

# ***Achieving Timing Closure***

**John Li**

# Agenda

---

- **Timing closure**的概念
- **Timing closure**的步骤
- 采用合适的**Coding Style**
- 进行适当的综合约束
- 管脚锁定
- 实施**Lattice constrains**
- **Map**
- 布局布线
- 控制**place and route**
- **Floorplanning the design**

# Timing closure的概念

---

- 当前FPGA的设计规模越来越大，复杂程度日益增加，同时要求系统的Perfromace也越来越高。
- 获得Timing目标越来越困难.
- 设计者必须采用各种技术提升系统性能以满足设计的Timing要求.



# Timing closure procedure

---

- 1.采用合适的**coding style**
- 2.进行适当的综合约束
- 3.管脚锁定
- 4.实施**Lattice constrains**
- 5.**Map**
- 5.布局布线
- 7.控制**place and route**
- 8.**Floorplanning the design**

# 采用合适的coding style

---

关于提升FPGA系统性能，工程师最容易想到的方法就是通过进行综合约束、布局布线约束、和其他的优化技术提升系统性能，当然这些都是设计过程中所必需的，但所有这些优化方法对于系统性能的提升都是有限的，系统的性能最终还是取决于工程师的设计（**coding style**），其中同步设计是最重要的一点。下面讨论一些具体的**coding** 技术，合理的运用这些**coding**技术能够尽可能的减小两级寄存器之间的延时从而获得更高的系统速度。

## -通用的coding style

### --Hierarchical Coding

- Team Based的设计：多个工程师可以同时参与到一个复杂设计中来。
- 加速设计和编译过程：关键模块可以单独修改而不会影像整个设计。
- 缩短设计周期：重复利用成熟模块。
- 模块可以容易被工程师理解和维护。
- 缺点：如果模块划分不合理，特别是模块边界设计处理不当会影响FPGA的资源利用率和最终的系统性能。

# 采用合适的coding style

---

---上述缺点可以通过细致的hierarchy设计来克服.

---hierarchical design 需遵循的规则:

----top level模块仅仅应该包含 instantiation statement , 即在顶层模块中调用子模块。

----任何I/O instantiation 应当包含在 top level模块中。

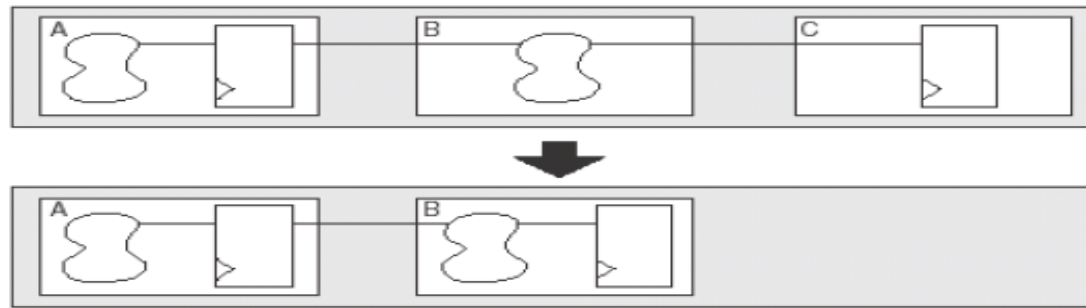
----任何输入输出器件的信号应当在top level模块中声明为: input、output和bi-directional Pin.

## --Design partitioning

---在sub module中register所有输出, 以保证所有sub module之间为同步设计, 获得更好的系统performance。



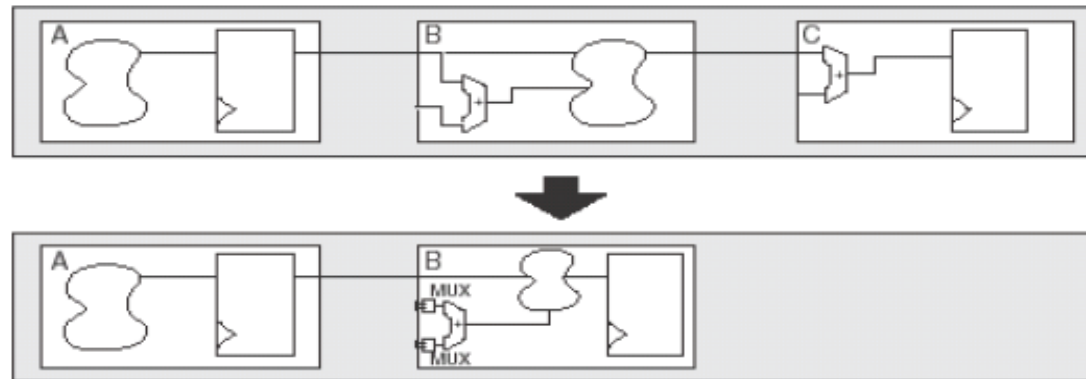
## 采用合适的coding style



---保证相关逻辑和共享资源在同一个模块中实现。

这样可以做到更好的资源共享，综合工具只能针对一定数量的逻辑进行优化；综合工具可以在一个模块内部优化整个关键路径；跨模块的关键路径也不会被有效的优化。

Figure 56: Merge Sharable Resource in the Same Block



# 采用合适的coding style

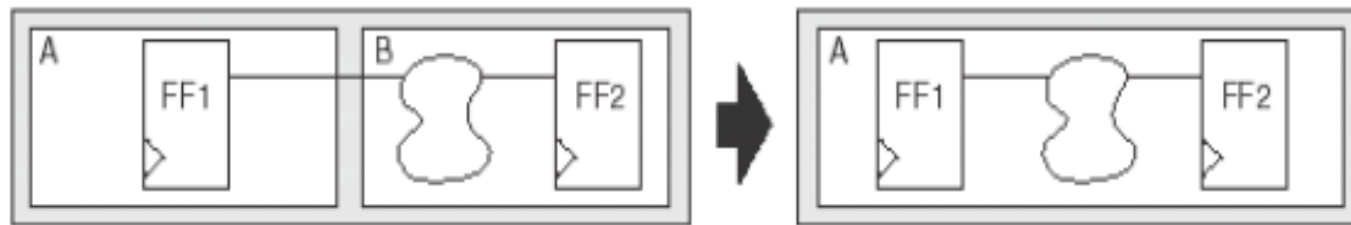
---针对不同的优化目标来划分模块

----分离关键路径和非关键路径可以获得更好的综合效果。

----设计者应该在充分考虑性能需求和资源需求的基础上进行逻辑设计. 针对不同的模块采用不同的优化策略，以避免相互影响。

---对于那些并不需要high performance的模块应该放松约束以节省和预留关键资源给关键路径。

Figure 57: Logic with the Same Relaxation Constraint



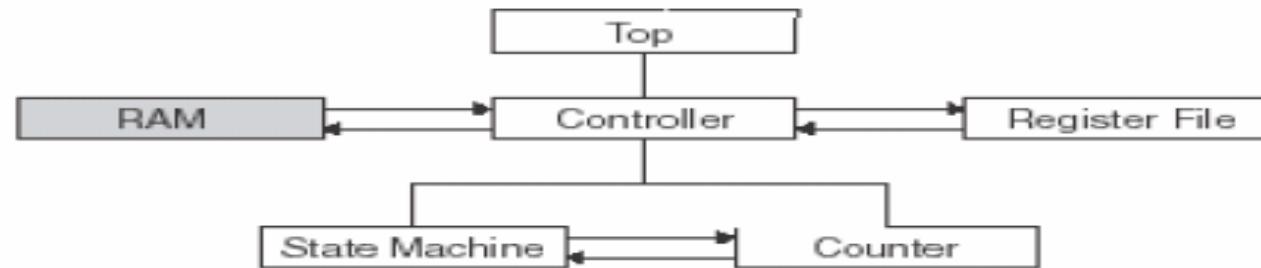


# 采用合适的coding style

---在单独的模块中保存实例化代码：

可以非常方便的在RAM行为仿真模型和实际的RAM块代码之间进行切换。

Figure 58: Separate RAM Block



---每个Module的规模在30~80 PFU：小模块由于资源有限不利于综合工具实施“resource sharing”算法；规模太大的模块一旦更改其中的一小部分就会导致整个模块重新综合，影响到一些不必要的逻辑，增大综合运行时间。

## --design registering

---利用流水设计提高系统性能，把一个较长的路径分割为多个短路径，并在多个时钟周期完成。

# 采用合适的coding style

Figure 59: Before Pipelining

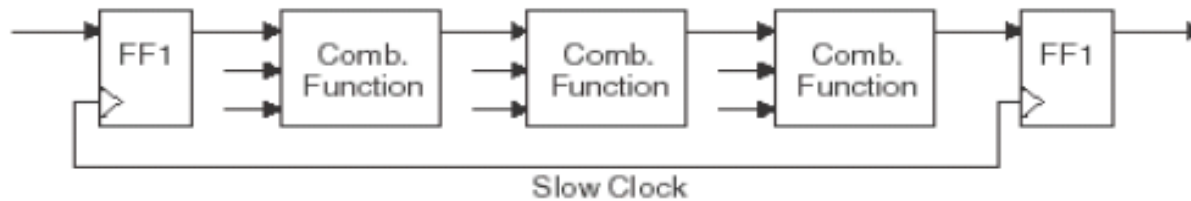
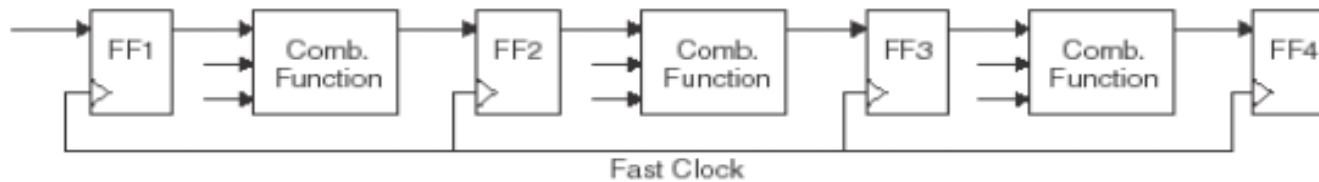


Figure 60: After Pipelining



---IF statement and CASE statement比较

----IF-THEN-ELSE statement 生成优先级的编码逻辑;CASE statement 实现balance逻辑.

----如果每个解码条件相对独立,这两种声明方式实现的功能是一样的。

# 采用合适的coding style

```
-- Case Statement — mutually exclusive conditions
process (s, x, y, z)
begin
  O1 <= '0';
  O2 <= '0';
  O3 <= '0';
  case (s) is
    when "00" => O1 <= x;
    when "01" => O2 <= y;
    when "10" => O3 <= z;
  end case;
end process;
```

```
-- If-Then-Else — mutually exclusive conditions
process (s, x, y, z)
begin
  O1 <= '0';
  O2 <= '0';
  O3 <= '0';
  if s = "00" then O1 <= x;
  elsif s = "01" then O2 <= y;
  elsif s = "10" then O3 <= z;
  end if;
end process;
```

- IF statement的主要缺陷：使设计不必要的复杂化；需要额外的逻辑来构建优先级Tree。
- 如果解码条件不是相对独立的, 那末IF-THEN-ELSE结构会导致最低优先级的output依赖于所有的控制条件。



## 采用合适的coding style

```
--A: If-Then-Else Statement: Complex O3 Equations
process(s1, s2, s3, x, y, z)
begin
    O1 <= '0';
    O2 <= '0';
    O3 <= '0';
    if s1 = '1' then
        O1 <= x;
    elsif s2 = '1' then
        O2 <= y;
    elsif s3 = '1' then
        O3 <= z;
    end if;
end process;
```

```
--B: If-Then-Else Statement: Simplified O3 Equation
process (s1, s2, s3, x, y, z)
begin
    O1 <= '0';
    O2 <= '0';
    O3 <= '0';
    if s1 = '1' then
        O1 <= x;
    end if;
    if s2 = '1' then
        O2 <= y;
    end if;
    if s3 = '1' then
        O3 <= z;
    end if;
end process;
```

equation for O3 output in example A is:

```
O3 <= z and (s3) and (not (s1 and s2));
```

If the same code can be written as in example B, most of the synthesis tools will remove the priority tree and decode the output as:

```
O3 <= z and s3;
```

# 采用合适的coding style

---

---如果output 不需要优先级控制, 最好使用CASE statement, 因为CASE声明中每个分支的优先级是一样的, 每个分支output是并发的。

## --避免不必要的Latch

---综合工具会引入Latch, 如果存在不完全的条件表达式: 象IF-THEN-ELSE声明中没有else子句.

---Latch会需要额外的资源, 同时引入组合反馈环从而产生异步时序问题。

---non-intended latch 是可以避免的:

使用时钟寄存器 或遍历所有的输入条件assign output 或是使用else (when others) 作为最后的子句

## 采用合适的coding style

```
LIBRARY ieee;
USE IEEE.std_logic_1164.all;

ENTITY nolatch IS
    PORT (a,b,c: IN STD_LOGIC;
          sel: IN STD_LOGIC_VECTOR (4 DOWNT0 0);
          oput: OUT STD_LOGIC);
END nolatch;

ARCHITECTURE rtl OF nolatch IS
BEGIN
    PROCESS (a,b,c,sel) BEGIN
        IF sel = "00000" THEN
            oput <= a;
        ELSIF sel = "00001" THEN
            oput <= b;
        ELSIF sel = "00010" THEN
            oput <= c;
        ELSE
            --- Prevents latch inference
            oput <= 'X'; --/
        END IF;
    END PROCESS;
END rtl;
```

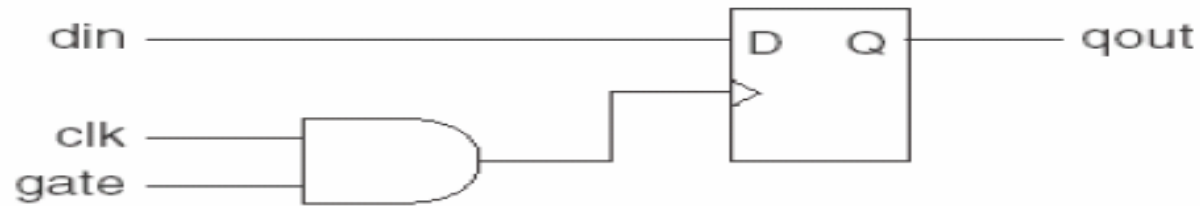


# 采用合适的coding style

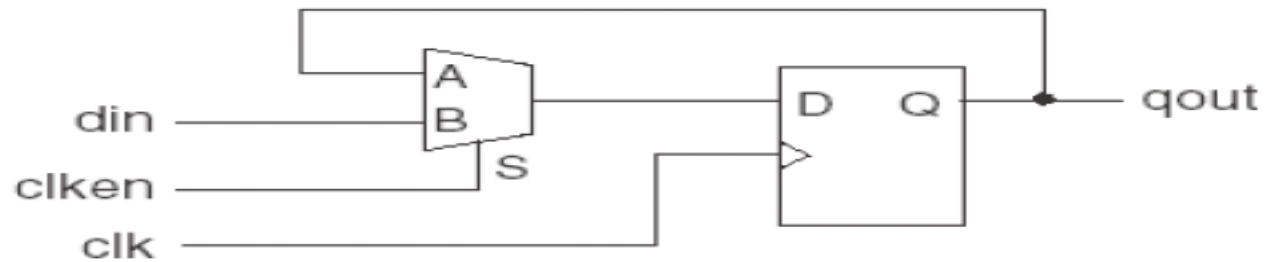
--用时钟使能信号代替门控时钟

---门控时钟会带来很多时序问题，最典型的是clock skew.

