheable
- PMA_GetRegion could retrieve the region info detail

How to run this application:

```
# Assume that you can set up the Tools and Nuclei SDK environment
# Use Nuclei ux900 Core RISC-V processor as example
# cd to the demo_pma directory
cd application/baremetal/demo_pma
# Clean the application first
make SOC=evalsoc BOARD=nuclei_fpga_eval CORE=ux900 DOWNLOAD=sram CCM_EN=1 clean
# Build and upload the application
make SOC=evalsoc BOARD=nuclei_fpga_eval CORE=ux900 DOWNLOAD=sram CCM_EN=1 upload
```

Expected output as below:

```
Nuclei SDK Build Time: May 23 2025, 15:02:30
Download Mode: SRAM
CPU Frequency 50005606 Hz
CPU HartID: 0
DCache Linesize is 64 bytes, ways is 2, setperway is 512, total size is 65536 bytes

array_test size: 64 * 64 bytes, addr: 0xa0013000

Set to NonCacheable region
Region type: 0x4,region base addr: 0xa0013000, region size: 0x1000, region status: 1
HPM4:0xf0000021, array_read_by_row_dcache_miss_noncacheable, 64

Set to Cacheable region
Region type: 0x0,region base addr: 0xa0013000, region size: 0x1000, region status: 1
HPM4:0xf0000021, array_read_by_row_dcache_miss_cacheable, 128
```

From output, considering array_read_by_row_dcache_miss_noncacheable counting the common part cache miss including array_test_r which belongs to Cacheable. So array_read_by_row_dcache_miss_cacheable minus array_read_by_row_dcache_miss_noncacheable, we get exactly the cache miss(here is the row number

64) that `array_test` brings in Cacheable region, and it demonstrates `array_test` brings no cache miss in Non-Cacheable region as expected.

Note:

- In Nuclei Evalsoc core ux900 for example, the sram/ddr memory locates originally in hardware-defined Cacheable region(which configured by rtl configuration stage), So this demo first covers original attribute by `NonCacheable`, then `Cacheable` (that's recovered)
- As the prority: `Non-Cacheable > Cacheable > Device`, it can't cover original attribute(`Cacheable`) by `Device`!

5.7.3 FreeRTOS applications

demo

This freertos demo application[112] is to show basic freertos task functions.

- Two freertos tasks are created
- A software timer is created

In Nuclei SDK, we provided code and Makefile for this `freertos demo` application.

- **RTOS = FreeRTOS** is added in its Makefile to include FreeRTOS service
- The **configTICK_RATE_HZ** in `FreeRTOSConfig.h` is set to 100, you can change it to other number according to your requirement.

How to run this application:

```
# Assume that you can set up the Tools and Nuclei SDK environment
# cd to the freertos demo directory
cd application/freertos/demo
# Clean the application first
make SOC=gd32vf103 clean
# Build and upload the application
make SOC=gd32vf103 upload
```

Expected output as below:

```
Nuclei SDK Build Time: Feb 21 2020, 14:56:00
Download Mode: FLASHXIP
CPU Frequency 109058823 Hz
Before StartScheduler
Enter to task_1
task1 is running 0.....
Enter to task_2
task2 is running 0.....
timers Callback 0
timers Callback 1
task1 is running 1.....
task2 is running 1.....
```

(continues on next page)

[112] https://github.com/Nuclei-Software/nuclei-sdk/tree/master/application/freertos/demo

(continued from previous page)

```
timers Callback 2
timers Callback 3
task1 is running 2.....
task2 is running 2.....
timers Callback 4
timers Callback 5
task1 is running 3.....
task2 is running 3.....
timers Callback 6
timers Callback 7
task1 is running 4.....
task2 is running 4.....
timers Callback 8
timers Callback 9
task1 is running 5.....
task2 is running 5.....
timers Callback 10
timers Callback 11
```

smpdemo

This freertos smpdemo application[113] is to show basic freertos smp task functions.

- x freertos tasks(different priorities) are created if your cpu has x cores according to the `SMP=x` settings
- A software timer is created
- Need to run using **DOWNLOAD=sram** mode

In Nuclei SDK, we provided code and Makefile for this `freertos smpdemo` application.

- **RTOS = FreeRTOS** is added in its Makefile to include FreeRTOS service
- The **configTICK_RATE_HZ** in `FreeRTOSConfig.h` is set to 100, you can change it to other number according to your requirement.

How to run this application:

```
# Assume that you can set up the Tools and Nuclei SDK environment
# cd to the freertos demo directory
cd application/freertos/smpdemo
# This need to run on NX900 SMPx2 CPU
# Clean the application first
make clean
# Build and upload the application
make upload
```

Expected output as below:

```
Nuclei SDK Build Time: May 28 2024, 13:17:41
Download Mode: SRAM
CPU Frequency 50322800 Hz
CPU HartID: 0
```

(continues on next page)

[113] https://github.com/Nuclei-Software/nuclei-sdk/tree/master/application/freertos/smpdemo

(continued from previous page)

```
Startup FreeRTOS SMP on hartid 0
Enter to task 1
task 1 prio 1 is running 0 on hart 0.....
Enter to task 0
task 0 prio 0 is running 0 on hart 0.....
task 1 prio 1 is running 1 on hart 1.....
task 0 prio 0 is running 1 on hart 0.....
task 1 prio 1 is running 2 on hart 1.....
task 0 prio 0 is running 2 on hart 0.....
task 1 prio 1 is running 3 on hart 1.....
task 0 prio 0 is running 3 on hart 0.....
task 1 prio 1 is running 4 on hart 1.....
task 0 prio 0 is running 4 on hart 0.....
task 1 prio 1 is running 5 on hart 0.....
timers Callback 0 on hart 1
task 0 prio 0 is running 5 on hart 1.....
task 1 prio 1 is running 6 on hart 1.....
task 0 prio 0 is running 6 on hart 0.....
task 1 prio 1 is running 7 on hart 1.....
task 0 prio 0 is running 7 on hart 0.....
task 1 prio 1 is running 8 on hart 1.....
task 0 prio 0 is running 8 on hart 0.....
task 1 prio 1 is running 9 on hart 1.....
task 0 prio 0 is running 9 on hart 0.....
task 1 prio 1 is running 10 on hart 0.....
timers Callback 1 on hart 1
```

5.7.4 UCOSII applications

demo

This ucosii demo application[114] is show basic ucosii task functions.

- 4 tasks are created
- 1 task is created first, and then create 3 other tasks and then suspend itself

In Nuclei SDK, we provided code and Makefile for this `ucosii demo` application.

- **RTOS = UCOSII** is added in its Makefile to include UCOSII service
- The **OS_TICKS_PER_SEC** in `os_cfg.h` is by default set to 50, you can change it to other number according to your requirement.

How to run this application:

```
# Assume that you can set up the Tools and Nuclei SDK environment
# cd to the ucosii demo directory
cd application/ucosii/demo
# Clean the application first
make SOC=gd32vf103 clean
```

(continues on next page)

[114] https://github.com/Nuclei-Software/nuclei-sdk/tree/master/application/ucosii/demo

(continued from previous page)

```
# Build and upload the application
make SOC=gd32vf103 upload
```

Expected output as below:

```
Nuclei SDK Build Time: Feb 21 2020, 15:00:35
Download Mode: FLASHXIP
CPU Frequency 108524271 Hz
Start ucosii...
create start task success
start all task...
task3 is running... 1
task2 is running... 1
task1 is running... 1
task3 is running... 2
task2 is running... 2
task3 is running... 3
task2 is running... 3
task1 is running... 2
task3 is running... 4
task2 is running... 4
task3 is running... 5
task2 is running... 5
task1 is running... 3
task3 is running... 6
task2 is running... 6
task3 is running... 7
task2 is running... 7
task1 is running... 4
task3 is running... 8
task2 is running... 8
task3 is running... 9
task2 is running... 9
task1 is running... 5
task3 is running... 10
task2 is running... 10
task3 is running... 11
task2 is running... 11
task1 is running... 6
task3 is running... 12
task2 is running... 12
```

5.7.5 RT-Thread applications

demo

This rt-thread demo application[115] is show basic rt-thread thread functions.

- main function is a pre-created thread by RT-Thread
- main thread will create 5 test threads using the same function `thread_entry`

In Nuclei SDK, we provided code and Makefile for this `rtthread demo` application.

- **RTOS = RTThread** is added in its Makefile to include RT-Thread service
- The **RT_TICK_PER_SECOND** in `rtconfig.h` is by default set to *100*, you can change it to other number according to your requirement.

How to run this application:

```
# Assume that you can set up the Tools and Nuclei SDK environment
# cd to the rtthread demo directory
cd application/rtthread/demo
# Clean the application first
make SOC=gd32vf103 clean
# Build and upload the application
make SOC=gd32vf103 upload
```

Expected output as below:

```
Nuclei SDK Build Time: Apr 14 2020, 10:14:30
Download Mode: FLASHXIP
CPU Frequency 108270000 Hz

 \ | /
- RT -     Thread Operating System
 / | \     3.1.3 build Apr 14 2020
 2006 - 2019 Copyright by rt-thread team
Main thread count: 0
thread 0 count: 0
thread 1 count: 0
thread 2 count: 0
thread 3 count: 0
thread 4 count: 0
thread 0 count: 1
thread 1 count: 1
thread 2 count: 1
thread 3 count: 1
thread 4 count: 1
Main thread count: 1
thread 0 count: 2
thread 1 count: 2
thread 2 count: 2
thread 3 count: 2
thread 4 count: 2
thread 0 count: 3
```

(continues on next page)

[115] https://github.com/Nuclei-Software/nuclei-sdk/tree/master/application/rtthread/demo

(continued from previous page)

```
thread 1 count: 3
thread 2 count: 3
thread 3 count: 3
thread 4 count: 3
Main thread count: 2
thread 0 count: 4
thread 1 count: 4
```

msh

This rt-thread msh application[116] demonstrates msh shell in serial console which is a component of rt-thread.

- `MSH_CMD_EXPORT(nsdk, msh nuclei sdk demo)` exports a command `nsdk` to msh shell

In Nuclei SDK, we provided code and Makefile for this `rtthread msh` application.

- **RTOS = RTThread** is added in its Makefile to include RT-Thread service
- **RTTHREAD_MSH := 1** is added in its Makefile to include RT-Thread msh component
- The **RT_TICK_PER_SECOND** in `rtconfig.h` is by default set to *100*, you can change it to other number according to your requirement.
- To run this application in *Nuclei Eval SoC* (page 65), the SoC clock frequency must be above 16MHz, if run in 8MHz, uart read is not correct due to bit error in uart rx process.

How to run this application:

```
# Assume that you can set up the Tools and Nuclei SDK environment
# cd to the rtthread msh directory
cd application/rtthread/msh
# Clean the application first
make SOC=gd32vf103 clean
# Build and upload the application
make SOC=gd32vf103 upload
```

Expected output as below:

```
Nuclei SDK Build Time: Dec 23 2020, 16:39:21
Download Mode: FLASHXIP
CPU Frequency 108810000 Hz

 \ | /
- RT -     Thread Operating System
 / | \     3.1.3 build Dec 23 2020
 2006 - 2019 Copyright by rt-thread team
Hello RT-Thread!
msh >help
RT-Thread shell commands:
list_timer       - list timer in system
list_mailbox     - list mail box in system
list_sem         - list semaphore in system
list_thread      - list thread
```

(continues on next page)

[116] https://github.com/Nuclei-Software/nuclei-sdk/tree/master/application/rtthread/msh

(continued from previous page)

```
version          - show RT-Thread version information
ps               - List threads in the system.
help             - RT-Thread shell help.
nsdk             - msh nuclei sdk demo

msh >ps
thread   pri  status      sp     stack size max used left tick  error
-------- ---  ------- ---------- ----------  ------  ---------- ---
tshell     6  ready   0x00000178 0x00001000    09%   0x00000008 000
tidle      7  ready   0x00000078 0x0000018c    30%   0x00000020 000
main       2  suspend 0x000000b8 0x00000200    35%   0x00000013 000
msh >nsdk
Hello Nuclei SDK!
msh >
```

demo_smode

This rt-thread demo smode application[117] is show how to use rt-thread in S-Mode.

It is similar to the normal rt-thread demo, but rt-thread itself is running in S-Mode, so we have to do some PMP and TEE configuration in M-Mode before go to S-Mode.

The main feature required is the TEE, and SSTC is also preferred.

