C-Sky Test Suit L 用户手册

平头哥

版本信息

版本	日期	描述	作者
1.0	2017-03-24	1、第一版	平头哥半导体有限公司

目录

图表目录

1 概述

1.1 CTS 介绍

C-SKY Test Suit（CTS）是平头哥为客户提供的一个强大的测试集合，其中包含的测试程序覆盖了 CPU 集成验证，功能/性能/功耗仿真以及最后的芯片测试。同时 CTS 也具有很强的可移植性，客户简单输入配置信息后就可将 CTS 映射到自己的仿真平台进行仿真。另外，我们也将 CTS 映射到了 Smart 平台，作为示例进行仿真和演示。

具体来说 CTS 有如下几个特点：

- 覆盖面全

CTS 里的测试程序覆盖了集成验证、功能验证、功耗仿真、性能仿真，功能演示、芯片测试等各个环节；

- 可移植性强

用户只需要简单填写 SOC 地址等信息，平头哥所提供的脚本可以将 CTS 快速的重映射到客户的 SOC 仿真平台；

- 虚拟时钟统计性能

CTS 里含有 dhrystone、coremark、中断响应速度等性能测试程序，这些测试程序都需要系统时钟计算性能，正常情况下，客户需要根据自身计时模块的设计方案修改驱动之后才能仿真性能，但为了方便客户，CTS 里设置了虚拟的时钟，由 CPU 内部进行读取，他的频率和 CPU 一致，这样就省去了客户自己修改时钟驱动的过程，在 SOC 设计早期就可以得到性能数据，并且我们对测试性能做了修改，统计性能根据程序执行的周期数得到，与仿真频率无关。

- 可持续开发

CTS 提供了完整的 C、汇编开发环境，RTL 仿真阶段就可以用 C 语言写测试程序对 SOC 进行仿真和验证。客户可以基于该平台构造自己想要的 C、汇编测试集合；

- 易于学习

平头哥会为客户提供已经映射好 CTS 的 SmartL 平台，SmartL 上有完整的仿真环境，客户可以先在该平台上进行熟悉和仿真，也可参考该平台搭建自己的仿真环境。

1.2 CTS 测试程序分类

CTS 测试程序可分为以下九大类：

测试程序 分类	说明	使用阶段
联通性测试程序	测试 CPU 是否被正确集成到了客户的 SOC 平台	集成验证

工艺映射测试程序	软核授权客户测试 memory 和 gated cell 自己是否替换正确	集成验证
功能测试程序	测试 CPU 细分功能顺带演示这些功能的使用方法，包括 MGU、Cache、中断向量表配置、DFT、低功耗等等。	功能仿真
C 语言程序举例	C 语言程序举例，汇编内嵌举例	功能仿真
最大功耗测试程序	构造了理论上 CPU 出现峰值功耗的场景，可用于系统功耗仿真（评估）。	功耗仿真
性能测试程序	包含了 dhrystone、coremark、linpack、memory copy 等性能测试程序，用于系统（CPU）性能仿真/评估	性能仿真
DFT 测试程序	用于仿真 DFT pattern（硬核才有）	功能仿真
覆盖率测试程序	包含代码覆盖率为 95%和 80%两个版本，测试程序带有结果自检功能，可作于芯片测试 pattern	芯片测试
关键路径测试程序	覆盖前端的关键路径，测试程序带有结果自检功能，可作于芯片测试 pattern	芯片测试

图表 1-1 CTS 测试程序分类

2 CTS 测试程序具体描述

2.1 Coverage 测试程序

测试程序名称: ckxx_cover_smoke_80.s

测试程序名称: ckxx_cover_smoke_94.s

Coverage 指的是针对代码行的覆盖率。该测试程序主要有以下几个特点：

1. 包含两个版本，内核部分 line coverage 分别为 95%版本和 80%的版本，由客户根据自己的测试成本进行选择
2. 测试程序带有自检功能，错误后会向固定地址写入特殊值。
3. 测试程序可根据 CPU 配置裁剪

100%覆盖率是难以达到的，原因是一些和总线相关的激励无法由指令构造出来的，比如（ready/grant/error/ebt），had 激励也无法由指令构造，LSU 内部大小端同样无法由指令构造激励覆盖。

CPU	覆盖率	目标覆盖率 80%
CK801	90%，4.4K	80%，4.1K
CK802	90%，5.8K	80%，5.5K
CK803S	94%，8.4K	80%，8.1K

图表 2-1 覆盖率和程序大小

80%的版本是按照时间优先原则删减出来的，删掉了复杂的循环，代码量虽然减小不多但是仿真时间有效减小，主要是考虑测试机的成本是按照时间算的，用户可根据自己的需求选择合适的版本。

2.2 Critical path 测试程序

测试程序名称: ckxx_critical.s

我们把 Critical path 分成三种三种：

（1） 综合不同配置得到的 critical path

（2） memory out path，如果客户选择速度很慢的 memory，可能成为关键路径

（3） OUTPUT path，如果接口时序比较差，可能成为关键路径

该测试程序带有自检查功能，可用于芯片测试。需要说明的是，这些 critical path 都是前端数据得出，实际最终的关键路径可能跟后端实现以及具体工艺都相关。下面列的 critical path 是前端在特定工艺下综合得出的，所以不保证一定是最终物理实现后关键路径。

2.2.1 CK801

Path group	Path 描述	测试程序简介
Reg to reg	配 mgu：Inst pop entry -> tcipif_xx_addr_reg Inst buf pop 出指令，decode 后得到给 lsu 的 offset，经过若干 mux 之后，base 和 offset 相加得到 lsu 的 addr 给 mgu 比较是否 hit hit 哪个 region，deny 信号传给 bmu，通向 tcip 的 req	配置 mgu，使访问地址在 user 态可读不可写通过 ld store 指令访问 tcip 地址，检查数据正确性,和是否进入过异常
	无 mgu：x_ibuf_pop0_reg -> ph_iu_gated_clk_reg_user_3_x_reg_dout_reg_23 ibuf 出来的指令经 decode 生成立即数，alu 做加法，完成之后结果回写寄存器	构造加法指令加法指令
Output	Ibuf_pop0 -> biu_pad_htrans[1] GPR 读取、LSU base + offset，过 MGU 生成 deny 信号传到 trans	配置 mgu 使访问地址在 user 态可读不可写，访问系统总线检查数据正确性，是否正确进入异常。
Memory	无	

