Nuclei SDK

Nuclei embedded Software Development Kit

Nuclei SDK

Release 0.3.5-dev

Nuclei

Oct 18, 2021

CONTENTS:

CHAPTER

ONE

OVERVIEW

1.1 Introduction

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 Demo SoC* (page 51) and *GD32VF103 SoC* (page 53)
- Provided realtime operation system service via *FreeRTOS* (page 67), *UCOSII* (page 68) and *RT-Thread* (page 69)
- Provided bare-metal service for embedded system software beginners and resource-limited use-cases.
- Currently Nuclei SDK didn't define any common device APIs to access GPIO/I2C/SPI/UART devices, it still relied on the device/peripheral APIs from firmware libraries from various silicon vendors, such as current supported *GD32VF103 SoC* (page 53).
- Applications are logically seperated into three parts:

[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

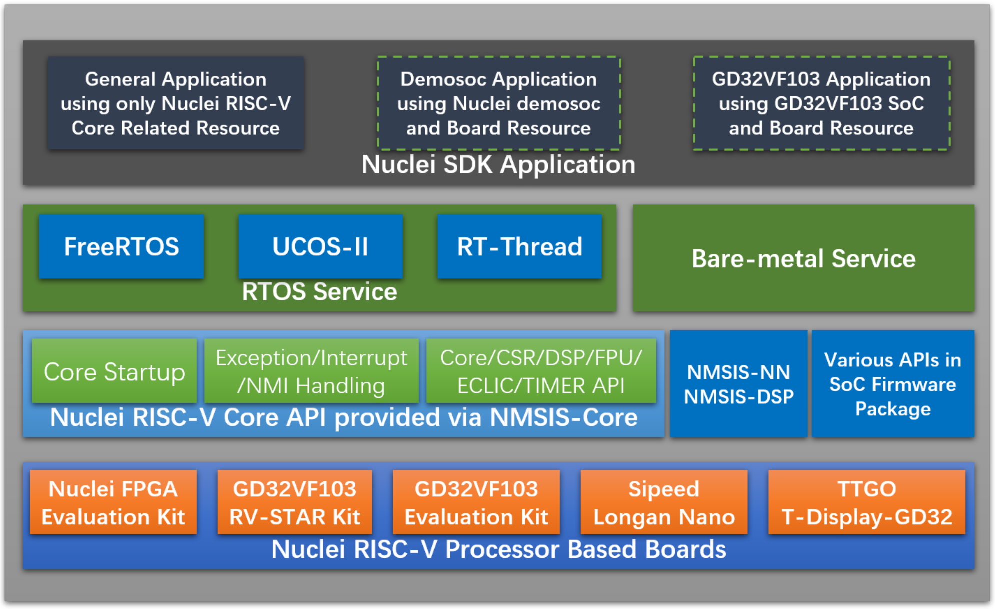

General Application using only Nuclei RISC-V Core Related Resource
Demosoc Application using Nuclei demosoc and Board Resource
GD32VF103 Application using GD32VF103 SoC and Board Resource
Nuclei SDK Application
FreeRTOS
UCOS-II
RT-Thread
RTOS Service
Bare-metal Service
Core Startup
Exception/Interrupt /NMI Handling
Core/CSR/DSP/FPU/ ECLIC/TIMER API
Nuclei RISC-V Core API provided via NMSIS-Core
NMSIS-NN NMSIS-DSP
Various APIs in SoC Firmware Package
Nuclei FPGA Evaluation Kit
GD32VF103 RV-STAR Kit
GD32VF103 Evaluation Kit
Sipeed Longan Nano
TTGO T-Display-GD32
Nuclei RISC-V Processor Based Boards

Fig. 1: Nuclei SDK Design and Architecture Diagram

- **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 Demo SoC applications**: These applications are not included in the Nuclei SDK software code, it is *maintained seperately*, it will use resource from Nuclei Demo SoC and its evaluation boards to develop applications, which will not be compatiable with different boards.
- **GD32VF103 SoC applications**: These applications are not included in the Nuclei SDK software code, it is *maintained seperately*, it will use resource from GD32VF103 SoC and its evaluation boards to develop applications, which will not be compatiable 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 39) section.

1.5 Copyright

1.6 License

Nuclei SDK is an opensource project licensed by *Apache License 2.0* (page 101).

CHAPTER

TWO

QUICK STARTUP

2.1 Use Nuclei SDK in Nuclei Studio

From **2020.08** release version of Nuclei Studio IDE, the nuclei-sdk **released** version will be deeply integrated with Nuclei Studio, and you can directly create nuclei-sdk project in Nuclei Studio IDE.

You can download **Nuclei Studio IDE** from Nuclei Download Center[5], and follow Nuclei_Studio_User_Guide.pdf[6] 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.

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, please check *Use Prebuilt Tools in Nuclei Studio* (page 5).

If you want to use latest toolchain, you can follow guides below:

- For Windows users, please check *Install and Setup Tools in Windows* (page 6)
- For Linux users, please check *Install and Setup Tools in Linux* (page 7)

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[7] 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_202010`, then you can find the prebuilt tools in `D:\Software\NucleiStudio_IDE_202010\NucleiStudio\toolchain`.
- In Linux, if you have extracted the Nuclei Studio IDE to `/home/labdev/NucleiStudio_IDE_202010`, then you can find the prebuilt tools in `/home/labdev/NucleiStudio_IDE_202010/NucleiStudio/toolchain`.

[5] https://nucleisys.com/download.php
[6] https://www.nucleisys.com/upload/files/doc/nucleistudio/Nuclei_Studio_User_Guide.pdf
[7] https://nucleisys.com/download.php

If you have downloaded and extracted the Nuclei Studio, then you can jump to *Get and Setup Nuclei SDK* (page 8).

2.2.2 Install and Setup Tools in Windows

Make sure you are using at least **Windows 7**, and then you can follow the following steps to download and install tools for you.

1. Create an `Nuclei` folder in your Windows Environment, such as `D:\Software\Nuclei`
2. Download the following tools from Nuclei Download Center[8], please check and follow the figure *Nuclei Tools need to be downloaded for Windows* (page 6).
   - **Nuclei RISC-V GNU Toolchain for Windows**, see number **1** in the figure *Nuclei Tools need to be downloaded for Windows* (page 6)
   - **Nuclei OpenOCD for Windows**, see number **2** in the figure *Nuclei Tools need to be downloaded for Windows* (page 6)
   - **Windows Build Tools**, see number **3** in the figure *Nuclei Tools need to be downloaded for Windows* (page 6)

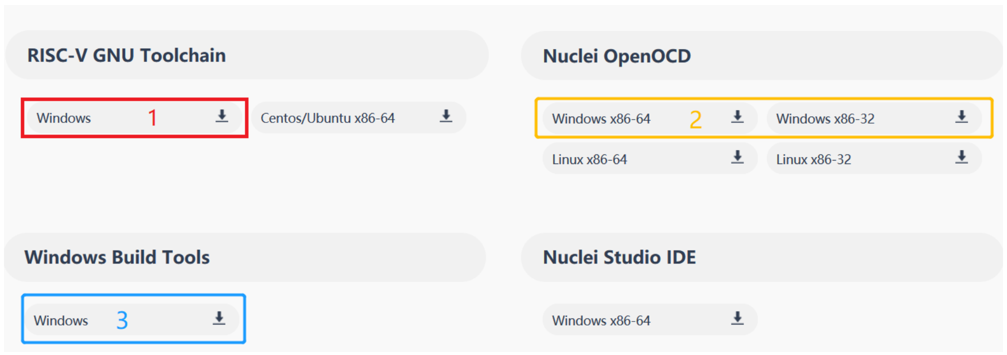
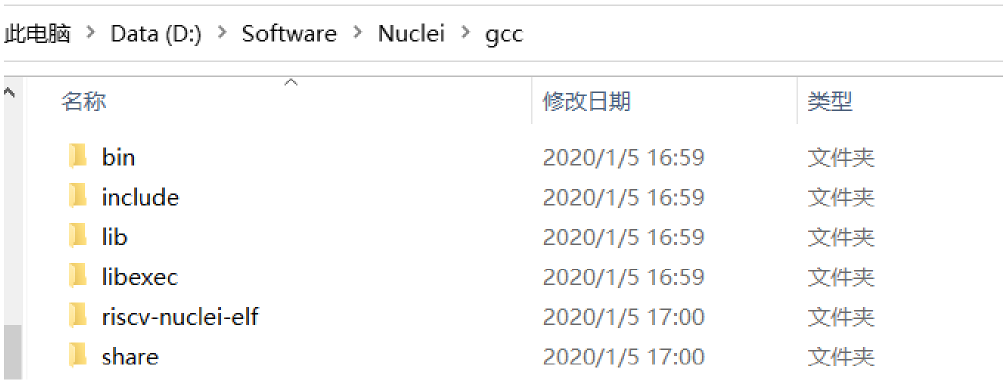


Fig. 1: Nuclei Tools need to be downloaded for Windows

3. Setup tools in previously created `Nuclei` folder, create `gcc`, `openocd` and `build-tools` folders.
   - **Nuclei RISC-V GNU Toolchain for Windows** Extract the download **gnu toolchain** into a temp folder, and copy the files into `gcc` folder, make sure the `gcc` directory structure looks like this figure *Nuclei RISC-V GCC Toolchain directory structure of gcc* (page 6)



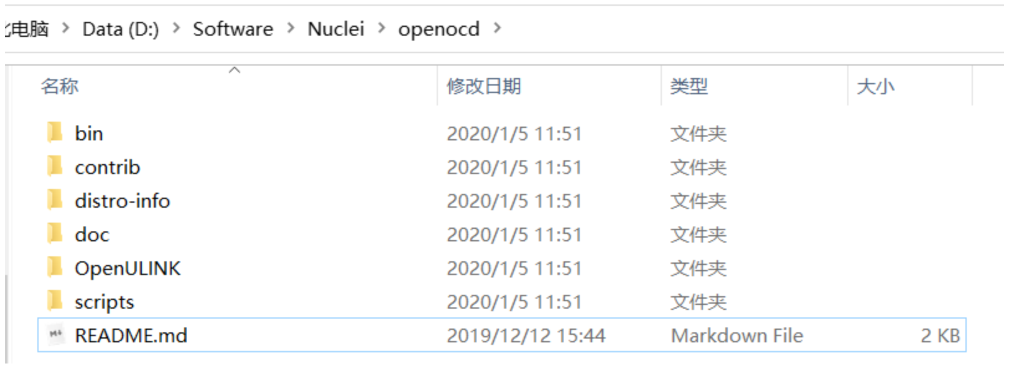
此电脑 > Data (D:) > Software > Nuclei > gcc

名称	修改日期	类型
bin	2020/1/5 16:59	文件夹
include	2020/1/5 16:59	文件夹
lib	2020/1/5 16:59	文件夹
libexec	2020/1/5 16:59	文件夹
riscv-nuclei-elf	2020/1/5 17:00	文件夹
share	2020/1/5 17:00	文件夹

Fig. 2: Nuclei RISC-V GCC Toolchain directory structure of gcc

[8] https://nucleisys.com/download.php

- **Nuclei OpenOCD for Windows** Extract the download **openocd** tool into a temp folder, and copy the files into `openocd` folder, make sure the `openocd` directory structure looks like this figure *Nuclei OpenOCD directory structure of openocd* (page 7)

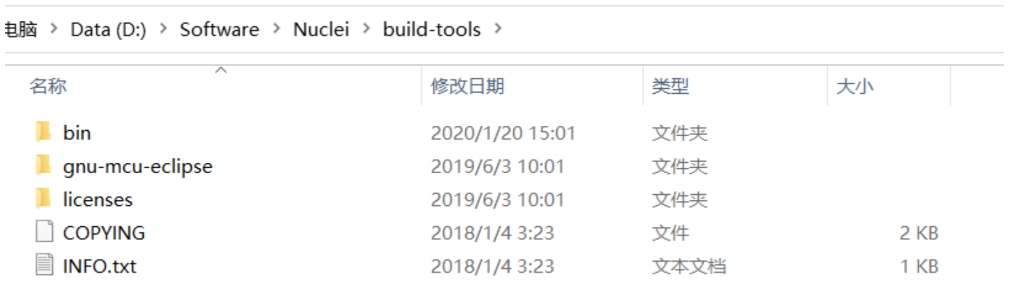


此电脑 > Data (D:) > Software > Nuclei > openocd >

名称	修改日期	类型	大小
bin	2020/1/5 11:51	文件夹	
contrib	2020/1/5 11:51	文件夹	
distro-info	2020/1/5 11:51	文件夹	
doc	2020/1/5 11:51	文件夹	
OpenULINK	2020/1/5 11:51	文件夹	
scripts	2020/1/5 11:51	文件夹	
README.md	2019/12/12 15:44	Markdown File	2 KB

Fig. 3: Nuclei OpenOCD directory structure of openocd

- **Windows Build Tools** Extract the download **build-tools** tool into a temp folder, and copy the files into `build-tools` folder, make sure the `build-tools` directory structure looks like this figure *Nuclei Windows Build Tools directory structure of build-tools* (page 7)



电脑 > Data (D:) > Software > Nuclei > build-tools >

名称	修改日期	类型	大小
bin	2020/1/20 15:01	文件夹	
gnu-mcu-eclipse	2019/6/3 10:01	文件夹	
licenses	2019/6/3 10:01	文件夹	
COPYING	2018/1/4 3:23	文件	2 KB
INFO.txt	2018/1/4 3:23	文本文档	1 KB

Fig. 4: Nuclei Windows Build Tools directory structure of build-tools

If you have setuped the prebuilt tools in Windows, then you can jump to *Get and Setup Nuclei SDK* (page 8).