How to run this application:

```
# Assume that you can set up the Tools and Nuclei SDK environment
# cd to the rtthread demo_smode directory
cd application/rtthread/demo_smode
# Clean the application first
# Assume you are using n300
make SOC=evalsoc CORE=n300 clean
# Build and upload the application
make SOC=evalsoc CORE=n300 upload
```

Expected output as below:

```
Nuclei SDK Build Time: Feb 21 2025, 11:12:24
Download Mode: ILM
CPU Frequency 16005857 Hz
CPU HartID: 0
Set ECLIC Timer S-Mode Interrupt and Software Timer S-Mode Interrupt to be executed in S-
↪Mode
Drop to S-Mode to prepare RT-Thread Environment

 \ | /
- RT -     Thread Operating System
 / | \     3.1.5 build Feb 21 2025
 2006 - 2020 Copyright by rt-thread team
Main thread count: 0
thread 0 count: 0
```

(continues on next page)

[117] https://github.com/Nuclei-Software/nuclei-sdk/tree/master/application/rtthread/demo_smode

(continued from previous page)

```
thread 1 count: 0
thread 2 count: 0
thread 3 count: 0
thread 4 count: 0
thread 0 count: 1
thread 1 count: 1
thread 2 count: 1
thread 3 count: 1
thread 4 count: 1
Main thread count: 1
thread 0 count: 2
thread 1 count: 2
thread 2 count: 2
thread 3 count: 2
thread 4 count: 2
thread 0 count: 3
thread 1 count: 3
thread 2 count: 3
thread 3 count: 3
thread 4 count: 3
Main thread count: 2
```

5.7.6 ThreadX applications

demo

This threadx demo application[118] is show basic ThreadX thread functions.

This threadx demo is modified based on https://github.com/eclipse-threadx/threadx/blob/v6.4.1_rel/samples/demo_threadx.c

In Nuclei SDK, we provided code and Makefile for this `threadx demo` application.

- **RTOS = ThreadX** is added in its Makefile to include ThreadX service
- The **TX_INCLUDE_USER_DEFINE_FILE** macro is defined in Makefile, so you can include customized user configuration file `tx_user.h`

How to run this application:

```
# Assume that you can set up the Tools and Nuclei SDK environment
# cd to the threadx demo directory
cd application/threadx/demo
# Clean the application first
make SOC=evalsoc clean
# Build and upload the application
make SOC=evalsoc upload
```

Expected output as below:

```
Nuclei SDK Build Time: May 28 2024, 13:26:41
Download Mode: ILM
```

(continues on next page)

[118] https://github.com/Nuclei-Software/nuclei-sdk/tree/master/application/threadx/demo

(continued from previous page)

```
CPU Frequency 50322800 Hz
CPU HartID: 0
thread 6_7 is running, current is 6, thread 6 counter 1, thread 7 counter 1
thread 6_7 is running, current is 7, thread 6 counter 2, thread 7 counter 1
thread 6_7 is running, current is 6, thread 6 counter 2, thread 7 counter 2
thread 6_7 is running, current is 7, thread 6 counter 3, thread 7 counter 2
thread 6_7 is running, current is 6, thread 6 counter 3, thread 7 counter 3
thread 6_7 is running, current is 7, thread 6 counter 4, thread 7 counter 3
thread 6_7 is running, current is 6, thread 6 counter 4, thread 7 counter 4
thread 6_7 is running, current is 7, thread 6 counter 5, thread 7 counter 4
```

CHAPTER

SIX

CHANGELOG

6.1 V0.8.0

Note:

- Two new benchmark cases `dhrystone_v2.2` and `whetstone_v1.2` are added in this release.
- In Nuclei Studio IDE, if you are importing this Nuclei SDK 0.8.0 as a NPK package, you will be able to see following versions in new project wizard:
  - **Dhrystone Benchmark, Version 2.1**: located in `application/baremetal/benchmark/dhrystone`, previous existed version
  - **Whetstone Benchmark, Roy Longbottom Version**: located in `application/baremetal/benchmark/whetstone`, previous existed version
  - **Dhrystone Benchmark, Version 2.2**: located in `application/baremetal/benchmark/dhrystone_v2.2`, **new** introduced version
  - **Whetstone Benchmark, Netlib Version 1.2**: located in `application/baremetal/benchmark/whetstone_v1.2`, **new** introduced version

This is release version `0.8.0` of Nuclei SDK.

- NMSIS
  - Fix wrong macro `PLIC_GetThreshold` & `PLIC_GetThreshold_S` implementation for `core_feature_plic.h`
  - Add `MTIME_SRW_CTRL` bitfields in **SysTimer_Type** structure for `core_feature_timer.h`
  - Optimize ECLIC API for better code performance in `core_feature_eclic.h`
  - Add SSTC support in `core_feature_timer.h`, a new macro called `__SSTC_PRESENT` is added
  - Update and add more CSR Union types
  - Add more CSR macros such shartid csr, worldguard csrs, and related csr bitfield macro
  - Add the `BENCH_XLEN_MODE` macro to enable more accurate cycle and HPM counter measurements for **RV32**, when `BENCH_XLEN_MODE` is enabled, the cycle/instret/time/hpm_counter will be 32 bits for rv32 and 64 bits for rv64.
  - Fix return type error of `__get_hpm_counter`
  - Add new APIs to read `cycle/instret/time/hpm_counter` with `XLEN` bits:
    - `unsigned long __read_cycle_csr()`

    * unsigned long __read_instret_csr()
    * unsigned long __read_time_csr()
    * unsigned long __read_hpm_counter(unsigned long idx)
  - Fix __clear_core_irq_pending and __clear_core_irq_pending_s implementation in core_feature_base.h
  - Fix __enable_sw_irq_s implementation in core_feature_base.h
  - Add PMA(Physical Memory Attribute) APIs for managing attribute type(Device/Non-Cacheable/Cacheable) of memory regions when **__PMA_PRESENT=1**
  - Fix and update HPM v1 event macro due to Nuclei ISA documentation update in nmsis_bench.h
  - Add new PMU v1 and v2 event macros in nmsis_bench.h
  - Add flushpipe and fence in each ccm operation API in core_feature_cache.h
  - Use 1UL instead of 1 in NMSIS/Core header files to avoid left shift overflow issue
- Application
  - Add more application code compile check message for better example requirement explanation
  - Add *demo_sstc* (page 114) to show how to SSTC(S-Mode timer interrupt extension)
  - Add *demo_smode* (page 140) to show how to run rt_thread in S-Mode, it will require TEE and PMP extension
  - Remove demo_spmp application due to hw sPMP upgraded to sMPU and no longer supported,
  - please use *demo_smpu* (page 115) now.
  - Add -fno-tree-tail-merge compiler option for threadx RTOS example compiling, which is required for correct
  - compiling
  - Fix *demo_vnice* (page 102) insufficient mask length when vlen > 128
  - Add more documentation for *demo_dsp* (page 97) example
  - Optimize *smphello* (page 99) spinlock usage and update doc for it
  - Optimize *demo_profiling* (page 118) example execution speed on hw from about 5min to 30s by decease the loop count
  - Update *demo* (page 134) example to use configTICK_TYPE_WIDTH_IN_BITS instead of configUSE_16_BIT_TICKS
  - Add *demo_pma* (page 132) case to show how to use PMA related API in core_feature_pma.h
  - Add *demo_smode_plic* (page 113) to show how to use PLIC in S-Mode, it will require PLIC and PMP extension
  - Increase freertos timer stack size from 256 to 512 due to timer task still generate vector instruction even with *AUTOVEC=0* (page 37)
  - Add two new benchmark cases *dhrystone_v2.2* (page 106) and *whetstone_v1.2* (page 109) which are the ones used in linux benchmark
  - Update Terapines ZCC dhrystone and coremark options for ZCC v4.0.0 and give better code size
  - -Ofast is deprecated in clang, use -O3 -ffast-math
- SoC

  - Add more documentation about IAR compiler support and porting notes, especially the vector table alignment with the `MTVT` CSR.
  - Add `nx1000/nx1000f/nx1000fd/ux1000/ux1000f/ux1000fd` in supported CPU *CORE* (page 30) list
  - Only enable i/d cache when ecc not present in evalsoc startup asm code to avoid x-state propagation during rtl simulation
  - Fix `#endif` not placed correctly when XLCFG_TEE=1 and CODESIZE=1 in system_evalsoc.c
  - Only initialize ECLIC SMode related registers when TEE really present for evalsoc
  - Place default vector entry for vector_table_s when SSTC present for evalsoc
  - Add `#define _DEFAULT_SOURCE` in all SoC's newlibc stub implementation to use BSD Standard API when compiler c standard is not gnu c standard `-std=gnu23`, such as `-std=c23`, to fix compiler error `error: implicit declaration of function 'TIMEVAL_TO_TIMESPEC' [-Wimplicit-function-declaration]`
  - Add `__SMODE_PRESENT` macro in `evalsoc.h` to represent s-mode present or not
  - Add support for smode clint and plic support for evalsoc
  - Add a README.md to introduce evalsoc reference implementation of NMSIS Device Templates in `SoC/evalsoc/README.md`
- RTOS
  - Add S-Mode RT-Thread support which rely on TEE feature, SSTC feature is preferred
  - Update FreeRTOS port to use `configTICK_TYPE_WIDTH_IN_BITS` instead of `configUSE_16_BIT_TICKS`
  - Cherry-pick a FreeRTOS incorrect error checking of prvCreateIdleTasks fix, see https://github.com/FreeRTOS/FreeRTOS-Kernel/commit/a49c35b5dc0f1f521eef3ef993d401af7f26f439
  - Add ThreadX module support for both RISC-V 32 and 64 bit
  - Add FreeRTOS lazy fp/vector registers save and restore support
- Build System
  - Add **COMPILE_PREFIX** support for *TOOLCHAIN* (page 28) `nuclei_llvm`, now both `nuclei_llvm` and `nuclei_gnu` support this variable, you can change it like this `COMPILE_PREFIX=/path/to/newgcc/bin/riscv64-unknown-elf-` when do make command
  - Add *AUTOVEC* (page 37) make variable, when **AUTOVEC=0**, it will disable auto vectorization as much as possible, this is useful for some application which require no auto vectorization
  - Add `GDB_UPLOAD_EXTRA_CMDS` make variable to execute extra commands after upload elf file to target
  - Add `run_xlmodel` make target for evalsoc to run Nuclei Near Cycle Model Simulation
- Tools
  - Add exclusive lock when program fpga for `nsdk_cli` tools
  - Update `hpm_parse.py` to match hpm v1 and v2 update
- Misc
  - Attach url of supply doc about debug with multiple FTDI devices in FAQ section

6.2 V0.7.1

This is release version `0.7.1` of Nuclei SDK.

- NMSIS
  - Fix Cache CCM related API compile fail using c++ compiler
  - **mfp16mode** csr is renamed to **mmisc_ctl1** due to hw changes
  - Update prebuilt NMSIS DSP/NN library to release 1.3.1
- SoC
  - Only call `EnableSUCCM` in `_premain_init` process when CCM present and S/U mode present defined in auto generated cpufeature.h
- Misc
  - Fix various typos found in source code and doc
  - Recommend evalsoc user to run *cpuinfo* (page 92) to check cpu features it present
  - If you want to do openocd rtos aware debug, you need to follow note in commit b7ed34e96
  - Evalsoc uart eclic irq maybe not working due to different cpu configuration

6.3 V0.7.0

This is release version `0.7.0` of Nuclei SDK.