图表 2-2 Ck801 critical path

2.2.2 CK802

Path group	Path 描述	测试程序简介
Reg to reg	x_intc_kid_27_int_enable-> x_nm_iu_wb_wb_data_buffer int_enable 信号存放在 VIC 寄存器 ISER 中，读取 ISER 时，VIC 内部寄存区 20 MUX 1，TCIP 9 MUX 1，BMU 4MUX1，unalign size 14 MUX 1，符号扩展 3 MUX1，GPR write en 6 MUX 1，回写通路 2MUX1.	通过 ld store 指令访问 VIC 中的 ISER
	x_nm_ifu_top_x_ifdp_ifu_iu_ex_src0_reg ->x_ifdp_ifu_iu_ex_imm_reg	构造拆分乘法指令，并验证结果正确性

	拆分乘法时，src0_reg 驱动读出 GPR，PC 和 had、GPR 3MUX1，ALU 操作数准备，32 加法，回写数据选择后 data forward IF 级，做数据选择后到进入目的寄存器	
Output	Ifu_iu_ex_src0_reg ->iahbl_pad_htrans[1] GPR 读取、LSU base + offset，过 MGU 生成 deny 信号传到 trans	配置 mgu，使访问地址在 user 态可读不可写，访问系统总线检查数据正确性，是否正确进入异常。
Memory	Memory OUTPUT	Invalid all cache 配置 4 片 cacheble 地址 使能 cache 写满同一个 index 索引的一个 set 中的 4 个或 2 个 way（看是 4 路还是两路组相连） 连续读同一个命中 cache 的地址 这样就能从每一个 memory 中进行读操作。

图表 2-3 Ck802 critical path

2.2.3 CK803S

Path group	Path 描述	测试程序简介
Reg to reg	idu_fpu_sel_reg[0] -> psr_c_reg Fpu 比较指令传递 c 位给 psr	构造浮点比较指令，比较 c 位跳转，check 跳转是否正确
	du_ex_sub_func_reg -> idu_ex_src1_reg Decode 出来的乘法操作，乘法结果 flop	执行不同操作数乘法指令
Output	Input_buffer_inst_reg-> Iahbl_pad_htrans[1] GPR 读取、LSU base + offset，过 MGU 生成 deny 信号传到 trans	配置 mgu，使访问地址在 user 态可读不可写，访问系统总线检查数据正确性，是否正确进入异常。
Memory	Memory OUTPUT	往 base addr 写 1 个 cache line 大小的数据 Invalid all cache

		配置 crcr 寄存器，配置 region0 base 为 base addr size 为 4k 的 cacheble 地址 Enable cache， write back ， non allocate 读 base_addr 地址，发生 cachemiss，refill 数据进 cache，然后往同一个地址写，再读。Selfcheck 读出来的数据是否是写进 cache 的数据。 对 offset 为一个 word 的地址做 5 的操作 一个 cache line 分 4 个 word 分别放入 4 个 bank 所以这样进行读写后，配置的 6 块 memory（1 块 dirty，1 块 tag，4 块 data）都进行了数据交换

图表 2-4 Ck803S critical path

2.3 Max power 测试程序

测试程序名称: ckxx_max_power.s

801、802、803s 都是单发射处理器，最大功耗体现在不断取指令并且触发功耗最大的运算单元，同时需要保证在各个 stage 中都有指令正在被处理。综上测试程序构造环境需满足以下条件:

（1） 不间断取 32 位指令,保证命中 cache

（2） 执行阶段需要触发功耗最大的运算单元

（3） 其他各级均不应该空闲，不出现 stall

所以在测试程序设计过程中设计的指令的选择根据配置情况依次优先选择 DSP 指令，快速乘法指令，lsu 指令和普通加法指令以满足条件 2，这些指令在使用时尽可能减少 stall 或 flush 的情况以满足条件 3，循环执行该段指令，在配置有 CACHE 的情况下保证后几次循环的指令取指来源为 CACHE，以满足条件 1。通过上述方法构造理论上最大功耗的场景，用于功耗评估功耗最大的情况。

2.4 联通性测试程序

测试程序名称：ckxx_connect.s

测试程序名称：ckxx_biu_interface.s

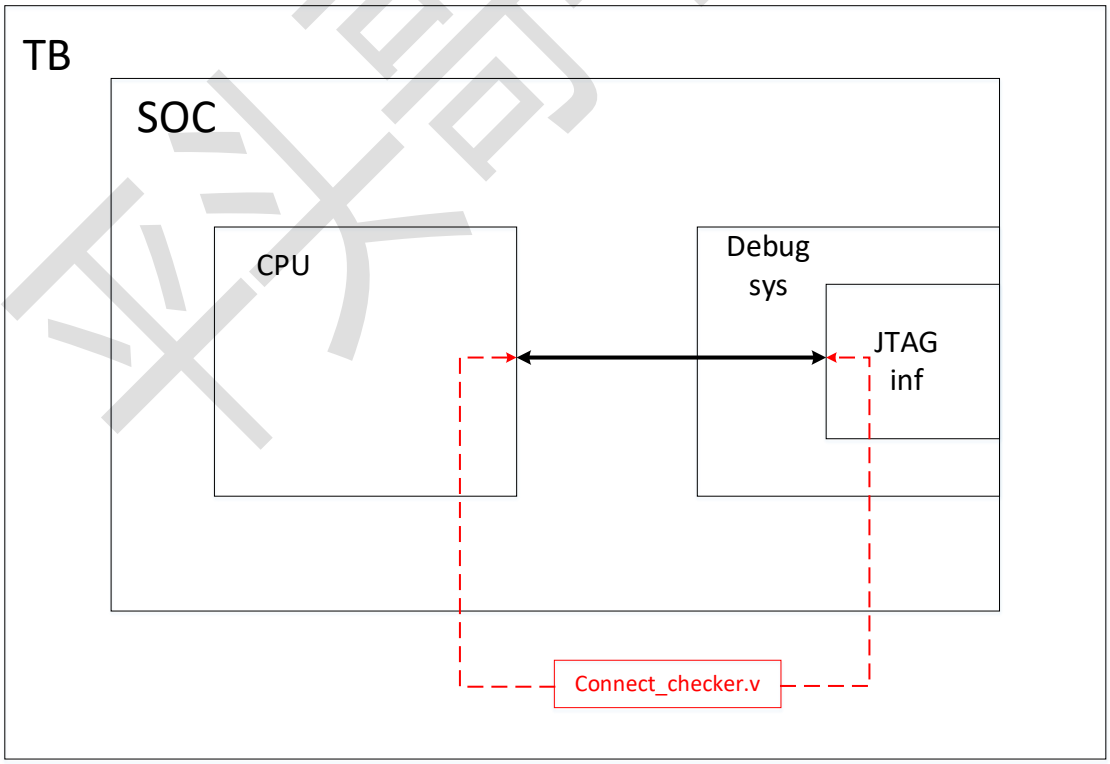
测试程序名称：ilite _interface.s

测试程序名称：dlite _interface.s

测试程序名称：vic_interface.s

该测试程序共有两个文件，包括一个 verilog 文件一个汇编文件，用于客户集成测试，需要客户在 verilog 文件中配置 CPU 的例化路径以及与 CPU 对应子模块联通模块路径。，客户在自己的 TB 里例化.v 或者加入到 vcs 的 verilog 列表中即可以进行仿真。测试程序分了一下几个子系统进行编写：

