SKYEYE

天目编程手册

版本　　1.3.5

日期　2013-12-30

目录

1 文档背景和介绍

1.1SkyEye 仿真平台介绍

SkyEye 仿真平台是北京迪捷数原公司开发的仿真平台产品，分为三个组件包：SkyEye 仿真平台，SkyEye 设备模型库以及 SkyEye 外设模型开发环境。其中 SkyEye 仿真平台，包含了仿真平台的 GUI，命令行接口，以及仿真平台基本的运行环境。SkyEye 设备模型库包含了 SkyEye 支持的各种各样设备模型，其中包含处理器模型，总线模型，外设模型等等，具体支持的设备模型请访问官方网站获得更多信息。SkyEye 外设模型开发环境包，是支持外设建模的一套编译器以及相应的开发环境。

目前 SkyEye 仿真平台已经广泛用于航空，航天，军工等领域，在系统软件开发和调试，软硬件协同开发，自动测试环境搭建上都有不可替代的作用。它不但可以独立运行，也可以和第三方的 IDE 相结合，如 Eclipse。

1.2SkyEye 的架构和编程介绍

SkyEye 仿真平台不但使一个完整的软件平台，它同时也提供了完善的 API 和文档，用户可以对 Skyeye 进行二次开发。SkyEye 的 API 包含仿真平台的控制 API，仿真平台中处理器的寄存器访问 API，内存访问 API，事件调度 API 等等，方便不同的用户对 SkyEye 平台进行定制和集成。关于 SkyEye 的 API 的详细内容，可以参考本文档的第二章节。

SkyEye 本身也包含了很多使用 SkyEye 的 API 实现的模块。如指令流记录模块，性能分析模块，代码覆盖率模块，这些模块也可以成为初次接触 SkyEye 编程的开发者的参考编程示例。在本文档的第四章节，我们给出了一些编程示例来做为 SkyEye 编程的参考。

2 SkyEye 的编程 API 介绍

2.1运行控制 API 接口

SIM_init()

声明：**void SIM_init()**

功能说明：调用 SIM_init 初始化 Simulator，SIM_init 负责做以下初始化工作：

初始化核心库中的所有数据结构

根据设置的动态库加载路径，加载各种模块

参数：无

返回值：无

SIM_start()

功能说明：根据配置文件，设置特定的数据结构，为运行做好准备。

参数：

返回值：

SIM_run()

功能说明：启动目标机器

参数：无

返回值：

SIM_stop(generic_core_t* core)

功能说明：停止指定的处理器核心

参数：指定的处理器核心

返回值：无

SIM_continue(generic_core_t* core)

功能说明：让指定的处理器核心继续运行。

参数：

返回值：

SIM_sched(int us)

功能说明：仿真平台运行指定的时间，然后停止并返回

参数：**us** 是让仿真平台运行的微秒数

返回值：

2.2命令行接口 API

add_command

声明：**exception_t add_command(char* command_str, rl_icpfunc_t* func, const char* help_str);**

功能说明：添加一个新的命令到命令行接口中。

参数：**command_str** 为命令的字符串，**func** 为执行命令时要调用的函数，**help_str** 是命令的帮助文档。

返回值：成功返回 **No_exp**，错误则返回相应异常的类型。

2.3回调函数接口 API

register_callback

声明：**exception_t register_callback(callback_func_t func, callback_kind_t kind);**

功能说明：注册一个新的回调函数到系统中。

参数：**func** 为要执行的回调函数，**kind** 是回调函数的类型。

返回值：成功返回 **No_exp**，错误则返回相应异常的类型。

2.4配置文件接口 API

get_current_config

声明：**skyeye_config_t* get_current_config();**

功能：获得当前的配置文件的数据结构

参数：无

返回值：当前配置文件的数据结构

skyeye_read_config

声明：**exception_t skyeye_read_config (char* conf_filename);**

功能：加载并解析相应的 skyeye 配置文件，保存在内存中。

参数：配置文件的文件名。

返回值：成功返回 **No_exp**，错误则返回相应的异常类型。

register_option

声明：**exception_t register_option(char* option_name, do_option_t do_option_func, char* helper);**

功能：注册新的配置文件选项到系统中。

参数：**option_name** 是配置选项的名称，**do_option_func** 是用来解析配置选项的函数，**helper** 是该配置选项的帮助文本。

返回值：成功则返回 **No_exp**，错误则返回相应的异常类型

2.5uart 相关的 API

skyeye_uart_write

声明：**int skyeye_uart_write(int devIndex, void *buf, size_t count,**
int *wroteBytes[MAX_UART_DEVICE_NUM]);