- Application
  - Add `demo_plic` case to show how to use PLIC related API in PLIC interrupt mode.
  - Add `demo_clint_timer` case to show how to use systimer in CLINT interrupt mode not ECLIC interrupt mode.
  - Update `demo_pmp` case to make it suitable for when PMP not present.
  - Change download mode from `ddr` to `sram` for smp and cache cases to be suitable for some custom soc sdk.
- NMSIS
  - Add more ECC related macros for `milm_ctl/mdlm_ctl/mcache_ctl` csr
  - Add more PLIC interrupt API in `core_feature_plic.h`
  - Add more interrupt related API when in plic interrupt mode, see changes in `core_feature_base.h`
  - Bump NMSIS version to 1.3.0 with updated NMSIS Core/DSP/NN header files and prebuilt library
- SoC
  - Add **Terapines ZCC NPK** support, require Nuclei Studio >= 2024.06
  - Merge newlib stub code from many files into one file called `stubs.c` for all SoC supported in Nuclei SDK
  - Enable I/D cache for evalsoc before data/bss initialization steps using `cpufeature.h` for faster data initialization
  - gd32vf103 default CORE name changed from `n205` to `n203` which are the same in software
  - gd32vw55x default CORE name changed from `n307fd` to `n300fd` which are the same in software
  - evalsoc default CORE name changed from `n307fd` to `n300fd` which are the same in software

  - Add plic interrupt and exception related handling code for evalsoc
  - Fix BPU is not enabled during startup for startup code for IAR compiler, which will increase performance of 600/900/1000 series a lot
- Build System
  - Introduce `XLCFG_xxx` make variable for evalsoc which is only internally used by Nuclei to overwrite default cpufeature.h macro definition, which will be useful for some applications such as demo_cidu, demo_cache, demo_spmp, demo_smpu and demo_smode_eclic
  - Introduce `ECC_EN` make variable for evalsoc which is only internally used by Nuclei to control whether ECC check is enabled or disabled.
  - Add core `n200e/n202/n202e` and remove `n205/n205e/n305/n307/n307fd` which can be replaced by `n203/n203e/n300/n300f/n300fd`
  - Prebuilt IAR projects and workbench are updated due to evalsoc support changes for plic and clint interrupt modes.
  - Add `SYSCLK` make variable for manually set default SYSTEM_CLOCK macro in evalsoc, it is useful for `CODESIZE=1` case
  - Add `QEMU_MC_EXTOPT` make variable to pass extra Nuclei Qemu `-M` machine options for evalsoc.
  - Add `QEMU_CPU_EXTOPT` make variable to pass extra Nuclei Qemu `-cpu` cpu options for evalsoc.

6.4 V0.6.0

This is release version `0.6.0` of Nuclei SDK.

Note:

- Please use **Nuclei Studio 2024.06** with this Nuclei SDK 0.6.0.
- There are many changes in this release, so we decide to name it as 0.6.0, not 0.5.1
- This version introduced **ThreadX and FreeRTOS-SMP support** for Nuclei RISC-V Processors.
- This version introduced a `profiling` middleware and an example to show code coverage and profiling technology using gcov and gprof in **Nuclei Studio 2024.06**.
- We introduced support for **Nuclei 100 series RISC-V CPU**, but in seperated Nuclei SDK branches called **master_n100** or **develop_n100**, see https://doc.nucleisys.com/nuclei_n100_sdk
- This version introduced support for gd32vw55x chip and Nuclei DLink Board.
- Better **Terapines ZCC** toolchain integrated in Nuclei SDK and Nuclei Studio, try ZStudio Lite version here https://www.terapines.com/products/
- Better **IAR Workbench** support in Nuclei SDK, with Baremetal SMP and FreeRTOS SMP supported.

- Application
  - Add ThreadX RTOS example to show how to use ThreadX in SDK.
  - Add Nuclei 1000 series benchmark flags for benchmark examples.
  - Add `demo_vnice` example to show how to use Nuclei Vector NICE feature.
  - Add `demo_profiling` example to how to use gprof and gcov in Nuclei Studio.

  - Add `smphello`, `demo_cidu` baremetal SMP examples in IAR workbench.
  - Add FreeRTOS `smpdemo` example to show how to use SMP version of FreeRTOS.
  - Optimize and fix `cpuinfo` example for better cpu feature dection.
  - Optimize benchmark gcc13 flags to provide better performance.
  - Fix wrong ipc calculating for benchmark examples.
  - Reset mcycle and minstret when read cycle or instret in benchmark examples.
  - Fix dhrystone strcmp_xlcz.S removed by make clean in windows.
  - Update benchmark flags for benchmark examples when compiled with Terapines ZCC Toolchain.
  - Fix `lowpower` example no need to use `newlib_full` library.
- NMSIS
  - Update many CSR structure defined in `core_feature_base.h` such as `CSR_MCFGINFO_Type`, `CSR_MDLMCTL_Type` and `CSR_MCACHECTL_Type` etc.
  - Add `__set_rv_cycle` and `__set_rv_instret` API to set cycle and instret csr registers.
  - Add `CSR_MTLBCFGINFO_Type` CSR structure in `core_feature_base.h`.
  - Fix protection type error in PMP/sPMP API.
  - Fix wrong CLIC_CLICINFO_VER_Msk and CLIC_CLICINFO_NUM_Msk macro value in `core_feature_eclic.h`
  - Add `__ROR64` in `core_compatiable.h`.
  - Add and update DSP intrinsic APIs in `core_feature_dsp.h`.
  - Add and update Nuclei customized CSRs in `riscv_encoding.h`.
  - Sync NMSIS DSP/NN library 1.2.1
- SoC
  - Redesign `evalsoc` reference SoC support software for better evalsoc and nuclei cpu support, see *Usage* (page 66)
  - Remove `-msave-restore` in npk.yml to fix dhrystone benchmark value is low in Nuclei Studio issue.
  - No need to get system clock using `get_cpu_freq` for gd32vf103.
  - In npk.yml, when pass `-isystem=` should be changed to `-isystem =` as a workaround for Nuclei Studio to pass correct system include header.
  - Update standard c library and arch ext prompt for soc npk.yml for better hints.
  - Add `gd32vf103c_dlink` board support for Nuclei DLink development.
  - Fix non-ABS relocation R_RISCV_JAL against symbol '_start' fail for nuclei_llvm toolchain
  - Add Nuclei `ux1000fd` support in both NPK and Makefile based Build System.
  - Add support for **gd32vw55x** SoC which is Gigadevice new Nuclei RISC-V N300 Processor based WiFi MCU.
  - Add **SPLITMODE** support for **evalsoc** when evaluate NA class Core.
  - Allow custom linker script if npk variable `linker_script` is not empty.
  - Explicit declare asm function in gcc asm code if that part of code is a function, which is required by `gprof` plugin in Nuclei Studio.

  - Clear zc bit for non zc elf in mmsic_ctl csr for cases when cpu is not reset but zc bit is set before.
  - Only print CSR value when CSR is present during `__premain_init` for **evalsoc**.
  - Fix undefined symbol when link cpp for clang `__eh_frame_start/__eh_frame_hdr_start/__eh_frame_end/__eh_frame_hdr_end`
  - Add **LDSPEC_EN**, **L2_EN** and **BPU_EN** for evalsoc in Makefile based build system to control load speculative, L2 cache and BPU enable or disable, which is only internally used.
  - Move eclic and interrupt and exception initialization from startup asm code into premain c code for evalsoc.
  - Optimize cpu startup when ECLIC not present it will not be initialized, which is helpful for CPU without ECLIC unit.
  - evalsoc `SystemIRegionInfo` variable is removed now, if you want to access to the base address of cpu internal device, you can use `*_BASEADDR`, such as `__CIDU_BASEADDR`.
  - Introduce an IAR startup asm code called `IAR/startup.S` for evalsoc to support SMP boot, and for SMP stack setup, different IAR linker script is required, see the iar linker script provided in `smphello` or `freertos/smpdemo`.

- Build System
  - Now disassemble elf will show no alias instructions
  - Add `u600*/u900*/ux1000fd` into support CORE list
  - Update and optimize toolchain support for Terapines ZCC Toolchain, which can provide better performance
  - In `Build/toolchain/nuclei_gnu.mk`, `-mmemcpy-strategy=scalar` option is replaced by `-mstringop-strategy=scalar` in official gcc 14, see https://gcc.gnu.org/git/?p=gcc.git;a=commit;h=4ae5a7336ac8e1ba57ee1e885b5b76ed86cdbfd5

- RTOS
  - Bump FreeRTOS Kernel to V11.1.0
  - Bump RTThread Nano to V3.1.5
  - Introduce FreeRTOS SMP support for Nuclei RISC-V CPU
  - Introduce Eclipse ThreadX v6.4.1 Support for Nuclei RISC-V CPU

- Misc
  - Add `Zc/Zicond` and `1000` series support in SDK CLI script used internally
  - Optimize gitlab ci jobs to speedup job execution time and better merge request pipeline check

6.5 V0.5.0

This is release version `0.5.0` of Nuclei SDK, please use it with Nuclei Studio 2023.10[119] release.

Note:

- This 0.5.0 version is a big change version for Nuclei SDK, it now support Nuclei Toolchain 2023.10[120], which have gnu toolchain and llvm toolchain in it, gcc version increased to gcc 13, and clang version used is clang 17. It will no longer support old gcc 10 version, since gcc and clang `-march` option changed a lot, such as b extension changed to `_zba_zbb_zbc_zbs`.

[119] https://github.com/Nuclei-Software/nuclei-studio/releases/tag/2023.10
[120] https://github.com/riscv-mcu/riscv-gnu-toolchain/releases/tag/nuclei-2023.10

- This version also introduced other compiler support such as terapines zcc and IAR compiler. For terapines zcc compiler, please visit https://www.terapines.com/ to contact them for toolchain evaluation, pass `TOOLCHAIN=terapines` during make to select terapines zcc compiler. For IAR compiler, please visit https://www.iar.com/riscv for IAR workbench evaluation, we provided iar projects to take a try with it.
- This version introduced libncrt v3.0.0 support, which split libncrt library into three parts, you need to take care when using newer toolchain.
- This version removed demosoc support, please use evalsoc instead.
- This version introduced qemu 8.0 support, old qemu will not be supported.
- This version introduced Nuclei Studio 2023.10 support which introduced llvm toolchain support via npk, so it can only works with 2023.10 or later version.
- This version required a lot of new npk features introduced in Nuclei Studio 2023.10[121], so it can only be imported as npk package in Nuclei Studio 2023.10[122] or later version.

- Application
  - Add cpuinfo case to dump nuclei cpu feature
  - Add stack check demo to demostrate nuclei stack check feature
  - Add support for gcc13/clang17/terapines/iar compiler
  - Fix missing break in `__set_hpm_event` function, take care if you are using this API.
  - For different compiler option support, we introduced `toolchain_$(TOOLCHAIN).mk` file to place toolchain specified options, see benchmark examples' Makefile
  - Optimize demo_cidu smp case
  - Optimize application code and makefile when port for clang, terapines zcc and iar compiler
  - Change *ARCH_EXT* (page 31) makefile comment for demo_dsp when using gcc 13
  - Auto choose proper CPU_SERIES and proper optimization flags for benchmark cases
  - Optimize whetstone cost to decrease execution time for better ci testing in qemu and fpga
  - Add Zc and Xxlcz extension optimization for coremark and dhrystone cases
  - Do specical adaption for demo_pmp/demo_spmp for iar compiler which require customized iar linker icf for this cases
  - Optimize benchmark flags when using gcc 13
- NMSIS
  - Add bench reset/sample/stop/stat and get usecyc/sumcyc/lpcnt APIs in NMSIS Core
  - Add more CSRs such as Zc/Stack Check in riscv_encoding.h
  - Rename NMSIS DSP/NN library name to match gcc 13 changes, eg. `b -> zba_zbb_zbc_zbs`, so the library name changed a lot
  - Add IAR compiler support in NMSIS Core
  - No more bitmanip extension intrinsic header <`rvintrin.h`> for gcc13
  - Fix `__RV_CLAMP` macro and add `__MACHINE/SUPERVISOR/USER_INTERRUPT` macros
  - Add `__get_hart_index` and `SysTimer_GetHartID` and modify `__get_hart_id` API

[121] https://github.com/Nuclei-Software/nuclei-studio/releases/tag/2023.10
[122] https://github.com/Nuclei-Software/nuclei-studio/releases/tag/2023.10

