

wujian100 sdk解读

简介

wujian100 sdk 为wujian100平台的软件开发包，包含了CPU核、芯片驱动程序、操作系统内核等。通过该SDK可以快速对wujian100-open进行测试与评估，同时用可以参考SDK中集成的各种常用组件以及示例程序进行应用开发，快速形成产品方案。

目录结构

```
sdk
├── board
├── csi_core
├── csi_driver
├── csi_kernel
├── libs
├── projects
├── utilities
└── VERSION
```

sdk目录如上所示，board目录内为与硬件相关的板级配置，包括系统启动外设初始化（例如：串口打印配置）、编译链接配置、example工程中外设的引脚配置等。csi_core目录存放cpu内核相关配置，通常不需要修改。csi_driver 目录存放外设驱动文件（例如：usart、spi等）。csi_kernel存放rtos操作系统文件，wujian100 sdk包内对接了alios的rhino。libs内为库文件，包含c库的实现以及一些常用的数据结构。projects内存放示例工程以及测试工程，utilities存放一些仿真、编译相关的配置文件，VERSION为软件版本记录。

编译工具

wujian100可以选用两种编译工具：

- 1) CDK：界面化的集成开发环境，运行于windows系统。
- 2) riscv64-elf-tools：riscv 工具链，命令行模式使用，运行于linux。

下面简要介绍安装riscv64-elf-tools：

- 1) 下载并解压riscv64-elf-tools工具包（riscv64-elf-x86_64-20190731.tar.gz）
- 2) 设置环境变量：sudo vim /etc/profile

在文件配置文件中添加工具链路径：export PATH=\$PATH:/home/lyt/tools/riscv64-elf-tools/bin

- 3) 刷新环境变量：source etc/profile
- 4) 检查工具链版本：riscv64-unknown-elf-gcc -v，若安装成功则显示以下内容：

```

Using built-in specs.
COLLECT_GCC=riscv64-unknown-elf-gcc
COLLECT_LTO_WRAPPER=/home/lyt/tools/riscv64-elf-
tools/bin/../libexec/gcc/riscv64-unknown-elf/8.1.0/lto-wrapper
Target: riscv64-unknown-elf
Configured with:
/ldhome/software/toolsbuild/slave/workspace/riscv64_build_elf_x86_64/build/./so
urce/riscv/riscv-gcc/configure --target=riscv64-unknown-elf --with-
mpc=/ldhome/software/toolsbuild/slave/workspace/riscv64_build_elf_x86_64/lib-
for-gcc-x86_64-linux/ --with-
mpfr=/ldhome/software/toolsbuild/slave/workspace/riscv64_build_elf_x86_64/lib-
for-gcc-x86_64-linux/ --with-
gmp=/ldhome/software/toolsbuild/slave/workspace/riscv64_build_elf_x86_64/lib-
for-gcc-x86_64-linux/ --
prefix=/ldhome/software/toolsbuild/slave/workspace/riscv64_build_elf_x86_64/inst
all --disable-shared --disable-threads --enable-languages=c,c++ --with-system-
zlib --enable-tls --with-newlib --with-
sysroot=/ldhome/software/toolsbuild/slave/workspace/riscv64_build_elf_x86_64/ins
tall/riscv64-unknown-elf --with-native-system-header-dir=/include --disable-
libmudflap --disable-libssp --disable-libquadmath --disable-libgomp --disable-
nls --src=../../source/riscv/riscv-gcc --enable-checking=yes --with-
pkgversion='C-SKY RISCv Tools V1.2.2 B20190731' --enable-multilib --with-
abi=lp64 --with-arch=rv64gcxcki 'CFLAGS_FOR_TARGET=-Os -mcmodel=medany'
'CXXFLAGS_FOR_TARGET=-Os -mcmodel=medany'
Thread model: single
gcc version 8.1.0 (C-SKY RISCv Tools V1.2.2 B20190731)

```

目录详解

board

```

board
├─ wujian100_open_evb
│   ├── board_init.c
│   ├── gcc_csky.ld
│   └─ include
│       ├── pin.h
│       ├── test_driver_config.h
│       └─ test_kernel_config.h