功能说明：

提供给虚拟串口硬件调用，可以把数据写入底层的物理接口模块中。

参数：

返回值：

skyeye_uart_read

声明：

int skyeye_uart_read(int devIndex, void *buf, size_t count, struct
timeval *timeout, int *retDevIndex);

功能说明：提供给虚拟串口硬件调用，可以把数据写入底层的物理链接层模块中。

2.6模块相关的 API

skyeye_load_all_module

声明：**void skyeye_load_all_module(const char* lib_dir, char***
lib_suffix);

功能描述：

参数：

返回值：

skyeye_load_module

声明：**exception_t SKY_load_module(const char* module_filename);**

功能描述：

参数：

返回值：

2.7内存访问相关的 API

bus_read

声明：**int bus_read(short size, int addr, uint32_t * value);**

功能描述：从总线上读一个数据到 **value** 指向的存储单元中。

参数

返回值：

bus_write

声明：**int bus_write(short size, int addr, uint32_t value);**

功能描述：从总线上写一个数据到某一个地址单元中。

参数：**size** 用来指定写数据的长度，**addr** 为写入的地址的值，**value** 为写入的数据。

返回值：

addr_mapping

声明：**exception_t addr_mapping(mem_bank_t* bank);**

功能描述：**addr_mapping** 接口用来申请一段地址空间,并来控制这段地址空间的访问。

参数 ：**bank**,是客户需要分配和填充的一个数据接口,定义如下：

```
typedef struct mem_bank
{
unsigned int addr, len;
char (*bank_write)(short size, int offset, unsigned int value);
char (*bank_read)(shortsize, int offset, unsigned int *result);
char filename[MAX_STR];
/* the name of object mapping to the bank */
char* objname;
unsigned type;
} mem_bank_t;
```

mem_bank_t 的域 **addr** 是这段地址空间的起始地址,**len** 为这段地址空间的长度。

bank_write 和 **bank_read** 分别为这段地址的读写函数。当目标程序对这段地址空间进行访问的时候,SkyEye 会调用这段地址空间对应的 **bank_write** 和 **bank_read** 进行访问。

Filename 为加载的文件,在初始化这段地址空间的时候,SkyEye 可以把一个数据文件加载到这段地址空间去。

objname 是一个字符串,用来标志一个对象的名称。

type 是这段地址空间的类型,有只读内存,可读写内存和 IO 这三种类型。

返回值：可能发生的异常类型。

2.8机器管理的 API

register_mach

声明：**void register_mach(char* mach_name, mach_init_t mach_init);**

功能描述：注册一个模拟的机器或者单板到系统中。

参数：

返回值：无

get_mach

声明：**machine_config_t * get_mach(const char* mach_name);**

功能描述：获得一个机器或者单板的数据结构

参数：机器的名称

返回值：模拟的机器的数据结构

send_signal

声明：**exception_t send_signal(interrupt_signal_t* signal);**

功能描述：向某一个处理器发出中断信号，一般外设会调用此接口。

参数：信号的类型和电平。**interrupt_signal_t** 的定义如下：

```
typedef union interrupt_signal{
arm_signal_t arm_signal;
mips_signal_t mips_signal;
powerpc_signal_t powerpc_signal;
}interrupt_signal_t;
```

它是一个联合，为不同的体系结构定义了不同的信号的结构体。

返回值：可能发生的异常类型。

2.9事件调度器的相关的 API

create_thread_scheduler

声明：**int create_thread_scheduler(unsigned int ms, sched_mode_t mode, sched_func_t func, void *arg, int *id);**

功能说明：创建一个线程控制调度的事件，并放入调度队列中。输入该事件发生的相对事件，是否是周期发生模式，响应函数，函数参数，一个整形指针得到调度器分配的 ID。

参数： **ms** 为事件发生的间隔时间；**mode** 表示是否是周期性事件 **(Oneshot_sched/Periodic_sched)**；

func 是事件处理函数的指针；**arg** 是处理函数的参数指针；**id** 表示获取标识的指针。

返回值：成功会返回 **No_exp**，否则会返回错误信息。

mod_thread_scheduler

声明：**int mod_thread_scheduler(int id, unsigned int ms, sched_mode_t mode);**

功能说明：修改调度队列中指定标识的事件。

参数：**id** 为事件的标识；**ms** 是事件的发生的间隔时间；**mode** 表示是否周期性事件 **(Oneshot_sched/Periodic_sched)**。