  - In <Device.h>, we introduced `__HARTID_OFFSET` and `__SYSTIMER_HARTID` macro to represent timer hart index relation with cpu hartid for AMP SoC
  - Update NMSIS Core/DSP/NN header files to NMSIS 1.2.0[123]
  - Update NMSIS DSP/NN prebuilt library to v1.2.0, and added F16 prebuilt library
- SOC
  - **CAUTION**: Demosoc support is removed since evalsoc is the successor, please use **evalsoc** now.
  - Set **RUNMODE_CCM_EN** macro when **CCM_EN** make variable passed and allow `__CCM_PRESENT` overwrite by **RUNMODE_CCM_EN** macro
  - Enable `__CIDU_PRESENT` macro passed via compiler option
  - Update cpu startup asm code to fix clang compile issue such as STB_WEAK warning and non-ABS relocation error
  - Update cpu startup asm code to support zcmt jump table
  - Update gnu linker files to support zcmt extension
  - Update gnu linker files to fix 2 byte gap issue, and align section to 8bytes and reorg sections
  - Update openocd configuration files to support openocd new version
  - Make `metal_tty_putc/getc` with `__USED` attribute to avoid `-flto` build and link fail
  - Add startup and exception code and iar linker icf files for IAR compiler support
  - Add new macros `__HARTID_OFFSET` and `__SYSTIMER_HARTID` in evalsoc.h
  - Add **HARTID_OFFSET** make variable to control hartid offset for evalsoc
  - Boot hartid check no longer only compare lower 8bits for evalsoc
  - Currently IAR compiler support is only for single core support, smp support is not yet ready and need to use in IAR workbench
  - Update Nuclei Studio NPK files to support both gcc and llvm toolchain support, this require Nuclei Studio 2023.10[124], which is incompatiable with previous IDE version.
- Build System
  - Fix semihost not working when link with semihost library
  - Add support for gcc 13, clang 17, terapines zcc toolchain using *TOOLCHAIN* (page 28) make variable, eg. `TOOLCHAIN=nuclei_gnu` for gnu gcc toolchain, `TOOLCHAIN=nuclei_llvm` for llvm toolchain, `TOOLCHAIN=terapines` for terapines zcc toolchain
  - Add support for libncrt v3.0.0, which spilt libncrt into 3 parts, the c library part, fileops part, and heapops part, so *NCRTHEAP* (page 41) and *NCRTIO* (page 41) makefile variable are added to support new version of libncrt, about upgrading libncrt, please check *STDCLIB* (page 39)
  - To support both gcc, clang, zcc, now we no longer use `--specs=nano.specs` like `--specs=` gcc only options, since clang don't support it, we directly link the required libraries according to the library type you want to use in Makefile, group all the required libraries using `--start-group archives --end-group` of linker option, see https://sourceware.org/binutils/docs/ld/Options.html, but when using Nuclei Studio, the Eclipse CDT based IDE didn't provided a good way to do library group, here is an issue tracking it, see https://github.com/eclipse-embed-cdt/eclipse-plugins/issues/592

[123] https://github.com/Nuclei-Software/NMSIS/releases/tag/1.2.0
[124] https://github.com/Nuclei-Software/nuclei-studio/releases/tag/2023.10

    * And also now we defaultly enabled `-nodefaultlibs` option to not use any standard system libraries when linking, so we need to specify the system libraries we want to use during linking, which is the best way to support both gcc and clang toolchain.
  - When using libncrt library, this is no need to link with other libgcc library, c library or math library, such as gcc libgcc library(`-lgcc`), newlib c library(`-lc/-lc_nano`) and math library(`-lm`), the c and math features are also provided in libncrt library
  - When using Nuclei Studio with imported Nuclei SDK NPK package, you might meet with undefined reference issue during link
  - The use of *ARCH_EXT* (page 31) is changed for new toolchain, eg. you can't pass `ARCH_EXT=bp` to represent b/p extension, instead you need to pass `ARCH_EXT=_zba_zbb_zbc_zbs_xxldspn1x`
  - Show CC/CXX/GDB when make showflags
  - Add u900 series cores support
  - No longer support gd32vf103 soc run on qemu
  - Add extra `-fomit-frame-pointer -fno-shrink-wrap-separate` options for Zc extension to enable zcmp instruction generation
  - Extra **CPU_SERIES** macro is passed such (200/300/600/900) during compiling for benchmark examples
  - When you want to select different nmsis library arch, please use *NMSIS_LIB_ARCH* (page 38) make variable, see demo_dsp as example
- Tools
  - A lot of changes mainly in nsdk cli configs have been made to remove support of demosoc, and change it to evalsoc
  - A lot of changes mainly in nsdk cli configs have been made to support newer *ARCH_EXT* (page 31) variable format
  - Add llvm ci related nsdk cli config files
  - Add Zc/Xxlcz fpga benchmark config files
  - Support qemu 8.0 in nsdk cli tools
  - Update configurations due to application adding and updating
- RTOS
  - Add freertos/ucosii/rtthread porting code for IAR compiler
  - Enable vector when startup new task for rtos for possible execute rvv related instruction exception
- Misc
  - Change gitlab ci to use Nuclei Toolchain 2023.10[125]
  - Add IAR workbench workspace and projects for evalsoc, so user can quickly evaluate IAR support in IAR workbench

[125] https://github.com/riscv-mcu/riscv-gnu-toolchain/releases/tag/nuclei-2023.10

6.6 V0.4.1

This is release version `0.4.1` of Nuclei SDK.

- Application
  - Add demo_cidu to demo cidu feature of Nuclei RISC-V Processor
  - Add demo_cache to demo ccm feature of Nuclei RISC-V Processor
  - Optimize demo_nice for rv64
  - Fix compile error when -Werror=shadow
  - Update helloworld and smphello due to mhartid changes
- NMSIS
  - Bump NMSIS to 1.1.1 release version, NMSIS DSP/NN prebuilt libraries are built with 1.1.1 release.
  - Add CIDU support via core_feature_cidu.h, and `__CIDU_PRESENT` macro is required in `<Device>.h` to represent CIDU present or not
  - Add macros of HPM m/s/u event enable, events type, events idx
  - Fix define error of HPM_INIT macro
  - Due to mhartid csr update for nuclei subsystem reference design, two new API added called `__get_hart_id` and `__get_cluster_id`
    - mhartid csr is now used to present cluster id and hart id for nuclei subsystem reference design
    - bit 0-7 is used for hart id in current cluster
    - bit 8-15 is used for cluster id of current cluster
    - for normal nuclei riscv cpu design, the mhartid csr is used as usual, but in NMSIS Core, we only take lower 8bits in use cases like systimer, startup code to support nuclei subsystem
- Build System
  - Add semihost support in build system via SEMIHOST make variable, if SEMIHOST=1, will link semihost library, currently only works with newlibc library, not working with libncrt
  - Add support for compile cpp files with suffix like .cc or .CC
  - Remove `--specs=nosys.specs` compile options used during compiling, since we have implement almost all necessary newlibc stub functions, no need to link the nosys version, which will throw warning of link with empty newlibc stub functions.
- SoC
  - Fix missing definition of BOOT_HARTID in `startup_demosoc.S`
  - Update demosoc and evalsoc interrupt id and handler definition for CIDU changes
  - Add `__CIDU_PRESENT` macro to control CIDU present or not in `demosoc.h` and `evalsoc.h` which is the `<Device>.h`
  - Add uart status get and clear api for evalsoc and demosoc, which is used by cidu demo
  - Add semihost support for all SoCs, currently only works with newlib, `SEMIHOST=1` control semihost support
  - Update openocd configuration file to support semihosting feature
  - Add extra run/restart command for openocd debug configuration in smp debug in npk for Nuclei Studio
  - Update smp/boot flow to match mhartid csr update

  - **BOOT_HARTID** is the choosen boot hart id in current cluster, not the full mhartid register value, for example, it the mhartid csr register is 0x0101, and the **BOOT_HARTID** should be set to 1, if you want hart 1 to be boot hart
  - Update and add more newlib stub functions in demosoc/evalsoc/gd32vf103 SoC's newlibc stub implementation, since we are no longer compile with `--specs=nosys.specs`

- CI
  - Add demo_cidu and demo_cache in ci configuration files, but expect it to run fail when run in qemu
  - Don't check certificate when download tool

- Tools
  - Modify openocd configuration file in nsdk_utils.oy support win32 now
  - Add new feature to generate cpu json when knowing cpu arch in nsdk_runcpu.py script
  - Add runresult_diff.py script to compare the difference of two runresult.xlsx.csvtable.json files, useful when do benchmark difference check
  - Add `--uniqueid <id>` option for nsdk cli tools

6.7 V0.4.0

This is release version `0.4.0` of Nuclei SDK.

- Application
  - Add *demo_pmp* (page 122) application to demostrate pmp feature.
  - Add *demo_spmp* (page 115) application to demostrate smode pmp feature, spmp is present when TEE feature is enabled.
  - Add *demo_smode_eclic* (page 110) application to demonstrate ECLIC interrupt with TEE feature of Nuclei Processor.
  - Changed `test/core` test case due to `EXC_Frame_Type` struct member name changes.
  - Fix XS bit set bug in demo_nice application.
  - Add return value in smphello application.

- NMSIS
  - Add `__CTZ` count trailing zero API in core_compatiable.h
  - Add `__switch_mode` switch risc-v privilege mode API in core_feature_base.h
  - Add `__enable_irq_s`, `__disable_irq_s` smode irq control(on/off) API in core_feature_base.h
  - Add `__set_medeleg` exception delegation API in core_feature_base.h
  - Update and add smode eclic related API in core_feature_eclic.h only present when **TEE_PRESENT=1**
  - Optimize APIs of PMP and add `__set_PMPENTRYx` and `__get_PMPENTRYx` API for easily PMP configuration in core_feature_pmp.h
  - Add spmp related APIs for smode pmp hardware feature when **__SPMP_PRESENT=1**
  - Add per-hart related APIs for systimer such as `SysTimer_SetHartCompareValue`, `SysTimer_SetHartSWIRQ` and etc in core_feature_timer.h, this is mainly needed when configure timer in smode per hart

  - Add TEE related csr macros in riscv_encoding.h
  - Add iregion offset macros and N3/VP mask in riscv_encoding.h and use it in demosoc/evalsoc implementation.
  - Add `ICachePresent/DCachePresent` API
  - Don't sub extra cost for BENCH_xxx API
  - Update NMSIS Core/DSP/NN and prebuilt library to version 1.1.0
- Build System
  - Add `intexc_<Device>_s.S` asm file into compiling for evalsoc and demosoc
  - Show `ARCH_EXT` information when run make info
  - Don't specify elf filename when run gdb, only specify it when do load to avoid some gdb internal error
  - Add `BOOT_HARTID` and `JTAGSN` support, which need to be done in SoC support code and build system
- SoC
  - Add smode interrupt and exception handling framework for evalsoc and demosoc, for details see code changes.
    * A new section called `.vector_s` is added(required in linker script) to store smode vector table which is initialized in `system_<Device>.c`
    * A new `intexc_<Device>_s.S` asm source file is added to handle s-mode interrupt and exception
    * A default smode exception register and handling framework is added in `system_<Device>.c`
    * **API Changes**: `Exception_DumpFrame` parameters changed to add mode passing in `system_<Device>.c/h`
    * **API Changes**: `EXC_Frame_Type` struct member mcause/mepc changed to cause/epc in `system_<Device>.c/h`
  - Print `\0` instead of `\r` when do simulation exit for better integration in Nuclei Studio QEMU simulation.
  - Add `clock` stub function for libncrt library in demosoc/evalsoc/gd32vf103 SoC support software.
  - Add `sram` download mode for evalsoc/demosoc, for details directly check the linker script
  - Change default `__ICACHE_PRESENT/__DCACHE_PRESENT` to 1 for evalsoc/demosoc, when evalsoc/demosoc startup, it will enable i/d cache if it really present.
  - Update openocd configuration files to remove deprecated command which might not be support in future
  - Merge smp and single core openocd config into one configuration for evalsoc and demosoc
  - Add **BOOT_HARTID** support for evalsoc and demosoc, which is used to specify the boot hartid, used together with **SMP** can support SMP or AMP run mode
  - Add **JTAGSN** support to specify a unified hummingbird jtag debugger via `adapter serial`
  - For AMP support, we can work together with Nuclei Linux SDK, see https://github.com/Nuclei-Software/nsdk_ampdemo
  - Add NPK support for SMP/AMP working mode, and sram download mode
- CI
  - Start to use Nuclei QEMU/Toolchain/OpenOCD 2022.12 in daily ci for gitlab runner
- Tools
  - Add `httpserver.py` tool to create a http server on selected folder, good to preview built documentation.

  - Fix many issues related to nsdk_cli scripts when integrated using fpga hardware ci flow.
  - Support extra parsing benchmark python script for nsdk_cli tools, see 5f546fa0
  - Add `nsdk_runcpu.py` tool to run fpga baremetal benchmark
- Documentation
  - Add `make preview` to preview build documentation.

6.8 V0.3.9

This is release version `0.3.9` of Nuclei SDK.

