数字集成电路低功耗物理实现技术与 UPF

孙轶群 *sun.yiqun@nationz.com.cn*
国民技术股份有限公司
Nationz Technologies Inc

摘要

本文从 CMOS 电路功耗原理入手，针对不同工艺尺寸下数字集成电路的低功耗物理实现方法进行描述，并着重描述了 Synopsys UPF（Unified Power Format）对低功耗设计的描述方法。UPF 是 Synopsys 公司提出的一种对芯片中电源域设计进行约束的文件格式。通过与 UPF 格式匹配的 Liberty 文件，UPF 约束文件可以被整套 Galaxy 物理实现平台的任何一个环节直接使用，并将设计者的电源设计约束传递给设计工具，由工具完成设计的实现工作，从而实现整套数字集成电路低功耗物理实现的流程。

1.0 概述

本文从数字集成电路低功耗设计原理下手，对设计中低功耗的实现技术进行描述，包括完成低功耗设计需要的库资料以及常用 EDA 工具对低功耗技术实现的方法。

2.0 CMOS 电路的低功耗设计原理

CMOS 电路功耗主要分 3 种，静态功耗主要与工艺以及电路结构相关，短路电流功耗主要与驱动电压、p-MOS 和 n-MOS 同时打开时产生的最大电流、翻转频率以及上升、下降时间有关，开关电流功耗主要与负载电容、驱动电压、翻转频率有关。做低功耗设计，就必须从这些影响功耗的因素下手。

3.0 低功耗设计手段及 Library 需求

低功耗的设计手段较为复杂，但对于不同的设计，或者不同的工艺，实现的方法却各不相同。

3.1 0.18um 及以上工艺

0.18um 及以上工艺，在低功耗设计手段上较为有限，主要原因在于，静态功耗很小，基本不用关心。
动态功耗方面，主要的功耗来自于 Switching Power，即与负载电容、电压以及工作中的信号翻转频率相关。减小负载电容，就必须在设计上下功夫，减少电路规模。减少信号翻转频率，除了降低时钟频率外，只有在设计上考虑，能不翻转的信号就不翻转。至于电压，由于 0.18um 及以上工艺的阈值电压有一定的限制，因此，供电电压降低，势必影响工作频率。
一般说来，在 0.18um 工艺下设计电路，主要有以下几种对低功耗设计的考虑。

3.1.1 静态功耗可以忽略

根据现有项目经验可知，利用 0.18um 工艺 Standard Cell 设计出来的某芯片，数字逻辑加上 Ram 和 Rom 约 40 万门的电路，在完全静止的状态下，功耗约 200uA 左右（实测数据为 400uA 左右，包括了 50uA Flash，30uA 的 PHY，113uA 的 VR，其他模拟部分漏电不大，因此这里估算为 200uA）。这样的功耗，我们是可以接受的。如果非要减少静态功耗，则可以参照 90nm 工艺的设计思路，专门设计高阈值电压的 MOSFET，或者专门设计切断电源所需的元件，但由此带来设计的复杂性，对 0.18um 工艺的影响还是很大的。如果设计规模没有那么大，且可以满足应用，往往还是可以忽略这个结果的。

3.1.2 时钟门控减小不必要的动态功耗

在寄存器的电路设计中，时钟输入端都会有一个反向器负载，就算输入端不发生变化，时钟的变化也会造成该反向器的变化，由此产生动态功耗。因此在如果该寄存器输入在某种条件下等于输出（即输出保持）时，可以将时钟门控住，以减少无效的时钟翻转。

时钟门控的实现原理如下图所示：

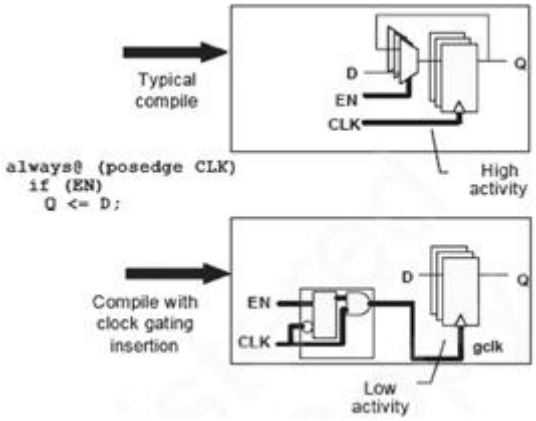

图 3-1 时钟门控原理图

如上图所示，由于现在的设计方式，大多数是同步设计，设计人员只考虑数据路径，时钟往往是不做处理的。因此如果要实现门控，只需要在设计电路时提供可以识别的控制信号，在综合的时候，EDA 工具就可以自动插入时钟门控。

利用 Design Compiler 进行时钟门控单元插入，在读入设计以及时序约束后，需要做以下设置：

1. set_clock_gating_style，设置时钟门控单元插入的约束
2. insert_clock_gating –global，开始插入时钟门控单元；
3. uniquify，将所有时钟门控单元做 uniquify 操作，以便后续 PR
4. hookup_testports -se_port ATPGSE_Pad -se_pin uPad/uATPGSE_Pad/C –verbose，将所有时钟门控单元的 scan_enable 信号与测试用 SE 信号连接起来。如果没有 ATPG，可以不用该句命令。
5. propagate_constraints -gate_clock，将门控单元信息传递给整个电路。
6. report_clock_gating 可以查看时钟门控单元插入的情况，以便做电路修改，或插入时钟门控单元设置的修改。

完成这些设置后，只需要和平常一样做系统综合即可。而在 DC 2008.09 版本以后，第 2~5 的步骤都可以省略，在利用 compile_ultra 进行优化时，第 2、3 步骤都会被自动执行，第 4、5 步骤会在 insert_dft 时被执行。

形式验证工具 Formality，在进行形式验证，需要设置 verification_clock_gatie_hold_mode 为 low、high 或者 any，Formality 就可以识别出时钟门控单元，并与 RTL 进行形式验证了。

3.1.2.1. Clock Gating Cells

这里所说的 Clock Gating Cell 是指专门设计的，集成式时钟门控单元（Integrated Clock Gating Cell，简称 ICG），就是利用 Latch 和与门/或门实现的一个独立的 Standard Cell，其优势在于以硬 IP 实现，时序易于掌握，物理实现中对布局布线有帮助。当然如果单元库中不提供专门的时钟门控单元，EDA 工具也可以利用与门、或门、Latch 甚至是寄存器等进行门控单元的实现，但效果都没有 ICG 好用。这里针对 ICG 的插入进行描述。

图 3-1 中的门控单元是一种典型的，利用负沿使能 Latch 以及与门组成的上升沿有效时钟门控单元，只有时钟下降沿后才会将时钟门控住，保证不产生时钟毛刺。

在 Liberty 格式文件中，某个 Cell，需要有 clock_gating_integrated_cell，才能让 EDA 工具认识到，该 Cell 是一种 ICG。不同的 clock_gating_integrated_cell 的设置，需要在 DC 设置 set_clock_gating_style 时做相应的设置，才可能被使用到，下面列举一些常用的设置：

同时，在 ICG 的不同 Pin 上，必须有以下属性，来告诉 DC 该 Pin 在 ICG 的使用中是什么功能（这里只列举常用的信息）：

clock_gate_enable_pin	该 pin 是时钟使能控制信号
clock_gate_out_pin	该 pin 是时钟输出信号
clock_gate_clock_pin	该 pin 是时钟输入信号
clock_gate_test_pin	该 pin 是 scan_enable 或 test_mode 信号

3.1.3 使用低电压的库进行设计

由于动态功耗中，驱动电压对功耗的影响也相当大，因此，如果能有一套电压只有 1V 的标准单元库，进行设计，仍然可以达到降低动态功耗的目的。但电压的降低，势必引起元件延时的增加，且由于 0.18um 工艺下，阈值电压一般在 0.4V 左右，驱动电压的稳定性需求也相当大，否则，可能会导致致命性的错误。法国的 Dolphin 公司是一家致力于低功耗设计的 IP 提供商，在 TSMC、SMIC 等 Foundry 的 0.18um 工艺下都提供了 1V 的逻辑单元库。下面列出 Dolphin 在 SMIC 0.18um 工艺下设计的一套 1V 逻辑单元库，和 SMIC 0.18um 工艺 Metro 标准单元库进行比较。比较中 Metro 标准单元库使用 1 个门的 BUFX2M，而 Dolphin 使用最小的 Buffer ni01d1 进行比较。

		area (um*um)	Average leakage(nW)	Rise delay (0.04pf,ns)	Typical Rise Energy (0.04pf,pJ)
SMIC18 METRO	BUFX2M	8.7808	0.048	0.182256	0.0266
Dolphin 1V for SMIC 0.18um	ni01d1	10.3488	0.01	0.6	0.005

由比较中可以看出，当电压下降到 1V 后，Rise Energy 下降了 80%以上，除了由于电压下降引起的功耗降低外，Dolphin 应该在电路结构等方面也做了处理，因此不但动态功耗减少了，而且静态功耗也减少了很多。但延时却大了很多，因此如果设计需要翻转的频率不高时，可以考虑利用低功耗的库进行设计，达到降低功耗的目的。如果速度要求很高，这个方法是不可行的。

3.2 90nm 及以下工艺

从 3.1.3 可以知道，降低驱动电压，可以减少动态功耗，但由于电压降低，驱动能力也同时被减弱，因此元件延时较大。为了解决这个问题，工艺尺寸开始减小，以便在减小驱动电压的情况下，增加宽长比（aspect ratio），以达到提高驱动电流的目的，保持元件延时。

同时进入更低尺寸的工艺，氧化层厚度也随之减小，以便减少阈值电压，进一步提高速度。但因为氧化层厚度在减小，漏电电流也变大了。在 90nm 及以下工艺中，漏电电流开始被设计人员关注。