返回值：成功会返回 **No_exp**，否则会返回错误信息。

del_thread_scheduler

声明：**int del_thread_scheduler(int id);**

功能说明：删除调度队列中指定标识的事件。

参数：**id** 是事件的标识。

返回值：成功会返回 **No_exp**，否则返回错误信息。

fini_thread_scheduler

声明：**int fini_thread_scheduler(void);**

功能说明：销毁线程调度队列。

参数：无

返回值：成功会返回 **No_exp ，**否则返回错误信息。

list_thread_scheduler

声明：**void list_thread_scheduler(void);**

功能说明：打印线程调度队列中的所有事件及其属性。

参数：无

返回值：无

create_timer_scheduler

声明：**int create_timer_scheduler(unsigned int ms, sched_mode_t mode, sched_func_t func, void *arg, int *id);**

功能说明：创建一个定时器调度事件，并放入调度队列中。输入该事件发生的间隔时间、是否是周期发生模式、响应函数、函数参数、一个整形的指针得到调度器分配的 ID。

参数：**ms** 为事件发生的间隔时间；**mode** 是否是周期性事件 **(Oneshot_sched/Periodic_sched)；func** 是事件处理函数的指针；**arg** 是处理函数的参数指针；**id** 表示获取标识的指针。

返回值：成功会返回 **No_exp，**否则返回错误信息。

mod_timer_scheduler

声明：**int** mod_timer_scheduler**(int id, unsigned int ms, sched_mode_t mode);**

功能说明: 修改调度队列中指定标识的事件。

参数：**Id** 为事件的标识；**ms** 是事件的发生的间隔时间；**mode** 表示是否周期性事件 **(Oneshot_sched/Periodic_sched)。**

返回值：成功会返回 **No_exp ，**否则返回错误信息。

del_timer_scheduler

声明：**int del_timer_scheduler(int id);**

功能说明：删除指定标识的定时器事件。

参数：**id，**事件标识。

返回值：成功会返回 **No_exp，**否则返回错误信息。

fini_timer_scheduler

声明：**int fini_timer_scheduler(void);**

功能说明:销毁定时器调度队列。

参数：无。

返回值：成功会返回 **No_exp**，否则返回错误信息。

list_timer_scheduler

声明：**void list_timer_scheduler(void);**

功能说明:打印定时器调度队列中的所有事件及其属性。

参数：无

返回值：无

3 SkyEye 提供的数据类型

3.1异常相关数据类型

exception_t

声明：

```
typedef enum{
/* No exception */
No_exp = 0,
/* Memory allocation exception */
Malloc_exp,
/* File open exception */
File_open_exp,
/* DLL open exception */
Dll_open_exp,
/* Invalid argument exception */
Invarg_exp,
/* Invalid module exception */
Invmod_exp,
/* wrong format exception for config file parsing */
Conf_format_exp,
/* some reference excess the predefiend range. Such as the index out of array range */
Excess_range_exp,
/* Unknown exception */
Unknown_exp
}exception_t;
```

描述：

No_exp，当函数按照正常的流程结束，没有发生任何异常。

Malloc_exp，当函数在分配内存出错的时候，返回的异常类型。

File_open_exp，当文件打开错误返回的异常类型。

Dll_open_exp，当 DLL 文件打开错误返回的异常类型。

Invarg_exp，当传入参数不合法的时候返回的异常类型。

Invmod_exp，当加载的模块不符合规范的时候返回的异常类型

Conf_format_exp，当配置文件选项有错误的时候，返回的异常类型

Excess_range_exp，当给定的索引超出预定义的范围的时候返回的异常类型

Unknown_exp，未知的异常类型。

3.2回调函数相关的数据类型

callback_kind_t

定义：

```
  typedef enum{
Step_callback = 0,
Mem_read_callback,
Mem_write_callback,
Bus_read_callback, /* called when memory write */
Bus_write_callback, /* called when memory read */
Exception_callback, /* called when some exceptions are triggered. */
Bootmach_callback, /* called when hard reset of machine */
Max_callback
  }callback_kind_t;
```