- Application
  - Add `lowpower` application to demonstrate low-power feature of Nuclei Processor.
  - Update `demo_nice` application due to RTL change in cpu.
  - Change dhrystone compiling options to match better with Nuclei CPU IP.
- NMSIS
  - Update riscv_encoding.h, a lot of changes in the CSRs and macros, VPU are added.
  - Add nmsis_bench.h, this header file will not be included in nmsis_core.h, if you want to use it, please directly include in your source code. It is used to help provide NMSIS benchmark and high performance monitor macro helpers.
  - Add hpm related API in core_feature_base.h
  - Add enable/disable vector API only when VPU available
- Build System
  - Fix upload program the pc is not set correctly to _start when cpu is reset in flash programming mode.
  - Add run_qemu_debug/run_xlspike_rbb/run_xlspike_openocd make targets
- SoC
  - Add npk support for smp, required to update ide plugin in Nuclei Studio 2022.04. And also a new version of qemu is required, if you want to run in qemu.
  - Add `evalsoc` in Nuclei SDK, `evalsoc` is a new evaluation SoC for Nuclei RISC-V Core, for next generation of cpu evaluation with iregion feature support. `demosoc` will be deprecated in future, when all our CPU IP provide iregion support.
  - **Important**: A lot of changes are made to linker script of SDK.
    * rodata are placed in data section for ilm/flash/ddrdownload mode, but placed in text section for flashxip download mode.
    * For ilm download mode, if you want to make the generated binary smaller, you can change REGION_ALIAS of DATA_LMA from `ram` to `ilm`.
    * Add `_text_lma/_text/_etext` to replace `_ilm_lma/_ilm/_eilm`, and startup code now using new ld symbols.
    * Use REGION_ALIAS to make linker script portable
    * Linker scripts of gd32vf103/evalsoc/demosoc are all changed.
  - FPU state are set to initial state when startup, not previous dirty state.

  - Vector are enabled and set to initial state when startup, when vector are enabled during compiling.
  - For latest version of Nuclei CPU IP, BPU cold init need many cycles, so we placed bpu enable before enter to main.

6.9 V0.3.8

This is release version `0.3.8` of Nuclei SDK.

- Application
  - Add `smphello` application to test baremetal smp support, this will do demonstration to boot default 2 core and each hart print hello world.
- NMSIS
  - Some macros used in NMSIS need to expose when DSP present
  - nmsis_core.h might be included twice, it might be included by <Device.h> and <riscv_math.h>
- Build
  - Add `SYSCLK` and `CLKSRC` make variable for gd32vf103 SoC to set system clock in hz and clock source, such as `SYSCLK=72000000 CLKSRC=hxtal`
  - Exclude source files using `EXCLUDE_SRCS` make variable in Makefile
  - `C_SRCS/ASM_SRCS/CXX_SRCS` now support wildcard pattern
  - `USB_DRV_SUPPORT` in gd32vf103 is removed, new `USB_DRIVER` is introduced, `USB_DRIVER=device/host/both` to choose device, host or both driver code.
  - `SMP`, `HEAPSZ` and `STACKSZ` make variable are introduced to control stack/heap size and smp cpu count used in SDK
- SoC
  - Add libncrt 2.0.0 support for demosoc and gd32vf103, libncrt stub functions need to be adapted, see 2e09b6b0 and 2e09b6b0
  - Fix ram size from 20K to 32K for gd32vf103v_eval and gd32vf103v_rvstar
  - Change demosoc eclic/timer baseaddr to support future cpu iregion feature, see eab28320d and 18109d04
  - Adapt system_gd32vf103.c to support control system clock in hz and clock source via macro **SYSTEM_CLOCK** and **CLOCK_USING_IRC8M** or **CLOCK_USING_HXTAL**
  - Merge various changes for gd32vf103 support from `gsauthof@github`, see PR #37, #38, #40
  - Remove usb config header files and usb config source code for gd32vf103
  - Change gd32vf103 linker scripts to support `HEAPSZ` and `STACKSZ`
  - Change demosoc linker scripts to support `HEAPSZ`, `STACKSZ` and `SMP`
  - Add baremetal SMP support for demosoc, user can pass `SMP=2` to build for 2 smp cpu.
- Tools
  - Record more flags in `nsdk_report.py` such as `NUCLEI_SDK_ROOT`, `OPENOCD_CFG` and `LINKER_SCRIPT`.
  - Fix nsdk_report.py generated runresult.xls file content is not correct when some application failed
  - Add benchmark c standard script in tools/misc/barebench
  - Change to support `SMP` variable

- OS
  - RT_HEAP_SIZE defined in cpuport.c is small, need to be 2048 for msh example when RT_USING_HEAP is enabled
  - Application can define RT_HEAP_SIZE in rtconfig.h to change the size

For detailed changes, please check commit histories since 0.3.7 release.

6.10 V0.3.7

This is release version `0.3.7` of Nuclei SDK.

- Application
  - **CAUTION**: Fix benchmark value not correct printed when print without float c library, which means the CSV printed value in previous release is not correct, please take care
  - Add **DHRY_MODE** variable to support different dhrystone run options in dhrystone benchmark, `ground`, `inline` and `best` are supported
- NMSIS
  - Bump to v1.0.4
  - Add B-extension support for NMSIS
  - Fix various issues reported in github
- Build - add `showflags` target to show compiling information and flags - add `showtoolver` target to show tool version used
- SoC
  - Change all un-registered interrupt default handler to `default_intexc_handler`, which means user need to register the interrupt handler using `ECLIC_SetVector` before enable it.
  - Add **RUNMODE** support only in `demosoc`, internal usage
  - Add jlink debug configuration for gd32vf103 soc
- Tools
  - Update `nsdk_report.py` script to support generate benchmark run result in excel.
  - Add `ncycm` cycle model runner support in `nsdk_bench.py`
  - Add `nsdk_runner.py` script for running directly on different fpga board with feature of programing fpga bitstream using vivado

For detailed changes, please check commit histories since 0.3.6 release.

6.11 V0.3.6

This is release version `0.3.6` of Nuclei SDK.

- Application
  - update coremark benchmark options for n900/nx900, which can provide better score number
  - benchmark value will be print in float even printf with float is not supported in c library
  - baremetal applications will exit with an return value in main
- NMSIS
  - add `__CCM_PRESENT` macro in NMSIS-Core, if CCM hardware unit is present in your CPU, `__CCM_PRESENT` macro need to be set to 1 in `<Device>.h`
  - Fixed mtvec related api comment in `core_feature_eclic.h`
  - Add safely write mtime/mtimecmp register for 32bit risc-v processor
  - rearrage #include header files for all NMSIS Core header files
  - removed some not good #pragma gcc diagnostic lines in `nmsis_gcc.h`
- Build
  - Add experimental `run_xlspike` and `run_qemu` make target support
  - `SIMU=xlspike` or `SIMU=qemu` passed in make will auto exit xlspike/qemu if main function returned
- SoC
  - Add xlspike/qemu auto-exit support for gd32vf103 and demosoc, required next version after Nuclei QEMU 2022.01

For detailed changes, please check commit histories since 0.3.5 release.

6.12 V0.3.5

This is release version `0.3.5` of Nuclei SDK.

Caution:

- This version introduce a lot of new features, and required Nuclei GNU Toolchain 2022.01
- If you want to import as NPK zip package into Nuclei Studio, 2022.01 version is required.
- If you want to have smaller code size for Nuclei RISC-V 32bit processors, please define `STDCLIB=libncrt_small` in your application Makefile, or change **STDCLIB** defined in `Build/Makefile.base` to make it available globally.

- Application
  - **DSP_ENABLE** and **VECTOR_ENABLE** are deprecated now in demo_dsp application, please use **ARCH_EXT** to replace it. `ARCH_EXT=p` equal to `DSP_ENABLE=ON`, `ARCH_EXT=v` equal to `VECTOR_ENABLE=ON`.
  - `demo_dsp` application no need to set include and libraries for NMSIS DSP library, just use `NMSIS_LIB = nmsis_dsp` to select NMSIS DSP library and set include directory.

  - Update coremark compile options for different Nuclei cpu series, currently 900 series options and 200/300/600 series options are provided, and can be selected by `CPU_SERIES`.
    * `CPU_SERIES=900`: the compiler options for Nuclei 900 series will be selected.
    * otherwise, the compiler options for Nuclei 200/300/600 series will be selected, which is by default for 300
  - Fix `whetstone` application compiling issue when compiled with v extension present
- SoC
  - Provide correct gd32vf103.svd, the previous one content is messed up.
  - `putchar/getchar` newlib stub are required to be implemented for RT-Thread porting
  - Added support for newly introduced nuclei c runtime library(libncrt).
  - Rearrange stub function folder for gd32vf103 and demosoc to support different c runtime library.
  - A lot changes happened in link scripts under SoC folder - heap section is added for libncrt, size controlled by `__HEAP_SIZE` - heap start and end ld symbols are `__heap_start` and `__heap_end` - stub function `sbrk` now using new heap start and end ld symbols - tdata/tbss section is added for for libncrt, thread local storage supported
  - For **flash** download mode, vector table are now placed in `.vtable` section now instead of `.vtable_ilm`, `VECTOR_TABLE_REMAPPED` macro is still required in **DOWNLOAD=flash** mode
  - flash program algo used in openocd for demosoc changed to nuspi, see changes in openocd_demosoc.cfg
- NMSIS
  - Update NMSIS Core/DSP/NN to version 1.0.3, see NMSIS 1.0.3 Changelog[126]
  - Update prebuilt NMSIS DSP/NN library to version 1.0.3 built by risc-v gcc 10.2
  - For NMSIS Core 1.0.3, no need to define `__RISCV_FEATURE_DSP` and `__RISCV_FEATURE_VECTOR` for `riscv_math.h` now, it is now auto-defined in `riscv_math_types.h`
- OS
  - Change RT-Thread porting to support libncrt and newlibc, mainly using putchar and getchar
- Build System
  - Introduce *STDCLIB* (page 39) makefile variable to support different c library.
  - **NEWLIB** and **PFLOAT** variable is deprecated in this release.
  - Introduce *ARCH_EXT* (page 31) makefile variable to support b/p/v extension.
  - Only link `-lstdc++` library when using **STDCLIB=newlib_xxx**
  - **RISCV_CMODEL** variable is added to choose code model, medlow or medany can be chosen, default is `medlow` for RV32 otherwise `medany` for RV64.
  - **RISCV_TUNE** variable is added to select riscv tune model, for Nuclei CPU, we added `nuclei-200-series`, `nuclei-300-series`, `nuclei-600-series` and `nuclei-900-series` in Nuclei RISC-V GNU toolchain >= 2021.12
- Contribution
  - Update contribution guide due to runtime library choices provided now.
- NPK

[126] https://doc.nucleisys.com/nmsis/changelog.html#v1-0-3

  - **newlibsel** configuration variable changed to **stdclib**, and is not compatiable.
    * **newlibsel=normal** change to **stdclib=newlib_full**
    * **newlibsel=nano_with_printfloat** changed to **stdclib=newlib_small**
    * **newlibsel=nano** changed to **stdclib=newlib_nano**
    * **stdclib** has more options, please see `SoC/demosoc/Common/npk.yml`
    * **nuclei_archext** is added as new configuration variable, see `SoC/demosoc/Common/npk.yml`
- tools
  - generate benchmark values in csv files when running nsdk_bench.py or nsdk_execute.py
  - fix xl_spike processes not really killed in linux environment when running nsdk_bench.py

For detailed changes, please check commit histories since 0.3.4 release.

6.13 V0.3.4

This is release version `0.3.4` of Nuclei SDK.

- CI
  - Fix gitlab ci fail during install required software
- Build System
  - build asm with -x assembler-with-cpp
- Tools
  - Fix `tools/scripts/nsdk_cli/configs/nuclei_fpga_eval_ci_qemu.json` description issue for dsp enabled build configs
  - Generate html report when run `tools/scripts/nsdk_cli/nsdk_bench.py`
  - nsdk_builder.py: modify qemu select cpu args,change `p` to `,ext=p`
- SoC
  - For demosoc, if you choose ilm and ddr download mode, then the data section's LMA is equal to VMA now, and there will be no data copy for data section, bss section still need to set to zero.
  - For demosoc, if you choose ilm and ddr download mode, The rodata section are now also placed in data section.
- NPK
  - add `-x assembler-with-cpp` in npk.yml for ssp

For detailed changes, please check commit histories since 0.3.3 release.

6.14 V0.3.3

This is release version `0.3.3` of Nuclei SDK.

- NPK
  - Fix NPK issues related to QEMU for demosoc and gd32vf103, and RTOS macro definitions in NPK
  - This SDK release required Nuclei Studio 2021.09-ENG1, 2021.08.18 build version

For detailed changes, please check commit histories since 0.3.2 release.

