上海交通大学
Shanghai Jiao Tong University

Verilog HDL 硬件描述语言（6）
逻辑综合基础

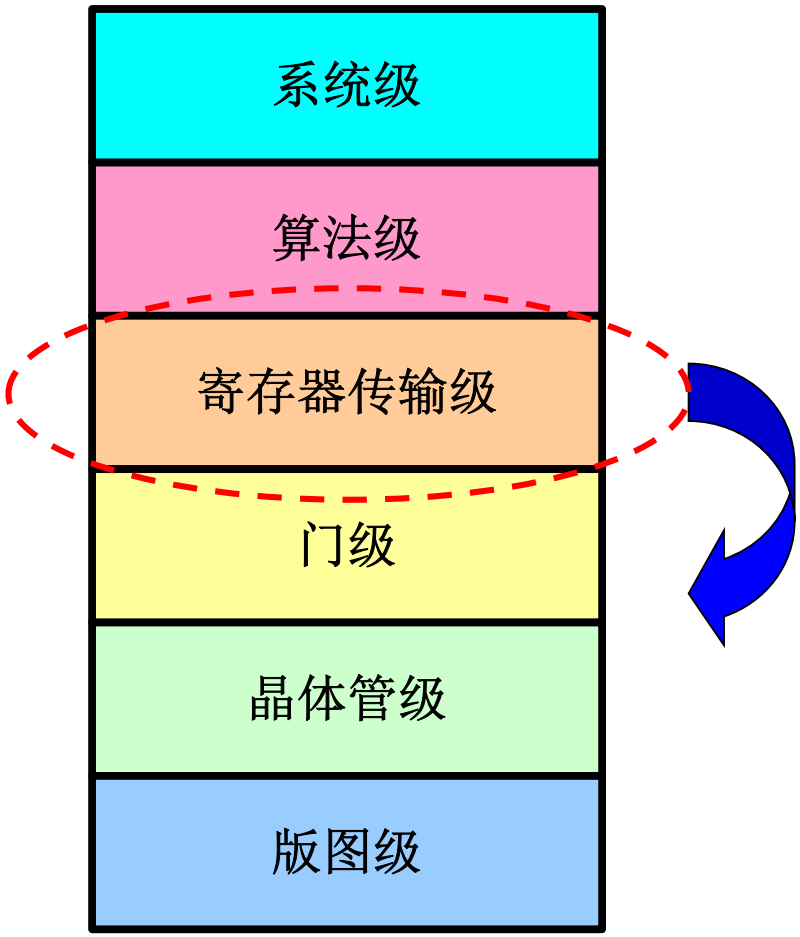
上海交通大学微电子学院
蒋剑飞

主要内容

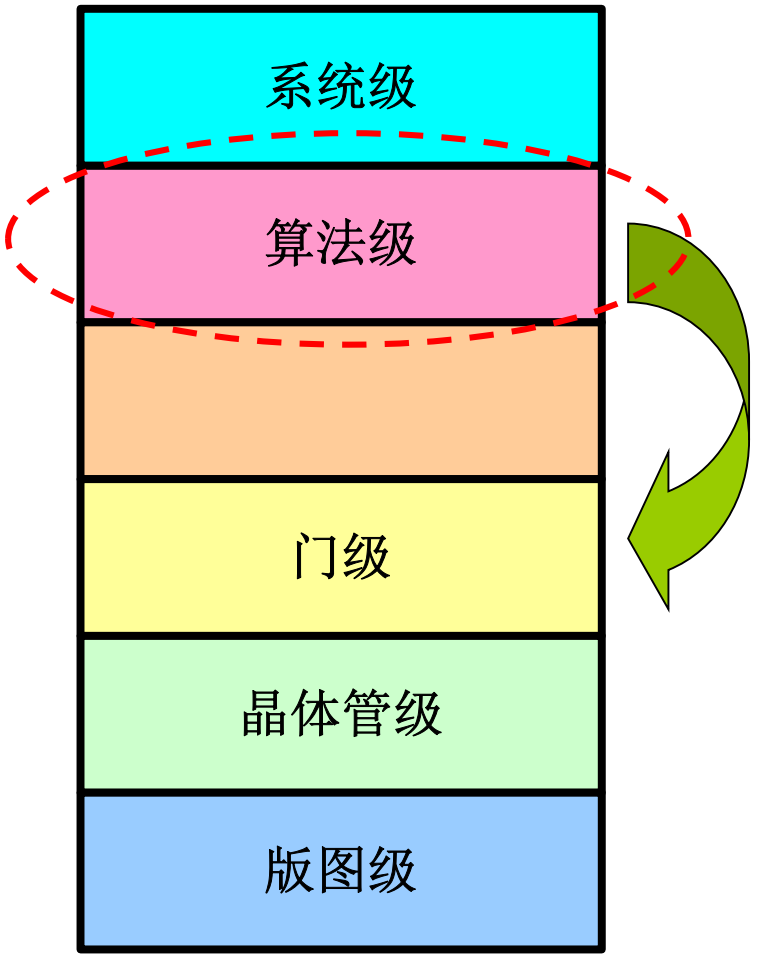
- 逻辑综合概念
- 数字IC实现策略
- 可综合设计

逻辑综合

RTL（HDL）综合

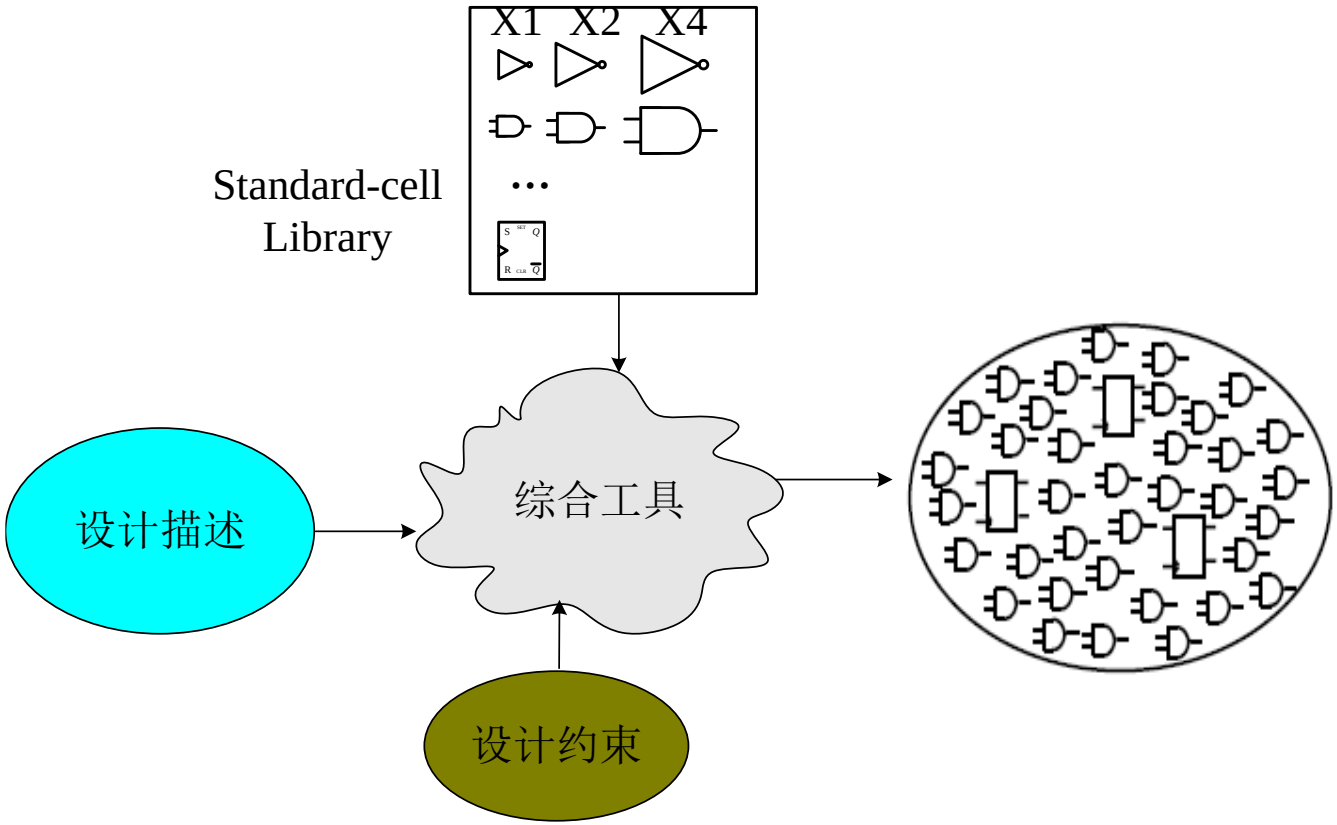


算法（C）综合



逻辑综合

- 逻辑综合——将设计从高层次的描述转换到优化的门级表示这样一个过程。

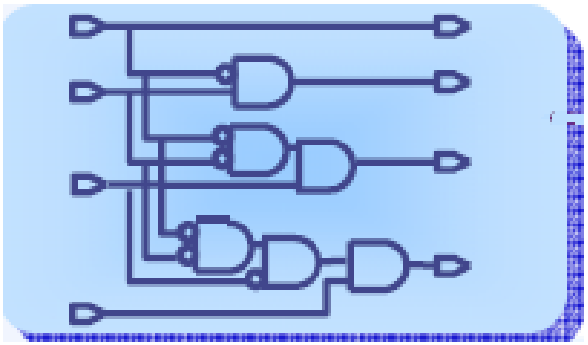


综合过程

```
residue = 16'h0000;
if (high_bits == 2'b10)
      residue = state_table[index];
else  state_table[index] = 16'h0000;
```

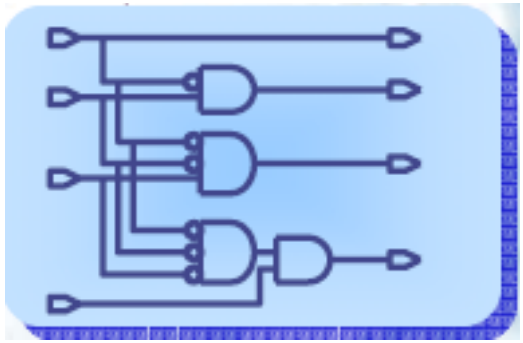