```

board目录存放wujian100的硬件配置，board_init.c用于配置系统启动的资源初始配置。board_init在main函数之前被调用，5行和6行初始化了计时timer，用于记录系统运行时间以及精准延时。8~10行初始化了串口，用于打印输出。

```

1 void board_init(void)
2 {
3     int32_t ret = 0;
4     /* init the console*/
5     clock_timer_init();
6     clock_timer_start();
7
8     console_handle = csi_usart_initialize(CONSOLE_IDX, NULL);
9     /* config the UART */
10    ret = csi_usart_config(console_handle, 115200, USART_MODE_ASYNCHRONOUS,
        USART_PARITY_NONE, USART_STOP_BITS_1, USAR    T_DATA_BITS_8);

```

```

11
12     if (ret < 0) {
13         return;
14     }
15 }

```

gcc_csky.ld 为编译链接配置，配置了编译生成的固件各个字段的存放位置。12~16行定义了I-SRAM和SRAM的空间，18行定义了heap size，22~25行定义了固件各个字段存放的位置。

```

1  /*
2   * Copyright (C) 2017-2019 Alibaba Group Holding Limited
3   */
4
5
6
7  /* *****
8   * @file      gcc_wujian.ld
9   * @brief     wujian linker file
10  * @version   v1.0
11  * @date      02. June 2017
12  */
13  /* *****
14  MEMORY
15  {
16      I-SRAM : ORIGIN = 0x0, LENGTH = 0x00010000 /* I-SRAM 64KB */
17      SRAM : ORIGIN = 0x20000000, LENGTH = 0x20030000 /* on-chip SRAM
18  192KB */
19  }
20
21  __min_heap_size = 0x200;
22  PROVIDE (__ram_end = 0x20030000);
23  PROVIDE (__heap_end = __ram_end);
24
25  REGION_ALIAS("REGION_TEXT", I-SRAM);
26  REGION_ALIAS("REGION_RODATA", I-SRAM);
27  REGION_ALIAS("REGION_DATA", SRAM);
28  REGION_ALIAS("REGION_BSS", SRAM);

```

include目录存放了板级的pin配置以及驱动测试配置和操作系统的测试配置文件，pin.h用于定义外设端口，如下定义了example usart使用的端口。

```

32 #define EXAMPLE_USART_IDX      0
33 #define EXAMPLE_PIN_USART_TX   PAD_UART0_SIN
34 #define EXAMPLE_PIN_USART_RX   PAD_UART0_SOUT
35 #define EXAMPLE_PIN_USART_TX_FUNC  0
36 #define EXAMPLE_PIN_USART_RX_FUNC  0

```

test_driver_config.h定义了wujian100的外设驱动测试配置，例如#define TEST_TIMER 1为启动Timer驱动测试，设置为0关闭该驱动测试。test_kernel_config.h定义了wujian100的rtos测试配置

```

21 #define TEST_TIMER 1
22 #define TEST_USART 1
23 #define TEST_GPIO 1
24 #define TEST_IIC 1
25 #define TEST_SPI 1
26 #define TEST_PWM 1
27 #define TEST_DMAM 1
28 #define TEST_RTC 1
29 #define TEST_WDT 1

```

csi_core

```

csi_core
├── include
│   ├── core_rv32.h
│   ├── csi_core.h
│   └── csi_rv32_gcc.h

```

csi_core目录存放cpu内核的寄存器级操作接口，例如cpu中断的开启和关闭、cpu中断的保存等。core_rv32.h内的csi_irq_save用于在挂起总中断时存储当前中断状态，csi_irq_restore用于再次开启中断响应后，恢复响应保存的中断。

csi_driver

```

csi_driver
├── include
└── wujian100_open

```

include目录内为外设驱动函数头文件。wujian100_open内为驱动源码、中断处理源码等。下面主要介绍wujian100_open内的驱动源码结构。

isr.c

isr.c存放中断入口函数，当外设等触发cpu中断后，会调用isr内的中断函数。例如timer中断响应函数：

```

64 ATTRIBUTE_ISR void TIM0_IRQHandler(void)
65 {
66     CSI_INTRPT_ENTER();
67     wj_oip_timer_irqhandler(0);
68     CSI_INTRPT_EXIT();
69 }

