HBird
SDK

HummingBird RISC-V Software Development Kit

HummingBird SDK

Release 0.1.4

Nuclei

Jan 17, 2023

CONTENTS:

CHAPTER

ONE

OVERVIEW

1.1 Introduction

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

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

HummingBird 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:

- The **NMSIS** used in this HummingBird SDK is **modified** for HummingBird RISC-V Core, which is not compatiable with **Nuclei NMSIS**, take care.
- HummingBird SDK is developed based on Nuclei SDK[3] 0.2.4 release, and will diverge in future.
- To get a pdf version of this documentation, please click HBird SDK Document[4]

1.2 Design and Architecture

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

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

- *HummingBird RISC-V Core API* (page 49) service is built on top of a modified version of NMSIS[5], so silicon vendors of HummingBird RISC-V processors can easily port their SoCs to HummingBird SDK, and quickly evaluate software on their SoC.
- **NMSIS-NN** and **NMSIS-DSP** library can be also used in HummingBird SDK, and the prebuilt libraries are included in **NMSIS/Library** folder of HummingBird SDK.
- Mainly support two HummingBird RISC-V Processor based SoCs, *HummingBird SoC* (page 118).

[1] https://github.com/Nuclei-Software/NMSIS
[2] https://github.com/Nuclei-Software/NMSIS
[3] https://github.com/nuclei-software/nuclei-sdk
[4] https://doc.nucleisys.com/hbird_sdk/hummingbirdsdk.pdf
[5] https://github.com/Nuclei-Software/NMSIS

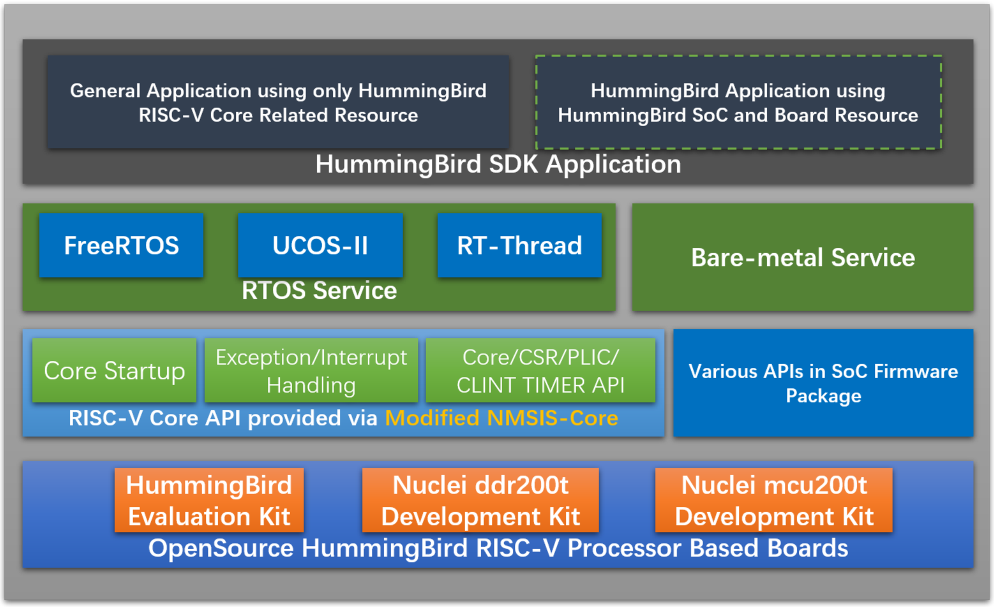

General Application using only HummingBird RISC-V Core Related Resource
HummingBird Application using HummingBird SoC and Board Resource
HummingBird SDK Application
FreeRTOS
UCOS-II
RT-Thread
RTOS Service
Bare-metal Service
Core Startup
Exception/Interrupt Handling
Core/CSR/PLIC/ CLINT TIMER API
RISC-V Core API provided via Modified NMSIS-Core
Various APIs in SoC Firmware Package
HummingBird Evaluation Kit
Nuclei ddr200t Development Kit
Nuclei mcu200t Development Kit
OpenSource HummingBird RISC-V Processor Based Boards

Fig. 1: HummingBird SDK Design and Architecture Diagram

- Provided realtime operation system service via *FreeRTOS* (page 128), *UCOSII* (page 129) and *RT-Thread* (page 130)
- Provided bare-metal service for embedded system software beginners and resource-limited use-cases.
- Currently HummingBird 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.
- Applications are logically seperated into three parts:
  - **General applications for all HummingBird RISC-V Processors**: In the HummingBird SDK software code, the applications provided are all general applications which can run on all HummingBird RISC-V Processors, with basic UART service to provide `printf` function.
  - **HummingBird SoC applications**: These applications are not included in the HummingBird SDK software code, it is *maintained seperately*, it will use resource from HummingBird 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 HummingBird SDK.

1.4 Contributing

Contributing to HummingBird 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

HummingBird SDK is an opensource project licensed by *Apache License 2.0* (page 151).

CHAPTER TWO

QUICK STARTUP

2.1 Setup Tools and Environment

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

- For Windows users, please check *Install and Setup Tools in Windows* (page 5)
- For Linux users, please check *Install and Setup Tools in Linux* (page 6)

2.1.1 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[6], please check and follow the figure *Nuclei Tools need to be downloaded for Windows* (page 5).
   - **Nuclei RISC-V GNU Toolchain for Windows**, see number **1** in the figure *Nuclei Tools need to be downloaded for Windows* (page 5)
   - **Nuclei OpenOCD for Windows**, see number **2** in the figure *Nuclei Tools need to be downloaded for Windows* (page 5)
   - **Windows Build Tools**, see number **3** in the figure *Nuclei Tools need to be downloaded for Windows* (page 5)

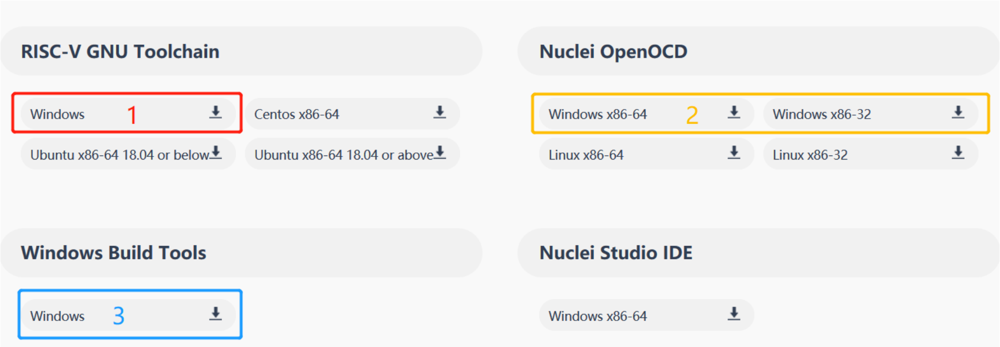
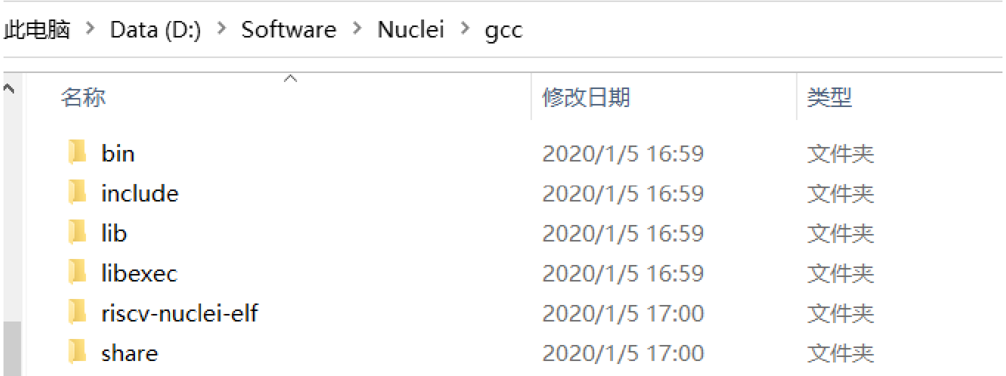


Fig. 1: Nuclei Tools need to be downloaded for Windows

3. Setup tools in previously created `Nuclei` folder, create `gcc`, `openocd` and `build-tools` folders.

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

- **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)

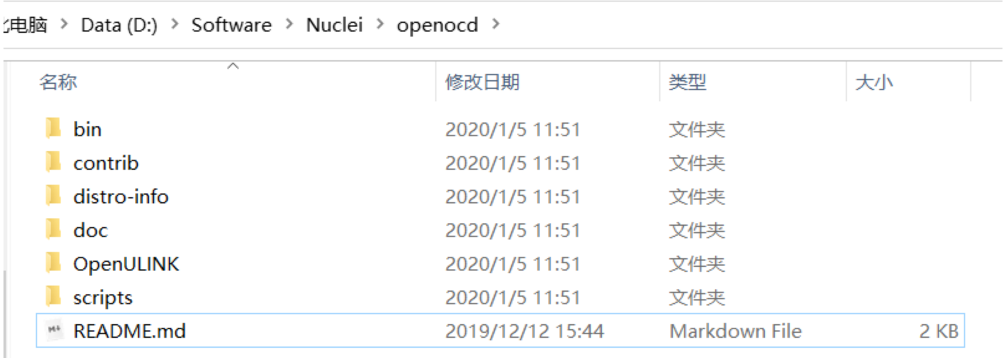


此电脑 › Data (D:) › Software › Nuclei › gcc

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

- **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 6)



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

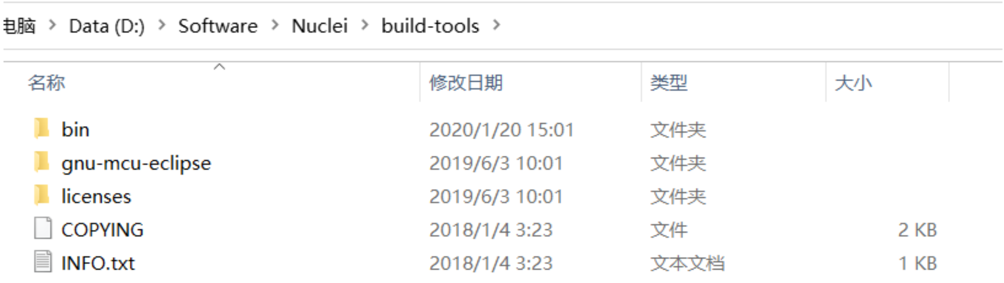
- **Windows Build Tools** Extract the download **build-tools** tool into a temp folder, and copy the files into `build-tools` folder, make sure the `build-tools` directory structure looks like this figure *Nuclei Windows Build Tools directory structure of build-tools* (page 7)

2.1.2 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[7], please check and follow the figure *Nuclei Tools need to be downloaded for Linux* (page 7).
   - **Nuclei RISC-V GNU Toolchain for Linux**, for **CentOS or Ubuntu < 18.04** click number **1-1**, for **Ubuntu >=18.04** click number **1-2** in the figure *Nuclei Tools need to be downloaded for Linux* (page 7)

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



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

- **Nuclei OpenOCD for Linux**, see number **2-1** for 64bit version in the figure *Nuclei Tools need to be downloaded for Linux* (page 7)
- **Make >= 3.82**: Install `Make` using `sudo apt-get install make` in Ubuntu, or `sudo yum install make` in CentOS.

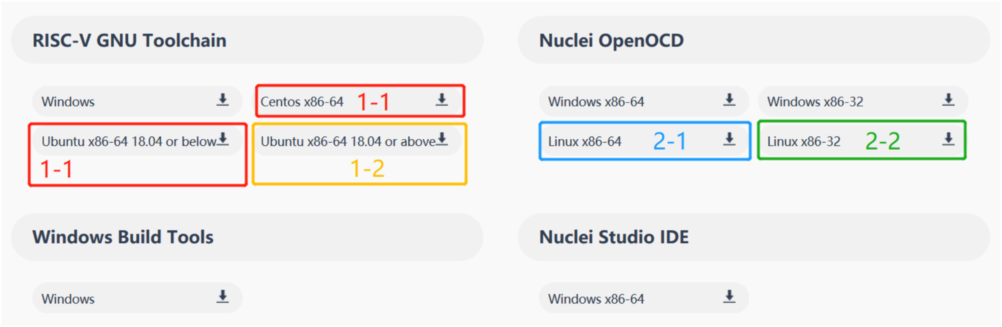


Fig. 5: Nuclei Tools need to be downloaded for Linux

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 5) to extract and copy necessary files.

Note:

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

2.2 Get and Setup HummingBird SDK

The source code of HummingBird SDK is maintained in Github[8] and Gitee[9].

- We mainly maintained github version, and gitee version is mirrored, just for fast access in China.
- Check source code in HummingBird SDK in Github[10].

[8] https://github.com

[9] https://gitee.com

[10] https://github.com/riscv-mcu/hbird-sdk

- Stable version of HummingBird SDK is maintained in **master** version, if you want release version of **HummingBird SDK**, please check in HummingBird SDK Release in Github[11].

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/riscv-mcu/hbird-sdk hbird-sdk` to clone source code into `hbird-sdk` folder

Note:

- If you have no internet access, you can also use pre-downloaded `hbird-sdk` code, and use it.
- 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 HummingBird SDK
  - **Windows** Create `setup_config.bat` in `hbird-sdk` folder, and open this file your editor, and paste the following content, assuming you followed *Install and Setup Tools in Windows* (page 5) and install tools into `D:\Software\Nuclei`, otherwise please use your correct tool root path.

```
set NUCLEI_TOOL_ROOT=D:\Software\Nuclei
```

  - **Linux** Create `setup_config.sh` in `hbird-sdk` folder, and open this file your editor, and paste the following content, assuming you followed *Install and Setup Tools in Linux* (page 6) and install tools into `~/Software/Nuclei`, otherwise please use your correct tool root path.

```
NUCLEI_TOOL_ROOT=~/Software/Nuclei
```

2.3 Build, Run and Debug Sample Application

Assume you have followed steps in *Get and Setup HummingBird SDK* (page 7) to clone source code and create `setup_config.bat` and `setup_config.sh`.

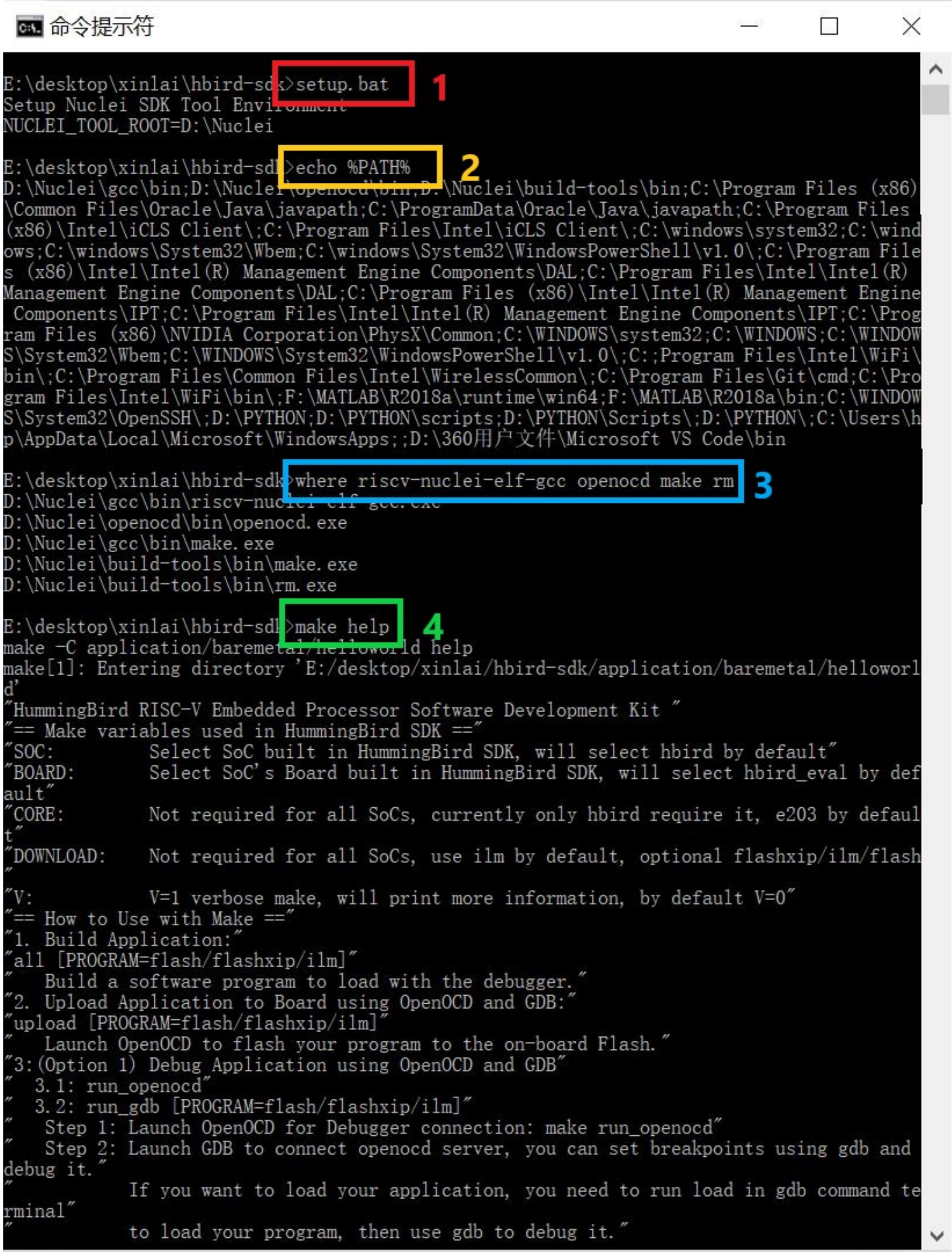
To build, run and debug application, you need to open command terminal in `hbird-sdk` folder.

- For **Windows** users, you can open windows command terminal and cd to `hbird-sdk` folder, then run the following commands to setup build environment for HummingBird SDK, the output will be similar as this screenshot *Setup Build Environment for HummingBird SDK in Windows Command Line* (page 9):

```
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 `hbird-sdk` folder, then run the following commands to setup build environment for HummingBird SDK, the output will be similar as this screenshot *Setup Build Environment for HummingBird SDK in Linux Bash* (page 10):

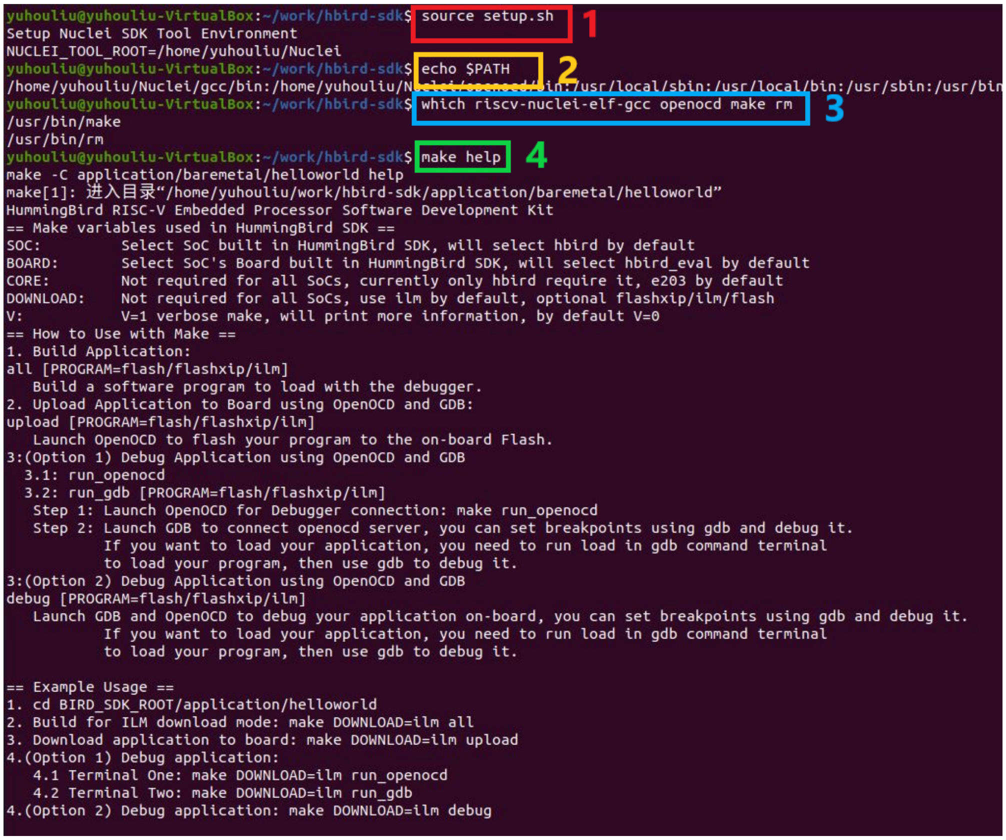
[11] https://github.com/riscv-mcu/hbird-sdk/releases



```
E:\desktop\xinlai\hbird-sdk>setup.bat
Setup Nuclei SDK Tool Environment
NUCLEI_TOOL_ROOT=D:\Nuclei

E:\desktop\xinlai\hbird-sdk>echo %PATH%
D:\Nuclei\gcc\bin;D:\Nuclei\openocd\bin;D:\Nuclei\build-tools\bin;C:\Program Files (x86)
\Common Files\Oracle\Java\javapath;C:\ProgramData\Oracle\Java\javapath;C:\Program Files
(x86)\Intel\iCLS Client\;C:\Program Files\Intel\iCLS Client\;C:\windows\system32;C:\wind
ows;C:\windows\System32\Wbem;C:\windows\System32\WindowsPowerShell\v1.0\;C:\Program File
s (x86)\Intel\Intel(R) Management Engine Components\DAL;C:\Program Files\Intel\Intel(R)
Management Engine Components\DAL;C:\Program Files (x86)\Intel\Intel(R) Management Engine
 Components\IPT;C:\Program Files\Intel\Intel(R) Management Engine Components\IPT;C:\Prog
ram Files (x86)\NVIDIA Corporation\PhysX\Common;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOW
S\System32\Wbem;C:\WINDOWS\System32\WindowsPowerShell\v1.0\;C:;Program Files\Intel\WiFi\
bin\;C:\Program Files\Common Files\Intel\WirelessCommon\;C:\Program Files\Git\cmd;C:\Pro
gram Files\Intel\WiFi\bin\;F:\MATLAB\R2018a\runtime\win64;F:\MATLAB\R2018a\bin;C:\WINDOW
S\System32\OpenSSH\;D:\PYTHON;D:\PYTHON\scripts;D:\PYTHON\Scripts\;D:\PYTHON\;C:\Users\h
p\AppData\Local\Microsoft\WindowsApps;;D:\360用户文件\Microsoft VS Code\bin

E:\desktop\xinlai\hbird-sdk>where riscv-nuclei-elf-gcc openocd make rm
D:\Nuclei\gcc\bin\riscv-nuclei-elf-gcc.exe
D:\Nuclei\openocd\bin\openocd.exe
D:\Nuclei\gcc\bin\make.exe
D:\Nuclei\build-tools\bin\make.exe
D:\Nuclei\build-tools\bin\rm.exe

E:\desktop\xinlai\hbird-sdk>make help
make -C application/baremetal/helloworld help
make[1]: Entering directory 'E:/desktop/xinlai/hbird-sdk/application/baremetal/hellowor l
d'
"HummingBird RISC-V Embedded Processor Software Development Kit "
"== Make variables used in HummingBird SDK =="
"SOC:         Select SoC built in HummingBird SDK, will select hbird by default"
"BOARD:       Select SoC's Board built in HummingBird SDK, will select hbird_eval by def
ault"
"CORE:        Not required for all SoCs, currently only hbird require it, e203 by defaul
t"
"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 te
rminal"
"           to load your program, then use gdb to debug it."
```

Fig. 6: Setup Build Environment for HummingBird SDK in Windows Command Line

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



```
yuhouliu@yuhouliu-VirtualBox:~/work/hbird-sdk$ source setup.sh
Setup Nuclei SDK Tool Environment
NUCLEI_TOOL_ROOT=/home/yuhouliu/Nuclei
yuhouliu@yuhouliu-VirtualBox:~/work/hbird-sdk$ echo $PATH
/home/yuhouliu/Nuclei/gcc/bin:/home/yuhouliu/Nuclei/openocd/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin
yuhouliu@yuhouliu-VirtualBox:~/work/hbird-sdk$ which riscv-nuclei-elf-gcc openocd make rm
/usr/bin/make
/usr/bin/rm
yuhouliu@yuhouliu-VirtualBox:~/work/hbird-sdk$ make help
make -C application/baremetal/helloworld help
make[1]: 进入目录“/home/yuhouliu/work/hbird-sdk/application/baremetal/helloworld”
HummingBird RISC-V Embedded Processor Software Development Kit
== Make variables used in HummingBird SDK ==
SOC:         Select SoC built in HummingBird SDK, will select hbird by default
BOARD:       Select SoC's Board built in HummingBird SDK, will select hbird_eval by default
CORE:        Not required for all SoCs, currently only hbird require it, e203 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
           to load your program, then use gdb to debug it.