RTL代码

转换



逻辑表达式

优化与映射



网表

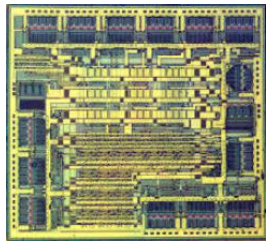
主要内容

- 逻辑综合概念
- 数字IC实现策略
- 可综合设计

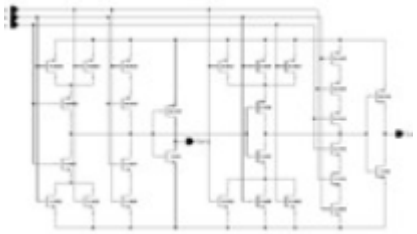
数字IC实现策略

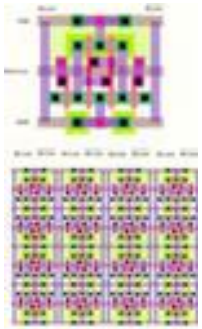
- 半定制设计

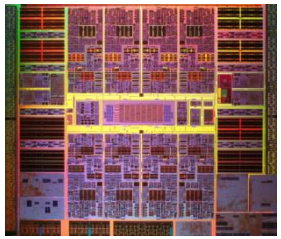
```
residue = 16'h0000;
if (high_bits == 2'b10)
      residue = state_table[index];
else  state_table[index] = 16'h0000;
```



- 定制设计







- 以阵列为基础的实现

```
residue = 16'h0000;
if (high_bits == 2'b10)
      residue = state_table[index];
else  state_table[index] = 16'h0000;
```