下面对在 90nm 工艺下进行低功耗设计及实现的一些手段结合常用 EDA 工具进行描述。

3.2.1 切断未使能电路的电源减小不必要的静态功耗

针对 SMIC 0.18um 工艺 Metro 标准单元库以及 TSMC 90nmLP 工艺高密度标准单元库（dbtcbn90lphdbwptc）进行比较，以一个门的 Buffer 来举例：

		Average leakage(nW)	Incremental	Typical Rise Energe (0.04pf,pJ)	Incremental
SMIC18 METRO	BUFX2M	0.048	-	0.0266	-
TSMC 90LP Biased Well	BUFFD1BWP	0.214	345.83%	0.00272	-89.77%

可以看出，90nm 工艺下的静态功耗，已经是 0.18um 工艺下功耗的 3.5 倍左右了。根据 3.1.1 可知，利用 0.18um 设计出来的，约 40 万门的电路，静态功耗，大约是 200uA（360uW，0.18um 工艺按 1.8V 供电

电压计算）。如果同样规模的电路，放在 90nm 工艺下，则可能达到 1.26mW 左右，即 1.05mA 左右的静态功耗（90nm 工艺按 1.2V 供电电压计算）。

既然，静态功耗这么大，那么在静止时，怎样才能将这些功耗减小呢？一个非常彻底的方法就是将静止状态电路的电源关断。

为了关断电源，就需要在电源网络和电路之间建立一个电源控制电路，他们被称为电源开关单元（Power Switching Cell），在需要关断时，控制 Power Switching Cell 将电路的供电关闭，否则打开，提供电源。

由于电源关断后的电路，其输出信号就没有电路驱动，对于其驱动的电路来说，就会出现输入浮空的状态。为了解决这个问题，就需要在关闭电源的电路输出端添加一个额外的保持电路，当其电源关闭后保持输出，而电源打开时，保持电路则表现的像一个 Buffer，输出等于输入即可。同时，如果被关闭电源地电路输入固定电压，也可能产生对地的电流，就需要一个特别的单元对该部分电流进行保护。这样的单元被称为隔离单元（Isolation Cell）。一般来说 Isolation Cell 的输出部分有较大的电容负载，也就是说 Isolation Cell 的延时将会比较大，对时序有一定的影响，是需要注意的。

当然，对于寄存器来说，如果断电，则原有的数据就无法保存，重新打开电源后，就一定会出现原有数据丢失的情况。因此可以为一些必须保持数据的寄存器建立一个备份设备，电源关闭前，将寄存器的数值保存到备份设备上，电源打开后从备份设备上将数据重新写入寄存器中。这种备份设备叫做保存寄存器单元（Retention Register Cells）。

对于 Power Switching Cell、Isolation Cell 以及 Retention Register Cell，他们在上电之后是不能关闭的，因此使用的电源也和正常功能不同，这些单元被成为常开逻辑单元（Always-On Logic Cells）

下面就各种不同的 Cell，描述其原理、库中保存的信息、以及实现流程。

3.2.1.1. Power Switching Cells

原理上说，Power Switching Cell 结构如下图所示：

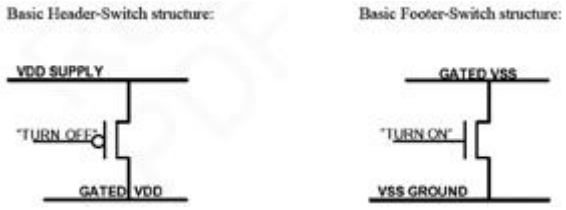

图 3-2 Power Switching Cell 电路结构示意图

从图中可以看到，Power Switching Cell 的设计原理非常简单，VDD 的控制（被称为 Header Switch），利用一个 P-MOSFET 来控制，当 TURN_OFF 信号为高时表示电路关闭，P-MOSFET 关闭，GATED_VDD 就不供电了，而 TURN_OFF 信号为低时表示电路打开，P-MOSFET 打开，GATED_VDD 等于 VDD_SUPPLY。同理，利用一个 N-MOSFET 来控制 VSS 电源是否供电，被称为 Footer Switch。

一般来说，只需要使用 Header Switch 或者 Footer Switch 就可以实现电路关断，其中 Header Switch 结构漏电较小，而 Footer Switch 结构控制效率高，且面积较小。

虽然原理非常容易理解，但电路设计起来非常复杂，需要考虑控制单元对电路供电的能力，考虑不能产生过大的功耗，等等。这里不详细介绍控制单元的设计，详细设计方法可以参看 Synopsys 公司和 ARM 公司联合出版的“Low Power Methodology Manual For System on Chip Design.”。

将 Power Switch Cell 设计成单独的器件，在实现时，控制某一块电路，叫做 Coarse Grain Power Switch Cell。

下面是一个 Coarse Grain Power Switch Cell 的 Liberty 格式描述

```
library(<coarse_grain_library_name>) { #library  描述开始
…
lu_table _template ( template_name ) #电压状态 template 描述，dc_current 组中会使用
variable_1 : input_voltage;
variable_2 : output_voltage;
```

```
index_1 ( <float>, … );
index_2 ( <float>, … );
}
…
cell(<cell_name>) { #某个 Power Switching Cell 描述开始
switch_cell_type : coarse_grain; #Switching Cell 类型是 coarse_grain，暂时只支持该类型
…
pg_pin ( <VDD/VSS pin name> ) { #申明电源和地的 pg_pin 格式
pg_type : primary_power | primary_ground; #他们是主电源和主地
direction : input ; #方向是输入
…
}
/* Virtual power and ground pins use “switch_function” to describe the logic to
shut off the attached design partition */
pg_pin ( <virtual VDD/VSS pin name>) { #申明内部电源和地，这就是输出电源/地的端口
pg_type : internal_power | internal_ground;
direction: output; #方向是输出
…
switch_function : “<function_string>”; #定义开断控制功能，例如 SLEEP
pg_function : “<function_string>”; #内部电源或地功能与输入的 pg_pin 一致，对于 header switch 来说就是 premary_power 的 Pin Name，对于 Footer Switch 来说就是 primary_ground 的 Pin Name
}
dc_current ( <dc_current_name> ) { #定义不同条件下输出 Pin 的稳定电流值，EDA 工具利用该数据计算 IR Drop，并进行 Switch 的优化。
related_switch_pin : <input_pin_name>; #定义控制开断的 Pin
related_pg_pin : <VDD pin name>; #定义可以被控制开断的电源 Pin，如果是 Footer Switch 则是地 Pin
related_internal_pg_pin : <Virtual VDD>; #定义不会被关闭的内部电源 Pin，Footer Switch 则是地 Pin
values(”<float>, …”); #定义不同状态下的该 Cell 输出的电流值
}
pin (SLEEP) { #Pin SLEEP 定义开始，SLEEP 只是举例
direction : input;
switch_pin : true; #表示该输入 Pin 是 switch pin，控制电源/地的开断
…
/* The acknowledge output pin uses “function” to represent the propagated switching signal
*/
pin(<acknowledge_output_pin_name>) { #定义应答输出 Pin 开始，完成开断后，与 switch pin 状态一致，有的 Switch Cell 可能没有该 pin
```

```
…
function : “<function_string>”; #功能定义，应该与 SLEEP 状态一致
power_down_function : “function_string”; #定义关断后电源状态，如对于
Header switch 来说可以是!VDD+VSS，而 Footer Switch 来说可以
是!VSS+VDD
direction : output;
…
} /* end pin group */
} /* end cell group */
```

另外，为了更好的控制电源通断控制，可以专门设计带有电源控制的逻辑单元，实现时，不需要添加额外的控制电路，这种电源开关结构叫做 Fine Grain Power Switch Cell，结构简单，但每个单元都有一个控制器，面积比较大。

下面是一个 Fine Grain Power Switch Cell 的 Liberty 格式描述

```
cell(<cell_name>) { #Fine Grain Power Switch Cell 都是某个 Cell 内的一部
分，不单独出现
is_macro_cell : true; #定义是不是 macro cell
switch_cell_type : coarse_grain | fine_grain; #多数设置为 fine_grain
pg_pin ( <power/ground pin name> ) { #定义电源信号，primary_是可以断开
的，backup_是不会断开的。
pg_type : primary_power | primary_ground | backup_power |
backup_ground;
direction: input | inout | output;
…
}
/* This is a special pg pin that uses “switch_function” to describe the logic to
shut
off the attached design partition */
pg_pin ( <internal power/ground pin name>) { #定义内部电源/地
direction: internal | input | output | inout;
pg_type : internal_power | internal_ground;
switch_function : “<function_string>”;
pg_function : “<function_string>”;
…
}
pin (<input_pin_name>) {
direction : input | inout;
switch_pin : true | false; #如果是 switch pin 就是 true
…
}
…
pin(<output_pin_name>) {
direction : output | inout;
power_down_function : <function_string>;
…
```

```
} /* end pin group */
} /* end cell group */
```

3.2.1.2. Isolation Cells

我们知道，在 CMOS 数字逻辑电路当中，当某根信号为 VDD 时，我们认为是逻辑 1，如果为 GND，则认为是逻辑 0。但当某块电路不供电后，其输出就失去了驱动，输出网络如果没有其他信号驱动，其电平就会为高阻态，换句话说，就是不知道电压是多少。

因此，在低功耗设计中，如果需要切断某块电路电源的话，则该电路的输出就会出现不定值，而与这些输出连接的电路也就出现输入浮空的状态，因此就需要在关断电路的输出及其连接的模块之间加一个 Isolation Cell，在电源关闭时，将输出网络固定在某一个电平上，而打开电源后，该电路的输出等于输入，不影响电路功能。

Isolation Cell 可以利用逻辑门来实现，利用与门实现可以使输出在关闭电源时为 0，被称为 Low Clamped Isolated Signal，利用或门实现可以使输出在关闭电源时为 1，被称为 High Clamped Isolated Signal，如下图所示，X 是输出信号，“ISOLN”和”ISOL”分别是开关控制信号：

