# 第四章 SRAM编译器

本章以TSMC 28nm SRAM为例，选择六管单元作为设计方案，介绍了SRAM编译器，然后详细分析SRAM编译器的表征（Characterization），提出预测三种仿真数据的方法，最后对预测的结果和预测方法进行了评价。

## 4.1 SRAM基本原理

### 4.1.1 SRAM架构

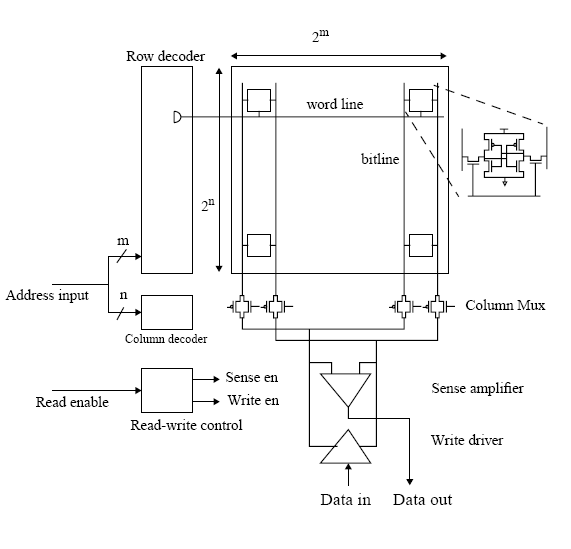
TSMC 28nm SRAM架构如图4-1。

图4-1 SRAM电路框架图

SRAM即静态随机存取存储器，SRAM主要由存储单元阵列（Core Cell Array）、地址译码器（Decoder）、灵敏放大器（Sense Amplifier）、控制电路（Control circuit）和缓冲/驱动电路（FFIO）。

如上图4-1，存储单元阵列有2n行和2m列，分别表示存储阵列有2m根字线（Word line）构成，每个字深存储2n根位线（Bit line）。每个SRAM包含2m×2n个存储单元。每个存储单元与一根字线和两根反相的位线相连接，并通过它们进行对存储单元的读写操作。

译码器分为行译码器（Row decoder）和列译码器（Column decoder）。行译码器接收地址信号，选中一根字线触发阵列的一行，在进行读操作的时候，字线驱动位线，反相的位线通过灵敏放大器读出存储单元的数据。

通常为了节省功耗和减少数据读取时间，存储器中的字数会远远多于每个字存储的位数，这样会使得存储器变得长而窄，因为位线过长会增加数据访问的速度。

### 4.1.2 SRAM存储单元

图4-2 SRAM 6管单元原理图

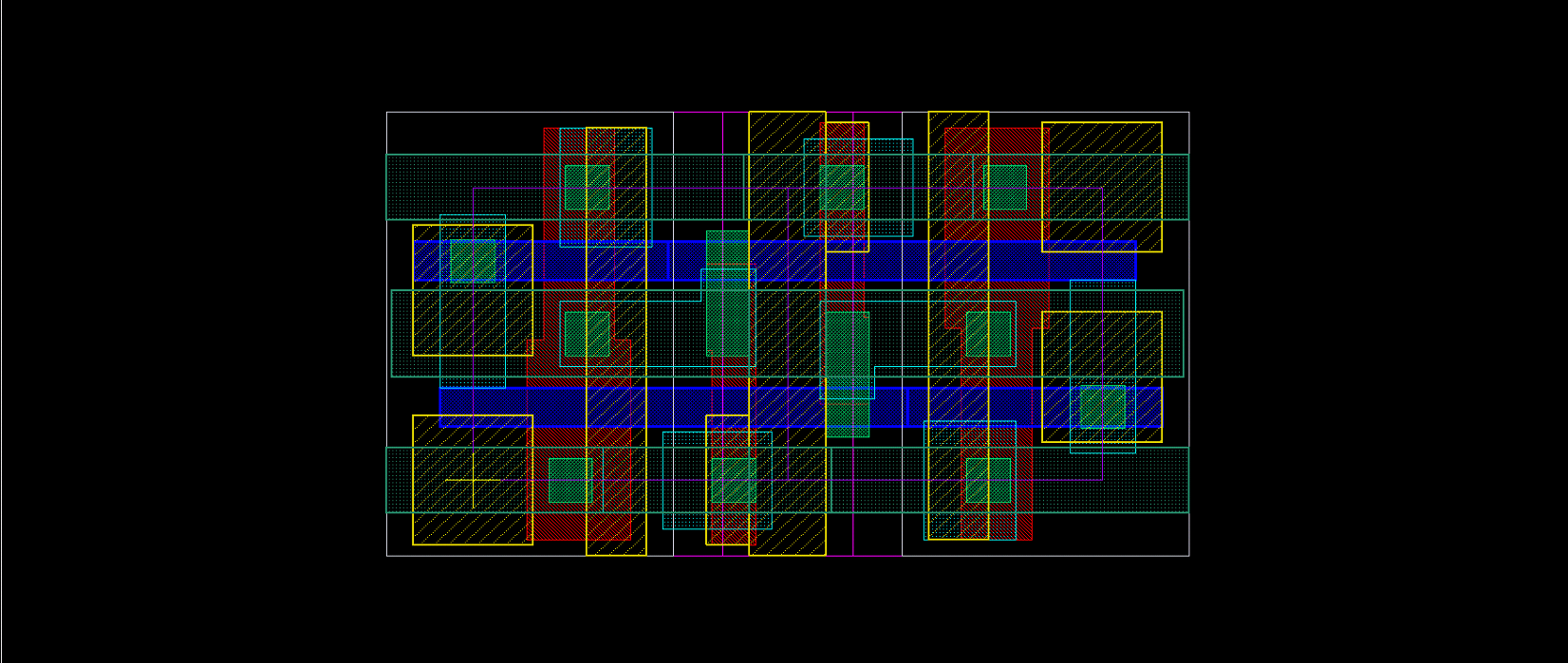
图中所示为6管SRAM单元，存储单元使用一根字线（word）、两根反相的位线（bit和bit\_b）。单元中包括了一对交叉耦合的反相器，并且每根位线链接了一个存取晶体管。一对互补的数据存储在交叉耦合的反相器上。如果数据受到轻微的干扰，由回路构成的正反馈将使数据恢复到VDD或GND。选中这个单元的字线就可以对这个单元进行读出或写入。