== Example Usage ==
1. cd BIRD_SDK_ROOT/application/helloworld
2. Build for ILM download mode: make DOWNLOAD=ilm all
3. Download application to board: make DOWNLOAD=ilm upload
4.(Option 1) Debug application:
   4.1 Terminal One: make DOWNLOAD=ilm run_openocd
   4.2 Terminal Two: make DOWNLOAD=ilm run_gdb
4.(Option 2) Debug application: make DOWNLOAD=ilm debug
```

Fig. 7: Setup Build Environment for HummingBird 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`

Here for a quick startup, this guide will take board *HummingBird Evaluation Kit* (page 122) 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 setuped build environment command terminal.

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

2.3.1 Hardware Preparation

Please check *Board* (page 122) 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 **HummingBird** board, and use Micro-USB 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[12] or Minicom in Linux[13], and minitor the serial port of the Board, the UART baudrate is *115200 bps*

2.3.2 Build Application

We need to build application for this board *HummingBird Evaluation Kit* (page 122) using this command line:

```
make SOC=hbird BOARD=hbird_eval CORE=e203 all
```

Here is the sample output of this command:

```
Current Configuration: RISCV_ARCH=rv32imac RISCV_ABI=ilp32 SOC=hbird BOARD=hbird_eval␣
↪CORE=e203 DOWNLOAD=ilm
Assembling :  ../../../SoC/hbird/Common/Source/GCC/intexc_hbird.S
Assembling :  ../../../SoC/hbird/Common/Source/GCC/startup_hbird.S
Compiling  :  ../../../SoC/hbird/Common/Source/Drivers/hbird_gpio.c
Compiling  :  ../../../SoC/hbird/Common/Source/Drivers/hbird_uart.c
Compiling  :  ../../../SoC/hbird/Common/Source/Stubs/close.c
Compiling  :  ../../../SoC/hbird/Common/Source/Stubs/fstat.c
Compiling  :  ../../../SoC/hbird/Common/Source/Stubs/gettimeofday.c
Compiling  :  ../../../SoC/hbird/Common/Source/Stubs/isatty.c
Compiling  :  ../../../SoC/hbird/Common/Source/Stubs/lseek.c
Compiling  :  ../../../SoC/hbird/Common/Source/Stubs/read.c
Compiling  :  ../../../SoC/hbird/Common/Source/Stubs/sbrk.c
Compiling  :  ../../../SoC/hbird/Common/Source/Stubs/write.c
Compiling  :  ../../../SoC/hbird/Common/Source/hbird_common.c
Compiling  :  ../../../SoC/hbird/Common/Source/system_hbird.c
Compiling  :  hello_world.c
Linking    :  hello_world.elf
text           data     bss     dec     hex filename
7944            112    2440   10496    2900 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=hbird BOARD=hbird_eval CORE=e203 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).

[12] http://ttssh2.osdn.jp/

[13] https://help.ubuntu.com/community/Minicom

2.3.3 Run Application

If the application is built successfully for this board *HummingBird Evaluation Kit* (page 122), then you can run it using this command line:

```
make SOC=hbird BOARD=hbird_eval CORE=e203 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 --pipe -f ../../../SoC/hbird/Board/hbi
         --batch -ex "monitor reset halt" -ex "monitor halt" -ex "monitor fl
 resume" -ex "monitor shutdown" -ex "quit"
 D:\Nuclei\gcc\bin\riscv-nuclei-elf-gdb.exe: warning: Couldn't determine a p
 Nuclei OpenOCD, 64-bit Open On-Chip Debugger 0.10.0+dev-00014-g0eae03214 (2
 Licensed under GNU GPL v2
 For bug reports, read
         http://openocd.org/doc/doxygen/bugs.html
 system_default_interrupt_handler (mcause=3735928559, sp=<optimized out>) at88
 188         printf("MTVAL : 0x%lx\r\n", __RV_CSR_READ(CSR_MBADADDR));
 JTAG tap: riscv.cpu tap/device found: 0x1e200a6d (mfg: 0x536 (Nuclei System
 halted at 0x8000050c due to debug interrupt
 cleared protection for sectors 0 through 63 on flash bank 0

 Loading section .init, size 0xc4 lma 0x80000000
 Loading section .text, size 0x1c6e lma 0x80000100
 Loading section .rodata, size 0x1ec lma 0x80001d70
 Loading section .data, size 0x70 lma 0x80001f5c
 Start address 0x80000000, load size 8078
 Transfer rate: 45 KB/sec, 2019 bytes/write.
 halted at 0x80000004 due to step
 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]
```

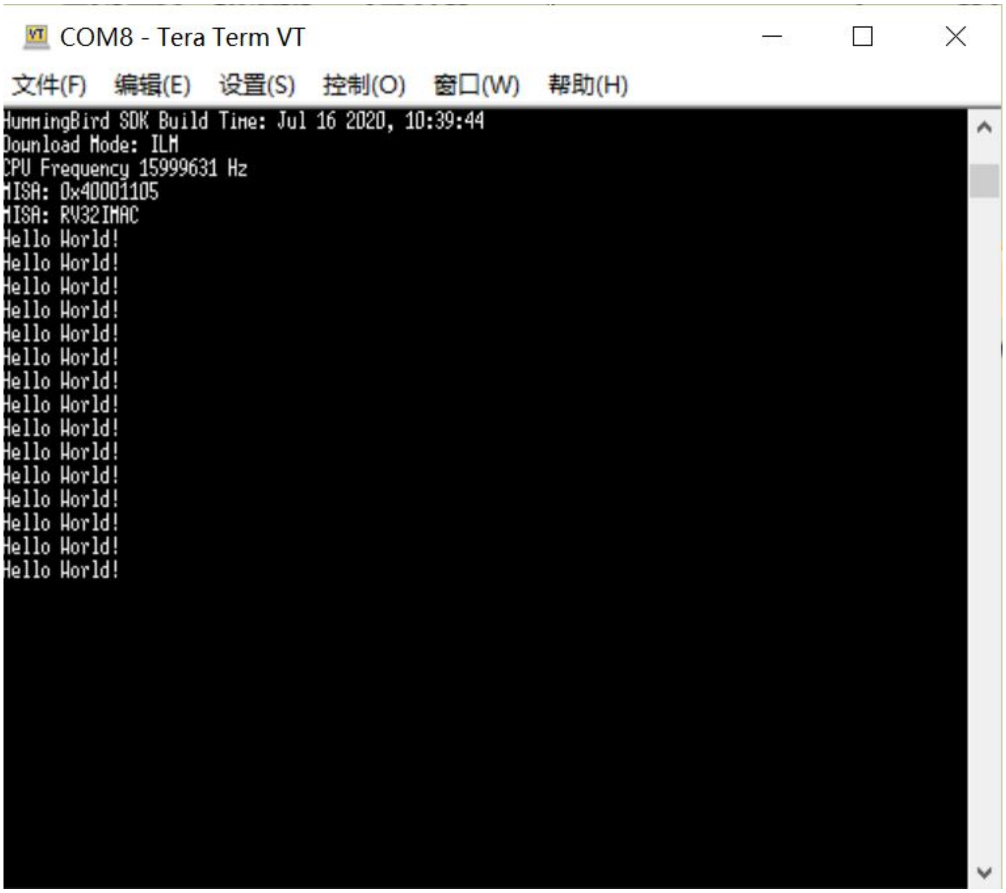
As you can see the application is uploaded successfully using `openocd` and `gdb`, then you can check the output in your UART terminal, see *HummingBird SDK Hello World Application UART Output* (page 13).

2.3.4 Debug Application

If the application is built successfully for this board *HummingBird Evaluation Kit* (page 122), then you can debug it using this command line:

```
make SOC=hbird BOARD=hbird_eval CORE=e203 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.



```
HummingBird SDK Build Time: Jul 16 2020, 10:39:44
Download Mode: ILM
CPU Frequency 15999631 Hz
MISA: 0x40001105
MISA: RV32IMAC
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: HummingBird SDK Hello World Application UART Output

```
"Download and debug hello_world.elf"
riscv-nuclei-elf-gdb hello_world.elf -ex "set remotetimeout 240" \
        -ex "target remote | openocd --pipe -f ../../../SoC/hbird/Board/hbi
D:\Nuclei\gcc\bin\riscv-nuclei-elf-gdb.exe: warning: Couldn't determine a p
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.htm
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-e
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/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
--Type <RET> for more, q to quit, c to continue without paging--
Reading symbols from hello_world.elf...
Remote debugging using | openocd --pipe -f ../../../SoC/hbird/Board/hbird_e
Nuclei OpenOCD, 64-bit Open On-Chip Debugger 0.10.0+dev-00014-g0eae03214 (2
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
system_default_interrupt_handler (mcause=3735928559, sp=<optimized out>)
    at ../../../SoC/hbird/Common/Source/system_hbird.c:188
188         printf("MTVAL : 0x%lx\r\n", __RV_CSR_READ(CSR_MBADADDR));
```

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
```

(continues on next page)

(continued from previous page)

```
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\hbird-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[14].
- 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.

2.4 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`

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

  - Open the `hello_world.c` in `application/baremetal/hello`, and replace the content using code below:

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

    int main(void)
    {
        printf("Hello World from HummingBird 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 8) 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:

    ```
    TARGET = hello

    HBIRD_SDK_ROOT = ../../..

    SRCDIRS = .

    INCDIRS = .

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

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

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

    int main(void)
    {
        printf("Hello World from HummingBird 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 8) to run or debug this new application.

Note:

- Please refer to *Application Development* (page 34) and *Build System based on Makefile* (page 19) for more information.

- If you want to access SoC related APIs, please use `hbird_sdk_soc.h` header file.
- If you want to access SoC and board related APIs, please use `hbird_sdk_hal.h` header file.
- For simplified application development, you can use `hbird_sdk_hal.h` directly.

2.5 Advanced Usage

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

- Click *Design and Architecture* (page 45) to learn about HummingBird SDK Design and Architecture, Board and SoC support documentation.
- Click *Developer Guide* (page 19) to learn about HummingBird SDK Build System and Application Development.
- Click *Application* (page 130) to learn about each application usage and expected output.

Note:

- If you met some issues in using this guide, please check *FAQ* (page 149), if still not solved, please *Submit your issue* (page 43).
- If you want to develop HummingBird SDK application in Nuclei Studio, you can also easily integrate the source code with it.
  1. Add required source code folders, and header file folders in IDE
  2. Check the compiler and linker options using extra **V=1** passed with *make*, and adapt the options in IDE
  3. Add extra macros definition and include folders in project configurations
  4. Build and debug project in IDE

CHAPTER

THREE

DEVELOPER GUIDE

3.1 Code Style

In HummingBird SDK, we use EditorConfig[15] to maintain our development coding styles.

Our editorconfig file[16] is maintained in the root directory of HummingBird SDK.

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

We use doxygen[17] to comment C/C++ source code.

3.2 Build System based on Makefile

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

3.2.1 Makefile Structure

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

This extra **<HBIRD_SDK_ROOT>/Makefile** introduce a new Make variable called **PROGRAM** to provide the ability to build or run application in **<HBIRD_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 **<HBIRD_SDK_ROOT>/Build** directory content list as below:

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

(continues on next page)

[15] https://editorconfig.org/

[16] https://github.com/riscv-mcu/hbird-sdk/tree/master/.editorconfig

[17] http://www.doxygen.nl/manual/docblocks.html

(continued from previous page)

```
Makefile.rules
Makefile.soc
```

The file or directory is used explained as below:

Makefile.base

This **Makefile.base** file is used as HummingBird SDK build system entry file, application's Makefile need to include this file to use all the features of HummingBird 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)[18], 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.
  - **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.

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

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 23)
- Collect all the C/C++/ASM compiling and link options

Makefile.components

This **Makefile.components** will include `build.mk` Makefiles of selected components defined via makefile variable *MIDDLEWARE* (page 29), the Makefiles are placed in the sub-folders of **<HBIRD_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 **<HBIRD_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.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
- Include **Makefile.files.<SOC>** which will include all the source code related to the *SOC* (page 25) and *BOARD* (page 26)
  - **Makefile.files.hbird**: Used to include source code for *HummingBird SoC* (page 118)

Makefile.soc

This **Makefile.soc** will include valid makefiles located in **<HBIRD_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 *HummingBird SoC* (page 118), we can support all the modes defined in *DOWNLOAD* (page 26), and **CORE** list defined in *Makefile.core* (page 23)
  - For *HummingBird SoC V2* (page 121), we can support all the modes defined in *DOWNLOAD* (page 26), and **CORE** list defined in *Makefile.core* (page 23)
- 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 `hbirdv2` as example:

```
SoC/hbirdv2
├── Board
│   └── hbird_fpga_eval
│       ├── Include
│       │   ├── board_hbird_fpga_eval.h
│       │   └── hbird_sdk_hal.h
│       ├── Source
│       │   └── GCC
│       └── openocd_hbirdv2.cfg
├── build.mk
└── Common
    ├── Include
    │   ├── hbirdv2.h
    │   ├── ... ...
    │   ├── hbirdv2_uart.h
    │   ├── hbird_sdk_soc.h
    │   └── system_hbirdv2.h
    └── Source
        ├── Drivers
        │   ├── ... ...
        │   └── hbirdv2_uart.c
        ├── GCC
        │   ├── intexc_hbirdv2.S
        │   └── startup_hbirdv2.S
        ├── Stubs
        │   ├── read.c
        │   ├── ... ...
        │   └── write.c
        ├── hbirdv2_common.c
        └── system_hbirdv2.c
```

Makefile.rtos

This **Makefile.rtos** will include **<HBIRD_SDK_ROOT>/OS/<RTOS>/build.mk** according to our *RTOS* (page 29) 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 29) 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.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 **e207**, and the **e207**'s **ARCH** and **ABI** are `rv32imafdc` and `ilp32d`, then you can add a new line like this `E207_CORE_ARCH_ABI = rv32imafdc ilp32d`, and append **e207** to **SUPPORTED_CORES** like this `SUPPORTED_CORES = e201 e201e e203 e205 e205f e205fd e207`

Note:

- The appended new **CORE** need to lower-case, e.g. *e207*
- The new defined variable **E207_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 **<HBIRD_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 *HummingBird Evaluation Kit* (page 122), you can create the **<HBIRD_SDK_ROOT>/Build/Makefile.global** as below:

```
SOC ?= hbird
BOARD ?= hbird_eval
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 *HummingBird Evaluation Kit* (page 122) board, e.g.
  - Build and run application for *HummingBird Evaluation Kit* (page 122): `make run`
  - Debug application for *HummingBird Evaluation Kit* (page 122): `make debug`
- If you create the **Makefile.global** like above sample code, you will also be able to use HummingBird 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=hbird BOARD=hbird_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 *HummingBird Evaluation Kit* (page 122), you can create `application/baremetal/helloworld/Makefile.local` as below:

```
SOC ?= hbird
BOARD ?= hbird_eval
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 HummingBird SDK Build System* (page 24).

Table 1: Make targets supported by HummingBird SDK Build System

target	description
help	display help message of HummingBird 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

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 27)

3.2.3 Makefile variables passed by make command

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

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

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 **<HBIRD_SDK_ROOT>/SoC** directory.

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

Table 2: Supported SoCs

SOC	Reference
hbird	*HummingBird SoC* (page 118)
hbirdv2	*HummingBird SoC V2* (page 121)

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 **<HBIRD_SDK_ROOT>/<SOC>/Board/** directory.

- *Supported Boards when SOC=hbird* (page 26)
- *Supported Boards when SOC=hbirdv2* (page 26)

Currently we support the following Boards.

Table 3: Supported Boards when SOC=hbird

BOARD	Reference
hbird_eval	*HummingBird Evaluation Kit* (page 122)

Table 4: Supported Boards when SOC=hbirdv2

BOARD	Reference
hbird_ddr_200t	*DDR200T Evaluation Kit* (page 124)
hbird_mcu_200t	*MCU200T Evaluation Kit* (page 126)

Note:

- If you only specify **SOC** variable in make command, it will use default **BOARD** and **CORE** option defined in Makefile.soc.<SOC>

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 26)

Table 5: Supported download modes

DOWNLOAD	Description
ilm	Program will be download into ilm/ram and run directly in ilm/ram, program 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

Note:

- *HummingBird SoC* (page 118) support all the download modes.

- **flashxip** mode in *HummingBird SoC* (page 118) is very slow due to the CORE frequency is very slow, and Flash speed is slow

CORE

CORE variable is used to declare the HummingBird RISC-V processor core of the application.

Currently it has these cores supported as described in table *Supported HummingBird RISC-V Processor cores* (page 27).

Table 6: Supported HummingBird RISC-V Processor cores

CORE	ARCH	ABI
e203e	rv32eac	ilp32e
e203	rv32imac	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 HummingBird 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 `hbird_eval` board for example, such as port `3344`:

- Open openocd server: `make SOC=hbird BOARD=hbird_eval CORE=e203 GDB_PORT=3344 run_openocd`
- connect gdb with openocd server: `make SOC=hbird BOARD=hbird_eval CORE=e203 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 HBIRD_BANNER=1 in hbird_sdk_hal.h.

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

The banner message looks like this:

```
HummingBird 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/timer_test/Makefile.

- *TARGET* (page 29)
- *HBIRD_SDK_ROOT* (page 29)
- *RTOS* (page 29)
- *MIDDLEWARE* (page 29)
- *PFLOAT* (page 29)
- *NEWLIB* (page 30)
- *NOGC* (page 30)
- *RTTHREAD_MSH* (page 30)

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.

HBIRD_SDK_ROOT

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

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

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

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 don't want disable this GC feature, you can set **NOGC=1**, GC feature will remove sections for you, but sometimes it might remove sections that are useful, e.g. For HummingBird SDK test cases, we use ctest framework, and we need to set **NOGC=1** to disable GC feature.

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.

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

- *INCDIRS* (page 31)
- *C_INCDIRS* (page 31)
- *CXX_INCDIRS* (page 31)
- *ASM_INCDIRS* (page 31)
- *SRCDIRS* (page 32)
- *C_SRCDIRS* (page 32)
- *CXX_SRCDIRS* (page 32)
- *ASM_SRCDIRS* (page 32)
- *C_SRCS* (page 32)
- *CXX_SRCS* (page 33)

- *ASM_SRCS* (page 33)
- *COMMON_FLAGS* (page 33)
- *CFLAGS* (page 33)
- *CXXFLAGS* (page 33)
- *ASMFLAGS* (page 33)
- *LDFLAGS* (page 34)
- *LDLIBS* (page 34)
- *LIBDIRS* (page 34)
- *LINKER_SCRIPT* (page 34)

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 **$(HBIRD_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 = $(HBIRD_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 HummingBird SDK application.

To develop a HummingBird 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

HBIRD_SDK_ROOT = path/to/your_hbird_sdk_root

SRCDIRS = .

INCDIRS = .

include $(HBIRD_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 `hbird_sdk_soc.h` in your application code.
   - If you just want to SoC and Board related resource, you can include header file `hbird_sdk_hal.h` in your application code.
   - For simplity, we recomment you to use `hbird_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 32): Add C/CPP/ASM source code located in the directories defined by this variable.
- *C_SRCDIRS* (page 32): Add C only source code located in the directories defined by this variable.
- *CXX_SRCDIRS* (page 32): Add CPP only source code located in the directories defined by this variable.
- *ASM_SRCDIRS* (page 32): Add ASM only source code located in the directories defined by this variable.

To add only selected source code in directory

- *C_SRCS* (page 32): Add C only source code files defined by this variable.
- *CXX_SRCS* (page 33): Add CPP only source code files defined by this variable.
- *ASM_SRCS* (page 33): 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 31): Include the directories defined by this variable for C/ASM/CPP code during compiling.
- *C_INCDIRS* (page 31): Include the directories defined by this variable for C only code during compiling.
- *CXX_INCDIRS* (page 31): Include the directories defined by this variable for CPP only code during compiling.
- *ASM_INCDIRS* (page 31): 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 33): This will add compiling flags for C/CPP/ASM source code.
- *CFLAGS* (page 33): This will add compiling flags for C source code.
- *CXXFLAGS* (page 33): This will add compiling flags for CPP source code.
- *ASMFLAGS* (page 33): This will add compiling flags for ASM source code.
- *LDFLAGS* (page 34): This will add linker flags when linking.

- *LDLIBS* (page 34): This will add extra libraries need to be linked.
- *LIBDIRS* (page 34): 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_plic* (page 133) for example usage of optimize for code size.

For more details about gcc optimization, please refer to Options That Control Optimization in GCC[19].

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 34) variable to set your linker script.

3.3.7 Set Default Make Options

Set Default Global Make Options For HummingBird SDK

If you want to change the global Make options for the HummingBird 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 HummingBird SDK Documentation

In HummingBird 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.

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

```
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 HummingBird SDK project is always welcome.

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

- *Port your HummingBird SoC into HummingBird SDK* (page 39)
- *Submit your issue* (page 43)
- *Submit your pull request* (page 43)

4.1 Port your HummingBird SoC into HummingBird SDK

If you want to port you HummingBird RISC-V Processor Core based Board to HummingBird SDK, you need to follow these steps:

Assume your SoC name is `ncstar`, based on HummingBird RISC-V core **e203**, 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 HummingBird 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
│   ├── hbird_sdk_soc.h
│   └── system_ncstar.h
└── Source
    ├── Drivers
    │   ├── peripheral_or_device_sources.c
    │   └── ......
    ├── GCC
    │   ├── intexc_ncstar.S
    │   └── startup_ncstar.S
    ├── Stubs
    │   ├── clock_getres.c
```

(continues on next page)

(continued from previous page)

```
    │   ├── clock_gettime.c
    │   ├── clock_settime.c
    │   ├── close.c
    │   ├── execve.c
    │   ├── 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 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
- 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 *hbirdv2* SoC.
- The **hbird_sdk_soc.h** file is very important, it is a HummingBird RISC-V 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
    │   └── hbird_sdk_hal.h
    ├── openocd_ncstar.cfg
    └── Source
```

(continues on next page)

(continued from previous page)

```
        ├── GCC
        │   └── gcc_ncstar_flashxip.ld
        └── ncstar_eval.c
```

Note:

- **The ncstar_eval** is the board folder name, if you have a new board, you can create a new folder in the same level
- **Include** folder contains the board related header files
- **Source** folder contains the board related source files
- **GCC** folder is optional, if your linker script for the board is different to the SoC, you need to put your linker script here
- **openocd_ncstar.cfg** file is the board related openocd debug configuration file
- **ncstar_eval.h** file contains board related definition or APIs and also include the **SoC** header file, you can refer to previously supported board such as `hbird_eval`
- **hbird_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 Makefiles related to `ncstar` in *HummingBird 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

HBIRD_SDK_SOC_BOARD := $(HBIRD_SDK_SOC)/Board/$(BOARD)
HBIRD_SDK_SOC_COMMON := $(HBIRD_SDK_SOC)/Common

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

RISCV_ARCH ?= rv32imac
RISCV_ABI ?= ilp32

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

INCDIRS += $(HBIRD_SDK_SOC_COMMON)/Include

C_SRCDIRS += $(HBIRD_SDK_SOC_COMMON)/Source \
             $(HBIRD_SDK_SOC_COMMON)/Source/Drivers \
             $(HBIRD_SDK_SOC_COMMON)/Source/Stubs
```

(continues on next page)

(continued from previous page)

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

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

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 hbird SoC.
- The hbird SoC is a FPGA based evaluation platform, it have ilm and dlm, so it support three *download modes* (page 26)
- The **hbird_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 **hbird_sdk_hal.h** must be created in Board include directory, it must include **hbird_sdk_soc.h** and board related header files.