2.2.3 Install and Setup Tools in Linux

Make sure you are using **Centos or Ubuntu 64 bit**, and then you can follow the following steps to download and install tools for you.

1. Create an `Nuclei` folder in your Linux Environment, such as `~/Software/Nuclei`
2. Download the following tools from Nuclei Download Center[9], please check and follow the figure *Nuclei Tools need to be downloaded for Linux* (page 8).
   - **Nuclei RISC-V GNU Toolchain for Linux**, for **CentOS or Ubuntu** click number **1** in the figure *Nuclei Tools need to be downloaded for Linux* (page 8)
   - **Nuclei OpenOCD for Linux**, see number **2-1** for 64bit version in the figure *Nuclei Tools need to be downloaded for Linux* (page 8)
   - **Make >= 3.82**: Install `Make` using `sudo apt-get install make` in Ubuntu, or `sudo yum install make` in CentOS.

[9] https://nucleisys.com/download.php

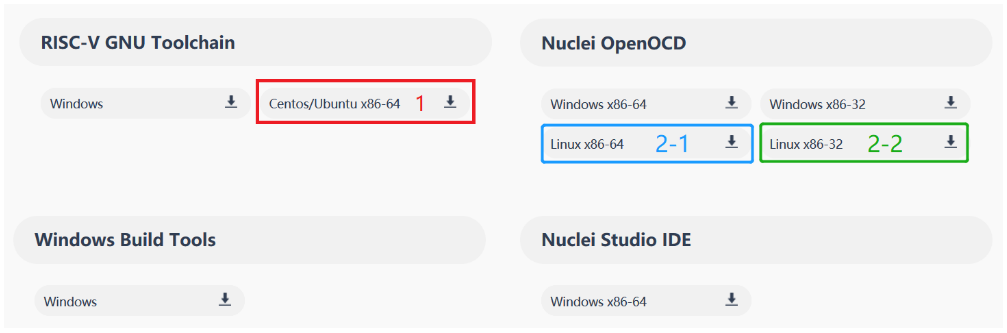


Fig. 5: Nuclei Tools need to be downloaded for Linux

3. Setup tools in previously created `Nuclei` folder, create `gcc` and `openocd` folders. Please follow similar steps described in **Step 3** in *Install and Setup Tools in Windows* (page 6) to extract and copy necessary files.

Note:

- Only `gcc` and `openocd` are required for Linux.
- Extract the downloaded Linux tools, not the windows version.

If you have setuped the prebuilt tools in Linux, then you can jump to *Get and Setup Nuclei SDK* (page 8).

2.3 Get and Setup Nuclei SDK

The source code of Nuclei SDK is maintained in Github[10] and Gitee[11].

- We mainly maintained github version, and gitee version is mirrored, just for fast access in China.
- Check source code in Nuclei SDK in Github[12] or Nuclei SDK in Gitee[13] 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[14].

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.

[10] https://github.com
[11] https://gitee.com
[12] https://github.com/Nuclei-Software/nuclei-sdk
[13] https://gitee.com/Nuclei-Software/nuclei-sdk
[14] 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
  - **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 5), and prebuilt tools located in `D:\Software\NucleiStudio_IDE_202010\NucleiStudio\toolchain`, otherwise please use your correct tool root path.

    ```
    set NUCLEI_TOOL_ROOT=D:\Software\NucleiStudio_IDE_202010\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_202010\NucleiStudio\toolchain`:

    ```
    $NUCLEI_TOOL_ROOT="D:\Software\NucleiStudio_IDE_202009\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 5) and prebuilt tools located in `/home/labdev/NucleiStudio_IDE_202010/NucleiStudio/toolchain`, otherwise please use your correct tool root path.

    ```
    NUCLEI_TOOL_ROOT=/home/labdev/NucleiStudio_IDE_202010/NucleiStudio/
    ↪toolchain
    ```

2.4 Build, Run and Debug Sample Application

Assume you have followed steps in *Get and Setup Nuclei SDK* (page 8) to clone source code and create files below:

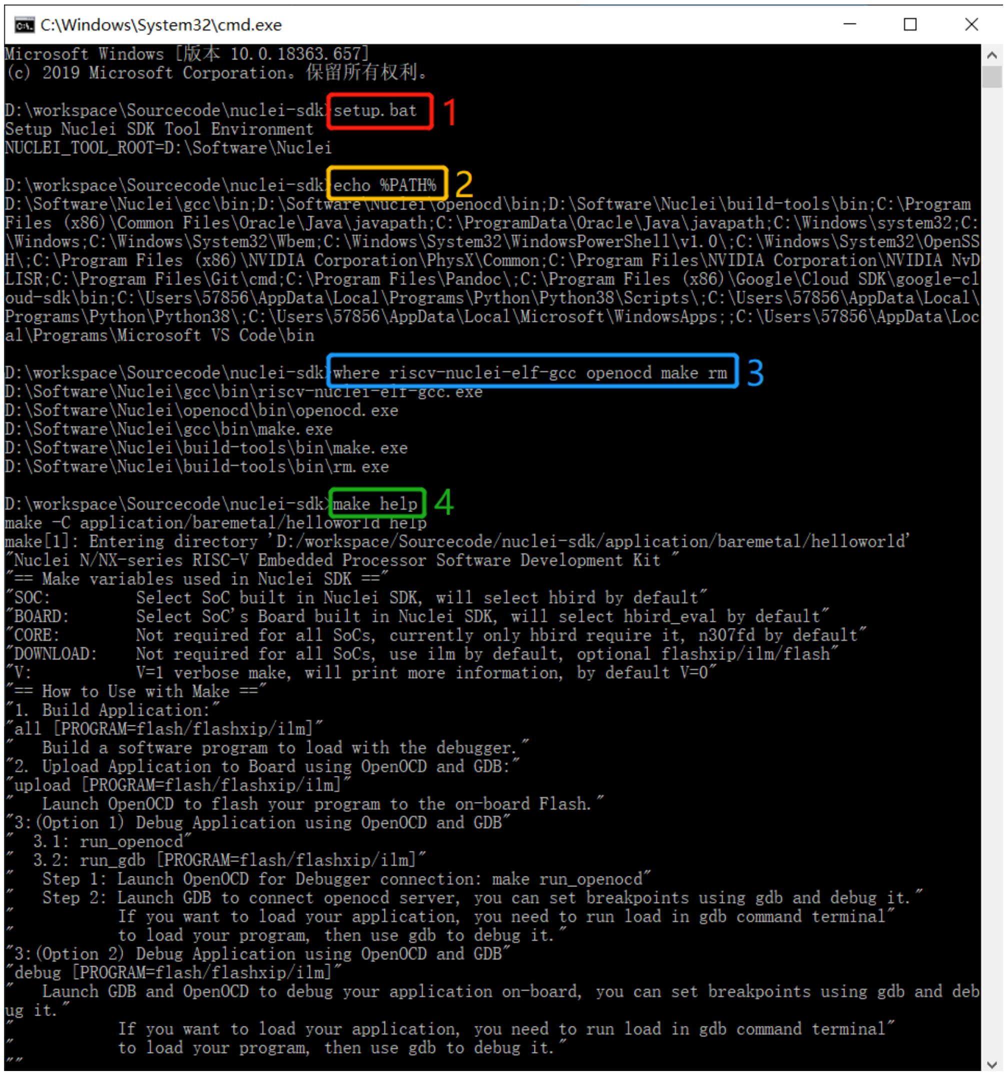
- `setup_config.bat` for run in **Windows Command Prompt** terminal
- `setup_config.ps1` for run in **Windows PowerShell** terminal
- `setup_config.sh` for run in **Linux Bash** terminal

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 10):

```
setup.bat
echo %PATH%
where riscv-nuclei-elf-gcc openocd make rm
make help
```

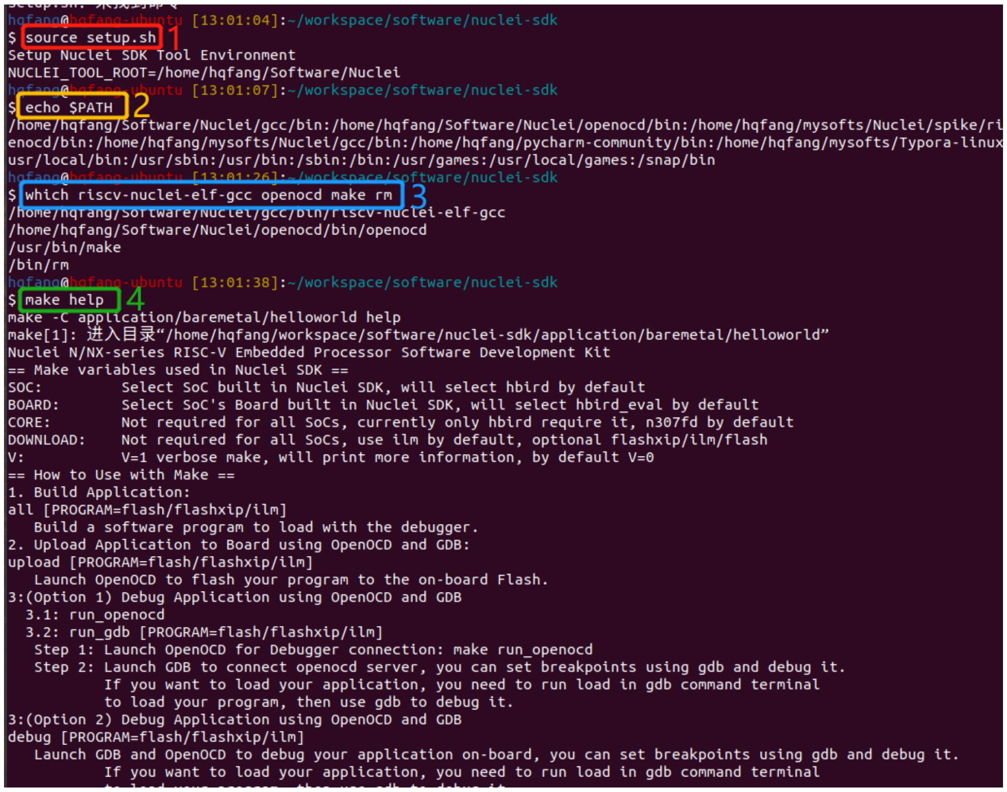
- For **Linux** users, you can open **Linux Bash** terminal and cd to `nuclei-sdk` folder, then run the following commands to setup build environment for Nuclei SDK, the output will be similar as this screenshot *Setup Build Environment for Nuclei SDK in Linux Bash* (page 11):



```
Microsoft Windows [版本 10.0.18363.657]
(c) 2019 Microsoft Corporation。保留所有权利。

D:\workspace\Sourcecode\nuclei-sdk>setup.bat
Setup Nuclei SDK Tool Environment
NUCLEI_TOOL_ROOT=D:\Software\Nuclei

D:\workspace\Sourcecode\nuclei-sdk>echo %PATH%
D:\Software\Nuclei\gcc\bin;D:\Software\Nuclei\openocd\bin;D:\Software\Nuclei\build-tools\bin;C:\Program
Files (x86)\Common Files\Oracle\Java\javapath;C:\ProgramData\Oracle\Java\javapath;C:\Windows\system32;C:
\Windows;C:\Windows\System32\Wbem;C:\Windows\System32\WindowsPowerShell\v1.0\;C:\Windows\System32\OpenSS
H\;C:\Program Files (x86)\NVIDIA Corporation\PhysX\Common;C:\Program Files\NVIDIA Corporation\NVIDIA NvD
LISR;C:\Program Files\Git\cmd;C:\Program Files\Pandoc\;C:\Program Files (x86)\Google\Cloud SDK\google-cl
oud-sdk\bin;C:\Users\57856\AppData\Local\Programs\Python\Python38\Scripts\;C:\Users\57856\AppData\Local\
Programs\Python\Python38\;C:\Users\57856\AppData\Local\Microsoft\WindowsApps;;C:\Users\57856\AppData\Loc
al\Programs\Microsoft VS Code\bin

D:\workspace\Sourcecode\nuclei-sdk>where riscv-nuclei-elf-gcc openocd make rm
D:\Software\Nuclei\gcc\bin\riscv-nuclei-elf-gcc.exe
D:\Software\Nuclei\openocd\bin\openocd.exe
D:\Software\Nuclei\gcc\bin\make.exe
D:\Software\Nuclei\build-tools\bin\make.exe
D:\Software\Nuclei\build-tools\bin\rm.exe

D:\workspace\Sourcecode\nuclei-sdk>make help
make -C application/baremetal/helloworld help
make[1]: Entering directory 'D:/workspace/Sourcecode/nuclei-sdk/application/baremetal/helloworld'
"Nuclei N/NX-series RISC-V Embedded Processor Software Development Kit "
"== Make variables used in Nuclei SDK =="
"SOC:         Select SoC built in Nuclei SDK, will select hbird by default"
"BOARD:       Select SoC's Board built in Nuclei SDK, will select hbird_eval by default"
"CORE:        Not required for all SoCs, currently only hbird require it, n307fd by default"
"DOWNLOAD:    Not required for all SoCs, use ilm by default, optional flashxip/ilm/flash"
"V:           V=1 verbose make, will print more information, by default V=0"
"== How to Use with Make =="
"1. Build Application:"
"all [PROGRAM=flash/flashxip/ilm]"
"   Build a software program to load with the debugger."
"2. Upload Application to Board using OpenOCD and GDB:"
"upload [PROGRAM=flash/flashxip/ilm]"
"   Launch OpenOCD to flash your program to the on-board Flash."
"3:(Option 1) Debug Application using OpenOCD and GDB"
"  3.1: run_openocd"
"  3.2: run_gdb [PROGRAM=flash/flashxip/ilm]"
"   Step 1: Launch OpenOCD for Debugger connection: make run_openocd"
"   Step 2: Launch GDB to connect openocd server, you can set breakpoints using gdb and debug it."
"           If you want to load your application, you need to run load in gdb command terminal"
"           to load your program, then use gdb to debug it."
"3:(Option 2) Debug Application using OpenOCD and GDB"
"debug [PROGRAM=flash/flashxip/ilm]"
"   Launch GDB and OpenOCD to debug your application on-board, you can set breakpoints using gdb and deb
ug it."
"           If you want to load your application, you need to run load in gdb command terminal"
"           to load your program, then use gdb to debug it."
""
```