**Figure 62: Asynchronous: Gated Clocking**



**Figure 63: Synchronous: Clock Enabling**



# 采用合适的coding style

Figure 64: Clock Enable Coding

## VHDL

```
Clock_Enable: process (clk, clken, din)
begin
    if (clk'event and clk = '1') then
        if (clken = '1') then
            qout <= din;
        end if;
    end if;
end process Clock_Enable;
```

## Verilog

```
always @(posedge clk)
    qout <= clken ? din : qout;
```

---时钟使能guideline in Lattice FPGA:

----Clock enable 只在触发器模式支持，Latch模式不支持.

----在一个slice中的触发器对共享一个Clock enable信号.

----所有的触发器支持positive clock enable输入.

----默认的时钟使能的优先级高于同步set/reset. 但是可以编程实现set/reset 的优先级高于clock enable.

# 采用合适的coding style

**Figure 65: Clock Enable over Synchronous LSR**

## VHDL

```
COUNT8: process (CLK, GRST)
begin
    if (GRST = '1') then
        cnt <= (others => '0');
    elsif (CLK'event and CLK = '1') then
        if (CKEN = '1') then
            cnt <= cnt + 1;
        elsif (LRST = '1') then
            cnt <= (others => '0');
        endif;
    endif;
end process COUNT8;
```

## Verilog

```
always @(posedge CLK or posedge GRST)
begin
    if (GRST)
        cnt = 4'b0;
    else if (CKEN)
        cnt = cnt + 1'b1;
    else if (LRST)
        cnt = 4'b0;
end
```

**Figure 66: Synchronous LSR over Clock Enable**

## VHDL

```
COUNT8: process (CLK, GRST)
begin
    if (GRST = '1') then
        cnt <= (others => '0');
    elsif (CLK'event and CLK = '1') then
        if (LRST = '1') then
            cnt <= (others => '0');
        elsif (CKEN = '1') then
            cnt <= cnt + 1;
        endif;
    endif;
end process COUNT8;
```

## Verilog

```
always @(posedge CLK or posedge GRST)
begin
    if (GRST)
        cnt = 4'b0;
    else if (LRST)
        cnt = 4'b0;
    else if (CKEN)
        cnt = cnt + 1'b1;
end
```



# 采用合适的coding style

---

## --针对分布式Memory的coding style

---不推荐使用Library Primitive去构建RAM或FIFO, 这是因为RAM块在各个系列FPGA中的structure都是唯一的. 综合工具并不能优化处理RAM的实现, 会产生低效率的网表 (for device fit) .

---强烈建议使用Iplexpress来构建RAM/FIFO.

## --针对高扇出网络控制综合工具

---Lattice FPGA 架构针对信号高扇出做了设计, 比如具有丰富的一级时钟和二级时钟资源以及全局复位GSR布线资源等等.

---综合工具在综合时会复制逻辑资源以减小扇出, 这样会导致占用过多的资源同时checking performance也会变得困难。

---综上所述在综合过程中有必要对fan out进行控制.

# 采用合适的coding style

## --双向buffer

---双向buffer可以和以和普通I/O一样在顶层文件中实例化.

---另一种方式就是在代码中实现.

**Figure 74: Verilog HDL RTL for Bidirectional Buffer**

```
module bireg (datain, clk, en_o, Qo1, Qio);
  input  [7:0] datain;
  input  clk, en_o;
  output [7:0] Qo1;
  inout  [7:0] Qio;
  reg    [7:0] Q_reg;
  reg    [7:0] Qio_int;
  wire   [7:0] Qo1;
  wire   [7:0] Qio;
  always @(posedge clk)
  begin
    Q_reg = datain;
  end
  always @(en_o or Q_reg)
  begin
    if (en_o)
      Qio_int <= Q_reg;
    else
      Qio_int <= 8'hz;
    end
    assign Qio = Qio_int;
    assign Qo1 = Qio;
  endmodule
```

## 采用合适的coding style

**Figure 75: VHDL RTL for Bidirectional Buffer**

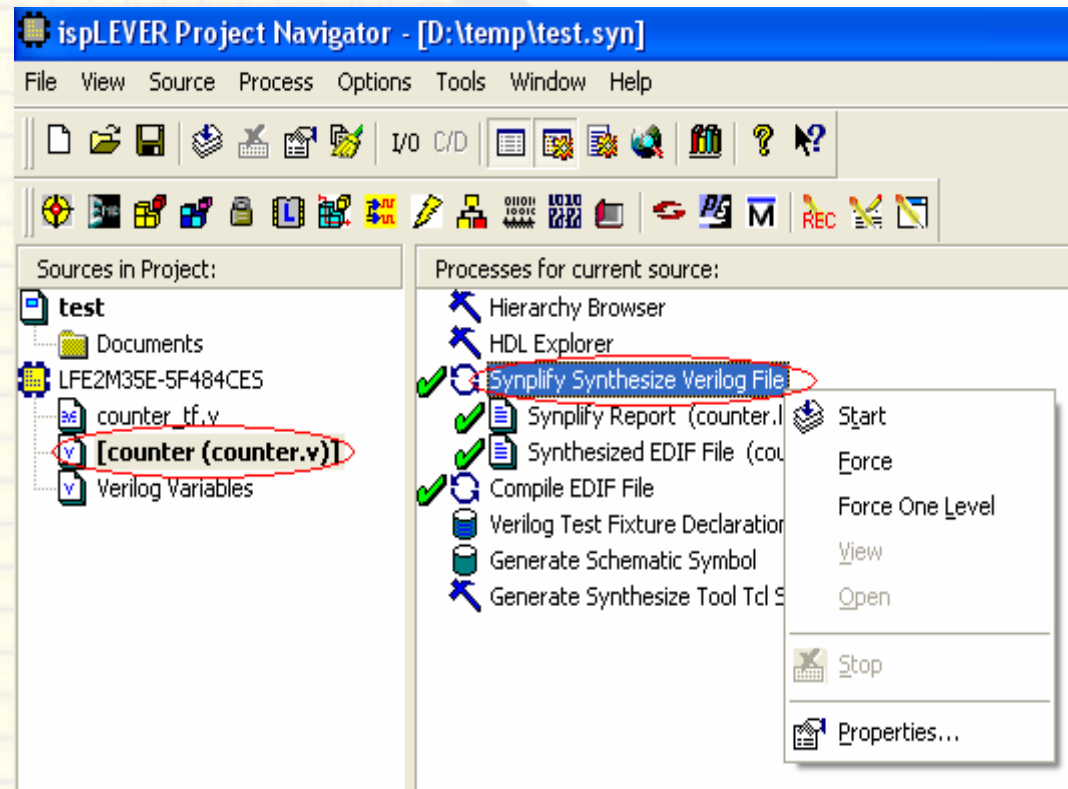
```
library ieee;
use ieee.std_logic_1164.all;

entity bireg is port (
    datain : in std_logic_vector (7 downto 0);
    clk,en_o : in std_logic;
    Qo1 : out std_logic_vector (7 downto 0);
    Qio : inout std_logic_vector (7 downto 0));
end bireg;
architecture beh of bireg is
    signal Q_reg : std_logic_vector (7 downto 0);
    signal Qio_int : std_logic_vector (7 downto 0);
begin
    process(clk,datain) begin
        if clk'event and clk = '1' then
            Q_reg <= datain;
        end if;
    end process;
    process(Q_reg,en_o) begin
        if en_o = '1' then
            Qio_int <= Q_reg ;
        else
            Qio_int <= (others=>'Z');
        end if;
    end process;
    Qio <= Qio_int;
    Qo1 <= Qio;
end;
```



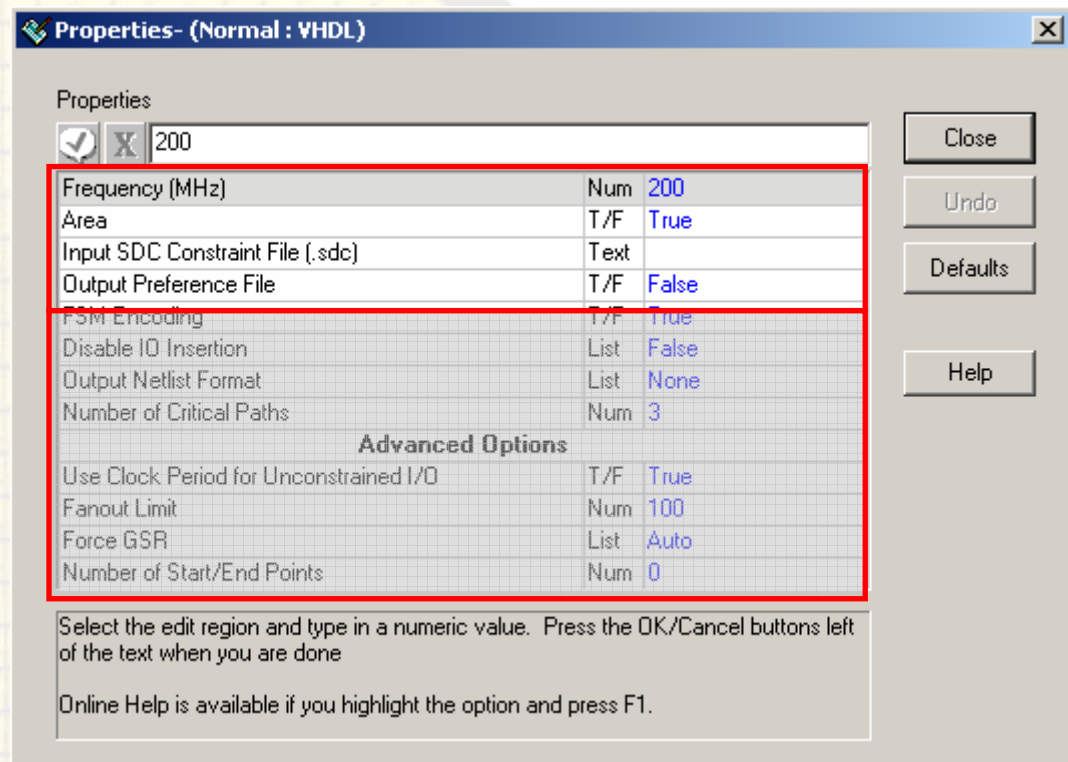
## 控制综合工具

- 在 project navigator 中选择顶层文件
- 在 Process for current source 中选择 synplify synthesize verilog file 右击鼠标
- 选择 Properties



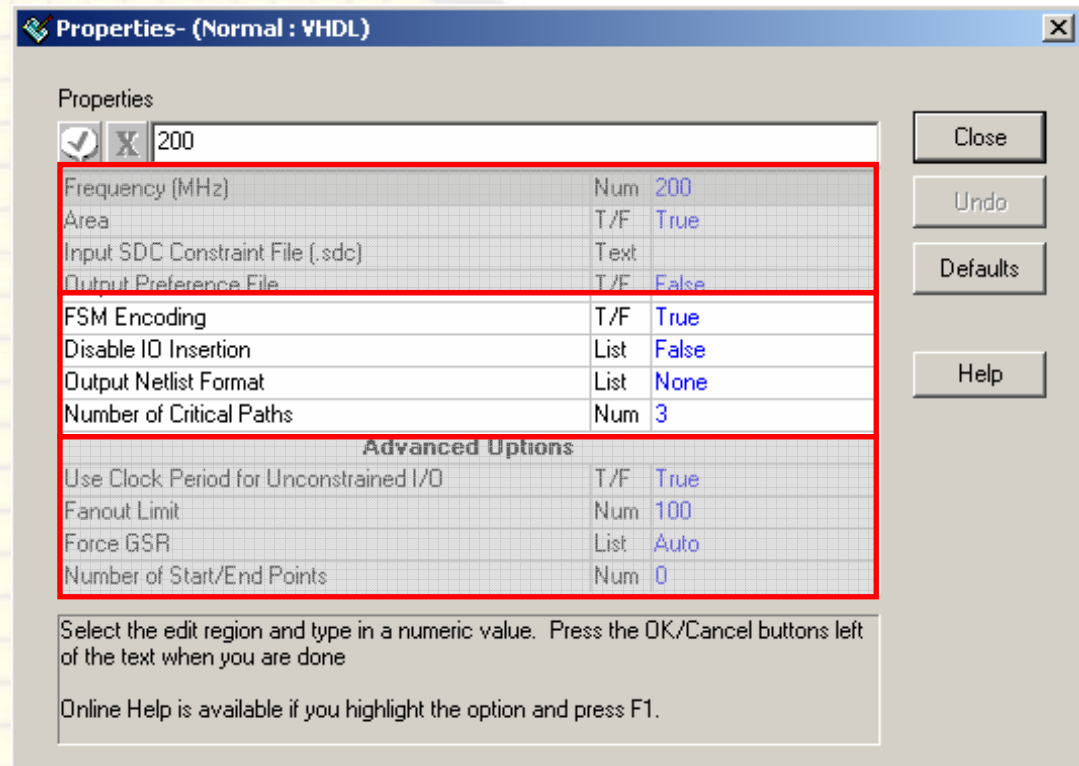
## 控制综合工具

- **Frequency** 选项中指定综合频率. 通常指定它为实际工作频率的两倍。
- 指定综合工具是执行面积优先还是速度优先算法。
- **SDC** 约束文件是 **Synplify** 的约束文件, 可以在这里引入来控制综合过程。
- 是否基于综合的约束产生.PRF文件. 该文件可以拷贝到Map后生成的.PRF文件中控制Place&Route



## 控制综合工具

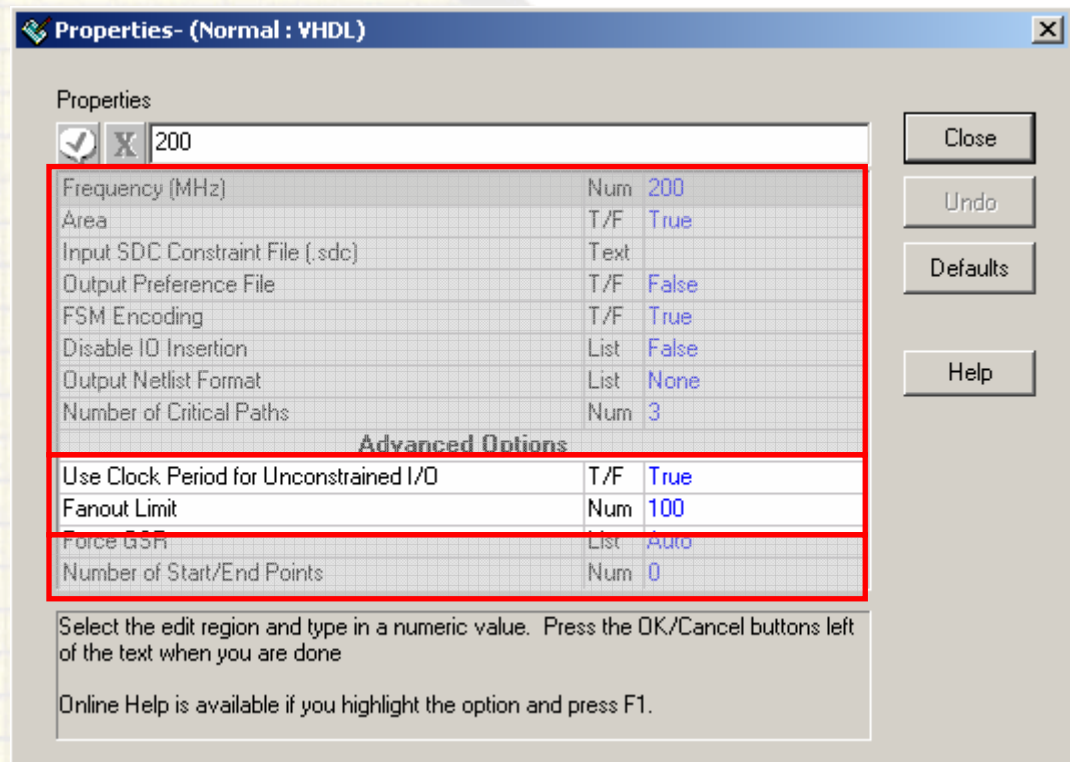
- Enables/disables FSM Compiler 控制状态机的综合. 设为 **True** (默认)时, FSM Compiler 会自动识别并优化设计中的状态机 (状态据数量小于5: One Hot ; 大于5小于16: binary ; 大于16: grey)
- Disable IO Insertion removes the IO buffer (比如综合IP或是黑盒文件时)
- 指定输出网表类型(None、VHDL, Verilog).
- 指定在时序报告中报告的关键路径的数目.





