CK CPU 软核
使用手册

声明：

杭州中天微系统有限公司(C-SKY Microsystems Co.,Ltd)保留本文档的所有权利。
本文档的内容有可能发生更改、更新、删除、变动，恕不另行通知。

公司地址：杭州市西湖区西斗门路 3 号天堂软件园 A 座 15 层
邮政编码：310012
电话：0571-88157059
传真：0571-88157059-8888
主页：www.c-sky.com
E-mail：info@c-sky.com

版本历史:

版本	日期	描述	作者
1.0	06/17/2016	第一版用户手册	杭州中天微系统有限公司
1.1	12/21/2016	增加 RAM 的读写时序描述	杭州中天微系统有限公司
1.2	12/05/2018	增加 FPGA 版本下文件替换描述	杭州中天微系统有限公司
1.3	12/12/2018	增加门级仿真中寄存器初始化参考示例	杭州中天微系统有限公司

目录：

图片目录：

表格目录

1. 概述

1.1. 简介

该文档主要面向 C-Sky 软核授权的用户。目的是引导客户快速替换 memory 和 gated cell 将代码映射到不同的工艺下，方便进行综合及后端流程。同时也方便进行各项评估，尝试通过替换不同规格的 memory 得到面积，功耗，时序等数据然后进行选择。对于部分需要将集成后的软核代码置于 FPGA 平台进行测试的客户，提供了 FPGA 版本下 memory/gated cell/designware 的替换操作。

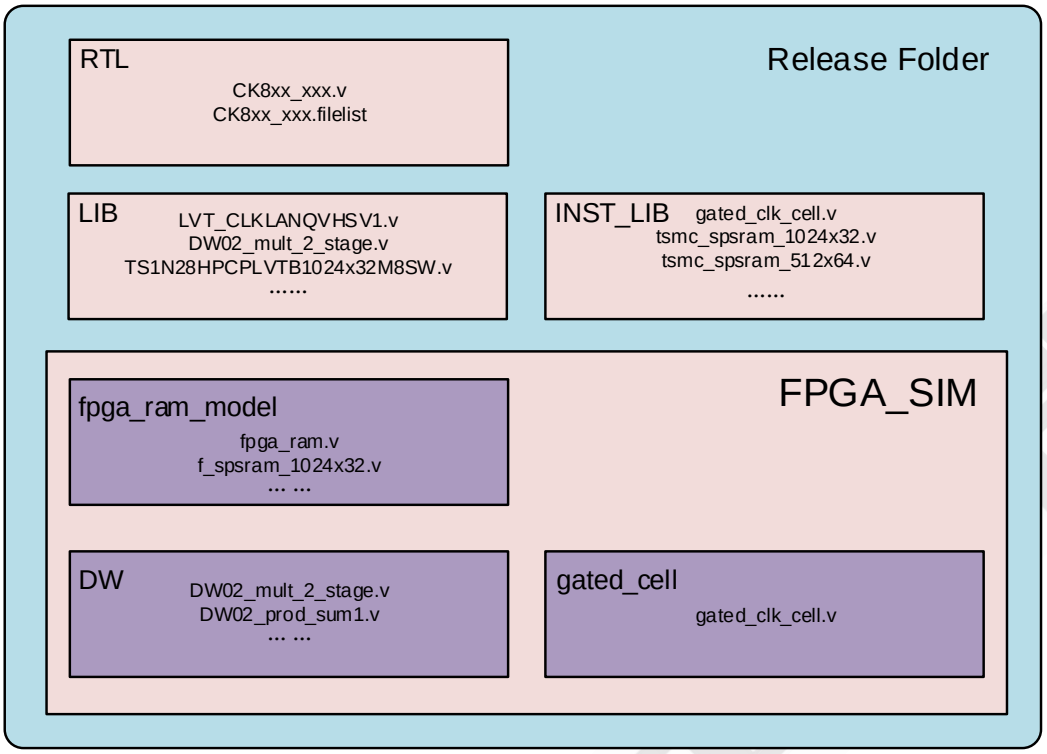
同时文档中还介绍了 memory bist 相关内容，支持用户用中天的 memory BIST，也支持用户自己插 memory BIST。