图 3-3 逻辑门组成的 Isolation Cell

利用逻辑门组成的 Isolation Cell，会产生延时，对速度很快的电路来说可能会有所影响，因此也可以利用一个 N-MOS 或者一个 P-MOS 以 Pull-Up 和 Pull-Down 电路实现，但这会产生多驱动问题，不是常用的方法。

利用逻辑门组成的 Isolation Cell 是常见的 Cell，下面所示是一个 Isolation Cell 的 Liberty 格式描述：

```
  cell(isolation_cell) {  is_isolation_cell : true ; #定义该 Cell 是 isolation_cell
…
pg_pin(<pg_pin_name_P>) { #电源 Pin 定义
pg_type : primary_power;
…
}
pg_pin(<pg_pin_name_G>) { #地 Pin 定义
pg_type : primary_ground;
…
}
pin (data) {
direction : input;
isolation_cell_data_pin : true ; #被控制的输入 Pin
…
}/* End pin group */
pin (enable) {
isolation_cell_enable_pin : true ; #控制信号
…
}/* End pin group */
…
pin (output) { #输出信号
direction : output;
power_down_function : (!pg_pin_name_P + pg_pin_name_G); #电源关闭后进入为 power down 状态
```

…

}/* End pin group */

}/* End Cell group */

3.2.1.3. Retention Register Cells

对于寄存器来说，当电源关闭后，其保存的数据也就不存在了，如果有些寄存器的数据，希望在断电后继续保持，则需要使用一个专门的 Retention Register Cell，在断电前把寄存器的值锁存起来，重新打开电源后将数值存回寄存器去。而该寄存器在断电时，会有一部分电源一直有电。

下图所示，是一种常用的 Retention Register Cell 基础结构：

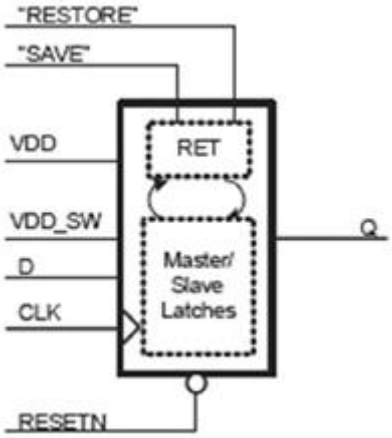

图 3-4 Retention Register Cell 基础结构图

如上图所示，Master/Slave Latches 是常规的寄存器，工作在 VDD_SW 电压域，也就是可以关断的电压域，D、CLK、RESETN 和 Q 是寄存器的控制和输出信号。RET 电路工作在 VDD 电压域，不会被关闭，当“SAVE”信号有效，RET 会把寄存器的值保存起来，而 RESTORE 有效，RET 将保存的数值写入寄存器中（“SAVE”和“RESTORE”具体时序关系需要参考 IP 提供商提供的数据，这里只是说明一下其功能）。

需要注意的是，一个 Retention Register Cell 比普通寄存器面积要大约 20%，如果设计的鲁棒性好一些，甚至会大超过 50%的面积。

Retention Register Cell 不但是可以保持寄存器的值，还可以保持 Latch 的值，两种不同的结构图如下所示：

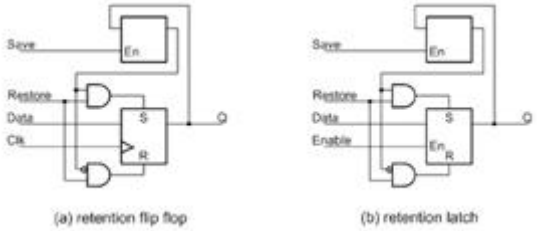

图 3-5 Retention Cell 的电路结构示意图

图中(a)表示寄存器的 Retention Cell，当 Save 和 Restore 都是 0 的时候，使能保持电路不工作，而寄存器的置位和复位端无效，寄存器正常工作。需要进入 Sleep 模式，首先需要停止寄存器的 Clk（Clk_On 信号关闭），接着准备进入 Sleep 模式。当 Save 为 1 后，Q 端数据被采集进保持电路中，然后可以关闭电源（Power_on 关闭）。打开电源后（Power_on 打开），将 Restore 置 1，如果保持电路输出为 1，则对寄存器做异步置位，否则做异步复位，使寄存器输出与保持电路一致。需要注意的是，Save 为从 0 置 1 后，不可以再有时钟改变 Q 端输出，且 Save 必须有一定的宽度，保证 Q 端数据被记录下来，同时关闭电源一

定要等数据被保持下来后才可以进行。打开电源后，Restore 一定要保持一定的时间，使寄存器充分复位或置位，同时在下一个时钟沿到来之前，一定要将 Restore 清 0，否则 Q 端就不会发生变化，电路可能就会出错。当数据被读回后，就可以打开时钟（Clk_On 打开），开始正常工作了。时序示意图如下所示：

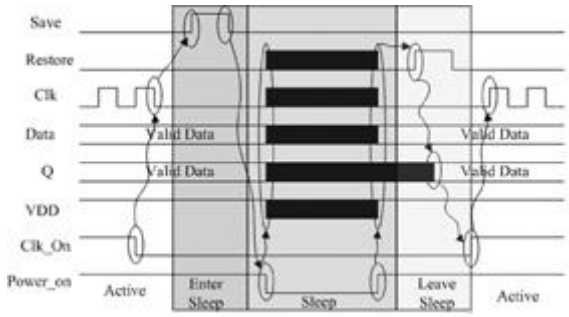

图 3-6 Retention Register 工作时序示意图

图中(b)是一个 Retention Latch 的电路结构示意图，原理与寄存器的 Retention Cell 类似，这里就不详细描述了。

下面所示是一个 Retention Cell 的 Liberty 格式描述，描述是基于一个常见寄存器或 Latch 的格式上进行的：

```
cell(<cell_name>) {
retention_cell : <retention_cell_style>;
pin(<pin_name>) {
retention_pin(<pin_class>, <disable_value>);
…
}
…
}
```

retention_cell_style 主要是定义一个器件识别名，不同功能名字不能一样。这里所说的功能，不止是寄存器或 latch 的功能，还需要考虑到 save 和 restore 过程是否一致。

retention_pin 是的定义表示该 pin 是与 retention cell 功能有关的，其中 pin_class 可以是 restore、save、save_restore，其中 restore 和 save 分表表示该 pin 是 restore 功能和 save 功能，而 save_restore 表示该 pin 根据不同电平为 save 或 restore 功能。disable_value 表示 save 或者 restore 无效时的电平是 1 还是 0。

需要注意的是 retention_pin 定义时一定要有 related_power_pin 和 related_ground_pin 定义其所述的电源域，否则编译时会报 Warning，并为该 Pin 定义 primary_电源域。

同时对于 latch 和寄存器的 Retention Cell 的 latch 和 ff 组描述，会与正常的 latch 和寄存器有所不同，如下，加粗字体为不同的部分：

```
latch (IQ, IQN) {
data_in: D & (SAVE & RESTORE) ;
enable : CLK ;
…
}
```

```
ff("IQ", "IQN") {
next_state : "D & (!SAVE & !RESTORE)" ;
clocked_on : "CP" ;
}
```

3.2.1.4. Always-On Logic Cells

有些 Cell 是不能够被关闭的，如 Power Switch Cell、Retention Register Cells、Isolation Cells，他们就被称为 Always-On Logic Cells。在这些 Cell 的 Liberty 格式描述中就会有一个属性“always-on”是 true。同时对于 Always-On Logic Cells，pg_pin 描述一般都会有两组，primary 和 backup，工具看到该 cell 为 Always On，就会把 2 组电源地都接到长开的电源/地上。

```
cell(always_on_cell) {
always_on : true ; #定义该 Cell 是 always_on_cell
…
pg_pin(<pg_pin_name_P>) { #电源 Pin 定义
pg_type : primary_power;
…
}
pg_pin(<pg_pin_name_G>) { #地 Pin 定义
pg_type : primary_ground;
…
}
pg_pin(<pg_pin_name_P>) { #电源 Pin 定义
pg_type : backup_power;
…
}
pg_pin(<pg_pin_name_G>) { #地 Pin 定义
pg_type : backup_ground;
…
}
…
}/* End Cell group */
```

3.2.2 利用不同 VT 值的库，实现静态功耗和时序的平衡（Multi-VT）

随着工艺尺寸的减小，Oxide 层厚度的减少，VT 值也一直在减少，这是为了在驱动电压较低的情况下，提高驱动速度。但同时静态功耗也随之增加。

对电路设计来说，并不是每一个部分的电路，路径延时都需要那么小，对于一些非关键路径来说，如果能够使用高 VT 值的元件，则可以在满足时序的前提下减小静态功耗了。

在同一种工艺下，实现不同的 VT 值，可以使用井偏置（Well Bias）技术，使 Substrate 的电压与 Source 的电压存在一定的电压差，就可以改变 VT 值了。使用较多的方法是分别对 N-MOSFET 和 P-MOSFET 增加 1 层 Mask 来提高 VT，或减小 VT。

通常情况下 IP 提供商会提供多套不同的单元库，按照不同的 VT 值进行设计。如 TSMC 90nm LP 工艺的单元库，就会提供普通 VT、High VT、Low VT 以及 Ultra Low VT 四套单元库。下面 TSMC 90nm LP 的四套单元库进行分析：

area
(um*um) Average leakage(nW) Rise delay

(0.04pf,ns) Typical Rise Energe
(0.04pf,pJ) dbtcbn90lphdbwphvttc BUFFD1BWPHVT 2.1952 0.017 0.2444 0.002716 dbtcbn90lphdbwptc BUFFD1BWP 2.1952 0.214 0.237176 0.00272 dbtcbn90lphdbwplvttc BUFFD1BWPLVT 2.1952 0.412 0.217184 0.002616 dbtcbn90lphdbwpulvttc BUFFD1BWPULVT 2.1952 5.055 0.193556 0.003624