- 时钟复位信号联通性测试
- 总线系统联通性测试
- 中断系统联通性测试
- 调试系统联通性测试 ，force 信号 setdr，检查状态，读取寄存器值比较。
- 低功耗系统联通性测试，force lpmd_b,
- DFT 信号联通性测试
- BIST 信号联通性测试



图表 2-5 **例化** connect checker

如图所示，connect 测试程序 被实例化在 tb 层，用户设定 CPU 的例化路径以及与被测系

统连接模块的例化路径。例如上图中的 debug 子系统测试，注意这里要求信号名字不能改变。这样就可以灌激励进 soc 层和 cpu 层来达到测试连接性的目的。

以 debug 子系统测试程序举例：

（1） 测试程序最开头定义了各个模块的端口信号在 soc 层的路径，tb.soc.debug_sys.jtag_inf

（2） 定义各个模块的端口信号在 CPU 层的路径，tb.soc.cpu

（3） Jtag 模块信号通过 force tb.soc.debug_sys.jtag_inf 上的 JTAG 信号来控制 HAD 写入 HCR 寄存器数据，set dr 位进入 debug mode，然后再通过 force 的方法 去操作 HAD 读 HCR 寄存器，如果读出正确的数据，则说明 jtag 信号线连接正确

其它模块的端口信号，由于在 debug mode，没有指令执行的条件，所以只能通过 force 源信号，然后检测另一端的信号是否发生翻转，来检测是否连接正确。

除了连接性测试程序，我们还提供一些功能性的联通性测试程序，如下：

1. 总线信号联通性测试：通过构造总线不同的传输类型，检查读回数据结果是否正确来验证总线信号联通性是否正确。
2. 中断信号联通性测试：通过设置外部时钟中断来触发中断，并验证 CPU 是否能正确响应来验证中断信号联通性是否正确。注：外部中断需要客户根据自己的 soc 上的中断来设置如何触发中断。

2.5 工艺替换测试程序

软核授权客户测试 memory 和 gated cell 自己是否替换正确。该测试程序也属于集成验证。用户需要配置 CPU 代码的存放的绝对路径。该测试程序作为一个 TB 去 instance 用户修改后的 verilog 文件。该测试程序需要客户提供 memory 的仿真库文件。

2.5.1 Memory

测试程序名称：ckxx_mem_test_tb.v

memory 测试从 3 个方面进行验证，具体如下：

（1） 接口信号连接正确性验证；

- 片选信号测试使能： 写信号使能，数据能写入。
- 片选信号测试不使能：写信号使能，数据不能写入。
- 位写信号使能：片选有效，数据能写入。
- 位写信号不使能：片选有效，数据写无效， Q 端为读出的 old 数据。
- 写使能信号，有些 memory 有写使能信号，一般都是由位写使能信号缩位与做出，在测试位写使能时已经测试到了。
- 读数据测试：片选有效，写信号无效，数据和写入的相同。

（2） 数据和地址信号位宽不匹配验证和地址位宽比实际的小时数据覆盖；

- 地址位宽测试：向 0 和 1/2 最大地址写入一个数据，检查地址 0 和 1/2 最大地址写入的数据和读出的数据和预期的数据相等。
- 数据位宽测试：向一个地址写入一个最大位宽的数据，检查写入的数据和读出的数据和预期的数据相等。

（3） mem 模型功能验证

- 向 mem 写数据时，下一个周期 Q 端的数据和写入的数据相等
- 读数据是，下一个周期返回预期的 data

2.5.2 Gated cell

测试程序名称：ckxx_mem_test_tb.v

gated cell 的验证程序和 2.5.1 的 mem 验证程序为同一个程序。

Gated cell 测试，根据时钟控制信号的两个来源（local_en,和外部 en）进行组合验证，具体如下：

（1） 测试 en 是否有效

- Local en 使能验证：外部 en=0，clk_out 延迟一个周期有效。
  clk_out 当前周期为 0,下一个时钟 clk_out 为 1。
- Local en 使能关闭验证：外部 en=0，下一个时钟 clk_out 失效。
  clk_out 当前周期为 1,下一个时钟 clk_out 为 0。
- Test 使能验证：local en =0，test en 有效 clk_out 当拍开始有效。
- Test 使能关闭：local en =0，test en 无效 clk_out 当拍开始无效。
- Test en 和 local en， 同时打开，clk_out 当拍有效。
- Test en 和 local en，同时关闭，clk_out 下一怕失效。

（2） 检测 gated cell 类型

- 检测***两个 en 信号为 0 时， clk_out 一定是 0。

2.6 Functional test 测试程序

2.6.1 基本指令

测试程序名称：ck801/802/803_smoke.s

按照 CPU 和配置选取基本指令，带有自检功能。

1) 验证 Ck801 指令集中 87 条指令的基本功能。
2) 验证 Ck802 指令集中 154 条指令的基本功能。
3) 验证 Ck803 指令集中 232 条指令的基本功能。