另外还有快速评估前端功耗数据的方法介绍以及针对门级仿真中对没有复位端的寄存器进行初始化的方法介绍。

2. 映射到不同工艺

2.1. RTL 代码结构

中天软核授权的客户都可以得到 release 文件夹，文件夹结构如下：



图片 1 release 文件夹内容

RTL 目录：包含了内核代码 CK8xx_xxx.v，以及一个 file list 文件 CK8xx_xxx.filelist，里面包含所有的.v 文件，综合时需要把该文件列表加载进去。

INST_LIB 目录：所有 memory 例化文件和 gated cell 例化文件。

LIB 目录：所有的 memory，gated cell 和 design ware 仿真模型。

FPGA_SIM/fpga_ram_model 目录：FPGA 平台下仿真的 memory 例化文件和模型文件。

FPGA_SIM/DW 目录：FPGA 平台下仿真的 design ware 文件。

FPGA_SIM/gated_cell 目录：FPGA 平台下仿真的 gated cell 文件。

2.2. 替换 memory（ASIC 版本）

2.2.1. 文件所在位置

所有需要修改的文件都在 INST_LIB 目录，且文件名都包含关键词 spsram_AAAxBBB, A、B 分别代表 memory 的深度和宽度。

2.2.2. 生成 memory

所有需要的 memory 信号都可以从 INST_LIB 里看出（AAAxBBB），在生成时我们只对一个选项有要求，就是所有的 memory 都需要支持位写使能信号。其他的选项用户可以根据自己的需求选择，包括 memory 的形状、时序、面积、功耗等因素。如果觉得有些 memory 过大或者过深，可以自行进行拼接，后面会介绍 memory 的拼接。

2.2.3. 端口名字

该文件例化了具体工艺的 memory 库 model，如下图所示：

```
35        sprf065ll_512x64 x_spsram_512x64(
36          .A    (A),
37          .D    (D),
38          .BWEN ({{32{WEN[1]}}, {32{WEN[0]}}}),
39          .WEN  (&WEN),
40          .CEN  (CEN),
41          .CLK  (CLK),
42          .Q    (Q)
43        );
```