图4-3 TSMC 28nm 6管单元的版图

如图4-3为TSMC 28nm SRAM Compiler的存储单元的版图。

### 4.1.2 SRAM Compiler

一般而言，Memory Compiler属于IPs的一种，目前部分Design House和Foundry都可以向客户提供配套的Memory Compiler。Memory Compiler是设计一套完整的存储器单元模块（SRAM、Flash等）。设计中包含不同大小的存储器单元模块，尽量做到单元模块的面积和速度最优。用户可以通过简单的界面定义实现目标要求的存储器单元模块的GDSII文件和网表文件等。由于用户不能更改电路的内部结构和版图，所以面积和速度就是Memory Compiler的竞争力所在。

常用SRAMs是全定制设计，全定制设计以存储器密度、速度、功耗和良率为主要设计目标，全定制设计需要相应较长的设计周期。能在较短时间内完成满足设计需求的SRAM，就意味着在高度竞争的市场中具有强有力的竞争力。SRAM Compiler就是一种解决方法，SRAM Compiler不仅能节约成本，而且能在很大程度的缩短设计周期。

SRAM Compiler是一种能生成不同存储器配置的计算机智能化程序。SRAM Compiler主要功能包括：产生存储器网表（netlist）、模拟最坏情况下读写路径以产生时序信息、为在原理图中取代SRAM而建立相应的符号、建立带有时序信息的模拟模型、产生对应SRAM的物理版图、建立电源栅结构等。通常SRAM Compiler用来为诸如门阵列或标准单元电路的专用集成电路（ASICs）生成存储器。

#### 4.1.2.1 SRAM Compiler设计流程

图4-4 SRAM Compiler设计流程图

SRAM Compiler通过调用设计电路结构中的子模块的版图来产生存储器模块版图，所以不同的电路设计结构得到的版图会对SRAM Compiler的面积有很大的影响。

如图4-4是SRAM Compiler的设计流程，发开一套SRAM Compiler的主要流程如下：根据客户的需求以及市场信息，提出对SRAM的设计需求，主要是SRAM所具备的一些功能和特征，并对存储器单元进行评估，并检查是否满足存储器设计的速度和面积要求；然后对SPEC的配置进行电路图设计及预仿真，其次是页元（Leaf cell）的版图设计，通过DRC和LVS规则对电路图和版图进行优化调整；使用SRAM Compiler的Tiling Engine来产生存储器模板（Memory Instance）版图，最后是对不同配置的Instance进行表征（Characterization），主要通过时序（Timing）、功耗（Power）和电容（Capacitance）来体现。

在Memory Characterization环节中，主要是仿真工作和仿真数据收集。实际设计中，仿真工作的时间几乎占据了全部的Memory Characterization环节。因为仿真工作的数量由SPEC配置和PVT共同决定。而SPEC与PVT参数包含了特征、MUX、Bank、Word、Bit、Input Slew、Output loading、Process Corner、Voltage以及Temperature等。参数的繁多决定了仿真工作的数量的繁多。

#### 4.1.2.2 SRAM Compiler整合流程

图4-5 SRAM Compiler整合流程图

1. Leaf cell的版图和电路在通过DRC和LVS之后，通过Tiling Engine的脚本生成不同instance以及instance的版图文件（LEF、GDSII文件）和电路描述文件（Verilog语言文件、SPICE文件）；
2. Characterization环节通过设计者给出的Instance列表对不同instance的版图文件和工艺文件进行RC抽取，生成更贴近实际工作状态的CIR文件，并针对不同instance的SPEC配置文件的变量参数进行配置，通过脚本自动生成仿真文件，并进行仿真；
3. 仿真完成后，通过测量描述语言对仿真结果文件进行数据抽取，生成Rawdata文件，并通过计算文件生成最后的LIB文件。
4. 最后将LIB文件、LEF文件、GDSII文件、Verilog文件和SPICE文件整合，编写Complier实现程序，使得用户可以通过自定义instance的大小与功能自动生成SRAM及其网表文件、版图文件与表征数据库等。

通过Compiler工具中的各种引擎模块调用设计中相应的设计输入文件，如SPEC配置文件、Compiler的物理单元库（GDS），网表单元库（Netlist）、Memory Development Language（MDL）语言脚本等文件，具体需要的文件见图；Memory Compiler的输出文件结构如图，大致有GDSII、LEF、SPICE和Verilog文件等。