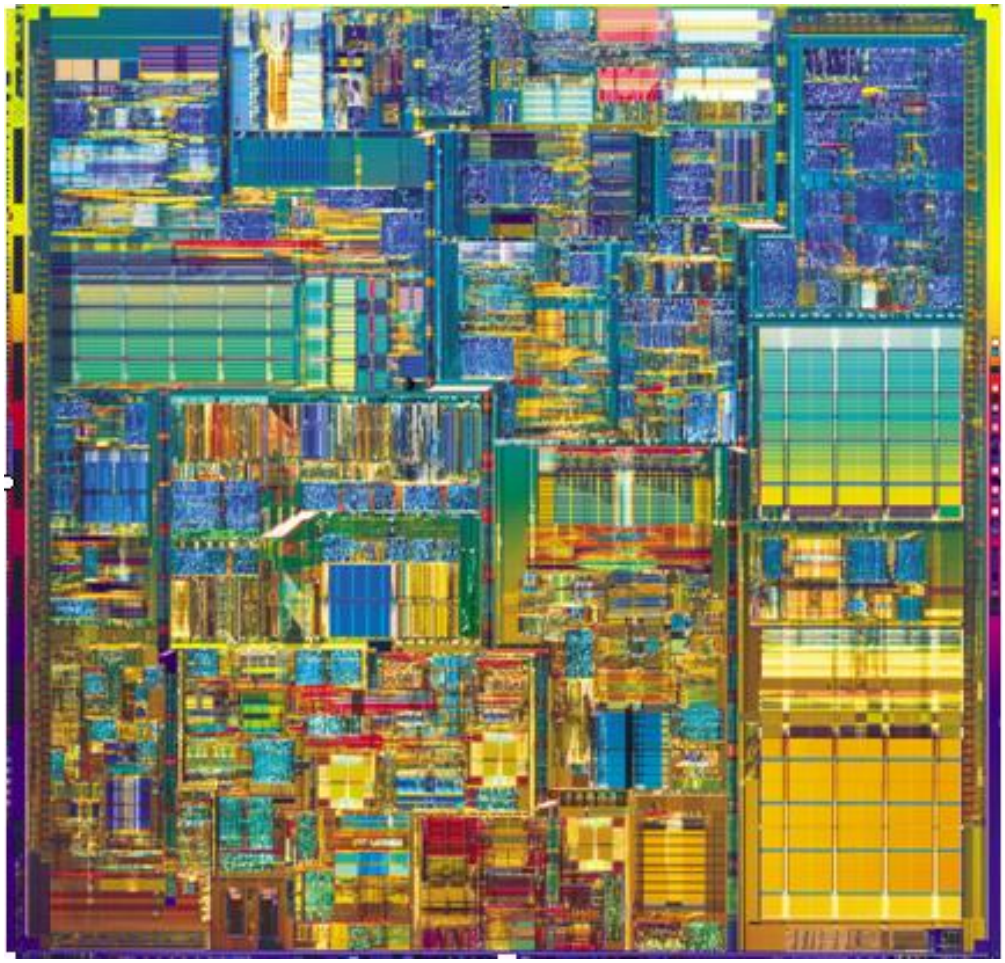
半定制设计

- 以单元为基础进行设计。重复利用有限的单元库来减少设计难度。
- 对于特定的工艺，单元库只需要设计一次与验证一次，以后重复使用。
- 受库本身约束，仔细调整能力较弱。
- 是目前最为流行的设计方法。

定制设计

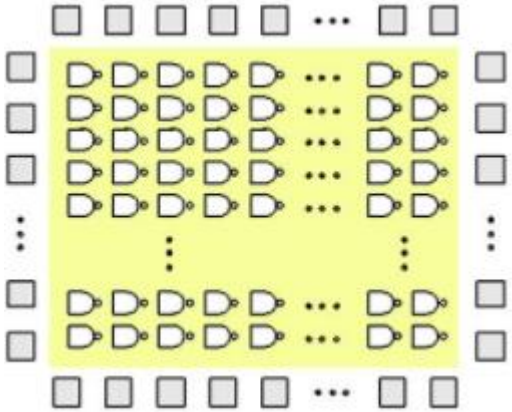
- 高性能高成本，在早期的处理器设计里应用，成本可以分摊到大量生产中。
- 在高性能处理器的特殊模块，不考虑成本的超级计算机中使用。
- 库单元设计，存储器设计。
- 缺点：对于与特定工艺相关性大。

