wh_sdk使用说明

wh_sdk用于WH32/WH64/LDJ32的应用开发和硬件仿真，目前支持Newlib函数库，支持FT2232/Olimex/Jlink下载，使用的工具链为RISC-V GNU工具链，使用的下载器是RISC-V OpenOCD，但是它不包含有RISC-V GNU工具链和RISC-V OpenOCD，所以在使用wh_sdk时，需要另外安装RISC-V GNU的工具链和RISC-V OpenOCD。

RISC-V GNU工具链下载链接 (https://github.com/riscv/riscv-gnu-toolchain)

RISC-V OpenOCD下载链接 (https://github.com/riscv/riscv-openocd)

1. wh_sdk目录架构描述

```
* make_env_var        //Makefile脚本文件的环境变量
* Makefile            //Makefile脚本文件
* output/             //存放编译之后的文件
* script/             //存放SDK软件开发平台需要用到的脚本文件
* software/           //存放用户程序和已经写好的demo
* wh_bsp/
  * env/
        * LS_Board/
            * init.c                    //上电后的初始化程序
            * Makefile
            * openocd.cfg               //FT2232下载器的OpenOCD配置文件
            * Olimex.cfg                //Olimex下载器的OpenOCD配置文件
            * jlink.cfg                 //Jlink下载器的OpenOCD配置文件
            * Start-WH/                 //Start-WH配置表
            * WH32_DDR/                 //通用版本
                    * setting.mk        //配置文件
        * no_change/
  * include/      //所有内核、外设头文件
  * libwrap/      //移植了一些Newlib的函数库
  * wh_lib/       //所有内核、外设源文件
    * core/
    * device/
    * wh_lib.mk
  * doc/          //SDK使用文档、API文档
  * LICENSE
  * README.md
```

2. SDK中已经包含的例程(在software目录下)

为了让用户尽快熟悉wh_sdk，software目录下已经写好了一些例程，用户可以直接使用这些例程进行测试。

除了`gpio_test`例程以外，其他的所有例程都需要使用到串口，默认串口设置为**115200-8E1**;

- software/
  - coremark/
    - 用于上板的coremark程序
    - 如果想修改迭代次数，请在coremark/Makefile中修改`ITEERATIONS`的值

- dhrystone/
  - 用于上板的dhrystone程序
  - 如果想修改迭代次数，请在dhrystone/Makefile中修改`ITEERATIONS`的值
- rtc_test/
  - MTIME定时器中断
  - 如果正常工作，该程序会在串口打印如下信息：

```
M-Mode timer interrupt test Start.
I'm here 0 times
I'm here 1 times
I'm here 2 times
......
```

- soft_int_test/
  - M-Mode软件中断
  - 如果正常工作，该程序会在串口打印如下信息：

```
M-Mode software interrupt test Start.
I'm here 0 times
I'm here 1 times
I'm here 2 times
......
```

- whetstone/
  - 用于上板的whestone程序
  - 如果想要修改迭代次数，请在`whetstone.c`文件内修改`loopstart`变量的值
- pmp_test/
  - 用于测试PMP
  - 要求处理器支持PMP
  - 如果成功工作，该程序将会在串口打印如下信息：

```
PMP TEST
Attempting to write to protected address

Program has exited with code:0x0000000000000007
```

- memory_test/
  - 用于测试栈段以后（不包括栈段）的256MiB内存是否可用
  - 如果成功工作，该程序会在串口打印如下信息：

```
Leave 0x40000000 ~ 0x4001f800 alone
-------TEST START-----------------------
Start test: 0x4001f800 ~ 0x4041f800
0x4001f800 ~ 0x4041f800 is OK, Next 4 MiB
Start test: 0x4041f800 ~ 0x4081f800
0x4041f800 ~ 0x4081f800 is OK, Next 4 MiB
Start test: 0x4081f800 ~ 0x40c1f800
0x4081f800 ~ 0x40c1f800 is OK, Next 4 MiB
Start test: 0x40c1f800 ~ 0x4101f800
```

```
0x40c1f800 ~ 0x4101f800 is OK, Next 4 MiB
Start test: 0x4101f800 ~ 0x4141f800
0x4101f800 ~ 0x4141f800 is OK, Next 4 MiB
Start test: 0x4141f800 ~ 0x4181f800
0x4141f800 ~ 0x4181f800 is OK, Next 4 MiB
......
```

- smode_test/
  - 用于测试S-Mode的软件中断和TIME中断
  - 要求处理器支持S-Mode
  - 测试S-Mode的软件中断，需要打开`smode_test.c`文件内的宏定义`SMODE_SOFT_INT`，如果成功工作，该程序会在串口打印如下信息：

    ```
    I'm handling S-Mode software interrupt.
    ```

  - 测试S-Mode的TIME中断，需要打开`smode_test.c`文件内的宏定义`SMODE_TIME_INT`，如果成功工作，该程序会在串口打印如下信息：