图4-6 Memory Compiler输入输出文件结构

## 4.2 Memory Characterization

### 4.2.1 仿真工作

Memory Characterization是SRAM Compiler设计中非常重要的环节，主要是instance的仿真工作以及仿真数据的收集。其中仿真数据主要是时序、功耗和电容。

Characterization的仿真工作具有一定的重复性，可以通过脚本的实现来生成Test bench，进行仿真、数据检查以及数据收集。Characterization减少了设计者的仿真时间，同时也可以减少一些人为的错误。

仿真的工作量主要由SPEC配置和PVT共同决定，SPEC配置有FEATURE、MUX、BANK、WORD、BIT、INPUT SLEW和OUTPUT LOADING等。

#### 4.2.1.1 FEATURE

本课题SRAM Compiler所采用的工艺是TSMC28nm的HPC+工艺版本，本课题SRAM Compiler所采用的功能如下：

1. Synchronous read and write operations
2. Column Mux options for the best aspect ratio
3. Fully customized layout density available in 0.9V ±10%
4. Three power management modes: Light Sleep, Deep Sleep and Shut Down to reduce static power
5. Bit write and global write operations available
6. Extra margin control option
7. Special test modes externally bypass read and write self-timing circuits
8. Redundancy inside to repair defective cell

|  |  |
| --- | --- |
| 1 | 具有deep sleep模式 |
| 2 | 具有shut down模式 |
| P | 具有light sleep功能 |
| B | 具有Bit write功能 |
| I | 具有IO redundancy功能 |
| M | 具有BIST功能 |
| E | 具有HVT Periphery |
| L | 具有LVT Periphery |

用户可以通过定义不同的word、bit和mux来优化存储器，从而实现面积、速度和功耗的最佳平衡。以上的特征都被集成到Memory Compiler中，用来实现系统应用和用户需求。在设定一些参数之后，Compiler就可以在很短的时间内生成memory instance的data sheet、GDSII Layout、LVS netlist和PR model等。

通常，Memory Compiler所提供的全部功能，用户在自定义生成SRAM的时候，根据需求选择其中几个或者全部的功能。这就要求在进行仿真的时候，针对不同的特征组合进行不同的仿真，这也是仿真工作量繁多的重要原因。

以下为需要进行仿真的特征组合：

-

1,2,b,i,m,p

12,1b,1i,1m,1p,2b,2i,2m,2p,bi,bm,bp,im,ip,mp

12b,12i,12m,12p,1bi,1bm,1bp,1im,1ip,1mp,2bi,2bm,2bp,2im,2ip,2mp,bim,bip,bmp,imp

12bi,12bm,12bp,12im,12ip,12mp,1bim,1bip,1bmp,1imp,2bim,2bip,2bmp,2imp,bimp

12bim,12bip,12bmp,12imp,1bimp,2bimp

12bimp

共计64组特征。针对每一组特征都有相应的仿真工作需要完成。

F1,F2:

b,bp

bi,bip

bim,bimp

bm,bmp

1b,2b.12b,1bp,2bp,12bp

1bi,2bi,12bi,1bip,2bip,12bip

1bim,2bim,12bim,1bimp,2bimp,12bimp

1bm,2bm,12bm,1bmp,2bmp,12bmp

1,2,12,1p,2p,12p

1i,2i,12i,1ip,2ip,12ip

1im,2im,12im,1imp,2imp,12imp

1m,2m,12m,1mp,2mp,12mp

-,p

I,ip

Im,imp

M,mp

F3:

Bim,bimp

Bi,bip

Bm,bmp

B,bp

1bim,2bim,12bim,1bimp,2bimp,12bimp

1bi,2bi,12bi,1bip,2bip,12bip

1bm,2bm,12bm,1bmp,2bmp,12bmp

1b,2b,12b,1bp,2bp,12bp

1im,2im,12im,1imp,2imp,12imp

1i,2i,12i,1ip,2ip,12ip

1m,2m,12m,1mp,2mp,12mp

1,2,12,1p,2p,12p

Im,imp

I,ip

M,mp

-,p

F4:

B,bm,bp,bmp

Bi,bim,bip,bimp

1b,2b,12b,1bm,2bm,12bm,1bp,2bp,12bp,1bmp,2bmp,12bmp

1bi,2bi,12bi,1bim,2bim,12bim,1bip,2bip,12bip,1bimp,2bip,12bimp

1,2,12,1m,2m,12m,1p,2p,12p,1mp,2mp,12mp

1i,2i,12i,1im,2im,12im,1ip,2ip,12ip,1imp,2imp,12imp

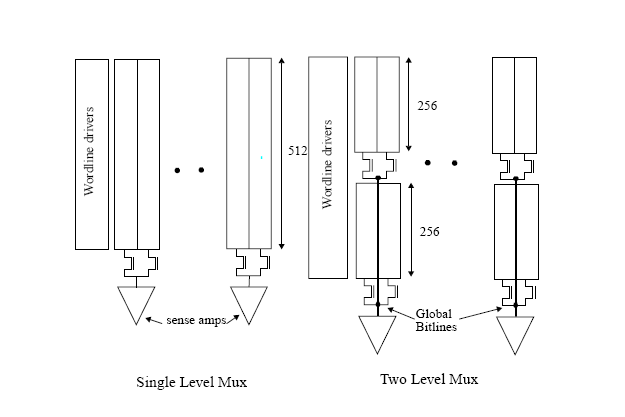
-,m,p,mp

I,im,ip,imp

#### INSTANCE Configration

用户可以通过定义不同的word、bit、mux和bank来定义需求的存储器。

通常随着存储阵列容量的变大，那么字线和位线就会变得更长。增加字线和位线的长度会使电容变大，从而导致延时和功耗的增加。因此，通常会将大容量的存储阵列分割成为多个较小的存储阵列，这样的较小的存储阵列就被称为体（Bank）。