2.6.2 浮点指令

测试程序名称：ck804_fpu_smoke.s

801、802、803s 没有浮点指令，覆盖 804 36 条浮点指令集。

2.6.3 DSP 指令

测试程序名称：ck804_dsp_smoke.s

801、802、803 没有 DSP 指令，覆盖 96 条 804 DSP 指令集。

2.6.4 模块测试测试程序

每个模块挑两个典型的测试程序 ，CK803S 模块测试程序列表：

Case name	Fuction
Ck803s_cp0_case.s	测试 cp0 basic 的指令（mtcr，mfcr，psrset，psrclr）
Ck803s_idu_case1.s	测试比较指令后的跳转能否正常跳转,测试 C 位能否正确 foward
Ck803s_idu_case2.s	测试 idu 在有 dependency 时跳转指令是否正确执行。
Ck803s_ifu_case1.s	测试 IFU 跳转分支指令
Ck803s_ifu_case2.s	测试 IFU 分别在 IBUS，DBUS，SBUS，取指令，指令执行正确
Ck803s_iu_case1.s	测试 ALU basic 指令
Ck803s_iu_case2.s	测试 Shift basic 指令
Ck803s_lsu_fuc_2.s	测试 load store 指令的 dependency
Ck803s_mad_case.s	测试 mad bacic 指令 function（mult mulsh divu divs）
Ck803s_mgu_case.s	测试 mgu basic 功能（配置 mgu 保护区域，设置访问权限，触发异常）
Ck803s_rtu_case1.s	1. 验证当异常出现在 split 指令未退休时，EPC 和 EPSR 会保存当前的 PC 和 PSR 2. 访问非对齐地址会触发非对齐异常
Ck803s_rtu_case2.s	测试 trace 异常时 会在进入异常前保存现场并在异常服务程序中清除 ee ie 位
CK802_bmu_case.s	该 case 的 caseID 设置为 803S 共用： IFU 分别在 IBUS，DBUS，SBUS，取指令，指令执行正确 数据分别在 IBUS，DBUS，SBUS，ld/st 指令执行正确

图表 2-6 CK803S 模块测试程序列表

Case name	Fuction
Ck802_bmu_case.s	IFU 分别在 IBUS，DBUS，SBUS，取指令，指令执行正确

	数据分别在 IBUS，DBUS，SBUS，ld/st 指令执行正确
Ck802_cp0_fuc.s	验证 mtcr 和 mfcr 指令读写 cpu 的 cr 寄存器。Trap 指令执行出 16 号异常
Ck802_ifu_fuc.s	执行各种分支和跳转指令，程序执行正确
Ck802_iu_split_exp.s	IU 执行 split 指令，出非对齐访问异常
Ck802_iu_alu.s	IU 执行带进位的有符号加减法，执行正确
Ck802_lsu_fuc.s	Lsu 部件从 IBUS 和 DBUS 对应的空间执行 LD/ST 操作结果正确
Ck802_mpu_fuc.s	未设置 mpu，Lsu 读写数据正常， 设置 mpu，lsu 读写指定地址出 access error。
Ck802_cache_fuc.s	Cache 可读不可写。 cache hit 情况下： 执行写操作后，数据更新到外部 memory，cache 中的数据不更新。 Cache 关闭情况下： 读写数据都是直接对外部 memory 进行读写。

图表 2-7 CK802 模块测试程序列表

Case name	Fuction
Ck801_bmu_case.s	IFU 分别在 IBUS，DBUS，SBUS，取指令，指令执行正确；数据分别在 IBUS，DBUS，SBUS，ld/st 指令执行正确
Ck801_cp0_fuc.s	验证 mtcr 和 mfcr 指令读写 cpu 的 cr 寄存器。Trap 指令执行出 16 号异常
Ck801_ifu_fuc.s	执行各种分支和跳转指令，程序执行正确
Ck801_iu_split_exp.s	IU 执行 split 指令，出非对齐访问异常
Ck801_iu_alu.s	IU 执行带进位的有符号加减法，执行正确
Ck801_lsu_fuc.s	Lsu 部件从 IBUS 和 DBUS 对应的空间执行 LD/ST 操作结果正确
Ck801_mpu_fuc.s	未设置 mpu，Lsu 读写数据正常， 设置 mpu，lsu 读写指定地址出 access error。

图表 2-8 CK801 模块测试程序列表

2.6.5 异常向量表配置测试程序

测试程序名称：exception_test.s

异常向量表配置测试程序演示设置异常和中断服务程序入口地址，包括；设置异常向量表地址寄存器 VBR；计算异常向量表偏移量，将异常服务程序的入口地址写到异常向量号对应的异常

向量表地址。

（1） 写非法指令异常服务程序和特权违反异常服务程序，都是跳到下一个 pc

（2） 一条 0xffff 指令构造出非法指令异常，进入异常服务程序后跳出，set psr 到 user 态，一条 mtcr 指令构造特权违反异常。

（3） 在写异常服务程序的 macro 中添加注释，详细说明怎么配置异常向量表寄存器

```
//
.macro SETEXP OFFSET, HANDLER_BEGIN, HANDLER_END
  lrw  r2, 0
  mtcr r2, cr<1,0>          //set VBR register to initial the exception handler base addr
  lrw  r1, \HANDLER_BEGIN   //set handler program begin addr
  mfcr r2, cr<1, 0>         //
  movi r3, \OFFSET          //OFFSET is the exception vector
  lsli r3, r3, 2            //shift
  addu r2, r2, r3           //calculate the exception vector entry addr
  st.w r1, (r2,0)           //store the exception handler program addr to the exception ve
ctor entry addr
  br \HANDLER_END
.endm
```

图表 2-9 异常服务程序设置

2.6.6 Cache 使用（cacheable 配置，invalid、触发 burst）

测试程序名称：Ck802/803_cache_test.s

Cache 使用，产生读写 burst，为用户提供使用 cache 的示例，包括：1.设置 CIR 寄存器来 invalid one 或者 invalid all；设置 CRCR 来配置 cache region；设置 CER 寄存器来使能 cache 和设置 write back 等 cache 策略

（1） 往 BASE_ADDR 写入一个数据 0xabcd1234

（2） 通过 tcip 映射地址设置 CIR 寄存器，invalid BASE_ADDR 所在的 cacheline

（3） 通过 tcip 映射地址设置 CRCR0 寄存器，设置 cache region 大小

（4） 通过 tcip 映射地址设置 CER 寄存器，使能 CACHE，配置 write back 等设置

（5） 从 BASE_ADDR load 数据，构造一次 refill ，在总线上产生 burst 请求，读回的数据 check 是否为写入内存的值

（6） CK802 和 CK803 在设置寄存器上稍有不同，已经在测试程序的注释中体现

2.6.7 MPU 配置

测试程序名称：Ck801/802/803_mgu_test.s

示例，如何通过设置寄存器来配置 mpu 内存访问的区域。设置寄存器来控制 mgu 的访问权限；配置当前操作的区域。

（1） ACC_ERR 的异常服务程序，都是跳到下一个 pc 继续执行

（2） 开始配置 MGU 相关寄存器，先设第一个 region 为 AP0=1’b11（user/super can read/write），base 为 0 size 为 4G

（3） 配置寄存器，设第二个 region 为 AP1 =2’b10(super mode can read/write;user mode only can read)base=0x10000 size 为 4k