表格中 dbtcbn90lphdbwptc 是正常 VT 的库，dbtcbn90lphdbwphvttc 是 High VT 的库，dbtcbn90lphdbwplvttc 是 Low VT 的库，dbtcbn90lphdbwpulvttc 是 Ultra Low VT 的库。分析时使用 1 个门的 Buffer 来进行，面积上都是 2.195um^2。

静态漏电功耗，High VT 库中，1 个门只有 0.017nW，甚至比 SMIC18 METRO 1 个门的漏电功耗还低（参看 3.2.1）。随着 VT 的减小，静态功耗逐渐变大，到 Ultra Low VT 时，其静态功耗，1 个门就有 5.055nW。但也因为 VT 很高，High VT 的 Buffer 延时很慢，比 Ultra Low VT 减慢了约 27%。

从表格看来，延时的变化并不非常明显，如果在时序上要求不是很高，则尽量还是需要使用 High VT 的库来实现

3.2.2.1. 常用 EDA 工具中 Multi-VT 的实现方法

从 Synopsys Multi-VT 实现过程主要是在逻辑综合（Logic Synthesis）阶段。DC 完成 Multi-VT 的实现，主要是在 target_library 中找出可以使用的所有逻辑单元，并在满足时序约束的情况下，使用最低 leakage power 的单元进行实现。

其实现步骤可以如下：

```
#读入不同 VT 的逻辑单元作为 target library，当 DC 有充分的选择空间
set_target_library {dbtcbn90lphdbwptc.db dbtcbn90lphdbwphvttc.db \
dbtcbn90lphdbwplvttc.db dbtcbn90lphdbwpulvttc.db}
#读入 HDL 代码
read_verilog design_include.v
#link design
current_design design_top
uniquify
link
#读入约束
source design_constraint.tcl
#其他设置
……
#设置最大漏电电流，该设置必须有，否则优化过程不考虑漏电的优化
set_max_leakage_power 0 mw
#开始编译
compile_ultra
```

由于综合时，时序信息并不完全准确，特别是 setup 类的时序。因此可以对 setup 时序做稍紧一些的约束，使结果进入 PR 工具后还能够满足时序，如果进入 PR 后仍然无法满足 setup 时序，PR 工具也可以利用类似的原理进行优化。

如在 IC Compiler 里，在布线后优化时可以使用如下语句来进行优化：

physopt –preserve_footprint –only_power_recovery –post_route –incremental

这里设置了-preserve_footprint 以及-only_power_recovery，因此只是针对相同 footprint 的单元做漏电功耗优化，也就是说如果某个使用了 High VT X2 的 Buffer 需要减小延时，则可以替换成 Low VT X2 的 Buffer，这样做可以保持原有的布线结果。

当出现 Multi-VT 的单元库后，每个单元库都会多一个 default_threshold_voltage_group 以及 threshold_voltage_group 属性来说明该单元库是 High VT、标准 VT 还是 Low VT。可以利用 report_threshold_voltage_group（DC 和 IC Compiler 都支持）指令报出设计中每种单元库单元占整个设计的百分比，如 High VT 的单元库占 60%，标准 VT 占 25%，等。这样做可以使设计者了解不同 VT 对自己设计的影响，如果速度不快，则 Low VT 的库可能占用很少，甚至没有使用，那么完全可以在设计过程中直接不使用该库。

如果库中没有 default_threshold_voltage_group 变量，我们可以在 DC 中设置：

set_attr -type string dbtcbn90lphdbwphvtwc.db: dbtcbn90lphdbwphvtwc default_threshold_voltage_group HVT

3.2.3 时钟门控减小不必要的动态功耗

时钟门控单元的插入在 3.1.2 已经有所描述，这里就不多进行描述了。同样的，标准单元库必须提供相应 Cell 的库才可以实现。

3.2.4 多供电电压，实现动态功耗与时序的平衡（Multi-Voltage）

我们知道，降低驱动电压 VDD，是减小动态功耗最快的方法，因此在满足时序的情况下，适当降低驱动电压，可以有效的减小动态功耗。而设计中可以使用多驱动电压的设计方法，对于速度要求快的电路，供高一些的驱动电压，如 1.3V，而速度要求不高的模块，则只需要供比较低的驱动电压，如 1.0V。

对于逻辑综合来说，DC 中，首先需要对不同电压域的电路设置不同的 operating_condition，综合工具就可以对该电压域电路进行初步分析和优化了。如果使用 UPF，则可以直接使用 load_upf，工具会根据 UPF 的描述自动寻找相应的库文件进行分析。如下所示：

```
set target_library “slow_14V slow 10V
……
read_verilog design_include.v
current_design design_top
link
#set_operating_condition –max slow_10V –max_library slow_10V
#uDesign/uDMA 速度快，需要使用 1.4V 供电
#set_operating_condition –max slow_14V –max_library slow_14V
–object_list \
#uDesign/uDMA
load_upf power.upf
……
check_mv_design
compile –scan
check_mv_design
……
```

这里 check_mv_design 主要是检查 UPF 对设计的描述是否正确，在 compile 之后再做一次是为了查看 compile 后，电路结构域 UPF 的描述是否正确。

接着，需要在不同电压电路之间，添加 Level_shifters，进行电压转换。如下所示：

check_level_shiftersinsert_level_shifters –all_clock_nets -verbose

DC 会根据不同电压域设置 operating condition 中 voltage 的数值或者 UPF 中的描述，查找 Level Shifter Cell 中 input_voltage_range 和 output_voltage_range 满足这些电压域需求的元件，在 2 个电压域的数据交互信号上添加 Level Shifter Cells。

3.2.4.1. 常用 EDA 工具中 Multi-VT 的实现方法

Level Shifters 主要是在多供电电压设计中，在 2 个不同电压域之间进行电压转换的器件，将某个电压域输出的逻辑电平转换成另外一个电压域可以识别的逻辑电平。

从功能上来看，就像一个 Buffer。

下面描述的是 level_shifter 的 Liberty 格式：

```
cell(level_shifter) {
is_level_shifter : true ; #定义为 true，则该 Cell 被认为是 lever shifter cell
level_shifter_type : HL | LH | HL_LH ; #定义电平转换方向，HL 表示高电平转低电平，LH 直低电平转高电平，HL_LH 表示都可以用
input_voltage_range (<float>, <float>); #输入电压范围，指该 Cell 输入连接的电路，电压工作范围，EDA 工具会根据 Operating Condition 的选择，检查该 Cell 是否满足电路需要，可以在功能 pin 中进行定义，如果在 cell 主体定义，则必须与 output_voltage_range 同时存在
output_voltage_range (<float>, <float>);#输出电压范围，指该 Cell 输出连接的电路，电压工作范围，EDA 工具会根据 Operating Condition 的选择，检查该 Cell 是否满足电路需要，可以在功能 pin 中进行定义，如果在 cell 主体定义，则必须与 input_voltage_range 同时存在
…
pg_pin(<pg_pin_name_P>) {
pg_type : primary_power;
std_cell_main_rail : true; #该 primary_power 连接在 Cell 设计中的主 rail
…
}
pg_pin(<pg_pin_name_G>) {
pg_type : primary_ground;
…
}
pin (data) {
direction : input;
input_signal_level : “<voltage_rail_name>”; #输入信号电压环名
input_voltage_range ( <float> , <float>);
level_shifter_data_pin : true ; #数据功能 Pin
…
}/* End pin group */
pin (enable) {
direction : input;
input_voltage_range ( <float> , <float>);
level_shifter_enable_pin : true ; #使能 Pin，如果 level shifter 还作为 isolation cell 的时候，使能信号会在电源关闭时停止 level shifter 功能，仅仅作为 isolation cell 存在
…
```

```
}/* End pin group */
pin (output) {
direction : output;
output_voltage_range ( <float> , <float>);
power_down_function : (!pg_pin_name_P + pg_pin_name_G);
…
}/* End pin group */
…
}/* End Cell group */
```

3.2.4.2. Liberty PG Pin 格式

Liberty PG Pin 格式，就是在传统的 Liberty 格式文件中，加上与 Power 有关的信息。在工艺尺寸较大的时候，逻辑单元基本上只工作在一个电源系统中，但在使用多电压设计的小工艺尺寸库文件中，则需要告诉分析工具，该 Cell 的电源和地是接在哪里的，每个 pin 所处的电压域在哪里？（实际上 IO 设计中应该已经有类似的数据，但因为不涉及到多电压设计的方法，在传统的 IO Liberty 中也很少见到 PG Pin 的格式）下面所示是 PG Pin 格式的描述

```
library(slow) { #library “slow”描述开始
…
voltage_map(vdd, 1.2); #  有一个 1.2V 的驱动电压，名为“vdd”
voltage_map(vss, 0.0); #  有一个 0V 的驱动电压，名为“vss”
…
operating_conditions(slow_12V) { #Operating Condition “slow_12V”定义开始
…
voltage : 1.2; #使用 1.2V 作为该 condition 的电压条件
…
} #Operating Condition 定义结束
…
default_operating_conditions : slow_12V; #default 使用”slow_12V”
cell(BUFFX1) { # BUFFX1 Cell 描述开始
pg_pin (“VDD”) { # BUFFX1 有一个 Pin “VDD”，定义为 pg_pin
voltage_name : “vdd”; # Pin “VDD”电压是接在 voltage_map ”vdd”上的，即 1.2V
pg_type : “primary_power” # Pin “VDD”是”primary_power”类型的
}
pg_pin (“VSS”) { # BUFFX1 有一个 Pin “VSS”，定义为 pg_pin
voltage_name : “vss”; # Pin “VSS”电压是接在 voltage_map ”vss”上的，即 0V
pg_type : “primary_ground” # Pin “VSS”是”primary_ground”类型的
}
…
leakage_power() { #leakage_power 定义开始
related_pg_pin : VDD; # VDD Pin 供电时的 leakage power
…
}#leakage_power 定义结束
```

```
…
pin (A) { #有一个功能 Pin ”A”
direction : input ; #”A”是一个输入
related_power_pin : VDD; #”A”是 VDD 供电电压域的
related_ground_pin : VSS; #”A”是 VSS 供地的电压域信号
…
}# “A”申明结束
…
pin (Y) { #有一个功能 Pin “Y”
direction : output; #”Y”是一个输出
power_down_function : (!VDD + VSS) ; #当 VDD 关断，VSS 开着的时候，
输出被关断
related_power_pin : VDD; #”Y”是 VDD 供电电压域的
related_ground_pin : VSS; #”Y”是 VSS 供地的电压域信号
internal_power() { #internal_power 申明开始
related_pg_pin : VDD; #internal power 由 VDD 供电引起
…
} /* end internal_power group */
…
}/* end pin group*/
…
}/* end cell group*/
…
}/* end library group*/
```