# 控制综合工具

- Enables/disables 未约束的IO是否要用clock period去约束它的 input to register延时和register to output 延时.
- 控制综合过程fan out. 当信号fan out 达到该限制时逻辑会被自动复制.



# 控制综合工具

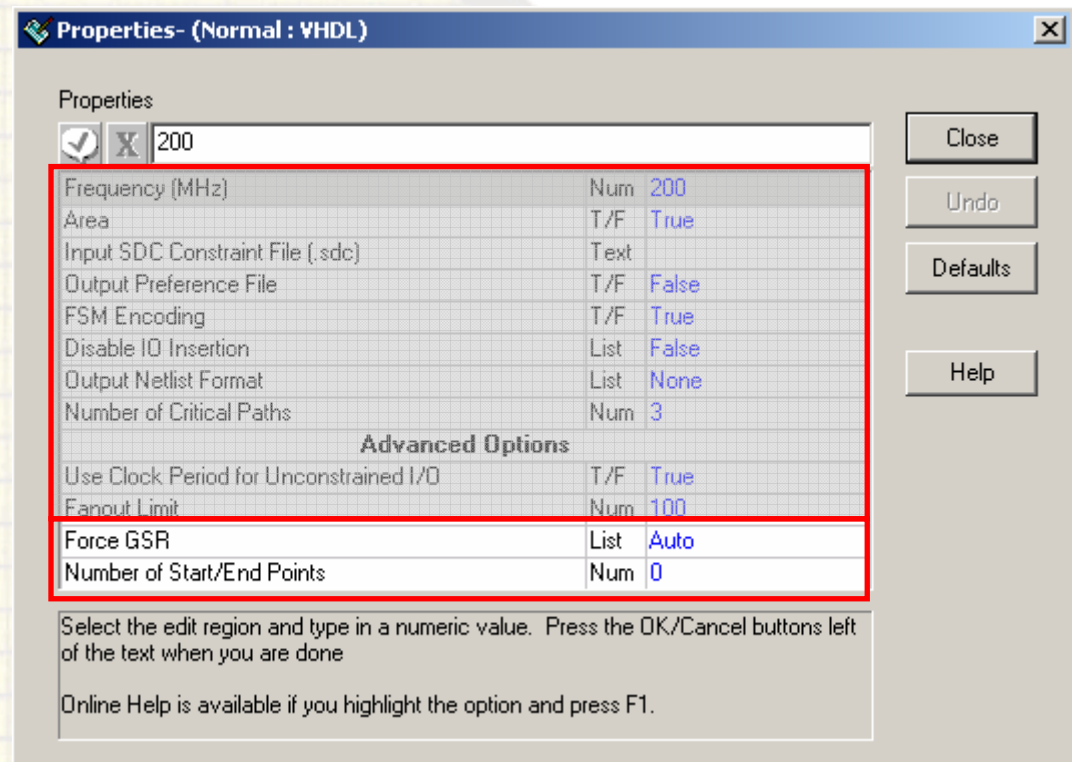
- 选择是否在设计中使用**GSR**布线资源:

**Auto** – 软件自己判定是否使用**GSR**布线资源.

**True** – 综合工具会永远使用**GSR**.

**False** – 不允许综合工具使用**GSR**布线资源.

- 软件在时序报告中报告关键路径时起点和终点的数目.



## 控制综合工具

- 选择**Precision** 作为综合工具时：一些 **Advanced** 选项有所区别。
- 是否使能**Retiming**。
- 约束全局的input delay ns。
- 约束全局的out put delay ns。
- 设置该选项为 **True**， 综合工具会把 **Set/Reset on DFF** 转换为**Latches**来实现( 当FPGA内部硬件架构不能同时支持set/reset时使用)。
- 设置该选型 **True** 软件会报告所有的时钟频率。
- 指定在**timing report**中**timing summary path**的数目。



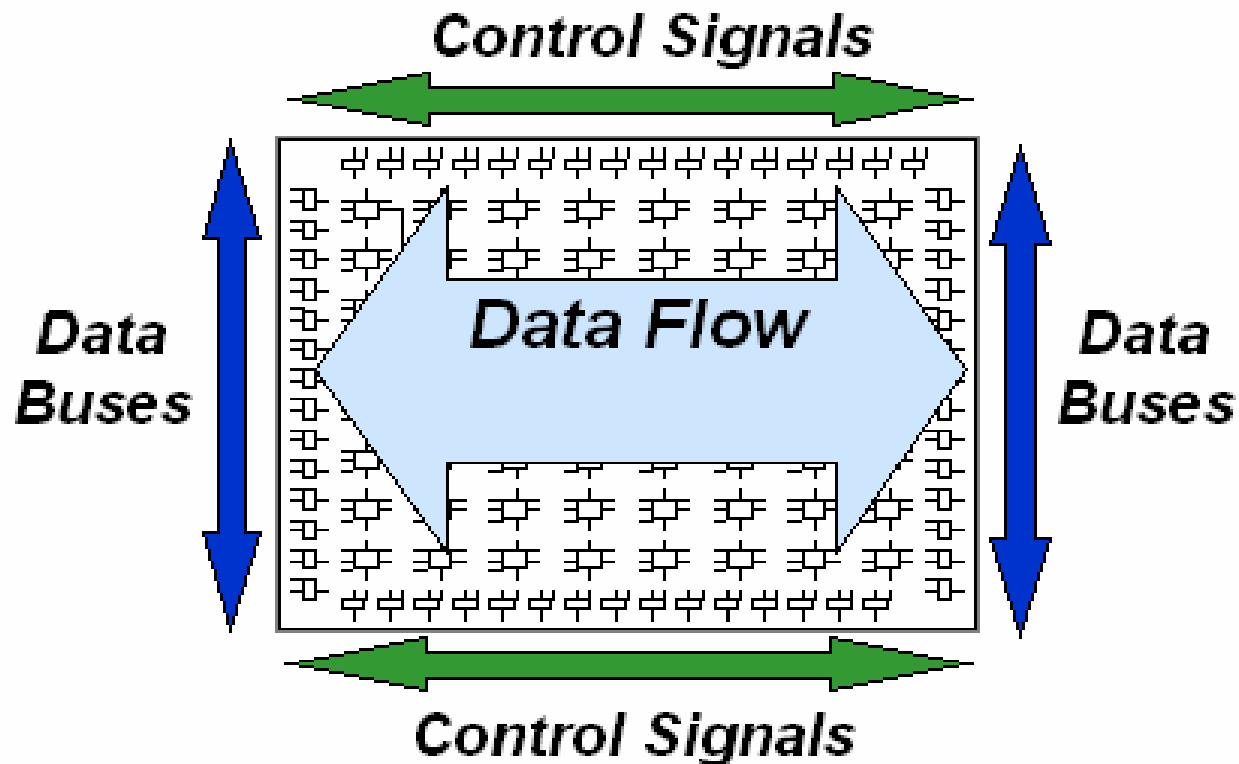
# 管脚锁定

管脚约束应该在单板的设计早期就开始考虑，综合考虑单板布局，**FPGA**内部架构和逻辑资源位置，以及业务数据流向来决定管脚的位置约束。

- 影响管脚锁定的因素: **PCB layout**, 信号完整性, 逻辑设计（数据流方向）, **FPGA**架构和资源分部等等。
- 很多设计都有一个大致的数据流方向，设计者要充分了解**FPGA**内部资源以及其信号流向，使这数据流向符合**FPGA**内部信号的流向，减小数据path延时从而获得更好的系统性能。
- 其他类型的逻辑 (控制、状态等等)需要位置集中布置使之易于满足时序要求。
- 还有一些逻辑可能需要在整个die内散布开，这样才会离源或目的接口较近。
- 我们需要做的是充分考虑整体逻辑设计，把数据路径（**Path**）和控制逻辑区分开。

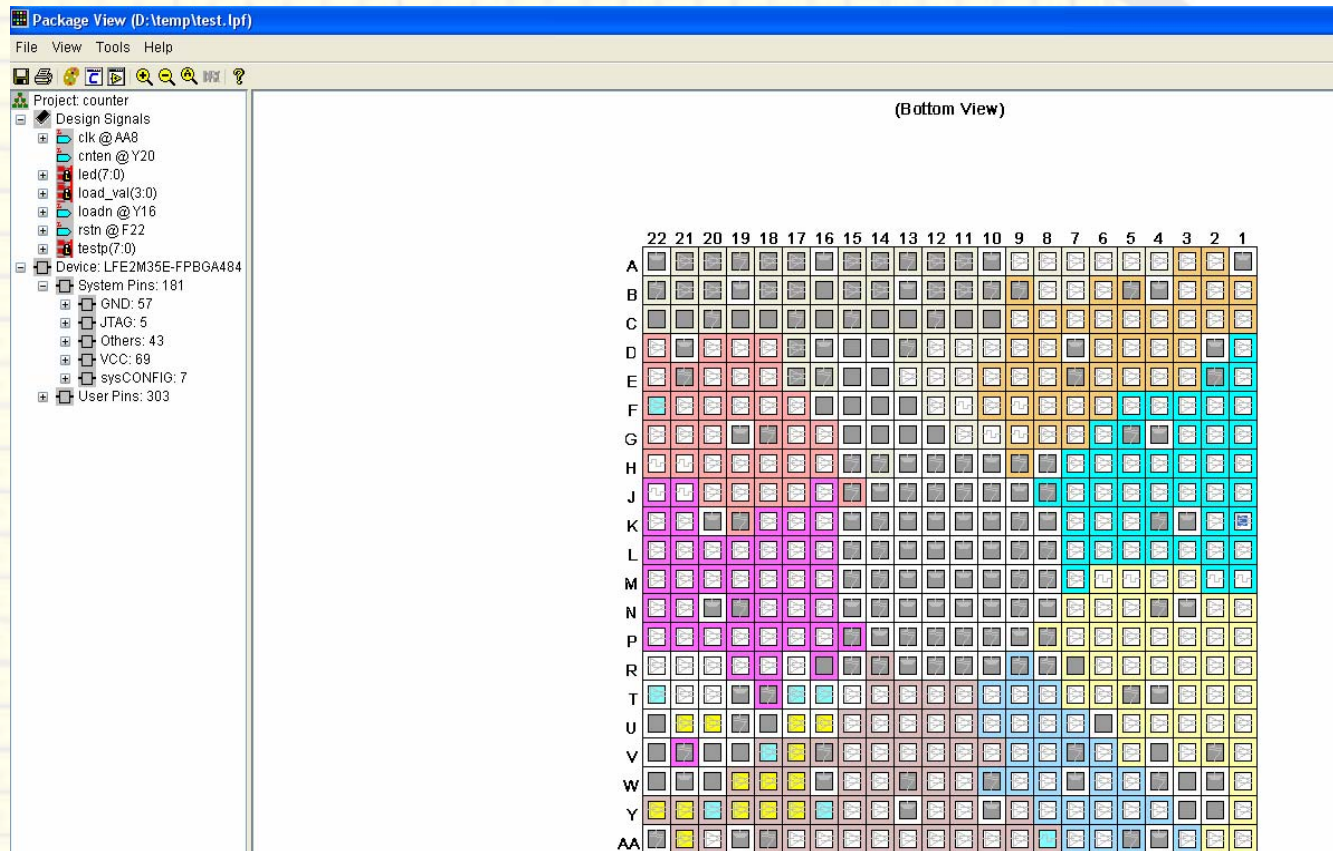
## 管脚锁定

- FPGA Fabric设计的比较适合水平数据流 (左到右的数据流,或右到左的数据流). 进位链垂直分布, 实现算术逻辑时 LSB在PFU列的bottom, MSB在PFU列的TOP端.
- Lattice FPGA基本的数据流如下图:



## 管脚锁定

- 其他需要考虑的因素：在8个Bank中电平标准的支持；差分标注的支持；高速串行IO (LVDS, LVPECL)的支持；管脚差分对的支持等等。
- **Package view**提供了一个图形化界面. **Bank**用不同的颜色来区分，设计者可以很方便的拖拽一个信号到指定管脚，所有双功能管脚、电源和地等等都很容易识别。