Fig. 6: Setup Build Environment for Nuclei SDK in Windows Command Prompt

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



```
hqfang@hqfang-ubuntu [13:01:04]:~/workspace/software/nuclei-sdk
$ source setup.sh
Setup Nuclei SDK Tool Environment
NUCLEI_TOOL_ROOT=/home/hqfang/Software/Nuclei
hqfang@hqfang-ubuntu [13:01:07]:~/workspace/software/nuclei-sdk
$ echo $PATH
/home/hqfang/Software/Nuclei/gcc/bin:/home/hqfang/Software/Nuclei/openocd/bin:/home/hqfang/mysofts/Nuclei/spike/ri
enocd/bin:/home/hqfang/mysofts/Nuclei/gcc/bin:/home/hqfang/pycharm-community/bin:/home/hqfang/mysofts/Typora-linux
usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
hqfang@hqfang-ubuntu [13:01:26]:~/workspace/software/nuclei-sdk
$ which riscv-nuclei-elf-gcc openocd make rm
/home/hqfang/Software/Nuclei/gcc/bin/riscv-nuclei-elf-gcc
/home/hqfang/Software/Nuclei/openocd/bin/openocd
/usr/bin/make
/bin/rm
hqfang@hqfang-ubuntu [13:01:38]:~/workspace/software/nuclei-sdk
$ make help
make -C application/baremetal/helloworld help
make[1]: 进入目录“/home/hqfang/workspace/software/nuclei-sdk/application/baremetal/helloworld”
Nuclei N/NX-series RISC-V Embedded Processor Software Development Kit
== Make variables used in Nuclei SDK ==
SOC:         Select SoC built in Nuclei SDK, will select hbird by default
BOARD:       Select SoC's Board built in Nuclei SDK, will select hbird_eval by default
CORE:        Not required for all SoCs, currently only hbird require it, n307fd by default
DOWNLOAD:    Not required for all SoCs, use ilm by default, optional flashxip/ilm/flash
V:           V=1 verbose make, will print more information, by default V=0
== How to Use with Make ==
1. Build Application:
all [PROGRAM=flash/flashxip/ilm]
   Build a software program to load with the debugger.
2. Upload Application to Board using OpenOCD and GDB:
upload [PROGRAM=flash/flashxip/ilm]
   Launch OpenOCD to flash your program to the on-board Flash.
3:(Option 1) Debug Application using OpenOCD and GDB
  3.1: run_openocd
  3.2: run_gdb [PROGRAM=flash/flashxip/ilm]
   Step 1: Launch OpenOCD for Debugger connection: make run_openocd
   Step 2: Launch GDB to connect openocd server, you can set breakpoints using gdb and debug it.
           If you want to load your application, you need to run load in gdb command terminal
           to load your program, then use gdb to debug it.
3:(Option 2) Debug Application using OpenOCD and GDB
debug [PROGRAM=flash/flashxip/ilm]
   Launch GDB and OpenOCD to debug your application on-board, you can set breakpoints using gdb and debug it.
           If you want to load your application, you need to run load in gdb command terminal
```

Fig. 7: Setup Build Environment for Nuclei SDK in Linux Bash

Note:

- 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 `riscv-nuclei-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 8).

Here for a quick startup, this guide will take board *GD32VF103V RV-STAR Kit* (page 59) 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 55) 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 59) 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[15] or Minicom in Linux[16], 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 25) and *BOARD* (page 26) make variable. eg. If you SoC is `demosoc` and Board is `nuclei_fpga_eval`, just pass `SOC=demosoc BOARD=nuclei_fpga_eval` to make instead of the one mentioned below. If your default board for this `demosoc` is `nuclei_fpga_eval`, then you don't need to pass `BOARD=nuclei_fpga_eval`.

2.4.2 Build Application

We need to build application for this board *GD32VF103V RV-STAR Kit* (page 59) using this command line:

```
make SOC=gd32vf103 BOARD=gd32vf103v_rvstar all
```

Here is the sample output of this command:

```
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
"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
```

(continues on next page)

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

(continued from previous page)

```
"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 59), then you can run it using this command line:

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

Here is the sample output of this command:

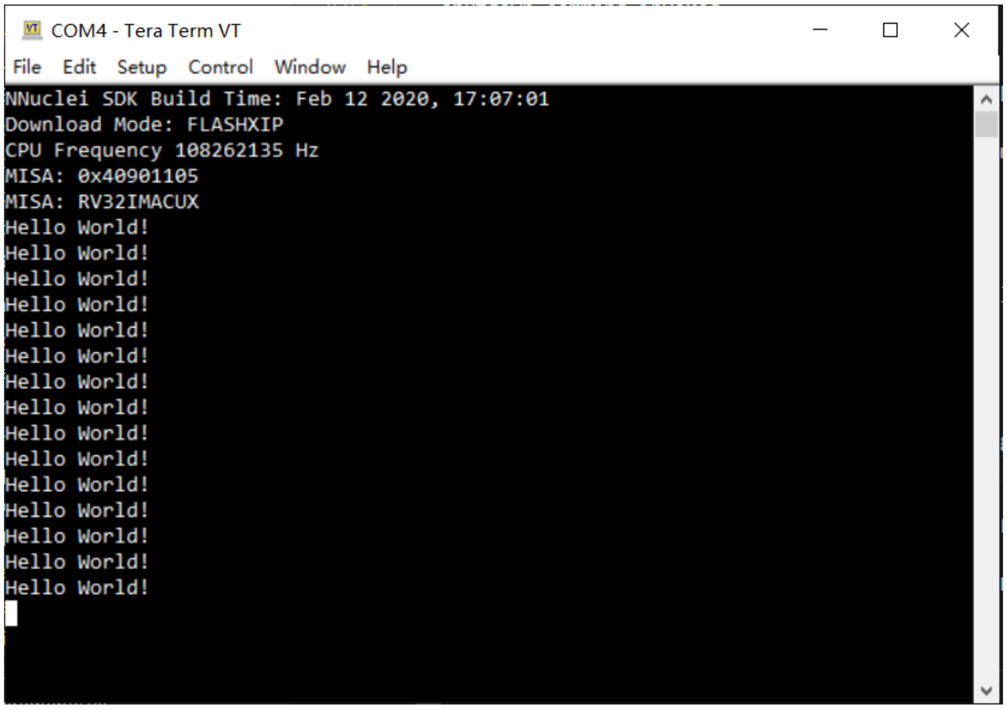
```
"Download and run hello_world.elf"
riscv-nuclei-elf-gdb hello_world.elf -ex "set remotetimeout 240" \
        -ex "target remote | openocd -c \"gdb_port pipe; log_output openocd.log\" -f .
↪./../../SoC/gd32vf103/Board/gd32vf103v_rvstar/openocd_gd32vf103.cfg" \
        --batch -ex "monitor halt" -ex "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\riscv-nuclei-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
```

(continues on next page)

(continued from previous page)

```
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 14).



```
NNuclei SDK Build Time: Feb 12 2020, 17:07:01
Download Mode: FLASHXIP
CPU Frequency 108262135 Hz
MISA: 0x40901105
MISA: RV32IMACUX
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
```

Fig. 8: Nuclei SDK Hello World Application UART Output

2.4.4 Debug Application

If the application is built successfully for this board *GD32VF103V RV-STAR Kit* (page 59), 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.

```
"Download and debug hello_world.elf"
riscv-nuclei-elf-gdb hello_world.elf -ex "set remotetimeout 240" \
```

(continues on next page)

(continued from previous page)

```
        -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\riscv-nuclei-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=riscv-nuclei-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)
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.

        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[17].
- 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:

```
// See LICENSE for license details.
#include <stdio.h>
```

(continues on next page)

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

(continued from previous page)

```c
#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 9) 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:

```makefile
TARGET = hello

NUCLEI_SDK_ROOT = ../../..

SRCDIRS = .

INCDIRS = .

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

  - Open `main.c` and edit the content as 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 9) to run or debug this new application.

Note:

- If your are looking for how to run for other boards, please ref to *Board* (page 55).
- Please refer to *Application Development* (page 36) 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 45) 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 70) to learn about each application usage and expected output.

Note:

- If you met some issues in using this guide, please check *FAQ* (page 97), if still not solved, please *Submit your issue* (page 42).
- 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[18] to download Nuclei Studio IDE, and refer to the Nuclei_Studio_User_Guide.pdf[19] for how to use it.
  2. 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[20] to learn about it
  3. 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[21] to learn more about it, or you can visit Light on onboard LED of RVSTAR board using PlatformIO(Chinese)[22] to play with PlatformIO for Nuclei.
  4. You can also use source code in Nuclei SDK as base, and easily integrate with other IDE tools, such as IAR workbench for RISC-V, Compiler-IDE and others.

[18] https://nucleisys.com/download.php
[19] https://www.nucleisys.com/upload/files/doc/nucleistudio/Nuclei_Studio_User_Guide.pdf
[20] https://github.com/riscv-mcu/ses_nuclei_sdk_projects
[21] https://platformio.org/platforms/nuclei
[22] https://www.rvmcu.com/community-topic-id-310.html

CHAPTER

THREE

DEVELOPER GUIDE

3.1 Code Style

In Nuclei SDK, we use EditorConfig[23] to maintain our development coding styles and astyle[24] tool to format our source code.

- Our editorconfig file[25] 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[26] 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:

[23] https://editorconfig.org/
[24] http://astyle.sourceforge.net/
[25] https://github.com/Nuclei-Software/nuclei-sdk/tree/master/.editorconfig
[26] http://www.doxygen.nl/manual/docblocks.html

```
gmsl/
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 25) to learn more.