在 PG Pin 格式中，有一个 pg_type 的属性，下表描述该属性的含义：

pg_type	简单描述	具体描述
primary_power	主电源	一般功能信号使用的电源，可以关断
primary_ground	主地	一般功能信号使用的地，可以关断
backup_power	备用电源	Always-on 的电源，主要用于 always-on logic cell 的常开电源
backup_ground	备用地	Always-on 的地，主要用于 always-on logic cell 的常开地
internal_power	内部电源	主要用于 Power Switch cell 的内部电源
internal_ground	内部地	主要用于 Power Switch cell 的内部地

3.2.4.3. 电压和频率的 Scaling 分析

有的时候，IP 提供商可能无法提供足够的单元库，例如提供商可能提供了 1.4V 的单元库和 1.0V 的单元库，但我们希望提供一个 1.2V 的电压，则需要进行电压的 Scaling。

首先我们需要建立一个独立的 operating_conditions，基于 slow_14V 的库，电压是 1.2V：

create_operating_conditions –name slow_12V –library slow_14V –process 1 –voltage 1.2 –temperature 125

然后对于需要使用 1.2V 的子电路，需要设置 operating_condition 为 slow_12V。分析和优化工具，会首先根据 slow_14V 提供的 model 计算出在 1.4V 驱动下，延时、功耗、电容等信息，然后利用 slow_14V 提供的 k_factor 进行 scaling，计算出 1.2V 条件下的数值。

利用这个方法可以在 IP 提供商没有提供所需分析环境的情况下，利用已有模型估计出在其他环境中出现的情况。但需要注意的是，当条件相差很大时，Scaling 的结果偏差会很大，建议不要使用这种方式进行 sign-off 工作。

至于频率 Scaling，则是在应用过程中可以减少频率，以减少翻转率，达到低功耗应用的需求。

Scaling 的使用最好不要超过标准单元库的 10%，否则结果完全不可行，同时如果采用 CCS 格式的电流源模型 Liberty，则 Scaling 结果会比普通的 NLDM 格式 Liberty 更精确一些。

3.3 其他未提及工艺

实际上，上述提到的 2 种工艺范围，甚至是现在已有的所有工艺技术，从本质上想进行低功耗设计，其思路都是一致的，方法无外乎关断电源、低压驱动（多电压混合设计）、Multi-VT 等方法，只是当工艺尺寸下降到一定程度后，IP 提供商能提供足够的数据，使 EDA 工具直接使用，减少工程师重复劳动的工作量。所以低功耗设计，首先应该从系统设计下手，在一个设计中，应该使用怎样的低功耗策略，采用那种工艺尺寸，有没有合适的单元库，如果没有库，是否自己可以设计，这些都必须首先考虑清楚。而工具，只是为了帮助我们实现自己的想法。

4.0 一个低功耗设计实现的例子

这里提供一个 DEMO（pl8051_extend_chip），可以使读者更快的理解低功耗设计的基础。下图所示，是该设计的功能结构图：

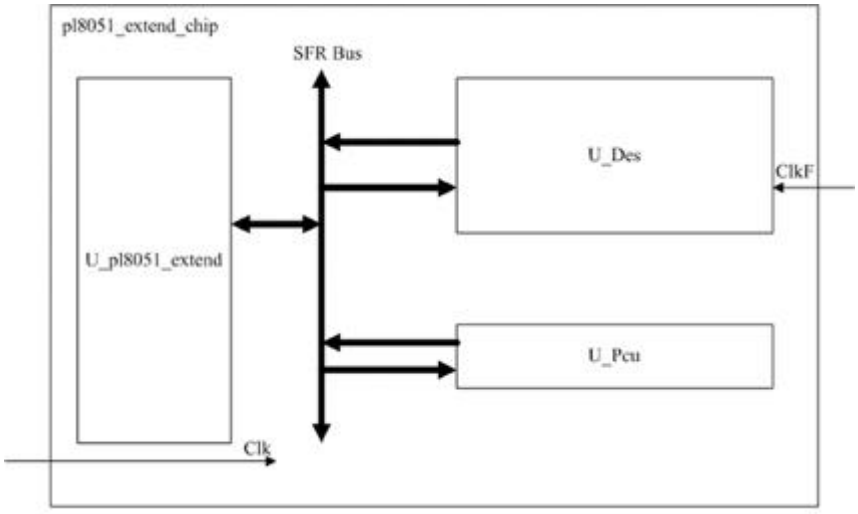

图 4-1 pl8051_extend_chip 功能结构图

由图 4-1 可以看出，整个设计主要是由一个 8051 控制器，通过 SFR 总线对其他模块进行控制，U_Des 是一个算法模块，U_Pcu 是功耗控制单元（Power Control Unit）。整个设计都在 Clk 的控制下进行工作，只有 U_Des 工作在 ClkF 下。

这里假设 Clk（28ns）是 ClkF（7ns）的 4 分频时钟，且与 ClkF 同源同向，这样才可以保证 U_Des 控制的正确性。

由于 U_Des 需要在较快的频率下工作，因此需要对 U_Des 进行一些特殊的设计，也就是对整个设计进行低功耗设计，如下所示：

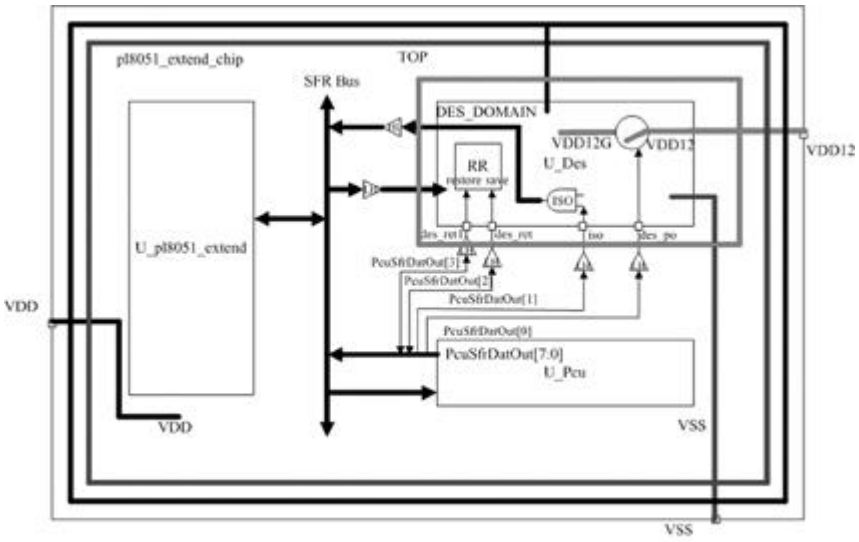

图 4-2 低功耗设计方案示例

上图主要描述以下信息：

1. 2 个电压域，TOP 以及 DES_DOMAIN。
2. 电源设计外部提供，分 1.2V 的 VDD12 和 1.0V 的 VDD10
3. 设计中除 DES 电路几乎所有电路工作在 1.0V 的 VDD 电压域。
4. U_Des 由于工作速度要求较高，工作在 1.2V 的 VDD12 电压域 DES_DOMAIN，但由于不使用的时候需要关断，以降低静态功耗，因此，通过一组 PowerSwitch 进行控制，控制信号来源为 U_Pcu 的 PcuSfrDatOut[0]输出。该电压域与 TOP 电压域共用 VSS 地信号。
5. 由于 U_Des 电路处于 VDD12 电压域，而 U_Pcu 处于 VDD 电压域，因此需要添加 Level Shifter 进行电平转换。
6. U_Des 电路输出信号，需要通过一个 Isolation Cell，在关断电源时提供稳定电平，而该电平为 1.2V，因此还需要利用 Level Shifter 转换成 1.0V 电压域信号，Isolation Cell 的 Enable 信号来自于 U_Pcu 模块的 PcuSfrDatOut[1]信号。
7. U_Des 电路中的寄存器需要使用 Retention Register，save 和 restore 控制信号分别来自于 U_Pcu 输出 PcuSfrDatOut[2]和 PcuSfrDatOut[3]经过 Level Shifter 产生的信号。

Synopsys 公司为低功耗设计提供了一整套的解决方案，即 UPF 设计流程。UPF（Unify Power Format）是一种描述功耗设计思想的文件，Synopsys Galaxy Implementation Platform 只需要读取 UPF 文件，就可以将低功耗设计思路实现，而 Synopsys Discovery Verification Platform 读取 UPF 后，可以对低功耗思路以及最终实现的电路进行验证。