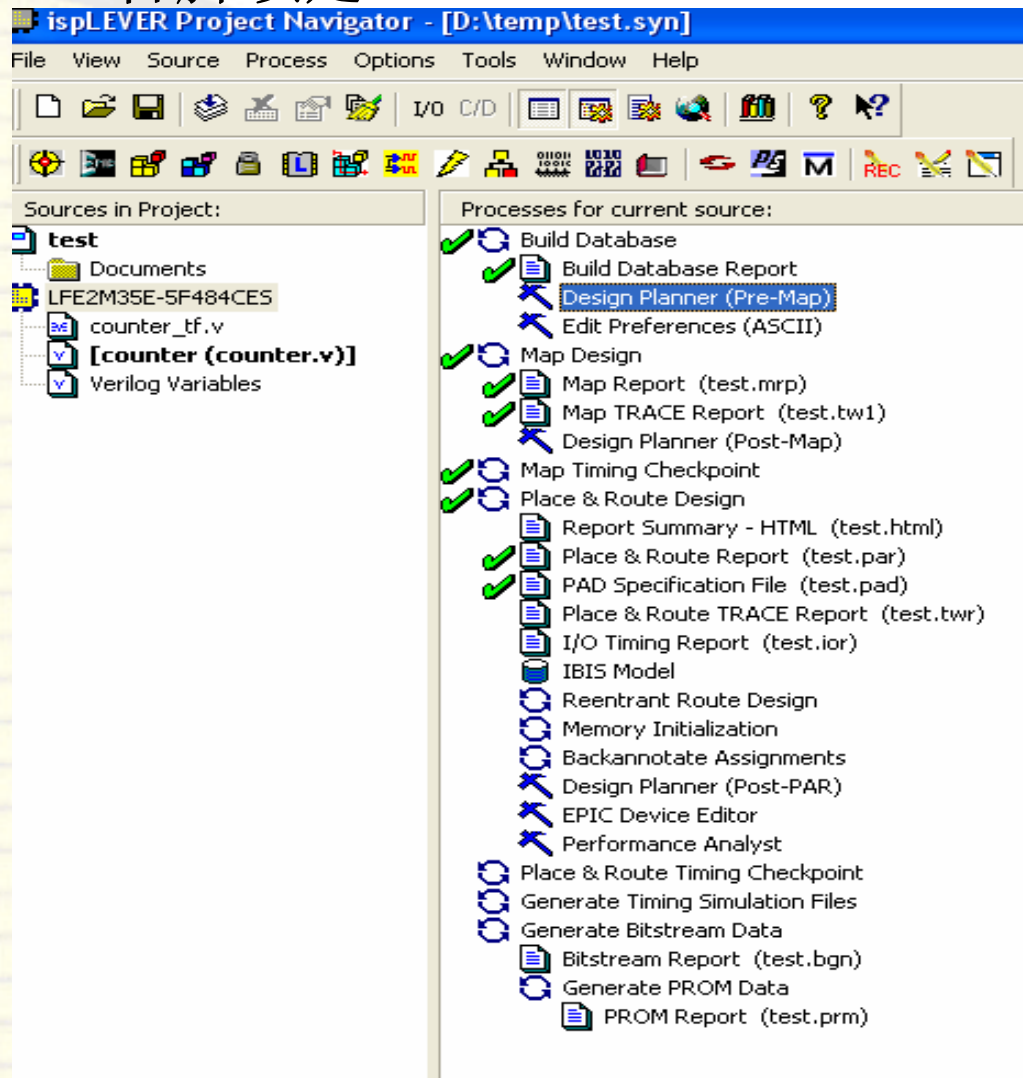


© LATTICE SEMICONDUCTOR CORPORATION



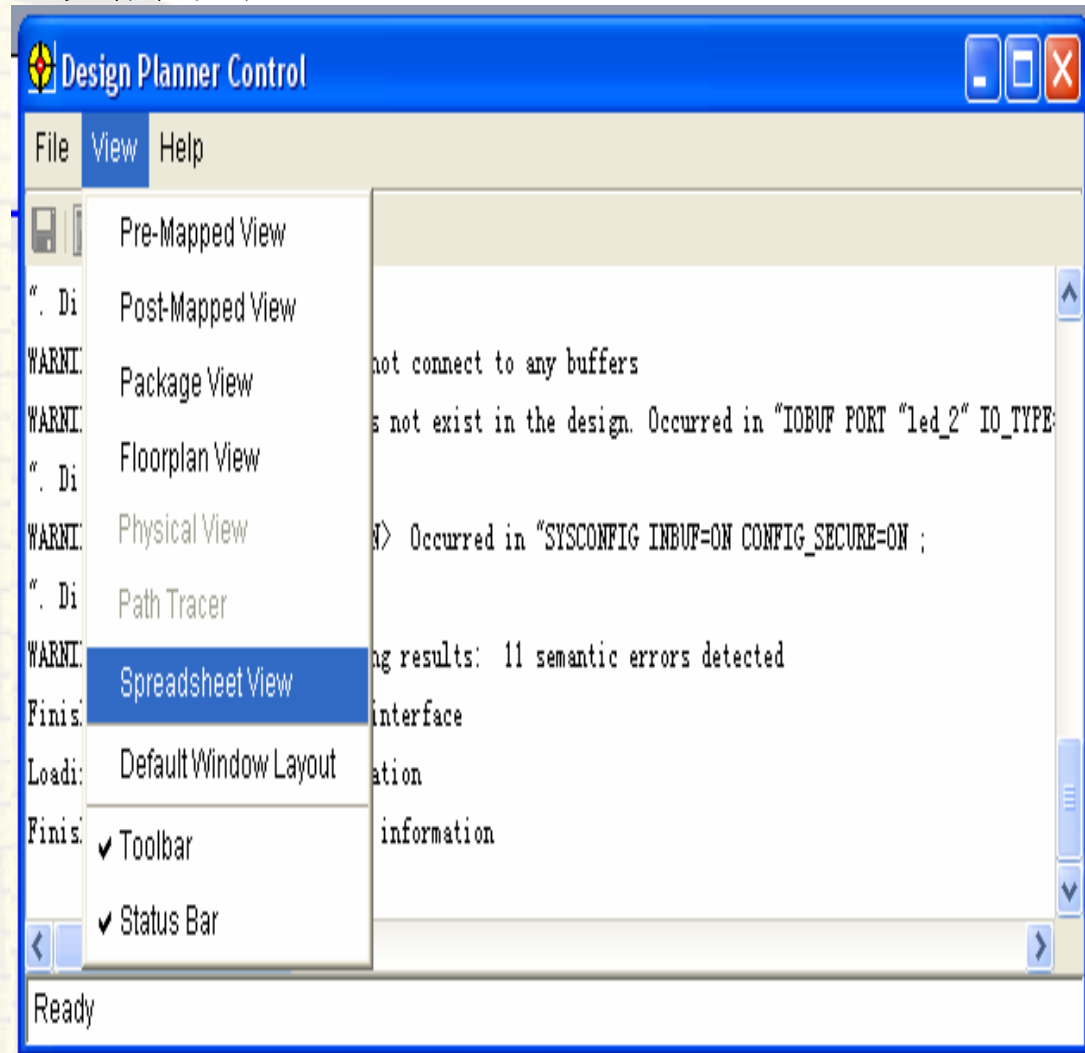
## 管脚锁定

- 除了图形化界面外，ispLever还支持其他管脚锁定方式。
- 在processes for current source窗口双击Design Planner (Pre-Map)。



## 管脚锁定

- Design Planner Control 窗口会被弹出。
- 在 **view** 菜单中选择 **Spreadsheet view**.



# 管脚锁定

- 在这个界面中你可以为任意地信号指定管脚位置。
- 另外所有的IO 属性都可指定，例如IO标准，上下拉电阻模式，驱动电流，open drain，slew rate 等等。

SpreadSheet View (test.lpf)

File Edit View Preference Tools Help

Save .test

Input Ports

- clk
- cnten
- load\_val(3:0)
- loadn
- rstn

Output Ports

- led(7:0)
- testp(7:0)

Nets

Cells

	Type	Name	Group by	Pin	Bank	Vref	IO_TYPE	PULLMODE	DRIVE	SLEWRATE	POICLAMP	OPENDRAIN
1	AllPorts		N/A	N/A	N/A	N/A	LVC MOS25	UP	N/A	FAST	OFF	OFF
2	Clock Input	clk	N/A				LVC MOS25	UP	NA	FAST	OFF	OFF
3	Input Port	cnten	N/A				LVC MOS25	UP	4	FAST	OFF	OFF
4	Input Port	load_val(0)	N/A				LVC MOS25	UP	NA	FAST	OFF	OFF
5	Input Port	load_val(1)	N/A				LVC MOS25	UP	NA	FAST	OFF	OFF
6	Input Port	load_val(2)	N/A				LVC MOS25	UP	NA	FAST	OFF	OFF
7	Input Port	load_val(3)	N/A				LVC MOS25	UP	NA	FAST	OFF	OFF
8	Input Port	loadn	N/A				LVC MOS25	UP	NA	FAST	OFF	OFF
9	Input Port	rstn	N/A				LVC MOS25	UP	NA	FAST	OFF	OFF
10	Output Port	led(0)	N/A				LVC MOS25	UP	12	FAST	OFF	OFF
11	Output Port	led(1)	N/A				LVC MOS25	UP	12	FAST	OFF	OFF
12	Output Port	led(2)	N/A				LVC MOS25	UP	12	FAST	OFF	OFF
13	Output Port	led(3)	N/A				LVC MOS25	UP	12	FAST	OFF	OFF
14	Output Port	led(4)	N/A				LVC MOS25	UP	12	FAST	OFF	OFF
15	Output Port	led(5)	N/A				LVC MOS25	UP	12	FAST	OFF	OFF
16	Output Port	led(6)	N/A				LVC MOS25	UP	12	FAST	OFF	OFF
17	Output Port	led(7)	N/A				LVC MOS25	UP	12	FAST	OFF	OFF
18	Output Port	testp(0)	N/A				LVC MOS25	UP	12	FAST	OFF	OFF
19	Output Port	testp(1)	N/A				LVC MOS25	UP	12	FAST	OFF	OFF
20	Output Port	testp(2)	N/A				LVC MOS25	UP	12	FAST	OFF	OFF
21	Output Port	testp(3)	N/A				LVC MOS25	UP	12	FAST	OFF	OFF
22	Output Port	testp(4)	N/A				LVC MOS25	UP	12	FAST	OFF	OFF
23	Output Port	testp(5)	N/A				LVC MOS25	UP	12	FAST	OFF	OFF
24	Output Port	testp(6)	N/A				LVC MOS25	UP	12	FAST	OFF	OFF
25	Output Port	testp(7)	N/A				LVC MOS25	UP	12	FAST	OFF	OFF

Group Name Type Member



# 实施Lattice constrain

双击design planner  
Spread sheet 将会 被弹出。

The screenshot shows the ispLEVER Project Navigator window with the 'Design Planner (Pre-Map)' process selected. A red arrow points from this process to the 'Spreadsheet View (test.lpl)' window, which displays a table of I/O pins.

**Sources in Project:**

- test
- Documents
- LFE2M35E-5F484CES
- counter\_tf.v
- [counter (counter.v)]
- Verilog Variables

**Processes for current source:**

- Build Database
- Build Database Report
- Design Planner (Pre-Map)
- Edit Preferences (ASCI)
- Map Design
- Map Report (test.mrp)
- Map TRACE Report (test.tw1)
- Design Planner (Post-Map)
- Map Timing Checkpoint
- Place & Route Design
- Report Summary - HTML (test.html)
- Place & Route Report (test.par)
- PAD Specification File (test.pad)
- Place & Route TRACE Report (test.twr)
- I/O Timing Report (test.iort)
- IBIS Model
- Reentrant Route Design
- Memory Initialization
- Backannotate Assignments
- Design Planner (Post-PAR)
- EPIC Device Editor
- Performance Analyst
- Place & Route Timing Checkpoint
- Generate Timing Simulation Files
- Generate Bitstream Data
- Bitstream Report (test.bgn)
- Generate PROM Data
- PROM Report (test.prm)

**Spreadsheet View (test.lpl)**

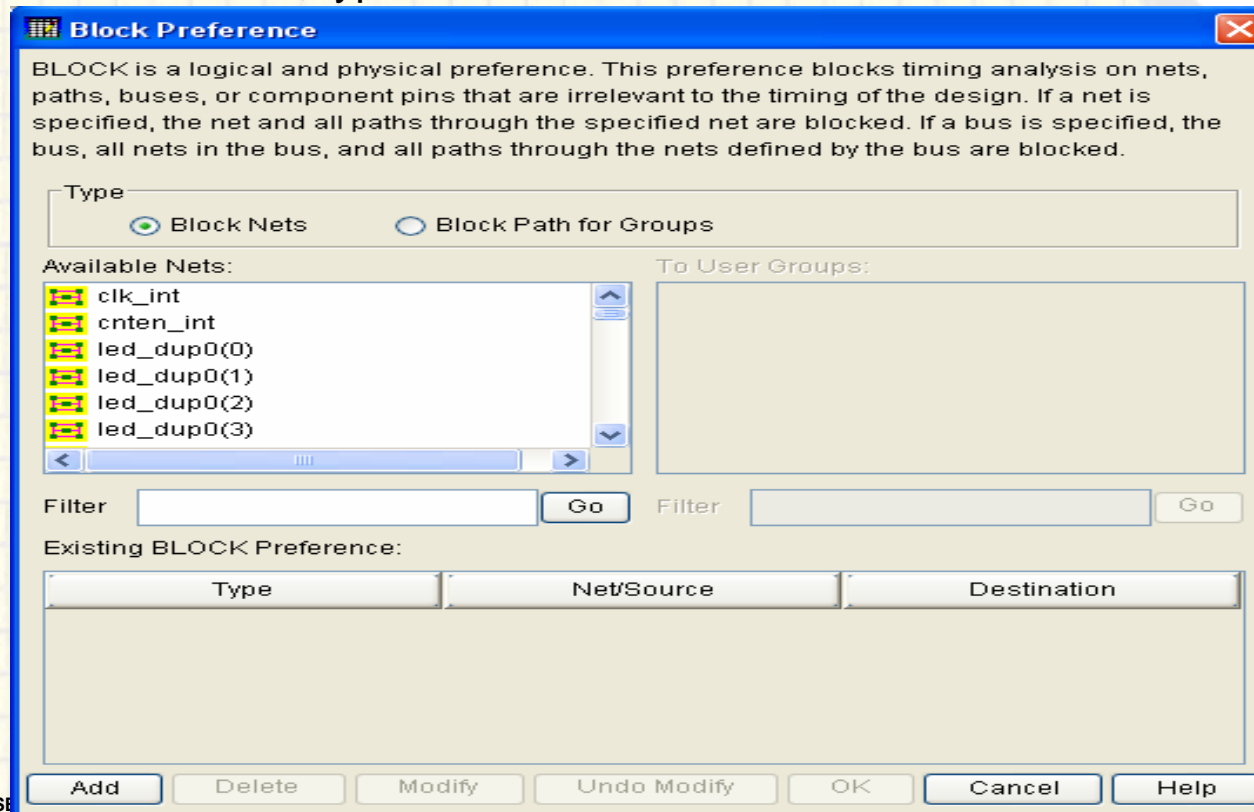
Type	Name	Gr...	Vref	IO_TYPE	PULLMODE	DRIVE	SLEWRATE	POCLAMP	
1	AIOPORTS	N/A	N/A	N/A	LVC MOS25	UP	N/A	FAST	OFF
2	Clock Input	clk	N/A	LVC MOS25	UP	N/A	FAST	OFF	
3	Input Port	cnten	N/A	LVC MOS25	UP	N/A	FAST	OFF	
4	Input Port	load_val(0)	N/A	LVC MOS25	UP	N/A	FAST	OFF	
5	Input Port	load_val(1)	N/A	LVC MOS25	UP	N/A	FAST	OFF	
6	Input Port	load_val(2)	N/A	LVC MOS25	UP	N/A	FAST	OFF	
7	Input Port	load_val(3)	N/A	LVC MOS25	UP	N/A	FAST	OFF	
8	Input Port	loadn	N/A	LVC MOS25	UP	N/A	FAST	OFF	
9	Input Port	rstm	N/A	LVC MOS25	UP	N/A	FAST	OFF	
10	Output Port	led(0)	N/A	LVC MOS25	UP	12	FAST	OFF	
11	Output Port	led(1)	N/A	LVC MOS25	UP	12	FAST	OFF	
12	Output Port	led(2)	N/A	LVC MOS25	UP	12	FAST	OFF	
13	Output Port	led(3)	N/A	LVC MOS25	UP	12	FAST	OFF	
14	Output Port	led(4)	N/A	LVC MOS25	UP	12	FAST	OFF	
15	Output Port	led(5)	N/A	LVC MOS25	UP	12	FAST	OFF	
16	Output Port	led(6)	N/A	LVC MOS25	UP	12	FAST	OFF	
17	Output Port	led(7)	N/A	LVC MOS25	UP	12	FAST	OFF	
18	Output Port	testp(0)	N/A	LVC MOS25	UP	12	FAST	OFF	
19	Output Port	testp(1)	N/A	LVC MOS25	UP	12	FAST	OFF	
20	Output Port	testp(2)	N/A	LVC MOS25	UP	12	FAST	OFF	
21	Output Port	testp(3)	N/A	LVC MOS25	UP	12	FAST	OFF	
22	Output Port	testp(4)	N/A	LVC MOS25	UP	12	FAST	OFF	
23	Output Port	testp(5)	N/A	LVC MOS25	UP	12	FAST	OFF	
24	Output Port	testp(6)	N/A	LVC MOS25	UP	12	FAST	OFF	
25	Output Port	testp(7)	N/A	LVC MOS25	UP	12	FAST	OFF	

# 实施Lattice constrain

- Assign block preference

当设定一个 BLOCK preference, the ispLEVER 在进行静态时序分析时会阻止指定的nets, paths, buses, and component pins (前提: 所有这些都是和同步时序无关的)

1. In Spreadsheet View, choose **Preference > Block Preference** to open the dialog box.
2. In the Filter text box, type **rst\*** and click **Go**.



## 实施Lattice constrain

- 3. 选择 **rstn\_int** 点击 **Add**.
- 4. 点击 **OK**. 一个 **BLOCK** preference 就定义好了, 被阻止的网络将被添加到 existing **BLOCK** Preference 列表中。

Existing BLOCK Preference:

Type	Net/Source	Destination
BLOCK_NET	rstn_int	

- 5. 选择 **Global** tab.
- 6. 在 **Preference Value** 下, 双击 **Block Asynchpaths** 从菜单中选择 **ON**.
- 7. 重复步骤6 for **Block Resetpaths** and **Block InterClock Domain Paths**.