通常为了节省功耗和减少数据读取时间，存储阵列中的字数会远远多于每个字存储的位数，这样会使得存储器变得长而窄，因为位线过长会增加数据访问的速度。

通常采用的另一个做法就是减少位线的长度。如图将一根正常的位线分割成两部分，每根位线的长度与其所驱动的单元数都减半，从而减少了位线的负载。这样子分割位线的做法叫做Column Mux。其中每根位线的长度被分成几份，Mux就等于几，且Mux总是2的幂值。在具体的SRAM Compiler设计中，会提供很多不同的字线、位线和Mux来供用户进行选择。

本课题的SRAM Compiler是小面积的存储器，设计中不分Bank，因此Bank的值为1。

Mux根据SPEC文件定义为2和4。当Mux取2时，Word最小取32，最大取512，步长为4，Bit最小取8，最大取144，步长为2；当Mux取4时，Word最小取64，最大取1024，步长为8，Bit最小取8，最大取72，步长为1。

|  |  |
| --- | --- |
| Word Range | |
| Column Mux=2 | 32,36,40,…508,512 |
| Column Mux=4 | 64,72,80,…1016,1024 |

|  |  |
| --- | --- |
| Bit Range | |
| Column Mux=2 | 8,10,12,…142,144 |
| Column Mux=4 | 8,9,10,…71,72 |

当Mux选定2或者4的时候，选择不同的Word和Bit的值即可确定Memory Instance的大小。从上表可以看出，当Mux=2的时候，Word可取120个值，Bit可以取140个值；当Mux=4的时候，Word可以取120个值，Bit可以取64个值。由此可计算出本课题的SRAM Compiler的Instance个数共有24480个。理论上，用户自定义生成的Memory Instance的可以是24480中的任何一种，这就要求每一个Memory Instance都有相应的仿真数据，供用户查询。

#### 4.2.1.3 IO

#### 4.2.1.4 FEATURE

#### 4.2.1.5 FEATURE

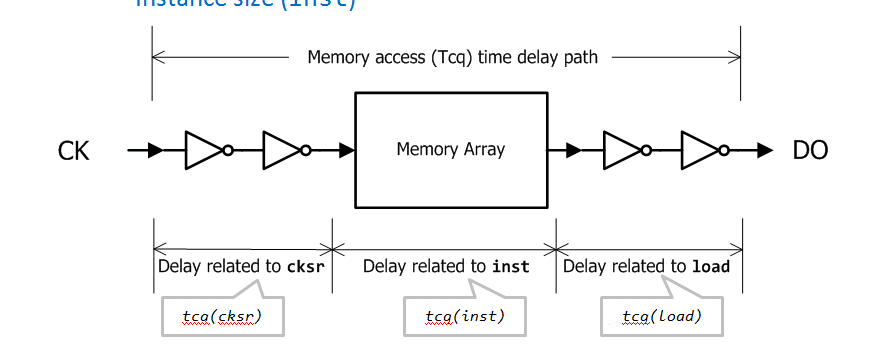
### 4.2.1 时序

SRAM有三种工作状态：保持（Standby）、写入（Write）和读出（Read）。

SRAM的数据保持时，芯片使能信号CEB处于无效状态，无论其他信号是什么状态，都不会对存储单元进行任何操作。

SRAM读取数据的时候，

SRAM时序同标准单元时序类似，以Tcq为例。

 由公式可以看出，Tcq主要有三部分组成，cksr是Clock slew rate，load是output loading。

其中tcq(cksr)只取最小instance和最大loading时的值；tcq(inst)只取最大cksr和最大loading时的值，tcq（load）取最大cksr和最小inst的值。

这就使得仿真次数和时间得到很大程度的减少，其中CKSR取值分别为0.004、0.050、0.100、0.200、0.500、0.750、1.000；output loading取值分别为0.001、0.020、0.050、0.100、0.250、0.300、0.350。

每一个instance的每一个时序都有7×7的查询表。

#### 读操作时序

SRAM读操作时序如图，其中WEB为写使能，CK是时钟信号，CEB为片选中信号，A为地址信号，DO为数据输出信号。

|  |  |
| --- | --- |
| **Parameter** | **Symbol** |
| Read cycle time | Trc |
| Data access time at the CK rising edge | Tcq |
| Output data hold time after the CK rising edge | Toh |
| Address setup time before the CK rising edge | Tas |
| Address hold time after the CK rising edge | Tah |
| GWEB,WEB setup time after the CK rising edge | Tws |
| GWEB,WEB hold time after the CK rising edge | Twh |
| CEB setup time before the CK rising edge | Tces |
| CEB setup time after the CK rising edge | Tceh |
| Clock high/low pulse widths | Thpw/Tlpw |