接下来，我们利用 UPF 描述该 DEMO 的功耗设计思路。

4.1 申明电压域以及虚拟电压接口

首先申明电压域，并申明 VDD、VDD12 以及 VSS 等电压端口：

```
create_power_domain TOP
create_power_domain DES_DOMAIN_domain –elements U_Des
create_supply_port VDD
create_supply_port VDD12
create_supply_port VSS
```

根据上述指令，工具会识别出如下图所示中画圈的信息，电压域及虚拟电压端口开始出现：

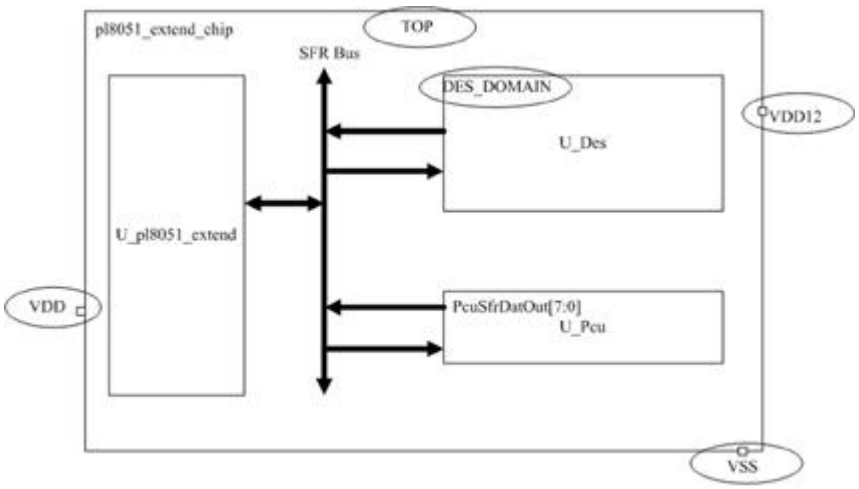

图 4-3 电压域及虚拟电压端口信息

需要注意的是电压端口只不过是虚拟存在的，并不是一定在顶层连接。

4.2 申明电源网络

接着申明每个电压域中的电源网络，建议从顶层往底层进行申明，如首先申明 TOP 的 VSS，VDD。：

```
create_supply_net VDD
create_supply_net VSS
```

以上语句添加下图中画圈部分：

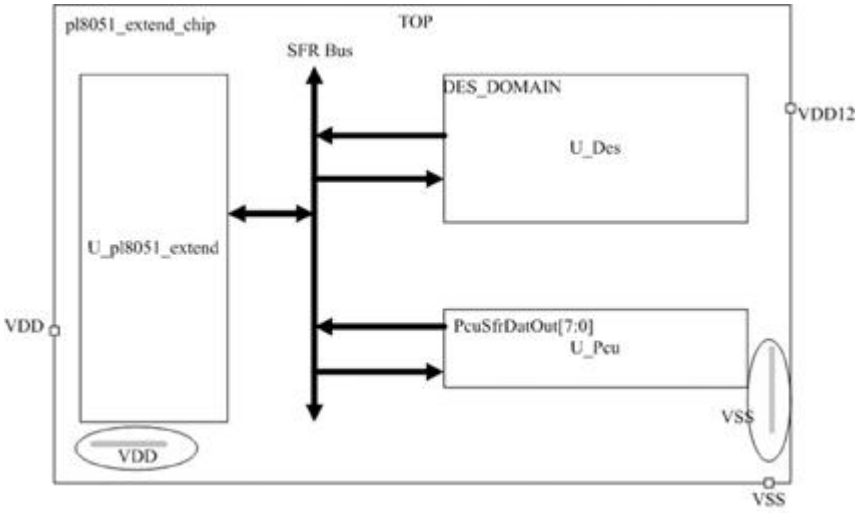

图 4-4 产生顶层电源网络

接着申明 DES_DOMAIN 的电源网络，但需要注意的是，如果需要插入 Level Shifter，最好是在与其有连接关系的电源域（TOP）同时申请电源线，并连接起来：

```
create_supply_net VDD12
create_supply_net VDD12 –domain DES_DOMAIN –reuse #DES_DOMAIN
的 VDD12 与 TOP 的 VSS 共用一个网络，因此需要-reuse
create_supply_net VDD12G
create_supply_net VDD12G –domain DES_DOMAIN -reuse
create_supply_net VSS –domain DES_DOMAIN –reuse
create_supply_net VDD –domain DES_DOMAIN –reuse
```

以上语句添加下图中画圈部分：

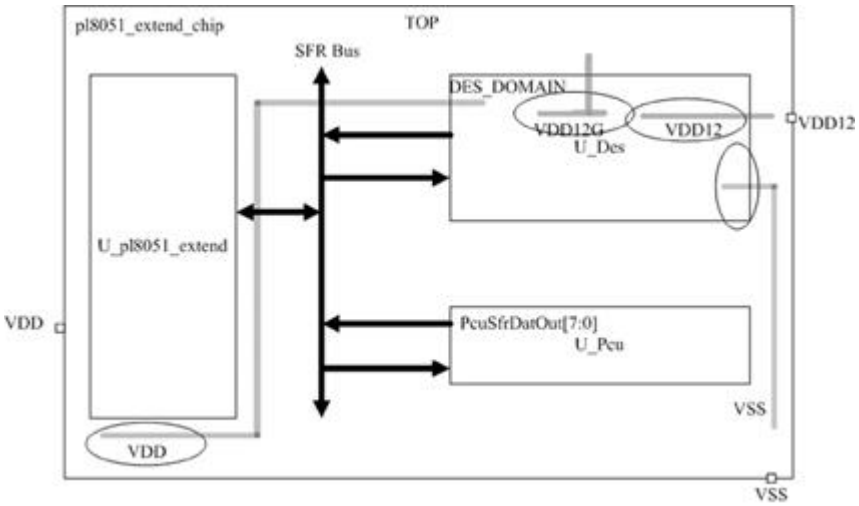

图 4-5 子电路的电源网路申明

然后将电源网络和电源端口连接起来：

```
connect_supply_net VDD12 –port {VDD12}
connect_supply_net VDD –port {VDD}
connect_supply_net VSS–port {VSS}
```

以上语句添加下图中画圈部分：

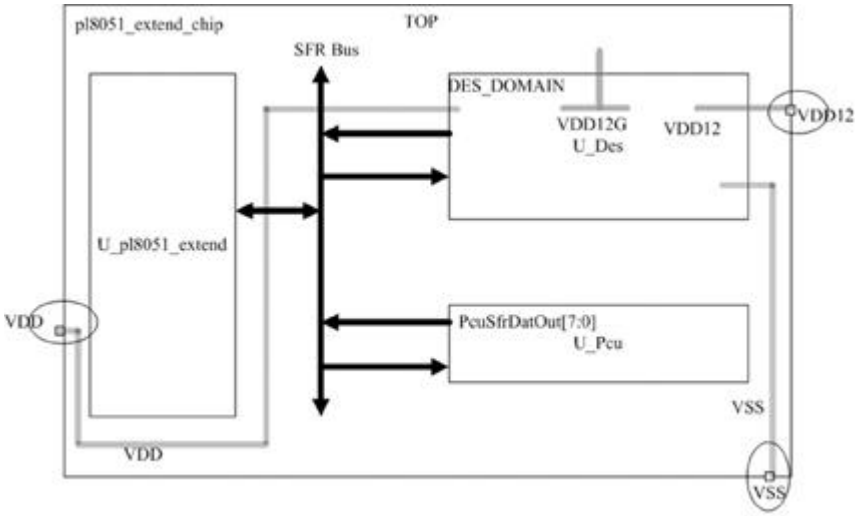

图 4-6 连接电源网络和电源端口

最后为每个电源域申明主电源网络来源，主电源就是该电压域普通逻辑工作使用的电源：

```
set_domain_supply_net TOP –primary_power_net VDD
–primary_ground_net VSS
set_domain_supply_net DES_DOMAIN –primary_power_net VDD12G
–primary_ground_net VSS
```

做完这一步，电源网络申明基本完成

4.3 添加 Power Switch

这添加 Power Switch 主要是 2 步，第一步创建 Power Switch，第二步 map Power Switch。可以参看以下脚本进行：

```
# Power Switch
```

```
create_power_switch des_sw \
    -domain DES_DOMAIN \
    -input_supply_port {in VDD12} \
    -output_supply_port {out VDD12G} \
    -control_port {des_po U_Pcu/PcuSfrDatOut[0]} \
    -on_state {state_on in {!des_po}} \
    -off_state {OFF {des_po}}

map_power_switch des_sw \
    -domain DES_DOMAIN \
    -lib_cell HDRSIHVTD0
```

create_power_switch 和 map_power_switch 在综合时只会检查语法，不会有实际效果，但在综合时将该 UPF 指令读入系统，可以由 DC 输出相应的 UPF 给 Synopsys 的 PR 工具，PR 工具可以自动识别这些指令，到时才会真正添加 Power Switch。2 句命令的结果示意在下图画圈中：

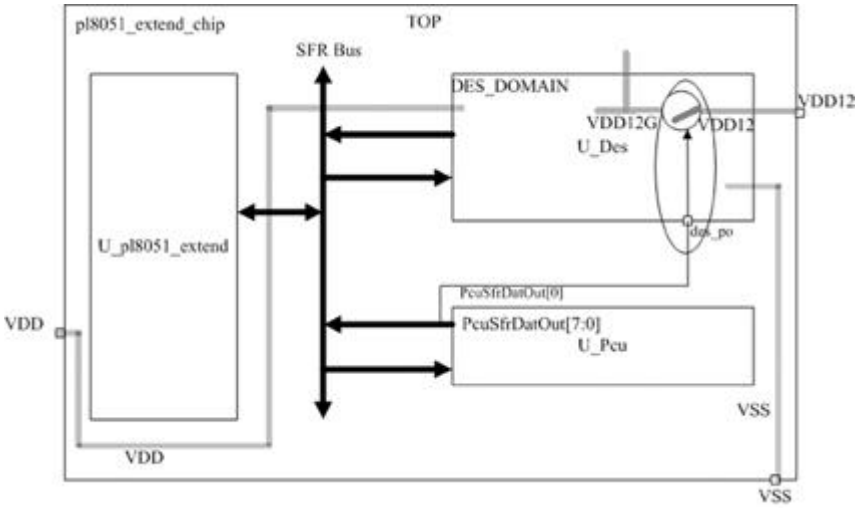

图 4-7 添加 Power Switch Cell

4.4 建立电源状态表格（Power State Table, PST）

在 Power Switch 插入以后，需要对电压域的有效电压，以及不同电压域之间的关系进行设置，同时建立电压分布及关断的情景（scenarios），可以用于静态电压分析、仿真等。