6.15 V0.3.2

This is release version `0.3.2` of Nuclei SDK.

- Build
  - **Important changes** about build system:
    - The SoC and RTOS related makefiles are moving to its own folder, and controlled By **build.mk** inside in in the SoC/<SOC> or OS/<RTOS> folders.
    - Middlware component build system is also available now, you can add you own middleware or library into `Components` folder, such as `Components/tjpgd` or `Components/fatfs`, and you can include this component using make variable `MIDDLEWARE` in application Makefile, such as `MIDDLEWARE := fatfs`, or `MIDDLEWARE := tjpgd fatfs`.
    - Each middleware component folder should create a `build.mk`, which is used to control the component build settings and source code management.
    - An extra `DOWNLOAD_MODE_STRING` macro is passed to represent the DOWNLOAD mode string.
    - In `startup_<Device>.S` now, we don't use `DOWNLOAD_MODE` to handle the vector table location, instead we defined a new macro called `VECTOR_TABLE_REMAPPED` to stand for whether the vector table's vma != lma. If `VECTOR_TABLE_REMAPPED` is defined, the vector table is placed in `.vtable_ilm`, which means the vector table is placed in flash and copy to ilm when startup.
  - Change openocd `--pipe` option to `-c "gdb_port pipe; log_output openocd.log"`
  - Remove `-ex "monitor flash protect 0 0 last off"` when upload or debug program to avoid error when openocd configuration file didn't configure a flash
  - Add `cleanall` target in **<NUCLEI_SDK_ROOT>/Makefile**, you can clean all the applications defined by `EXTRA_APP_ROOTDIRS` variable
  - Fix `size` target of build system
- Tools
  - Add `nsdk_cli` tools in Nuclei SDK which support run applications
    - **tools/scripts/nsdk_cli/requirements.txt**: python module requirement file
    - **tools/scripts/nsdk_cli/configs**: sample configurations used by scripts below
    - **tools/scripts/nsdk_cli/nsdk_bench.py**: nsdk bench runner script
    - **tools/scripts/nsdk_cli/nsdk_execute.py**: nsdk execute runner script
- SoC

  - Add general bit operations and memory access APIs in <Device>.h, eg. _REG32(p, i), FLIP_BIT(regval, bitofs)
  - DOWNLOAD_MODE_xxx macros are now placed in <Device>.h, which is removed from riscv_encoding.h, user can define different DOWNLOAD_MODE_xxx according to its device/board settings.
  - DOWNLOAD_MODE_STRING are now used to show the download mode string, which should be passed eg. -DOWNLOAD_MODE_STRING=\"flash\", it is used in system_<Device>.c
  - DOWNLOAD_MODE_xxx now is used in startup_<Device>.S to control the vector table location, instead a new macro called VECTOR_TABLE_REMAPPED is used, and it should be defined in SoC/<SOC>/build.mk if the vector table's LMA and VMA are different.
- NMSIS
  - Bump NMSIS to version 1.0.2
- OS
  - Fix OS task switch bug in RT-Thread

6.16 V0.3.1

This is official version 0.3.1 of Nuclei SDK.

Caution:

- We are using demosoc to represent the Nuclei Evaluation SoC for customer to replace the old name hbird.
- The hbird SoC is renamed to demosoc, so the SoC/hbird folder is renamed to SoC/demosoc, and the SoC/hbird/Board/hbird_eval is renamed to SoC/demosoc/Board/nuclei_fpga_eval.

- SoC
  - board: Add support for TTGO T-Display-GD32, contributed by tuupola[127]
  - Add definitions for the Interface Association Descriptor of USB for GD32VF103, contributed by michahoiting[128].
  - **IMPORTANT**: hbird SoC is renamed to demosoc, and hbird_eval is renamed to nuclei_fpga_eval
    - Please use SOC=demosoc BOARD=nuclei_fpga_eval to replace SOC=hbird BOARD=hbird_eval
    - The changes are done to not using the name already used in opensource Hummingbird E203 SoC.
    - Now demosoc is used to represent the Nuclei Demo SoC for evaluation on Nuclei FPGA evaluation Board(MCU200T/DDR200T)
- Documentation
  - Update msh application documentation
  - Add basic documentation for **TTGO T-Display-GD32**
  - Add Platformio user guide(written in Chinese) link in get started guide contributed by Maker Young
- Application
  - Increase idle and finsh thread stack for RT-Thread, due to stack size is not enough for RISC-V 64bit

[127] https://github.com/tuupola
[128] https://github.com/michahoiting

  - Set rt-thread example tick hz to 100, and ucosii example tick hz to 50
- Build
  - Format Makefile space to tab
  - Add $(TARGET).dasm into clean targets which are missing before
- Code style
  - Format source files located in application, OS, SoC, test using astyle tool

6.17 V0.3.0

This is official version `0.3.0` of Nuclei SDK.

- SoC
  - Add more newlib stub functions for all SoC support packages
  - Dump extra csr `mdcause` in default exception handler for hbird
  - Add Sipeed Longan Nano as new supported board
  - Add **gd32vf103c_longan_nano** board support, contributed by tuupola[129] and RomanBuchert[130]
- Documentation
  - Add `demo_nice` application documentation
  - Add `msh` application documentation
  - Update get started guide
  - Add **gd32vf103c_longan_nano** board Documentation
  - Update board documentation structure levels
- Application
  - Cleanup unused comments in dhrystone
  - Add new `demo_nice` application to show Nuclei NICE feature
  - Add new `msh` application to show RT-Thread MSH shell component usage
- NMSIS
  - Fix typo in CLICINFO_Type._reserved0 bits
  - Fix `__STRBT`, `__STRHT`, `__STRT` and `__USAT` macros
- OS
  - Add `msh` component source code into RT-Thread RTOS source code
  - Add `rt_hw_console_getchar` implementation
- Build
  - Add `setup.ps1` for setting up environment in windows powershell

[129] https://github.com/tuupola
[130] https://github.com/RomanBuchert

6.18 V0.2.9

This is official version `0.2.9` of Nuclei SDK.

- SoC
  - Remove `ftdi_device_desc "Dual RS232-HS"` line in openocd configuration.

    Note: Newer version of RVSTAR and Hummingbird Debugger have changed the FTDI description from "Dual RS232-HS" to "USB <-> JTAG-DEBUGGER", to be back-compatiable with older version, we just removed this `ftdi_device_desc "Dual RS232-HS"` line. If you want to select specified JTAG, you can add this `ftdi_device_desc` according to your description.

  - Fix typos in **system_<Device>.c**
  - Fix gpio driver implementation bugs of hbird
  - Enable more CSR(micfg_info, mdcfg_info, mcfg_info) show in gdb debug
- Documentation
  - Add more faqs
- Build System
  - Remove unnecessary upload gdb command
  - Remove upload successfully message for `make upload`

6.19 V0.2.8

This is the official release version `0.2.8` of Nuclei SDK.

- SoC
  - Fixed implementation for `_read` newlib stub function, now scanf can be used correctly for both gd32vf103 and hbird SoCs.
- Misc
  - Update platformio package json file according to latest platformio requirements

6.20 V0.2.7

This is the official release version `0.2.7` of Nuclei SDK.

- OS
  - Fix OS portable code, configKERNEL_INTERRUPT_PRIORITY should set to default 0, not 1. 0 is the lowest abs interrupt level.
- Application
  - Fix configKERNEL_INTERRUPT_PRIORITY in FreeRTOSConfig.h to 0
- NMSIS
  - Change timer abs irq level setting in function SysTick_Config from 1 to 0

6.21 V0.2.6

This is the official release version `0.2.6` of Nuclei SDK.

- Application
  - Fix typo in rtthread demo code
  - Update helloworld application to parse vector extension
- NMSIS
  - Update NMSIS DSP and NN library built using NMSIS commit 3d9d40ff
- Documentation
  - Update quick startup nuclei tool setup section
  - Update build system documentation
  - Fix typo in application documentation

6.22 V0.2.5

This is the official release version `0.2.5` of Nuclei SDK.

This following changes are maded since `0.2.5-RC1`.

- SoC
  - For **SOC=hbird**, in function `_premain_init` of `system_hbird.c`, cache will be enable in following cases:
    - If `__ICACHE_PRESENT` is set to 1 in `hbird.h`, I-CACHE will be enabled
    - If `__DCACHE_PRESENT` is set to 1 in `hbird.h`, D-CACHE will be enabled
- Documentation
  - Fix several invalid cross reference links
- NMSIS
  - Update and use NMSIS 1.0.1

6.23 V0.2.5-RC1

This is release `0.2.5-RC1` of Nuclei SDK.

- Documentation
  - Fix invalid links used in this documentation
  - Rename *RVStar* to *RV-STAR* to keep alignment in documentation
- NMSIS
  - Update and use NMSIS 1.0.1-RC1
  - Add NMSIS-DSP and NMSIS-NN library for RISC-V 32bit and 64bit
  - Both RISC-V 32bit and 64bit DSP instructions are supported

- SoC
  - All startup and system init code are adapted to match design changes of NMSIS-1.0.1-RC1
    * *_init* and *_fini* are deprecated for startup code, now please use *_premain_init* and *_postmain_fini* instead
    * Add *DDR* download mode for Hummingbird SoC, which downloaded program into DDR and execute in DDR

6.24 V0.2.4

This is release `0.2.4` of Nuclei SDK.

- Application
  - Upgrade the `demo_dsp` application to a more complicated one, and by default, `DSP_ENABLE` is changed from `OFF` to `ON`, optimization level changed from `O2` to no optimization.
- SoC
  - Update openocd configuration file for Hummingbird FPGA evaluation board, If you want to use `2-wire` mode of JTAG, please change `ftdi_oscan1_mode off` in `openocd_hbird.cfg` to `ftdi_oscan1_mode on`.
  - Add `delay_1ms` function in all supported SoC platforms
  - Fix bugs found in uart and gpio drivers in hbird SoC
  - Move `srodata` after `sdata` for ILM linker script
  - Change bool to BOOL to avoid cpp compiling error in gd32vf103
  - Fix `adc_mode_config` function in gd32vf103 SoC
- Build System
  - Add **GDB_PORT** variable in build system, which is used to specify the gdb port of openocd and gdb when running `run_openocd` and `run_gdb` targets
  - Add Nuclei N/NX/UX 600 series core configurations into *Makefile.core*
  - Add -lstdc++ library for cpp application
  - Generate hex output for dasm target
  - Optimize Makefile to support MACOS

6.25 V0.2.3

This is release `0.2.3` of Nuclei SDK.

- OS
  - Add **RT-Thread 3.1.3** as a new RTOS service of Nuclei SDK, the kernel source code is from RT-Thread Nano project.
  - Update UCOSII source code from version `V2.91` to `V2.93`
  - The source code of UCOSII is fetched from https://github.com/SiliconLabs/uC-OS2/

  - **Warning**: Now for UCOSII application development, the `app_cfg.h`, `os_cfg.h` and `app_hooks.c` are required, which can be also found in https://github.com/SiliconLabs/uC-OS2/tree/master/Cfg/Template

- Application
  - Add **RT-Thread** demo application.
  - Don't use the `get_cpu_freq` function in application code, which currently is only for internal usage, and not all SoC implementations are required to provide this function.
  - Use `SystemCoreClock` to get the CPU frequency instead of using `get_cpu_freq()` in `whetstone` application.
  - Update UCOSII applications due to UCOSII version upgrade, and application development for UCOSII also required little changes, please refer to *UCOSII* (page 88)
  - Fix `time_in_secs` function error in `coremark`, and cleanup `coremark` application.
- Documentation
  - Add documentation about RT-Thread and its application development.
  - Update documentation about UCOSII and its application development.
  - Update `coremark` application documentation.
- Build System
  - Add build system support for RT-Thread support.
  - Build system is updated due to UCOSII version upgrade, the `OS/UCOSII/cfg` folder no longer existed, so no need to include it.
- SoC
  - Update SoC startup and linkscript files to support RT-Thread
- Misc
  - Add `SConscript` file in Nuclei SDK root, this file is used by RT-Thread package.

6.26 V0.2.2

This is release `0.2.2` of Nuclei SDK.

- OS
  - Update UCOSII portable code
  - Now both FreeRTOS and UCOSII are using similar portable code, which both use `SysTimer Interrupt` and `SysTimer Software Interrupt`.
- Documentation
  - Update documentation about RTOS

6.27 V0.2.1

This is release `0.2.1` of Nuclei SDK.