#### 写操作时序

|  |  |
| --- | --- |
| **Parameter** | **Symbol** |
| Write cycle time | Twc |
| Address setup time before the CK rising edge | Tas |
| Address hold time after the CK rising edge | Tah |
| GWEB,WEB setup time after the CK rising edge | Tws |
| GWEB,WEB hold time after the CK rising edge | Twh |
| Input data setup time before the CK rising edge | Tds |
| Input data hold time after the CK rising edge | Tdh |
| CEB setup time before the CK rising edge | Tces |
| CEB setup time after the CK rising edge | Tceh |
| Output data invalid after the CK rising edge | Twdx |
| Output data valid after the CK rising edge | Twdv |
| Clock high/low pulse widths | Thpw/Tlpw |

#### 电源模式时序

|  |  |
| --- | --- |
| **Parameter** | **Symbol** |
| LS fall setup time before CK rises | Tflss |
| LS rise setup time before CK rises | Trlsh |
| DS fall setup time before CK rises | Tfdss |
| DS rise setup time before CK rises | Trdsh |
| DS rise to output hold previous data time | Tdsoh |
| DS rise to all output fall to logic ‘0’ delay | Tdsq |
| SD fall setup time before CK rises | Tfsds |
| SD rise setup time before CK rises | Trsdh |
| SD rise to output hold previous data times | Tsdoh |
| SD rise to all output fall to logic ‘0’ delay | Tsdq |

在Light sleep模式，SRAM可以在不到一个完整的时钟周期之内恢复（wake up），但其静态功耗也是三种模式中最大的；Shut down模式具有最少的静态功耗，但其恢复时间是三种模式中最长的；Deep sleep模式的静态功耗和恢复时间都是介于两者之间。

Instance配置

|  |  |
| --- | --- |
| **Word Range** | |
| Column Mux = 2 | 32，80，128，192，256，320，384，448，512 |
| Column Mux = 4 | 64，160，256，384，512，640，768，896，1024 |

|  |  |
| --- | --- |
| Bit range | |
| Column Mux = 2 | 8，24，48，72，96，120，144 |
| Column Mux = 4 | 8，12，24，36，48，60，72 |

共计126个instance

PVT

|  |  |
| --- | --- |
| Process | TT FG SG |
| Voltage | 0.72 0.81 0.9 0.99 1.08 |
| Temp | -40 25 125 |

以W32B8M2B1为例，当cksr = 0.004，load = 0.001，datasr = 0.004时，Tcq在45个corner下的值。

本课题分别从读操作时序、写操作时序和电源模式中选择20个时序，分别是Tcq、Toh、Troslp、Tfoslp、Tsdq、Tsdoh、Tdsq、Tdsoh、Tas、Tah、Tds、Tdh、Tceh、Tces、Tws、Twh、Thpw、Tlpw、Trc、Twc。

|  |  |  |  |
| --- | --- | --- | --- |
|  | FG1P08VN40C | TT0P9V25C | SG0P72V125C |
| Tcq | 0.2169 | 0.3015 | 0.4945 |
| Toh | 0.1824 | 0.2409 | 0.3753 |
| Troslp | 0.0100 | 0.0100 | 0.0113 |
| Tfoslp | 0.0100 | 0.0100 | 0.0100 |
| Tsdq | 0.2459 | 0.3445 | 0.5670 |
| Tsdoh | 0.2319 | 0.3197 | 0.5198 |
| Tdsq | 0.2459 | 0.3445 | 0.5670 |
| Tdsoh | 0.2319 | 0.3197 | 0.5198 |
| Tas | 0.0240 | 0.0359 | 0.0619 |
| Tah | 0.0350 | 0.0486 | 0.0775 |
| Tds | 0.0100 | 0.0100 | 0.0196 |
| Tdh | 0.0526 | 0.0698 | 0.1068 |
| Tceh | 0.0363 | 0.0509 | 0.0839 |
| Tces | 0.0609 | 0.0878 | 0.1475 |
| Tws | 0.0364 | 0.0550 | 0.1002 |
| Twh | 0.0526 | 0.0703 | 0.1097 |
| Thpw | 0.0655 | 0.0926 | 0.1517 |
| Tlpw | 0.1040 | 0.1500 | 0.2545 |
| Trc | 0.2548 | 0.3481 | 0.5620 |
| Twc | 0.2548 | 0.3510 | 0.5888 |

本课题中采用4个instance，他们分别是W32B8M2B1、W32B144M2B1、W512B8M2B1和W512B144M2B1。Instance的形状与阵列的大小密切相关，从版图中我们可以看出存储阵列几乎占据了SRAM版图98%的面积。4.1中讲到通常阵列会被设计成行数较少、列数较多的结构，因为位线的负载是通过并联的，过长的位线会使得SRAM读写的速度变得很慢。相同的存储容量的SRAM会因为其字线和位线的不同具有不同的读写速度，也就是时序。

Instance的形状与字数和位数线性相关，Instance总是矩形，其长通常为Word/Mux，其宽通常为Bit/Mux。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Inst | W32B8M2B1 | W32B144M2B1 | W512B8M2B1 | W512M144M2B1 |
| Word×Bit | 16×16 | 16×288 | 256×16 | 256×288 |
| 特征 | 面积最小 | 高瘦 | 矮扁 | 面积最大 |
| 存储容量 | 0.4MB | 4.5MB | 4MB | 72MB |