Intel Pentium 4

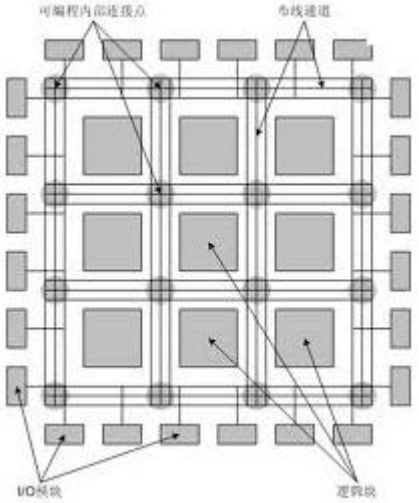


以阵列为基础的实现方法

- 门阵列/门海



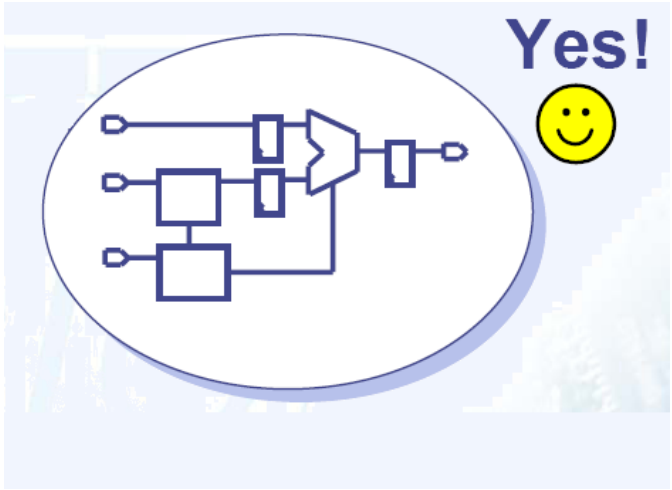
- 现场可编程门阵列（FPGA）

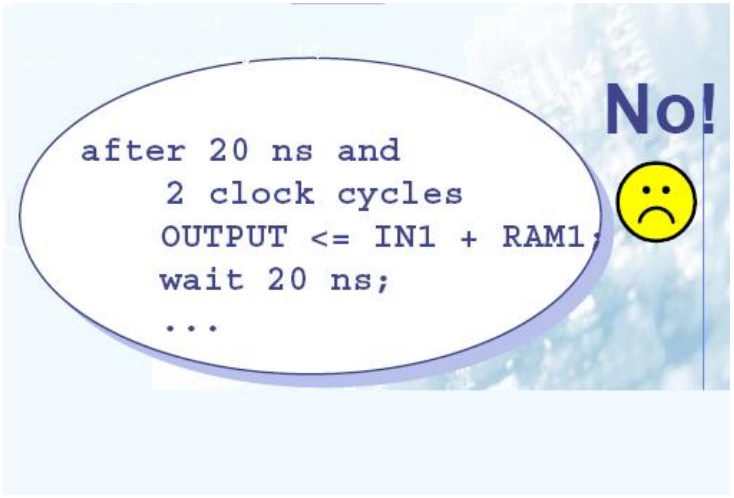


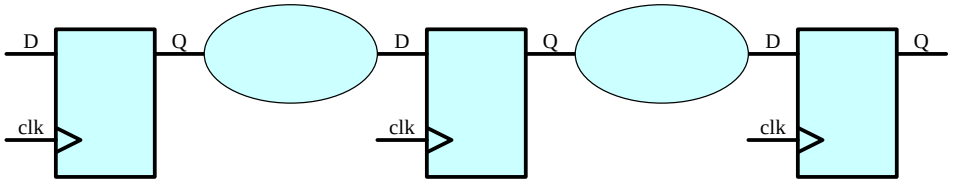
主要内容

- 逻辑综合概念
- 数字IC实现策略
- 可综合设计

设计描述







Always think of hardware!

标准单元库

- 标准单元库包括版图库、符号库、电路逻辑库等。
- 包含了组合逻辑、时序逻辑、功能单元和特殊类型单元。
- 是集成电路芯片后端设计过程中的基础部分。一般每个工艺厂商在每个工艺下都会提供相应的标准单元。

标准单元库

- Base Cells
  - ADDF
  - ADDFH
  - ADDH
  - AND2
  - AND3
  - AND4
  - AOI21
  - AOI211
  - AOI22
  - AOI221
  - AOI222
  - AOI2BB1
  - AOI2BB2
  - AOI31
  - AOI32
  - AOI33
  - BUF
  - CLKBUF
  - CLKINV
  - DFF
  - DFFHQ
  - DFFN
  - DFFNR
  - DFFNS
  - DFFNSR

Cell Description

The AND2 cell provides the logical AND of two inputs (A, B). The output (Y) is represented by the logic equation:

$Y = (A \bullet B)$

Logic Symbol

Functions

A	B	Y
0	x	0
x	0	0
1	1	1

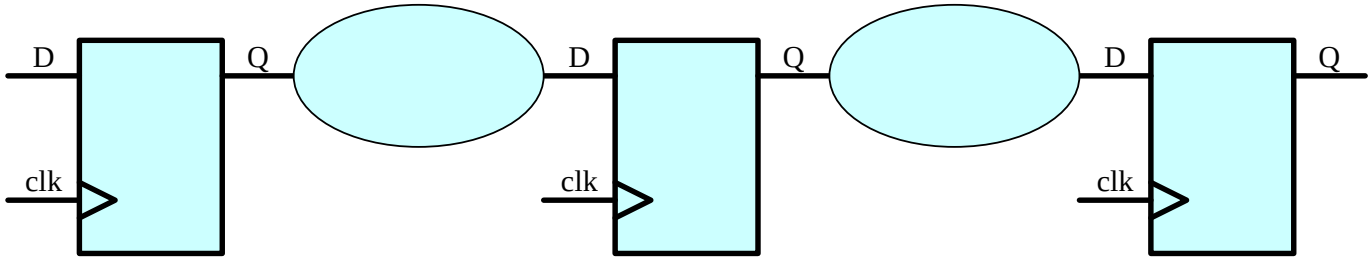
Cell Size

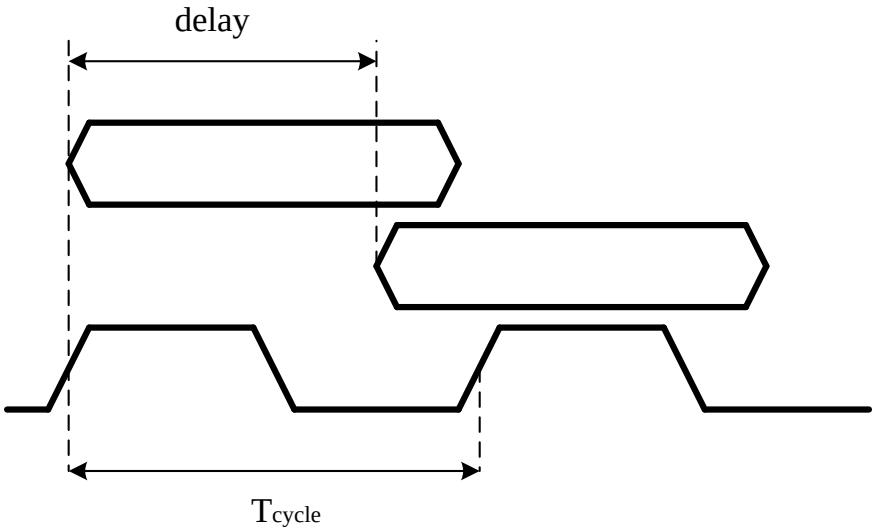
Drive Strength	Height (µm)	Width (µm)
AND2XL	5.04	2.64
AND2X1	5.04	2.64
AND2X2	5.04	2.64
AND2X4	5.04	3.30