图片 2 例化 memory

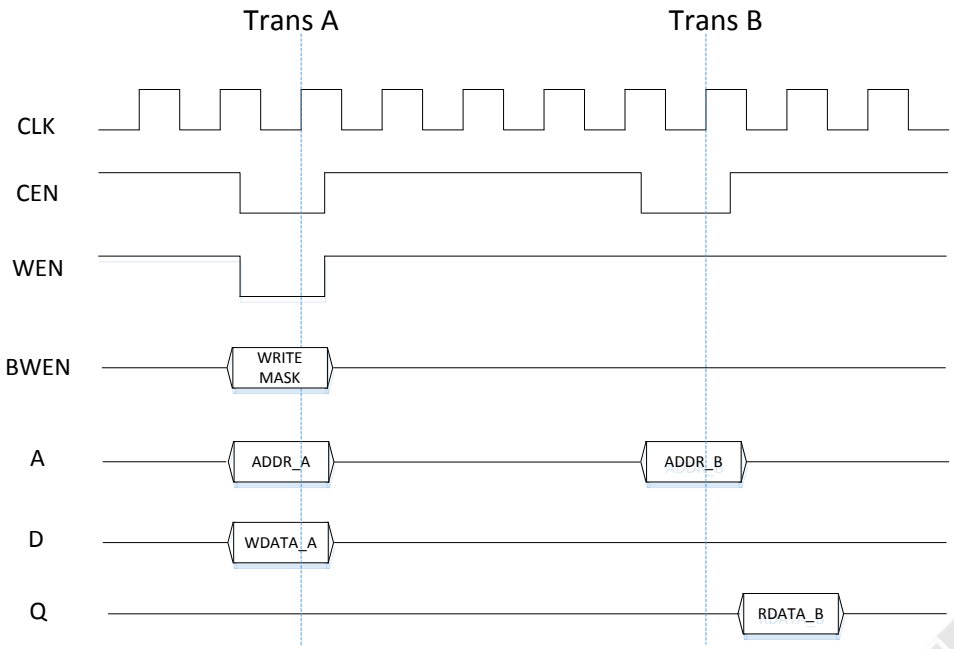
表 1 memory 信号列表

信号线名	功能	连接
A	地址线	RAM 的地址线端口
D	写入数据线	RAM 的数据输入端
WEN*	低电平有效，位写使能信号	RAM 位写使能信号端 缩位与之后做 RAM 写使能信号
CEN*	低电平有效，位片选信号	RAM 使能信号
CLK	时钟	RAM 时钟端
Q	输出数据线	RAM 的输出

*这两根信号都是低电平有效，具体揭发要看 RAM 的相应信号是否也是低电平有效，另外，不同 vender 提供 ram 可能还有一些其他的控制 port，针对这些 port 用户需要根据自己的需要按照 RAM vender 的用户手册进行处理。

2.2.4. RAM 的读写时序

下图是一个 RAM 的读写时序图，Trans A 是一个写请求，Trans B 是一个读请求。写部分数据是 BWEN 控制的，参考图 2 的具体连接方法。Trans B 时钟上升沿采到到输入读请求，下一个 cycle 将数据输出到 Q 端。



图片 3 RAM 的读写时序图

2.2.5. 修改 memory 例化文件

修改 memory 就是将 RAM 库文件替换掉，比如这里就将上一节中的 memory 换成了 synopsys 提供的 memory，多出的管脚参考 vender 提供的手册后 tie 上了具体的值。之后将原来例化 memory 的代码注释或者删除。

```
46        sprf065lp_synop_512x64 x_spsram_512x64(
47          .A    (A),
48          .D    (D),
49          .BWEB ({{32{WEN[1]}}, {32{WEN[0]}}}),
50          .WEB  (&WEN),
51          .CEB  (CEN),
52          .CLK  (CLK),
53          .TURBO(1'b1),
54          .RTSEL(1'b0),
55          .TSEL (2'b01),
56          .Q    (Q)
57        );
```

图片 4 修改 memory 例化文件

2.2.6. 拼接 memory

如果有些 memory 规格生成不出来，或者客户对该规格 memory 的形状或者其他一些特性不满意，用户可以用更小的 memory 拼接出需要的 memory，比如一块 2048x32 的 memory：

```
127       spsram040g_2048x32 x_spsram_2048x32(
128         .A    (A),
129         .D    (D),
130         .BWEB ({{8{WEN[3]}},{8{WEN[2]}},{8{WEN[1]}},{8{WEN[0]}}}),
131         .WEB  (&WEN),
132         .CEB  (CEN),
133         .CLK  (CLK),
134         .DELAY(2'b00),
135         .TEST (2'b00),
136         .Q    (Q)
137       );
```

图片 5 需要拼接的 memory

某个工艺下并不支持该规格的 memory，就需要用用其他规格 memory 进行拼接，下面是用两块 1024x32 的 memory 进行拼接的例子，当然也可以 2048x16 的拼接，那是另外一种接法。

```
assign CEN0 = CEN |  A[ADDR_WIDTH-1];
assign CEN1 = CEN | ~A[ADDR_WIDTH-1];

always@(posedge CLK)
begin
  if (!CEN)
  begin
    bank_sel <= A[ADDR_WIDTH-1];
  end
  else
  begin
    bank_sel <= bank_sel;
  end
end
assign  Q[31:0] = bank_sel ? Q1[31:0] : Q0[31:0];
sprf065lp_1024x32 x_spsram_1024x32_bank0(
      .A    (A[ADDR_WIDTH-2:0]),
      .D    (D),
      .BWEB ({ {8{WEN[3]}}, {8{WEN[2]}},{8{WEN[1]}}, {8{WEN[0]}} }
            ),
      .WEB  (&WEN),
      .CEB  (CEN0),
      .CLK  (CLK),
      .TURBO(1'b1),
      .RTSEL(1'b0),
      .TSEL (2'b01),
      .Q    (Q0)
    );

  sprf065lp_1024x32 x_spsram_1024x32_bank1(
      .A    (A[ADDR_WIDTH-2:0]),
      .D    (D),
      .BWEB ({ {8{WEN[3]}}, {8{WEN[2]}},{8{WEN[1]}}, {8{WEN[0]}} }
            ),
      .WEB  (&WEN),
      .CEB  (CEN1),
      .CLK  (CLK),
      .TURBO(1'b1),
      .RTSEL(1'b0),
      .TSEL (2'b01),
      .Q    (Q1)
    );
```