This file will include optional *Makefile.global* (page 23) and *Makefile.local* (page 24) 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
- *Makefile.misc* (page 20): 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)[27], 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.

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.

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

  - **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 21) and *Makefile.rtos* (page 22)
- 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 25) makefile variable setting.

It will define the following items:

- **DOWNLOAD** and **CORE** variables
  - For *Nuclei Demo SoC* (page 51), we can support all the modes defined in *DOWNLOAD* (page 27), and **CORE** list defined in *Makefile.core* (page 23)
  - For *GD32VF103 SoC* (page 53), 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 `demosoc` as example:

```
SoC/demosoc
├── Board
│   └── nuclei_fpga_eval
│       ├── Include
│       │   ├── board_nuclei_fpga_eval.h
│       │   └── nuclei_sdk_hal.h
│       ├── Source
│       │   └── GCC
│       └── openocd_demosoc.cfg
├── build.mk
└── Common
    ├── Include
    │   ├── demosoc.h
    │   ├── ... ...
    │   ├── demosoc_uart.h
    │   ├── nuclei_sdk_soc.h
    │   └── system_demosoc.h
    └── Source
        ├── Drivers
        │   ├── ... ...
        │   └── demosoc_uart.c
        ├── GCC
        │   ├── intexc_demosoc.S
        │   └── startup_demosoc.S
        ├── Stubs
        │   ├── read.c
        │   ├── ... ...
        │   └── write.c
        ├── demosoc_common.c
        └── system_demosoc.c
```

Makefile.rtos

This **Makefile.rtos** will include **<NUCLEI_SDK_ROOT>/OS/<RTOS>/build.mk** according to our *RTOS* (page 30) 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 30) 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 31), 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 n205 n205e n305 n307 n307fd 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 59), 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 59) board, e.g.
  - Build and run application for *GD32VF103V RV-STAR Kit* (page 59): `make run`

- Debug application for *GD32VF103V RV-STAR Kit* (page 59): `make debug`

- The *GD32VF103V RV-STAR Kit* (page 59) 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=demosoc BOARD=nuclei_fpga_eval DOWNLOAD=ilm`

Makefile.local

As the *Makefile.global* (page 23) 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 59), 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 24).

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
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
run_gdb	build and start gdb process with selected configuration
debug	build and debug application with selected configuration
size	show program size

Note:

- The selected configuration is controlled by *Makefile variables passed by make command* (page 25)
- For `run_openocd` and `run_gdb` target, if you want to change a new gdb port, you can pass the variable *GDB_PORT* (page 29)

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 25)
- *BOARD* (page 26)
- *DOWNLOAD* (page 27)
- *CORE* (page 28)
- *SIMULATION* (page 28)
- *GDB_PORT* (page 29)
- *V* (page 29)
- *SILENT* (page 29)

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.

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

Currently we support the following SoCs, see *Supported SoCs* (page 25).

Table 2: Supported SoCs

SOC	Reference
gd32vf103	*GD32VF103 SoC* (page 53)
demosoc	*Nuclei Demo SoC* (page 51)

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 26)
- *Supported Boards when SOC=demosoc* (page 26)

Currently we support the following SoCs.

Table 3: Supported Boards when SOC=gd32vf103

BOARD	Reference
gd32vf103v_rvstar	*GD32VF103V RV-STAR Kit* (page 59)
gd32vf103v_eval	*GD32VF103V Evaluation Kit* (page 60)
gd32vf103c_longan_nano	*Sipeed Longan Nano* (page 61)
gd32vf103c_t_display	*Sipeed Longan Nano* (page 61)

Table 4: Supported Boards when SOC=demosoc

BOARD	Reference
nuclei_fpga_eval	*Nuclei FPGA Evaluation Kit* (page 55)

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.

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 27)

Table 5: Supported download modes

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

Note:

- *GD32VF103 SoC* (page 53) only support **DOWNLOAD=flashxip**
- *Nuclei Demo SoC* (page 51) support all the download modes.
- **flashxip** mode in *Nuclei Demo SoC* (page 51) 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 demosoc, 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 `demosoc.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.

CORE

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

Currently it has these cores supported as described in table *Supported Nuclei Processor cores* (page 28).

Table 6: Supported Nuclei Processor cores

CORE	ARCH	ABI
n201	rv32iac	ilp32
n201e	rv32eac	ilp32e
n203	rv32imac	ilp32
n203e	rv32emac	ilp32e
n205	rv32imac	ilp32
n205e	rv32emac	ilp32e
n305	rv32imac	ilp32
n307	rv32imafc	ilp32f
n307fd	rv32imafdc	ilp32d
n600	rv32imac	ilp32
n600f	rv32imafc	ilp32f
n600fd	rv32imafdc	ilp32d
nx600	rv64imac	lp64
nx600f	rv64imafc	lp64f
nx600fd	rv64imafdc	lp64d
ux600	rv64imac	lp64
ux600f	rv64imafc	lp64f
ux600fd	rv64imafdc	lp64d

When **CORE** is selected, the **ARCH** and **ABI** are set, and it will affect the compiler options, eg. If **CORE=n205**, then `ARCH=rv32imac, ABI=ilp32`, riscv arch related compile and link options will be passed, for this case, it will be `-march=rv32imac -mabi=ilp32 -mcmodel=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.

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=demosoc BOARD=nuclei_fpga_eval CORE=n307 GDB_PORT=3344 run_openocd`
- connect gdb with openocd server: `make SOC=demosoc BOARD=nuclei_fpga_eval CORE=n307 GDB_PORT=3344 run_gdb`

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* (page 30)
- *NUCLEI_SDK_ROOT* (page 30)
- *MIDDLEWARE* (page 31)
- *RTOS* (page 30)
- *PFLOAT* (page 31)
- *NEWLIB* (page 31)
- *NOGC* (page 31)
- *RTTHREAD_MSH* (page 32)

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 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 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 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 `UCOSII` application, you need to have an `rtconfig.h` header file prepared in you application. See examples in `application/rtthread`.

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.

PFLOAT

PFLOAT variable is used to enable floating point value print when using the newlib nano(**NEWLIB=nano**).

PFLOAT=1 will enable float print, it will pass extra `-u _printf_float` through link option.

If you don’t use newlib nano, this variable will have no affect.

when `

NEWLIB

NEWLIB variable is used to select which newlib version will be chosen.

If **NEWLIB=nano**, then newlib nano will be selected. About newlib, please visit https://sourceware.org/newlib/README.

If **NEWLIB=**, then normal newlib will be used.

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* (page 33)
- *C_INCDIRS* (page 33)
- *CXX_INCDIRS* (page 33)
- *ASM_INCDIRS* (page 33)
- *SRCDIRS* (page 33)
- *C_SRCDIRS* (page 33)
- *CXX_SRCDIRS* (page 34)
- *ASM_SRCDIRS* (page 34)
- *C_SRCS* (page 34)
- *CXX_SRCS* (page 34)
- *ASM_SRCS* (page 34)
- *COMMON_FLAGS* (page 34)
- *CFLAGS* (page 35)
- *CXXFLAGS* (page 35)
- *ASMFLAGS* (page 35)
- *LDFLAGS* (page 35)
- *LDLIBS* (page 35)
- *LIBDIRS* (page 35)
- *LINKER_SCRIPT* (page 36)

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.

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.

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.

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

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

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.

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 link extra math library, you can add a newline `LDFLAGS += -lm` in you application Makefile.

Libraries (-lfoo) could also be added to the LDLIBS variable instead.

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 **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 33): Add C/CPP/ASM source code located in the directories defined by this variable.
- *C_SRCDIRS* (page 33): Add C only source code located in the directories defined by this variable.
- *CXX_SRCDIRS* (page 34): Add CPP only source code located in the directories defined by this variable.
- *ASM_SRCDIRS* (page 34): Add ASM only source code located in the directories defined by this variable.

To add only selected source code in directory

- *C_SRCS* (page 34): Add C only source code files defined by this variable.
- *CXX_SRCS* (page 34): Add CPP only source code files defined by this variable.
- *ASM_SRCS* (page 34): Add ASM only source code 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 33): Include the directories defined by this variable for C/ASM/CPP code during compiling.
- *C_INCDIRS* (page 33): Include the directories defined by this variable for C only code during compiling.
- *CXX_INCDIRS* (page 33): Include the directories defined by this variable for CPP only code during compiling.
- *ASM_INCDIRS* (page 33): 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 34): This will add compiling flags for C/CPP/ASM source code.
- *CFLAGS* (page 35): This will add compiling flags for C source code.
- *CXXFLAGS* (page 35): This will add compiling flags for CPP source code.
- *ASMFLAGS* (page 35): This will add compiling flags for ASM source code.
- *LDFLAGS* (page 35): This will add linker flags when linking.
- *LDLIBS* (page 35): This will add extra libraries need to be linked.
- *LIBDIRS* (page 35): 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 72) for example usage of optimize for code size.

For more details about gcc optimization, please refer to Options That Control Optimization in GCC[28].

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

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 36) variable to set your linker script.

The default linker script used for different boards can be found in *Board* (page 55).

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 23).

Set Local Make Options For Your Application

If you want to change the application level Make options, you can add the *Makefile.local* (page 24).

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
```

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 39)
- *Submit your issue* (page 42)
- *Submit your pull request* (page 43)

4.1 Port your Nuclei SoC into Nuclei SDK

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

Assume your SoC name is `ncstar`, based on Nuclei core **n307**, 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 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
└── Source
    ├── Drivers
    │   ├── peripheral_or_device_sources.c
    │   └── ......
    ├── GCC
    │   ├── intexc_ncstar.S
    │   └── startup_ncstar.S
    ├── Stubs
    │   ├── clock_getres.c
    │   ├── clock_gettime.c
    │   ├── clock_settime.c
    │   ├── close.c
    │   ├── execve.c
```

(continues on next page)

(continued from previous page)

```
        │       ├── exit.c
        │       ├── fork.c
        │       ├── fstat.c
        │       ├── getpid.c
        │       ├── gettimeofday.c
        │       ├── isatty.c
        │       ├── kill.c
        │       ├── link.c
        │       ├── lseek.c
        │       ├── open.c
        │       ├── read.c
        │       ├── sbrk.c
        │       ├── stat.c
        │       ├── times.c
        │       ├── unlink.c
        │       ├── wait.c
        │       └── write.c
        ├── 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 porting code, mainly `_write`, `_read`, `_sbrk` stub function, take `SoC/demosoc/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 *demosoc* SoC.

– 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:

- **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 := n307
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 \
             $(NUCLEI_SDK_SOC_COMMON)/Source/Stubs

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
```

(continues on next page)

(continued from previous page)

```
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/demosoc/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 `demosoc` and `gd32vf103` SoC.
- The `demosoc` SoC is a FPGA based evaluation platform, it have `ilm` and `dlm`, so it support three *download modes* (page 27)
- 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[29].

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 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
│   ├── Makefile.base
│   ├── Makefile.conf
│   ├── Makefile.core
│   ├── Makefile.components
│   ├── Makefile.files
│   ├── Makefile.global
│   ├── Makefile.misc
│   ├── Makefile.rtos
│   ├── Makefile.rules
│   └── Makefile.soc
├── doc
```

(continues on next page)

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

(continued from previous page)

```
│   ├── build
│   ├── source
│   ├── Makefile
│   └── requirements.txt
├── NMSIS
│   ├── Core
│   ├── DSP
│   ├── NN
│   └── Library
├── OS
│   ├── FreeRTOS
│   ├── UCOSII
│   └── RTThread
├── SoC
│   ├── gd32vf103
│   └── demosoc
├── 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/demosoc/*.

  In each SoC's include directory, *nuclei_sdk_soc.h* must be provided, and include the soc header file, for example, *SoC/demosoc/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/demosoc/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[30] 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 Plaform for PlatformIO[31].

- **SConscript**

  RT-Thread package scons build script, used in RT-Thread package development[32].

- **Makefile**

  An external Makefile just for build, run, debug application without cd to any coresponding 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*.