论文高速SRAM编译器时序算法中提出了一种时序模型：



方法1

本方法可以实现通过确定的CKSR、DATASR和LOAD的时序预测其他所有的CKSR、DATASR和LOAD的时序。

本方法使用cksr=0.004，datasr=0.004，load=0.001时的时序，去预测了其他时序。

Instance预测：

第二时序预测通过某个instance的全corner来预测其他instance的时序。可以实现相同cksr、datasr、load的时候，通过单个instance的全corner时序以及6、8组目标instance全corner的时序，可以实现全corner时序的预测。

方法一：

以W32B8M2B1全corner时序，W32B144M2B1、W512B8M2B1和W512B144M2B1的8个corner时序来预测W32B144M2B1的其他corner时序。

方法一中的8个corner分别是FG1P08VN40C、FG0P9V25C、FG1P08V25C、TT0P81V25C、TT0P99V25C、SG0P72VN40C、SG0P9V25C和SG1P08V125C。

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Tcq** | **toh** | **troslp** | **tfoslp** | **tsdq** | **tsdoh** | **tdsq** | **tdsoh** | **tas** | **tah** | **tds** | **tdh** | **tceh** | **tces** | **tws** | **twh** | **thpw** | **tlpw** | **trc** | **twc** |
| 1 | 99 | 99 | 98 | 100 | 99 | 99 | 99 | 99 | 99 | 99 | 100 | 99 | 99 | 99 | 98 | 99 | 99 | 98 | 99 | 99 |
| 2 | 99 | 99 | 98 | 100 | 99 | 99 | 99 | 99 | 97 | 99 | 97 | 99 | 99 | 99 | 99 | 99 | 99 | 99 | 99 | 99 |
| 3 | 99 | 99 | 98 | 100 | 99 | 99 | 99 | 99 | 97 | 99 | 100 | 99 | 99 | 99 | 98 | 99 | 99 | 98 | 99 | 99 |

从表中可以看出本方法对于预测其他instance的时序效果非常好。

方法二：

以W32B8M2B1全corner时序，W32B144M2B1、W512B8M2B1和W512B144M2B1的6个corner时序来预测W32B144M2B1的其他corner时序。

本方法中使用6个corner分别是FG1P08VN40C、FG1P08V25C、TT0P81V25C、TT0P99V25C、SG0P72VN40C和SG0P72VN40C。

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Tcq** | **toh** | **troslp** | **tfoslp** | **tsdq** | **tsdoh** | **tdsq** | **tdsoh** | **tas** | **tah** | **tds** | **tdh** | **tceh** | **tces** | **tws** | **twh** | **thpw** | **tlpw** | **trc** | **twc** |
| 1 | 99 | 99 | 98 | 100 | 99 | 99 | 99 | 99 | 99 | 99 | 100 | 99 | 99 | 99 | 97 | 99 | 99 | 99 | 99 | 99 |
| 2 | 99 | 99 | 98 | 100 | 99 | 99 | 99 | 99 | 97 | 99 | 95 | 99 | 99 | 98 | 99 | 99 | 99 | 99 | 99 | 99 |
| 3 | 99 | 99 | 98 | 100 | 99 | 99 | 99 | 99 | 96 | 99 | 100 | 99 | 99 | 99 | 97 | 99 | 99 | 99 | 99 | 99 |

从表中可以看出本方法对于预测其他instance的时序效果非常好。

方法三：

以W32B8M2B1全corner时序，W32B144M2B1、W512B8M2B1和W512B144M2B1的8个corner时序来预测W32B144M2B1的其他corner时序。本方法中的八个corner分别是FG0P81VN40C、FG0P99VN40C、FG0P9V25C、TT0P9VN40C、TT0P9V125C、SG0P81VN40C、SG0P99VN40C和SG0P9V125C。

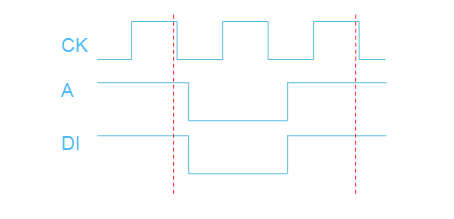
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Tcq** | **toh** | **troslp** | **tfoslp** | **tsdq** | **tsdoh** | **tdsq** | **tdsoh** | **tas** | **tah** | **tds** | **tdh** | **tceh** | **tces** | **tws** | **twh** | **thpw** | **tlpw** | **trc** | **twc** |
| 1 | 99 | 99 | 98 | 100 | 99 | 99 | 99 | 99 | 99 | 99 | 100 | 99 | 99 | 99 | 98 | 99 | 99 | 98 | 99 | 99 |
| 2 | 99 | 99 | 98 | 100 | 99 | 99 | 99 | 99 | 97 | 99 | 96 | 99 | 99 | 99 | 99 | 99 | 99 | 99 | 99 | 99 |
| 3 | 99 | 99 | 98 | 100 | 99 | 99 | 99 | 99 | 97 | 99 | 100 | 99 | 99 | 99 | 98 | 99 | 99 | 98 | 99 | 99 |