图片 6 拼接 memory

2.2.7. 修改 filelist

Memory 替换完后需要将，新的 memory 仿真模型文件和库文件都准备好，可以放入 LIB 文件里也可以放到用户自己的文件里，重要的是需要修改 filelist，删除以前调用的 mode 加入现在调用的文件路径。

2.3. 替换 memory（FPGA 版本）

本节旨在对 FPGA 平台下所需 memory 文件替换操作进行说明。

2.3.1. 文件所在位置

在 release 文件夹的 FPGA_SIM 目录下存放着 FPGA 平台下仿真以及制作 FPGA bit 文件时所需的所有需要替换的文件。其中 memory 文件在 fpga_ram_model 目录下，且文件名都包含关键词 f_spsram_AAAxBBB.v，A、B 分别代表 memory 的深度和宽度。

2.3.2. 端口名字

根据写使能信号作用域的不同，在每个 memory 文件里可由一块或者多块位宽不定的 memory 模型拼接而成。

```
fpga_ram #(WRAP_SIZE,ADDR_WIDTH) ram0(
  .PortAClk (CLK),
  .PortAAddr(addr),
  .PortADataIn (ram0_din),
  .PortAWriteEnable(ram0_wen),
  .PortADataOut(ram0_dout));


fpga_ram #(WRAP_SIZE,ADDR_WIDTH) ram1(
  .PortAClk (CLK),
  .PortAAddr(addr),
  .PortADataIn (ram1_din),
  .PortAWriteEnable(ram1_wen),
  .PortADataOut(ram1_dout));

fpga_ram #(1,ADDR_WIDTH) ram2(
  .PortAClk (CLK),
  .PortAAddr(addr),
  .PortADataIn (ram2_din),
  .PortAWriteEnable(ram2_wen),
  .PortADataOut(ram2_dout));
```

图片 7 FPGA 版本 memory 拼接

如上图所示为某个 f_spsram_64x59.v 具体例化的三块 FPGA 的仿真模型 fpga_ram，每块 fpga_ram 的数据输入位宽由下图所示指定。

```
parameter ADDR_WIDTH = 6;
parameter WRAP_SIZE  = 29;

assign ram0_wen = !CEN && !WEN[0];
assign ram1_wen = !CEN && !WEN[1];
assign ram2_wen = !CEN && !WEN[2];
//din
assign ram0_din[WRAP_SIZE-1:0] = D[WRAP_SIZE-1:0];
assign ram1_din[WRAP_SIZE-1:0] = D[2*WRAP_SIZE-1:WRAP_SIZE];
assign ram2_din                = D[2*WRAP_SIZE];
```

图片 8 FPGA 版本 memory 接口（一）

同理 f_spsram_64x59.v 的数据输出也由三块 fpga_ram 的数据输出拼接而成。

```
assign Q[WRAP_SIZE-1:0]            = ram0_dout[WRAP_SIZE-1:0];
assign Q[2*WRAP_SIZE-1:WRAP_SIZE]  = ram1_dout[WRAP_SIZE-1:0];
assign Q[2*WRAP_SIZE]              = ram2_dout;
```