（4） 设置 CCR 来使能 MGU

（5） 分别用 user 和 super 去访问 10000 11000 12000。在 user 态访问 10000 会进入异常，在异常中把 s 位打开，变成 super 态。再访问这三个地址，selfcheck。

2.6.8 低功耗示例

不属于 cts，作为 SmartL 的演示程序，因为不知道具体 SOC 的低功耗方案，可能执行低功耗指令之后整个 SOC 都掉电，因此不作为 CTS 提供，平头哥 CPU 的低功耗演示程序可在 SmartL 里找到，具体参考《smartL 用户手册》。

2.6.9 中断嵌套

测试程序名称：ck801_802_803_usecintc_nest.s

中断嵌套高优先级的中断可以打断低优先级的中断，同优先级的不能打断。中断嵌套验证测试程序如下：

（1） 设置中断响应 flag，记录最新响应的中断号，每个中断服务程序都会先去更新该 flag

（2） 配置中断使能，中断优先级，int1 = int2<int3<int4<int5

（3） 触发 int1 中断，进入该中断的中断服务程序，在该中断服务程序，设置 int2 pending，检测中断响应 flag，若响应 int2 则测试程序 fail。

（4） Int1 异常服务程序处理完之后，响应 int2 异常服务程序，在 int2 中断服务程序中依次设置 int3，int4 和 int5 pending，通过中断 flag 检测高优先级的 int3，int4，int5 能嵌套 int2。

2.6.10 世界切换，EBR 使用

测试程序名称：ck801_802_803_tee_wsc_fuc.s

安全示例程序用于验证，cpu 核代码是否能再安全态和非安全态下运行，具体验证如下：

（1） 安全态配置 ebr，并注释说明

（2） 非可信世界下读写安全非安全的状态寄存器（T_EPSR，NT_EPSR,T_EPC,NT_EPC,第三组控制寄存器），非安全态的控制寄存器可读写，安全态的控制寄存器不可写，读为 0。

（3） 世界切换指令调用

（4） 安全态读写安全太和非安全态的控制寄存器，可读可写。

2.6.11 安全 Cache 测试测试程序

测试程序名称：Ck803_sec_cache_fuc.s

检测 cache 的安全防护功能是否正确：

（1） 切换到可信世界打开 cache，设置可信世界的 cacheable region 4G，配置 cache 为只读，不可写。

（2） 写安全 memory 和非安全 memory

（3） 分别 load 带安全属性的数据和非安全属性的数据到 cache。

（4） 修改安全和非安全数据，通过 store 操作写回。（cache 不可写，修改 memory）。

（5） 可信世界访问 cache，读非安全数据和步骤 2 中写入的相同。

（6） 可信世界访问 cache，读安全数据和步骤 2 中写入的相同。

（7） 切换到非可信世界。

（8） 非可信世界访问 cache，读非安全数据和步骤 2 中写入的相同。

（9） 非可信世界访问 cache 读安全数据，访问出 accerr。

（10） 切换到可信世界，关闭 cache，检测步骤 4 成功修改了 memory 中的数据。

2.6.12 安全中断

测试程序名称：ck801_802_803_usecw_secintc_nest.s

测试程序名称：ck801_802_803_secw_secintc_nest.s

测试程序名称：ck801_802_803_usecintc_nest.s

可信世界响应非安全中断：

（1） 进入可信世界

（2） 设置中断使能，中断优先级 int1 = int2<int3<int4<int5，设置所有中断为非安全中断。

（3） 通过 tcip 设置 int1 pending，进入 int1 的中断服务程序，保存 psr 信息，退出 int1 中断服务时，返回到可信世界。

（4） 检查保存的 psr 信息，保存的 psr 中 T 位为 0。

非可信世界响应安全中断：

（1） 进入可信世界

（2） 设置 int1 中断使能，且为安全中断。

（3） 设置 coretimer，N 个周期之后触发 int1

（4） 进入死循环，等待 int1 中断

（5） Int1 中断服务程序修改非可信世界 EPC，返回后跳出死循环

可信世界响应安全中断：

（1） 进入可信世界

（2） 设置所有中断使能，中断优先级 int1，设置所有中断为安全中断。

（3） 通过 tcip 设置 int1 pending，进入 int1 的中断服务程序，保存 psr 信息。

（4） 检查保存的 psr 信息，保存的 psr 中 T 位为 1。

非可信世界响应非安全中断，详细见中断嵌套测试程序；

2.6.13 世界切换硬件压栈和弹站

测试程序名称：ck801_802_803_usecw_secintc_nest.s

（1） 可信世界发生非安全中断，硬件压栈。

- 硬件压栈 T_PSR、T_PC（next PC）至安全超级用户堆栈。
- 硬件压栈可信世界现场 GPR 至安全超级用户堆栈（CK803S 压栈 r0~r13、r15、r28，r14 不压栈；CK802 压栈 r0~r13、r15，r14 不压栈,；CK801 压栈 r0~r8、r13、r15，r14 不压栈）。
- 硬件清零可信世界现场 GPR（被清零的寄存器与压栈相同）。
- 读取 r0~r8，检测是否为 0

（2） 非安全中断服务程序 rte 返回可信世界，硬件弹栈

- 硬件弹栈，从安全超级用户堆栈硬件弹栈恢复 GPR 现场。
- 硬件弹栈，从安全超级用户堆栈硬件弹栈 T_PSR 和 T_PC 作为可信世界现场。
- 读取 r0~r8，检测 GPR 是否恢复

（3） 非可信世界执行世界切换指令 wsc 到可信世界，执行硬件压栈

- 硬件压栈，非可信世界的 NT_PSR 和 NT_PC 到非可信世界的超级用户堆栈。

（4） 可信世界 rte 返回非可信世界，执行硬件弹栈

- 硬件弹栈，从非可信世界的超级用户堆栈硬件弹栈非可信世界的 NT_PSR 和 NT_PC 作为现场。

2.6.14 CoreTimer 使用

测试程序名称：ck801_802_803_coretimer_fuc.s

Coretimer 能够唤醒低功耗，具体验证如下：

（1） 设置中断使能，设置 psr ee 和 ie 位

（2） 设置 coretimer 控制寄存器，N 个周期后触发中断。

（3） 进入一个死循环。

（4） CoreTimer 在 N 个周期后触发中断，进入中断服务程序跳出步骤 3 中的死循环。

2.6.15 调试测试程序举例

测试程序名称：ck801/802/803s_had_test_jtag.s, ck801/802/803s_had_test_jtag.v

调试测试程序示例：1.提供操作 jtag 信号来进行 debug 的示例。包含读写 HAD 寄存器

（1） 提供两个文件，一个 verilog 文件一个汇编文件，verilog force jtag 信号，来告诉用户怎么通过 force jtag 信号来进入 debug mode 去读取或者写相关寄存器。所以在 tb 层也