描述：回调函数的类型，分别描述如下

Step_callback，是用来在虚拟机单步执行的时候可以被调用的函数。目前在断点模块，单步执行模块等中应

用。

Mem_read_callback，是在虚拟机读内存的时候可以被调用的函数。

Mem_write_callback，是在虚拟机写内存的时候可以被调用的函数。

Bus_read_callback，是在虚拟机读总线数据的时候被调用的函数，

Bus_write_callback，是在虚拟机写数据到总线的时候被调用的函数，

Bootmach_callback，是在机器重新起动的时候所调用的启动函数

Exception_callback，是在虚拟机发生异常的时候可以被调用的函数。

Max_callback，在这里只是用来表示异常类型的数目。

callback_func_t

定义：**typedef void(*callback_func_t)(generic_arch_t* arch_instance);**

描述：回调函数的定义。

3.3配置文件相关的数据类型

skyeye_option_t

声明

typedef struct skyeye_option_s

```
{
char *option_name;
int (*do_option) (struct skyeye_option_s * this_opion,
int num_params, const char *params[]);
char* helper;
struct skyeye_option_t *next;
} skyeye_option_t;
```

描述：

用来实现每个配置选项的数据结构。其中 **do_option** 是用来解析模块的函数。**helper** 是此配置选项的描述。

do_option_t

声明

```
typedef int(*do_option_t)(struct skyeye_option_t *option, int
num_params, const char *params[]) ;
```

描述：定义了解析配置文件选项的函数。

3.4模块相关的数据类型

skyeye_module_t

声明：

```
typedef struct skyeye_module_s{
/*
* the name for module, should defined in module as an varaible.
*/
char* module_name;
/*
* the library name that contains module
*/
char* filename;
/*
* the handler for module operation.
*/
void* handler;
/*
* next node of module linklist.
*/
struct skyeye_module_s *next;
```

```
}skyeye_module_t;
```

描述：用来记录每个模块的数据结构。

4 SkyEye 的编程示例

4.1记录运行时的 PC 的模块编程示例

介绍了 skyeye 模块编程的一个示例程序 log-pc 模块。本模块演示了如何编写独立的 skyeye 模块，并加载编译和运行的过程。模块功能主要是用来记录 skyeye 执行执的所有 PC 指令。

4.2编写相关代码

模块由以下三个文件组成。

log.c：实现了记录 PC 的功能，

```
#include <stdlib.h>
#include <stdio.h>
#include "skyeye_arch.h"
#include "skyeye_callback.h"
/* flag to enable log function. */
static int enable_log_flag;
/* the log file for record. */
static const char* log_filename = "./pc.log";
/* fd of log_filename */
static FILE* log_fd;
/* callback function for step exeuction. Will record pc here. */
static void log_pc_callback(generic_arch_t* arch_instance){
if(enable_log_flag){
fprintf(log_fd, "pc=0x%x\n", arch_instance->get_pc());
}
}
/* enable log functionality */
static void com_log_pc(char* arg){
enable_log_flag = 1;
}
/* some initialization for log functionality */
int log_init(){
exception_t exp;
/* open file for record pc */
log_fd = fopen(log_filename, "w");
```

```
    if(log_fd == NULL){
    fprintf(stderr, "Can not open the file %s for log-pc module.\n",
log_filename);
    return;
    }
    /* register callback function */
    register_callback(log_pc_callback, Step_callback);
    /* add correspinding command */
    add_command("log-pc", com_log_pc, "record the every pc to log
file.\n");
    }

    /* destruction function for log functionality */
    int log_fini(){
    if(log_fd != NULL){
    fclose(log_fd);
    }
    }
```

log_module.c：用来实现模块的加载和卸载函数

```
    #include <stdio.h>
    #include <errno.h>
    #include "skyeye_types.h"
    #include "skyeye_arch.h"
    #include "skyeye_module.h"
    /* module name */
    const char* skyeye_module = "log-pc";
    /* module initialization and will be executed automatically when loading. */
    void module_init(){
    log_init();
    }
    /* module destruction and will be executed automatically when unloading */
    void module_fini(){
    log_fini();
    }
```

Makefile ： 负责编译整个模块，并安装至 skyeye 的模块目录下。

```
SKYEYE_PREFIX=/opt/skyeye/
CC=gcc
liblog.so:
$(CC) -I${SKYEYE_PREFIX}/include/include\
       -L$(SKYEYE_PREFIX)/lib/skyeye/ -lcommon
-shared -oliblog.so log.c log_module.c
clean:
rm liblog.so *.o -r -f
install:
cp liblog.so ${SKYEYE_PREFIX}/lib/skyeye/
```