图片 9 FPGA 版本 memory 接口（二）

2.3.3. 修改 memory 例化文件

同 ASIC 版本一样，修改 FPGA 版本下 memory 例化文件是指将 INST_LIB 下面对应大小的 memory wrap 文件里面例化的指定工艺的 memory 仿真模型替换为上节提到的 f_spsram_AAAxBBB.v。如下图所示为将 umc_spsram_64x59.v 里面例化的 memory 模型替换为 FPGA 下的仿真模型。

```
//U28HPC_ARM_RFS64X60M60 x_spsram_64x59(
//  .EMA        (3'b011),
//  .EMAW       (2'b01),
//  //.EMAS       (1'b0),
//  .TEN        (1'b1),
//  .RET1N      (1'b1),
//  .SE         (1'b0),
//  .DFTRAMBYP  (1'b0),
//  .A          (A),
//  .D          (D),
//  .GWEN       (&WEN),
//  .WEN        ({1'b1,WEN[2],{29{WEN[1]}},{29{WEN[0]}}}),
//  .CEN        (CEN),
//  .CLK        (CLK),
//  .Q          (Q)
//);

f_spsram_64x59  x_spsram_64x59(
  .A          (A),
  .CEN        (CEN),
  .CLK        (CLK),
  .D          (D),
  .Q          (Q),
  .WEN        (WEN)
);
```

图片 10 修改 FPGA 版本 memory 例化文件

2.3.4. 修改 filelist

修改完例化的 memory 文件后需要将上述的 f_spsram_AAAxBBB.v 以及 fpga_ram.v 文件加入 CK8xx_xxx.filelist，同时将 LIB/目录下被替换的原指定工艺的 memory 文件去掉。

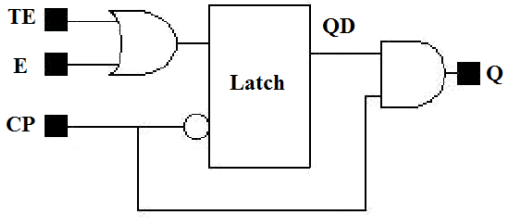
2.4. 替换 gated cell（ASIC 版本）

2.4.1. 文件所在位置

所有需要修改的文件在 INST_LIB 目录，文件名为 gated_clk_cell.v，该文件里例化了特定工艺下的 gated cell。

2.4.2. 选择 gated cell

中天所用的 geted cell 都是上升沿有效的，其逻辑图和真值表如下：



Truth Table

INPUT			OUTPUT
TE	CP	E	Q
0	⌐	0	0
0	⌐	1	1
1	⌐	x	1
x	0	x	0
x	1	x	QD

图片 11 正沿触发 gated cell

一般工艺库都会有类 gated cell，一个是正沿触发而另一类是负沿触发的，注意需要选择和以上真值表一样的正沿的 gated cell。

一般工艺库里同一类有不同规格的 gated cell，这就需要用户根据自己的需求进行选择，平衡速度和功耗等因素，如下图例子是 TSMC28 工艺下正沿触发 gated cell 的规格：

Cell Information

(Characterization Condition:Process=Slow-Slow-Global,Voltage=0.72v,Temp=125degreeC)

PG Pin=VDD

Cell Name	Gate Count	Width(um)	Leakage(nW)		
			Min.	Ave.	Max.
CKLNQD12BWP30P140	12	3.92	204.402	245.23958	289.03
CKLNQD16BWP30P140	15	4.76	261.038	322.157	379.021
CKLNQD1BWP30P140	5.5	2.1	64.8902	69.24762	76.4891
CKLNQD20BWP30P140	18	5.6	316.508	398.60492	468.128
CKLNQD24BWP30P140	20	6.16	368.175	457.71108	542.016
CKLNQD2BWP30P140	6	2.24	74.3782	82.31652	89.418
CKLNQD3BWP30P140	6.5	2.38	84.2502	95.67109	109.96
CKLNQD4BWP30P140	7	2.52	95.644	109.48958	128.89
CKLNQD6BWP30P140	9	3.08	126.833	156.76508	178.899
CKLNQD8BWP30P140	10	3.36	152.72	186.31308	215.794
CKLNQOPTMAD16BWP30P140	18.5	6.16	342.611	380.62767	436.376
CKLNQOPTMAD4BWP30P140	10	3.78	165.473	176.1605	194.235
CKLNQOPTMAD8BWP30P140	12.5	4.48	220.022	242.23692	268.998