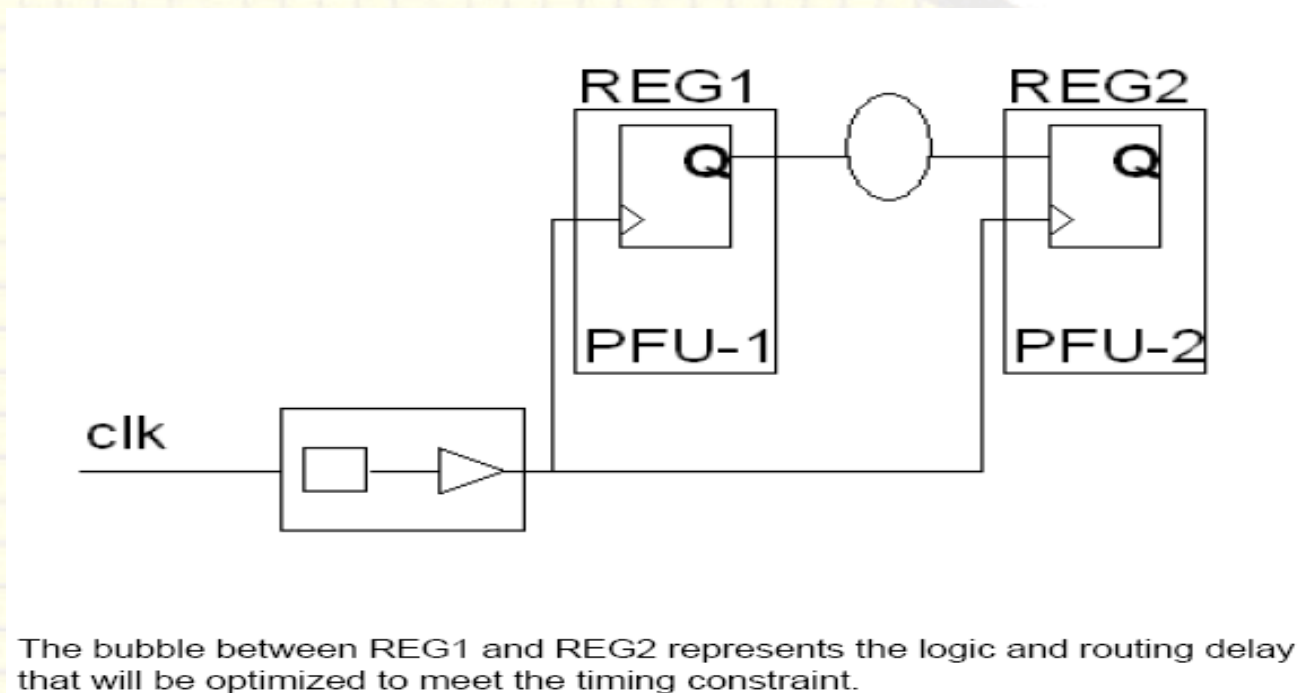
	Preference Name ▲	Preference Value
1	Block Asynchpaths	ON
2	Block InterClock Domain Paths	ON
3	Block RD During WR Paths	OFF
4	Block Resetpaths	ON

- 8. Choose **File > Save** 在 .lpf 文件中保存 **BLOCK** preferences。



## 实施Lattice constrain

- Assign clock preference



- 我们将对clock频率采用过约束技术（20%），以获得更好的布局布线结果。

# 实施Lattice constrain

- 1. 在Spreadsheet View, 选择 **Preference > Period/Frequency** 打开对话框。
- 2. 选择下列选项:
- Type: **FREQUENCY**
- Second Type: **Clock Port**
- The Available Clock Ports display is updated.

**PERIOD/FREQUENCY Preference**

FREQUENCY is a logical and physical preference. It identifies the minimum operating frequency for all sequential output to sequential input pins clocked by the specified net. If no net name is given, the preference applies to all clock nets in the design that do not have a specific clock frequency preference.

Type

☐ PERIOD

☒ FREQUENCY

Second Type

☐ None

☐ Clock Net

☒ Clock Port

Available Clock Ports:

clk

Filter: clk Go

Frequency: MHz

Hold Margin: ns

PAR\_ADJ:

Existing Preference:

Type	Object Type	Net/Port	Time/Freq	Hold Margin	PAR_ADJ
FREQUENCY...	CLKPORT	clk	120 MHz		

Add Delete Modify Undo Modify OK Cancel Help

## 实施Lattice constrain

- 3. Select clock port **clk**. Specify **120** in the Frequency box and select **MHz**.
- 6. Click **Add**, and then click **OK**.

Existing Preference:

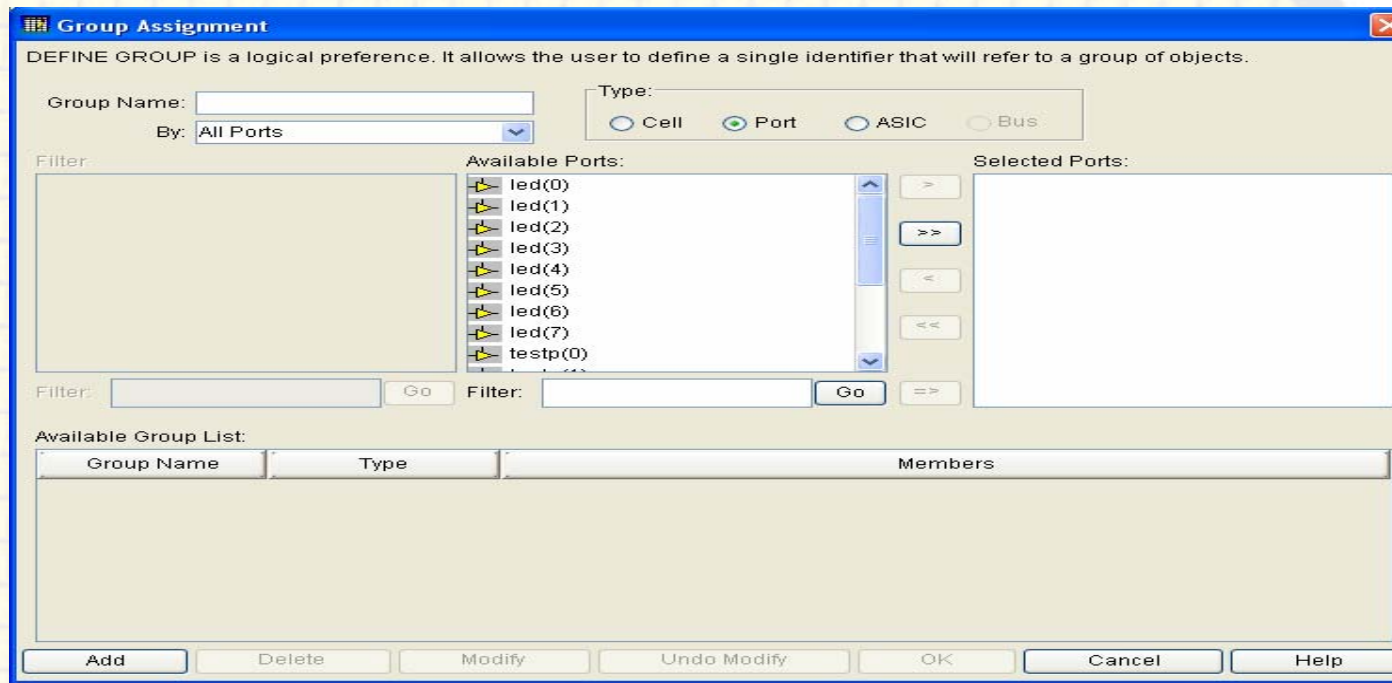
Type	Object Type	Net/Port	Time/Freq	Hold Margin	PAR_ADJ
FREQUENCY...	CLKPORT	clk	120 MHz		

- 7. Choose **File > Save** to save the clock preferences to the .lpf



## 实施Lattice constrain



- **Assign IO timing preference**
- *To group I/Os:*
- 1. 在Spreadsheet View,选择**Preference > Grouping Assignment** 打开对话框。
- 2. 在Group Name 文本框, 键入**load**.
- 3. 在Filter 文本框中键入**load\*** 然后点击**GO**.
- 4. 在Available Ports窗口下选择 **load\_val(0)**至**load\_val(3)**. 使用Shift键选中所有4个Port。



## 实施Lattice constrain

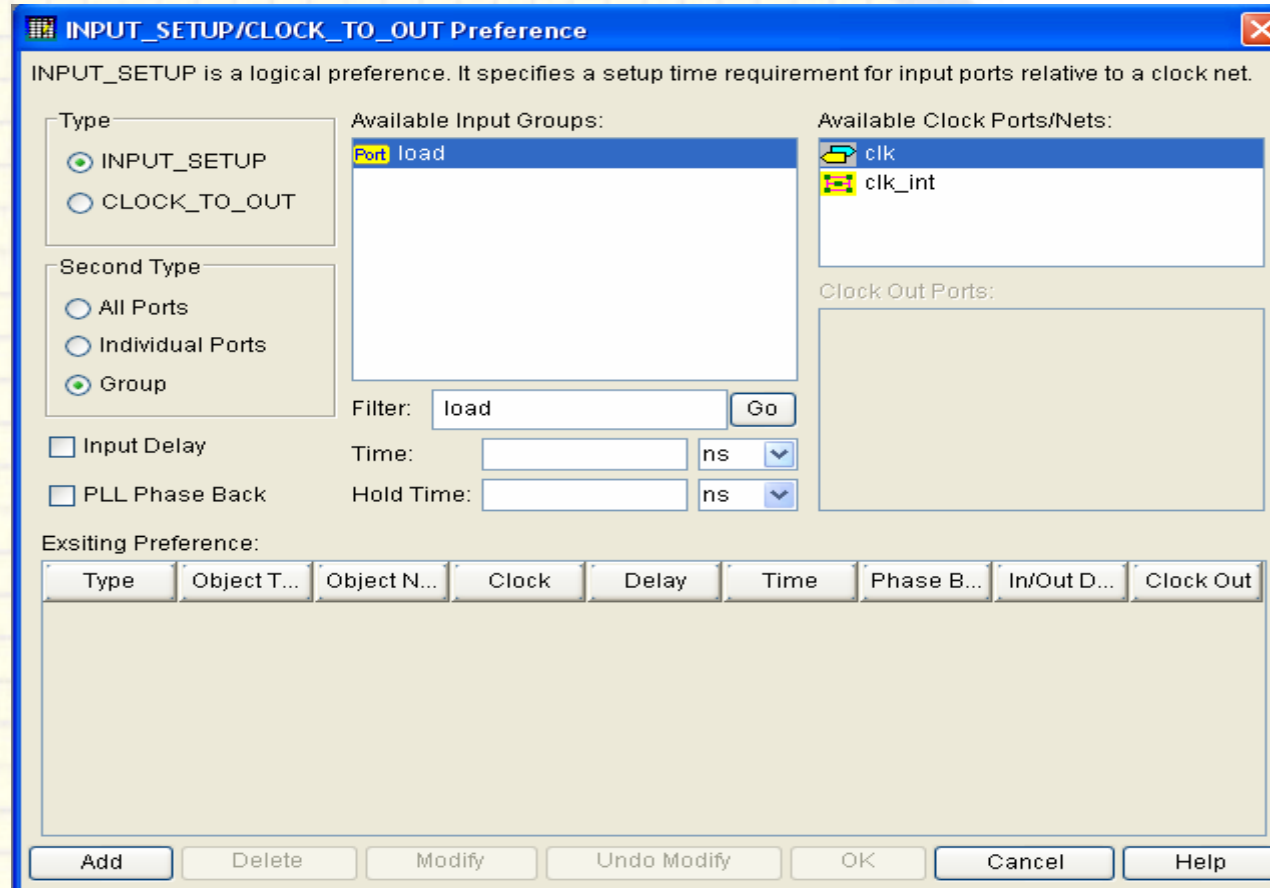
- 5. 点击 > 按钮添加Port到Selected Ports list.
- 6. 点击 **Add** button 产生一个新的Group并添加到Available Group List.
- 7. 在Group Name 文本框, 键入**led\_out**.
- 8. 在 Filter文本框, 键入**led\*** 然后点击**GO**.
- 9. 在Available Ports, 选中**led(0)至led(7)**. 使用Shift键选中所有8个Port.
- 10. 点击 > button 添加所有选中Port到Selected Ports list.
- 11. 点击**Add** button创建一个新的Group并添加到Available Group List.
- 12. 点击**OK**.
- 可以看到load and led\_out group被添加到了左下方的Group Name list。

Available Group List:

Group Name	Type	Members
 load	PORT	load_val(0), load_val(1), load_val(2), load_val(3)
 led_out	PORT	led(0), led(1), led(2), led(3), led(4), led(5), led(6), led(7)

## 实施Lattice constrain

- To create I/O timing preferences:
- 1. 选择**Preference > Input\_Setup/Clock\_To\_Out** 打开对话框.





## 实施Lattice constrain

- 2. 选择下列选项:
- Type: **INPUT\_SETUP**
- Second Type: **Group**
- Time: **6 (ns)**
- 可用的Input Groups和可用的Clock Ports/Nets lists 在窗口被更新。
- 3.在Available Input Groups list选中**led**.
- 4.在Available Clock Ports/Nets list选中**clk**.
- 5.点击**Add**.
- 已经存在的Preference List将被更新。
- 6. 选择下列选型:
- Type: **CLOCK\_TO\_OUT**
- Second Type: **Group**
- Time: **5 (ns)**
- 可用的Output Groups和可用的Clock Ports/Nets lists 在窗口被更新。

## 实施Lattice constrain

- 7.在Available Output Groups list中选择**led\_out**.
- 8.在Available Clock Ports/Nets list中选择**clk**.
- 9. 点击**Add**, 然后点击**OK**.

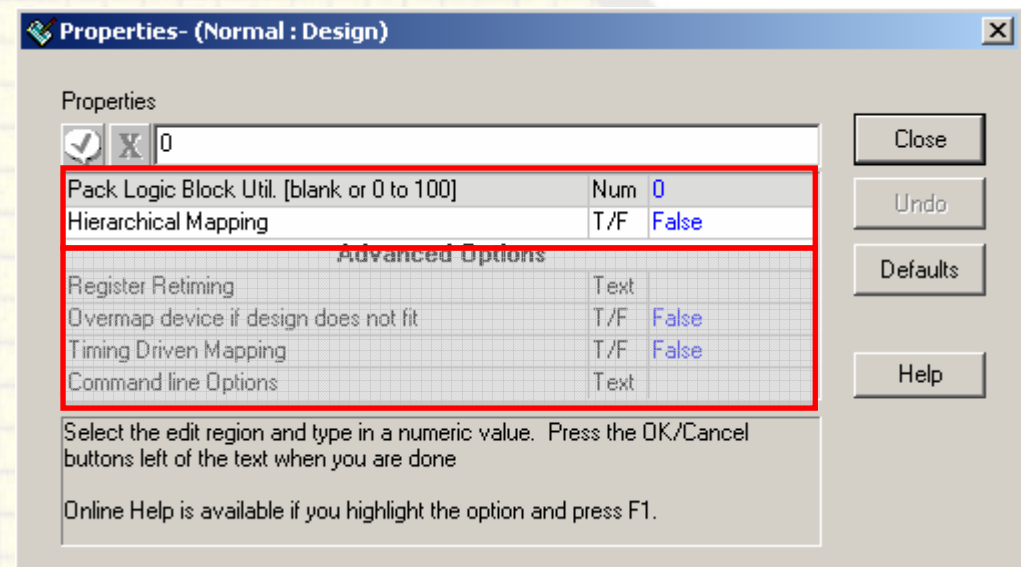
Exsiting Preference:

Type	Object T...	Object N...	Clock	Delay	Time	Phase B...	In/Out D...	Clock Out
INPUT_S...	GROUP	load	CLKPOR...	6 ns			Not Input ...	
CLOCK_T...	GROUP	led_out	CLKPOR...	5 ns			Not Outpu...	