设计约束

- 环境约束（温度、工艺、电压）
- 工作频率
- 电路面积
- 最大扇出、最大电容
- 端口延时

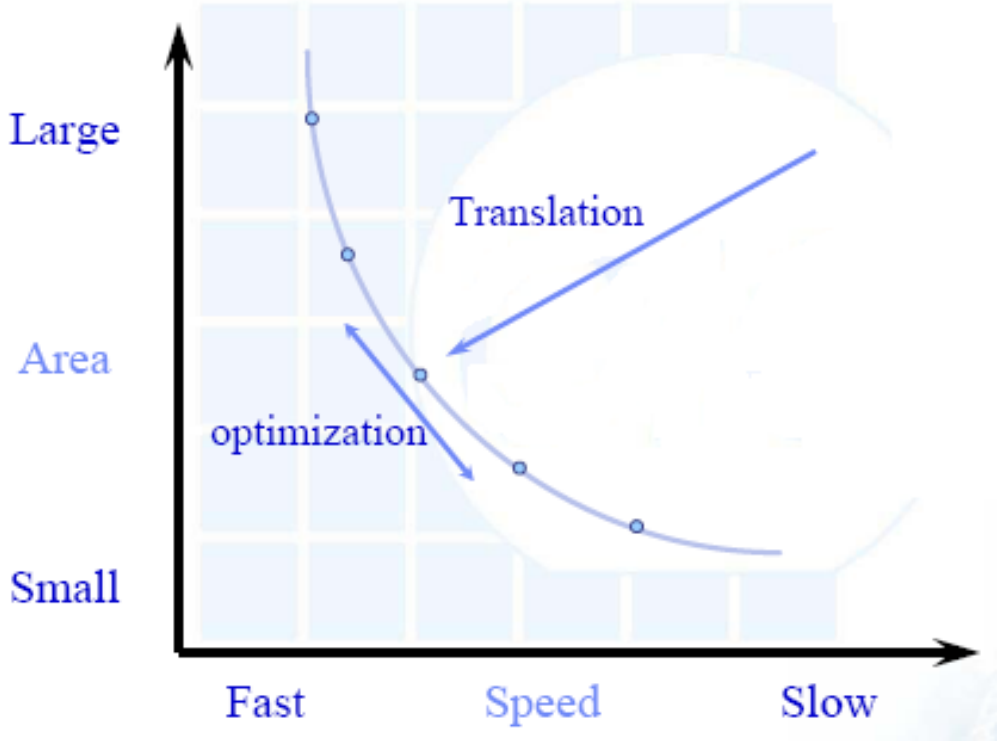
时序基础





$$T_{cycle} > T_d + T_{setup}$$

设计优化过程


Large
Area
Small
Translation
optimization
Fast
Speed
Slow

综合设计的优点

- 人工干预少，自动化程度高，用相对较高层次的抽象描述设计。
- 高层次的设计可以不用太关注电路的实现与限制。
- 从高层次抽象到门级电路转换快。
- 可以实现不同工艺的转移与设计的复用。

综合工具的弱点

- 时钟发生电路
- 存储器
- 专用宏单元

主要的综合工具

- Design Compiler (Synopsys)
- RTL Compiler (Cadence)
- Precision RTL （Menctor Graphics）、FPGA Compiler （Synopsys)、Synplify （Synplicity）

可综合RTL代码设计

- 可综合设计语法是Verilog语言的一个子集，使不同的RTL综合工具可以一致地实现相同的结果，并保证与仿真结果的相同。

- IEEE制定了专门的标准来规范Verilog语言的可综合设计，IEEE Standard for Verilog® Register Transfer Level Synthesis （IEEE1364.1-2002） （www.ieee.org)

可综合语法结构

Construct Type	Keyword or Description	Notes
ports	input, inout, output	inout应该在顶层的IO使用
parameters	parameter	设计更有继承性
module definition	module	
signals and variables	wire, reg, tri	支持向量
instantiation	module instances	避免使用基本门
function and tasks	function , task	忽略延时信息
procedural	always, if, else, case, casex, casez	不支持initial
procedural blocks	begin,end,namedblocks, disable	支持命名块的disable操作
data flow	assign	延时信息被忽略
named Blocks	disable	支持disable
loops	for, while, forever	While 和 forever 必须包含 @(posedge clk) 或者 @(negedge clk)

不可综合的语法结构

Construct Type	Notes
initial	只可用在Testbench设计中
events	用在Testbench设计中
real	不支持real类型
time	不支持time类型
force and release	不支持强制操作
assign deassign	不支持对于寄存器的强制操作
fork join	不支持，可以用非阻塞赋值实现相同的功能
primitives	不支持，多用于库单元描述
table	不支持，多用于库单元描述

不支持以下基本单元的描述： nmos， pmos，cmos，rnmos，rpmos，rcmos， pullup，pulldown， rtran， tranif0，tranif1， rtranif0， rtranif1