4.2 Submit your issue

If you find any issue related to HummingBird SDK project, you can open an issue in https://github.com/riscv-mcu/hbird-sdk/issues

4.3 Submit your pull request

If you want to contribute your code to HummingBird SDK project, you can open an pull request in https://github.com/riscv-mcu/hbird-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

HummingBird SDK is developed based on Modified **NMSIS**, all the SoCs supported in it are following the Modified NMSIS-Core Device Templates Guidance.

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

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

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

For example, we have ported HummingBird SDK to use Segger embedded Studio and PlatformIO.

5.1.1 Directory Structure

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

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

Here is the directory structure for this HummingBird SDK.

```
$HBIRD_SDK_ROOT
├── application
│   ├── baremetal
│   ├── freertos
│   ├── ucosii
│   └── rtthread
├── Build
│   ├── gmsl
│   ├── Makefile.base
│   ├── Makefile.conf
│   ├── Makefile.components
│   ├── Makefile.core
│   ├── Makefile.files
│   ├── Makefile.global
│   ├── Makefile.misc
│   ├── Makefile.rtos
│   ├── Makefile.rules
│   └── Makefile.soc
├── doc
```

(continues on next page)

(continued from previous page)

```
│   ├── build
│   ├── source
│   ├── Makefile
│   └── requirements.txt
├── NMSIS
│   ├── Core
│   ├── DSP
│   ├── NN
│   └── Library
├── OS
│   ├── FreeRTOS
│   ├── UCOSII
│   └── RTThread
├── SoC
│   ├── hbird
│   └── hbirdv2
├── test
│   ├── core
│   ├── ctest.h
│   ├── LICENSE
│   └── README.md
├── LICENSE
├── Makefile
├── NMSIS_VERSION
├── README.md
├── setup.bat
└── setup.sh
```

- **application**

  This directory contains all the application softwares for this HummingBird 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 HummingBird SDK, the directory name for SoC and its boards should always in lower case.

  Here we mainly support HummingBird processor cores running in Hummingbird FPGA evaluation board, the support package placed in *SoC/hbird/* and *SoC/hbirdv2/*.

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

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

- **Build**

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

- **NMSIS**

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

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

- **OS**

  This directory provided two RTOS package we suppported which are **FreeRTOS** and **UCOSII**.

- **LICENSE**

  HummingBird SDK license file.

- **NMSIS_VERSION**

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

- **Makefile**

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

- **setup.sh**

  HummingBird 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
```

- **setup.bat**

  HummingBird 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 HummingBird SDK project components is list as below:

- *HummingBird RISC-V Processor* (page 48): How HummingBird RISC-V Processor Core is used in HummingBird SDK
- *SoC* (page 118): How HummingBird RISC-V processor code based SoC device is supported in HummingBird SDK
- *Board* (page 122): How HummingBird RISC-V based SoC's Board is supported in HummingBird SDK
- *Peripheral* (page 127): How to use the peripheral driver in HummingBird SDK
- *RTOS* (page 128): What RTOSes are supported in HummingBird SDK
- *Application* (page 130): How to use pre-built applications in HummingBird SDK

5.2 HummingBird RISC-V Processor

HummingBird RISC-V 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 RISC-V Spec[20] to learn more about Official RISC-V Instruction Set Architecture.

[20] https://riscv.org/specifications/

5.2.1 Introduction

Open source HummingBird RISC-V Processor provides the following RISC-V Cores for AIoT:

- **E200 series:** Designed for ultra-low power consumption and embedded scenarios, perfectly replaces the arm Cortex-M series cores.

5.2.2 NMSIS in HummingBird SDK

This HummingBird SDK is built based on the **modified** NMSIS[21] framework, user can access *NMSIS Core API* (page 49), NMSIS DSP API[22] and NMSIS NN API[23] provided by NMSIS[24].

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

NMSIS Core For HummingBird RISC-V

NMSIS Core API

If you want to access doxygen generated NMSIS Core API, please click NMSIS Core Doxygen API Documentation.

Version Control

group `NMSIS_Core_VersionControl`

Version #define symbols for NMSIS release specific C/C++ source code.

We followed the semantic versioning 2.0.0[25] to control NMSIS version. The version format is **MAJOR.MINOR.PATCH**, increment the:

1. MAJOR version when you make incompatible API changes,
2. MINOR version when you add functionality in a backwards compatible manner, and
3. PATCH version when you make backwards compatible bug fixes.

The header file `nmsis_version.h` is included by each core header so that these definitions are available.

Example Usage for NMSIS Version Check:

```
#if defined(__NMSIS_VERSION) && (__NMSIS_VERSION >= 0x00010105)
   #warning "Yes, we have NMSIS 1.1.5 or later"
#else
   #error "We need NMSIS 1.1.5 or later!"
#endif
```

Note: This NMSIS-Core is modified to match requirements of HummingBird RISC-V Core

[21] https://github.com/Nuclei-Software/NMSIS
[22] https://doc.nucleisys.com/nmsis/dsp/api/index.html
[23] https://doc.nucleisys.com/nmsis/nn/api/index.html
[24] https://github.com/Nuclei-Software/NMSIS

Unnamed Group

`__HBIRD_RISCV_REV` (0x0100)

HummingBird RISC-V revision number.

Reversion number format: [15:8] revision number, [7:0] patch number

Defines

`__NMSIS_VERSION_MAJOR` (1U)

Represent the NMSIS major version.

The NMSIS major version can be used to differentiate between NMSIS major releases.

`__NMSIS_VERSION_MINOR` (0U)

Represent the NMSIS minor version.

The NMSIS minor version can be used to query a NMSIS release update including new features.

`__NMSIS_VERSION_PATCH` (1U)

Represent the NMSIS patch version.

The NMSIS patch version can be used to show bug fixes in this package.

`__NMSIS_VERSION` ((*__NMSIS_VERSION_MAJOR* (page 50) << 16U) | (*__NMSIS_VERSION_MINOR* (page 50) << 8) | *__NMSIS_VERSION_PATCH* (page 50))

Represent the NMSIS Version.

NMSIS Version format: **MAJOR.MINOR.PATCH**

- MAJOR: *__NMSIS_VERSION_MAJOR* (page 50), stored in `bits [31:16]` of *__NMSIS_VERSION* (page 50)
- MINOR: *__NMSIS_VERSION_MINOR* (page 50), stored in `bits [15:8]` of *__NMSIS_VERSION* (page 50)
- PATCH: *__NMSIS_VERSION_PATCH* (page 50), stored in `bits [7:0]` of *__NMSIS_VERSION* (page 50)

Compiler Control

group `NMSIS_Core_CompilerControl`

Compiler agnostic #define symbols for generic c/c++ source code.

The NMSIS-Core provides the header file **nmsis_compiler.h** with consistent #define symbols for generate C or C++ source files that should be compiler agnostic. Each NMSIS compliant compiler should support the functionality described in this section.

The header file **nmsis_compiler.h** is also included by each Device Header File <device.h> so that these definitions are available.

[25] https://semver.org/

Defines

__has_builtin(x) (0)

__ASM __asm

Pass information from the compiler to the assembler.

__INLINE inline

Recommend that function should be inlined by the compiler.

__STATIC_INLINE static inline

Define a static function that may be inlined by the compiler.

__STATIC_FORCEINLINE __attribute__((always_inline)) static inline

Define a static function that should be always inlined by the compiler.

__NO_RETURN __attribute__((__noreturn__))

Inform the compiler that a function does not return.

__USED __attribute__((used))

Inform that a variable shall be retained in executable image.

__WEAK __attribute__((weak))

restrict pointer qualifier to enable additional optimizations.

__VECTOR_SIZE(x) __attribute__((vector_size(x)))

specified the vector size of the variable, measured in bytes

__PACKED __attribute__((packed, aligned(1)))

Request smallest possible alignment.

__PACKED_STRUCT struct __attribute__((packed, aligned(1)))

Request smallest possible alignment for a structure.

__PACKED_UNION union __attribute__((packed, aligned(1)))

Request smallest possible alignment for a union.

__UNALIGNED_UINT16_WRITE(addr, val) (void)((((struct *T_UINT16_WRITE* (page 52) *)(void *)(addr))->v) = (val))

Pointer for unaligned write of a uint16_t variable.

__UNALIGNED_UINT16_READ(addr) (((const struct *T_UINT16_READ* (page 52) *)(const void *)(addr))->v)

Pointer for unaligned read of a uint16_t variable.

__UNALIGNED_UINT32_WRITE(addr, val) (void)((((struct *T_UINT32_WRITE* (page 52) *)(void *)(addr))->v) = (val))

Pointer for unaligned write of a uint32_t variable.

__UNALIGNED_UINT32_READ(addr) (((const struct *T_UINT32_READ* (page 52) *)(const void *)(addr))->v)

Pointer for unaligned read of a uint32_t variable.

__ALIGNED(x) __attribute__((aligned(x)))

Minimum `x` bytes alignment for a variable.

__RESTRICT __restrict

restrict pointer qualifier to enable additional optimizations.

__COMPILER_BARRIER() *__ASM* (page 51) volatile("":::"memory")

Barrier to prevent compiler from reordering instructions.

__USUALLY(exp) __builtin_expect((exp), 1)

provide the compiler with branch prediction information, the branch is usually true

__RARELY(exp) __builtin_expect((exp), 0)

provide the compiler with branch prediction information, the branch is rarely true

__INTERRUPT

Use this attribute to indicate that the specified function is an interrupt handler.

Variables

__PACKED_STRUCT **T_UINT16_WRITE**

Packed struct for unaligned uint16_t write access.

__PACKED_STRUCT **T_UINT16_READ**

Packed struct for unaligned uint16_t read access.

__PACKED_STRUCT **T_UINT32_WRITE**

Packed struct for unaligned uint32_t write access.

__PACKED_STRUCT **T_UINT32_READ**

Packed struct for unaligned uint32_t read access.

Core CSR Register Access

group **NMSIS_Core_CSR_Register_Access**

Functions to access the Core CSR Registers.

The following functions or macros provide access to Core CSR registers.

- *Core CSR Encodings* (page 69)
- *Core CSR Registers* (page 57)

Defines

`__RV_CSR_SWAP`(csr, val)

CSR operation Macro for csrrw instruction.

Read the content of csr register to __v, then write content of val into csr register, then return __v

Parameters

- **`csr`** – CSR macro definition defined in *Core CSR Registers* (page 57), eg. *CSR_MSTATUS* (page 59)
- **`val`** – value to store into the CSR register

Returns the CSR register value before written

`__RV_CSR_READ`(csr)

CSR operation Macro for csrr instruction.

Read the content of csr register to __v and return it

Parameters

- **`csr`** – CSR macro definition defined in *Core CSR Registers* (page 57), eg. *CSR_MSTATUS* (page 59)

Returns the CSR register value

`__RV_CSR_WRITE`(csr, val)

CSR operation Macro for csrw instruction.

Write the content of val to csr register

Parameters

- **`csr`** – CSR macro definition defined in *Core CSR Registers* (page 57), eg. *CSR_MSTATUS* (page 59)
- **`val`** – value to store into the CSR register

`__RV_CSR_READ_SET`(csr, val)

CSR operation Macro for csrrs instruction.

Read the content of csr register to __v, then set csr register to be __v | val, then return __v

Parameters

- **`csr`** – CSR macro definition defined in *Core CSR Registers* (page 57), eg. *CSR_MSTATUS* (page 59)
- **`val`** – Mask value to be used wih csrrs instruction

Returns the CSR register value before written

`__RV_CSR_SET`(csr, val)

CSR operation Macro for csrs instruction.

Set csr register to be csr_content | val

Parameters

- **`csr`** – CSR macro definition defined in *Core CSR Registers* (page 57), eg. *CSR_MSTATUS* (page 59)
- **`val`** – Mask value to be used wih csrs instruction

__RV_CSR_READ_CLEAR(csr, val)

CSR operation Macro for csrrc instruction.

Read the content of csr register to __v, then set csr register to be __v & ~val, then return __v

Parameters

- **csr** – CSR macro definition defined in *Core CSR Registers* (page 57), eg. *CSR_MSTATUS* (page 59)
- **val** – Mask value to be used wih csrrc instruction

Returns the CSR register value before written

__RV_CSR_CLEAR(csr, val)

CSR operation Macro for csrc instruction.

Set csr register to be csr_content & ~val

Parameters

- **csr** – CSR macro definition defined in *Core CSR Registers* (page 57), eg. *CSR_MSTATUS* (page 59)
- **val** – Mask value to be used wih csrc instruction

Functions

__STATIC_FORCEINLINE void __enable_irq (void)

Enable IRQ Interrupts.

Enables IRQ interrupts by setting the MIE-bit in the MSTATUS Register.

Remark

Can only be executed in Privileged modes.

__STATIC_FORCEINLINE void __disable_irq (void)

Disable IRQ Interrupts.

Disables IRQ interrupts by clearing the MIE-bit in the MSTATUS Register.

Remark

Can only be executed in Privileged modes.

__STATIC_FORCEINLINE void __enable_ext_irq (void)

Enable External IRQ Interrupts.

Enables External IRQ interrupts by setting the MEIE-bit in the MIE Register.

Remark

Can only be executed in Privileged modes.

`__STATIC_FORCEINLINE void __disable_ext_irq (void)`

Disable External IRQ Interrupts.

Disables External IRQ interrupts by clearing the MEIE-bit in the MIE Register.

Remark

Can only be executed in Privileged modes.

`__STATIC_FORCEINLINE void __enable_timer_irq (void)`

Enable Timer IRQ Interrupts.

Enables Timer IRQ interrupts by setting the MTIE-bit in the MIE Register.

Remark

Can only be executed in Privileged modes.

`__STATIC_FORCEINLINE void __disable_timer_irq (void)`

Disable Timer IRQ Interrupts.

Disables Timer IRQ interrupts by clearing the MTIE-bit in the MIE Register.

Remark

Can only be executed in Privileged modes.

`__STATIC_FORCEINLINE void __enable_sw_irq (void)`

Enable software IRQ Interrupts.

Enables software IRQ interrupts by setting the MSIE-bit in the MIE Register.

Remark

Can only be executed in Privileged modes.

`__STATIC_FORCEINLINE void __disable_sw_irq (void)`

Disable software IRQ Interrupts.

Disables software IRQ interrupts by clearing the MSIE-bit in the MIE Register.

Remark

Can only be executed in Privileged modes.

`__STATIC_FORCEINLINE void __disable_core_irq (uint32_t irq)`

Disable Core IRQ Interrupt.

Disable Core IRQ interrupt by clearing the irq bit in the MIE Register.

Remark

Can only be executed in Privileged modes.

`__STATIC_FORCEINLINE void __enable_core_irq (uint32_t irq)`

Enable Core IRQ Interrupt.

Enable Core IRQ interrupt by setting the irq bit in the MIE Register.

Remark

Can only be executed in Privileged modes.

`__STATIC_FORCEINLINE uint32_t __get_core_irq_pending (uint32_t irq)`

Get Core IRQ Interrupt Pending status.

Get Core IRQ interrupt pending status of irq bit.

Remark

Can only be executed in Privileged modes.

`__STATIC_FORCEINLINE void __clear_core_irq_pending (uint32_t irq)`

Clear Core IRQ Interrupt Pending status.

Clear Core IRQ interrupt pending status of irq bit.

Remark

Can only be executed in Privileged modes.

`__STATIC_FORCEINLINE uint64_t __get_rv_cycle (void)`

Read whole 64 bits value of mcycle counter.

This function will read the whole 64 bits of MCYCLE register

Remark

It will work for both RV32 and RV64 to get full 64bits value of MCYCLE

Returns The whole 64 bits value of MCYCLE

`__STATIC_FORCEINLINE uint64_t __get_rv_instret (void)`

Read whole 64 bits value of machine instruction-retired counter.

This function will read the whole 64 bits of MINSTRET register

Remark

It will work for both RV32 and RV64 to get full 64bits value of MINSTRET

Returns The whole 64 bits value of MINSTRET

`__STATIC_FORCEINLINE uint64_t __get_rv_time (void)`

Read whole 64 bits value of real-time clock.

This function will read the whole 64 bits of TIME register

Remark

It will work for both RV32 and RV64 to get full 64bits value of TIME

Attention only available when user mode available

Returns The whole 64 bits value of TIME CSR

Core CSR Encoding

Core CSR Register Definitions

group `NMSIS_Core_CSR_Registers`

NMSIS Core CSR Register Definitions.

The following macros are used for CSR Register Defintions.

Defines

`CSR_USTATUS` 0x0

`CSR_FFLAGS` 0x1

`CSR_FRM` 0x2

`CSR_FCSR` 0x3

`CSR_CYCLE` 0xc00

`CSR_TIME` 0xc01

`CSR_INSTRET` 0xc02

CSR_HPMCOUNTER3 0xc03

CSR_HPMCOUNTER4 0xc04

CSR_HPMCOUNTER5 0xc05

CSR_HPMCOUNTER6 0xc06

CSR_HPMCOUNTER7 0xc07

CSR_HPMCOUNTER8 0xc08

CSR_HPMCOUNTER9 0xc09

CSR_HPMCOUNTER10 0xc0a

CSR_HPMCOUNTER11 0xc0b

CSR_HPMCOUNTER12 0xc0c

CSR_HPMCOUNTER13 0xc0d

CSR_HPMCOUNTER14 0xc0e

CSR_HPMCOUNTER15 0xc0f

CSR_HPMCOUNTER16 0xc10

CSR_HPMCOUNTER17 0xc11

CSR_HPMCOUNTER18 0xc12

CSR_HPMCOUNTER19 0xc13

CSR_HPMCOUNTER20 0xc14

CSR_HPMCOUNTER21 0xc15

CSR_HPMCOUNTER22 0xc16

CSR_HPMCOUNTER23 0xc17

`CSR_HPMCOUNTER24` 0xc18

`CSR_HPMCOUNTER25` 0xc19

`CSR_HPMCOUNTER26` 0xc1a

`CSR_HPMCOUNTER27` 0xc1b

`CSR_HPMCOUNTER28` 0xc1c

`CSR_HPMCOUNTER29` 0xc1d

`CSR_HPMCOUNTER30` 0xc1e

`CSR_HPMCOUNTER31` 0xc1f

`CSR_SSTATUS` 0x100

`CSR_SIE` 0x104

`CSR_STVEC` 0x105

`CSR_SSCRATCH` 0x140

`CSR_SEPC` 0x141

`CSR_SCAUSE` 0x142

`CSR_SBADADDR` 0x143

`CSR_SIP` 0x144

`CSR_SPTBR` 0x180

`CSR_MSTATUS` 0x300

`CSR_MISA` 0x301

`CSR_MEDELEG` 0x302

`CSR_MIDELEG` 0x303

CSR_MIE 0x304

CSR_MTVEC 0x305

CSR_MCOUNTEREN 0x306

CSR_MSCRATCH 0x340

CSR_MEPC 0x341

CSR_MCAUSE 0x342

CSR_MBADADDR 0x343

CSR_MIP 0x344

CSR_PMPCFG0 0x3a0

CSR_PMPCFG1 0x3a1

CSR_PMPCFG2 0x3a2

CSR_PMPCFG3 0x3a3

CSR_PMPADDR0 0x3b0

CSR_PMPADDR1 0x3b1

CSR_PMPADDR2 0x3b2

CSR_PMPADDR3 0x3b3

CSR_PMPADDR4 0x3b4

CSR_PMPADDR5 0x3b5

CSR_PMPADDR6 0x3b6

CSR_PMPADDR7 0x3b7

CSR_PMPADDR8 0x3b8

CSR_PMPADDR9 0x3b9

CSR_PMPADDR10 0x3ba

CSR_PMPADDR11 0x3bb

CSR_PMPADDR12 0x3bc

CSR_PMPADDR13 0x3bd

CSR_PMPADDR14 0x3be

CSR_PMPADDR15 0x3bf

CSR_TSELECT 0x7a0

CSR_TDATA1 0x7a1

CSR_TDATA2 0x7a2

CSR_TDATA3 0x7a3

CSR_DCSR 0x7b0

CSR_DPC 0x7b1

CSR_DSCRATCH 0x7b2

CSR_MCYCLE 0xb00

CSR_MINSTRET 0xb02

CSR_MHPMCOUNTER3 0xb03

CSR_MHPMCOUNTER4 0xb04

CSR_MHPMCOUNTER5 0xb05

CSR_MHPMCOUNTER6 0xb06

CSR_MHPMCOUNTER7 0xb07

CSR_MHPMCOUNTER8 0xb08

CSR_MHPMCOUNTER9 0xb09

CSR_MHPMCOUNTER10 0xb0a

CSR_MHPMCOUNTER11 0xb0b

CSR_MHPMCOUNTER12 0xb0c

CSR_MHPMCOUNTER13 0xb0d

CSR_MHPMCOUNTER14 0xb0e

CSR_MHPMCOUNTER15 0xb0f

CSR_MHPMCOUNTER16 0xb10

CSR_MHPMCOUNTER17 0xb11

CSR_MHPMCOUNTER18 0xb12

CSR_MHPMCOUNTER19 0xb13

CSR_MHPMCOUNTER20 0xb14

CSR_MHPMCOUNTER21 0xb15

CSR_MHPMCOUNTER22 0xb16

CSR_MHPMCOUNTER23 0xb17

CSR_MHPMCOUNTER24 0xb18

CSR_MHPMCOUNTER25 0xb19

CSR_MHPMCOUNTER26 0xb1a

CSR_MHPMCOUNTER27 0xb1b

CSR_MHPMCOUNTER28 0xb1c

CSR_MHPMCOUNTER29 0xb1d

CSR_MHPMCOUNTER30 0xb1e

CSR_MHPMCOUNTER31 0xb1f

CSR_MUCOUNTEREN 0x320

CSR_MSCOUNTEREN 0x321

CSR_MHPMEVENT3 0x323

CSR_MHPMEVENT4 0x324

CSR_MHPMEVENT5 0x325

CSR_MHPMEVENT6 0x326

CSR_MHPMEVENT7 0x327

CSR_MHPMEVENT8 0x328

CSR_MHPMEVENT9 0x329

CSR_MHPMEVENT10 0x32a

CSR_MHPMEVENT11 0x32b

CSR_MHPMEVENT12 0x32c

CSR_MHPMEVENT13 0x32d

CSR_MHPMEVENT14 0x32e

CSR_MHPMEVENT15 0x32f

CSR_MHPMEVENT16 0x330

CSR_MHPMEVENT17 0x331

CSR_MHPMEVENT18 0x332

`CSR_MHPMEVENT19` 0x333

`CSR_MHPMEVENT20` 0x334

`CSR_MHPMEVENT21` 0x335

`CSR_MHPMEVENT22` 0x336

`CSR_MHPMEVENT23` 0x337

`CSR_MHPMEVENT24` 0x338

`CSR_MHPMEVENT25` 0x339

`CSR_MHPMEVENT26` 0x33a

`CSR_MHPMEVENT27` 0x33b

`CSR_MHPMEVENT28` 0x33c

`CSR_MHPMEVENT29` 0x33d

`CSR_MHPMEVENT30` 0x33e

`CSR_MHPMEVENT31` 0x33f

`CSR_MVENDORID` 0xf11

`CSR_MARCHID` 0xf12

`CSR_MIMPID` 0xf13

`CSR_MHARTID` 0xf14

`CSR_CYCLEH` 0xc80

`CSR_TIMEH` 0xc81

`CSR_INSTRETH` 0xc82

`CSR_HPMCOUNTER3H` 0xc83

`CSR_HPMCOUNTER4H` 0xc84

`CSR_HPMCOUNTER5H` 0xc85

`CSR_HPMCOUNTER6H` 0xc86

`CSR_HPMCOUNTER7H` 0xc87

`CSR_HPMCOUNTER8H` 0xc88

`CSR_HPMCOUNTER9H` 0xc89

`CSR_HPMCOUNTER10H` 0xc8a

`CSR_HPMCOUNTER11H` 0xc8b

`CSR_HPMCOUNTER12H` 0xc8c

`CSR_HPMCOUNTER13H` 0xc8d

`CSR_HPMCOUNTER14H` 0xc8e

`CSR_HPMCOUNTER15H` 0xc8f

`CSR_HPMCOUNTER16H` 0xc90

`CSR_HPMCOUNTER17H` 0xc91

`CSR_HPMCOUNTER18H` 0xc92

`CSR_HPMCOUNTER19H` 0xc93

`CSR_HPMCOUNTER20H` 0xc94

`CSR_HPMCOUNTER21H` 0xc95

`CSR_HPMCOUNTER22H` 0xc96

`CSR_HPMCOUNTER23H` 0xc97

`CSR_HPMCOUNTER24H` 0xc98

`CSR_HPMCOUNTER25H` 0xc99

`CSR_HPMCOUNTER26H` 0xc9a

`CSR_HPMCOUNTER27H` 0xc9b

`CSR_HPMCOUNTER28H` 0xc9c

`CSR_HPMCOUNTER29H` 0xc9d

`CSR_HPMCOUNTER30H` 0xc9e

`CSR_HPMCOUNTER31H` 0xc9f

`CSR_MCYCLEH` 0xb80

`CSR_MINSTRETH` 0xb82

`CSR_MHPMCOUNTER3H` 0xb83

`CSR_MHPMCOUNTER4H` 0xb84

`CSR_MHPMCOUNTER5H` 0xb85

`CSR_MHPMCOUNTER6H` 0xb86

`CSR_MHPMCOUNTER7H` 0xb87

`CSR_MHPMCOUNTER8H` 0xb88

`CSR_MHPMCOUNTER9H` 0xb89

`CSR_MHPMCOUNTER10H` 0xb8a

`CSR_MHPMCOUNTER11H` 0xb8b

`CSR_MHPMCOUNTER12H` 0xb8c

`CSR_MHPMCOUNTER13H` 0xb8d

`CSR_MHPMCOUNTER14H` 0xb8e

CSR_MHPMCOUNTER15H 0xb8f

CSR_MHPMCOUNTER16H 0xb90

CSR_MHPMCOUNTER17H 0xb91

CSR_MHPMCOUNTER18H 0xb92

CSR_MHPMCOUNTER19H 0xb93

CSR_MHPMCOUNTER20H 0xb94

CSR_MHPMCOUNTER21H 0xb95

CSR_MHPMCOUNTER22H 0xb96

CSR_MHPMCOUNTER23H 0xb97

CSR_MHPMCOUNTER24H 0xb98

CSR_MHPMCOUNTER25H 0xb99

CSR_MHPMCOUNTER26H 0xb9a

CSR_MHPMCOUNTER27H 0xb9b

CSR_MHPMCOUNTER28H 0xb9c

CSR_MHPMCOUNTER29H 0xb9d

CSR_MHPMCOUNTER30H 0xb9e

CSR_MHPMCOUNTER31H 0xb9f

CSR_MTVT 0x307

CSR_MNXTI 0x345

CSR_MINTSTATUS 0x346

CSR_MSCRATCHCSW 0x348

`CSR_MSCRATCHCSWL` 0x349

`CSR_MCLICBASE` 0x350

`CSR_MCOUNTINHIBIT` 0x320

`CSR_MNVEC` 0x7C3

`CSR_MSUBM` 0x7C4

`CSR_MDCAUSE` 0x7C9

`CSR_MCACHE_CTL` 0x7CA

`CSR_MMISC_CTL` 0x7D0

`CSR_MSAVESTATUS` 0x7D6

`CSR_MSAVEEPC1` 0x7D7

`CSR_MSAVECAUSE1` 0x7D8

`CSR_MSAVEEPC2` 0x7D9

`CSR_MSAVECAUSE2` 0x7DA

`CSR_MSAVEDCAUSE1` 0x7DB

`CSR_MSAVEDCAUSE2` 0x7DC

`CSR_PUSHMSUBM` 0x7EB

`CSR_MTVT2` 0x7EC

`CSR_JALMNXTI` 0x7ED

`CSR_PUSHMCAUSE` 0x7EE

`CSR_PUSHMEPC` 0x7EF

`CSR_SLEEPVALUE` 0x811

CSR_TXEVT 0x812

CSR_WFE 0x810

Other Core Related Macros

group **NMSIS_Core_CSR_Encoding**

NMSIS Core CSR Encodings.

The following macros are used for CSR encodings

Defines

MSTATUS_UIE 0x00000001

MSTATUS_SIE 0x00000002

MSTATUS_HIE 0x00000004

MSTATUS_MIE 0x00000008

MSTATUS_UPIE 0x00000010

MSTATUS_SPIE 0x00000020

MSTATUS_HPIE 0x00000040

MSTATUS_MPIE 0x00000080

MSTATUS_SPP 0x00000100

MSTATUS_MPP 0x00001800

MSTATUS_FS 0x00006000

MSTATUS_XS 0x00018000

MSTATUS_MPRV 0x00020000

MSTATUS_PUM 0x00040000

MSTATUS_MXR 0x00080000

MSTATUS_VM 0x1F000000

MSTATUS32_SD 0x80000000

MSTATUS64_SD 0x8000000000000000

MSTATUS_FS_INITIAL 0x00002000

MSTATUS_FS_CLEAN 0x00004000

MSTATUS_FS_DIRTY 0x00006000

SSTATUS_UIE 0x00000001

SSTATUS_SIE 0x00000002

SSTATUS_UPIE 0x00000010

SSTATUS_SPIE 0x00000020

SSTATUS_SPP 0x00000100

SSTATUS_FS 0x00006000

SSTATUS_XS 0x00018000

SSTATUS_PUM 0x00040000

SSTATUS32_SD 0x80000000

SSTATUS64_SD 0x8000000000000000

CSR_MCACHE_CTL_IE 0x00000001

CSR_MCACHE_CTL_DE 0x00010000

DCSR_XDEBUGVER (3U<<30)

DCSR_NDRESET (1<<29)

DCSR_FULLRESET (1<<28)

DCSR_EBREAKM (1<<15)

DCSR_EBREAKH (1<<14)

DCSR_EBREAKS (1<<13)

DCSR_EBREAKU (1<<12)

DCSR_STOPCYCLE (1<<10)

DCSR_STOPTIME (1<<9)

DCSR_CAUSE (7<<6)

DCSR_DEBUGINT (1<<5)

DCSR_HALT (1<<3)

DCSR_STEP (1<<2)

DCSR_PRV (3<<0)

DCSR_CAUSE_NONE 0

DCSR_CAUSE_SWBP 1

DCSR_CAUSE_HWBP 2

DCSR_CAUSE_DEBUGINT 3

DCSR_CAUSE_STEP 4

DCSR_CAUSE_HALT 5

MCONTROL_TYPE(xlen) (0xfULL<<((xlen)-4))

MCONTROL_DMODE(xlen) (1ULL<<((xlen)-5))

MCONTROL_MASKMAX(xlen) (0x3fULL<<((xlen)-11))

MCONTROL_SELECT (1<<19)

MCONTROL_TIMING (1<<18)

MCONTROL_ACTION (0x3f<<12)

MCONTROL_CHAIN (1<<11)

MCONTROL_MATCH (0xf<<7)

MCONTROL_M (1<<6)

MCONTROL_H (1<<5)

MCONTROL_S (1<<4)

MCONTROL_U (1<<3)

MCONTROL_EXECUTE (1<<2)

MCONTROL_STORE (1<<1)

MCONTROL_LOAD (1<<0)

MCONTROL_TYPE_NONE 0

MCONTROL_TYPE_MATCH 2

MCONTROL_ACTION_DEBUG_EXCEPTION 0

MCONTROL_ACTION_DEBUG_MODE 1

MCONTROL_ACTION_TRACE_START 2

MCONTROL_ACTION_TRACE_STOP 3

MCONTROL_ACTION_TRACE_EMIT 4

MCONTROL_MATCH_EQUAL 0

MCONTROL_MATCH_NAPOT 1

MCONTROL_MATCH_GE 2

MCONTROL_MATCH_LT 3

MCONTROL_MATCH_MASK_LOW 4

MCONTROL_MATCH_MASK_HIGH 5

MCAUSE_INTERRUPT (1ULL<<((__riscv_xlen)-1))

MIP_SSIP (1 << *IRQ_S_SOFT* (page 74))

MIP_HSIP (1 << *IRQ_H_SOFT* (page 74))

MIP_MSIP (1 << *IRQ_M_SOFT* (page 75))

MIP_STIP (1 << *IRQ_S_TIMER* (page 75))

MIP_HTIP (1 << *IRQ_H_TIMER* (page 75))

MIP_MTIP (1 << *IRQ_M_TIMER* (page 75))

MIP_SEIP (1 << *IRQ_S_EXT* (page 75))

MIP_HEIP (1 << *IRQ_H_EXT* (page 75))

MIP_MEIP (1 << *IRQ_M_EXT* (page 75))

MIE_SSIE *MIP_SSIP* (page 73)

MIE_HSIE *MIP_HSIP* (page 73)

MIE_MSIE *MIP_MSIP* (page 73)

MIE_STIE *MIP_STIP* (page 73)

MIE_HTIE *MIP_HTIP* (page 73)

MIE_MTIE *MIP_MTIP* (page 73)

MIE_SEIE *MIP_SEIP* (page 73)

MIE_HEIE *MIP_HEIP* (page 73)

MIE_MEIE *MIP_MEIP* (page 73)

WFE_WFE 0x1

MCOUNTINHIBIT_IR (1<<2)

MCOUNTINHIBIT_CY (1<<0)

MMISC_CTL_NMI_CAUSE_FFF (1<<9)

MMISC_CTL_MISALIGN (1<<6)

MMISC_CTL_BPU (1<<3)

SIP_SSIP *MIP_SSIP* (page 73)

SIP_STIP *MIP_STIP* (page 73)

PRV_U 0

PRV_S 1

PRV_H 2

PRV_M 3

VM_MBARE 0

VM_MBB 1

VM_MBBID 2

VM_SV32 8

VM_SV39 9

VM_SV48 10

IRQ_S_SOFT 1

IRQ_H_SOFT 2

IRQ_M_SOFT 3

IRQ_S_TIMER 5

IRQ_H_TIMER 6

IRQ_M_TIMER 7

IRQ_S_EXT 9

IRQ_H_EXT 10

IRQ_M_EXT 11

IRQ_COP 12

IRQ_HOST 13

DEFAULT_RSTVEC 0x00001000

DEFAULT_NMIVEC 0x00001004

DEFAULT_MTVEC 0x00001010

CONFIG_STRING_ADDR 0x0000100C

EXT_IO_BASE 0x40000000

DRAM_BASE 0x80000000

FRM_RNDMODE_RNE 0x0
FPU Round to Nearest, ties to Even.

FRM_RNDMODE_RTZ 0x1
FPU Round Towards Zero.

FRM_RNDMODE_RDN 0x2
FPU Round Down (towards -inf)

FRM_RNDMODE_RUP 0x3
FPU Round Up (towards +inf)

FRM_RNDMODE_RMM 0x4

FPU Round to nearest, ties to Max Magnitude.

FRM_RNDMODE_DYN 0x7

In instruction's rm, selects dynamic rounding mode.

In Rounding Mode register, Invalid

FFLAGS_AE_NX (1<<0)

FPU Inexact.

FFLAGS_AE_UF (1<<1)

FPU Underflow.

FFLAGS_AE_OF (1<<2)

FPU Overflow.

FFLAGS_AE_DZ (1<<3)

FPU Divide by Zero.

FFLAGS_AE_NV (1<<4)

FPU Invalid Operation.

FREG(idx) f##idx

Floating Point Register f0-f31, eg.

f0 -> *FREG(0)* (page 76)

PMP_R 0x01

PMP_W 0x02

PMP_X 0x04

PMP_A 0x18

PMP_A_TOR 0x08

PMP_A_NA4 0x10

PMP_A_NAPOT 0x18

PMP_L 0x80

PMP_SHIFT 2

PMP_COUNT 16

PTE_V 0x001

PTE_R 0x002

PTE_W 0x004

PTE_X 0x008

PTE_U 0x010

PTE_G 0x020

PTE_A 0x040

PTE_D 0x080

PTE_SOFT 0x300

PTE_PPN_SHIFT 10

PTE_TABLE(PTE) (((PTE) & (*PTE_V* (page 77) | *PTE_R* (page 77) | *PTE_W* (page 77) | *PTE_X* (page 77))) == *PTE_V* (page 77))

CAUSE_MISALIGNED_FETCH 0x0

End of Doxygen Group NMSIS_Core_CSR_Registers.

CAUSE_FAULT_FETCH 0x1

CAUSE_ILLEGAL_INSTRUCTION 0x2

CAUSE_BREAKPOINT 0x3

CAUSE_MISALIGNED_LOAD 0x4

CAUSE_FAULT_LOAD 0x5

CAUSE_MISALIGNED_STORE 0x6

CAUSE_FAULT_STORE 0x7

CAUSE_USER_ECALL 0x8

`CAUSE_SUPERVISOR_ECALL` 0x9

`CAUSE_HYPERVISOR_ECALL` 0xa

`CAUSE_MACHINE_ECALL` 0xb

`DCAUSE_FAULT_FETCH_PMP` 0x1

`DCAUSE_FAULT_FETCH_INST` 0x2

`DCAUSE_FAULT_LOAD_PMP` 0x1

`DCAUSE_FAULT_LOAD_INST` 0x2

`DCAUSE_FAULT_LOAD_NICE` 0x3

`DCAUSE_FAULT_STORE_PMP` 0x1

`DCAUSE_FAULT_STORE_INST` 0x2

Register Define and Type Definitions

group `NMSIS_Core_Registers`

Type definitions and defines for core registers.

Defines

`__RISCV_XLEN` 32

Refer to the width of an integer register in bits(either 32 or 64)

Typedefs

typedef uint32_t **`rv_csr_t`**

Type of Control and Status Register(CSR), depends on the XLEN defined in RISC-V.

Core

group `NMSIS_Core_Base_Registers`

Type definitions and defines for base core registers.

union `CSR_MISA_Type`

#include <core_feature_base.h> Union type to access MISA register.

Public Members

rv_csr_t (page 78) **a**

bit: 0 Atomic extension

rv_csr_t (page 78) **b**

bit: 1 Tentatively reserved for Bit-Manipulation extension

rv_csr_t (page 78) **c**

bit: 2 Compressed extension

rv_csr_t (page 78) **d**

bit: 3 Double-precision floating-point extension

Type used for csr data access.

rv_csr_t (page 78) **e**

bit: 4 RV32E base ISA

rv_csr_t (page 78) **f**

bit: 5 Single-precision floating-point extension

rv_csr_t (page 78) **g**

bit: 6 Additional standard extensions present

rv_csr_t (page 78) **h**

bit: 7 Hypervisor extension

rv_csr_t (page 78) **i**

bit: 8 RV32I/64I/128I base ISA

rv_csr_t (page 78) **j**

bit: 9 Tentatively reserved for Dynamically Translated Languages extension

rv_csr_t (page 78) **_reserved1**

bit: 10 Reserved

rv_csr_t (page 78) **l**

bit: 11 Tentatively reserved for Decimal Floating-Point extension

rv_csr_t (page 78) **m**

bit: 12 Integer Multiply/Divide extension

rv_csr_t (page 78) **n**

bit: 13 User-level interrupts supported

rv_csr_t (page 78) **_reserved2**

bit: 14 Reserved

rv_csr_t (page 78) **p**

bit: 15 Tentatively reserved for Packed-SIMD extension

rv_csr_t (page 78) **q**

bit: 16 Quad-precision floating-point extension

rv_csr_t (page 78) **_resreved3**

bit: 17 Reserved

rv_csr_t (page 78) **s**

bit: 18 Supervisor mode implemented

rv_csr_t (page 78) **t**

bit: 19 Tentatively reserved for Transactional Memory extension

rv_csr_t (page 78) **u**

bit: 20 User mode implemented

rv_csr_t (page 78) **v**

bit: 21 Tentatively reserved for Vector extension

rv_csr_t (page 78) **_reserved4**

bit: 22 Reserved

rv_csr_t (page 78) **x**

bit: 23 Non-standard extensions present

rv_csr_t (page 78) **_reserved5**

bit: 24..29 Reserved

rv_csr_t (page 78) **mxl**

bit: 30..31 Machine XLEN

struct *CSR_MISA_Type* (page 79)::[anonymous] **b**

Structure used for bit access.

union **CSR_MSTATUS_Type**

#include <core_feature_base.h> Union type to access MSTATUS configure register.

Public Members

rv_csr_t (page 78) **_reserved0**

bit: 0 Reserved

rv_csr_t (page 78) **sie**

bit: 1 supervisor interrupt enable flag

rv_csr_t (page 78) **_reserved1**

bit: 2 Reserved

rv_csr_t (page 78) **mie**

bit: 3 Machine mode interrupt enable flag

rv_csr_t (page 78) **_reserved2**

bit: 4 Reserved

rv_csr_t (page 78) **spie**

bit: 3 Supervisor Privilede mode interrupt enable flag

rv_csr_t (page 78) **_reserved3**

bit: Reserved

rv_csr_t (page 78) **mpie**

bit: mirror of MIE flag

rv_csr_t (page 78) **_reserved4**

bit: Reserved

rv_csr_t (page 78) **mpp**

bit: mirror of Privilege Mode

rv_csr_t (page 78) **fs**

bit: FS status flag

rv_csr_t (page 78) **xs**

bit: XS status flag

rv_csr_t (page 78) **mprv**

bit: Machine mode PMP

rv_csr_t (page 78) **sum**

bit: Supervisor Mode load and store protection

rv_csr_t (page 78) **_reserved6**

bit: 19..30 Reserved

rv_csr_t (page 78) **sd**

bit: Dirty status for XS or FS

struct *CSR_MSTATUS_Type* (page 81)::[anonymous] **b**

Structure used for bit access.

rv_csr_t (page 78) **d**

Type used for csr data access.

union **CSR_MTVEC_Type**

#include <core_feature_base.h> Union type to access MTVEC configure register.

Public Members

rv_csr_t (page 78) **mode**

bit: 0..2 interrupt mode control

rv_csr_t (page 78) **addr**

bit: 3..31 mtvec address

struct *CSR_MTVEC_Type* (page 82)::[anonymous] **b**

Structure used for bit access.

rv_csr_t (page 78) **d**

Type used for csr data access.

union **CSR_MCAUSE_Type**

#include <core_feature_base.h> Union type to access MCAUSE configure register.

Public Members

rv_csr_t (page 78) **exccode**

bit: 11..0 exception or interrupt code

rv_csr_t (page 78) **_reserved0**

bit: 15..12 Reserved

rv_csr_t (page 78) **mpil**

bit: 23..16 Previous interrupt level

rv_csr_t (page 78) **_reserved1**

bit: 26..24 Reserved

rv_csr_t (page 78) **mpie**

bit: 27 Interrupt enable flag before enter interrupt

rv_csr_t (page 78) **mpp**

bit: 29..28 Privilede mode flag before enter interrupt

rv_csr_t (page 78) **minhv**

bit: 30 Machine interrupt vector table

rv_csr_t (page 78) **interrupt**

bit: 31 trap type.

0 means exception and 1 means interrupt

struct *CSR_MCAUSE_Type* (page 82)::[anonymous] **b**

Structure used for bit access.

rv_csr_t (page 78) **d**

Type used for csr data access.

union **CSR_MCOUNTINHIBIT_Type**

#include <core_feature_base.h> Union type to access MCOUNTINHIBIT configure register.

Public Members

rv_csr_t (page 78) **cy**

bit: 0 1 means disable mcycle counter

rv_csr_t (page 78) **_reserved0**

bit: 1 Reserved

rv_csr_t (page 78) **ir**

bit: 2 1 means disable minstret counter

rv_csr_t (page 78) **_reserved1**

bit: 3..31 Reserved

struct *CSR_MCOUNTINHIBIT_Type* (page 83)::[anonymous] **b**

Structure used for bit access.

rv_csr_t (page 78) **d**

Type used for csr data access.

PLIC

group **NMSIS_Core_PLIC_Registers**

Type definitions and defines for plic registers.

Defines

PLIC_PRIORITY_OFFSET _AC(0x0000,UL)

PLIC Priority register offset.

PLIC_PRIORITY_SHIFT_PER_SOURCE 2

PLIC Priority register offset shift per source.

PLIC_PENDING_OFFSET _AC(0x1000,UL)

PLIC Pending register offset.

PLIC_PENDING_SHIFT_PER_SOURCE 0

PLIC Pending register offset shift per source.

PLIC_ENABLE_OFFSET _AC(0x2000,UL)

PLIC Enable register offset.

PLIC_ENABLE_SHIFT_PER_TARGET 7

PLIC Enable register offset shift per target.

PLIC_THRESHOLD_OFFSET _AC(0x200000,UL)

PLIC Threshold register offset.

PLIC_CLAIM_OFFSET _AC(0x200004,UL)

PLIC Claim register offset.

PLIC_THRESHOLD_SHIFT_PER_TARGET 12

PLIC Threshold register offset shift per target.

PLIC_CLAIM_SHIFT_PER_TARGET 12

PLIC Claim register offset shift per target.

PLIC_BASE __PLIC_BASEADDR

PLIC Base Address.

SysTimer

group **NMSIS_Core_SysTimer_Registers**

Type definitions and defines for system timer registers.

Defines

SysTimer_MSIP_MSIP_Pos 0U

SysTick Timer MSIP: MSIP bit Position.

SysTimer_MSIP_MSIP_Msk (1UL << *SysTimer_MSIP_MSIP_Pos* (page 85))

SysTick Timer MSIP: MSIP Mask.

SysTimer_MTIMER_Msk (0xFFFFFFFFFFFFFFFFULL)

SysTick Timer MTIMER value Mask.

SysTimer_MTIMERCMP_Msk (0xFFFFFFFFFFFFFFFFULL)

SysTick Timer MTIMERCMP value Mask.

SysTimer_MSIP_Msk (0xFFFFFFFFUL)

SysTick Timer MSIP value Mask.

SysTimer_BASE __SYSTIMER_BASEADDR

SysTick Base Address.

SysTimer ((*SysTimer_Type* (page 85) *) *SysTimer_BASE* (page 85))

SysTick configuration struct.

struct **SysTimer_Type**

#include <core_feature_timer.h> Structure type to access the System Timer (SysTimer).

Structure definition to access the system timer(SysTimer).

Remark

CPU Intrinsic Functions

`__STATIC_FORCEINLINE void __NOP (void)`

`__STATIC_FORCEINLINE void __WFI (void)`

`__STATIC_FORCEINLINE void __EBREAK (void)`

`__STATIC_FORCEINLINE void __ECALL (void)`

`__STATIC_FORCEINLINE void __enable_mcycle_counter (void)`

`__STATIC_FORCEINLINE void __disable_mcycle_counter (void)`

`__STATIC_FORCEINLINE void __enable_minstret_counter (void)`

`__STATIC_FORCEINLINE void __disable_minstret_counter (void)`

`__STATIC_FORCEINLINE void __enable_all_counter (void)`

`__STATIC_FORCEINLINE void __disable_all_counter (void)`

`__STATIC_FORCEINLINE void __FENCE_I (void)`

`__STATIC_FORCEINLINE uint8_t __LB (volatile void *addr)`

`__STATIC_FORCEINLINE uint16_t __LH (volatile void *addr)`

`__STATIC_FORCEINLINE uint32_t __LW (volatile void *addr)`

`__STATIC_FORCEINLINE void __SB (volatile void *addr, uint8_t val)`

`__STATIC_FORCEINLINE void __SH (volatile void *addr, uint16_t val)`

`__STATIC_FORCEINLINE void __SW (volatile void *addr, uint32_t val)`

`__STATIC_FORCEINLINE uint32_t __CAS_W (volatile uint32_t *addr, uint32_t oldval, uint32_t newval)`

`__STATIC_FORCEINLINE uint32_t __AMOSWAP_W (volatile uint32_t *addr, uint32_t newval)`

`__STATIC_FORCEINLINE int32_t __AMOADD_W (volatile int32_t *addr, int32_t value)`

__STATIC_FORCEINLINE int32_t __AMOAND_W (volatile int32_t *addr, int32_t value)

__STATIC_FORCEINLINE int32_t __AMOOR_W (volatile int32_t *addr, int32_t value)

__STATIC_FORCEINLINE int32_t __AMOXOR_W (volatile int32_t *addr, int32_t value)

__STATIC_FORCEINLINE uint32_t __AMOMAXU_W (volatile uint32_t *addr, uint32_t value)

__STATIC_FORCEINLINE int32_t __AMOMAX_W (volatile int32_t *addr, int32_t value)

__STATIC_FORCEINLINE uint32_t __AMOMINU_W (volatile uint32_t *addr, uint32_t value)

__STATIC_FORCEINLINE int32_t __AMOMIN_W (volatile int32_t *addr, int32_t value)

__FENCE(p, s) *__ASM* (page 51) volatile ("fence " #p "," #s : : : "memory")

__RWMB() __FENCE(iorw,iorw)

__RMB() __FENCE(ir,ir)

__WMB() __FENCE(ow,ow)

__SMP_RWMB() __FENCE(rw,rw)

__SMP_RMB() __FENCE(r,r)

__SMP_WMB() __FENCE(w,w)

__CPU_RELAX() *__ASM* (page 51) volatile ("" : : : "memory")

group **NMSIS_Core_CPU_Intrinsic**

Functions that generate RISC-V CPU instructions.

The following functions generate specified RISC-V instructions that cannot be directly accessed by compiler.

Defines

__FENCE(p, s) *__ASM* (page 51) volatile ("fence " #p "," #s : : : "memory")

Execute fence instruction, p -> pred, s -> succ.

the FENCE instruction ensures that all memory accesses from instructions preceding the fence in program order (the `predecessor set`) appear earlier in the global memory order than memory accesses from instructions appearing after the fence in program order (the `successor set`). For details, please refer to The RISC-V Instruction Set Manual

Parameters

- **p** – predecessor set, such as iorw, rw, r, w
- **s** – successor set, such as iorw, rw, r, w

__RWMB() __FENCE(iorw,iorw)

Read & Write Memory barrier.

__RMB() __FENCE(ir,ir)

Read Memory barrier.

__WMB() __FENCE(ow,ow)

Write Memory barrier.

__SMP_RWMB() __FENCE(rw,rw)

SMP Read & Write Memory barrier.

__SMP_RMB() __FENCE(r,r)

SMP Read Memory barrier.

__SMP_WMB() __FENCE(w,w)

SMP Write Memory barrier.

__CPU_RELAX() *__ASM* (page 51) volatile ("" : : : "memory")

CPU relax for busy loop.

Functions

__STATIC_FORCEINLINE void __NOP (void)

NOP Instruction.

No Operation does nothing. This instruction can be used for code alignment purposes.

__STATIC_FORCEINLINE void __WFI (void)

Wait For Interrupt.

The Wait for Interrupt instruction (WFI) provides a hint to the implementation that the current hart can be stalled until an interrupt might need servicing. Execution of the WFI instruction can also be used to inform the hardware platform that suitable interrupts should preferentially be routed to this hart.

__STATIC_FORCEINLINE void __EBREAK (void)

Breakpoint Instruction.

Causes the processor to enter Debug state. Debug tools can use this to investigate system state when the instruction at a particular address is reached.

__STATIC_FORCEINLINE void __ECALL (void)

Environment Call Instruction.

The ECALL instruction is used to make a service request to the execution environment.

__STATIC_FORCEINLINE void __enable_mcycle_counter (void)

Enable MCYCLE counter.

Clear the CY bit of MCOUNTINHIBIT to 0 to enable MCYCLE Counter

__STATIC_FORCEINLINE void __disable_mcycle_counter (void)

Disable MCYCLE counter.

Set the CY bit of MCOUNTINHIBIT to 1 to disable MCYCLE Counter

__STATIC_FORCEINLINE void __enable_minstret_counter (void)

Enable MINSTRET counter.

Clear the IR bit of MCOUNTINHIBIT to 0 to enable MINSTRET Counter

__STATIC_FORCEINLINE void __disable_minstret_counter (void)

Disable MINSTRET counter.

Set the IR bit of MCOUNTINHIBIT to 1 to disable MINSTRET Counter

__STATIC_FORCEINLINE void __enable_all_counter (void)

Enable MCYCLE & MINSTRET counter.

Clear the IR and CY bit of MCOUNTINHIBIT to 1 to enable MINSTRET & MCYCLE Counter

__STATIC_FORCEINLINE void __disable_all_counter (void)

Disable MCYCLE & MINSTRET counter.

Set the IR and CY bit of MCOUNTINHIBIT to 1 to disable MINSTRET & MCYCLE Counter

__STATIC_FORCEINLINE void __FENCE_I (void)

Fence.i Instruction.

The FENCE.I instruction is used to synchronize the instruction and data streams.

__STATIC_FORCEINLINE uint8_t __LB (volatile void *addr)

Load 8bit value from address (8 bit)

Load 8 bit value.

Parameters addr – **[in]** Address pointer to data

Returns value of type uint8_t at (*addr)

__STATIC_FORCEINLINE uint16_t __LH (volatile void *addr)

Load 16bit value from address (16 bit)

Load 16 bit value.

Parameters addr – **[in]** Address pointer to data

Returns value of type uint16_t at (*addr)

__STATIC_FORCEINLINE uint32_t __LW (volatile void *addr)

Load 32bit value from address (32 bit)

Load 32 bit value.

Parameters addr – **[in]** Address pointer to data

Returns value of type uint32_t at (*addr)

__STATIC_FORCEINLINE void __SB (volatile void *addr, uint8_t val)

Write 8bit value to address (8 bit)

Write 8 bit value.

Parameters

- **addr** – **[in]** Address pointer to data
- **val** – **[in]** Value to set

__STATIC_FORCEINLINE void __SH (volatile void *addr, uint16_t val)

Write 16bit value to address (16 bit)

Write 16 bit value.

Parameters

- **addr** – **[in]** Address pointer to data
- **val** – **[in]** Value to set

__STATIC_FORCEINLINE void __SW (volatile void *addr, uint32_t val)

Write 32bit value to address (32 bit)

Write 32 bit value.

Parameters

- **addr** – **[in]** Address pointer to data
- **val** – **[in]** Value to set

__STATIC_FORCEINLINE uint32_t __CAS_W (volatile uint32_t *addr, uint32_t oldval, uint32_t newval)

Compare and Swap 32bit value using LR and SC.

Compare old value with memory, if identical, store new value in memory. Return the initial value in memory. Success is indicated by comparing return value with OLD. memory address, return 0 if successful, otherwise return !0

Parameters

- **addr** – **[in]** Address pointer to data, address need to be 4byte aligned
- **oldval** – **[in]** Old value of the data in address
- **newval** – **[in]** New value to be stored into the address

Returns return the initial value in memory

__STATIC_FORCEINLINE uint32_t __AMOSWAP_W (volatile uint32_t *addr, uint32_t newval)

Atomic Swap 32bit value into memory.

Atomically swap new 32bit value into memory using amoswap.d.

Parameters

- **addr** – **[in]** Address pointer to data, address need to be 4byte aligned
- **newval** – **[in]** New value to be stored into the address

Returns return the original value in memory

__STATIC_FORCEINLINE int32_t __AMOADD_W (volatile int32_t *addr, int32_t value)

Atomic Add with 32bit value.

Atomically ADD 32bit value with value in memory using amoadd.d.

Parameters

- `addr` – **[in]** Address pointer to data, address need to be 4byte aligned
- `value` – **[in]** value to be ADDed

Returns return memory value + add value

`__STATIC_FORCEINLINE int32_t __AMOAND_W (volatile int32_t *addr, int32_t value)`

Atomic And with 32bit value.

Atomically AND 32bit value with value in memory using amoand.d.

Parameters

- `addr` – **[in]** Address pointer to data, address need to be 4byte aligned
- `value` – **[in]** value to be ANDed

Returns return memory value & and value

`__STATIC_FORCEINLINE int32_t __AMOOR_W (volatile int32_t *addr, int32_t value)`

Atomic OR with 32bit value.

Atomically OR 32bit value with value in memory using amoor.d.

Parameters

- `addr` – **[in]** Address pointer to data, address need to be 4byte aligned
- `value` – **[in]** value to be ORed

Returns return memory value | and value

`__STATIC_FORCEINLINE int32_t __AMOXOR_W (volatile int32_t *addr, int32_t value)`

Atomic XOR with 32bit value.

Atomically XOR 32bit value with value in memory using amoxor.d.

Parameters

- `addr` – **[in]** Address pointer to data, address need to be 4byte aligned
- `value` – **[in]** value to be XORed

Returns return memory value ^ and value

`__STATIC_FORCEINLINE uint32_t __AMOMAXU_W (volatile uint32_t *addr, uint32_t value)`

Atomic unsigned MAX with 32bit value.

Atomically unsigned max compare 32bit value with value in memory using amomaxu.d.

Parameters

- `addr` – **[in]** Address pointer to data, address need to be 4byte aligned
- `value` – **[in]** value to be compared

Returns return the bigger value

__STATIC_FORCEINLINE int32_t __AMOMAX_W (volatile int32_t *addr, int32_t value)

Atomic signed MAX with 32bit value.

Atomically signed max compare 32bit value with value in memory using amomax.d.

Parameters

- **addr** – **[in]** Address pointer to data, address need to be 4byte aligned
- **value** – **[in]** value to be compared

Returns the bigger value

__STATIC_FORCEINLINE uint32_t __AMOMINU_W (volatile uint32_t *addr, uint32_t value)

Atomic unsigned MIN with 32bit value.

Atomically unsigned min compare 32bit value with value in memory using amominu.d.

Parameters

- **addr** – **[in]** Address pointer to data, address need to be 4byte aligned
- **value** – **[in]** value to be compared

Returns the smaller value

__STATIC_FORCEINLINE int32_t __AMOMIN_W (volatile int32_t *addr, int32_t value)

Atomic signed MIN with 32bit value.

Atomically signed min compare 32bit value with value in memory using amomin.d.

Parameters

- **addr** – **[in]** Address pointer to data, address need to be 4byte aligned
- **value** – **[in]** value to be compared

Returns the smaller value

Peripheral Access

__I volatile const

__O volatile

__IO volatile

__IM volatile const

__OM volatile

__IOM volatile

_VAL2FLD(field, value) (((uint32_t)(value) << field ## _Pos) & field ## _Msk)

_FLD2VAL(field, value) (((uint32_t)(value) & field ## _Msk) >> field ## _Pos)

group **NMSIS_Core_PeriphAccess**

Naming conventions and optional features for accessing peripherals.

The section below describes the naming conventions, requirements, and optional features for accessing device specific peripherals. Most of the rules also apply to the core peripherals.

The **Device Header File <device.h>** contains typically these definition and also includes the core specific header files.

Defines

__I volatile const

Defines 'read only' permissions.

__O volatile

Defines 'write only' permissions.

__IO volatile

Defines 'read / write' permissions.

__IM volatile const

Defines 'read only' structure member permissions.

__OM volatile

Defines 'write only' structure member permissions.

__IOM volatile

Defines 'read/write' structure member permissions.

_VAL2FLD(field, value) (((uint32_t)(value) << field ## _Pos) & field ## _Msk)

Mask and shift a bit field value for use in a register bit range.

The macro _VAL2FLD uses the #define's _Pos and _Msk of the related bit field to shift bit-field values for assigning to a register.

Example:

```
PLIC->CFG = _VAL2FLD(CLIC_CLICCFG_NLBIT, 3);
```

Parameters

- **field** – **[in]** Name of the register bit field.
- **value** – **[in]** Value of the bit field. This parameter is interpreted as an uint32_t type.

Returns Masked and shifted value.

_FLD2VAL(field, value) (((uint32_t)(value) & field ## _Msk) >> field ## _Pos)

Mask and shift a register value to extract a bit filed value.

The macro _FLD2VAL uses the #define's _Pos and _Msk of the related bit field to extract the value of a bit field from a register.

Example:

```
nlbits = _FLD2VAL(CLIC_CLICCFG_NLBIT, PLIC->CFG);
```

Parameters

- **field** – **[in]** Name of the register bit field.
- **value** – **[in]** Value of register. This parameter is interpreted as an uint32_t type.

Returns Masked and shifted bit field value.

Systick Timer(SysTimer)

SysTimer API

__STATIC_FORCEINLINE void SysTimer_SetLoadValue (uint64_t value)

__STATIC_FORCEINLINE uint64_t SysTimer_GetLoadValue (void)

__STATIC_FORCEINLINE void SysTimer_SetCompareValue (uint64_t value)

__STATIC_FORCEINLINE uint64_t SysTimer_GetCompareValue (void)

__STATIC_FORCEINLINE void SysTimer_SetSWIRQ (void)

__STATIC_FORCEINLINE void SysTimer_ClearSWIRQ (void)

__STATIC_FORCEINLINE uint32_t SysTimer_GetMsipValue (void)

__STATIC_FORCEINLINE void SysTimer_SetMsipValue (uint32_t msip)

__STATIC_INLINE uint32_t SysTick_Config (uint64_t ticks)

__STATIC_FORCEINLINE uint32_t SysTick_Reload (uint64_t ticks)

group **NMSIS_Core_SysTimer**

Functions that configure the Core System Timer.

Functions

`__STATIC_FORCEINLINE void SysTimer_SetLoadValue (uint64_t value)`

Set system timer load value.

This function set the system timer load value in MTIMER register.

Remark

- Load value is 64bits wide.
- SysTimer_GetLoadValue

Parameters `value` – **[in]** value to set system timer MTIMER register.

`__STATIC_FORCEINLINE uint64_t SysTimer_GetLoadValue (void)`

Get system timer load value.

This function get the system timer current value in MTIMER register.

Remark

- Load value is 64bits wide.
- SysTimer_SetLoadValue

Returns current value(64bit) of system timer MTIMER register.

`__STATIC_FORCEINLINE void SysTimer_SetCompareValue (uint64_t value)`

Set system timer compare value.

This function set the system Timer compare value in MTIMERCMP register.

Remark

- Compare value is 64bits wide.
- If compare value is larger than current value timer interrupt generate.
- Modify the load value or compare value less to clear the interrupt.
- SysTimer_GetCompareValue

Parameters `value` – **[in]** compare value to set system timer MTIMERCMP register.

`__STATIC_FORCEINLINE uint64_t SysTimer_GetCompareValue (void)`

Get system timer compare value.

This function get the system timer compare value in MTIMERCMP register.

Remark

- Compare value is 64bits wide.
- SysTimer_SetCompareValue

Returns compare value of system timer MTIMERCMP register.

`__STATIC_FORCEINLINE void SysTimer_SetSWIRQ (void)`

Trigger or set software interrupt via system timer.

This function set the system timer MSIP bit in MSIP register.

Remark

- Set system timer MSIP bit and generate a SW interrupt.
- SysTimer_ClearSWIRQ
- SysTimer_GetMsipValue

`__STATIC_FORCEINLINE void SysTimer_ClearSWIRQ (void)`

Clear system timer software interrupt pending request.

This function clear the system timer MSIP bit in MSIP register.

Remark

- Clear system timer MSIP bit in MSIP register to clear the software interrupt pending.
- SysTimer_SetSWIRQ
- SysTimer_GetMsipValue

`__STATIC_FORCEINLINE uint32_t SysTimer_GetMsipValue (void)`

Get system timer MSIP register value.

This function get the system timer MSIP register value.

Remark

- Bit0 is SW interrupt flag. Bit0 is 1 then SW interrupt set. Bit0 is 0 then SW interrupt clear.

- SysTimer_SetSWIRQ
- SysTimer_ClearSWIRQ

Returns Value of Timer MSIP register.

`__STATIC_FORCEINLINE void SysTimer_SetMsipValue (uint32_t msip)`

Set system timer MSIP register value.

This function set the system timer MSIP register value.

Parameters `msip` – **[in]** value to set MSIP register

`__STATIC_INLINE uint32_t SysTick_Config (uint64_t ticks)`

System Tick Configuration.

Initializes the System Timer and its non-vector interrupt, and starts the System Tick Timer.

In our default implementation, the timer counter will be set to zero, and it will start a timer compare non-vector interrupt when it matchs the ticks user set, during the timer interrupt user should reload the system tick using SysTick_Reload function or similar function written by user, so it can produce period timer interrupt.

See also:

- SysTimer_SetCompareValue; SysTimer_SetLoadValue

Parameters `ticks` – **[in]** Number of ticks between two interrupts.

Returns 0 Function succeeded.

Returns 1 Function failed.

`__STATIC_FORCEINLINE uint32_t SysTick_Reload (uint64_t ticks)`

System Tick Reload.

Reload the System Timer Tick when the MTIMECMP reached TIME value

See also:

- SysTimer_SetCompareValue
- SysTimer_SetLoadValue

Parameters `ticks` – **[in]** Number of ticks between two interrupts.

Returns 0 Function succeeded.

Returns 1 Function failed.

Interrupts and Exceptions

Interrupt and Exception API

enum **IRQn**

Values:

enumerator **Reserved0_IRQn**

enumerator **Reserved1_IRQn**

enumerator **Reserved2_IRQn**

enumerator **SysTimerSW_IRQn**

enumerator **Reserved4_IRQn**

enumerator **Reserved5_IRQn**

enumerator **Reserved6_IRQn**

enumerator **SysTimer_IRQn**

enumerator **Reserved8_IRQn**

enumerator **Reserved9_IRQn**

enumerator **Reserved10_IRQn**

enumerator **Reserved11_IRQn**

enumerator **Reserved12_IRQn**

enumerator **Reserved13_IRQn**

enumerator **Reserved14_IRQn**

enumerator **Reserved15_IRQn**

enumerator **PLIC_INT0_IRQn**

enumerator **PLIC_INT1_IRQn**

enumerator `PLIC_INT_MAX`

`__STATIC_FORCEINLINE void PLIC_SetThreshold (uint32_t thresh)`

`__STATIC_FORCEINLINE uint32_t PLIC_GetThreshold (void)`

`__STATIC_FORCEINLINE void PLIC_EnableInterrupt (uint32_t source)`

`__STATIC_FORCEINLINE void PLIC_DisableInterrupt (uint32_t source)`

`__STATIC_FORCEINLINE uint32_t PLIC_GetInterruptEnable (uint32_t source)`

`__STATIC_FORCEINLINE void PLIC_SetPriority (uint32_t source, uint32_t priority)`

`__STATIC_FORCEINLINE uint32_t PLIC_GetPriority (uint32_t source, uint32_t priority)`

`__STATIC_FORCEINLINE uint32_t PLIC_ClaimInterrupt (void)`

`__STATIC_FORCEINLINE void PLIC_CompleteInterrupt (uint32_t source)`

`__STATIC_FORCEINLINE void PLIC_Init (uint32_t num_sources)`

`__STATIC_FORCEINLINE void __set_trap_entry (rv_csr_t addr)`

`__STATIC_FORCEINLINE rv_csr_t __get_trap_entry (void)`

group `NMSIS_Core_IntExc`

Functions that manage interrupts and exceptions via the PLIC.

Enums

enum `IRQn`

Definition of IRQn numbers.

The core interrupt enumeration names for IRQn values are defined in the file **<Device>.h**.

- Interrupt ID(IRQn) from 0 to 18 are reserved for core internal interrupts.
- Interrupt ID(IRQn) start from 19 represent device-specific external interrupts.
- The first device-specific interrupt has the IRQn value 19.

The table below describes the core interrupt names and their availability in various Nuclei Cores.

Values:

enumerator **Reserved0_IRQn**

Internal reserved.

enumerator **Reserved1_IRQn**

Internal reserved.

enumerator **Reserved2_IRQn**

Internal reserved.

enumerator **SysTimerSW_IRQn**

System Timer SW interrupt.

enumerator **Reserved4_IRQn**

Internal reserved.

enumerator **Reserved5_IRQn**

Internal reserved.

enumerator **Reserved6_IRQn**

Internal reserved.

enumerator **SysTimer_IRQn**

System Timer Interrupt.

enumerator **Reserved8_IRQn**

Internal reserved.

enumerator **Reserved9_IRQn**

Internal reserved.

enumerator **Reserved10_IRQn**

Internal reserved.

enumerator **Reserved11_IRQn**

Internal reserved.

enumerator **Reserved12_IRQn**

Internal reserved.

enumerator **Reserved13_IRQn**

Internal reserved.

enumerator **Reserved14_IRQn**

Internal reserved.

enumerator **Reserved15_IRQn**

Internal reserved.

enumerator **PLIC_INT0_IRQn**

0 plic interrupt, means no interrupt

enumerator **PLIC_INT1_IRQn**

1st plic interrupt

enumerator **PLIC_INT_MAX**

Number of total plic interrupts.

Functions

__STATIC_FORCEINLINE void PLIC_SetThreshold (uint32_t thresh)

Set priority threshold value of plic.

This function set priority threshold value of plic for current hart.

Remark

See also:

- PLIC_GetThreshold

Parameters **thresh** – **[in]** threshold value

__STATIC_FORCEINLINE uint32_t PLIC_GetThreshold (void)

Get priority threshold value of plic.

This function get priority threshold value of plic.

Remark

See also:

- PLIC_SetThreshold

Returns priority threshold value for current hart

`__STATIC_FORCEINLINE void PLIC_EnableInterrupt (uint32_t source)`

Enable interrupt for selected source plic.

This function enable interrupt for selected source plic of current hart.

Remark

See also:

- PLIC_DisableInterrupt

Parameters `source` – **[in]** interrupt source

`__STATIC_FORCEINLINE void PLIC_DisableInterrupt (uint32_t source)`

Disable interrupt for selected source plic.

This function disable interrupt for selected source plic of current hart.

Remark

See also:

- PLIC_EnableInterrupt

Parameters `source` – **[in]** interrupt source

`__STATIC_FORCEINLINE uint32_t PLIC_GetInterruptEnable (uint32_t source)`

Get interrupt enable status for selected source plic.

This function get interrupt enable for selected source plic of current hart.

Remark

See also:

- PLIC_EnableInterrupt
- PLIC_DisableInterrupt

Parameters `source` – **[in]** interrupt source

Returns enable status for selected interrupt source for current hart

`__STATIC_FORCEINLINE void PLIC_SetPriority (uint32_t source, uint32_t priority)`

Set interrupt priority for selected source plic.

This function set interrupt priority for selected source plic of current hart.

Remark

See also:

- PLIC_GetPriority

Parameters

- `source` – **[in]** interrupt source
- `priority` – **[in]** interrupt priority

`__STATIC_FORCEINLINE uint32_t PLIC_GetPriority (uint32_t source, uint32_t priority)`

Get interrupt priority for selected source plic.

This function get interrupt priority for selected source plic of current hart.

Remark

See also:

- PLIC_SetPriority

Parameters

- `source` – **[in]** interrupt source
- `priority` – **[in]** interrupt priority

`__STATIC_FORCEINLINE uint32_t PLIC_ClaimInterrupt (void)`

Claim interrupt for plic of current hart.

This function claim interrupt for plic of current hart.

Remark

A successful claim will also atomically clear the corresponding pending bit on the interrupt source. The PLIC can perform a claim at any time and the claim operation is not affected by the setting of the priority threshold register.

See also:

- PLIC_CompleteInterrupt

  Returns the ID of the highest priority pending interrupt or zero if there is no pending interrupt

`__STATIC_FORCEINLINE void PLIC_CompleteInterrupt (uint32_t source)`

Complete interrupt for plic of current hart.

This function complete interrupt for plic of current hart.

Remark

The PLIC signals it has completed executing an interrupt handler by writing the interrupt ID it received from the claim to the claim/complete register. The PLIC does not check whether the completion ID is the same as the last claim ID for that target. If the completion ID does not match an interrupt source that is currently enabled for the target, the completion is silently ignored.

See also:

- PLIC_ClaimInterrupt

  Returns the ID of the highest priority pending interrupt or zero if there is no pending interrupt

`__STATIC_FORCEINLINE void PLIC_Init (uint32_t num_sources)`

Perform init for plic of current hart.

This function perform initialization steps for plic of current hart.

Remark

- Disable all interrupts
- Set all priorities to zero
- Set priority threshold to zero

`__STATIC_FORCEINLINE void __set_trap_entry (rv_csr_t addr)`

Set Trap entry address.

This function set trap entry address to 'CSR_MTVEC'.

Remark

- This function use to set trap entry address to 'CSR_MTVEC'.

See also:

- __get_trap_entry

Parameters `addr` – **[in]** trap entry address

`__STATIC_FORCEINLINE rv_csr_t __get_trap_entry (void)`

Get trap entry address.

This function get trap entry address from 'CSR_MTVEC'.

Remark

- This function use to get trap entry address from 'CSR_MTVEC'.

See also:

- __set_trap_entry

Returns trap entry address

FPU Functions

group **`NMSIS_Core_FPU_Functions`**

Functions that related to the RISC-V FPU (F and D extension).

Nuclei provided floating point unit by RISC-V F and D extension.

- `F extension` adds single-precision floating-point computational instructions compliant with the IEEE 754-2008 arithmetic standard, __RISCV_FLEN = 32. The F extension adds 32 floating-point registers, f0-f31, each 32 bits wide, and a floating-point control and status register fcsr, which contains the operating mode and exception status of the floating-point unit.
- `D extension` adds double-precision floating-point computational instructions compliant with the IEEE 754-2008 arithmetic standard. The D extension widens the 32 floating-point registers, f0-f31, to 64 bits, __RISCV_FLEN = 64

Defines

`__RISCV_FLEN` 64

`__get_FCSR``()` *__RV_CSR_READ* (page 53)(*CSR_FCSR* (page 57))

Get FCSR CSR Register.

`__set_FCSR`(val) *__RV_CSR_WRITE* (page 53)(*CSR_FCSR* (page 57), (val))

Set FCSR CSR Register with val.

`__get_FRM``()` *__RV_CSR_READ* (page 53)(*CSR_FRM* (page 57))

Get FRM CSR Register.

`__set_FRM`(val) *__RV_CSR_WRITE* (page 53)(*CSR_FRM* (page 57), (val))

Set FRM CSR Register with val.

__get_FFLAGS() *__RV_CSR_READ* (page 53)(*CSR_FFLAGS* (page 57))

Get FFLAGS CSR Register.

__set_FFLAGS(val) *__RV_CSR_WRITE* (page 53)(*CSR_FFLAGS* (page 57), (val))

Set FFLAGS CSR Register with val.

__enable_FPU() *__RV_CSR_SET* (page 53)(*CSR_MSTATUS* (page 59), *MSTATUS_FS* (page 69))

Enable FPU Unit.

__disable_FPU() *__RV_CSR_CLEAR* (page 54)(*CSR_MSTATUS* (page 59), *MSTATUS_FS* (page 69))

Disable FPU Unit.

- We can save power by disable FPU Unit.
- When FPU Unit is disabled, any access to FPU related CSR registers and FPU instructions will cause illegal Instuction Exception.

__RV_FLW(freg, addr, ofs)

Load a single-precision value from memory into float point register freg using flw instruction.

The FLW instruction loads a single-precision floating point value from memory address (addr + ofs) into floating point register freg(f0-f31)

Remark

- FLW and FSW operations need to make sure the address is 4 bytes aligned, otherwise it will cause exception code 4(Load address misaligned) or 6 (Store/AMO address misaligned)
- FLW and FSW do not modify the bits being transferred; in particular, the payloads of non-canonical NaNs are preserved

Parameters

- **freg** – **[in]** The floating point register, eg. *FREG(0)* (page 76), f0
- **addr** – **[in]** The memory base address, 4 byte aligned required
- **ofs** – **[in]** a 12-bit immediate signed byte offset value, should be an const value

__RV_FSW(freg, addr, ofs)

Store a single-precision value from float point freg into memory using fsw instruction.

The FSW instruction stores a single-precision value from floating point register to memory

Remark

- FLW and FSW operations need to make sure the address is 4 bytes aligned, otherwise it will cause exception code 4(Load address misaligned) or 6 (Store/AMO address misaligned)
- FLW and FSW do not modify the bits being transferred; in particular, the payloads of non-canonical NaNs are preserved

Parameters

- **freg** – **[in]** The floating point register(f0-f31), eg. *FREG(0)* (page 76), f0
- **addr** – **[in]** The memory base address, 4 byte aligned required
- **ofs** – **[in]** a 12-bit immediate signed byte offset value, should be an const value

__RV_FLD(freg, addr, ofs)

Load a double-precision value from memory into float point register freg using fld instruction.

The FLD instruction loads a double-precision floating point value from memory address (addr + ofs) into floating point register freg(f0-f31)

Remark

- FLD and FSD operations need to make sure the address is 8 bytes aligned, otherwise it will cause exception code 4(Load address misaligned) or 6 (Store/AMO address misaligned)
- FLD and FSD do not modify the bits being transferred; in particular, the payloads of non-canonical NaNs are preserved.

Attention

- Function only available for double precision floating point unit, FLEN = 64

Parameters

- **freg** – **[in]** The floating point register, eg. *FREG(0)* (page 76), f0
- **addr** – **[in]** The memory base address, 8 byte aligned required
- **ofs** – **[in]** a 12-bit immediate signed byte offset value, should be an const value

__RV_FSD(freg, addr, ofs)

Store a double-precision value from float point freg into memory using fsd instruction.

The FSD instruction stores double-precision value from floating point register to memory

Remark

- FLD and FSD operations need to make sure the address is 8 bytes aligned, otherwise it will cause exception code 4(Load address misaligned) or 6 (Store/AMO address misaligned)
- FLD and FSD do not modify the bits being transferred; in particular, the payloads of non-canonical NaNs are preserved.

Attention

- Function only available for double precision floating point unit, FLEN = 64

Parameters

- **freg** – **[in]** The floating point register(f0-f31), eg. *FREG(0)* (page 76), f0
- **addr** – **[in]** The memory base address, 8 byte aligned required

- **ofs** – **[in]** a 12-bit immediate signed byte offset value, should be an const value

__RV_FLOAD *__RV_FLD* (page 107)

Load a float point value from memory into float point register freg using flw/fld instruction.

- For Single-Precison Floating-Point Mode(__FPU_PRESENT == 1, __RISCV_FLEN == 32): It will call *__RV_FLW* (page 106) to load a single-precision floating point value from memory to floating point register
- For Double-Precison Floating-Point Mode(__FPU_PRESENT == 2, __RISCV_FLEN == 64): It will call *__RV_FLD* (page 107) to load a double-precision floating point value from memory to floating point register

Attention Function behaviour is different for __FPU_PRESENT = 1 or 2, please see the real function this macro represent

__RV_FSTORE *__RV_FSD* (page 107)

Store a float value from float point freg into memory using fsw/fsd instruction.

- For Single-Precison Floating-Point Mode(__FPU_PRESENT == 1, __RISCV_FLEN == 32): It will call *__RV_FSW* (page 106) to store floating point register into memory
- For Double-Precison Floating-Point Mode(__FPU_PRESENT == 2, __RISCV_FLEN == 64): It will call *__RV_FSD* (page 107) to store floating point register into memory

Attention Function behaviour is different for __FPU_PRESENT = 1 or 2, please see the real function this macro represent

SAVE_FPU_CONTEXT()

Save FPU context into variables for interrupt nesting.

This macro is used to declare variables which are used for saving FPU context, and it will store the nessary fpu registers into these variables, it need to be used in a interrupt when in this interrupt fpu registers are used.

Remark

- It need to be used together with *RESTORE_FPU_CONTEXT* (page 109)
- Don't use variable names __fpu_context in your ISR code
- If you isr code will use fpu registers, and this interrupt is nested. Then you can do it like this:

```
void core_mtip_handler(void)
{
    // !!!Interrupt is enabled here!!!
    // !!!Higher priority interrupt could nest it!!!

    // Necessary only when you need to use fpu registers
```

(continues on next page)

(continued from previous page)

```
    // in this isr handler functions
    SAVE_FPU_CONTEXT();

    // put you own interrupt handling code here

    // pair of SAVE_FPU_CONTEXT()
    RESTORE_FPU_CONTEXT();
}
```

RESTORE_FPU_CONTEXT()

Restore necessary fpu registers from variables for interrupt nesting.

This macro is used restore necessary fpu registers from pre-defined variables in *SAVE_FPU_CONTEXT* (page 108) macro.

Remark

- It need to be used together with *SAVE_FPU_CONTEXT* (page 108)

Typedefs

typedef uint64_t **rv_fpu_t**

Type of FPU register, depends on the FLEN defined in RISC-V.

System Device Configuration

group **NMSIS_Core_SystemConfig**

Functions for system and clock setup available in system_<device>.c.

HummingBird provides a template file **system_Device.c** that must be adapted by the silicon vendor to match their actual device. As a **minimum requirement**, this file must provide:

- A device-specific system configuration function, *SystemInit* (page 110).
- A global variable that contains the system frequency, *SystemCoreClock* (page 111).
- Global c library _premain_init and _postmain_fini functions called right before and after calling main function.
- Vendor customized interrupt, exception handling code, see *Interrupt and Exception Handling* (page 111)

The file configures the device and, typically, initializes the oscillator (PLL) that is part of the microcontroller device. This file might export other functions or variables that provide a more flexible configuration of the microcontroller system.

And this file also provided common interrupt, exception exception handling framework template, Silicon vendor can customize these template code as they want.

Attention Be aware that a value stored to `SystemCoreClock` during low level initializaton (i.e. *`SystemInit()`* `(page 110)`) might get overwritten by C libray startup code and/or .bss section initialization. Thus its highly recommended to call *SystemCoreClockUpdate* (page 110) at the beginning of the user `main()` routine.

Note: Please pay special attention to the static variable `SystemCoreClock`. This variable might be used throughout the whole system initialization and runtime to calculate frequency/time related values. Thus one must assure that the variable always reflects the actual system clock speed.

Functions

void **`SystemCoreClockUpdate`**(void)

Function to update the variable *SystemCoreClock* (page 111).

Updates the variable *SystemCoreClock* (page 111) and must be called whenever the core clock is changed during program execution. The function evaluates the clock register settings and calculates the current core clock.

void **`SystemInit`**(void)

Function to Initialize the system.

Initializes the microcontroller system. Typically, this function configures the oscillator (PLL) that is part of the microcontroller device. For systems with a variable clock speed, it updates the variable *SystemCoreClock* (page 111). SystemInit is called from the file **startup**.

void **`SystemBannerPrint`**(void)

Banner Print for HummingBird SDK.

int32_t **`Core_Register_IRQ`**(uint32_t irqn, void *handler)

Register a riscv core interrupt and register the handler.

This function set interrupt handler for core interrupt

Remark

- This function use to configure riscv core interrupt and register its interrupt handler and enable its interrupt.

Parameters

- **`irqn`** – **[in]** interrupt number
- **`handler`** – **[in]** interrupt handler, if NULL, handler will not be installed

Returns -1 means invalid input parameter. 0 means successful.

int32_t **`PLIC_Register_IRQ`**(uint32_t source, uint8_t priority, void *handler)

Register a specific plic interrupt and register the handler.

This function set priority and handler for plic interrupt

Remark

- This function use to configure specific plic interrupt and register its interrupt handler and enable its interrupt.

Parameters

- **source** – **[in]** interrupt source
- **priority** – **[in]** interrupt priority
- **handler** – **[in]** interrupt handler, if NULL, handler will not be installed

Returns -1 means invalid input parameter. 0 means successful.

Variables

uint32_t **SystemCoreClock** = SYSTEM_CLOCK

Variable to hold the system core clock value.

Holds the system core clock, which is the system clock frequency supplied to the SysTick timer and the processor core clock. This variable can be used by debuggers to query the frequency of the debug timer or to configure the trace clock speed.

Attention Compilers must be configured to avoid removing this variable in case the application program is not using it. Debugging systems require the variable to be physically present in memory so that it can be examined to configure the debugger.

Interrupt Exception NMI Handling

group **NMSIS_Core_IntExcNMI_Handling**

Functions for interrupt, exception handle available in system_<device>.c.

HBIRD provide a template for interrupt, exception handling. Silicon Vendor could adapat according to their requirement. Silicon vendor could implement interface for different exception code and replace current implementation.

Defines

MAX_SYSTEM_EXCEPTION_NUM 11

Max exception handler number.

Typedefs

typedef void (***EXC_HANDLER**)(unsigned long mcause, unsigned long sp)

Exception Handler Function Typedef.

Note: This typedef is only used internal in this system_<Device>.c file. It is used to do type conversion for registered exception handler before calling it.

typedef void (***INT_HANDLER**)(unsigned long mcause, unsigned long sp)

Functions

static uint32_t **core_exception_handler**(unsigned long mcause, unsigned long sp)

Common Exception handler entry.

This function provided a command entry for exception. Silicon Vendor could modify this template implementation according to requirement.

Remark

- RISCV provided common entry for all types of exception. This is proposed code template for exception entry function, Silicon Vendor could modify the implementation.
- For the core_exception_handler template, we provided exception register function *Exception_Register_EXC* (page 113) which can help developer to register your exception handler for specific exception number.

static void **system_default_exception_handler**(unsigned long mcause, unsigned long sp)

System Default Exception Handler.

This function provided a default exception handling code for all exception ids. By default, It will just print some information for debug, Vendor can customize it according to its requirements.

static void **system_default_interrupt_handler**(unsigned long mcause, unsigned long sp)

System Default Interrupt Handler.

This function provided a default interrupt handling code for all interrupt ids.

static void **Exception_Init**(void)

Initialize all the default core exception handlers.

The core exception handler for each exception id will be initialized to *system_default_exception_handler* (page 112).

Note: Called in _init function, used to initialize default exception handlers for all exception IDs static void **Interrupt_Init**(void)

Initialize all the default interrupt handlers.

The interrupt handler for each exception id will be initialized to *system_default_interrupt_handler* (page 112).

Note: Called in _init function, used to initialize default interrupt handlers for all interrupt IDs

void **Exception_Register_EXC**(uint32_t EXCn, unsigned long exc_handler)

Register an exception handler for exception code EXCn.

- For EXCn < *MAX_SYSTEM_EXCEPTION_NUM* (page 111), it will be registered into SystemExceptionHandlers[EXCn-1].

Parameters

- **EXCn** – See EXCn_Type
- **exc_handler** – The exception handler for this exception code EXCn

void **Interrupt_Register_CoreIRQ**(uint32_t irqn, unsigned long int_handler)

Register an core interrupt handler for core interrupt number.

- For irqn <= 10, it will be registered into SystemCoreInterruptHandlers[irqn-1].

Parameters

- **irqn** – See IRQn
- **int_handler** – The core interrupt handler for this interrupt code irqn

void **Interrupt_Register_ExtIRQ**(uint32_t irqn, unsigned long int_handler)

Register an external interrupt handler for plic external interrupt number.

- For irqn <= __PLIC_INTNUM, it will be registered into SystemExtInterruptHandlers[irqn-1].

Parameters

- **irqn** – See IRQn
- **int_handler** – The external interrupt handler for this interrupt code irqn

unsigned long **Interrupt_Get_CoreIRQ**(uint32_t irqn)

Get an core interrupt handler for core interrupt number.

Parameters **irqn** – See IRQn

Returns The core interrupt handler for this interrupt code irqn

unsigned long **Interrupt_Get_ExtIRQ**(uint32_t irqn)

Get an external interrupt handler for external interrupt number.

Parameters **irqn** – See IRQn

Returns The external interrupt handler for this interrupt code irqn

unsigned long **`Exception_Get_EXC`**(uint32_t EXCn)

Get current exception handler for exception code EXCn.

- For EXCn < *MAX_SYSTEM_EXCEPTION_NUM* (page 111), it will return SystemExceptionHandlers[EXCn-1].

Parameters **`EXCn`** – See EXCn_Type

Returns Current exception handler for exception code EXCn, if not found, return 0.

uint32_t **`core_trap_handler`**(unsigned long mcause, unsigned long sp)

Common trap entry.

This function provided a command entry for trap. Silicon Vendor could modify this template implementation according to requirement.

Remark

- RISCV provided common entry for all types of exception including exception and interrupt. This is proposed code template for exception entry function, Silicon Vendor could modify the implementation.
- If you want to register core exception handler, please use *Exception_Register_EXC* (page 113)
- If you want to register core interrupt handler, please use *Interrupt_Register_CoreIRQ* (page 113)
- If you want to register external interrupt handler, please use *Interrupt_Register_ExtIRQ* (page 113)

Variables

static unsigned long **`SystemExceptionHandlers`**[MAX_SYSTEM_EXCEPTION_NUM]

Store the exception handlers for each exception ID.

Note:

- This SystemExceptionHandlers are used to store all the handlers for all the exception codes RISC-V core provided.
- Exception code 0 - 11, totally 12 exceptions are mapped to SystemExceptionHandlers[0:11]

static unsigned long **`SystemExtInterruptHandlers`**[__PLIC_INTNUM]

static unsigned long **`SystemCoreInterruptHandlers`**[10]

ARM Compatiable Functions

group `NMSIS_Core_ARMCompatiable_Functions`

A few functions that compatiable with ARM CMSIS-Core.

Here we provided a few functions that compatiable with ARM CMSIS-Core, mostly used in the DSP and NN library.

Defines

`__ISB`() __RWMB()

Instruction Synchronization Barrier, compatiable with ARM.

`__DSB`() __RWMB()

Data Synchronization Barrier, compatiable with ARM.

`__DMB`() __RWMB()

Data Memory Barrier, compatiable with ARM.

`__LDRBT`(ptr) __LB((ptr))

LDRT Unprivileged (8 bit), ARM Compatiable.

`__LDRHT`(ptr) __LH((ptr))

LDRT Unprivileged (16 bit), ARM Compatiable.

`__LDRT`(ptr) __LW((ptr))

LDRT Unprivileged (32 bit), ARM Compatiable.

`__STRBT`(val, ptr) __SB((ptr), (val))

STRT Unprivileged (8 bit), ARM Compatiable.

`__STRHT`(val, ptr) __SH((ptr), (val))

STRT Unprivileged (16 bit), ARM Compatiable.

`__STRT`(val, ptr) __SW((ptr), (val))

STRT Unprivileged (32 bit), ARM Compatiable.

`__PKHBT`(ARG1, ARG2, ARG3)

Halfword packing instruction.

Combines bits[15:0] of val1 with bits[31:16] of val2 levitated with the val3.

`__PKHTB`(ARG1, ARG2, ARG3)

Halfword packing instruction.

Combines bits[31:16] of val1 with bits[15:0] of val2 right-shifted with the val3.

Functions

`__STATIC_FORCEINLINE int32_t __SSAT (int32_t val, uint32_t sat)`

Signed Saturate.

Saturates a signed value.

Parameters

- **`value`** – **[in]** Value to be saturated
- **`sat`** – **[in]** Bit position to saturate to (1..32)

Returns Saturated value

`__STATIC_FORCEINLINE uint32_t __USAT (int32_t val, uint32_t sat)`

Unsigned Saturate.

Saturates an unsigned value.

Parameters

- **`value`** – **[in]** Value to be saturated
- **`sat`** – **[in]** Bit position to saturate to (0..31)

Returns Saturated value

`__STATIC_FORCEINLINE uint32_t __REV (uint32_t value)`

Reverse byte order (32 bit)

Reverses the byte order in unsigned integer value. For example, 0x12345678 becomes 0x78563412.

Parameters **`value`** – **[in]** Value to reverse

Returns Reversed value

`__STATIC_FORCEINLINE uint32_t __REV16 (uint32_t value)`

Reverse byte order (16 bit)

Reverses the byte order within each halfword of a word. For example, 0x12345678 becomes 0x34127856.

Parameters **`value`** – **[in]** Value to reverse

Returns Reversed value

`__STATIC_FORCEINLINE int16_t __REVSH (int16_t value)`

Reverse byte order (16 bit)

Reverses the byte order in a 16-bit value and returns the signed 16-bit result. For example, 0x0080 becomes 0x8000.

Parameters **`value`** – **[in]** Value to reverse

Returns Reversed value

`__STATIC_FORCEINLINE uint32_t __ROR (uint32_t op1, uint32_t op2)`

Rotate Right in unsigned value (32 bit)

Rotate Right (immediate) provides the value of the contents of a register rotated by a variable number of bits.

Parameters

- **op1** – **[in]** Value to rotate
- **op2** – **[in]** Number of Bits to rotate(0-31)

Returns Rotated value

`__STATIC_FORCEINLINE uint32_t __RBIT (uint32_t value)`

Reverse bit order of value.

Reverses the bit order of the given value.

Parameters **value** – **[in]** Value to reverse

Returns Reversed value

`__STATIC_FORCEINLINE uint8_t __CLZ (uint32_t data)`

Count leading zeros.

Counts the number of leading zeros of a data value.

Parameters **data** – **[in]** Value to count the leading zeros

Returns number of leading zeros in value

The prebuilt NMSIS-DSP and NMSIS-NN libraries without dsp are also provided in HummingBird SDK, see `NMSIS/Library/` folder.

Note:

- To support RT-Thread in HBird-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 = .;
KEEP(*(SORT(.rti_fn*)))
__rt_init_end = .;
/* section information for finsh shell */
. = ALIGN(4);
__fsymtab_start = .;
KEEP(*(FSymTab))
```

(continues on next page)