从表中可以看出本方法对于预测其他instance的时序效果非常好。

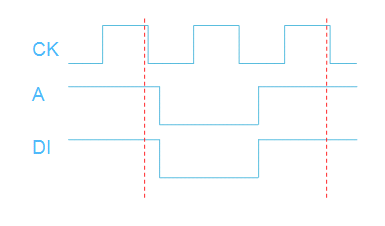
### 4.2.2 功率

需要测量的电流：

|  |  |
| --- | --- |
| IR\_AVG | 读电流平均值 |
| IW\_AVG | 写电流平均值 |
| IRW\_AVG | 读写电流平均值 |
| ISTD\_WITH\_CK\_AVG | 带时钟的standby电流平均值 |
| ISTD\_NO\_CK\_AVG | 不带时钟的standby电流平均值 |
| I\_PEAK\_MAX | 电流峰值 |
| I\_LS\_AVG | Light sleep电流平均值 |
| I\_SL\_AVG | Deep sleep电流平均值 |
| I\_DS\_AVG | Shut down电流平均值 |

读电流：

如图，读电流的测量时间从CK的下降沿到下一个下降沿，在读周期中只有DI、A、CK在翻转，其他的Pin都是standby，且A和DI应该变化两次。

写电流：

如图，写电流的测量时间从CK的下降沿到CK的下一个下降沿，在写周期中只有DI、A、CK在翻转，其他的Pin都处于standby，且A和DI都翻转两次。

Light sleep电流

在light sleep下，所有的Pin都保持原值，然后在light slepp的最后几个cycle，选择两个周期测量。

Deep sleep电流

在deep sleep mode下，所有pin都保持原值，在deep sleep的最后几个cycle，选择两个周期测量。

Shut down电流

在shut down模式下，所有的pin都保持原值，在shut down的最后几个cycle，选择两个周期测量。

功耗选择了六个参数分别是dc ac\_read ac\_write ls ds sd

其中ac\_read，ac\_write分别是：

以W32B8M2B1\_12BIMPS为例，表中是全corner的六个参数的

本方法共采集了18个instance的全corner仿真数据，通过程序处理和运算得到每组仿真数据分别为6个参数。

共得到18组45×6的数据集。鉴于功率参数的复杂性，本方法的训练集采用4个instance的数据集，上一节提到了字深的长短和位长的大小对时序的影响，同样的，字深的长达和位长的大小对功率也有同样的影响。

本方法考虑到样本的多样性，以及数据的差异性，采用4个形状各异的instance作为训练集：

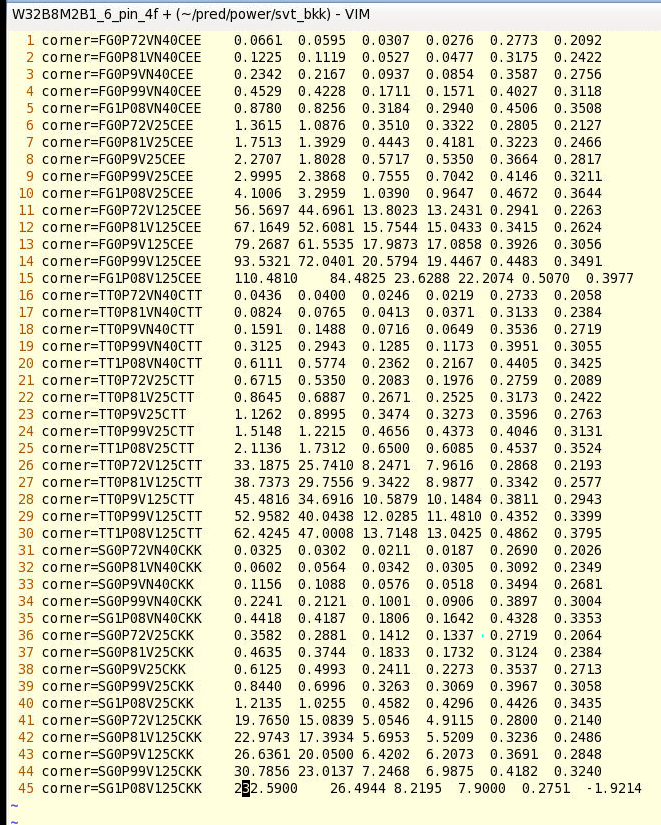
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Inst | W32B8M2B1 | W32B144M2B1 | W512B8M2B1 | W512M144M2B1 |
| Word×Bit | 16×16 | 16×288 | 256×16 | 256×288 |
| 特征 | 面积最小 | 高瘦 | 矮扁 | 面积最大 |
| 存储容量 | 0.4MB | 4.5MB | 4MB | 72MB |

以其他14个instance的数据作为测试集，分别作了14次学习和预测。

分别是W80B8M2B1、W128B8M2B1、W192B8M2B1、W256B8M2B1、W320B8M2B1、W384B8M2B1、W448B8M2B1、W80B144M2B1、W128B144M2B1、W192B144M2B1、W256B144M2B1、W320B144M2B1、W384B144M2B1、W448B144M2B1等14个instance。

因为功率参数受驱动电压波动的影响会有很明显的波动，通常在仿真阶段SRAM Compiler会选择三种电压作为仿真基准电压，分别是HVT（High Voltage Threshold）、SVT（Standard Voltage Threshold）和LVT（Low Voltage Threshold）。通常功耗会随着阈值电压的增加而减少，就是相同条件下，HVT的功耗最小，LVT的功耗最大。

以SVT为例，如图。

从表中可以看出，随着不同corner、电压的变化，功率参数的波动比较规律。但是随着温度的变化，六个功率参数的变化差异较大。

本方法将不同的温度分开进行学习。以W80B8M2B1为目标例，

125C

对125C的数据进行学习，

|  |  |
| --- | --- |
| 训练集 | 测试集 |
| FG0P72V125C  FG1P08V125C  TT0P9V125C  SG0P72V125C  SG1P08V125C | FG0P81V125C  FG0P9V125C  FG0P99V125C  TT0P72V125C  TT0P81V125C  TT0P99V125C  TT1P08V125C  SG0P81V125C  SG0P9V125C  SG0P99V125C |