- 可以看到关键的INPUT\_SETUP and CLOCK\_TO\_OUT preferences 已经被定义好了。
- 10. Choose **File > Save**保存I/O timing preferences至.lpf文件.
- 11. Choose **File > Exit**关闭Design Planner.

## Map(生成physical design database file .ncd)

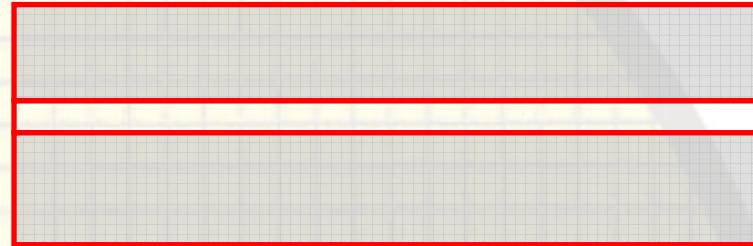
- Map使用 .ngd and .lpf 文件生成 physical database 文件.ncd file, device-specific component网表文件. 任和更新的preferences将会被写入physical preference file (.prf).
- 设定器件内部PFU Packing密度:  
Mapper把这个参数转换成 PFU数量。该参数范围 0-100% (100 = 最小的packing; 0 = 最大的密度)  
。如果使能了这个选项但是没有指定器件资源使用率, 默认值为 97。  
使用这个选项会导致denser design 但同时会增加布局布线的难度.
- 设置该选项**True** 将实现hierarchical mapping 而不是flat mapping.  
MAP 不会在同一个PFU中 pack不同 top level的 module. 这会通过Logic Grouping给Place&route提供更好的 Place选择。





## Map(生成physical design database file .ncd)

- Register Retiming技术可以在组合逻辑路径上移动寄存器的位置，从而均衡各个寄存器之间的延时，获得最好的Performance。
- Retiming支持下面6个timing preferences: INPUT\_SETUP, CLOCK\_TO\_OUT, FREQUENCY, MULTICYCLE, MAXDELAY, and PERIOD. 其他的preferences不会影响retiming.
- 请参照下面的关于Retiming的详细说明。



## Map(生成physical design database file .ncd)

- Retiming可以有4中选择:
- **TRUE**
- 使能register retiming.
- 在文本框键入 “TRUE”.
- **FALSE**
- 关闭register retiming.
- 在文本框键入 “FALSE”.
- **EFFORT**
- 控制算法运行的叠代次数， 算法每一次运行都会试图发现最优化的register retiming方案。 EFFORT的值为 1-6, 1 代表least effort and 6 most effort. 默认值为 4。
- 有一点需要说明的是high effort 未必一定获得最好的结果,同时会导致更长的软件Map时间和更多的register使用量。

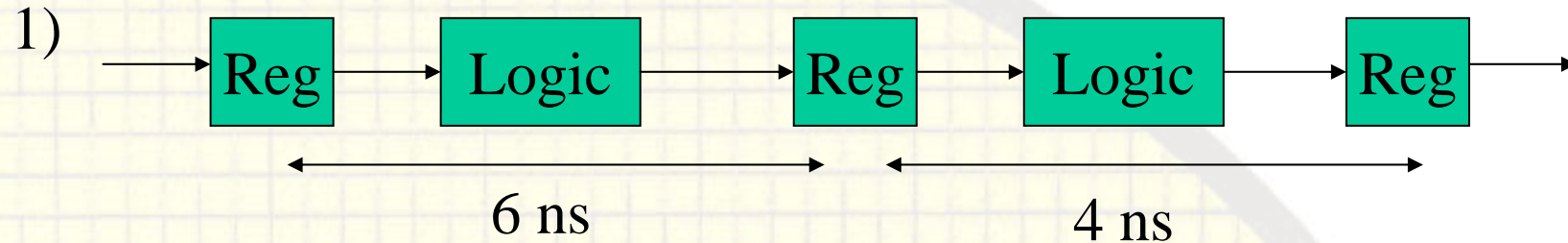
## Map(生成physical design database file .ncd)

- **DETOUR**
- 控制算法运行的叠代次数，每一次算法运行都会试图估算最优的retiming布线。Detour取值1-6, 1代表least severity estimate and 6 the most.
- DETOUR 值取决于器件内部布线资源的多少。通常器件的布线资源越多就可以设置更低的DETOUR值。
- Default values:
- LatticeSCM/SC: 2
- LatticeECP/ECP2/ECP2M/EC: 4
- LatticeXP/XP2: 5
- MachXO: 5
- 同时设置EFFORT和DETOUR可以在文本框键入如下参数:
- EFFORT=*value1*:DETOUR=*value2*
- For example,
- EFFORT=2:DETOUR=6

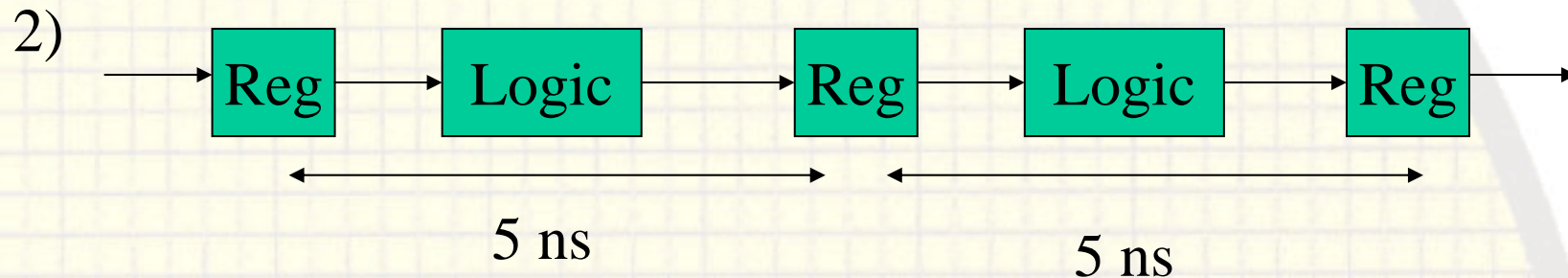


## Map(生成physical design database file .ncd)

- Why retiming???



在这个case中  $f_{\max}$  is  $1/6\text{ns} = 166\text{ MHz}$



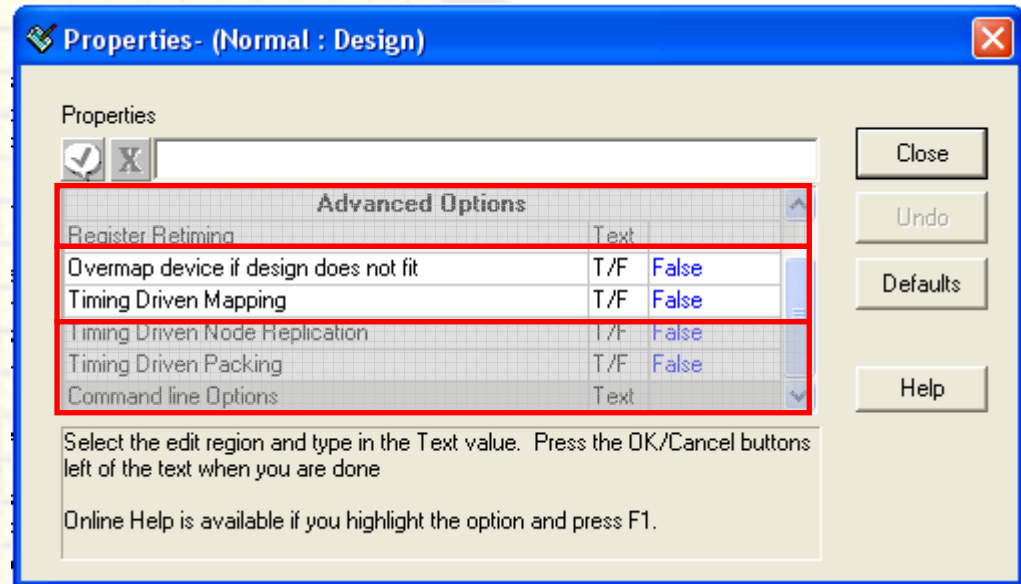
如果我们能够平衡各个寄存器之间的组合逻辑延时就可以获得更快的  $f_{\max}$ . 在这个例子中通过Retiming技术  $F_{\max}$ 可以达到 $1/5\text{ns} = 200\text{ MHz}$ 。

### Why not Retiming every time you might ask?

Its dangerous as you are actually changing the logic of your design.

## Map(生成physical design database file .ncd)

- MAP process会生成一个physical database (.ncd) 文件即使器件很小不能fit整个设计，只是这个NCD file 不能被用于Place & Route。这个功能在下面的情况下是非常有用的：当你想观察整个mapping结果然后决定要砍掉多少逻辑时。
- 允许软件实施 timing-driven mapping 从而进一步优化关键路径。Timing Driven Mapping读取preference file 计算出所有关键路径的slacks. mapping 会基于Slack 来优化关键路径。



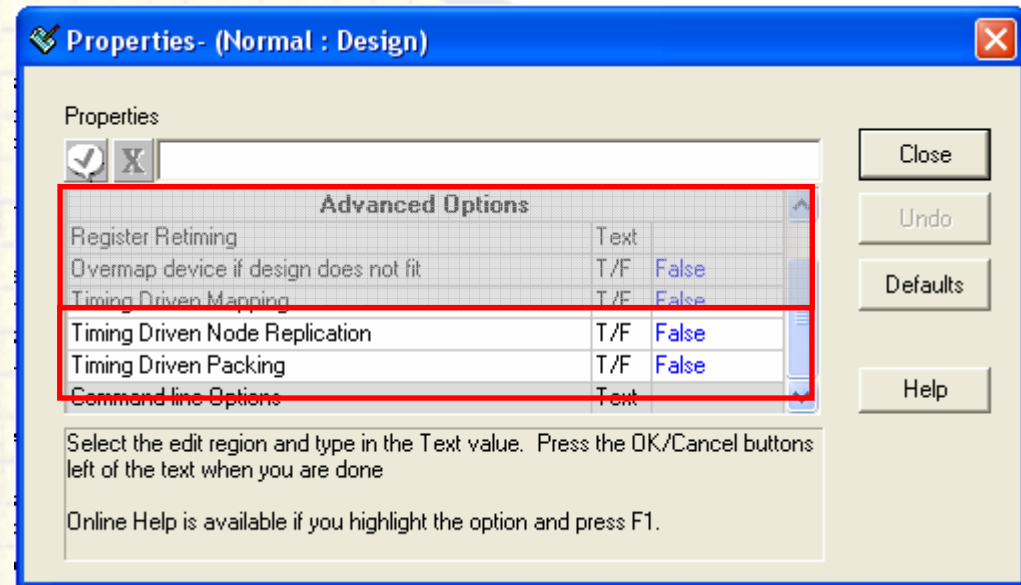
## Map(生成physical design database file .ncd)

- 允许软件实施 timing driven的逻辑复制. 设置为**True**, 如果LUT-4到触发器的扇出很大, 软件会自动复制LUT-4(该查找表属于timing path), 每个触发器会配置一个LUT-4。

这就会Pack一个slice中的LUT/FF。

Default is **False**.

- 允许软件在一个slice中实施timing driven packing: LUT/FF, FF/LUT, 和LUT/LUT, 减小逻辑之间的延时。选项有 **True** and **False** (默认)。





## Map(生成physical design database file .ncd)

- 估算**baseline performance**
- 1.在这个步骤中将基于逻辑设计的pre-route版本运行静态时序分析工具**TRACE**。从而发现影响系统性能的worst case路径。
- 2.通常pro-route静态时序分析的结果将比route版本结果快一倍。
- 3.双击“map trace report”来运行 trace 生成\*.twr文件。
- 4.在Window菜单选中Text Editor, 在Text Editor中选择**File > Open**. Files of Type选择**All Files**,找到test.twr文件点击**Open**.
- 5.使用**Edit > Find** 命令在Trace report中锁定下面的语句来发现worst case路径:
  - *Preference: FREQUENCY NET "clk\_c"*

# Map(生成physical design database file .ncd)

在Translation和Map之后运行TRACE 能够获得设计timing的早期估计和设计的baseline performance.

在一些heavily interconnected or high fan out设计中布线大致要占总path延时的70%, 例如在Map TRACE reports中如果系统达到 ~ 364 MHz, 那末最终布线后的结果大致是  $.3(364 \text{ MHz}) = 109 \text{ MHz}$

- *Connections not covered by the preferences*
- 5. 在Mapping优化方法中比较有效的能提高系统性能的方法是register retiming。

# 布局布线

- 初次布局布线的目的是建立设计的**baseline performance**
- 将运行 ispLEVER Place & Route 和静态时序分析工具 TRACE 来建立设计的 **baseline performance**. 设计者可以比较之前定义的 **timing preferences** 和实现，检查那些满足了约束那些没有满足。
- 1.双击 “**place&route design**” 在默认的约束下运行布局布线工具。
- 2. Window菜单Text Editor, choose **File > Open**. 文件类型选择**All Files**，找到 **test.par** file, 点击 **Open**，在报告的开始, 列出了设计所使用的各种FPGA内部资源.

- Device utilization summary:

– PIO	24/458	5% used
•	24/303	7% bonded
– SLICE	38/17100	<1% used
– GSR	1/1	100% used
– Number of Signals:	125	
– Number of Connections:	207	



# 布局布线

- 3. Page down 在 PAR report 会列出高扇出信号(clocks, set/resets (sr load), 和 clock 使能信号) 布线资源的分配.
  - The following 1 signal is selected to use the primary clock routing resource:
  - clk\_int (driver: clk, clk load #: 16)
  - No signal is selected as DCS clock.
  - No signal is selected as secondary clock.
  - Signal rstn\_int is selected as Global Set/Reset.
  - Starting Placer Phase 0.
  - .....
  - Finished Placer Phase 0. REAL time: 18 secs
  - Starting Placer Phase 1.
  - Placer score = 364687.
  - .....
  - Placer score = 95606.
  - Finished Placer Phase 1. REAL time: 35 secs
  - Starting Placer Phase 2.
  - Placer score = 95606
  - Finished Placer Phase 2. REAL time: 35 secs