不支持===与！==操作

内置门

- 多输入门

  and、nand、or、nor、xor、xnor
- 多输出门

  buf、not
- 三态门

  bufif0、bufif1、notif0、notif1
- 上拉、下拉电阻

  pullup、pulldown
- MOS开关

  cmos、nmos、pmos、rcmos、rnmos、rpmos
- 双向开关

  tran、tranif0、tranif1、rtran、rtranif0、rtranif1

内置门说明

- 内置门的逻辑功能与对应的逻辑表达式相同，内置逻辑门与驱动强度（strength）一起，用于门级与开关级建模。

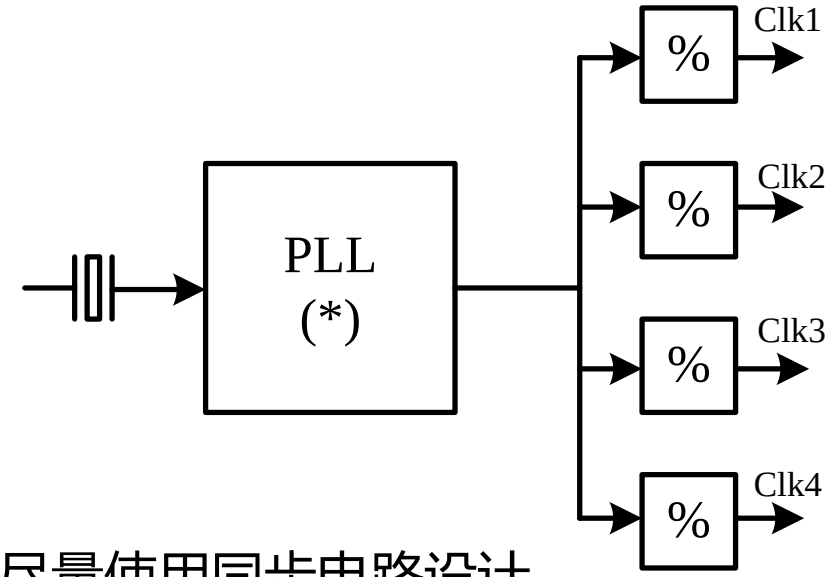
- 一般不在RTL级描述中使用，一般用于库单元，后端网表的建模。

RTL代码中的常见问题

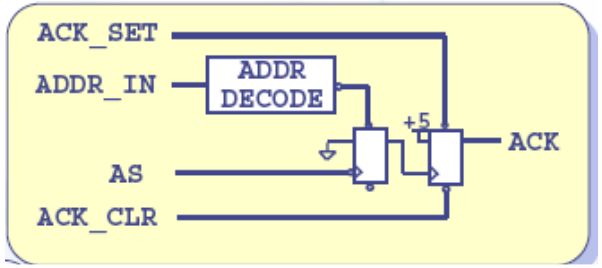
- 同步设计
- 电路reset
- 使用上升沿设计
- 仿真与综合结果的不一致

使用同步设计

- 尽量使用一个时钟，或者同一个时钟源的时钟。



同步电路可以稳定的工作，尽量使用同步电路设计。

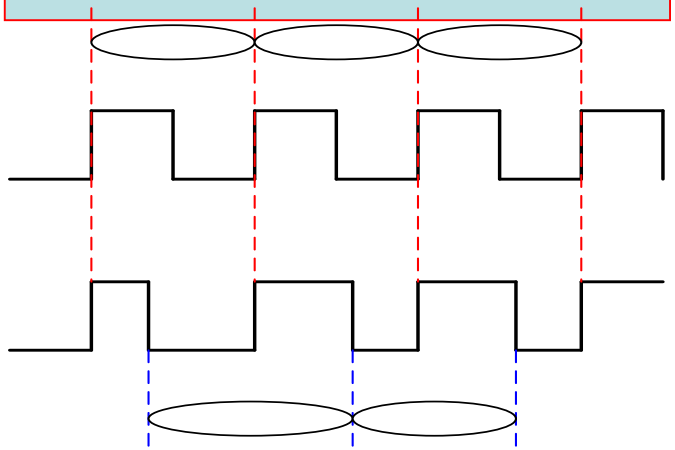


使用上升沿设计

- 在设计中，使用一个时钟沿工作。

好的风格

```
…
always @(posedge clk)
begin
…
end
```



无法保证duty cycle，要避免的设计

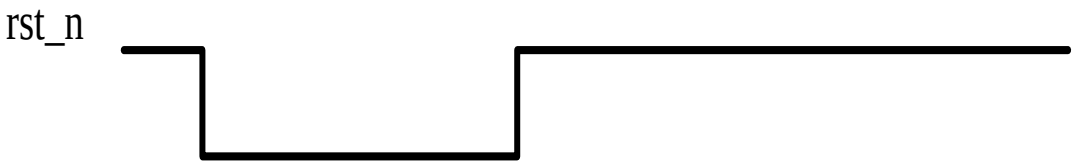
```
…
always @(posedge clk)
begin
…
end

always @(negedge clk)
begin
…
end
```

电路reset

- 在电路设计中要求有一个初始态，在电路工作前初始化所有的寄存器（组合逻辑的输入可以溯源自寄存器的输出）。

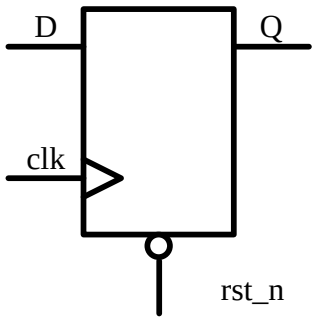
- 异步reset



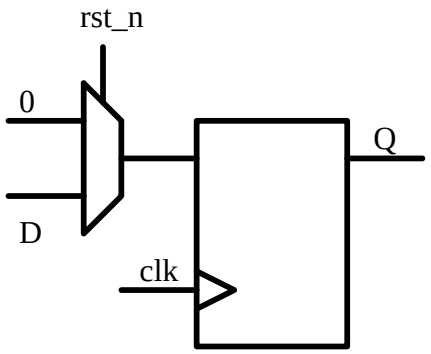
- 同步reset

异步与同步reset

```
…
always @(posedge clk or negedge rst_n)
If(~rst_n)
…
else
…
```




```
…
always @(posedge clk)
If(~rst_n)
…
else
…
```



常见的错误

```
initial
begin
a=1'b0;
b=1'b0
…
end
always @(posedge clk or
negedge rst_n)
if(~rst_n)
a<=1'b0
else
a<=b;
```

```
always @(negedge rst_n)
if(!rst_n)
begin
a=1'b0;
b=1'b0
…
end
always @(posedge clk or
negedge rst_n)
if(~rst_n)
a<=1'b0
else
a<=b;
```