图片 12 TSMC28 工艺下正沿触发 gated cell 的规格表

2.4.3. 端口名字

gated_clk_cell.v 文件中例化了具体工艺库的 gated cell，如下图所示：

```
61    CKLNQD8BWP30P140 x_gated_clk_cell (
62                     .CP          (clk_in        ),
63                     .TE          (SE            ),
64                     .E           (clk_en_bf_latch),
65                     .Q           (clk_out       )
66                     );
```

图片 13 例化 gated cell

表 2 gated cell 信号列表

信号线名	功能	连接
clk_in	Clock 输入	Clock input
SE	Pad_yy_test_mode || Pad_yy_bist_tst_en || Pad_yy_gated_clk_en_b	Test clock enable
Clk_en_bf_latch	Clock function 使能信号	Function Clock enable
clk_out	Clock 输出	Clock output

2.4.4. 更改 gated_clk_cell.v 文件

选定 gated cell 之后修改 gated_clk_cell.v 文件，如下图所示：

```
//    CKLNQD8BWP30P140 x_gated_clk_cell (
//                  .CP           (clk_in),
//                  .TE             (SE),
//                  .E      (clk_en_bf_latch),
//                  .Q          (clk_out)
//                  );

      PREICG_X0P5B_A9TL40 x_gated_clk_cell (
        .ECK  (clk_out ),
        .E    (clk_en_bf_latch ),
        .SE   (SE ),
        .CK   (clk_in )
      );
```

图片 14 修改例化文件

将原 gated cell 替换掉，接上新选的 gated cell。

2.4.5. 更新 filelist

Gated cell 替换完后需要将新的 gated cell 仿真模型文件准备好，可以放入 LIB 文件里也可以放到用户自己的文件里，重要的是需要修改 filelist，删除以前调用的 mode 加入现在调用的文件路径。

2.4.6. 不使用前端插的 gated cell

后端工具支持自动插 gated cell，如果客户不希望使用中天前端手动插的 gated cell，可以用简单注释掉 gated_clk_cell.v 里的 instance gated cell 部分代码然后加上行代码，如下图所示：

```
63 //    CKLNQD8BWP30P140 x_gated_clk_cell (
64 //                  .CP           (clk_in),
65 //                  .TE             (SE),
66 //                  .E      (clk_en_bf_latch),
67 //                  .Q          (clk_out)
68 //                  );
69
70 assign clk_out = clk_in;
```

图片 15 不使用前端插入的 gated cell

这样前端插的 gated cell 都没了，客户可以按照自己的意愿插入 gated cell。

2.5. 替换 gated cell（FPGA 版本）

将 CK8xx_xxx.filelist 里面包含的 INST_LIB 下的 gated_clk_cell.v 及其例化的特定工艺的 gated cell（LIB 目录下）替换为 FPGA_SIM/gated_cell/目录下的 gated_clk_cell.v 文件。

2.6. 替换 design ware（FPGA 版本）

对于部分版本的 ASIC 中存在的 synopsys 公司的 designware，为了方便不同 FPGA 厂商的工具软件都可以对其进行综合，我们提供了 FPGA 下可综合的代码版本，位于 FPGA_SIM/DW 目录下。只需将 CKxx_xxx.filelist 里面包含的 LIB 目录下的 designware 替换为 FPGA_SIM/DW 目录下同名的文件即可。