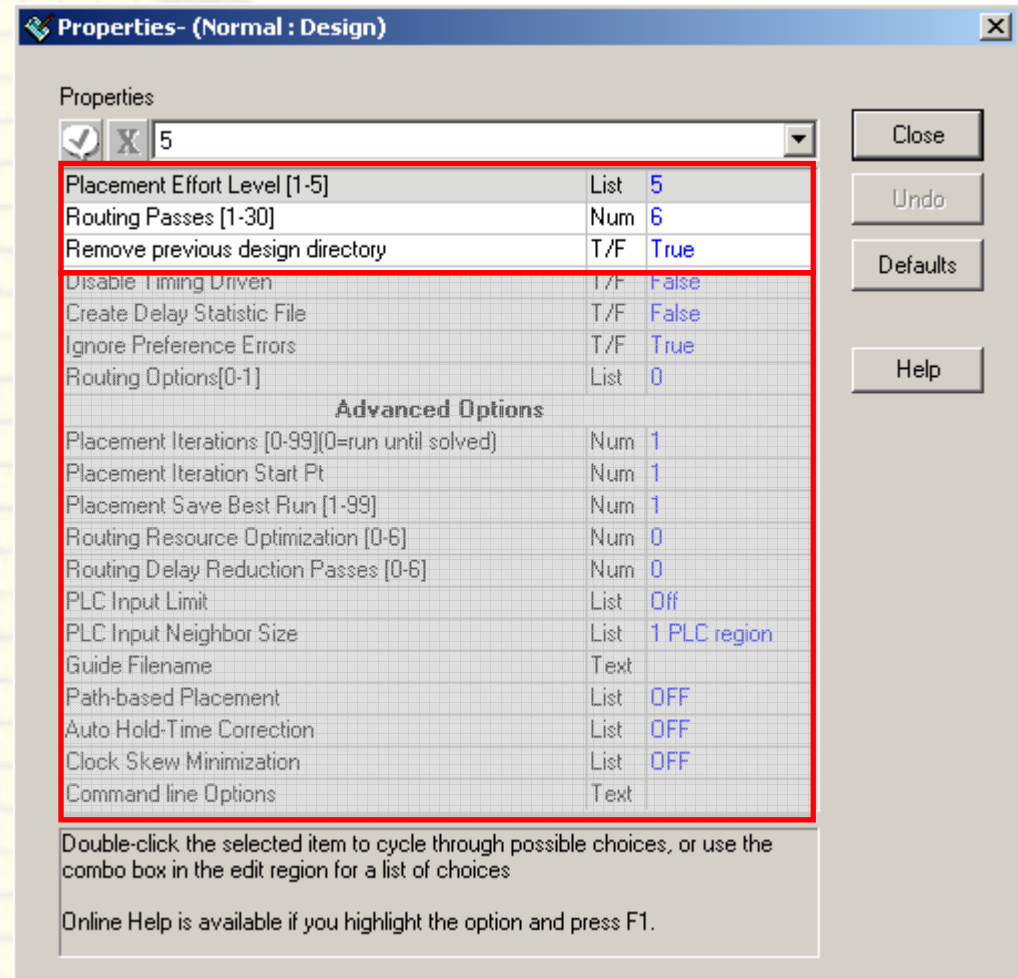
# 布局布线

```
----- Clock Report -----  
- Global Clock Resources:  
- CLK_PIN : 1 out of 8 (12%)  
- PLL : 0 out of 8 (0%)  
- DCS : 0 out of 8 (0%)  
- Quadrants All (TL, TR, BL, BR) - Global Clocks:  
- PRIMARY "clk_int" from CLK_PIN "AA8", driver "clk", clk load = 16  
- PRIMARY : 1 out of 8 (12%)  
- DCS : 0 out of 2 (0%)  
- SECONDARY: 0 out of 4 (0%)  
- Edge Clocks:  
- No edge clock selected  
----- End of Clock Report -----
```

- 4. Window菜单Text Editor, choose **File > Open**. 文件类型选择**All Files**, 找到**test.twr**文件, 点击**Open**.
  - 该报告是一个post-route后的时序报告。
  - 列出了寄存器之间的 **worst-case path**, 设计者通过分析**worst-case path**来决定采取何种优化方法来提升性能(例如修改源代码或是采用优化算法控制布局布线提升系统**performance**)

# 控制place and route

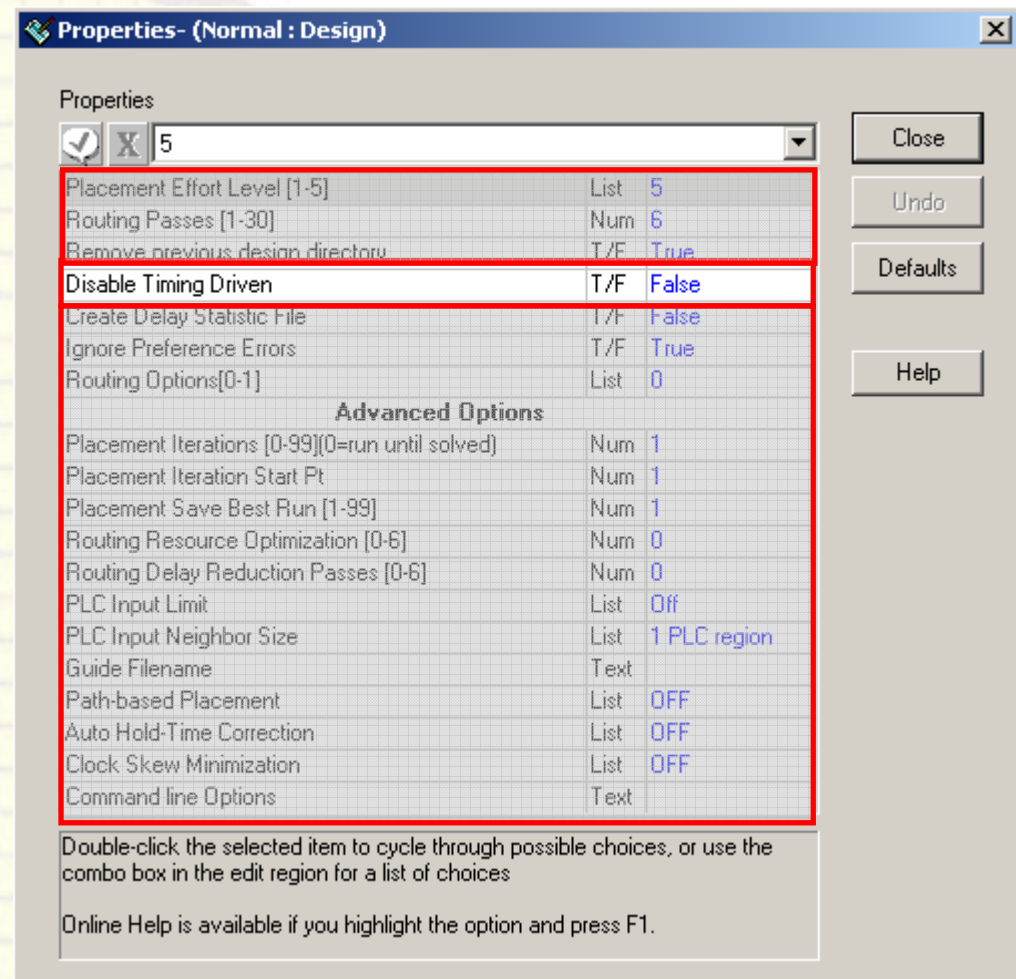
- 布局布线Effort 1 lowest , 5 highest.
- 设置Place&route工具运行布线pass (1-30)上限. 一个布线pass是针对未布通网络进行布线的一次尝试. 如果一个设计中pack或是时序约束非常严格那末在布局布线时就需要多个routing passes 才能完成布局布线. 如果该选项没有被设置工具将会自行决定Pass的次数。.
- 在运行PAR之前是否删除上次布局布线生成的目录. 如果已经运行过PAR并想保存上次的结果需要选False.





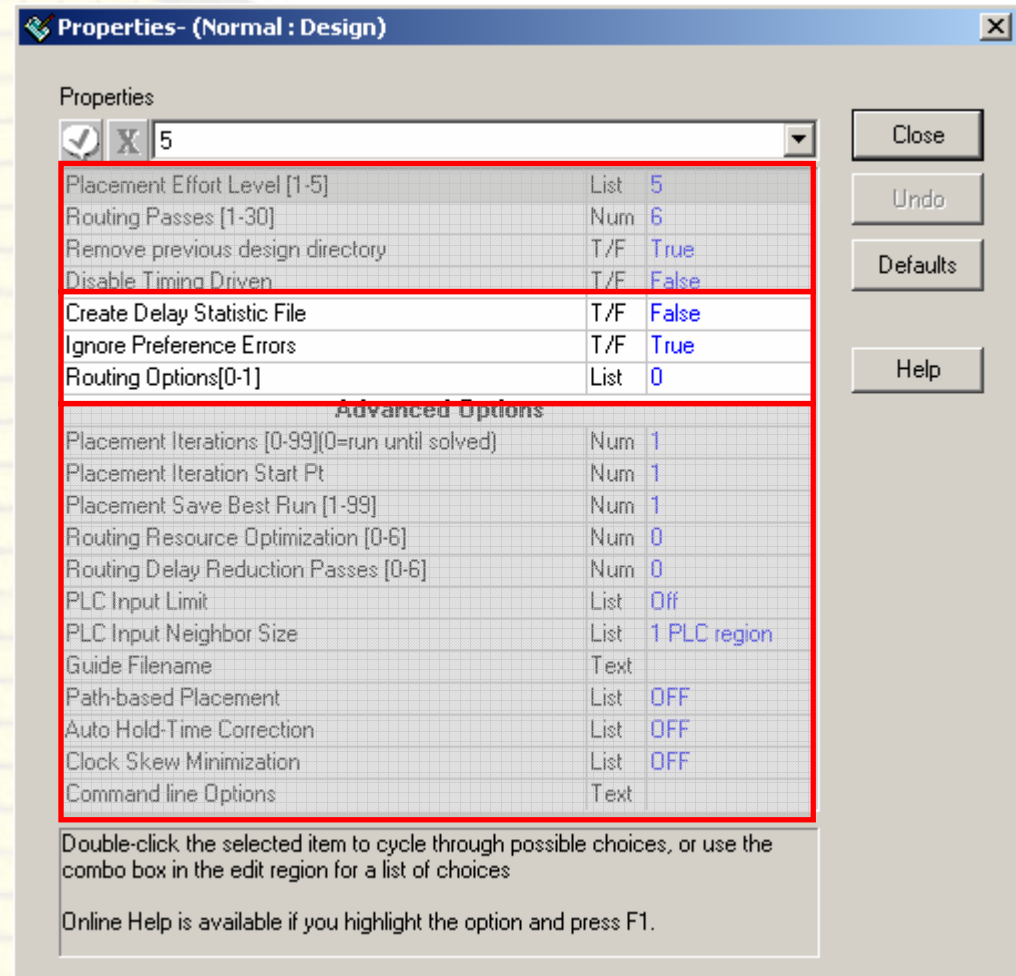
# 控制place and route

- 运行非Timing driven的PAR，仅仅实施基于cost-based布局布线。
- 设计者已经把约束写入了prference文件，但是希望执行非timing driven的快速布局布线从而对该设计在特定器件上能否实现布局布线和布局布线的难易进行初步了解和认知。



## 控制place and route

- 如果选TRUE, 延时报告文件 (.dly) 在每次PAR run之后输出. 延时报告包含每次PAR run的所有网络的延时信息. 报告会列出最差的20个网络.
- TRUE: Place & Route在运行过程中会忽略preference errors 而继续运行.  
False: 一旦出现preference error Place& Route就会中止并输出error信息.
- Routing 算法选择: 1 Routing时short connection具有高优先级, 0 is reversed. 当遇到timing问题时可以分别尝试这两种算法.

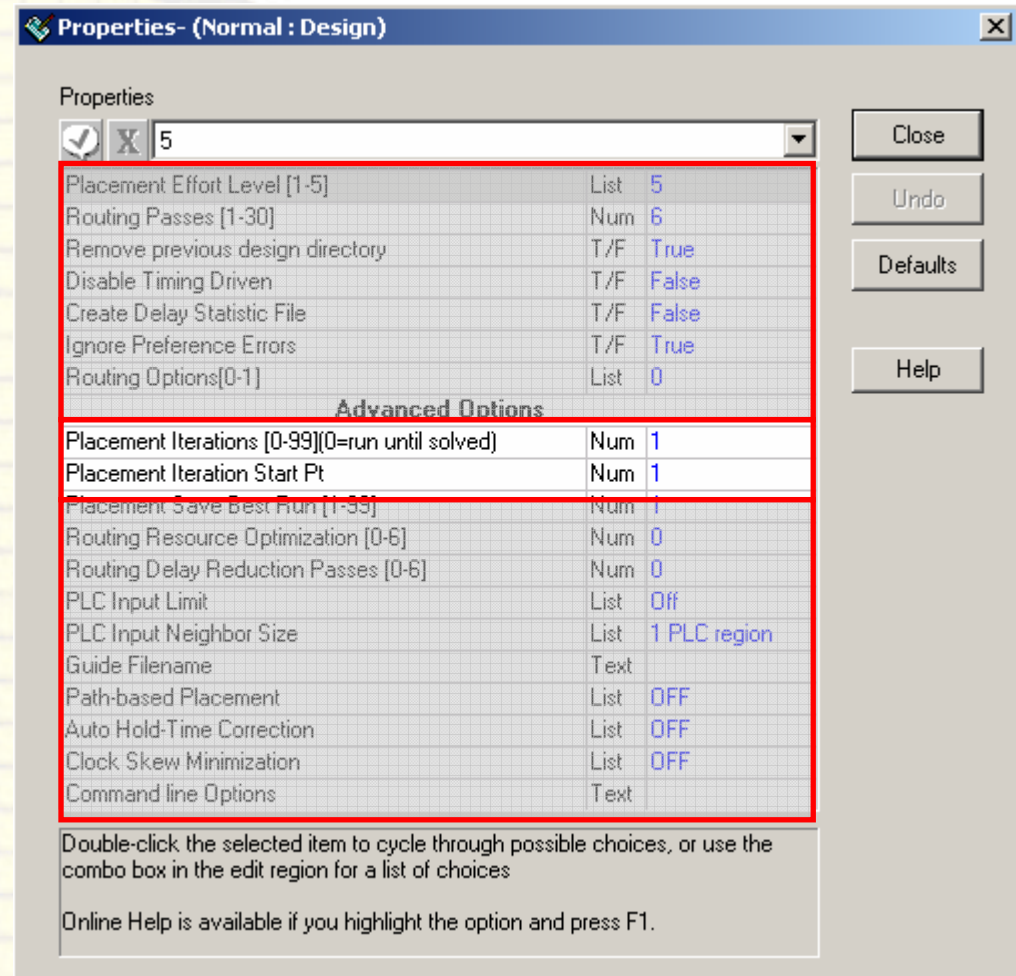


# 控制place and route

- 规定在特定的**Placement effort**下，运行的布局布线的叠代次数 (0-99) (不管布局布线是否完成)

0=run until solved

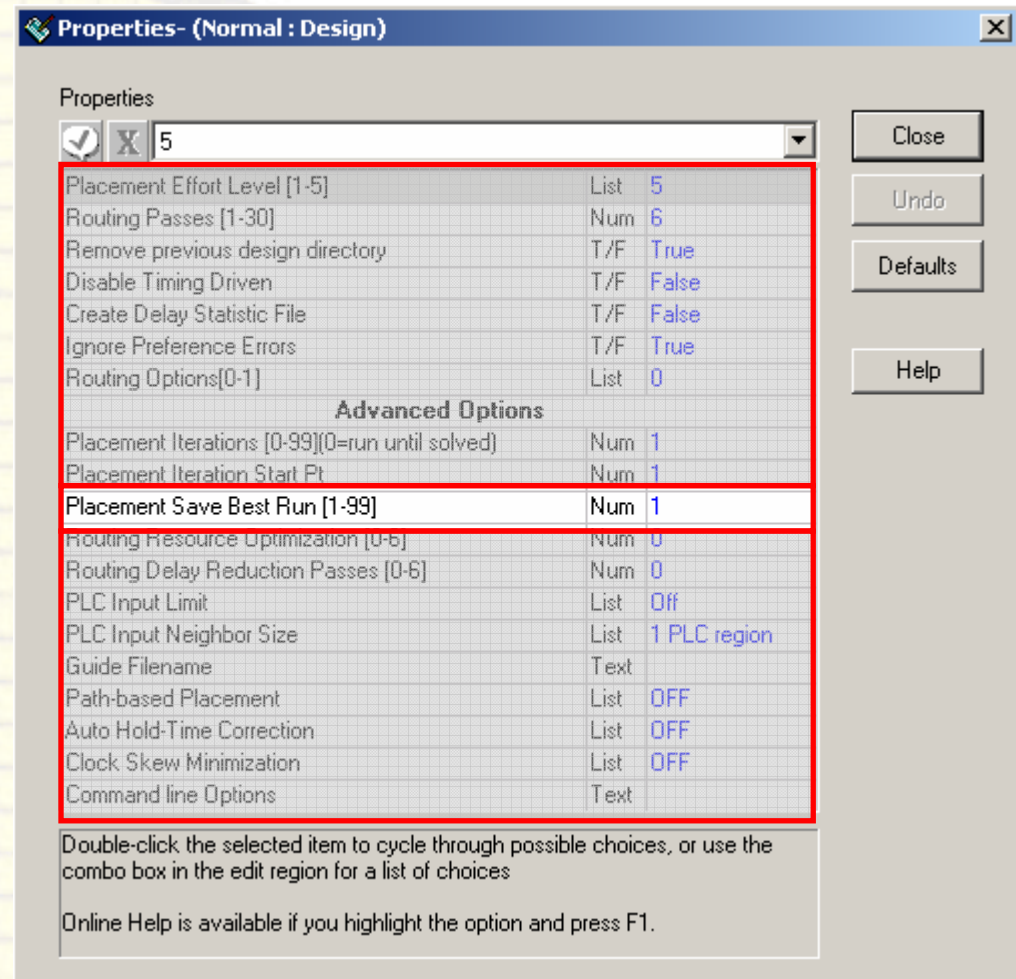
- 每次叠代使用不同的 **cost table (seed)** 会产生不同的**physical database**文件 (.ncd)。
- 如果指定了**cost table**, 叠代将以指定的**cost table**值开始运行布局布线。
- 指定布局布线叠代的**cost table**(1~100)。