需要例化这个 module 或者加入仿真的 verilog 文件列表

（2） 类似前面联通性测试程序，定义 tb 层的 jtag 信号路径

（3） Reset jtag 信号

（4） 控制 jtag 信号移数据进 HCR 寄存器，达到写 HCR 的 dr 位的目的

（5） 控制 jtag 信号将 HCR 寄存器中的数据移出，达到读 HCR 寄存器的目的

（6） Check 读出的值是否正确

2.6.16 BIST 测试程序

测试程序名称：ck802/803s_mbist.s, ck802/803s_mbist.v

本测试程序设计用于测试及演示 mem 的 bist 功能。用户配置 CPU 例化路径，测试程序以一个 task 的形式交互，客户的 tb 调研该 task 即可进行仿真。

（1） 一段时间后对 CPU 进行 reset

（2） Wait 各个 memory 的 smbist_done 信号

（3） 如 done 且 fail 则测试程序 fail,否则测试程序测试通过

2.6.17 DFT 仿真测试程序（高端）

根据以前的讨论 SmartL 上不提供 DFT 测试测试程序，因为低端核都是以软核的形式授权，暂时供该测试程序。

2.7 性能测试程序

测试程序名称：dhry_1_main.c

测试程序名称：core_main.c

测试程序名称：linpack_main.c

测试程序名称：mem_cope_main.c

测试程序名称：mem_set_main.c

测试程序名称：int_ack_test.c

性能测试程序可用于 RTL/FPGA 仿真，用于评估 SOC 平台性能，因为用于 RTL 仿真，所以测试的循环遍数默认都改小了一点，否则仿真时间会比较长。另外性能测试程序需要用到 timer，跑测试程序之前用户需要修改 timer 的驱动以适用客户的 SOC 平台。主要包括如下 testbench：

（1） Dhrystone，测试 CPU 整体性能

（2） Coremark，测试 CPU 整体性能

（3） Linpack，浮点性能（803S）

（4） Memory copy，内存拷贝性能测试

（5） Memory set，内存初始化性能测试

（6） 中断响应实时性，中断响应速度性能测试

如果用户不想使用自己的时钟驱动仿真性能，CTS 里默认设置了使用虚拟时钟，由 CPU 内部进行读取，他的频率和 CPU 一致，这样就省去了在不同 SOC 平台上客户都需要自己修改时钟驱动的过程。虚拟时钟的频率默认为 100MHz，但经过修改 dhrystone、coremark、memory copy、memory set、中断响应的性能是和仿真频率无关的，如果客户不愿使用修改过的程序，可将测试程序开头的宏定义 VCUNT_SIM 给注释掉，注释掉之后客户需要根据自己的 SOC timer 修改 timer 的驱动程序。并且如果仿真 CPU 时钟不是该频率需要修改测试程序里的宏定义。性能单位可参考 3.2.6 小结。

2.8 C 语言程序示例

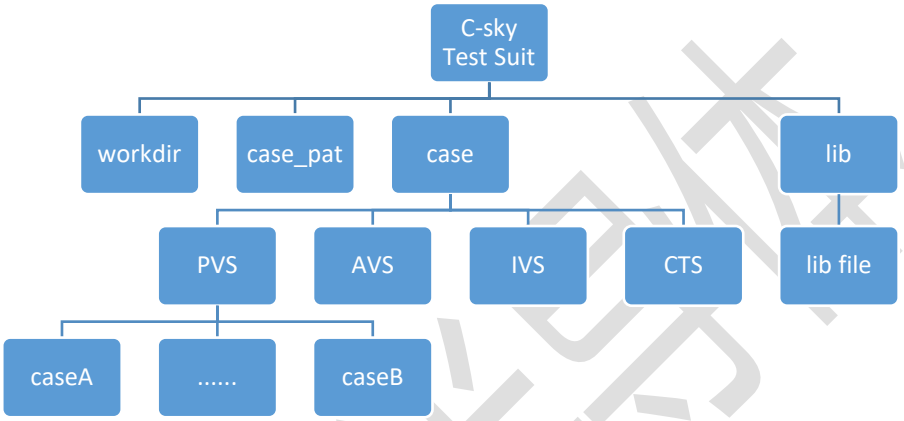
测试程序名称：hello_world_main.c

为了方便客户快速熟悉平头哥的仿真环境，该测试程序提供了一个简单的 C 语言程序例子，包括基本信息打印和 C 内嵌汇编等内容。

3 CTS 使用说明

3.1 CTS 目录结构

CTS 的目录结构如下图所示：



图表 3-1 CTS 的结构目录

注释：

1. Lib 目录包括了 linkfile， makefile，pat_gen.pl， crt0，clib/，CTS_monitor.v 等文件 ；
2. Workdir 目录是用户生产 pat 文件的工作目录；
3. Case 目录里包含了所有 CTS 的测试程序；
4. Case_pat 目录放的是由 case 目录里的测试程序对应生成的 pat 文件；

3.2 CTS 使用步骤

3.2.1 配置 SOC 平台信息

进入 smartL/tools/CTS_MAP_TOOLS，在 CTS 重映射到客户平台之前，客户需要先打开该目录下的 cts_map_cfig.h，配置一些目标平台的基本信息，如下截图所示：

```
//INST address and size
`define IMEM_ADDR 0x10000000
`define IMEM_SIZE  0x20000
//DATA address and size
`define DMEM_ADDR 0x20000000
`define DMEM_SIZE 0x10000

//TSSP TUSP SSP USP address
`define TSSP 0x6000f8f8
`define TUSP 0x6000f4f8
`define SSP  0x6000fff8
`define USP  0x6000fcf8
//CPU IMEM DMEM instance path
`define CPU_TOP          tb.soc.ahb.ck***_top
`define SOC_IMEM_INS     tb.soc.ahb.mema
`define SOC_DMEM_INS     tb.soc.ahb.memb

//base address
`define BASE_ADDR     0x60000100
//print address
`define PRINT_ADDR    32'h6000fff8

//toolchain
`define TOOL_EXTENSION  /home/wangmz/tmp/tool_chain
```