4.3代码编译与安装

编译模块

创建一个新的目录，并把上面的 log.c log_module.c Makefile 都放入此目录。在编译模块之前，先确认 skyeye 已经安装到系统中，可以查看/opt/skyeye/include/include/中是否有相关头文件存在，如 skyeye_types.h, skyeye_arch.h 等等。

如果 skyeye 安装正常，我们可以在你的模块目录下运行"make"来编译模块生成 log.so 文件。

模块安装运行"make install"命令，可以把你编译的模块安装到 skyeye 的相应模块目录，skyeye 会在启动的时候加载模块。

4.4运行测试

进入到/opt/skyeye/testsuite/arm_hello 目录下，测试我们的 log-pc 模块的运行。运行 skyeye 如下：

```
ksh@linux-gvai:/opt/skyeye/testsuite/arm_hello> ../../bin/skyeye
SkyEye is an Open Source project under GPL. All rights of different parts or modules are reserved by their author. Any modification or redistributions of SkyEye should note remove or modify the
annoucement of SkyEye copyright.
Get more information about it, please visit the homepage http://www.skyeye.org.
Type "help" to get command list.
(skyeye)
```

然后再运行 list-modules 可以发现，我们的 log-pc 模块已经被加载，输出如下：

```
(skyeye)list-modules
Module Name File Name
nandflash /opt/skyeye/lib/skyeye/libnandflash.so
```

```
arm /opt/skyeye/lib/skyeye/libarm.so
log-pc /opt/skyeye/lib/skyeye/log.so
bfin /opt/skyeye/lib/skyeye/libbfin.so
log-pc /opt/skyeye/lib/skyeye/liblog.so
uart /opt/skyeye/lib/skyeye/libuart.so
mips /opt/skyeye/lib/skyeye/libmips.so
net /opt/skyeye/lib/skyeye/libnet.so
code_cov /opt/skyeye/lib/skyeye/libcodecov.so
sparc /opt/skyeye/lib/skyeye/libsparc.so
ppc /opt/skyeye/lib/skyeye/libppc.so
touchscreen /opt/skyeye/lib/skyeye/libts.so
coldfire /opt/skyeye/lib/skyeye/libcoldfire.so
flash /opt/skyeye/lib/skyeye/libflash.so
lcd /opt/skyeye/lib/skyeye/liblcd.so
gdbserver /opt/skyeye/lib/skyeye/libgdbserver.so
(skyeye)
```

然后还可以运行我们在前面 log.c 文件中注册的命令 log-pc，来使能日志功能：

```
ksh@linux-gvai:/opt/skyeye/testsuite/arm_hello> ../../bin/skyeye -e arm_hello
SkyEye is an Open Source project under GPL. All rights of different parts or modules are reserved by their author. Any modification or redistributions of SkyEye should note remove or modify the
annoucement of SkyEye copyright.
Get more information about it, please visit the homepage http://www.skyeye.org.
Type "help" to get command list.
(skyeye)start
arch: arm
cpu info: armv3, arm7tdmi, 41007700, fff8ff00, 0
In do_mach_option, mach info: name at91, mach_init addr 0xb72a0f70
uart_mod:3, desc_in:, desc_out:, converter:
In create_uart_console
cpu info: armv3, arm7tdmi
SKYEYE: use arm7100 mmu ops
In SIM_start, Set PC to the address 0x0
(skyeye)log-pc
(skyeye)
```

如果一切顺利的话，你在退出 SkyEye 的时候，发现当前目录中会多了一个 pc.log 文件，里面记录了你刚刚在 SkyEye 上运行的软件的指令流。

4.5性能监控器的模块编程

4.5.1 性能监控器模块的介绍

性能监控器模块主要是用来统计 SkyEye 的本身的性能的模块。它的原理主要是统计 SkyEye 在单位时间内执行的指令的条数。

4.5.2 性能监控器模块的代码实现

性能监控器模块的代码位于 utils/perf_monitor/目录，共有两个文件 pmon.c pmon_module.c。

pmon.c: 性能监控的主要实现。

pmon_module.c: 性能监控的模块注册文件。

4.5.3 性能监控器模块的代码分析

在模块加载的时候，性能监控主要是添加了一个"pmon”的命令。当我们在 SkyEye 命令行上运行 pmon 命令的时候，它调用相应的函数 com_pmon，代码如下：

```
30 /* enable log functionality */
31 static void com_pmon(char* arg){
32 enable_pmon_flag = 1;
33 /* open file for record performance data */
34 pmon_fd = fopen(pmon_filename, "w");
35  if(pmon_fd == NULL){
36 fprintf(stderr, "Can not open the file %s for pmon module.\n ",
          pmon_filename);
37 return;
38 }
39 pthread_t id;
40 generic_arch_t* arch_instance = get_arch_instance("");
41 create_thread(pmon_count, arch_instance, &id);
42
```