    ```
    I'm handling S-Mode timer interrupt.
    ```

- gpio_test/
  - 流水灯，GPIO的测试程序
- uart_test/
  - 串口回显，UART0的测试程序
  - 此程序用到了串口中断
- sv32_test/
  - 用于测试SV32虚拟地址
  - 要求处理器支持S-Mode和SV32
  - 如果运行成功，则会在串口显示如下信息：

    ```
    TEST BEGIN!
    Start to test the function of sv32 system.
    Environment call from S-mode!Hardware exits S-mode!
    TEST PASS!
    TEST END!
    ```

3. 编译例程

在编译例程之前，请看4.2 步骤二：修改setting.mk文件配置好settting.mk文件，修改好settting.mk文件后，如果你想编译coremark例程用于上板，那么在终端执行`make software PROGRAM=coremark`，其他例程也可以用这种方式编译。

4. wh_sdk使用教程

下面以创建一个新的工程yexc_OK为例，说明wh_sdk的使用方法。

4.1 步骤一：添加工具链的路径

```
$ export RISCV_PATH=toolchain/path/on/your/pc CROSS_COMPILE=riscv64-unknown-elf
```

如果你的工具链前缀名称是riscv32-unknown-elf，需要这样设置CROSS_COMPILE=riscv32-unknown-elf

4.2 步骤二：修改setting.mk文件

首先，你需要确定你使用的RV Core支持哪些指令集，以及Memory、外设的地址映射，是否有DCache； 例如：你的RV Core支持RV32IMAFC，MEM地址为0x4000_0000，大小为4M，有DCache，外设地址从0x1000_0000开始，那么进入WH32_DDR目录，修改setting.mk文件：

```
$ cd wh_bsp/env/LS_Board/WH32_DDR
$ cat setting.mk
RISCV_ARCH := rv32imafc          #这里写支持的RV指令集
RISCV_ABI  := ilp32f             #这里写ABI
MEM_BASE   := 0x40000000         #这里写MEM地址
MEM_SIZE   := 4M                 #这里写MEM大小
DTIM       := 0                  #如果置1，说明有DTIM，否则，没有
DTIM_BASE  := 0x80000000         #如果有DTIM的话，这里写DTIM的地址
DTIM_SIZE  := 4K                 #如果有DTIM的话，这里写DTIM的大小
CLK_FRQ    := 48000000           #这里写时钟频率
PLIC_BASE  := 0x0C000000         #这里写PLIC的基址
CLINT_BASE := 0x02000000         #这里写CLINT的基址
MMIO_BASE  := 0x10000000         #这里写MMIO外设的基址
INT_NUM    := 16                 #这里写PLIC的中断数量
CORE_NUM   := 1                  #这里写CPU核的数量
ECII_EN    := 0                  #ROCC使能位
```

如果你已经有了对应的setting.mk文件，那么可以直接将该setting.mk文件复制到WH32_DDR目录下

4.3 步骤三：创建一个新的工程

例如，你想要创建一个名为 yexc_OK 的工程，那么需要修改 make_env_var 配置文件:

```
$ cd wh_sdk
$ cat make_env_var
BOARD ?= LS_Board
RELEASE ?= WH32_DDR
PROGRAM ?= yexc_OK               #这里写你要创建、编译的工程目录名称
HOST_IP := localhost
```

接着，输入命令:

```
$ make create_project
```

然后，software目录下就会出现一个 yexc_OK 的目录，并且会有一个自动生成的C源文件和一个Makefile脚本。

4.4 步骤四：为新的工程增加新的源文件（可选）

例如，你想为上面新建的工程 yexc_OK 增加源文件 yexc_OK1.c、yexc_OK2.c，以及头文件 yexc_OK1.h、yexc_OK2.h

```
    $ cd yexc_OK
    $ cat Makefile
    TARGET      :=    yexc_OK                       # 这里写最后生成的ELF文件名称
    ASM_SRCS    :=                                  # 如果有汇编文件的话，把汇编文件
名称写在这里，注意要以 .S 后缀结尾
    C_SRCS      :=    yexc_OK1.c yexc_OK2.c         # 如果C语言源文件的话，把文件名
称写在这里
    HEADERS     :=    yexc_OK1.h yexc_OK2.h         # 如果C语言头文件的话，把文件名
称写在这里
    CFLAGS      :=   -O3                            # 编译器编译选项，可填可不填
    LDFLAGS     :=                                  # 链接器选项，一般不用填
```

4.5 步骤五：编译

1. 完成了源文件的编写之后，需要进行编译：

```
    $ cd wh_sdk
    $ make software
```

注意：请在make_env_var文件中将PROGRAM变量设置为：PROGRAM ?= yexc_OK

4.6 步骤六：下载

1. 配置OpenOCD路径

```
    $ export RISCV_OPENOCD_PATH=openocd_path
```

2. 配置openocd.cfg文件:

   WH SDK提供三种下载方式，分别是jlink、Olimex、FT2232，默认是FT2232，所以如果需要使用Olimex下载，需要执行以下命令

```
    $ export OPENOCDCFG_NAME=Olimex.cfg
```