  ```
  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
  │   ├── riscv-nuclei-elf
  │   └── share
  └── openocd
      ├── bin
      ├── contrib
      ├── distro-info
      ├── OpenULINK
      ├── scripts
      └── share
  ```

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

- **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
  │   ├── riscv-nuclei-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 48): How Nuclei Processor Core is used in Nuclei SDK
- *SoC* (page 51): How Nuclei processor code based SoC device is supported in Nuclei SDK
- *Board* (page 55): How Nuclei based SoC's Board is supported in Nuclei SDK
- *Peripheral* (page 66): How to use the peripheral driver in Nuclei SDK
- *RTOS* (page 67): What RTOSes are supported in Nuclei SDK
- *Application* (page 70): 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[33] to learn more about Nuclei RISC-V Instruction Set Architecture.

[33] https://doc.nucleisys.com/nuclei_spec/

5.2.1 Introduction

Nuclei provides the following RISC-V IP Products[34] 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.
- **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.

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[35] framework, user can access NMSIS Core API[36], NMSIS DSP API[37] and NMSIS NN API[38] provided by NMSIS[39].

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:

```
#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 = .;
```

(continues on next page)

[34] https://nucleisys.com/product.php
[35] https://github.com/Nuclei-Software/NMSIS
[36] https://doc.nucleisys.com/nmsis/core/api/index.html
[37] https://doc.nucleisys.com/nmsis/dsp/api/index.html
[38] https://doc.nucleisys.com/nmsis/nn/api/index.html
[39] https://github.com/Nuclei-Software/NMSIS

(continued from previous page)

```
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 the `_write` and `_read` stub functions for `printf` functions
- 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 51)
- If you want to learn more about Board, please click *Board* (page 55)
- If you want to learn more about Peripheral, please click *Peripheral* (page 66)

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.

Nuclei Demo SoC is an evaluation FPGA SoC from Nuclei for customer to evaluate Nuclei RISC-V Process Core.

Overview

To easy user to evaluate Nuclei Processor Core, the prototype SoC (called Nuclei Demo 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, GPIO, SPI, I2C, etc.

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

The SoC diagram can be checked as below *Nuclei Demo SoC Diagram* (page 51)

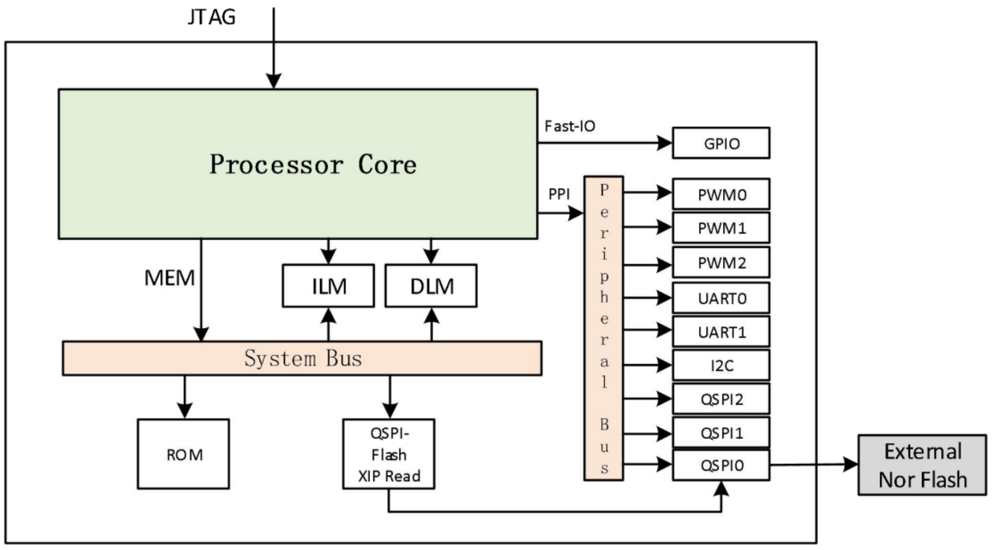
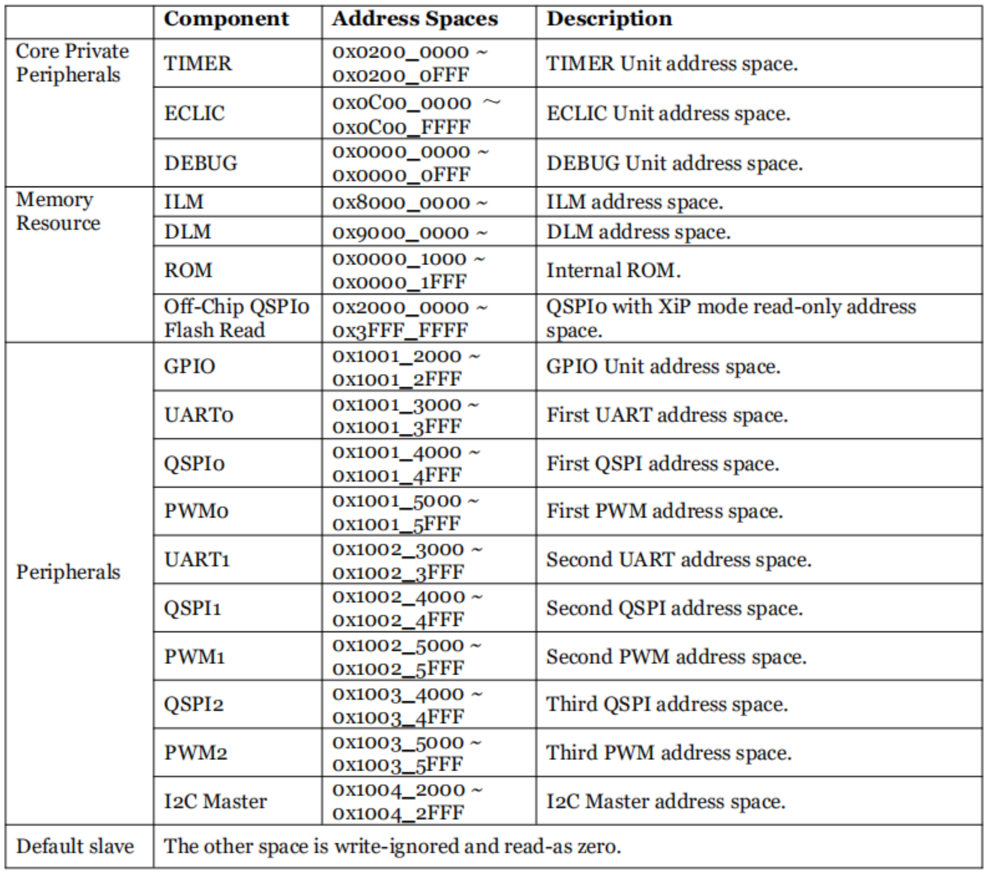


Fig. 1: Nuclei Demo SoC Diagram

The SoC memory map for SoC resources is as below *Nuclei Demo SoC Memory Map* (page 52)

If you want to learn more about this evaluation SoC, please get the `<Nuclei_Eval_SoC_Intro.pdf>` from Nuclei[40].

[40] https://nucleisys.com/



	Component	Address Spaces	Description
Core Private Peripherals	TIMER	0x0200_0000 ~ 0x0200_0FFF	TIMER Unit address space.
	ECLIC	0x0C00_0000 ~ 0x0C00_FFFF	ECLIC Unit address space.
	DEBUG	0x0000_0000 ~ 0x0000_0FFF	DEBUG Unit address space.
Memory Resource	ILM	0x8000_0000 ~	ILM address space.
	DLM	0x9000_0000 ~	DLM address space.
	ROM	0x0000_1000 ~ 0x0000_1FFF	Internal ROM.
	Off-Chip QSPI0 Flash Read	0x2000_0000 ~ 0x3FFF_FFFF	QSPI0 with XiP mode read-only address space.
Peripherals	GPIO	0x1001_2000 ~ 0x1001_2FFF	GPIO Unit address space.
	UART0	0x1001_3000 ~ 0x1001_3FFF	First UART address space.
	QSPI0	0x1001_4000 ~ 0x1001_4FFF	First QSPI address space.
	PWM0	0x1001_5000 ~ 0x1001_5FFF	First PWM address space.
	UART1	0x1002_3000 ~ 0x1002_3FFF	Second UART address space.
	QSPI1	0x1002_4000 ~ 0x1002_4FFF	Second QSPI address space.
	PWM1	0x1002_5000 ~ 0x1002_5FFF	Second PWM address space.
	QSPI2	0x1003_4000 ~ 0x1003_4FFF	Third QSPI address space.
	PWM2	0x1003_5000 ~ 0x1003_5FFF	Third PWM address space.
	I2C Master	0x1004_2000 ~ 0x1004_2FFF	I2C Master address space.
Default slave	The other space is write-ignored and read-as zero.		

Fig. 2: Nuclei Demo SoC Memory Map

Supported Boards

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

- *Nuclei FPGA Evaluation Kit* (page 55)

Usage

If you want to use this **Nuclei demosoc SoC** in Nuclei SDK, you need to set the *SOC* (page 25) Makefile variable to `demosoc`.

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

5.3.2 GD32VF103 SoC

GD32VF103 SoC is the first general RISC-V MCU from GigaDevice Semiconductor[41] 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/

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 54)

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

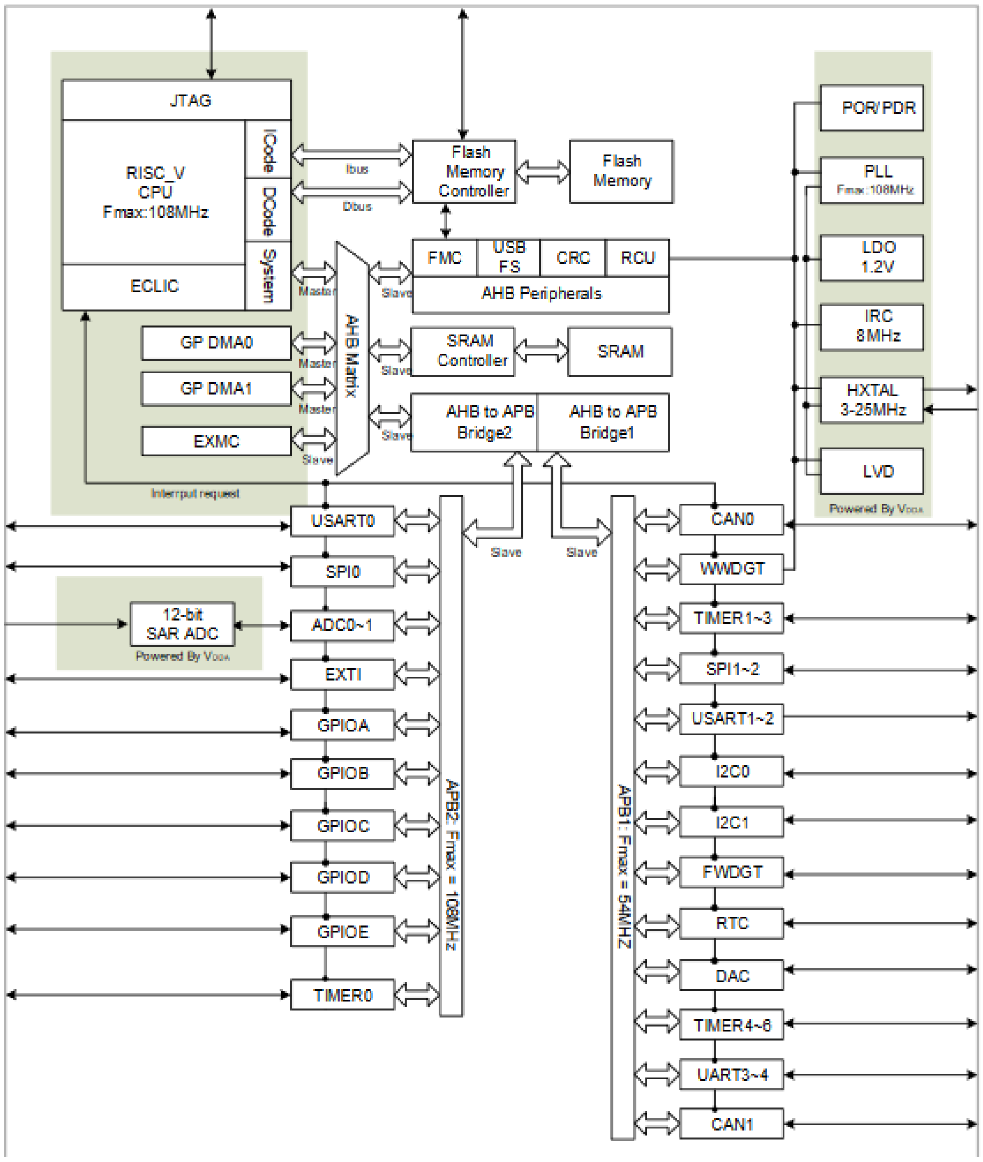


Fig. 3: GD32VF103 SoC Diagram

Supported Boards

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

- *GD32VF103V Evaluation Kit* (page 60)
- *GD32VF103V RV-STAR Kit* (page 59)
- *Sipeed Longan Nano* (page 61)
- *TTGO T-Display-GD32* (page 65)

Usage

If you want to use this **GD32VF103** SoC in Nuclei SDK, you need to set the *SOC* (page 25) Makefile variable to `gd32vf103`.

```
# 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 all
```

Note:

- Since this `gd32vf103` SoC is a real chip, it is using Nuclei RISC-V N205 core, so the **CORE** is fixed to `n205`
- If you want to use **USB** related functions of **GD32VF103**, you need add `USB_DRV_SUPPORT = 1` in your application Makefile.
- In the `usb_conf.h` file of GD32VF103 firmware library, you can configure the USB driver.