建立电源状态表格，可以参看以下脚本：

```
add_port_state VDD –state {LV 1.0} #VDD 是 1.0V，属于低电压域（LV）状态
add_port_state VDD12 –state {HV 1.2} #VDD12 是 1.2V，属于高电压域（HV）状态
add_port_state des_sw/out –state {HV 1.2} \  #Power Switch 输出是 1.2V，属于高电压域（HV）状态
                -state {OFF off} #关闭状态（OFF），无电压（off）
add_port_state VSS –state {ON 0.0} #VSS 处于常开状态（ON），电压 0V
create_pst design_pst –supplies {VDD VDD12 VDD12G} #建立一个 PST，由 3 个电源状态不同组成
add_pst_state design_work –pst design_pst –state {LV HV HV} #design_work 状态下，三组电源都开
add_pst_state stand_by –pst top_pst –state {LV HV OFF} #stand_by 状态下，
```

des_sw/out 输出电源为关闭状态

需要注意 add_pst_state 的-state 括号里的描述顺序，必须与 create_pst 的-supplies 里一致，选择的状态必须与 add_port_state 的-state 里描述的一致才可以。

这样就建立了下表所示的 PST，design_pst：

	VDD	VDD12	VDD12G
design_work	LV	HV	HV
stand_by	LV	HV	OF

由此建立了 PST，分析工具或仿真工具就知道需要对哪些状态进行分析了。

4.5 插入 Isolation Cell

下面需要为 DES_DOMAIN 输出给 TOP 的输出信号添加 Isolation Cell。脚本如下所示

```
# Isolation Cell
set_isolation des_iso_out \
    -domain DES_DOMAIN \
    -isolation_power_net VDD12 -isolation_ground_net VSS \ #隔离后只有
VDD12 供电
    -clamp_value 1 \ #关电后输出逻辑 1
    -applies_to outputs #对于所有输出添加

set_isolation_control des_iso_out \
    -domain DES_DOMAIN \
    -isolation_signal U_Pcu/PcuSfrDatOut[1] \
    -isolation_sense high \ #控制信号为高时 ISO 有效
-location self # Isolation Cell 添加在 DES_DOMAIN 里面
```

这样就可以产生如下图所示画圈部分电路：

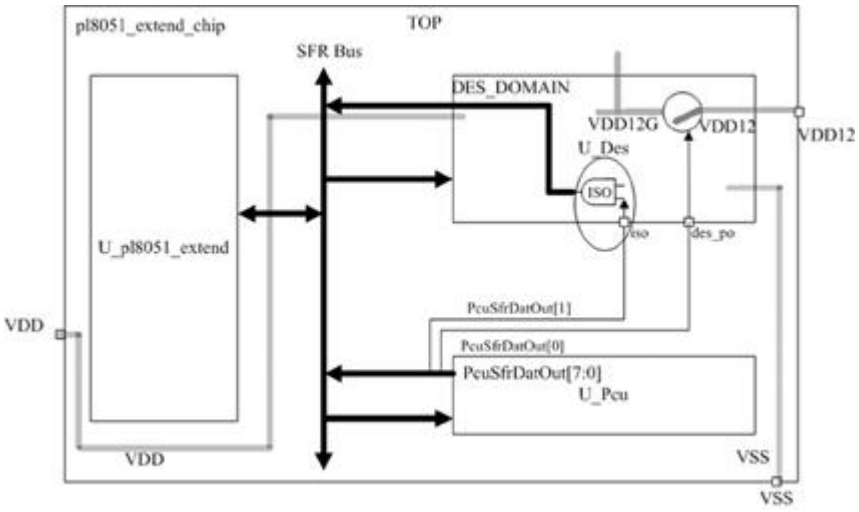

图 4-8 插入 Isolation Cell

4.6 替换 Retention Register Cell

接着需要使用 Retention Register Cell 将 DES_DOMAIN 中在关断电源后需要保留数据的寄存器替换为 Retention Register Cell，可以使用如下脚本：

```
# Retention Register
set_retention des_ret -domain DES_DOMAIN \
    -retention_power_net VDD12G -retention_ground_net VSS
#retention_power_net 是关电的 Power，表示需要保持 VDD12G 电源域的数据

set_retention_control des_ret -domain DES_DOMAIN \
    -save_signal {U_Pcu/PcuSfrDatOut[2] high} \
    -restore_signal {U_Pcu/PcuSfrDatOut[3] low}

map_retention_cell des_ret \
    -domain DES_DOMAIN \
-lib_cell_type RSDFCSD1 #RSDFCSD1 是带有异步置位的 Retention Register
Type，可以从 lib 文件中 retention_cell 属性中看到。
```

根据以上描述，会按照下图画圈示意图描述电路：

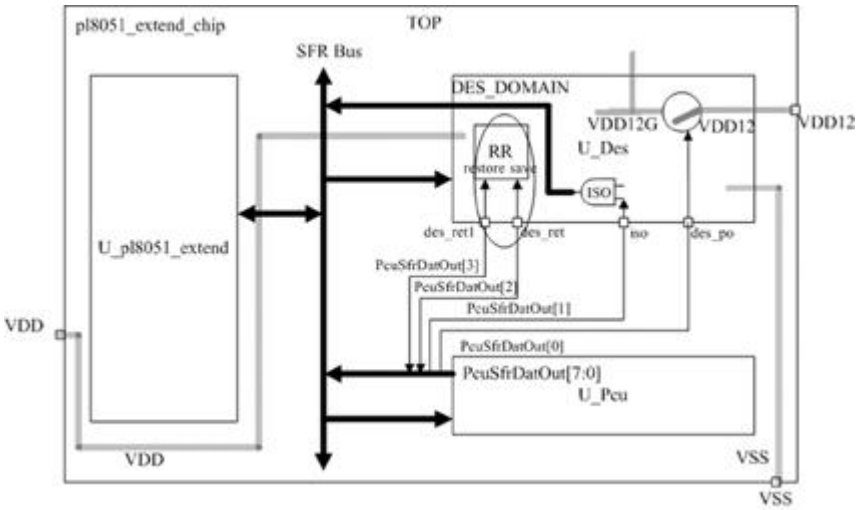

图 4-9 替换 Retention Register Cell

4.7 插入 Level Shifters

插入 Level Shifters 是根据 PST 的定义进行，根据 PST 定义看来，DES_DOMAIN 和 TOP 之间需要插入 Level Shifters，其中 DES_DOMAIN 输出的 pin（Isolation Cell 的输出）需要插入 H->L 的 Level Shifter，而其输入则需要插入 L->H 的 Level Shifter。