- Build System
  - Add extra linker options `-u _isatty -u _write -u _sbrk -u _read -u _close -u _fstat -u _lseek` in Makefile.conf to make sure if you pass extra `-flto` compile option, link phase will not fail
- Documentation
  - Add documentation about how to optimize for code size in application development, using `demo_eclic` as example.
- OS
  - Update FreeRTOS to version V10.3.1
  - Update FreeRTOS portable code
- NMSIS
  - Update NMSIS to release `v1.0.0-beta1`

6.28 V0.2.0-alpha

This is release `0.2.0-alpha` of Nuclei SDK.

- Documentation
  - Initial verison of Nuclei SDK documentation
  - Update Nuclei-SDK README.md
- Application
  - Add `demo_eclic` application
  - Add `demo_dsp` application
  - `timer_test` application renamed to `demo_timer`
- Build System
  - Add comments for build System
  - Small bug fixes
- **NMSIS**
  - Change `NMSIS/Include` to `NMSIS/Core/Include`
  - Add `NMSIS/DSP` and `NMSIS/NN` header files
  - Add **NMSIS-DSP** and **NMSIS-NN** pre-built libraries

6.29 V0.1.1

This is release `0.1.1` of Nuclei SDK.

Here are the main features of this release:

- Support Windows and Linux development in command line using Make
- Support development using PlatformIO, see https://github.com/Nuclei-Software/platform-nuclei
- Support Humming Bird FPGA evaluation Board and GD32VF103 boards
  - The **Humming Bird FPGA evaluation Board** is used to run evaluation FPGA bitstream of Nuclei N200, N300, N600 and NX600 processor cores
  - The **GD32VF103 boards** are running using a real MCU from Gigadevice which is using Nuclei N200 RISC-V processor core
- Support different download modes flashxip, ilm, flash for our FPGA evaluation board

CHAPTER

SEVEN

FAQ

7.1 Why I can't download application?

- **Case 1: Remote communication error. Target disconnected.: Success.**

```
Nuclei OpenOCD, 64-bit Open On-Chip Debugger 0.10.0+dev-00014-g0eae03214 (2019-12-12-
↪07:43)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
Remote communication error.  Target disconnected.: Success.
"monitor" command not supported by this target.
"monitor" command not supported by this target.
"monitor" command not supported by this target.
You can't do that when your target is ``exec'
"monitor" command not supported by this target.
"monitor" command not supported by this target.
```

Please check whether your driver is installed successfully via replace target `upload` to `run_openocd` as the board user manual described, especially, for **RV-STAR** and **Nuclei Eval SoC Evaluation** boards, For windows, you need to download the **HummingBird Debugger Windows Driver** from https://nucleisys.com/developboard.php, and install it.

If still not working, please check whether your JTAG connection is good or your CPU core is OK.

Note: The USB driver might lost when you re-plug the USB port, you might need to reinstall the driver.

- **Case 2: bfd requires flen 4, but target has flen 0**

```
bfd requires flen 4, but target has flen 0
"monitor" command not supported by this target.
"monitor" command not supported by this target.
"monitor" command not supported by this target.
You can't do that when your target is `exec'
"monitor" command not supported by this target.
"monitor" command not supported by this target.
```

bfd is abbreviation for **Binary File Descriptor**.

This is caused by the target core flen is 0, which means it didn't have float point unit in it, but your program is compiled using flen = 4, single point float unit used, which is incompatible, similar cases such as `bfd requires flen 8, but target has flen 4`

Just change your CORE to proper core settings and will solve this issue.

For example, if you compile your core with `CORE=n300f`, just change it to `CORE=n300`.

- **Case 3: bfd requires xlen 8, but target has xlen 4**

```
bfd requires xlen 8, but target has xlen 4
"monitor" command not supported by this target.
"monitor" command not supported by this target.
"monitor" command not supported by this target.
You can't do that when your target is ``exec'
"monitor" command not supported by this target.
"monitor" command not supported by this target.
```

This issue is caused by the program is a riscv 64 program, but the core is a riscv 32 core, so just change your program to be compiled using a riscv 32 compile option.

For example, if you compile your core with `CORE=ux600`, just change it to `CORE=n300`.

7.2 How to select correct FTDI debugger?

From Nuclei SDK release 0.2.9, the openocd configuration file doesn't contain ftdi_device_desc[131] line by default, so if there are more than one FTDI debuggers which has the same VID/PID(0x0403/0x6010) as Nuclei Debugger Kit use, then you might need to add extra `ftdi device_desc` line in the openocd configuration file to describe the FTDI device description.

Or you can add extra `adapter serial your_serial_no` for your debugger, you can check its serial number via windows FT_PROG tool.

NOTE: for windows, you need to add an extra `A` to the serial number, eg. your serial number is `FT6S9RD6`, then this extra openocd config line should be `adapter serial "FT6S9RD6A"` for windows.

- For **Nuclei FPGA Evaluation Board**, you can check the openocd configuration file in *SoC/evalsoc/Board/nuclei_fpga_eval/openocd_evalsoc.cfg*.
- For **Nuclei RVSTAR Board**, you can check the openocd configuration file in *SoC/gd32vf103/Board/gd32vf103v_rvstar/openocd_gd32vf103.cfg*.

For more details, please check Debug with multiple FTDI devices[132]

7.3 Why I can't download application in Linux?

Please check that whether you have followed the debugger kit manual[133] to setup the USB JTAG drivers correctly. The windows steps and linux steps are different, please take care.

[131] http://openocd.org/doc/html/Debug-Adapter-Configuration.html
[132] https://doc.nucleisys.com/nuclei_studio_supply/27-debug_with_multiple_ftdi_devices/
[133] https://nucleisys.com/developboard.php#ddr200t

7.4 Why the provided application is not running correctly in my Nuclei FPGA Evaluation Board?

Please check the following items:

1. Did you program the correct Nuclei Evaluation FPGA bitstream?
2. Did you re-power the board, when you just programmed the board with FPGA bitstream?
3. Did you choose the right **CORE** as the Nuclei Evaluation FPGA bitstream present?
4. If your application is RTOS demos, did you run in `flashxip` mode, if yes, it is expected due to flash speed is really slow, you'd better try `ilm` or `flash` mode.
5. If still not working, you might need to check whether the FPGA bitstream is correct or not?

7.5 Why ECLIC handler can't be installed using ECLIC_SetVector?

If you are running in `FlashXIP` download mode, it is expected, since the vector table is placed in Flash area which can't be changed during running time.

You can only use this `ECLIC_SetVector` API when your vector table is placed in RAM which can be changed during running time, so if you want to write portable application, we recommended you to use exactly the eclic handler names defined in **startup_<device>.S**.

7.6 Access to github.com is slow, any workaround?

Access speed to github.com sometimes is slow and not stable, but if you want to clone source code, you can also switch to use our mirror site maintained in gitee.com.

This mirror will sync changes from github to gitee every 6 hours, that is 4 times a day.

You just need to replace the github to gitee when you clone any repo in **Nuclei-Software** or **riscv-mcu**.

For example, if you want to clone **nuclei-sdk** using command `git clone https://github.com/Nuclei-Software/nuclei-sdk`, then you can achieve it by command `git clone https://gitee.com/Nuclei-Software/nuclei-sdk`

7.7 `.text' will not fit in region `ilm' or `.bss' will not fit in region `ram'

If you met similar message as below when build an application:

```
xxx/bin/ld: cifar10.elf section `.text' will not fit in region `ilm'
xxx/bin/ld: cifar10.elf section `.bss' will not fit in region `ram'
xxx/bin/ld: section .stack VMA [000000009000f800,000000009000ffff] overlaps section .bss␣
↪VMA [00000000900097c0,00000000900144eb]
xxx/bin/ld: region `ilm' overflowed by 43832 bytes
xxx/bin/ld: region `ram' overflowed by 0 bytes
```

It is caused by the program is too big, our default link script is 64K ILM, 64K DLM, 4M SPIFlash for Nuclei Demo/Eval SoC.

If your core has bigger ILM or DLM, you can change related linker script file according to your choice.

For example, if you want to change linker script for evalsoc on nuclei_fpga_eval ilm download mode: `ILM to 512K, DLM to 256K`, then you can change link script file `SoC/evalsoc/Board/nuclei_fpga_eval/Source/GCC/gcc_evalsoc_ilm.ld` as below:

```
diff --git a/SoC/evalsoc/Board/nuclei_fpga_eval/Source/GCC/gcc_evalsoc_ilm.ld b/SoC/
↪evalsoc/Board/nuclei_fpga_eval/Source/GCC/gcc_evalsoc_ilm.ld
index 1ac5b90..08451b3 100644
--- a/SoC/evalsoc/Board/nuclei_fpga_eval/Source/GCC/gcc_evalsoc_ilm.ld
+++ b/SoC/evalsoc/Board/nuclei_fpga_eval/Source/GCC/gcc_evalsoc_ilm.ld
@@ -28,8 +28,8 @@ ENTRY( _start )
 MEMORY
 {

-  ilm (rxa!w) : ORIGIN = 0x80000000, LENGTH = 64K
-  ram (wxa!r) : ORIGIN = 0x90000000, LENGTH = 64K
+  ilm (rxa!w) : ORIGIN = 0x80000000, LENGTH = 512K
+  ram (wxa!r) : ORIGIN = 0x90000000, LENGTH = 256K
 }
```

7.8 cc1: error: unknown cpu 'nuclei-300-series' for '-mtune'

This *mtune* option is introduced in Nuclei SDK 0.3.5, used to select optimized gcc pipeline model for Nuclei RISC-V Core series such as 200/300/600/900 series, and this feature required Nuclei GNU Toolchain 2022.01, please upgrade to this version or later ones.

7.9 undefined reference to __errno when using libncrt library

When you are using libncrt library, and linked with `-lm`, you may face below issues

```
/home/share/devtools/toolchain/nuclei_gnu/linux64/newlibc/2023.10.14/gcc/bin/../lib/gcc/
↪riscv64-unknown-elf/13.1.1/../../../../riscv64-unknown-elf/bin/ld: /home/share/
↪devtools/toolchain/nuclei_gnu/linux64/newlibc/2023.10.14/gcc/bin/../lib/gcc/riscv64-
↪unknown-elf/13.1.1/../../../../riscv64-unknown-elf/lib/rv32imafdc/ilp32d/libm.a(libm_a-
↪w_exp.o): in function `.L1':
w_exp.c:(.text.exp+0x4a): undefined reference to `__errno'
/home/share/devtools/toolchain/nuclei_gnu/linux64/newlibc/2023.10.14/gcc/bin/../lib/gcc/
↪riscv64-unknown-elf/13.1.1/../../../../riscv64-unknown-elf/bin/ld: /home/share/
↪devtools/toolchain/nuclei_gnu/linux64/newlibc/2023.10.14/gcc/bin/../lib/gcc/riscv64-
↪unknown-elf/13.1.1/../../../../riscv64-unknown-elf/lib/rv32imafdc/ilp32d/libm.a(libm_a-
↪w_exp.o): in function `.L0 ':
w_exp.c:(.text.exp+0x6e): undefined reference to `__errno'
/home/share/devtools/toolchain/nuclei_gnu/linux64/newlibc/2023.10.14/gcc/bin/../lib/gcc/
↪riscv64-unknown-elf/13.1.1/../../../../riscv64-unknown-elf/bin/ld: /home/share/
↪devtools/toolchain/nuclei_gnu/linux64/newlibc/2023.10.14/gcc/bin/../lib/gcc/riscv64-
↪unknown-elf/13.1.1/../../../../riscv64-unknown-elf/lib/rv32imafdc/ilp32d/libm.a(libm_a-
↪w_log.o): in function `log':
w_log.c:(.text.log+0x28): undefined reference to `__errno'
/home/share/devtools/toolchain/nuclei_gnu/linux64/newlibc/2023.10.14/gcc/bin/../lib/gcc/
↪riscv64-unknown-elf/13.1.1/../../../../riscv64-unknown-elf/bin/ld: w_log.c:(.text.
↪log+0x46): undefined reference to `__errno'
```

(continues on next page)

(continued from previous page)

```
/home/share/devtools/toolchain/nuclei_gnu/linux64/newlibc/2023.10.14/gcc/bin/../lib/gcc/
↪riscv64-unknown-elf/13.1.1/../../../../riscv64-unknown-elf/bin/ld: /home/share/
↪devtools/toolchain/nuclei_gnu/linux64/newlibc/2023.10.14/gcc/bin/../lib/gcc/riscv64-
↪unknown-elf/13.1.1/../../../../riscv64-unknown-elf/lib/rv32imafdc/ilp32d/libm.a(libm_a-
↪math_err.o): in function `with_errno':
math_err.c:(.text.with_errno+0x12): undefined reference to `__errno'
collect2: error: ld returned 1 exit status
```

You can fix it by not link `-lm` library, since libncrt library already provided math library feature, so no need to link this math library.