在第 34 行打开一个日志文件来写性能统计数据，然后在 41 行创建一个线程，并在线程中运行性能统计分析函数 pmon_count。

性能分析统计功能主要在 pmon_count 中实现，它由一个独立的线程进行执行。代码如下：

```
17 static void pmon_count(generic_arch_t* arch_instance){
```

```
int seconds = 0;
uint32 steps = 0;
uint32 last_steps = 0;
/* Test if skyeye is in running state. */
if(!skyeye_is_running())
return;
while(enable_pmon_flag){
last_steps = arch_instance->get_step();
sleep(1);
seconds++;
steps = arch_instance->get_step();
fprintf(pmon_fd,"In %d seconds, MIPS=%d\n", seconds, (steps -
        last_steps));
}
}
```

上面统计函数的主体是一个 while 循环，它每一秒就获得当前执行的指令数目，然后打印到日志文件中，这样我们可以获得每一秒 SkyEye 执行指令的数目，从而判断模拟器的性能。

4.5.4 性能监控器的编译，加载和使用

性能监控器的使用可以参考<<SkyEye User Manual>>的相关章节。

5 SkyEye 与 SystemC

5.1 SystemC 和 SkyEye 的集成原理

SkyEye 仿真平台不但提供了一个仿真平台，而且也提供了一个具有二次开发能力的仿真平台库。我们提供了丰富的 API 提供了仿真平台的控制，获得仿真环境信息，配置仿真平台的能力。基于 SkyEye 的二次开发能力，我们可以把 SkyEye 集成到客户已有的 SystemC 的仿真平台中。其中 SystemC 的 sc_main 作为主控程序，调用 SkyEye 的 API 对 SkyEye 仿真平台进行控制。

5.2 运行 SkyEye 的 SystemC 示例

5.2.1 编译 SystemC

下载 systemc-2.3.0 的软件包。假设软件包名称为 systemc-2.3.0.tgz。然后参考 systemc-2.3.0 的 INSTALL 文件，运行如下命令编译 systemc-2.3.0：

./configure ; make ;make install ;

注意需要创建一个 build 目录，进行编译安装。

5.2.2 编译 Skyeye

编译 SkyEye 的时候使能 systemc，如下命令

```
./configure --with-systemc=SYSTEMC_INSTALL_DIR
```

SYSTEMC_INSTALL_DIR 是编译 SystemC 时指定或默认的安装目录。

在 skyeye 编译安装完成之后，会在 bin 目录下生成一个 skyeye_systemc 的二进制文件。

5.2.3 测试 SystemC 的输出

运行 skyeys_systemc 执行 arm_hello，输出如下：

```
ksh@ksh-Z68A-D3H-B3:/opt/testsuite/win_gui/install/testsuite/ar
m_hello$
../../bin/skyeye_systemc -n -e arm_hello


            SystemC 2.3.0-ASI --- Oct 14 2013 16:40:24
        Copyright (c) 1996-2012 by all Contributors,
```

```
       ALL RIGHTS RESERVED


SkyEye 1.3.5
SkyEye is an Open Source project under GPL. All rights of different
parts or modules are reserved by their author. Any modification or
redistributions of SkyEye should not remove or modify the
annoucement
of SkyEye copyright.
Get more information about it, please visit the
homepage http://www.skyeye.org.
Type "help" to get command list.
 In create_uart_console
1 core is initialized.
load section .text: addr = 0x01000000  size = 0x00000084.
load section .glue_7: addr = 0x01000084  size = 0x00000000.
load section .glue_7t: addr = 0x01000084  size = 0x00000000.
load section .data: addr = 0x01002000  size = 0x00001000.
not load section .bss: addr = 0x01003000  size = 0x00000000 .
not load section .debug_abbrev: addr = 0x00000000  size =
0x0000004e .
not load section .debug_info: addr = 0x00000000  size =
0x00000187 .
not load section .debug_line: addr = 0x00000000  size =
0x000000a9 .
not load section .debug_pubnames: addr = 0x00000000  size =
0x0000001c .
not load section .debug_aranges: addr = 0x00000000  size =
0x00000020 .
load elf arm_hello succeed
```

其 SytemC 示例源代码请参考 utils/systemc/skyeye_systemc.cpp 文件。