3. Memory bist

3.1. 客户使用中天的 memory bist

3.1.1. 中天的 memory bist 介绍

请参考相关核的集成手册

3.1.2. Memory BIST 端口信号（视处理器配置而定）

表 3 memory bist 信号列表

Signal name	I/O	reset	Function
bht_array_smbist_done	O	0	bist 测试完成信号: 0：表明 BHT array memory 自测试未完成； 1:表明 BHT array memory 自测试完成。
bht_array_smbist_fail	O	0	bist 测试失败信号: 0:表明 BHT array memory 自测试成功，memory 工作正常； 1:表明 BHT array memory 自测试失败。
btb_data_smbist_done	O	0	bist 测试完成信号: 0：表明 BTB data memory 自测试未完成； 1：表明 BTB data memory 自测试完成。
btb_data_smbist_fail	O	0	bist 测试失败信号: 0：表明 BTB data memory 自测试成功，memory 工作正常； 1：表明 BTB data memory 自测试失败。
btb_tag_smbist_done	O	0	bist 测试完成信号: 0:表明 BTB tag memory 自测试未完成； 1：表明 BTB tag memory 自测试完成。
btb_tag_smbist_fail	O	0	bist 测试失败信号: 0：表明 BTB tag memory 自测试成功，memory 工作正常； 1：表明 BTB tag memory 自测试失败。

dcache_da_array_smbist_done	O	0	bist 测试完成信号: 0：表明数据 cache 的 data memory 自测试未完成； 1：表明数据 cache 的 data memory 自测试完成。
dcache_da_array_smbist_fail	O	0	bist 测试失败信号: 0：表明数据 cache 的 data memory 自测试成功，memory 工作正常； 1：表明数据 cache 的 data memory 自测试失败。
dcache_dirty_smbist_done	O	0	bist 测试完成信号: 0：表明数据 cache 的 dirty 位 memory 自测试未完成； 1：表明数据 cache 的 dirty 位 memory 自测试完成。
dcache_dirty_smbist_fail	O	0	bist 测试失败信号: 0：表明数据 cache 的 dirty 位 memory 自测试成功，memory 工作正常； 1：表明数据 cache 的 dirty 位 memory 自测试失败。
dcache_tag_smbist_done	O	0	bist 测试完成信号: 0：表明数据 cache 的 tag memory 自测试未完成； 1：表明数据 cache 的 tag memory 自测试完成。
dcache_tag_smbist_fail	O	0	bist 测试失败信号: 0：表明数据 cache 的 tag memory 自测试成功，memory 工作正常； 1：表明数据 cache 的 tag memory 自测试失败。
icache_data_array_smbist_done	O	0	bist 测试完成信号: 0：表明指令 cache 的 data memory 自测试未完成； 1：表明指令 cache 的 data memory 自测试完成。
icache_data_array_smbist_fail	O	0	bist 测试失败信号: 0：表明指令 cache 的 data memory 自测试成功，memory 工作正常；

			1：表明指令 cache 的 data memory 自测试失败。
icache_tag_smbist_done	O	0	bist 测试完成信号: 0：表明指令 cache 的 tag memory 自测试未完成； 1：表明指令 cache 的 tag memory 自测试完成。
icache_tag_smbist_fail	O	0	bist 测试失败信号: 0：表明指令 cache 的 tag memory 自测试成功，memory 工作正常； 1：表明指令 cache 的 tag memory 自测试失败。
jtlb_da_smbist_done	O	0	bist 测试完成信号: 0：表明 jtlb 的 data memory 自测试未完成； 1：表明表明 jtlb 的 data memory 自测试完成。
jtlb_da_smbist_fail	O	0	bist 测试失败信号: 0：表明表明 jtlb 的 data memory 自测试成功，memory 工作正常； 1：表明表明 jtlb 的 data memory 自测试失败。
jtlb_tg_smbist_done	O	0	bist 测试完成信号: 0：表明 jtlb 的 tag memory 自测试未完成； 1：表明 jtlb 的 tag memory 自测试完成。
jtlb_tg_smbist_fail	O	0	bist 测试失败信号: 0：表明 jtlb 的 tag memory 自测试成功，memory 工作正常； 1：表明 jtlb 的 tag memory 自测试失败。
l0_icache_data_smbist_done	O	0	bist 测试完成信号: 0：表明 l0 指令 cache 的 data memory 自测试未完成； 1：l0 指令 cache 的 data memory 自测试完成。
l0_icache_data_smbist_fail	O	0	bist 测试失败信号: 0：表明 l0 指令 cache 的 data memory 自测试成功，memory 工作正常；