(continued from previous page)

```
__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 HummingBird 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 HummingBird 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 HummingBird 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`
  - PLIC interrupts are disabled and priorities set to 0
  - Global interrupt is disabled

Note:

- If you want to learn more about SoC, please click *SoC* (page 118)
- If you want to learn more about Board, please click *Board* (page 122)
- If you want to learn more about Peripheral, please click *Peripheral* (page 127)

5.3 SoC

5.3.1 HummingBird SoC

HummingBird SoC is an evaluation FPGA SoC based on HummingBird RISC-V Core for customer to evaluate HummingBird Process Core.

Note: HummingBird SoC is no longer maintained now, there is a v2 version, please click *HummingBird SoC V2* (page 121) to learn about it.

To get the up to date documentation about this SoC, please click:

- HummingBird SoC project source code[26]

Overview

To easy user to evaluate HummingBird RISC-V Processor Core, the prototype SoC (called Hummingbird SoC) is provided for evaluation purpose.

This prototype SoC includes:

- Processor Core, it can be RISC-V 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 *HummingBird SoC Diagram* (page 119)

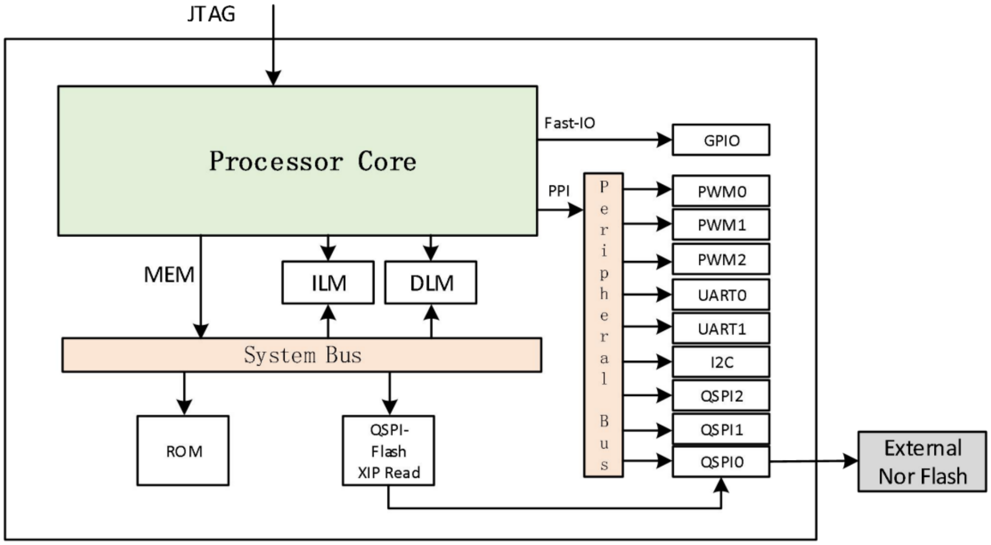
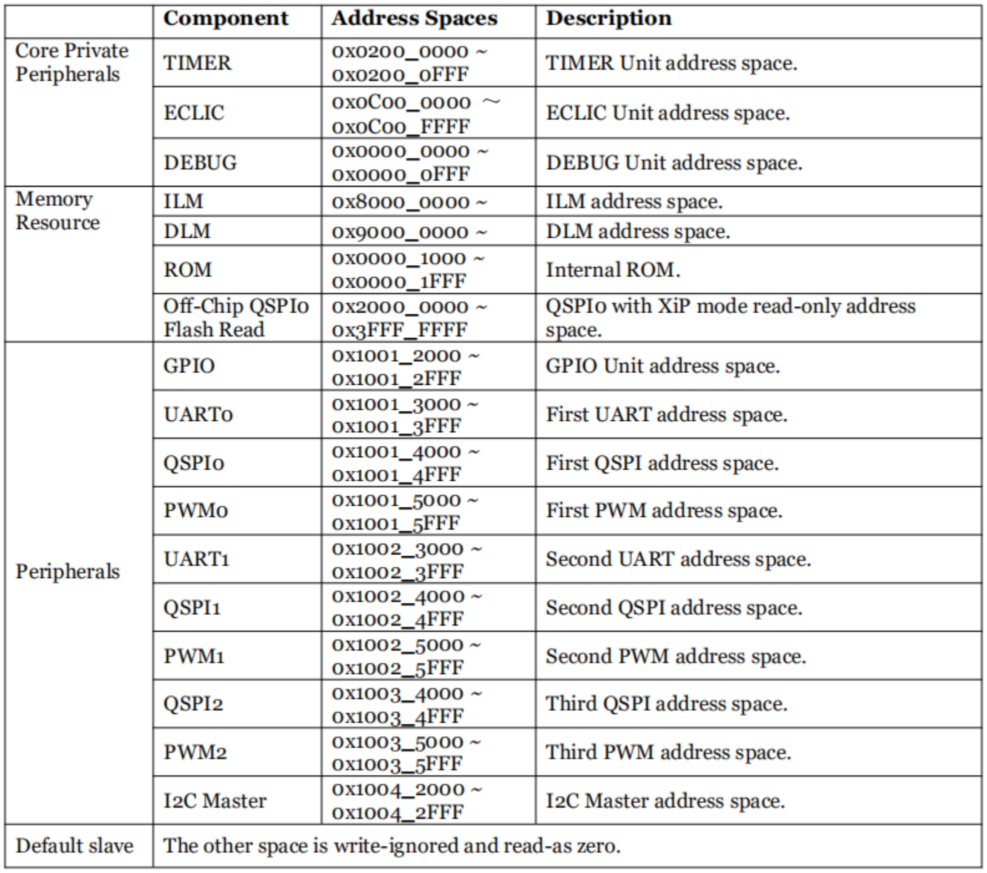


Fig. 1: HummingBird SoC Diagram

The SoC memory map for SoC resources is as below *HummingBird SoC Memory Map* (page 120)

If you want to learn more about this evaluation SoC, please check HummingBird SoC project source code[27].

[26] https://github.com/SI-RISCV/e200_opensource
[27] https://github.com/SI-RISCV/e200_opensource



	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: HummingBird SoC Memory Map

Supported Boards

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

- *HummingBird Evaluation Kit* (page 122)

Usage

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

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

5.3.2 HummingBird SoC V2

HummingBird SoC V2 is an evaluation FPGA SoC based on HummingBird RISC-V Core for customer to evaluate HummingBird Process Core.

To get the up to date documentation about this SoC, please click:

- HummingBird SoC V2 online documentation[28]
- HummingBird SoC V2 project source code[29]

Overview

To easy user to evaluate HummingBird RISC-V Processor Core, the prototype SoC (called Hummingbird SoC) is provided for evaluation purpose.

This prototype SoC includes:

- Processor Core, it can be RISC-V 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 *HummingBird V2 SoC Diagram* (page 122)

If you want to learn more about this evaluation SoC, please click HummingBird SoC V2 online documentation[30].

[28] https://doc.nucleisys.com/hbirdv2

[29] https://github.com/riscv-mcu/e203_hbirdv2

[30] https://doc.nucleisys.com/hbirdv2

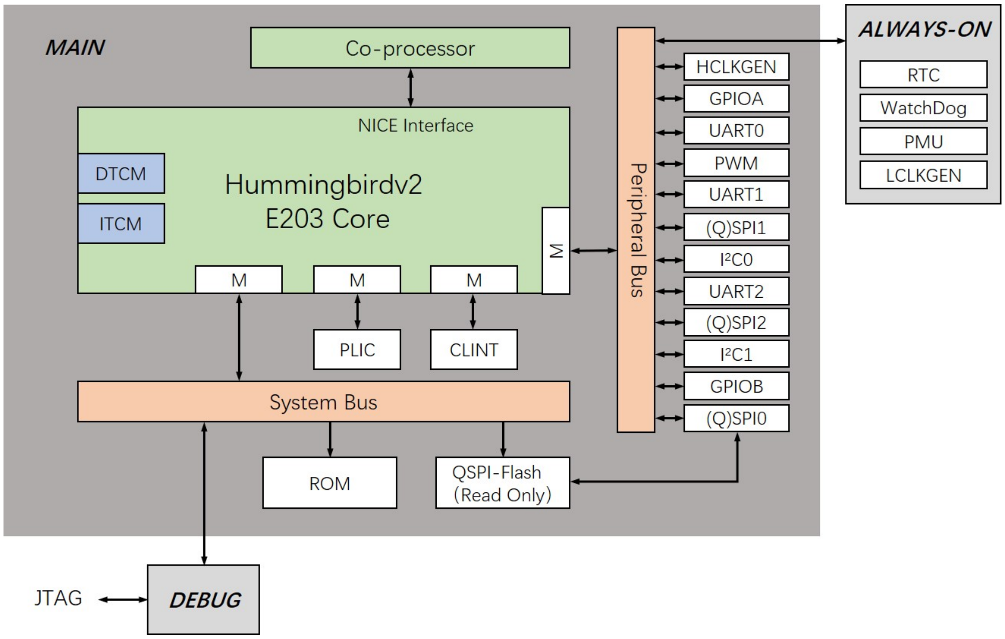


Fig. 3: HummingBird V2 SoC Diagram

Supported Boards

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

- *DDR200T Evaluation Kit* (page 124)
- *MCU200T Evaluation Kit* (page 126)

Usage

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

```
# Choose SoC to be hbird
# the following command will build application
# using default hbird SoC based board
# defined in Build System and application Makefile
make SOC=hbirdv2 all
```

5.4 Board

5.4.1 HummingBird Evaluation Kit

Overview

Nuclei have customized a FPGA evaluation board (called Hummingbird Evaluation Kit), which can be programmed with HummingBird SoC FPGA bitstream.

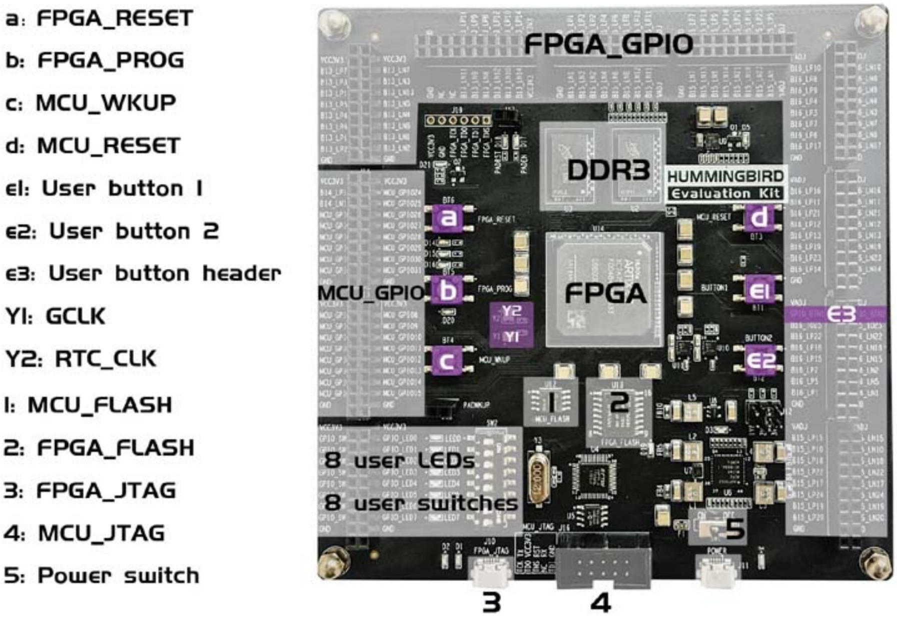


Fig. 4: HummingBird FPGA Evaluation Kit

Click HummingBird FPGA Evaluation Kit Board Documents[31] to access the documents of this board.

Setup

Follow the guide in HummingBird FPGA Evaluation Kit Board Documents[32] to setup the board, make sure the following items are set correctly:

- Use **Hummingbird 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.
- The HummingBird SoC FPGA bitstream with HummingBird RISC-V evaluation core inside is programmed to this board.
- Following steps in board user manual to setup JTAG drivers for your development environment

[31] https://nucleisys.com/developboard.php

[32] https://nucleisys.com/developboard.php

How to use

For **HummingBird Evaluation board**:

- **DOWNLOAD** support all the modes list in *DOWNLOAD* (page 26)
- **CORE** support all the cores list in *CORE* (page 27)

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

```
# Clean the application with DOWNLOAD=ilm CORE=e203
make SOC=hbird BOARD=hbird_eval DOWNLOAD=ilm CORE=e203 clean
# Build the application with DOWNLOAD=ilm CORE=e203
make SOC=hbird BOARD=hbird_eval DOWNLOAD=ilm CORE=e203 all
# Upload the application using openocd and gdb with DOWNLOAD=ilm CORE=e203
make SOC=hbird BOARD=hbird_eval DOWNLOAD=ilm CORE=e203 upload
# Debug the application using openocd and gdb with DOWNLOAD=ilm CORE=e203
make SOC=hbird BOARD=hbird_eval DOWNLOAD=ilm CORE=e203 debug
```

Note:

- You can change the value passed to **CORE** according to the HummingBird RISC-V Core the HummingBird 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 HummingBird board due to program running in Flash is really too slow. If you want to try these demos, please use `ilm` or `flash` download mode.

5.4.2 DDR200T Evaluation Kit

Overview

Nuclei have customized a FPGA evaluation board (called DDR200T Evaluation Kit), which can be programmed with HummingBird SoC FPGA bitstream.

Click DDR200T Evaluation Kit Board Documents[33] to access the documents of this board.

Setup

Follow the guide in DDR200T Evaluation Kit Board Documents[34] to setup the board, make sure the following items are set correctly:

- Use **Hummingbird 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.
- The HummingBird SoC FPGA bitstream with HummingBird RISC-V evaluation core inside is programmed to this board.

[33] https://nucleisys.com/developboard.php

[34] https://nucleisys.com/developboard.php

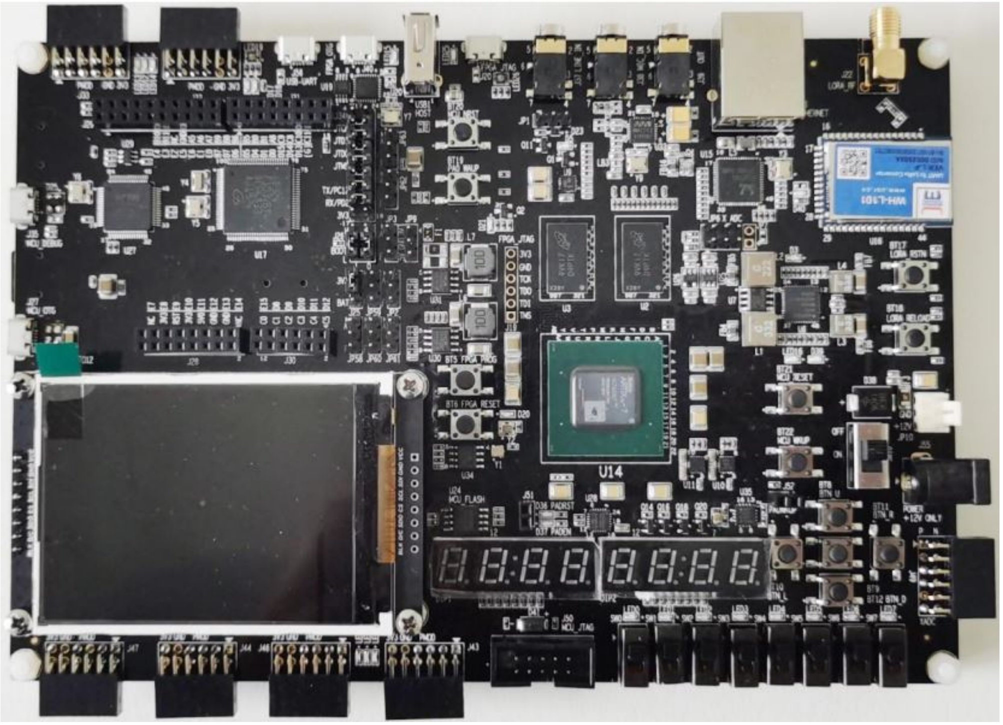


Fig. 5: DDR200T Evaluation Kit

- Following steps in board user manual to setup JTAG drivers for your development environment

How to use

For **DDR200T Evaluation board**:

- **DOWNLOAD** support all the modes list in *DOWNLOAD* (page 26)
- **CORE** support all the cores list in *CORE* (page 27)

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

```
# Clean the application with DOWNLOAD=ilm CORE=e203
make SOC=hbirdv2 BOARD=ddr200t DOWNLOAD=ilm CORE=e203 clean
# Build the application with DOWNLOAD=ilm CORE=e203
make SOC=hbirdv2 BOARD=ddr200t DOWNLOAD=ilm CORE=e203 all
# Upload the application using openocd and gdb with DOWNLOAD=ilm CORE=e203
make SOC=hbirdv2 BOARD=ddr200t DOWNLOAD=ilm CORE=e203 upload
# Debug the application using openocd and gdb with DOWNLOAD=ilm CORE=e203
make SOC=hbirdv2 BOARD=ddr200t DOWNLOAD=ilm CORE=e203 debug
```

Note:

- You can change the value passed to **CORE** according to the HummingBird RISC-V Core the HummingBird 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 HummingBird board due to program running in Flash is really too slow. If you want to try these demos, please use `ilm` or `flash` download mode.

5.4.3 MCU200T Evaluation Kit

Overview

Nuclei have customized a FPGA evaluation board (called MCU200T Evaluation Kit), which can be programmed with HummingBird SoC FPGA bitstream.

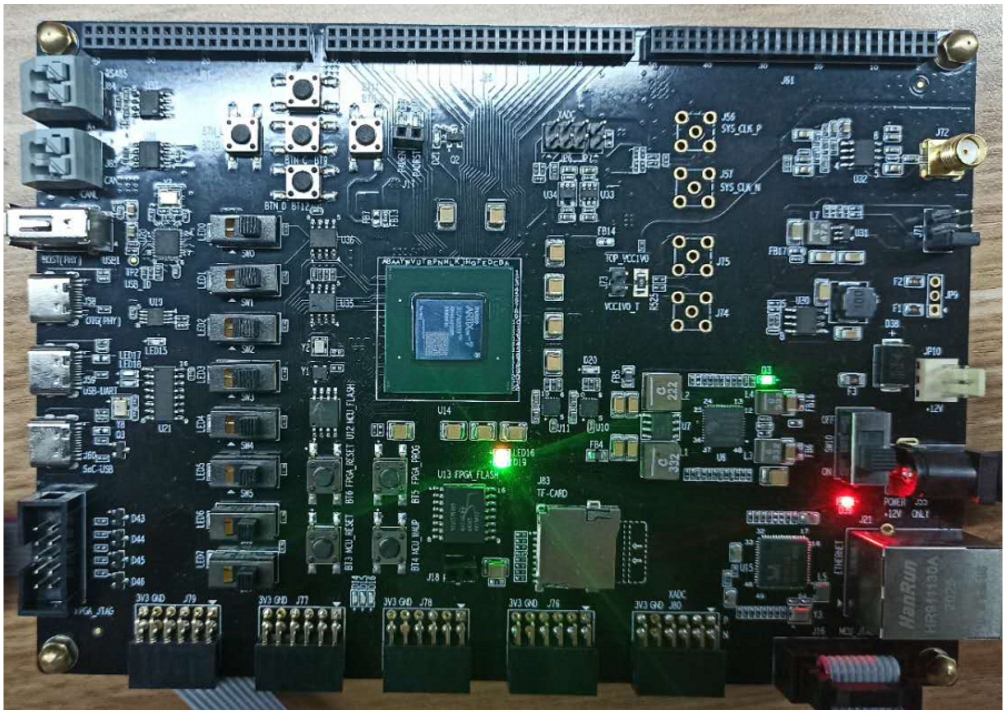


Fig. 6: MCU200T Evaluation Kit

Click MCU200T Evaluation Kit Board Documents[35] to access the documents of this board.

Setup

Follow the guide in MCU200T Evaluation Kit Board Documents[36] to setup the board, make sure the following items are set correctly:

- Use **Hummingbird 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.
- The HummingBird SoC FPGA bitstream with HummingBird RISC-V evaluation core inside is programmed to this board.

[35] https://nucleisys.com/developboard.php

[36] https://nucleisys.com/developboard.php

- Following steps in board user manual to setup JTAG drivers for your development environment

How to use

For **MCU200T Evaluation board**:

- **DOWNLOAD** support all the modes list in *DOWNLOAD* (page 26)
- **CORE** support all the cores list in *CORE* (page 27)

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

```
# Clean the application with DOWNLOAD=ilm CORE=e203
make SOC=hbirdv2 BOARD=mcu200t DOWNLOAD=ilm CORE=e203 clean
# Build the application with DOWNLOAD=ilm CORE=e203
make SOC=hbirdv2 BOARD=mcu200t DOWNLOAD=ilm CORE=e203 all
# Upload the application using openocd and gdb with DOWNLOAD=ilm CORE=e203
make SOC=hbirdv2 BOARD=mcu200t DOWNLOAD=ilm CORE=e203 upload
# Debug the application using openocd and gdb with DOWNLOAD=ilm CORE=e203
make SOC=hbirdv2 BOARD=mcu200t DOWNLOAD=ilm CORE=e203 debug
```

Note:

- You can change the value passed to **CORE** according to the HummingBird RISC-V Core the HummingBird 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 HummingBird board due to program running in Flash is really too slow. If you want to try these demos, please use `ilm` or `flash` download mode.

5.5 Peripheral

5.5.1 Overview

Regarding the peripheral support(such as UART, GPIO, SPI, I2C and etc.) in HummingBird 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 HummingBird SDK, you can use include the `hbird_sdk_soc.h`, then you can write application which only use the resources of RISC-V Core.

If you want to use all the board resources, you can include the `hbird_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 `hbird_sdk_soc.h` of each SoC, and `hbird_sdk_hal.h` of each board.

For SoC firmware library APIs:

- You can find the **HummingBird SoC firmware library APIs** in `SoC/hbird/Common/Include`

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

For Board related APIs:

- You can find the **HummingBird EVAL Board related APIs** in `SoC/hbird/Board/hbird_eval/Include`

If you just want to use all the APIs of Board and SoC, you just need to include `hbird_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 HummingBird 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 HummingBird SDK, please don't reconfigure `SysTimer` and `SysTimer Software Interrupt`, since it is already used by RTOS portable code.

5.6.2 FreeRTOS

FreeRTOS[37] 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 core internal interrupt, and `SysTimer Software Interrupt` is initialized as core internal interrupt.

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

In HummingBird 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:

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

- You can check the `application\freertos\demo` for reference
- Current version of FreeRTOS used in HummingBird 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[38] 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 HummingBird RISC-V processors.

In HummingBird 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 HummingBird SDK is `V2.93.00`
- If you want to change the OS ticks per seconds, you can change the `OS_TICKS_PER_SEC` defined in `os_cfg.h`

Warning:

- For HummingBird 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.

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

5.6.4 RT-Thread

RT-Thread (page 130) 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[39], which only provide kernel code of RT-Thread, which is easy to be intergated with HummingBird 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

In HummingBird 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

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`

5.7 Application

5.7.1 Overview

In HummingBird 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`

If you want to develop your own application in HummingBird SDK, please click *Application Development* (page 34) to learn more about it.

The following applications are running using HummingBird board.

[39] https://github.com/RT-Thread/rtthread-nano

5.7.2 Bare-metal applications

helloworld

This helloworld application[40] 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 HummingBird SDK environment
# cd to the helloworld directory
cd application/baremetal/helloworld
# Clean the application first
make SOC=hbird BOARD=hbird_eval CORE=e203 clean
# Build and upload the application
make SOC=hbird BOARD=hbird_eval CORE=e203 upload
```

Expected output as below:

```
HummingBird SDK Build Time: Jul 16 2020, 11:18:08
Download Mode: ILM
CPU Frequency 15999631 Hz
MISA: 0x40001105
MISA: RV32IMAC
0: Hello World From RISC-V Processor!
1: Hello World From RISC-V Processor!
2: Hello World From RISC-V Processor!
3: Hello World From RISC-V Processor!
4: Hello World From RISC-V Processor!
5: Hello World From RISC-V Processor!
6: Hello World From RISC-V Processor!
7: Hello World From RISC-V Processor!
8: Hello World From RISC-V Processor!
9: Hello World From RISC-V Processor!
10: Hello World From RISC-V Processor!
11: Hello World From RISC-V Processor!
12: Hello World From RISC-V Processor!
13: Hello World From RISC-V Processor!
14: Hello World From RISC-V Processor!
15: Hello World From RISC-V Processor!
16: Hello World From RISC-V Processor!
17: Hello World From RISC-V Processor!
18: Hello World From RISC-V Processor!
19: Hello World From RISC-V Processor!
```

[40] https://github.com/riscv-mcu/hbird-sdk/tree/master/application/baremetal/helloworld

demo_timer

This demo_timer application[41] is used to demostrate how to use the CORE TIMER API including the Timer Interrupt and Timer Software Interrupt.