LS板使用的FT2232下载，所以这一步可以跳过

3. 打开OpenOCD

```
    $ cd wh_sdk
    $ make run_openocd
```

终端打印以下信息则说明OpenOCD已经连接到开发板

```
Open On-Chip Debugger 0.10.0+dev-00614-g998fed1fe (2019-06-04-17:42)
Licensed under GNU GPL v2
For bug reports, read
    http://openocd.org/doc/doxygen/bugs.html
adapter speed: 1000 kHz
Info : auto-selecting first available session transport "jtag". To override use
'transport select <transport>'.
Info : clock speed 1000 kHz
Info : JTAG tap: riscv.cpu tap/device found: 0x00000a73 (mfg: 0x539 (<unknown>),
part: 0x0000, ver: 0x0)
Info : datacount=1 progbufsize=16
```

```
Info : Disabling abstract command reads from CSRs.
Info : Examined RISC-V core; found 1 harts
Info :  hart 0: XLEN=32, misa=0x40141125
Info : Listening on port 3333 for gdb connections
Info : Listening on port 6666 for tcl connections
Info : Listening on port 4444 for telnet connections
```

如果终端打印以下信息，则说明OpenOCD没有权限对USB设备进行操作

```
Open On-Chip Debugger 0.10.0+dev-00614-g998fed1fe (2019-06-10-09:25)
Licensed under GNU GPL v2
For bug reports, read
    http://openocd.org/doc/doxygen/bugs.html
adapter speed: 1000 kHz
Info : auto-selecting first available session transport "jtag". To override use
'transport select <transport>'.
Error: libusb_open() failed with LIBUSB_ERROR_ACCESS
Error: no device found
Error: unable to open ftdi device with vid 0403, pid 6010, description 'Dual
RS232-HS', serial '*' at bus location '*'

Makefile:99: recipe for target 'run_openocd' failed
make: *** [run_openocd] Error
```

为了让OpenOCD具有操作USB设备的权限，可以这样执行该命令：

```
sudo make run_openocd RISCV_OPENOCD_PATH=openocd_path
 OPENOCDCFG_NAME=openocd.cfg
```

4. 打开另外一个终端执行命令（开启了新的终端之后，需要重新执行**步骤4.1**）：

```
$ make upload
```

终端打印以下信息则说明程序已经下载到开发板

```
JTAG tap: riscv.cpu tap/device found: 0x00000a73 (mfg: 0x539 (<unknown>), part:
0x0000, ver: 0x0)
invalid subcommand "protect 0 64 last off"
in procedure 'flash'
Loading section .init, size 0x82 lma 0x40000000
Loading section .text, size 0x14f0 lma 0x40000084
Loading section .rodata, size 0xe8 lma 0x40001574
Loading section .eh_frame, size 0x2c lma 0x4000165c
Loading section .init_array, size 0x4 lma 0x40001688
Loading section .data, size 0x434 lma 0x40001690
Start address 0x40000000, load size 6846
Transfer rate: 43 KB/sec, 1141 bytes/write.
```

4.7 调试程序

使用RISC-V GDB可以对程序进行调试，当OpenOCD已经和板子连上之后，就可以在终端输入：

```
$ make run_gdb
```

执行成功的效果如下：

```
yexc@server124:~/wh_sdk_releaseV001$ make run_gdb
/home/common/riscv-tools/install/riscv-gnu-toolchains/bin/riscv64-unknown-elf-
gdb software/rtc_test/rtc_test.elf -ex "target extended-remote
192.168.168.144:3333" -ex "load"
GNU gdb (GDB) 8.2.50.20181127-git
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "--host=x86_64-pc-linux-gnu --target=riscv64-unknown-
elf".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from software/rtc_test/rtc_test.elf...
Remote debugging using 192.168.168.144:3333
_start () at
/home/yexc/wh_sdk_releaseV001/wh_bsp/env/LS_Board/WH32_DDR/start.S:16
16      la gp, __global_pointer$
Loading section .init, size 0x4a lma 0x40000000
Loading section .text, size 0x9a2 lma 0x4000004c
Loading section .rodata, size 0x40 lma 0x400009f0
Loading section .eh_frame, size 0x2c lma 0x40000a30
Loading section .data, size 0x434 lma 0x40000a60
Start address 0x40000000, load size 3724
Transfer rate: 42 KB/sec, 744 bytes/write.
(gdb)
```

4.7.1 GDB常用命令

本节仅仅介绍一些常用的GDB命令，没有介绍到的GDB命令可以通过输入`help all`获取。

命令	例子	作用
next	next	单步执行，缩写命令为n
print	print var	打印变量var的值，缩写命令为p var
break	break 32	缩写命令为b 32，在程序的第12行设置一个断点，也可以对函数名设置断点
bt	bt	查看堆栈信息
finish	finish	退出当前函数
c	c	继续执行，直到遇到断点，是continue命令的缩写
quit	q	退出GDB，q是quit命令的缩写