插入 Level Shifter 在 Compile 过程中自动进行，也可以利用 UPF 命令 set_level_shifter 定义插入的规则。

插入 Level Shifter 后产生的电路变化示意图如下图画圈处所示：

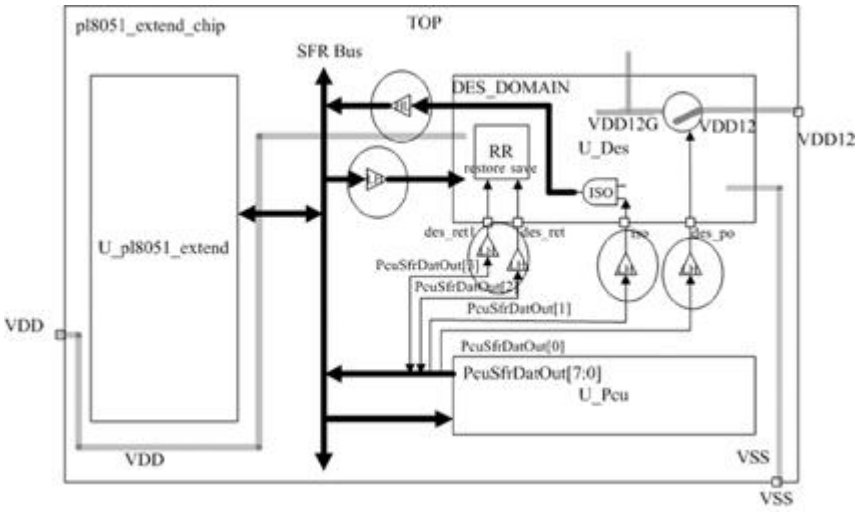

图 4-10 插入 Level Shifters

4.8 UPF Demo

```
######################
## CREATE POWER DOMAIS
######################
create_power_domain TOP
create_power_domain DES_DOMAIN -elements U_Des

######################
## TOPLEVEL CONNECTIONS
######################
#VDD
create_supply_port VDD
create_supply_net VDD -domain TOP
create_supply_net VDD -domain DES_DOMAIN -reuse
connect_supply_net VDD -port VDD

#VSS
create_supply_port VSS
create_supply_net VSS -domain TOP
create_supply_net VSS -domain DES_DOMAIN -reuse
connect_supply_net VSS -port VSS

#VDD12
create_supply_port VDD12
create_supply_net VDD12 -domain TOP
create_supply_net VDD12 -domain DES_DOMAIN -reuse
connect_supply_net VDD12 -port VDD12

#VDD12G
create_supply_net VDD12G -domain DES_DOMAIN
```

```
create_supply_net VDD12G -domain TOP -reuse

#####################
## PRIMARY POWER NETS
#####################
set_domain_supply_net TOP    -primary_power_net
VDD    -primary_ground_net VSS
set_domain_supply_net DES_DOMAIN   -primary_power_net VDD12G
-primary_ground_net VSS

###################
## DES_DOMAIN SETUP
###################
# Power Switch
create_power_switch des_sw \
    -domain DES_DOMAIN \
    -input_supply_port {in VDD12} \
    -output_supply_port {out VDD12G} \
    -control_port {des_po U_Pcu/PcuSfrDatOut[0]} \
    -on_state {state_on in {!des_po}} \
    -off_state {OFF {des_po}}

map_power_switch des_sw \
    -domain DES_DOMAIN \
    -lib_cell HDRSIHVTD0

# Isolation Cell
set_isolation des_iso_out \
    -domain DES_DOMAIN \
    -isolation_power_net VDD12 -isolation_ground_net VSS \
    -clamp_value 1 \
    -applies_to outputs

set_isolation_control des_iso_out \
    -domain DES_DOMAIN \
    -isolation_signal U_Pcu/PcuSfrDatOut[1] \
    -isolation_sense high \
    -location self

# Retention Register
set_retention des_ret -domain DES_DOMAIN \
    -retention_power_net VDD12G -retention_ground_net VSS

set_retention_control des_ret -domain DES_DOMAIN \
```

```
    -save_signal {U_Pcu/PcuSfrDatOut[2] high} \
    -restore_signal {U_Pcu/PcuSfrDatOut[3] low}

map_retention_cell des_ret \
    -domain DES_DOMAIN \
    -lib_cell_type RSDFCRD1

# Level Shifter
set_level_shifter des_ls_lh \
    -domain DES_DOMAIN \
    -applies_to inputs \
    -threshold 0.1 \
    -rule low_to_high \
    -location parent

set_level_shifter des_ls_hl \
    -domain DES_DOMAIN \
    -applies_to outputs \
    -threshold 0.1 \
    -rule high_to_low \
    -location parent

######################
# ADD PORT STATE INFO
######################
add_port_state VDD -state {LV 0.84}
add_port_state VDD12 -state {HV 1.08}
add_port_state VSS -state {ON 0.0}
add_port_state des_sw/out -state {HV 1.08} \
                    -state {OFF off}

##############
## CREATE PST
##############
create_pst design_pst -supplies {VDD VDD12 VDD12G VSS}
add_pst_state design_work -pst design_pst -state {LV HV HV ON}
add_pst_state stand_by -pst design_pst -state {LV HV OFF ON}
```

4.9 DC UPF Flow Script Demo

```
source -e ../scr/synopsys_dc.setup
source -e ../scr/ variable.tcl
set_clock_gating_style -min ${CG_MIN_BITWIDTH} -max_fanout
${CG_MAX_FANOUT} \
                        -sequential_cell ${CG_SEQ_CELL} \
                        -positive_edge_logic ${CG_POS_CELL_LIST} \
```

```
                        -negative_edge_logic ${CG_NEG_CELL_LIST} \
                        -control_point ${CG_CONTROL_POINT} \
                        -control_signal ${CG_CONTROL_SIGNAL} \
                        -setup ${CG_SETUP_VALUE} -hold ${CG_HOLD_VALUE}
set_operand_isolation_style -logic adaptive
set_operand_isolation_slack 0.1
set_svf ${NETLIST_PATH}${top}${VER}.svf
read_verilog ${RTL_INCLUDE}
current_design $top
uniquify
link
load_upf ${SCRIPT_PATH}power.upf
set auto_insert_level_shifters_on_clocks all ;#can be added in backend flow
set_voltage 1.08 -object_list {VDD12 VDD12G}
set_voltage 0.84 -object_list VDD
set_voltage 0.0 -object_list VSS
source -e -v ${SCRIPT_PATH}timing.tcl
#M1~M9 are avaliable, M1 will be used by STD
#set_ignored_layers -min_routing_layer M4
set_ignored_layers -max_routing_layer M7
report_ignored_layers > ${REPORT_PATH}${top}_ignored_layers.rpt
set_clock_gating_check -setup ${CG_SETUP_CHECK} -hold
${CG_HOLD_CHECK} [all_clocks]
set_fix_multiple_port_nets -feedthroughs -outputs -buffer_constant
set_attr -type string tcbn90lphpwc0d70d9_pg.db:tcbn90lphpwc0d70d9
default_threshold_voltage_group SVT_LS_LH
set_attr -type string tcbn90lphphvtwc0d70d9_pg.db:tcbn90lphphvtwc0d70d9
default_threshold_voltage_group HVT_LP_LH
set_attr -type string tcbn90lphpwc0d7_pg.db:tcbn90lphpwc0d7
default_threshold_voltage_group SVT_LV
set_attr -type string tcbn90lphphvtwc0d7_pg.db:tcbn90lphphvtwc0d7
default_threshold_voltage_group HVT_LV
set_attr -type string tcbn90lphpwc_pg.db:tcbn90lphpwc
default_threshold_voltage_group SVT
set_attr -type string tcbn90lphphvtwc_pg.db:tcbn90lphphvtwc
default_threshold_voltage_group HVT
set_attr -type string tcbn90lphphvtcgwc_pg.db:tcbn90lphphvtcgwc
default_threshold_voltage_group HVT_CG
set_attr -type string tcbn90lphphvtwc0d90d9_pg.db:tcbn90lphphvtwc0d90d9
default_threshold_voltage_group HVT_ISO
set_attr -type string tcbn90lphphvtwc0d90d7_pg.db:tcbn90lphphvtwc0d90d7
default_threshold_voltage_group HVT_LS_HL
set_attr -type string tcbn90lphpwc0d90d7_pg.db:tcbn90lphpwc0d90d7
default_threshold_voltage_group SVT_LS_HL
```

```
check_mv_design -verbose
 compile_ultra -scan -no_autoungroup -no_boundary_optimization
change_names -rules verilog -hierarchy
write -format ddc -hierarchy -output ${DDC_PATH}${top}_noscan${VER}.ddc
write -format verilog -hierarchy -output ${NETLIST_PATH}${top}_noscan${VER}.v
set write_sdc_output_lumped_net_capacitance false
set write_sdc_output_net_resistance false
write_sdc -version 1.5 ${SDC_PATH}${top}_cons${VER}.sdc
check_design > ${REPORT_PATH}${top}_check_design_postcomp.rpt
report_area > ${REPORT_PATH}${top}_area.rpt
check_mv_design –verbose > ${REPORT_PATH}${top}_check_mv_design.rpt
report_constraint -all > ${REPORT_PATH}${top}_all_vio.rpt
report_clock_gating -gating_elements >> ${REPORT_PATH}${top}_icg.rpt
report_operand_isolation -verbose -isolated > ${REPORT_PATH}${top}_opiso.rpt
report_threshold_voltage_group > ${REPORT_PATH}${top}_VT.rpt
report_threshold_voltage_group -verbose >> ${REPORT_PATH}${top}_VT.rpt
save_upf ${NETLIST_PATH}${top}${VER}_dc.upf #upf' for next part of flow
```

5.0 总结

本文从 CMOS 电路功耗原理入手，针对不同工艺尺寸下数字集成电路的低功耗物理实现方法进行描述，并通过一个例子介绍了 Synopsys UPF（Unified Power Format）文件对低功耗设计的描述原理。UPF 是 Synopsys 公司提出的一种对芯片中电源域设计进行约束的文件格式。通过与 UPF 格式匹配的 Liberty 文件，UPF 约束文件可以被整套 Galaxy 物理实现平台的任何一个环节直接使用，并将设计者的电源设计约束传递给设计工具，由工具完成设计的实现工作，从而实现整套数字集成电路低功耗物理实现的流程。

---

[1] Page 14, "Low Power Methodology Manual For System on Chip Design.", Micheal Keating, David Flynn, Robert Aitken, Alan Gibbons, KaiJian Shi, ISBN 978-0-387-71818-7

[2] Page 42, "Low Power Methodology Manual For System on Chip Design.", Micheal Keating, David Flynn, Robert Aitken, Alan Gibbons, KaiJian Shi, ISBN 978-0-387-71818-7

[3] Page 2-29, "Library Compiler™ User Guide: Modeling Timing, Signal Integrity, and Power in Technology Libraries Version A-2007.12, December 2007"

[4] Page 2-32, "Library Compiler™ User Guide: Modeling Timing, Signal Integrity, and Power in Technology Libraries Version A-2007.12, December 2007"

[5] Page 46, "Low Power Methodology Manual For System on Chip Design.", Micheal Keating, David Flynn, Robert Aitken, Alan Gibbons, KaiJian Shi, ISBN 978-0-387-71818-7

[6] Page 2-23, "Library Compiler™ User Guide: Modeling Timing, Signal Integrity, and Power in Technology Libraries Version A-2007.12, December 2007"

[7] Page 55, "Low Power Methodology Manual For System on Chip Design.", Micheal Keating, David Flynn, Robert Aitken, Alan Gibbons, KaiJian Shi, ISBN 978-0-387-71818-7

[8] Page 2-46, "Library Compiler™ User Guide: Modeling Timing, Signal Integrity, and Power in Technology Libraries Version A-2007.12, December 2007"

[9] Page 2-11, "Library Compiler™ User Guide: Modeling Timing, Signal Integrity, and Power in Technology Libraries Version A-2007.12, December 2007"

[10]Page 2-2, “Library Compiler™ User Guide: Modeling Timing, Signal Integrity, and Power in Technology Libraries Version A-2007.12, December 2007”