- Both interrupts are registered as 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 HummingBird SDK environment
# cd to the demo_timer directory
cd application/baremetal/demo_timer
# Clean the application first
make SOC=hbird BOARD=hbird_eval CORE=e203 clean
# Build and upload the application
make SOC=hbird BOARD=hbird_eval CORE=e203 upload
```

Expected output as below:

```
HummingBird SDK Build Time: Jul 16 2020, 11:43:13
Download Mode: ILM
CPU Frequency 16006512 Hz
MTimer IRQ handler 1
init timer and start
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
MTimer SW IRQ handler 9
MTimer SW IRQ handler 10
MTimer msip and mtip interrupt test finish and pass
```

[41] https://github.com/riscv-mcu/hbird-sdk/tree/master/application/baremetal/demo_timer

demo_plic

This demo_plic application[42] is used to demostrate how to use the PLIC 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_plic on HummingBird target

COMMON_FLAGS	text(bytes)	data(bytes)	bss(bytes)	total(bytes)
	9608	112	2500	12220
-flto	9552	112	2500	12164
-Os	7316	112	2500	9928
-Os -flto	6942	112	2500	9554
-Os -msave-restore -fno-unroll-loops	7360	112	2500	9972
-Os -msave-restore -fno-unroll-loops -flto	7008	112	2500	9620

- This is an example of triggering an external interrupt
- Two GPIO rising edge interrupts are used
- When the button 1 and button 2 are pressed respectively the program triggers the external rising edge interrupt and the interrupt processing function will show which button triggered the interrupt on the serial port

How to run this application:

```
# Assume that you can set up the Tools and HummingBird SDK environment
# cd to the demo_plic directory
cd application/baremetal/demo_plic
# Clean the application first
make SOC=hbird BOARD=hbird_eval CORE=e203 clean
# Build and upload the application
make SOC=hbird BOARD=hbird_eval CORE=e203 upload
# Press button1 and button2, see uart output
```

Expected output as below:

```
HummingBird SDK Build Time: Jul 16 2020, 16:37:14
Download Mode: ILM
CPU Frequency 15999303 Hz
Enter Button 1 interrupt
Enter Button 1 interrupt
Enter Button 2 interrupt
Enter Button 2 interrupt
```

[42] https://github.com/riscv-mcu/hbird-sdk/tree/master/application/baremetal/demo_plic

demo_dsp

This demo_dsp application[43] is used to demostrate how to NMSIS-DSP API.

- Mainly show how we can use DSP library without dsp instructions and header files.
- It mainly demo the `riscv_conv_xx` functions and its reference functions

Note:

- For other HummingBird 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 HummingBird SDK environment
# cd to the demo_dsp directory
cd application/baremetal/demo_dsp
# Clean the application first
make SOC=hbird BOARD=hbird_eval CORE=e203 DSP_ENABLE=OFF clean
# Build and upload the application
make SOC=hbird BOARD=hbird_eval CORE=e203 DSP_ENABLE=OFF upload
```

Expected output as below:

```
HummingBird SDK Build Time: Jul 16 2020, 15:55:06
Download Mode: ILM
CPU Frequency 16006512 Hz
CSV, riscv_conv_q31, 4103925
CSV, ref_conv_q31, 12979250
SUCCESS, riscv_conv_q31
CSV, riscv_conv_q15, 437418
CSV, ref_conv_q15, 882230
SUCCESS, riscv_conv_q15
CSV, riscv_conv_q7, 839
CSV, ref_conv_q7, 2382
SUCCESS, riscv_conv_q7
CSV, riscv_conv_fast_q15, 357503
CSV, ref_conv_fast_q15, 774856
SUCCESS, riscv_conv_fast_q15
CSV, riscv_conv_fast_q31, 1918358
CSV, ref_conv_fast_q31, 13692367
SUCCESS, riscv_conv_fast_q31
CSV, riscv_conv_opt_q15, 524310
CSV, ref_conv_opt_q15, 882232
SUCCESS, riscv_conv_opt_q15
CSV, riscv_conv_opt_q7, 1535
CSV, ref_conv_opt_q7, 2382
SUCCESS, riscv_conv_opt_q7
CSV, riscv_conv_fast_opt_q15, 454263
```

(continues on next page)

[43] https://github.com/riscv-mcu/hbird-sdk/tree/master/application/baremetal/demo_dsp

(continued from previous page)

```
CSV, ref_conv_fast_opt_q15, 789929
SUCCESS, riscv_conv_fast_opt_q15
all test are passed. Well done!
```

coremark

This coremark benchmark application[44] 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 HummingBird 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
- **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 `HummingBird` based boards supported in HummingBird SDK, we suggest to change `-DITERATIONS=500` to `-DITERATIONS=5000`

How to run this application:

```
# Assume that you can set up the Tools and HummingBird SDK environment
# cd to the coremark directory
cd application/baremetal/benchmark/coremark
# Clean the application first
make SOC=hbird BOARD=hbird_eval CORE=e203 clean
# Build and upload the application
make SOC=hbird BOARD=hbird_eval CORE=e203 upload
```

Expected output as below:

```
HummingBird SDK Build Time: Jul 16 2020, 16:01:58
Download Mode: ILM
CPU Frequency 15999631 Hz
Start to run coremark for 500 iterations
2K performance run parameters for coremark.
CoreMark Size    : 666
Total ticks      : 233879271
Total time (secs): 14.617908
Iterations/Sec   : 34.204621
```

(continues on next page)

[44] https://github.com/riscv-mcu/hbird-sdk/tree/master/application/baremetal/benchmark/coremark

(continued from previous page)

```
Iterations       : 500
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      : 0xa14c
Correct operation validated. See readme.txt for run and reporting rules.
CoreMark 1.0 : 34.204621 / 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: 500
     (total_ticks is: 233879271
 (*) Assume the core running at 1 MHz
     So the CoreMark/MHz can be caculated by:
     (Iterations*1000000/total_ticks) = 2.137855 CoreMark/MHz
```

dhrystone

This dhrystone benchmark application[45] 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 HummingBird 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 HummingBird SDK environment
# cd to the dhrystone directory
cd application/baremetal/benchmark/dhrystone
```

(continues on next page)

[45] https://github.com/riscv-mcu/hbird-sdk/tree/master/application/baremetal/benchmark/dhrystone

(continued from previous page)

```
# Clean the application first
make SOC=hbird BOARD=hbird_eval CORE=e203 clean
# Build and upload the application
make SOC=hbird BOARD=hbird_eval CORE=e203 upload
```

Expected output as below:

```
HummingBird SDK Build Time: Jul 16 2020, 16:15:27
Download Mode: ILM
CPU Frequency 15999959 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
        should be:   Number_Of_Runs + 10
Ptr_Glob->
  Ptr_Comp:          -1879035440
        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:          -1879035440
        should be:   (implementation-dependent), same as above
  Discr:             0
        should be:   0
  Enum_Comp:         1
        should be:   1
  Int_Comp:          18
```

(continues on next page)

(continued from previous page)

```
        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:
220000037
       So the DMIPS/MHz can be caculated by:
       1000000/(User_Cycle/Number_Of_Runs)/1757 = 1.293527 DMIPS/MHz
```

whetstone

This whetstone benchmark application[46] 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 HummingBird 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
- 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 HummingBird SDK environment
# cd to the whetstone directory
cd application/baremetal/benchmark/whetstone
# Clean the application first
make SOC=hbird BOARD=hbird_eval CORE=e203 clean
# Build and upload the application
make SOC=hbird BOARD=hbird_eval CORE=e203 upload
```

Expected output as below:

```
HummingBird SDK Build Time: Jul 16 2020, 16:18:26
Download Mode: ILM
CPU Frequency 15997337 Hz
```

(continues on next page)

[46] https://github.com/riscv-mcu/hbird-sdk/tree/master/application/baremetal/benchmark/whetstone

(continued from previous page)

```

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

Use 1  passes (x 100)

          Single Precision C/C++ Whetstone Benchmark

Loop content                  Result              MFLOPS      MOPS   Seconds

N1 floating point -1.12475013732910156         0.144              0.133
N2 floating point -1.12274742126464844         0.144              0.930
N3 if then else    1.00000000000000000                   0.000    0.000
N4 fixed point    12.00000000000000000                   0.806    0.391
N5 sin,cos etc.    0.49909299612045288                   0.014    6.086
N6 floating point  0.99999982118606567         0.128              4.225
N7 assignments     3.00000000000000000                  72.090    0.003
N8 exp,sqrt etc.   0.75110614299774170                   0.010    3.664

MWIPS                                              0.648             15.431


MWIPS/MHz                                          0.041             15.431
```

5.7.3 FreeRTOS applications

demo

This freertos demo application[47] is show basic freertos task functions.

- Two freertos tasks are created
- A software timer is created

In HummingBird SDK, we provided code and Makefile for this `freertos demo` application.

- **RTOS = FreeRTOS** is added in its Makefile to include FreeRTOS service
- The **configTICK_RATE_HZ** in `FreeRTOSConfig.h` is set to 200, 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 HummingBird SDK environment
# cd to the freertos demo directory
cd application/freertos/demo
# Clean the application first
make SOC=hbird BOARD=hbird_eval CORE=e203 clean
# Build and upload the application
make SOC=hbird BOARD=hbird_eval CORE=e203 upload
```

[47] https://github.com/riscv-mcu/hbird-sdk/tree/master/application/freertos/demo

Expected output as below:

```
HummingBird SDK Build Time: Jul 16 2020, 17:15:24
Download Mode: ILM
CPU Frequency 15998320 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[48] is show basic ucosii task functions.

- 4 tasks are created
- 1 task is created first, and then create 3 other tasks and then suspend itself

In HummingBird 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 200, you can change it to other number according to your requirement.

How to run this application:

[48] https://github.com/riscv-mcu/hbird-sdk/tree/master/application/ucosii/demo

```
# Assume that you can set up the Tools and HummingBird SDK environment
# cd to the ucosii demo directory
cd application/ucosii/demo
# Clean the application first
make SOC=hbird BOARD=hbird_eval CORE=e203 clean
# Build and upload the application
make SOC=hbird BOARD=hbird_eval CORE=e203 upload
```

Expected output as below:

```
HummingBird SDK Build Time: Jul 16 2020, 17:20:13
Download Mode: ILM
CPU Frequency 15998320 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
```

5.7.5 RT-Thread applications

demo

This rt-thread demo application[49] 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 HummingBird 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 *200*, 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 HummingBird SDK environment
# cd to the rtthread demo directory
cd application/rtthread/demo
# Clean the application first
make SOC=hbird BOARD=hbird_eval CORE=e203 clean
# Build and upload the application
make SOC=hbird BOARD=hbird_eval CORE=e203 upload
```

Expected output as below:

```
HummingBird SDK Build Time: Jul 16 2020, 17:22:44
Download Mode: ILM
CPU Frequency 16000286 Hz

 \ | /
- RT -     Thread Operating System
 / | \     3.1.3 build Jul 16 2020
 2006 - 2019 Copyright by rt-thread team
Main thread count: 0
thread 0 count: 0
thread 1 count: 0
thread 2 count: 0
thread 3 count: 0
thread 4 count: 0
thread 0 count: 1
thread 1 count: 1
thread 2 count: 1
thread 3 count: 1
thread 4 count: 1
Main thread count: 1
thread 0 count: 2
thread 1 count: 2
thread 2 count: 2
thread 3 count: 2
thread 4 count: 2
thread 0 count: 3
```

(continues on next page)

[49] https://github.com/riscv-mcu/hbird-sdk/tree/master/application/rtthread/demo

(continued from previous page)

```
thread 1 count: 3
thread 2 count: 3
thread 3 count: 3
thread 4 count: 3
Main thread count: 2
thread 0 count: 4
thread 1 count: 4
thread 2 count: 4
thread 3 count: 4
thread 4 count: 4
thread 0 count: 5
thread 1 count: 5
thread 2 count: 5
thread 3 count: 5
thread 4 count: 5
Main thread count: 3
thread 0 count: 6
thread 1 count: 6
thread 2 count: 6
thread 3 count: 6
thread 4 count: 6
thread 0 count: 7
thread 1 count: 7
thread 2 count: 7
thread 3 count: 7
thread 4 count: 7
Main thread count: 4
thread 0 count: 8
thread 1 count: 8
thread 2 count: 8
thread 3 count: 8
thread 4 count: 8
thread 0 count: 9
thread 1 count: 9
thread 2 count: 9
thread 3 count: 9
thread 4 count: 9
```

msh

This rt-thread msh application[50] demonstrates a shell in serial console which is a component of rt-thread.

- `MSH_CMD_EXPORT(hbird, msh hbird demo)` exports a command `hbird` to shell

In HummingBird 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 *200*, you can change it to other number according to your requirement.

[50] https://github.com/riscv-mcu/hbird-sdk/tree/master/application/rtthread/msh

How to run this application:

```
# Assume that you can set up the Tools and HummingBird SDK environment
# cd to the rtthread msh directory
cd application/rtthread/msh
# Clean the application first
make SOC=hbird BOARD=hbird_eval CORE=e203 clean
# Build and upload the application
make SOC=hbird BOARD=hbird_eval CORE=e203 upload
```

Expected output as below:

```
HummingBird SDK Build Time: Nov 25 2020, 09:18:36
Download Mode: FLASH
CPU Frequency 15978659 Hz

\ | /
- RT -     Thread Operating System
/ | \     3.1.3 build Nov 25 2020
2006 - 2019 Copyright by rt-thread team
Hello RT-Thread!
msh >
RT-Thread shell commands:
list_timer list_mailbox list_sem list_thread version ps help hbird
msh >hbird
Hello HBird SDK!
msh >
```

CHAPTER

SIX

CHANGELOG

6.1 V0.1.4

This is release version `0.1.4` of HBird SDK.

- SoC
  - Fix PLIC example fail in Nuclei Studio, due to `SOC_HBIRDV2` not defined in npk.yml
- NMSIS
  - Fix typo of `global:  true` in npk.yml
- CI
  - Update gitlab & github ci workflow

6.2 V0.1.3

This is release version `0.1.3` of HBird 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.
  - 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 **<HBIRD_SDK_ROOT>/Makefile**, you can clean all the applications defined by `EXTRA_APP_ROOTDIRS` variable
  - Fix `size` target of build system

- SoC
    - hbird and hbirdv2 SoC cores only support e203 and e203e now.

6.3 V0.1.2

This is official `0.1.2` of HummingBird SDK.

Here are the main changes since last release:

- SOC
    - More more newlib stub functions for hbird and hbirdv2 SoC
- doc
    - Update changelog
    - Add rt-thread msh application doc
- application
    - Add rt-thread msh application
- Build
    - Add `RTTHREAD_MSH` makefile variable which is valid only for RTThread
- OS
    - Add RT-Thread MSH shell component into RT-Thread source code
- CI
    - Add initial github workflow support for building documentation and sdk

6.4 V0.1.1

This is official `0.1.1` of HummingBird SDK.

Here are the main changes since last release:

- SOC
    - More drivers are added to hbirdv2
- doc
    - Update changelog
- application
    - Fix typos in rt-thread application
    - Update freertos application

6.5 V0.1.0

This is official release `0.1.0` of HummingBird SDK.

Here are the main features of this release:

- HummingBird SDK is developed based on **Nuclei SDK version 0.2.4** release.
- Support Windows and Linux development in command line using Make
- Support HummingBird FPGA evaluation board and HummingBird FPGA DDR-200T evaluation board
  - The **HummingBird FPGA evaluation board** is used to run evaluation FPGA bitstream of HummingBird E201, E203, E205 processor cores
  - The **HummingBird FPGA DDR-200T evaluation board** is used to run evaluation FPGA bitstream of HummingBird E201, E203, E205 processor cores
- Support different download modes *flashxip*, *ilm*, *flash* for HummingBird FPGA evaluation board
- Support different RTOSes such as FreeRTOS, UCOS-II and RT-Thread
- This *hbird-sdk* is forked from nuclei-sdk[51] , and adapted for opensource HummingBird RISC-V Core.

[51] https://github.com/nuclei-software/nuclei-sdk

CHAPTER

SEVEN

FAQ

7.1 Why I can’t download application in Windows?

If you met the following issue as below message showed:

```
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.
"Successfully uploaded hello_world.elf "
```

Please check whether your driver is installed successfully as the board user manual described, especially, for **HummingBird Evaluation** boards, you need to download the **HummingBird Debugger Windows Driver** from https://nucleisys.com/developboard.php, and install it.

Note: The USB driver might lost when you re-plug the USB port, you might need to reinstall the driver.

7.2 Why I can’t download application in Linux?

Please check that whether you have followed the board user manual to setup the USB JTAG drivers correctly. The windows steps and linux steps are different, please take care.

7.3 Why the provided application is not running correctly in my HummingBird Evaluation Board?

Please check the following items:

1. Did you program the correct HummingBird 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 HummingBird 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?

CHAPTER EIGHT

LICENSE

```
                              Apache License
                        Version 2.0, January 2004
                     http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

   "License" shall mean the terms and conditions for use, reproduction,
   and distribution as defined by Sections 1 through 9 of this document.

   "Licensor" shall mean the copyright owner or entity authorized by
   the copyright owner that is granting the License.

   "Legal Entity" shall mean the union of the acting entity and all
   other entities that control, are controlled by, or are under common
   control with that entity. For the purposes of this definition,
   "control" means (i) the power, direct or indirect, to cause the
   direction or management of such entity, whether by contract or
   otherwise, or (ii) ownership of fifty percent (50%) or more of the
   outstanding shares, or (iii) beneficial ownership of such entity.

   "You" (or "Your") shall mean an individual or Legal Entity
   exercising permissions granted by this License.

   "Source" form shall mean the preferred form for making modifications,
   including but not limited to software source code, documentation
   source, and configuration files.

   "Object" form shall mean any form resulting from mechanical
   transformation or translation of a Source form, including but
   not limited to compiled object code, generated documentation,
   and conversions to other media types.

   "Work" shall mean the work of authorship, whether in Source or
   Object form, made available under the License, as indicated by a
   copyright notice that is included in or attached to the work
   (an example is provided in the Appendix below).

   "Derivative Works" shall mean any work, whether in Source or Object
```

(continues on next page)

(continued from previous page)

```
   form, that is based on (or derived from) the Work and for which the
   editorial revisions, annotations, elaborations, or other modifications
   represent, as a whole, an original work of authorship. For the purposes
   of this License, Derivative Works shall not include works that remain
   separable from, or merely link (or bind by name) to the interfaces of,
   the Work and Derivative Works thereof.

   "Contribution" shall mean any work of authorship, including
   the original version of the Work and any modifications or additions
   to that Work or Derivative Works thereof, that is intentionally
   submitted to Licensor for inclusion in the Work by the copyright owner
   or by an individual or Legal Entity authorized to submit on behalf of
   the copyright owner. For the purposes of this definition, "submitted"
   means any form of electronic, verbal, or written communication sent
   to the Licensor or its representatives, including but not limited to
   communication on electronic mailing lists, source code control systems,
   and issue tracking systems that are managed by, or on behalf of, the
   Licensor for the purpose of discussing and improving the Work, but
   excluding communication that is conspicuously marked or otherwise
   designated in writing by the copyright owner as "Not a Contribution."

   "Contributor" shall mean Licensor and any individual or Legal Entity
   on behalf of whom a Contribution has been received by Licensor and
   subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of
   this License, each Contributor hereby grants to You a perpetual,
   worldwide, non-exclusive, no-charge, royalty-free, irrevocable
   copyright license to reproduce, prepare Derivative Works of,
   publicly display, publicly perform, sublicense, and distribute the
   Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of
   this License, each Contributor hereby grants to You a perpetual,
   worldwide, non-exclusive, no-charge, royalty-free, irrevocable
   (except as stated in this section) patent license to make, have made,
   use, offer to sell, sell, import, and otherwise transfer the Work,
   where such license applies only to those patent claims licensable
   by such Contributor that are necessarily infringed by their
   Contribution(s) alone or by combination of their Contribution(s)
   with the Work to which such Contribution(s) was submitted. If You
   institute patent litigation against any entity (including a
   cross-claim or counterclaim in a lawsuit) alleging that the Work
   or a Contribution incorporated within the Work constitutes direct
   or contributory patent infringement, then any patent licenses
   granted to You under this License for that Work shall terminate
   as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the
   Work or Derivative Works thereof in any medium, with or without
   modifications, and in Source or Object form, provided that You
   meet the following conditions:
```

(continues on next page)

(continued from previous page)

```
   (a) You must give any other recipients of the Work or
       Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices
       stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works
       that You distribute, all copyright, patent, trademark, and
       attribution notices from the Source form of the Work,
       excluding those notices that do not pertain to any part of
       the Derivative Works; and

   (d) If the Work includes a "NOTICE" text file as part of its
       distribution, then any Derivative Works that You distribute must
       include a readable copy of the attribution notices contained
       within such NOTICE file, excluding those notices that do not
       pertain to any part of the Derivative Works, in at least one
       of the following places: within a NOTICE text file distributed
       as part of the Derivative Works; within the Source form or
       documentation, if provided along with the Derivative Works; or,
       within a display generated by the Derivative Works, if and
       wherever such third-party notices normally appear. The contents
       of the NOTICE file are for informational purposes only and
       do not modify the License. You may add Your own attribution
       notices within Derivative Works that You distribute, alongside
       or as an addendum to the NOTICE text from the Work, provided
       that such additional attribution notices cannot be construed
       as modifying the License.

   You may add Your own copyright statement to Your modifications and
   may provide additional or different license terms and conditions
   for use, reproduction, or distribution of Your modifications, or
   for any such Derivative Works as a whole, provided Your use,
   reproduction, and distribution of the Work otherwise complies with
   the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,
   any Contribution intentionally submitted for inclusion in the Work
   by You to the Licensor shall be under the terms and conditions of
   this License, without any additional terms or conditions.
   Notwithstanding the above, nothing herein shall supersede or modify
   the terms of any separate license agreement you may have executed
   with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade
   names, trademarks, service marks, or product names of the Licensor,
   except as required for reasonable and customary use in describing the
   origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
   agreed to in writing, Licensor provides the Work (and each
```

(continues on next page)

(continued from previous page)

```
   Contributor provides its Contributions) on an "AS IS" BASIS,
   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
   implied, including, without limitation, any warranties or conditions
   of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
   PARTICULAR PURPOSE. You are solely responsible for determining the
   appropriateness of using or redistributing the Work and assume any
   risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
   whether in tort (including negligence), contract, or otherwise,
   unless required by applicable law (such as deliberate and grossly
   negligent acts) or agreed to in writing, shall any Contributor be
   liable to You for damages, including any direct, indirect, special,
   incidental, or consequential damages of any character arising as a
   result of this License or out of the use or inability to use the
   Work (including but not limited to damages for loss of goodwill,
   work stoppage, computer failure or malfunction, or any and all
   other commercial damages or losses), even if such Contributor
   has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
   the Work or Derivative Works thereof, You may choose to offer,
   and charge a fee for, acceptance of support, warranty, indemnity,
   or other liability obligations and/or rights consistent with this
   License. However, in accepting such obligations, You may act only
   on Your own behalf and on Your sole responsibility, not on behalf
   of any other Contributor, and only if You agree to indemnify,
   defend, and hold each Contributor harmless for any liability
   incurred by, or claims asserted against, such Contributor by reason
   of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

   To apply the Apache License to your work, attach the following
   boilerplate notice, with the fields enclosed by brackets "[]"
   replaced with your own identifying information. (Don't include
   the brackets!)  The text should be enclosed in the appropriate
   comment syntax for the file format. We also recommend that a
   file or class name and description of purpose be included on the
   same "printed page" as the copyright notice for easier
   identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0
```

(continues on next page)

(continued from previous page)

```
Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
```

CHAPTER NINE

GLOSSARY

API (Application Program Interface) A defined set of routines and protocols for building application software.

DSP (Digital Signal Processing) is the use of digital processing, such as by computers or more specialized digital signal processors, to perform a wide variety of signal processing operations.

ISR (Interrupt Service Routine) Also known as an interrupt handler, an ISR is a callback function whose execution is triggered by a hardware interrupt (or software interrupt instructions) and is used to handle high-priority conditions that require interrupting the current code executing on the processor.

NN (Neural Network) is a network or circuit of neurons, or in a modern sense, an artificial neural network, composed of artificial neurons or nodes.

XIP (eXecute In Place) a method of executing programs directly from long term storage rather than copying it into RAM, saving writable memory for dynamic data and not the static program code.

CHAPTER

TEN

APPENDIX

- **Nuclei 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/
- **HummingBird 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/
- **HummingBird SDK Documentation**: https://doc.nucleisys.com/hbird_sdk/
- **NMSIS Documentation**: https://doc.nucleisys.com/nmsis/

CHAPTER ELEVEN

INDICES AND TABLES

- genindex
- search

INDEX