图表 3-2 重映射信息配置表

其中 IMEM_ADDR、IMEM_SIZE、DMEM_ADDR、DMEM_SIZE 分别为指令 MEM 的基地址和大小，数据 MEM 基地址和大小；

TSSP、TUSP、SSP、SSP 分别为可信世界超级用户态堆栈、可信世界用户态堆栈、非可信世界超级用户态堆栈、非可信世界用户态堆栈；如果 CPU 没有 TEE 配置可以无视可信世界的两个堆栈。

CPU_TOP、SOC_IMEM_INS、SOC_DMEM_INS 分别为 CPU 例化路径、IMEM 的例化路径、DMEM 例化路径；

BASE_ADDR 为测试程序内可使用的内存基地址址；

PRINT_ADDR 为仿真结果的打印地址；

TOOL_EXTENSION 为仿真工具链地址，平头哥提供工具链解压后地址；

用户需要根据实际情况设定相应的部分，脚本会根据用户设定生成一个适配当前配置的环境。

3.2.2 进行 CTS 重映射

重映射由脚本 smartL/tools/CTS_MAP_TOOLS/cts_map_tools.pl 完成，填好重映射配置表之后用户只需运行该脚本就可以完成文件夹创建、仿真环境创建、配置替换、测试程序挑选、pat 文件生成等一系列操作，生成用户 CTS 所需文件，该文件会调用下面所有有用的脚本，完成所有

remap 操作。

该脚本之后会生成 C-sky Test Suit 目录，如图 3-1 所示。

3.2.3 集成修改 csky_monitor.v

在进行仿真之前还需要把 csky_monitor.v 文件例化在其 SOC 仿真的文件中，csky_monitor.v 内容主要包含了以下三部分：

1. mem 的初始化：将 pat 文件读入指令和数据 memory；
2. 仿真结束条件监控；
3. 虚拟时钟，用于标准量化性能。

集成之后还需把将要跑测试程序的指令和数据 pat 文件修改为指令和数据存储的读入文件。如下图所示：

```
////////////////////////////////////////
// Memory Initialization
////////////////////////////////////////
integer i;
reg [31:0] mem_inst_temp [integer];
reg [31:0] mem_data_temp [integer];
initial
begin
  $display("\t******START TO LOAD PROGRAM******\n");
  $readmemh("inst.pat", mem_inst_temp);
  $readmemh("data.pat", mem_data_temp);
  for(i=0;i<mem_inst_temp.size;i=i+1)
  begin
    `SOC_IMEM_INS.ram0.U1.mem[i][7:0] = mem_inst_temp[i][31:24];
    `SOC_IMEM_INS.ram1.U1.mem[i][7:0] = mem_inst_temp[i][23:16];
    `SOC_IMEM_INS.ram2.U1.mem[i][7:0] = mem_inst_temp[i][15: 8];
    `SOC_IMEM_INS.ram3.U1.mem[i][7:0] = mem_inst_temp[i][ 7: 0];
  end
```

图表 3-3 读入 pat 文件

3.2.4 跑 CTS 仿真

因为客户的 tb 里已经例化了 csky_monitor.v，所以直接仿真原有的 tb 文件即可。考虑到客户自己的 tb 文件里已经对 CPU 的 clock 和 reset 进行了设置，避免重复赋值 csky_monitor.v 里并没有对 CPU 的 clock 和 reset 进行控制。

有些测试程序带有.v 文件，或者就是纯.v 文件，这种测试程序仿真之前需要将.v 文件加入到仿真的 verilog 文件列表中，否则测试程序无法运行，这些测试程序如下表所列：

分类	名称
.s 和.v 配合仿真，需将.v 加到仿真器的 verilog 列表中	Mbist.v/s
	ck801_connect.v/s

	had_test.v/s
只有.v	Ck80x_mem_test_tb.v,具体见该测试程序目录下的 readme

图表 3-4 verilog 结合仿真的测试程序

3.2.5 调试常见问题

如果发现测试程序仿真失败或者跑死，可以从以下几个方面来确认问题所在。

- 检查信号连接是否完整

确认所有信号是否完全连接完整，尤其是 CPU 的输入，具体方式可以在 CPU 的接口上将所有的输入信号的波形检查一遍是否存在高阻态，其中需要注意的是扫描链相关的信号线如果出现高阻态是可以忽视的。如果其他信号存在高阻态则需要将这些输入信号连接完成。

- 检查时钟和复位信号

确认所有的时钟信号和复位信号是否按照时钟产生器和复位产生器的格式生成，如果时钟或者复位信号不正确则会直接导致 CPU 无法正确工作。

- 检查总线是否发起请求

检查总线是否发起第一次请求，在复位完成以后 CPU 会进入重启异常，然后回从该异常向量表发起请求取得异常服务程序入口地址。该异常向量表地址一般为 0。

- 检查总线是否响应请求

在 CPU 发起请求之后，总线以及外围的存储器会立即响应，并将 CPU 需要的数据返回。第一次返回的数据一般为程序的入口地址，也就是重启异常的异常服务程序起始地址。

- 检查是否出现异常

观察 PSR 的输出值，判断处理器是否进入异常，确定是几号异常然后找到触发异常的具体指令加以分析。

- 检查是否修改指令区

程序跑飞有可能因为程序中的指令区被程序自身修改，导致程序没有按照需求执行，修改指令区是一件非常危险的事情，所以在程序设计的时候一定要避免出现这种情况。

3.2.6 性能统计反馈

仿真完如下几个测试程序，都会打印出仿真结果，请将结果填入下表然后返还给平头哥，我们会分析结果，如果发现问题的话会给出提升性能的建议。

名称	性能	单位
Dhrystone		Dmips/MHz
Coremark		Iterations/sec
Memory copy		bytes/Cycle

Memory set		bytes/Cycle
Int ack		Cycles
Linpack		Kflops

图表 3-5 性能测试程序

注意经过修改后的 dhrystone、coremark、memory copy、memory set、中断响应的性能是和仿真频率无关的，但是 linpack 是和仿真频率相关的，即便使用了 vtimer 也需要将里面的频率改为仿真频率才能得出准确的性能。修改后如果在性能统计过程中客户希望使用自己的 timer，客户需要修改测试程序里调用 timer 相关函数的驱动，并且把测试程序开头的宏定义 VCUNT_SIM 注释掉。

3.3 客户开发测试程序

pat_gen.pl 脚本是编译脚本，可以将 C 或者汇编写的测试程序编译成仿真用的 pat 文件。该脚本在执行 cts_map_tools.pl 之后，会被复制至 CSKY_Test_Suit 文件夹下，并且已经根据用户配置进行了重新设定，用户可以使用 pat_gen.pl ./case 的方式单独对某一个测试程序进行编译，生成的 pat 文件存放在 CSKY_Test_Suit/workdir 下。