7.10 undefined reference to fclose/sprintf similar API provided in system libraries

From 0.5.0 release, we no longer use `--specs=` option to select library we want to use, and we also passed `-nodefaultlibs` options to not use standard system libraries, this changes are made to support both gcc and clang toolchain, so in Nuclei SDK build system, we control the needed system libraries to be linked as required by `STDCLIB` make variable, for details, please check `Build/toolchain/*.mk` makefiles, and also we use linker's group libraries feature `--start-group archives --end-group` to repeatly search undefined reference in the group libraries, but this feature is not enabled in Eclipse CDT based IDE like Nuclei Studio, which undefined reference is searched in the order of library specified on the command line, so you may meet issue like undefined fclose reference even you linked newlib nano c library `-lc_nano` if the library order is not good, so to fix this issue, you may need to place the library in a good order and need to repeatly link it, such as `-lgcc -lc_nano -lm -lsemihost -lgcov -lgcc -lc_nano`, and also we have opened an issue to track it, see https://github.com/eclipse-embed-cdt/eclipse-plugins/issues/592

7.11 fatal error: rvintrin.h: No such file or directory

If you are using Nuclei Toolchain 2023.10, `rvintrin.h` no longer exist for B extension, please don't include this header file. If you want to use an intrinsic API for B extension, you need to write using c asm intrinsic.

7.12 riscv-nuclei-elf-gcc: not found when using Nuclei Studio 2023.10

`riscv-nuclei-elf-gcc` (gcc10) has changed to `riscv64-unknown-elf-gcc` (gcc13) since Nuclei Studio 2023.10 or Nuclei RISC-V Toolchain 2023.10, so if you are using older toolchain created npk package or ide project, you may face this build fail issue, you can follow the user guide of Nuclei Studio 2023.10 to fix this issue, see chapter 8.

CHAPTER

EIGHT

LICENSE

```
                              Apache License
                        Version 2.0, January 2004
                     http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

   "License" shall mean the terms and conditions for use, reproduction,
   and distribution as defined by Sections 1 through 9 of this document.

   "Licensor" shall mean the copyright owner or entity authorized by
   the copyright owner that is granting the License.

   "Legal Entity" shall mean the union of the acting entity and all
   other entities that control, are controlled by, or are under common
   control with that entity. For the purposes of this definition,
   "control" means (i) the power, direct or indirect, to cause the
   direction or management of such entity, whether by contract or
   otherwise, or (ii) ownership of fifty percent (50%) or more of the
   outstanding shares, or (iii) beneficial ownership of such entity.

   "You" (or "Your") shall mean an individual or Legal Entity
   exercising permissions granted by this License.

   "Source" form shall mean the preferred form for making modifications,
   including but not limited to software source code, documentation
   source, and configuration files.

   "Object" form shall mean any form resulting from mechanical
   transformation or translation of a Source form, including but
   not limited to compiled object code, generated documentation,
   and conversions to other media types.

   "Work" shall mean the work of authorship, whether in Source or
   Object form, made available under the License, as indicated by a
   copyright notice that is included in or attached to the work
   (an example is provided in the Appendix below).

   "Derivative Works" shall mean any work, whether in Source or Object
```

(continues on next page)

(continued from previous page)

```
   form, that is based on (or derived from) the Work and for which the
   editorial revisions, annotations, elaborations, or other modifications
   represent, as a whole, an original work of authorship. For the purposes
   of this License, Derivative Works shall not include works that remain
   separable from, or merely link (or bind by name) to the interfaces of,
   the Work and Derivative Works thereof.

   "Contribution" shall mean any work of authorship, including
   the original version of the Work and any modifications or additions
   to that Work or Derivative Works thereof, that is intentionally
   submitted to Licensor for inclusion in the Work by the copyright owner
   or by an individual or Legal Entity authorized to submit on behalf of
   the copyright owner. For the purposes of this definition, "submitted"
   means any form of electronic, verbal, or written communication sent
   to the Licensor or its representatives, including but not limited to
   communication on electronic mailing lists, source code control systems,
   and issue tracking systems that are managed by, or on behalf of, the
   Licensor for the purpose of discussing and improving the Work, but
   excluding communication that is conspicuously marked or otherwise
   designated in writing by the copyright owner as "Not a Contribution."

   "Contributor" shall mean Licensor and any individual or Legal Entity
   on behalf of whom a Contribution has been received by Licensor and
   subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of
   this License, each Contributor hereby grants to You a perpetual,
   worldwide, non-exclusive, no-charge, royalty-free, irrevocable
   copyright license to reproduce, prepare Derivative Works of,
   publicly display, publicly perform, sublicense, and distribute the
   Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of
   this License, each Contributor hereby grants to You a perpetual,
   worldwide, non-exclusive, no-charge, royalty-free, irrevocable
   (except as stated in this section) patent license to make, have made,
   use, offer to sell, sell, import, and otherwise transfer the Work,
   where such license applies only to those patent claims licensable
   by such Contributor that are necessarily infringed by their
   Contribution(s) alone or by combination of their Contribution(s)
   with the Work to which such Contribution(s) was submitted. If You
   institute patent litigation against any entity (including a
   cross-claim or counterclaim in a lawsuit) alleging that the Work
   or a Contribution incorporated within the Work constitutes direct
   or contributory patent infringement, then any patent licenses
   granted to You under this License for that Work shall terminate
   as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the
   Work or Derivative Works thereof in any medium, with or without
   modifications, and in Source or Object form, provided that You
   meet the following conditions:
```

(continues on next page)

(continued from previous page)

```
   (a) You must give any other recipients of the Work or
       Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices
       stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works
       that You distribute, all copyright, patent, trademark, and
       attribution notices from the Source form of the Work,
       excluding those notices that do not pertain to any part of
       the Derivative Works; and

   (d) If the Work includes a "NOTICE" text file as part of its
       distribution, then any Derivative Works that You distribute must
       include a readable copy of the attribution notices contained
       within such NOTICE file, excluding those notices that do not
       pertain to any part of the Derivative Works, in at least one
       of the following places: within a NOTICE text file distributed
       as part of the Derivative Works; within the Source form or
       documentation, if provided along with the Derivative Works; or,
       within a display generated by the Derivative Works, if and
       wherever such third-party notices normally appear. The contents
       of the NOTICE file are for informational purposes only and
       do not modify the License. You may add Your own attribution
       notices within Derivative Works that You distribute, alongside
       or as an addendum to the NOTICE text from the Work, provided
       that such additional attribution notices cannot be construed
       as modifying the License.

   You may add Your own copyright statement to Your modifications and
   may provide additional or different license terms and conditions
   for use, reproduction, or distribution of Your modifications, or
   for any such Derivative Works as a whole, provided Your use,
   reproduction, and distribution of the Work otherwise complies with
   the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,
   any Contribution intentionally submitted for inclusion in the Work
   by You to the Licensor shall be under the terms and conditions of
   this License, without any additional terms or conditions.
   Notwithstanding the above, nothing herein shall supersede or modify
   the terms of any separate license agreement you may have executed
   with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade
   names, trademarks, service marks, or product names of the Licensor,
   except as required for reasonable and customary use in describing the
   origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
   agreed to in writing, Licensor provides the Work (and each
```

(continues on next page)

(continued from previous page)

```
   Contributor provides its Contributions) on an "AS IS" BASIS,
   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
   implied, including, without limitation, any warranties or conditions
   of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
   PARTICULAR PURPOSE. You are solely responsible for determining the
   appropriateness of using or redistributing the Work and assume any
   risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
   whether in tort (including negligence), contract, or otherwise,
   unless required by applicable law (such as deliberate and grossly
   negligent acts) or agreed to in writing, shall any Contributor be
   liable to You for damages, including any direct, indirect, special,
   incidental, or consequential damages of any character arising as a
   result of this License or out of the use or inability to use the
   Work (including but not limited to damages for loss of goodwill,
   work stoppage, computer failure or malfunction, or any and all
   other commercial damages or losses), even if such Contributor
   has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
   the Work or Derivative Works thereof, You may choose to offer,
   and charge a fee for, acceptance of support, warranty, indemnity,
   or other liability obligations and/or rights consistent with this
   License. However, in accepting such obligations, You may act only
   on Your own behalf and on Your sole responsibility, not on behalf
   of any other Contributor, and only if You agree to indemnify,
   defend, and hold each Contributor harmless for any liability
   incurred by, or claims asserted against, such Contributor by reason
   of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

   To apply the Apache License to your work, attach the following
   boilerplate notice, with the fields enclosed by brackets "[]"
   replaced with your own identifying information. (Don't include
   the brackets!)  The text should be enclosed in the appropriate
   comment syntax for the file format. We also recommend that a
   file or class name and description of purpose be included on the
   same "printed page" as the copyright notice for easier
   identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0
```

(continues on next page)

(continued from previous page)

```
Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
```

CHAPTER

NINE

GLOSSARY

API
: (Application Program Interface) A defined set of routines and protocols for building application software.

DSP
: (Digital Signal Processing) is the use of digital processing, such as by computers or more specialized digital signal processors, to perform a wide variety of signal processing operations.

ISR
: (Interrupt Service Routine) Also known as an interrupt handler, an ISR is a callback function whose execution is triggered by a hardware interrupt (or software interrupt instructions) and is used to handle high-priority conditions that require interrupting the current code executing on the processor.

NN
: (Neural Network) is a network or circuit of neurons, or in a modern sense, an artificial neural network, composed of artificial neurons or nodes.

XIP
: (eXecute In Place) a method of executing programs directly from long term storage rather than copying it into RAM, saving writable memory for dynamic data and not the static program code.

CHAPTER

TEN

APPENDIX

- **Nuclei Tools and Documents**: https://nucleisys.com/download.php
- **Nuclei Software Opensource Organization**: https://github.com/Nuclei-Software
- **RISC-V MCU Opensource Organization**: https://github.com/riscv-mcu
- **Nuclei Toolchain Repo**: https://github.com/riscv-mcu/riscv-gnu-toolchain
- **Nuclei OpenOCD Repo**: https://github.com/riscv-mcu/riscv-openocd
- **Nuclei QEMU Repo**: https://github.com/riscv-mcu/qemu
- **Nuclei SDK**: https://github.com/Nuclei-Software/nuclei-sdk
- **NMSIS**: https://github.com/Nuclei-Software/NMSIS
- **Nuclei AI Library**: https://github.com/Nuclei-Software/nuclei-ai-library
- **Nuclei RISC-V IP Products**: https://www.nucleisys.com/product.php
- **Nuclei Tools Documentation**: https://doc.nucleisys.com/nuclei_tools
- **Nuclei Studio Supply Documents**: https://github.com/Nuclei-Software/nuclei-studio
- **RISC-V MCU Community Website**: https://www.riscv-mcu.com/
- **Nuclei RISC-V CPU Spec**: https://doc.nucleisys.com/nuclei_spec
- **RISC-V ISA Specfications(Ratified)**: https://riscv.org/technical/specifications
- **RISC-V ISA Specification(Latest)**: https://github.com/riscv/riscv-isa-manual/releases
- **RISC-V Architecture Profiles**: https://github.com/riscv/riscv-profiles
- **RISC-V Bitmanip(B) Extension Spec**: https://github.com/riscv/riscv-bitmanip
- **RISC-V Packed SIMD(P) Extension Spec**: https://github.com/riscv/riscv-p-spec
- **RISC-V Cryptography(K) Extension Spec**: https://github.com/riscv/riscv-crypto
- **RISC-V Vector(V) Extension Spec**: https://github.com/riscv/riscv-v-spec
- **RISC-V Vector Intrinsic API Spec**: https://github.com/riscv-non-isa/rvv-intrinsic-doc
- **RISC-V ISA Extension Spec Status**: https://wiki.riscv.org/display/HOME/Specification+Status
- **Nuclei Bumblebee Core Document**: https://github.com/nucleisys/Bumblebee_Core_Doc

CHAPTER

ELEVEN

INDICES AND TABLES

- genindex
- search

INDEX