```

TIM0_IRQHandler为注册到中断向量列表内的中断响应函数，wj_oip_timer_irqhandler为驱动内实现的中断函数。当timer0中断触发时，cpu执行TIM0_IRQHandler，从而调用驱动内的中断执行函数。

devices.c

devices.c 管理芯片外设的基地址获取等。例如timer驱动：

```

57 const sg_timer_config[CONFIG_TIMER_NUM] = {
58     {WJ_TIMER0_BASE, TIM0_IRQn, TIM0_IRQHandler},
59     {WJ_TIMER1_BASE, TIM1_IRQn, TIM1_IRQHandler},
60     {WJ_TIMER2_BASE, TIM2_IRQn, TIM2_IRQHandler},

```

```

61     {WJ_TIMER3_BASE, TIM3_IRQn, TIM3_IRQHandler},
62     {WJ_TIMER4_BASE, TIM4_IRQn, TIM4_IRQHandler},
63     {WJ_TIMER5_BASE, TIM5_IRQn, TIM5_IRQHandler},
64     {WJ_TIMER6_BASE, TIM6_IRQn, TIM6_IRQHandler},
65     {WJ_TIMER7_BASE, TIM7_IRQn, TIM7_IRQHandler},
66     {WJ_TIMER8_BASE, TIM8_IRQn, TIM8_IRQHandler},
67     {WJ_TIMER9_BASE, TIM9_IRQn, TIM9_IRQHandler},
68     {WJ_TIMER10_BASE, TIM10_IRQn, TIM10_IRQHandler},
69     {WJ_TIMER11_BASE, TIM11_IRQn, TIM11_IRQHandler},
70     {WJ_TIMER12_BASE, TIM12_IRQn, TIM12_IRQHandler},
71     {WJ_TIMER13_BASE, TIM13_IRQn, TIM13_IRQHandler},
72     {WJ_TIMER14_BASE, TIM14_IRQn, TIM14_IRQHandler},
73     {WJ_TIMER15_BASE, TIM15_IRQn, TIM15_IRQHandler},
74 };
75
76 int32_t target_get_timer_count(void)
77 {
78     return CONFIG_TIMER_NUM;
79 }
80
81 int32_t target_get_timer(int32_t idx, uint32_t *base, uint32_t *irq, void
**handler)
82 {
83     if (idx >= target_get_timer_count()) {
84         return -1;
85     }
86
87     if (base != NULL) {
88         *base = sg_timer_config[idx].base;
89     }
90
91     if (irq != NULL) {
92         *irq = sg_timer_config[idx].irq;
93     }
94
95     if (handler != NULL) {
96         *handler = sg_timer_config[idx].handler;
97     }
98
99     return idx;
100 }

```

驱动函数内通过调用target_get_timer，获取timer外设的寄存器基地址，中断号，和中断入口函数，然后将中断入口函数注册到中断向量表，打开对应中断，实现中断响应。sg_timer_config数组内存放各个timer外设的基地址、中断号和中断入口函数。

pinmux.c

pinmux.c用于管理pin功能复用，由于wujian100的pin没有复用功能，因此内部函数实现为空。

system.c

system.c用于平台启动的初始化，SystemInit函数会在上文提到的board_init函数之前被调用，该函数会处理外设时钟的使能、cpu频率配置等。

startup.s

startup.s文件用于系统启动，内部定义了中断向量列表、执行了system_init，board_init，将数据搬运到对应的内存地址、初始化清空bss数据段等。

sys_freq.c

系统时钟频率获取接口，例如：int32_t drv_get_cpu_freq(int32_t idx)用于获取cpu频率。通过该文件的频率获取接口可以读取各个外设的时钟源频率等。

include/soc.h

include目录下为外设驱动实现层对应的头文件，soc.h内定义了芯片寄存器的基地址、外设中断号、系统时钟频率等等。

include/pin_name.h

定义wujian100 gpio引脚的名称，通过该定义索引控制gpio。

projects

```
projects
├── benchmark
├── examples
└── tests
```

projects内存放了cpu性能测试工程、外设和rtos的demo工程、外设测试工程。