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 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.

- **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.

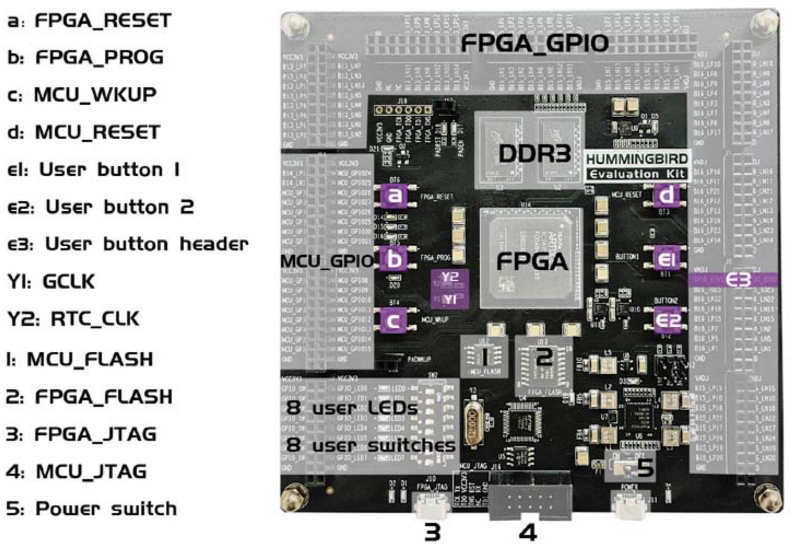


Fig. 4: Nuclei FPGA Evaluation Kit, 100T Version

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[42].

- **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.

Click Nuclei FPGA Evaluation Kit Board Documents[43] to access the documents of these boards.

Setup

Follow the guide in Nuclei FPGA Evaluation Kit Board Documents[44] 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.

[42] https://github.com/Nuclei-Software/nuclei-linux-sdk
[43] https://nucleisys.com/developboard.php
[44] https://nucleisys.com/developboard.php

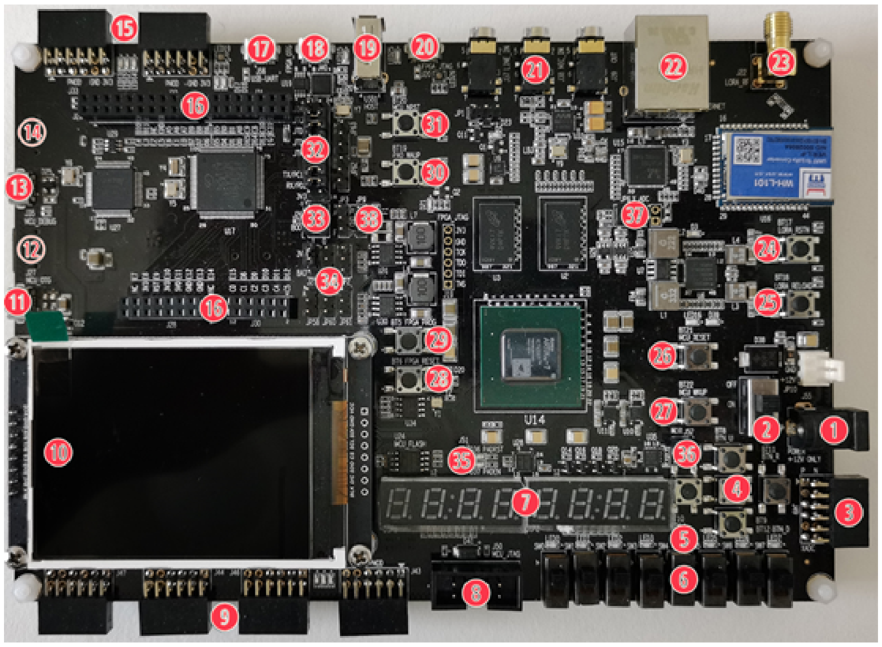


Fig. 5: Nuclei FPGA Evaluation Kit, DDR 200T Version

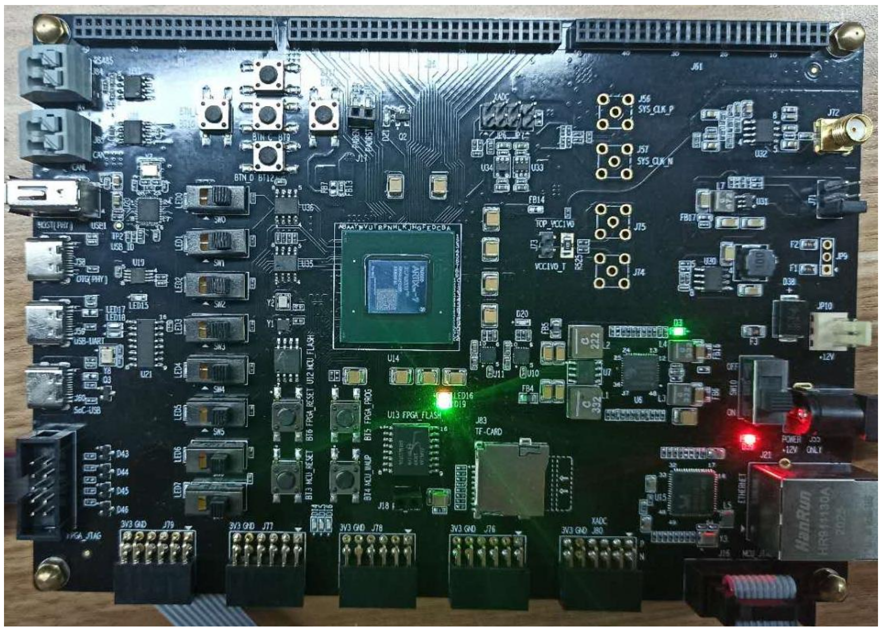


Fig. 6: Nuclei FPGA Evaluation Kit, MCU 200T Version

- Following steps in debugger kit manual[45] to setup JTAG drivers for your development environment

How to use

For **Nuclei FPGA Evaluation board**:

- **DOWNLOAD** support all the modes list in *DOWNLOAD* (page 27)
  - You can find default used linker scripts for different download modes in `SoC/nuclei/Board/nuclei_fpga_eval/Source/GCC/`
    - `gcc_demosoc_ilm.ld`: Linker script file for `DOWNLOAD=ilm`
    - `gcc_demosoc_flash.ld`: Linker script file for `DOWNLOAD=flash`
    - `gcc_demosoc_flashxip.ld`: Linker script file for `DOWNLOAD=flashxip`
    - `gcc_demosoc_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 38)
  - 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[46] in the linker script file it according to your SoC memory information.
- **CORE** support all the cores list in *CORE* (page 28)
- Its openocd configuration file can be found in `SoC/nuclei/Board/nuclei_fpga_eval/openocd_demosoc.cfg`

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

```
# Clean the application with DOWNLOAD=ilm CORE=n307
make SOC=demosoc BOARD=nuclei_fpga_eval DOWNLOAD=ilm CORE=n307 clean
# Build the application with DOWNLOAD=ilm CORE=n307
make SOC=demosoc BOARD=nuclei_fpga_eval DOWNLOAD=ilm CORE=n307 all
# Upload the application using openocd and gdb with DOWNLOAD=ilm CORE=n307
make SOC=demosoc BOARD=nuclei_fpga_eval DOWNLOAD=ilm CORE=n307 upload
# Debug the application using openocd and gdb with DOWNLOAD=ilm CORE=n307
make SOC=demosoc BOARD=nuclei_fpga_eval DOWNLOAD=ilm CORE=n307 debug
```

Note:

- 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.

[45] https://www.nucleisys.com/theme/package/Nuclei_FPGA_DebugKit_Intro.pdf
[46] https://sourceware.org/binutils/docs/ld/MEMORY.html

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.

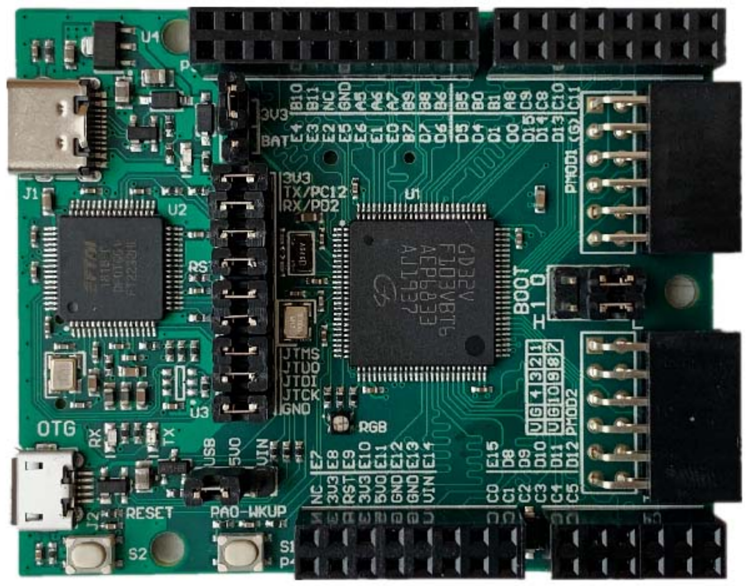


Fig. 7: GD32VF103V RV-STAR Board

Click GD32VF103V RV-STAR Development Kit[47] to access the documents of this board.

Click online RV-STAR Development Board Overview[48] to get basic information of this board.

Setup

Follow the guide in GD32VF103V RV-STAR Development Kit[49] 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[50] to setup JTAG drivers for your development environment

[47] https://nucleisys.com/developboard.php
[48] https://doc.nucleisys.com/nuclei_board_labs/hw/hw.html#rv-star
[49] https://nucleisys.com/developboard.php
[50] https://doc.nucleisys.com/nuclei_board_labs/hw/hw.html#on-board-debugger-driver