阻塞与非阻塞

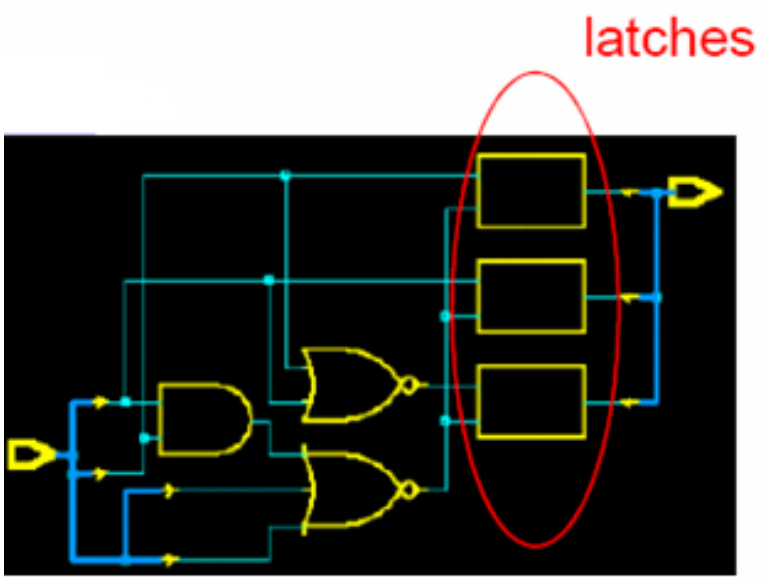
- 在过程中，用阻塞描述组合逻辑。
- 在过程中，用非阻塞描述时序逻辑。
- 在设计中，尽量将时序逻辑和组合逻辑分在不同的过程块中描述。

Latch的推断（1）

- case语句中使用default分支

```
always @(bcd) begin
case (bcd)
4'd0:out=3'b001;
4'd1:out=3'b010;
4'd2:out=3'b100;
endcase
end
```

```
always @(bcd) begin
case (bcd)
4'd0:out=3'b001;
4'd1:out=3'b010;
4'd2:out=3'b100;
default:out=3’bxxx;
endcase
end
```



Latch的推断（2）

- if else语句中描述所有的分支情况

```
module if_ex (A, B, C, D, SEL, MUX_OUT);
input A, B, C, D;
input [1:0] SEL;
output MUX_OUT;
reg MUX_OUT;
always @ (A or B or C or D or SEL)
begin
if (SEL == 2'b00)
MUX_OUT = A;
else if (SEL == 2'b01)
MUX_OUT = B;
else if (SEL == 2'b10)
MUX_OUT = C;
else
MUX_OUT = 0;
end
endmodule
```

补充内容1

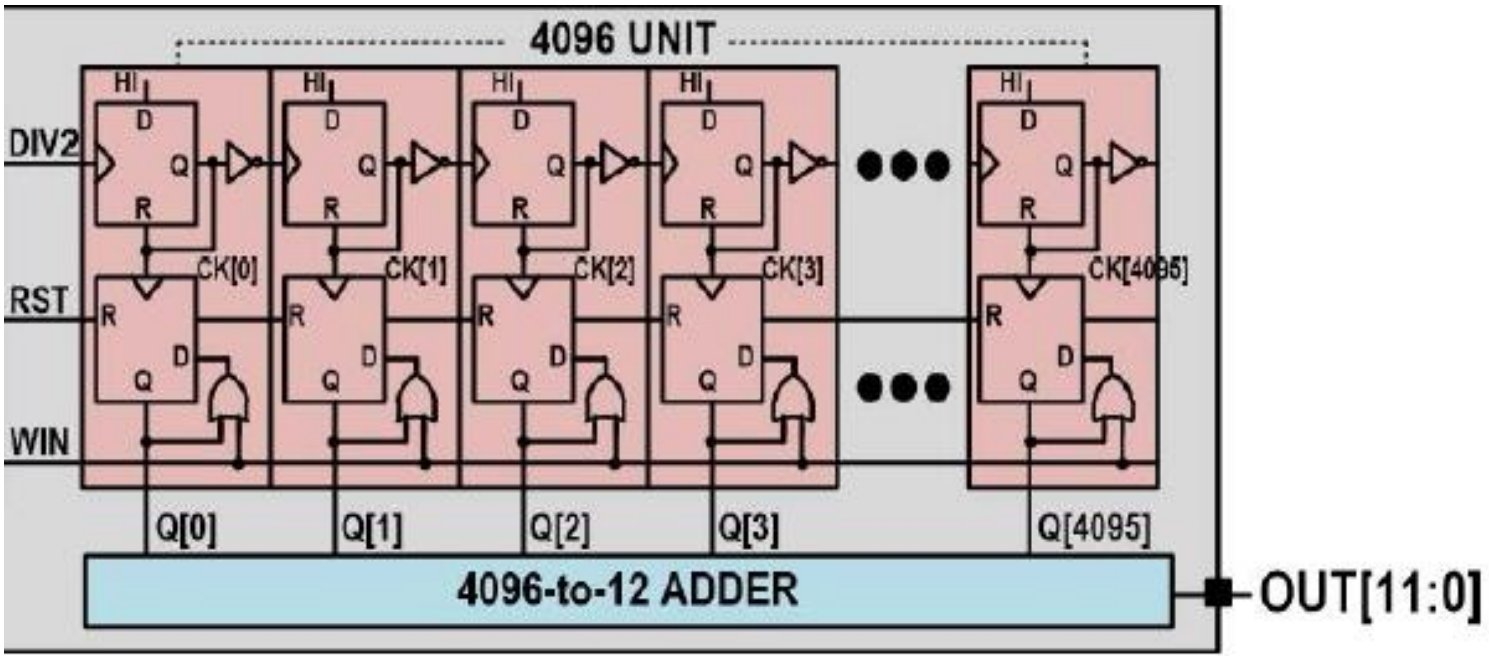
- define：常用于定义常量，可以跨模块、跨文件;
  作用范围适合整个工程;

- parameter：用于常量定义;
  本module内有效的定义;
  用于模块间常量传递;

- localparam：用于常量定义;
  本module内有效的定义;
  不可用于常量量传递;