			1：表明 l0 指令 cache 的 data memory 自测试失败。
l0_icache_tag_smbist_done	O	0	bist 测试完成信号: 0：表明 l0 指令 cache 的 tag memory 自测试未完成； 1：表明 l0 指令 cache 的 tag memory 自测试完成。
l0_icache_tag_smbist_fail	O	0	bist 测试失败信号: 0：表明 l0 指令 cache 的 tag memory 自测试成功，memory 工作正常； 1：表明 l0 指令 cache 的 tag memory 自测试失败。
pad_yy_bist_tst_en	I	-	bist 测试使能信号: 不用该信号时，需要接 0。

3.2. 客户自己插入 memory bist

目前很多 EDA 工具都支持插 memory bist，如果客户不用中天的 memory bist，可以用专门的工具直接在 RAM 的端口上插入 memory bist，只需要在综合时将中天的 memory bist 输入信号(pad_yy_bist_tst_en)绑死为 0 并且输出悬空，这样综合工具就会将中天的 memory bist 逻辑给优化掉。

4. 快速前仿真功耗评估

时序和面积的评估都可以通过综合得到，数据可以比较快被获得，但功耗仿数据麻烦一点需要先用综合出的网表进行门仿真，即便此时没有连线延时，memory 等都会出 X，调试起来比较麻烦耗时。所以这里提供一个快速评估功耗的办法，供客户参考。

高端处理器的前仿真功耗绝大部分由 memory 功耗和 clock 构成，而 memory 的 input 和 gated cell（用前端插的 gated cell）的翻转都能由 RTL 仿真波形直接映射过来，非常准确。所以可以利用 RTL 跑出的波形来反标到门级网表进而估计出总功耗。具体可以参考 PT 的用户手册。

需要注明，前端仿真功耗没有连线等信息，功耗数据和最终功耗数据是有较大差别的，只能作为参考，通过一定的比率折算评估出最终的功耗。

5. 门级仿真中不含复位端寄存器初始化

为了节省面积，数据通路上的寄存器通常不含复位端。在门级仿真中为了防止这类寄存

器的初始 X 态传播开来，可以对其进行初始化操作。这里，我们介绍一种修改标准工艺库的方法。不同工艺库下都会有门级仿真中用到的各种 standard cell 的模型描述文件，以 TSMC 为例，我们只需修改 standard cell 中寄存器相关的 cell，如下所示：

```
`celldefine
module DFD1BWP (D, CP, Q, QN);
    input D, CP;
    output Q, QN;
    reg notifier;
    `ifdef NTC
        wire D_d, CP_d;
        pullup (CDN);
        pullup (SDN);
        tsmc_dff (Q_buf, D_d, CP_d, CDN, SDN, notifier);
        initial begin $deposit(Q_buf,1'b0); end
        buf (Q, Q_buf);
        not (QN, Q_buf);
    `else
        pullup (CDN);
        pullup (SDN);
        tsmc_dff (Q_buf, D, CP, CDN, SDN, notifier);
        initial begin $deposit(Q_buf,1'b0); end
        buf (Q, Q_buf);
        not (QN, Q_buf);
    `endif
```

图片 16 寄存器 Q 端赋值示例

对于 DFD1BWP 这个不含置位端和复位端的寄存器来说，可以通过调用 VCS 的 deposit 这个 task 来将 tsmc_dff 的 Q_buf 端口值在仿真初始时刻无条件赋值为 0。因为该操作只会在仿真初始时刻发挥作用，后续仿真过程中若有违例或者 X 态，不会影响其传播，因此可以选择将所有类型的寄存器模型都采用这种方式进行赋值操作。