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 38)
- 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.

```
# 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.

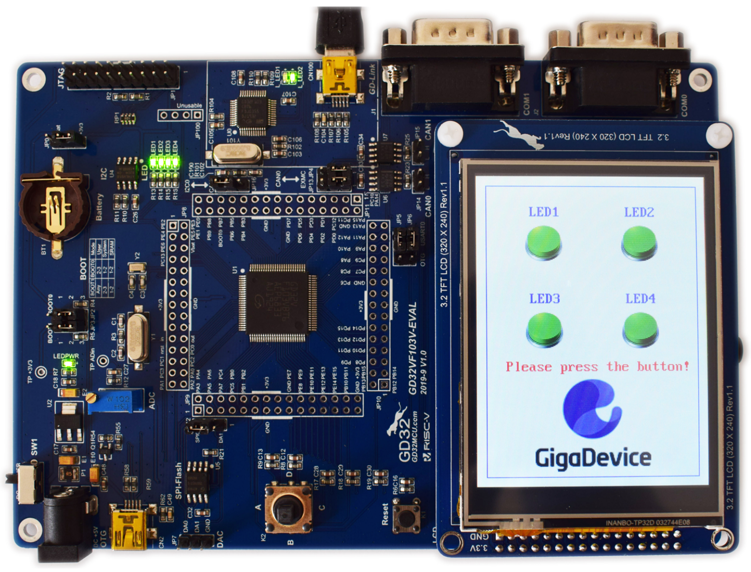


Fig. 8: GD32VF103V-EVAL Board

If you want to learn about this board, please click GD32VF103V EVAL Board Documents[51].

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

Setup

Follow the guide in GD32VF103V EVAL Board Documents[52] 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

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 38)
- 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.

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

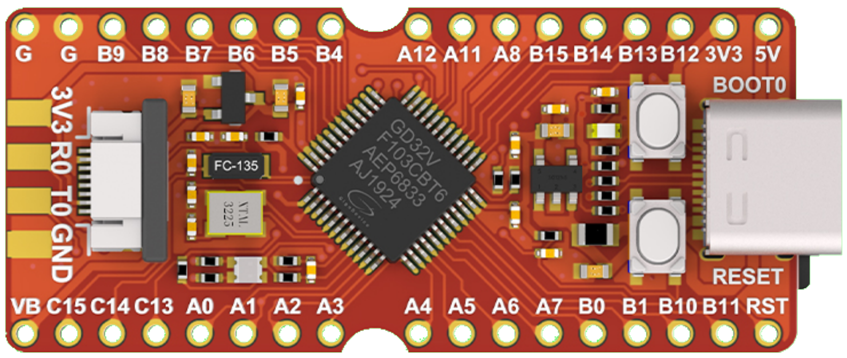


Fig. 9: Sipeed Longan Nano Board.

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

Schematic

Resources

Click Sipeed Longan Nano Documentation[53] to get all information about this board from Sipeed website.

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 63)

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

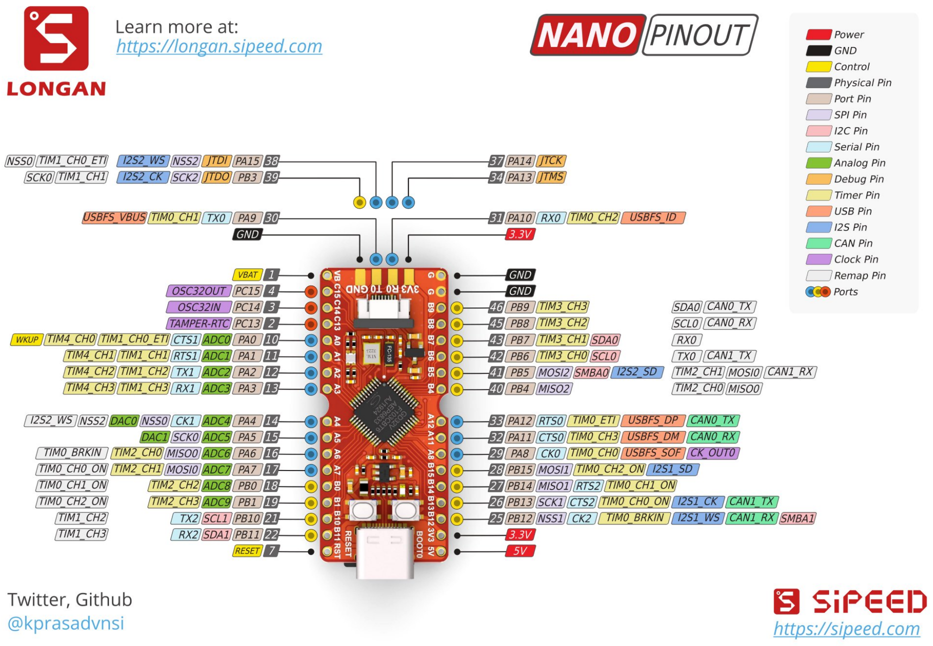


Fig. 10: Sipeed Longan Nano Pinout.

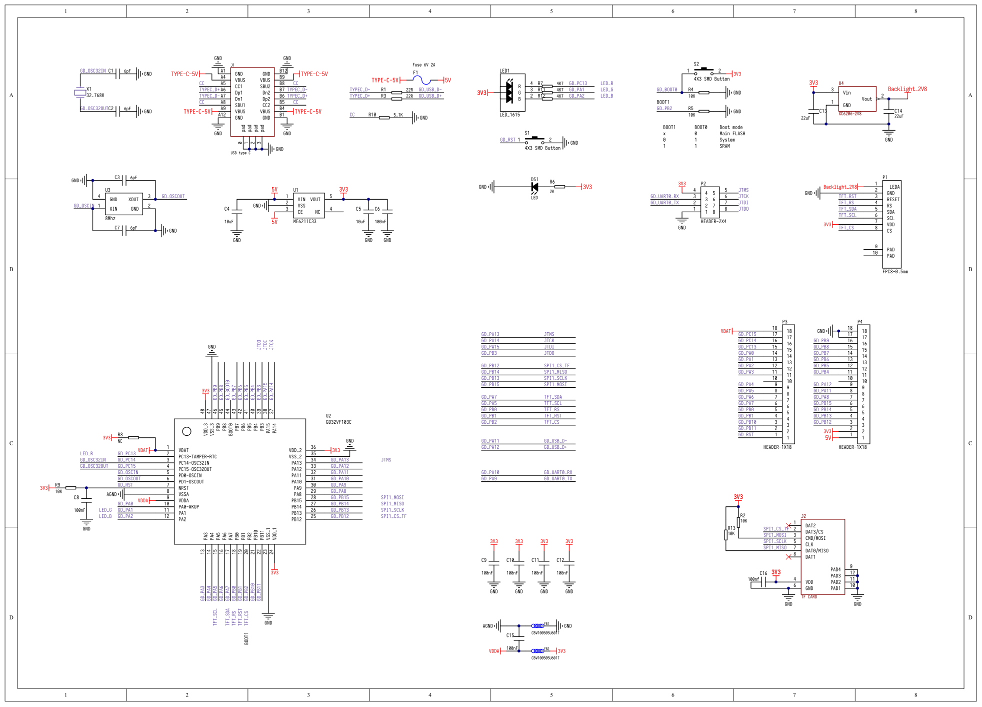


Fig. 11: Sipeed Longan Nano Schematic.

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 38)
- 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 TTGO T-Display-GD32

Overview

The TTGO T-Display-GD32[54] is a minimal board from LilyGo using the GD32VF103CBT6 as main MCU.

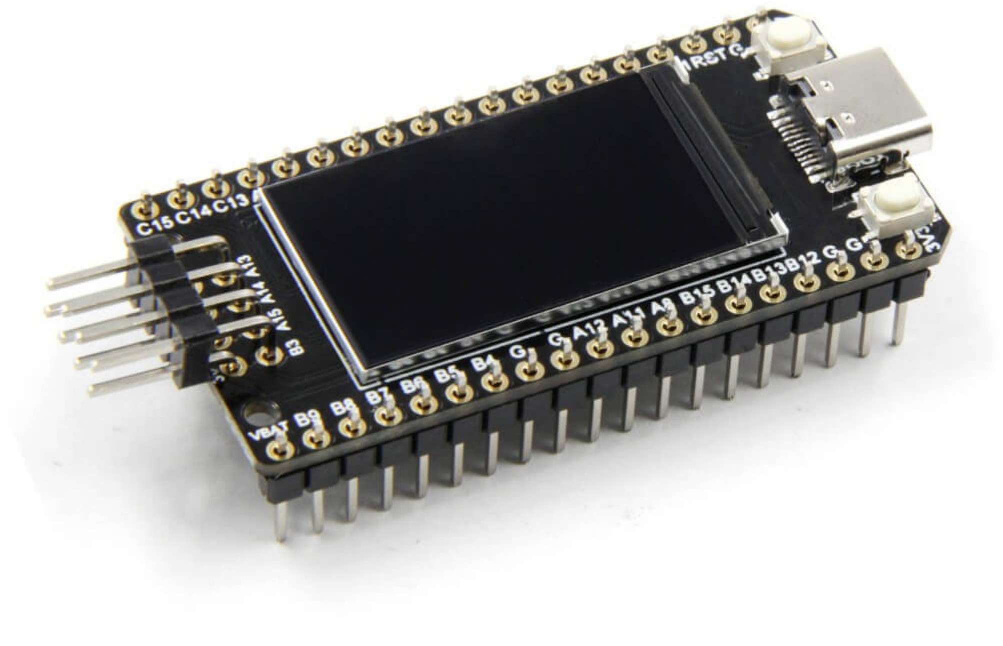


Fig. 12: TTGO T-Display-GD32 Board

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

[54] 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 38)
- 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
```

5.5 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 **Nuclei Demo SoC firmware library APIs** in `SoC/demosoc/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_DRV_SUPPORT = 1` in your application.

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/demosoc/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 three most-used RTOSes in the world, **FreeRTOS**, **UCOSII** and **RT-Thread** from China.

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[55] 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.

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

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

Note:

- You can check the `application\freertos\demo` for reference
- Current version of FreeRTOS used in Nuclei SDK is `V10.3.1`
- 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

5.6.3 UCOSII

UCOSII[56] 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\demo` for 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`

[56] https://www.micrium.com/

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 69) RT-Thread 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[57], which only provide kernel code of RT-Thread, which is easy to be intergated with Nuclei SDK.

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:

- 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[58].

[57] https://github.com/RT-Thread/rtthread-nano
[58] https://github.com/RT-Thread/rt-thread/tree/master/bsp/nuclei/

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 demostrate 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`

If you want to develop your own application in Nuclei SDK, please click *Application Development* (page 36) to learn more about it.

The following applications are running using RV-STAR board.

5.7.2 Bare-metal applications

helloworld

This helloworld application[59] 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 BOARD=gd32vf103v_rvstar clean
# Build and upload the application
make SOC=gd32vf103 BOARD=gd32vf103v_rvstar 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!
```

(continues on next page)

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

(continued from previous page)

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

demo_timer

This demo_timer application[60] is used to demostrate how to use the CORE TIMER API including the Timer Interrupt and Timer Software Interrupt.

- Both interrupts are registered as non-vector interrupt.
- First the timer interrupt will run for 10 times
- Then the software timer interrupt will start to run for 10 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 BOARD=gd32vf103v_rvstar clean
# Build and upload the application
make SOC=gd32vf103 BOARD=gd32vf103v_rvstar 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 IRQ handler 6
MTimer IRQ handler 7
MTimer IRQ handler 8
MTimer IRQ handler 9
MTimer IRQ handler 10
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 SW IRQ handler 6
MTimer SW IRQ handler 7
MTimer SW IRQ handler 8
```

(continues on next page)

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

(continued from previous page)

```
MTimer SW IRQ handler 9
MTimer SW IRQ handler 10
MTimer msip and mtip interrupt test finish and pass
```

demo_eclic

This demo_eclic application[61] is used to demostrate how to use the ECLIC API and Interrupt.

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`.

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 periodly
- 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 then timer interrupt level, so when timer software interrupt is triggerred, 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 then timer interrupt level, so when timer software interrupt is triggerred, 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
```

(continues on next page)

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

(continued from previous page)

```
make SOC=gd32vf103 BOARD=gd32vf103v_rvstar clean
# Build and upload the application
make SOC=gd32vf103 BOARD=gd32vf103v_rvstar 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 periodly
-------------------
[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
[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 periodly
-------------------
[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
```

(continues on next page)

(continued from previous page)

```
[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
-------------------
[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_dsp

This demo_dsp application[62] is used to demostrate how to NMSIS-DSP API.

- Mainly show how we can use DSP library and header files.
- It mainly demo the `riscv_conv_xx` functions and its reference functions
- If your Nuclei Processor Core has DSP feature enabled, you can pass extra `DSP_ENABLE=ON` in your make command to use NMSIS-DSP library with DSP enabled.
- By default, the application will use NMSIS-DSP library with DSP enabled.