本方法的预测结果如下：

25C

|  |  |
| --- | --- |
| 训练集 | 测试集 |
| FG0P72V25C  FG1P08V25C  TT0P9V25C  SG0P72V25C  SG1P08V25C | FG0P81V25C  FG0P9V25C  FG0P99V25C  TT0P72V25C  TT0P81V25C  TT0P99V25C  TT1P08V25C  SG0P81V25C  SG0P9V25C  SG0P99V25C |

N40C

|  |  |
| --- | --- |
| 训练集 | 测试集 |
| FG0P72VN40C  FG1P08VN40C  TT0P9VN40C  SG0P72VN40C  SG1P08VN40C | FG0P81VN40C  FG0P9VN40C  FG0P99VN40C  TT0P72VN40C  TT0P81VN40C  TT0P99VN40C  TT1P08VN40C  SG0P81VN40C  SG0P9VN40C  SG0P99VN40C |

将来；每个功耗参数的准确度取平均值。

以下就是

125C

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 99.38% | 99.58% | 99.38% | 99.48% | 96.33% | 96.12% |
| 99.46% | 99.64% | 99.08% | 99.09% | 96.45% | 96.25% |
| 99.49% | 99.67% | 98.71% | 98.66% | 96.45% | 96.22% |
| 99.51% | 99.61% | 98.40% | 98.19% | 96.50% | 96.47% |
| 99.53% | 99.66% | 98.18% | 97.75% | 96.39% | 96.44% |
| 99.55% | 99.64% | 97.98% | 97.36% | 96.36% | 96.56% |
| 99.57% | 99.67% | 97.80% | 96.98% | 96.14% | 96.53% |
| 99.47% | 99.48% | 99.46% | 98.83% | 96.32% | 96.01% |
| 99.50% | 99.49% | 99.60% | 98.87% | 96.31% | 96.32% |
| 99.59% | 99.45% | 99.71% | 98.86% | 95.77% | 95.69% |
| 99.69% | 99.53% | 99.68% | 98.98% | 95.81% | 95.79% |
| 99.74% | 99.62% | 99.62% | 99.05% | 95.86% | 96.03% |
| 99.79% | 99.69% | 99.59% | 99.10% | 95.88% | 96.12% |
| 99.82% | 99.75% | 99.58% | 99.16% | 95.97% | 96.24% |

25

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 98.44% | 98.98% | 98.35% | 97.87% | 99.58% | 99.70% |
| 98.46% | 99.13% | 98.20% | 97.47% | 99.64% | 99.69% |
| 98.62% | 99.46% | 98.14% | 97.20% | 99.60% | 99.67% |
| 98.59% | 99.54% | 98.05% | 96.93% | 99.64% | 99.68% |
| 98.51% | 99.38% | 97.94% | 96.67% | 99.61% | 99.70% |
| 98.48% | 99.37% | 97.89% | 96.50% | 99.61% | 99.75% |
| 98.48% | 99.45% | 97.85% | 96.36% | 99.59% | 99.73% |
| 99.38% | 99.42% | 99.76% | 99.10% | 99.55% | 99.64% |
| 99.48% | 99.52% | 99.74% | 98.94% | 99.62% | 99.67% |
| 99.55% | 99.36% | 99.57% | 98.87% | 99.57% | 99.63% |
| 99.71% | 99.58% | 99.46% | 98.76% | 99.56% | 99.69% |
| 99.74% | 99.58% | 99.37% | 98.67% | 99.59% | 99.72% |
| 99.77% | 99.54% | 99.31% | 98.59% | 99.60% | 99.71% |
| 99.70% | 99.51% | 99.28% | 98.45% | 99.63% | 99.69% |

N40

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 99.82% | 99.84% | 99.53% | 99.45% | 99.75% | 99.77% |
| 99.80% | 99.85% | 99.47% | 99.35% | 99.77% | 99.78% |
| 99.77% | 99.82% | 99.45% | 99.32% | 99.80% | 99.87% |
| 99.82% | 99.86% | 99.43% | 99.29% | 99.77% | 99.87% |
| 99.83% | 99.84% | 99.42% | 99.27% | 99.80% | 99.88% |
| 99.82% | 99.84% | 99.41% | 99.26% | 99.77% | 99.86% |
| 99.82% | 99.82% | 99.40% | 99.24% | 99.81% | 99.89% |
| 99.54% | 99.54% | 99.68% | 99.37% | 99.79% | 99.73% |
| 99.70% | 99.64% | 99.77% | 99.57% | 99.73% | 99.83% |
| 99.68% | 99.70% | 99.82% | 99.51% | 99.82% | 99.82% |
| 99.66% | 99.63% | 99.85% | 99.57% | 99.74% | 99.89% |
| 99.70% | 99.75% | 99.91% | 99.61% | 99.76% | 99.90% |
| 99.67% | 99.69% | 99.89% | 99.58% | 99.84% | 99.90% |
| 99.71% | 99.73% | 99.90% | 99.58% | 99.84% | 99.79% |

### 4.2.3 pincap

## 4.3 本章小结