附件 A：CTS 测试程序列表

测试程序 分类	测试程序名称	备注
联通性测试程序	ck801_connect.v/s	集成验证
	ck802_connect.v/s	集成验证
	ck803_connect.v/s	集成验证
工艺映射测试程序	Ck802_mem_test_tb.v	集成验证，包括 memory 和 gated_cell 测试
	Ck803_mem_test_tb.v	集成验证，包括 memory 和 gated_cell 测试
Function 测试程序		
801 模块测试程序	Ck801_802_803_bmu_case.s	IFU 分别在 IBUS，DBUS，SBUS，取指令，指令执行正确；数据分别在 IBUS,DBUS,SBUS,ld/st 指令执行正确
	Ck801_802_cp0_fuc.s	验证 mtcr 和 mfcr 指令读写 cpu 的 cr 寄存器。Trap 指令执行出 16 号异常
	Ck801_ifu_fuc.s	执行各种分支和跳转指令，程序执行正确
	Ck801_iu_split_exp.s	IU 执行 split 指令，出非对齐访问异常
	Ck801_iu_alu.s	IU 执行带进位的有符号加减法，执行正确
	Ck801_lsu_fuc.s	Lsu 部件从 IBUS 和 DBUS 对应的空间执行 LD/ST 操作结果正确
	Ck801_mpu_fuc.s	未设置 mpu，Lsu 读写数据正常，设置 mpu，lsu 读写指定地址出 access error。
802 模块测试程序	Ck801_802_803_bmu_case.s	IFU 分别在 IBUS，DBUS，SBUS，取指令，指令执行正确 数据分别在 IBUS，DBUS，SBUS，ld/st 指令执行正确
	Ck801_802_cp0_fuc.s	验证 mtcr 和 mfcr 指令读写 cpu

		的 cr 寄存器。Trap 指令执行出 16 号异常
	Ck802_ifu_fuc.s	执行各种分支和跳转指令，程序执行正确
	Ck802_iu_split_exp.s	IU 执行 split 指令，出非对齐访问异常
	Ck802_iu_alu.s	IU 执行带进位的有符号加减法，执行正确
	Ck802_lsu_fuc.s	Lsu 部件从 IBUS 和 DBUS 对应的空间执行 LD/ST 操作结果正确
	Ck802_mpu_fuc.s	测试访问权限， 测试读写权限；
	Ck802_cache_fuc.s	Cache 可读不可写。 Cache hit 情况下： 执行写操作后，数据更新到外部 memory，cache 中的数据不更新。 Cache 关闭情况下： 读写数据都是直接对外部 memory 进行读写。
803S 模块测试程序	Ck803s_cp0_case.s	测试 cp0 basic 的指令（mtcr，mfcr，psrset，psrclr）
	Ck803s_idu_case1.s	测试比较指令后的跳转能否正常跳转，测试 C 位能否正确 foward
	Ck803s_idu_case2.s	测试 idu 在有 dependency 时跳转指令是否正确执行。
	Ck803s_ifu_case1.s	测试 IFU 跳转分支指令
	Ck803s_ifu_case2.s	测试 IFU 分别在 IBUS，DBUS，SBUS，取指令，指令执行正确
	Ck803s_iu_case1.s	测试 ALU basic 指令
	Ck803s_iu_case2.s	测试 Shift basic 指令
	Ck803s_lsu_fuc_2.s	测试 load store 指令的 dependency
	Ck803s_mad_case.s	测试 mad bacic 指令 function（mult mulsh divu divs）
	Ck803s_mgu_case.s	测试 mgu basic 功能（配置 mgu

		保护区域，设置访问权限，触发异常）
	Ck803s_rtu_case1.s	验证当异常出现在 split 指令中时，EPC 和 EPSR 会保存当前的 PC 和 PSR 访问非对齐地址会触发非对齐异常
	Ck803s_rtu_case2.s	测试 trace 异常时 会在进入异常前保存现场并在异常服务程序中清除 ee ie 位
MGU 设置示例	mgu_test.s	示例
异常向量表配置示例、	exception_test.s	示例
Cache 使用示例	ck802_cache_test.s	示例
	ck803_cache_test.s	示例
TEE 安全示例	tee_wsc_fuc.rs	示例，可信世界和非可信世界读写相关控制寄存器。
TEE cache 示例	ck802_tee_cache.s	安全 cache 演示 暂时未写，802cache 不能访问 SYSTEMBUS
	ck803_tee_cache.s	安全 cache 演示 安全世界可以访问安全和非安全数据。 非安全世界访问安全数据出 accerr
CoreTimer 设置示例	Ck801_802_coretimer_fuc.rs	示例，通过 coretimer 进入中断服务程序，跳出死循环。
INT 嵌套，非安全中断	Ck801_802_unsecintc_nest.s	示例： 非安全中断嵌套，可信世界，发生非安全中断，会执行世界切换。
非可信世界，安全中断	Ck801_802_unsecw_secint.s	非可信世界，通过 tcip 设置安全中断无效
可信世界，安全中断	Ck801_802_secw_secintc.s	可信世界执行安全中断，不发生世界切换
调试示例 case	had_test.v，had_test.s	示例

Bist 测试程序	Mbist.v/s	Memory bist 测试
Low power 测试程序	Ckcpu_wait.s	示例
	Ckcpu_doze.s	示例
	Ckcpu_stop.s	示例
C 语言程序举例	hello_world.c	C 语言编程例程
Max power 测试程序	Ckcpu_max_power.s	仿真功耗
	ck803s_max_power.s	仿真功耗
性能测试程序	Dhrystone.c	整体性能
	coremark.c	整体性能
	Linpack.c	浮点性能
	mem_copy.c	内存拷贝性能
	mem_set.c	内存初始化性能
	Int_ack_acc.s	中断响应性能
Coverage 测试程序	801_cover_80.rs	Coverage 为 80%
	801_cover_95.rs	Coverage 为 90%
	802_cover_80.rs	Coverage 为 80%
	802_cover_95.rs	Coverage 为 90%
	803s_cover_80.rs	Coverage 为 80%
	803s_cover_95.rs	Coverage 为 94%
Critical path 测试程序	ck801_critical.s	芯片测试
	ck802_critical.s	芯片测试
	ck803s_critical.s	芯片测试

总共大概 70 个测试程序。