补充内容2

- 问题提出


4096 UNIT
DIV2
RST
WIN
HI
D
Q
R
CK[0]
CK[1]
CK[2]
CK[3]
CK[4095]
Q[0]
Q[1]
Q[2]
Q[3]
Q[4095]
4096-to-12 ADDER
OUT[11:0]

generate

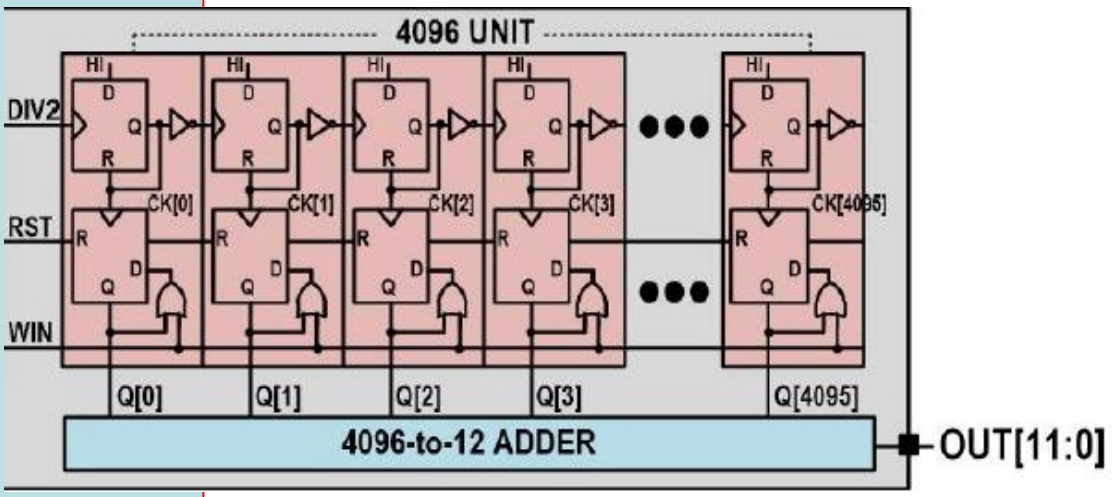
- Verilog generate 语句是用于编写
  - 可配置、可综合RTL的强大构造;
  - 用于创建模块和代码的多个实例化;
  - 有条件地实例化代码块。

```
genvar 循环变量名;
generate
   // generate循环语句
   // generate 条件语句
   // generate 分支语句
   // 嵌套的generate语句
endgenerate
```

Verilog-2001增加了四个关键字generate，endgenerate, genvar, localparam，genvar是一个新增的数据类型，用在generate的循环中的标尺变量必须定义为gnevar类型;

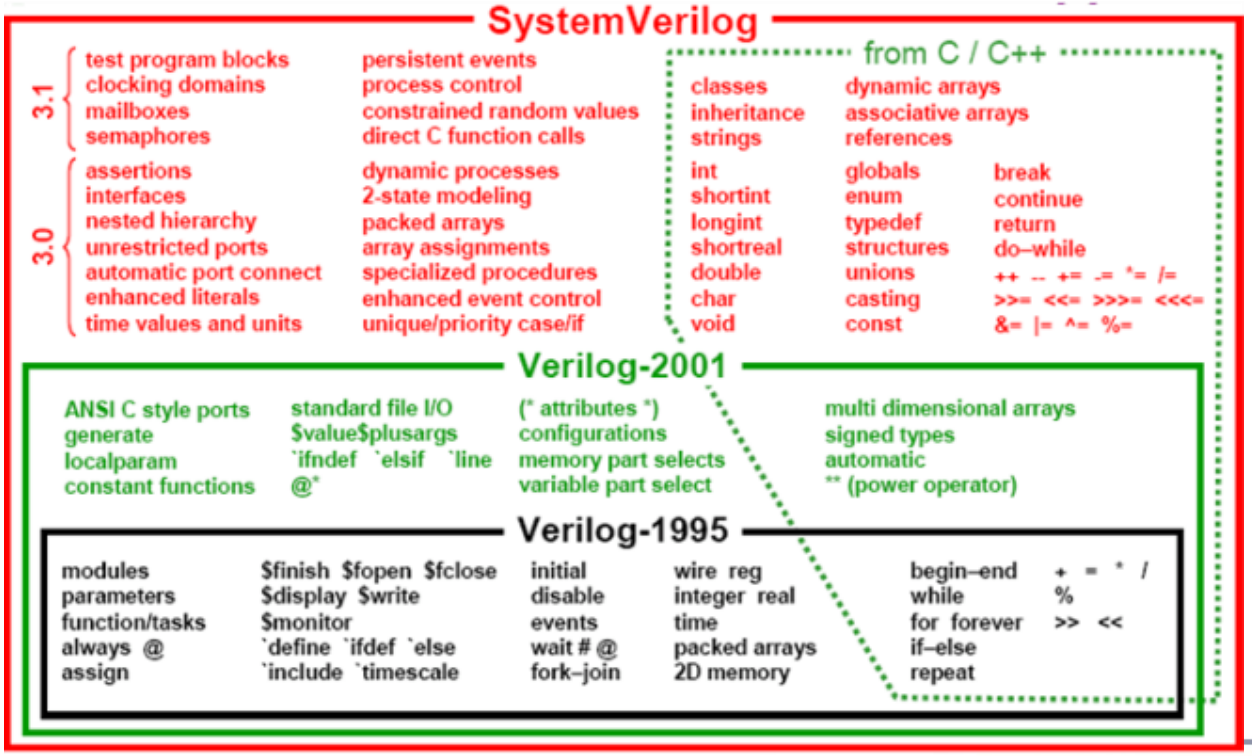
generate

```
genvar ff_id;
generate    //first flipflop array
for(ff_id=0; ff_id<TDCW; ff_id=ff_id+1)
begin
if(ff_id==0)
d_flipflop   i_dff (
                    .clk(clkx2)//clk
                   ,.reset(ck[ff_id])
                   ,.d(1'b1)
                   ,.q(ck[ff_id])
                   ,.qn(qn[ff_id]));
else
d_flipflop  i_dff (
                     .clk(qn[ff_id-1])
                    ,.reset(ck[ff_id])
                    ,.d(1'b1)
                    ,.q(ck[ff_id])
                    ,.qn(qn[ff_id]));
end
endgenerate
```



Systemverilog

- IEEE_std_1364_1995 (Verilog) (675 pages)
- IEEE_std_1364_2001 (Verilog)
- IEEE_std_1364_2005 (Verilog)
- IEEE_std_1800_2012 (System Verilog)
- IEEE_std_1800_2017 (System Verilog) (1315 pages)



课程回顾

- Verilog描述基础
- 组合逻辑与行为建模
- 时序辑与Testbench设计
- 状态机设计
- 高级设计话题
- 可综合设计基础