Note:

- From version 0.2.4, this demo is upgraded to a more complex version which shows the usage of `riscv_conv_xx` functions, and `DSP_ENABLE` is changed from `OFF` to `ON` by default.
- The GD32VF103 SoC didn't has DSP enabled, so this SoC can only use NMSIS-DSP library with DSP disabled, so please pass extra `DSP_ENABLE=OFF` when run make.
- 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 BOARD=gd32vf103v_rvstar DSP_ENABLE=OFF clean
# Build and upload the application
make SOC=gd32vf103 BOARD=gd32vf103v_rvstar DSP_ENABLE=OFF upload
```

Expected output as below:

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

```
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
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!
```

demo_nice

This demo_nice application[63] is used to demostrate 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[64].

- Mainly show how to use NICE intrinsic function with compiler.
- It only works with Nuclei RISC-V Processor with the hardware NICE demo intergated.

Note:

- If didn't work with gd32vf103 processor.

How to run this application:

```
# Assume that you can set up the Tools and Nuclei SDK environment
# Use Nuclei UX607 RISC-V processor as example, hardware NICE demo intergated
# cd to the demo_dsp directory
```

(continues on next page)

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

(continued from previous page)

```
cd application/baremetal/demo_nice
# Clean the application first
make SOC=demosoc BOARD=nuclei_fpga_eval CORE=ux600 clean
# Build and upload the application
make SOC=demosoc BOARD=nuclei_fpga_eval CORE=ux600 upload
```

Expected output as below:

```
Nuclei SDK Build Time: Nov 26 2020, 11:14:51
Download Mode: ILM
CPU Frequency 15999631 Hz

Nuclei Nice Acceleration Demonstration
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
2. Do nice matrix column sum and row sum
3. Compare reference and nice result
  1) Reference result:
the sum of each row is :
                130     140     240
the sum of each col is :
                60      160     290
  2) Nice result:
the sum of each row is :
                130     140     240
the sum of each col is :
                60      160     290
  3) Compare reference vs nice: PASS
4. Performance summary
         normal:
              instret: 511, cycle: 790
         nice  :
              instret: 125, cycle: 227
```

coremark

This coremark benchmark application[65] 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.

- By default, this application runs for 500 iterations, you can also change this in Makefile. e.g. Change this `-DITERATIONS=500` to value such as `-DITERATIONS=5000`
- macro **PERFORMANCE_RUN=1** is defined

[65] https://github.com/Nuclei-Software/nuclei-sdk/tree/master/application/baremetal/benchmark/coremark

- **PFLOAT = 1** is added in its Makefile to enable float value print

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=500` 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
# change ITERATIONS value in Makefile for gd32vf103 based board to 5000
# Clean the application first
make SOC=gd32vf103 BOARD=gd32vf103v_rvstar clean
# Build and upload the application
make SOC=gd32vf103 BOARD=gd32vf103v_rvstar upload
```

Expected output as below:

```
Nuclei SDK Build Time: Mar 30 2020, 18:08:53
Download Mode: FLASHXIP
CPU Frequency 107190000 Hz
Start to run coremark for 5000 iterations
2K performance run parameters for coremark.
CoreMark Size    : 666
Total ticks      : 1622809457
Total time (secs): 15.139593
Iterations/Sec   : 330.259868
Iterations       : 5000
Compiler version : GCC9.2.0
Compiler flags   : -O2 -flto -funroll-all-loops -finline-limit=600 -ftree-dominator-
↪opts -fno-if-conversion2 -fselective-scheduling -fno-code-hoisting -fno-common -
↪funroll-loops -finline-functions -falign-functions=4 -falign-jumps=4 -falign-loops=4
Memory location  : STACK
seedcrc          : 0xe9f5
[0]crclist       : 0xe714
[0]crcmatrix     : 0x1fd7
[0]crcstate      : 0x8e3a
[0]crcfinal      : 0xbd59
Correct operation validated. See readme.txt for run and reporting rules.
CoreMark 1.0 : 330.259868 / GCC9.2.0 -O2 -flto -funroll-all-loops -finline-limit=600 -
↪ftree-dominator-opts -fno-if-conversion2 -fselective-scheduling -fno-code-hoisting -
↪fno-common -funroll-loops -finline-functions -falign-functions=4 -falign-jumps=4 -
↪falign-loops=4 / STACK


Print Personal Added Addtional Info to Easy Visual Analysis

    (Iterations is: 5000
    (total_ticks is: 1622809457
(*) Assume the core running at 1 MHz
    So the CoreMark/MHz can be caculated by:
    (Iterations*1000000/total_ticks) = 3.081076 CoreMark/MHz
```

dhrystone

This dhrystone benchmark application[66] is used to run DHRYSTONE Benchmark Software.

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.

- **PFLOAT = 1** is added in its Makefile to enable float value print
- You can change `Number_Of_Runs` in `dhry_1.c` line 134 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=gd32vf103 BOARD=gd32vf103v_rvstar clean
# Build and upload the application
make SOC=gd32vf103 BOARD=gd32vf103v_rvstar upload
```

Expected output as below:

```
Nuclei SDK Build Time: Feb 21 2020, 14:23:55
Download Mode: FLASHXIP
CPU Frequency 108801980 Hz

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
```

(continues on next page)

[66] https://github.com/Nuclei-Software/nuclei-sdk/tree/master/application/baremetal/benchmark/dhrystone

(continued from previous page)

```
        should be:   Number_Of_Runs + 10
Ptr_Glob->
  Ptr_Comp:          536883352
        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:          536883352
        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
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:
223500116
       So the DMIPS/MHz can be caculated by:
       1000000/(User_Cycle/Number_Of_Runs)/1757 = 1.273270 DMIPS/MHz
```

whetstone

This whetstone benchmark application[67] is used to run C/C++ Whetstone Benchmark Software (Single or Double Precision).

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.

- **PFLOAT = 1** is added in its Makefile to enable float value print

[67] https://github.com/Nuclei-Software/nuclei-sdk/tree/master/application/baremetal/benchmark/whetstone

- 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=gd32vf103 BOARD=gd32vf103v_rvstar clean
# Build and upload the application
make SOC=gd32vf103 BOARD=gd32vf103v_rvstar upload
```

Expected output as below:

```
Nuclei SDK Build Time: Feb 21 2020, 14:50:15
Download Mode: FLASHXIP
CPU Frequency 109069306 Hz

##########################################
Single Precision C Whetstone Benchmark Opt 3 32 Bit
Calibrate
       1.96 Seconds          1   Passes (x 100)
       9.81 Seconds          5   Passes (x 100)

Use 5  passes (x 100)

          Single Precision C/C++ Whetstone Benchmark

Loop content                  Result              MFLOPS      MOPS   Seconds

N1 floating point -1.12475013732910156         1.053              0.091
N2 floating point -1.12274742126464844         1.053              0.638
N3 if then else    1.00000000000000000              108527.617     0.000
N4 fixed point    12.00000000000000000                  5.630     0.280
N5 sin,cos etc.    0.49909299612045288                  0.109     3.829
N6 floating point  0.99999982118606567         1.082              2.493
N7 assignments     3.00000000000000000                419.794     0.002
N8 exp,sqrt etc.   0.75110614299774170                  0.075     2.492

MWIPS                                                5.089              9.825


MWIPS/MHz                                            0.046              9.825
```

5.7.3 FreeRTOS applications

demo

This freertos demo application[68] is 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

[68] https://github.com/Nuclei-Software/nuclei-sdk/tree/master/application/freertos/demo

- 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 BOARD=gd32vf103v_rvstar clean
# Build and upload the application
make SOC=gd32vf103 BOARD=gd32vf103v_rvstar 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.....
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
```

5.7.4 UCOSII applications

demo

This ucosii demo application[69] 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

[69] https://github.com/Nuclei-Software/nuclei-sdk/tree/master/application/ucosii/demo

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 BOARD=gd32vf103v_rvstar clean
# Build and upload the application
make SOC=gd32vf103 BOARD=gd32vf103v_rvstar 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[70] 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 BOARD=gd32vf103v_rvstar clean
# Build and upload the application
make SOC=gd32vf103 BOARD=gd32vf103v_rvstar 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
thread 1 count: 3
thread 2 count: 3
thread 3 count: 3
```

(continues on next page)

[70] https://github.com/Nuclei-Software/nuclei-sdk/tree/master/application/rtthread/demo

(continued from previous page)

```
thread 4 count: 3
Main thread count: 2
thread 0 count: 4
thread 1 count: 4
```

msh

This rt-thread msh application[71] 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 Demo SoC* (page 51), 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 BOARD=gd32vf103v_rvstar clean
# Build and upload the application
make SOC=gd32vf103 BOARD=gd32vf103v_rvstar 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
version          - show RT-Thread version information
ps               - List threads in the system.
help             - RT-Thread shell help.
nsdk             - msh nuclei sdk demo

msh >ps
```

(continues on next page)

[71] https://github.com/Nuclei-Software/nuclei-sdk/tree/master/application/rtthread/msh

(continued from previous page)

```
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 >
```

CHAPTER

SIX

CHANGELOG

6.1 V0.3.5-dev

This is version `0.3.5-dev` of Nuclei SDK, which is still in development.

- SoC
  - Provide correct gd32vf103.svd, the previous one content is messed up.

For detailed changes, please check commit histories since 0.3.4 release.

6.2 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.

6.3 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.

6.4 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

6.5 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[72]
  - Add definitions for the Interface Association Descriptor of USB for GD32VF103, contributed by michahoiting[73].
  - **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

[72] https://github.com/tuupola
[73] https://github.com/michahoiting

- Application
  - Increase idle and finsh thread stack for RT-Thread, due to stack size is not enough for RISC-V 64bit
  - 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

6.6 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[74] and RomanBuchert[75]
- 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

[74] https://github.com/tuupola
[75] https://github.com/RomanBuchert

6.7 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`

6.8 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

6.9 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

6.10 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

6.11 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

6.12 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

6.13 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

6.14 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 68)
  - 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.

6.15 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

6.16 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`

6.17 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

6.18 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 Demo 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 addbreviation 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 incompatiable, similar cases such as `bfd requires flen 8, but target has flen 4`

Just change your CORE to proper core settings will solve this issue.

For example, if you compile your core with `CORE=n307`, just change it to `CORE=n305`.

- **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=n305`.

7.2 How to select correct FDTI debugger?

From Nuclei SDK release 0.2.9, the openocd configuration file doesn't contain ftdi_device_desc[76] 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.

- For **Nuclei FPGA Evaluation Board**, you can check the openocd configuration file in *SoC/demosoc/Board/nuclei_fpga_eval/openocd_demosoc.cfg*.
- For **Nuclei RVSTAR Board**, you can check the openocd configuration file in *SoC/gd32vf103/Board/gd32vf103v_rvstar/openocd_gd32vf103.cfg*.

7.3 Why I can't download application in Linux?

Please check that whether you have followed the debugger kit manual[77] to setup the USB JTAG drivers correctly. The windows steps and linux steps are different, please take care.

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?

[76] http://openocd.org/doc/html/Debug-Adapter-Configuration.html
[77] https://www.nucleisys.com/theme/package/Nuclei_FPGA_DebugKit_Intro.pdf

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 this `ECLIC_SetVector` API in 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 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 nuclei_fpga_eval ilm download mode: `ILM to 512K, DLM to 256K`, then you can change link script file `SoC/demosoc/Board/nuclei_fpga_eval/Source/GCC/gcc_demosoc_ilm.ld` as below:

```
diff --git a/SoC/demosoc/Board/nuclei_fpga_eval/Source/GCC/gcc_demosoc_ilm.ld b/SoC/
↪demosoc/Board/nuclei_fpga_eval/Source/GCC/gcc_demosoc_ilm.ld
index 1ac5b90..08451b3 100644
--- a/SoC/demosoc/Board/nuclei_fpga_eval/Source/GCC/gcc_demosoc_ilm.ld
+++ b/SoC/demosoc/Board/nuclei_fpga_eval/Source/GCC/gcc_demosoc_ilm.ld
@@ -28,8 +28,8 @@ ENTRY( _start )
 MEMORY
 {

-  ilm (rxai!w) : ORIGIN = 0x80000000, LENGTH = 64K
-  ram (wxa!ri) : ORIGIN = 0x90000000, LENGTH = 64K
+  ilm (rxai!w) : ORIGIN = 0x80000000, LENGTH = 512K
```

(continues on next page)

(continued from previous page)

```
+   ram (wxa!ri) : ORIGIN = 0x90000000, LENGTH = 256K
 }
```

CHAPTER EIGHT

LICENSE

(continues on next page)

(continued from previous page)

```
   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:

   (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
```

(continues on next page)

(continued from previous page)

```
          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
      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
```

(continues on next page)

(continued from previous page)

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 RISCV Tools and Documents**: https://nucleisys.com/download.php
- **Nuclei riscv-openocd**: https://github.com/riscv-mcu/riscv-openocd
- **Nuclei riscv-binutils-gdb**: https://github.com/riscv-mcu/riscv-binutils-gdb
- **Nuclei riscv-gnu-toolchain**: https://github.com/riscv-mcu/riscv-gnu-toolchain
- **Nuclei riscv-newlib**: https://github.com/riscv-mcu/riscv-newlib
- **Nuclei riscv-gcc**: https://github.com/riscv-mcu/riscv-gcc
- **Nuclei Software Organization in Github**: https://github.com/Nuclei-Software/
- **Nuclei Software Organization in Gitee**: https://gitee.com/Nuclei-Software/
- **Nuclei SDK**: https://github.com/Nuclei-Software/nuclei-sdk
- **NMSIS**: https://github.com/Nuclei-Software/NMSIS
- **Nuclei Bumblebee Core Document**: https://github.com/nucleisys/Bumblebee_Core_Doc
- **Nuclei RISC-V IP Products**: https://www.nucleisys.com/product.php
- **RISC-V MCU Community Website**: https://www.riscv-mcu.com/
- **Nuclei Spec Documentation**: https://doc.nucleisys.com/nuclei_spec/
- **Nuclei SDK Documentation**: https://doc.nucleisys.com/nuclei_sdk/
- **NMSIS Documentation**: https://doc.nucleisys.com/nmsis/

CHAPTER

ELEVEN

INDICES AND TABLES

- genindex
- search

INDEX