## 控制place and route

- 设定保存Place& Route最好结果的数量 (1-99) (默认1)。
- 如果不指定具体的数字,所有的PLACE & ROUTE 结果都 将被保存。
- 由Scoring system来确定最好的输出结果。
- 该选项并不关心软件执行多少次叠代以及使用什么样的effort level。它仅仅比较每次布局布线的结果并按照软件设定保留最优结果physical database files (.ncd)。

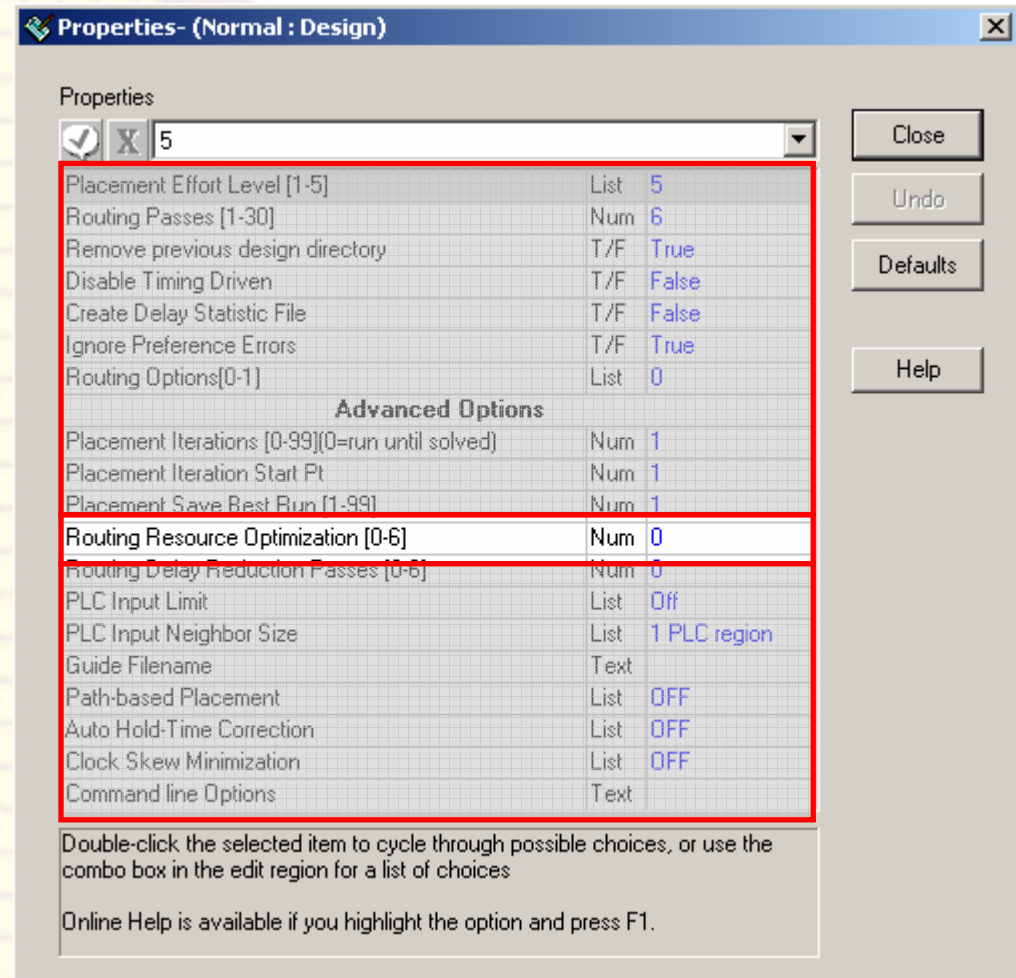


# 控制place and route

- 控制cost-based cleanup routing(在布线之前每个网络计算一个权值), 如果设计中始终有未布通网络时可以尝试此设置。
- 该选项确定需要运行cost-based cleanup routing 的pass数目。

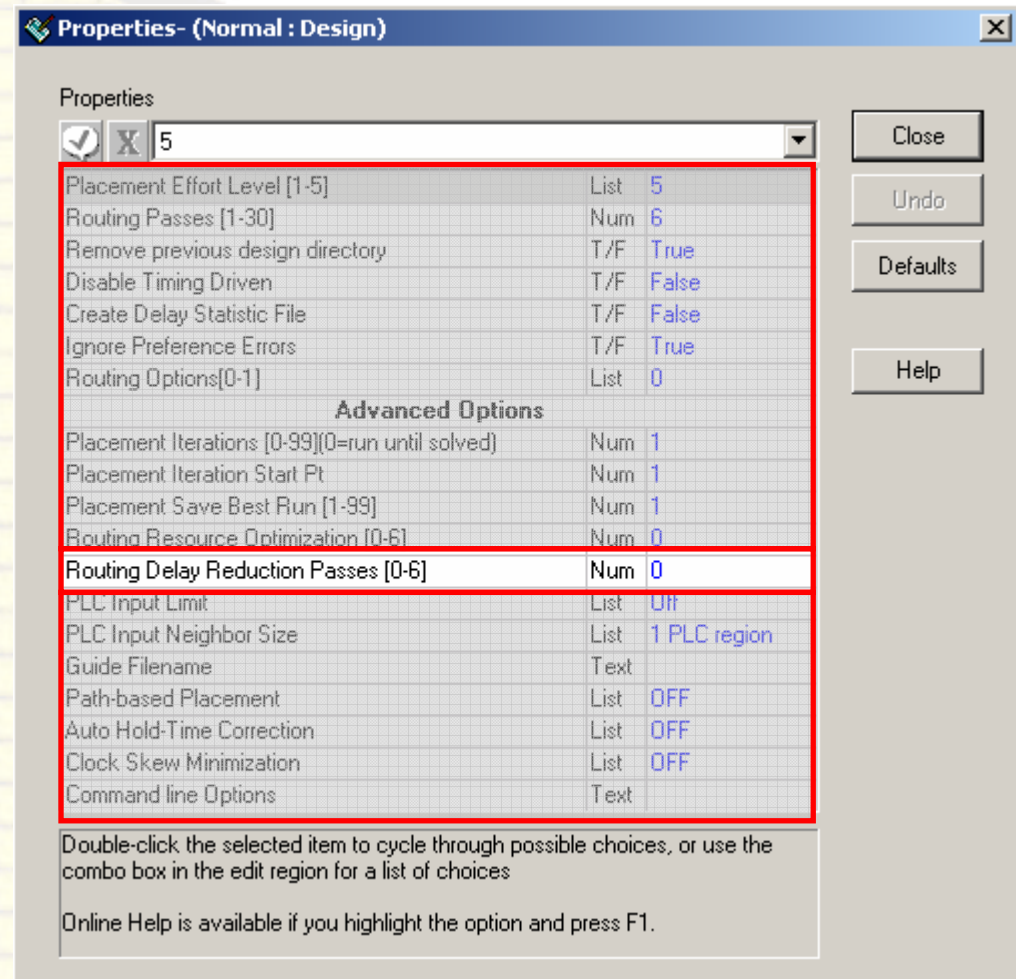
如果没有使用, 工具会运行1个cost-based cleanup pass.

如果需要即运行cost-based cleanup routing pass又运行delay-based cleanup routing passes, the cost-based passes 会先于the delay-based passes运行。



## 控制place and route

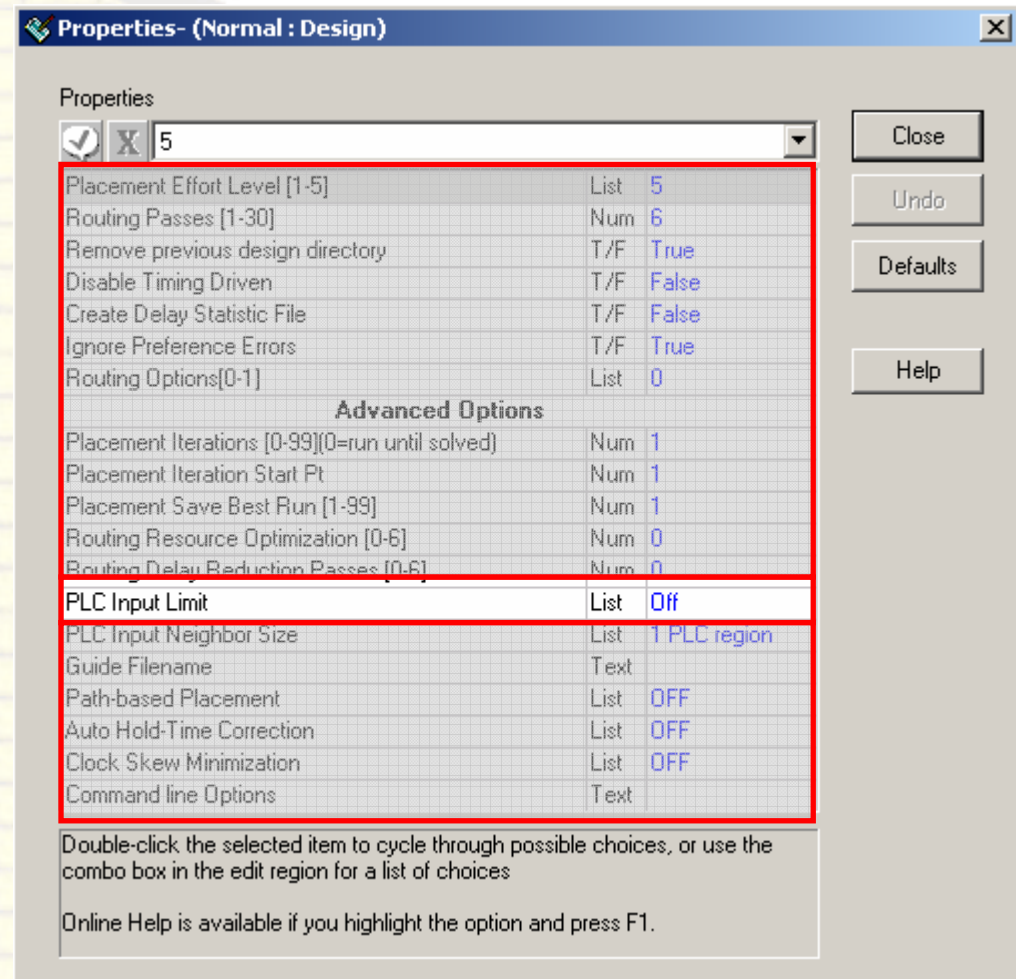
- 控制delay-based cleanup pass (在布线之前每个网络计算出确切的延时). 如果设计中始终有未布通网络时可以尝试此设置。
- 该选项确定需要运行delay-based cleanup routing的pass数目。
- 第一次delay-based cleanup routing pass 会有很大的性能提升. 运行大于5次之后基本不会再有性能提升。





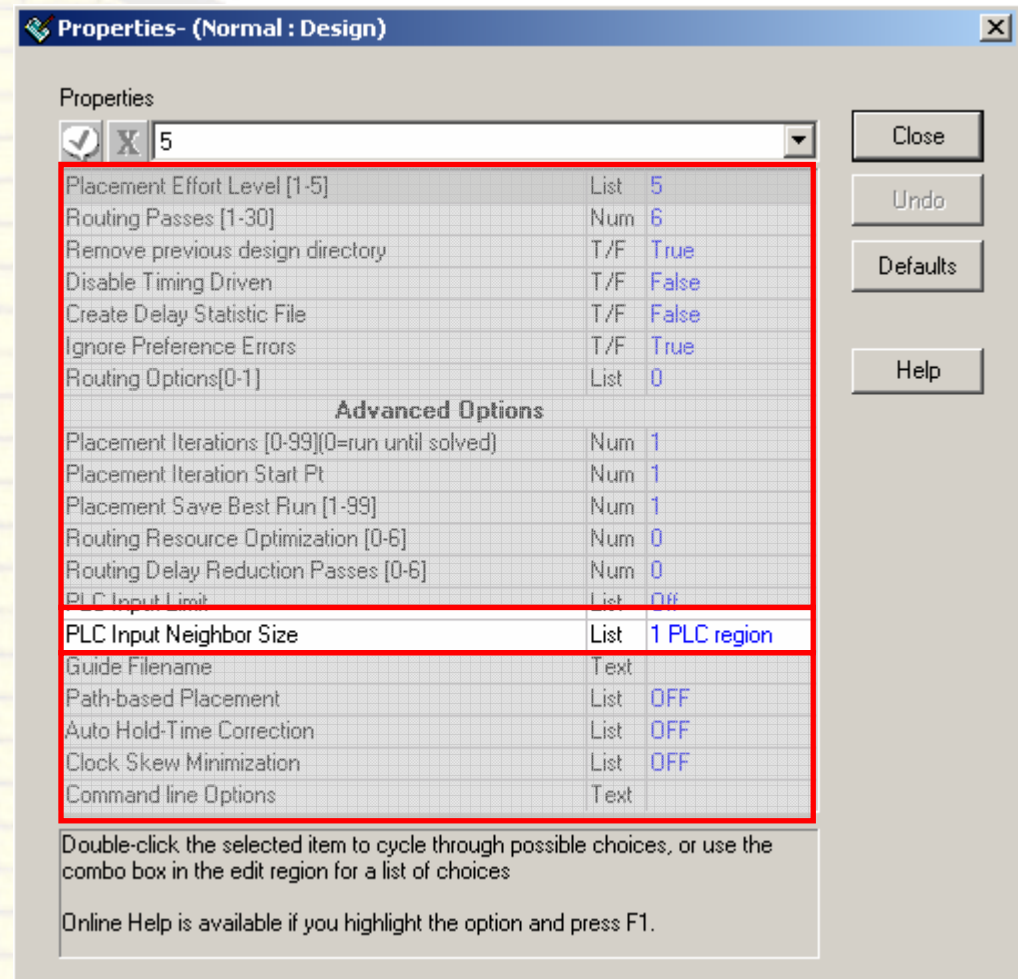
## 控制place and route

- 限制PLC (PFU/PFF) 输入信号数目以减少严重冲突区域的布线数量，四种选择：Off (0,默认), Low (1), midum(2) 和High (3).
- Off (0), 没有任何输入数量的限制只要总数不超过硬件的限制。
- Low, Medium,和High 允许的输入信号数量逐渐增多.
- 在严重布线冲突区域可以尝试设置为Low。负面的影响是降低Fmax。
- 如果确实产生了Fmax问题, 可以尝试Medium或High的设置同时设置不同的PLC Input Neighbor Size.



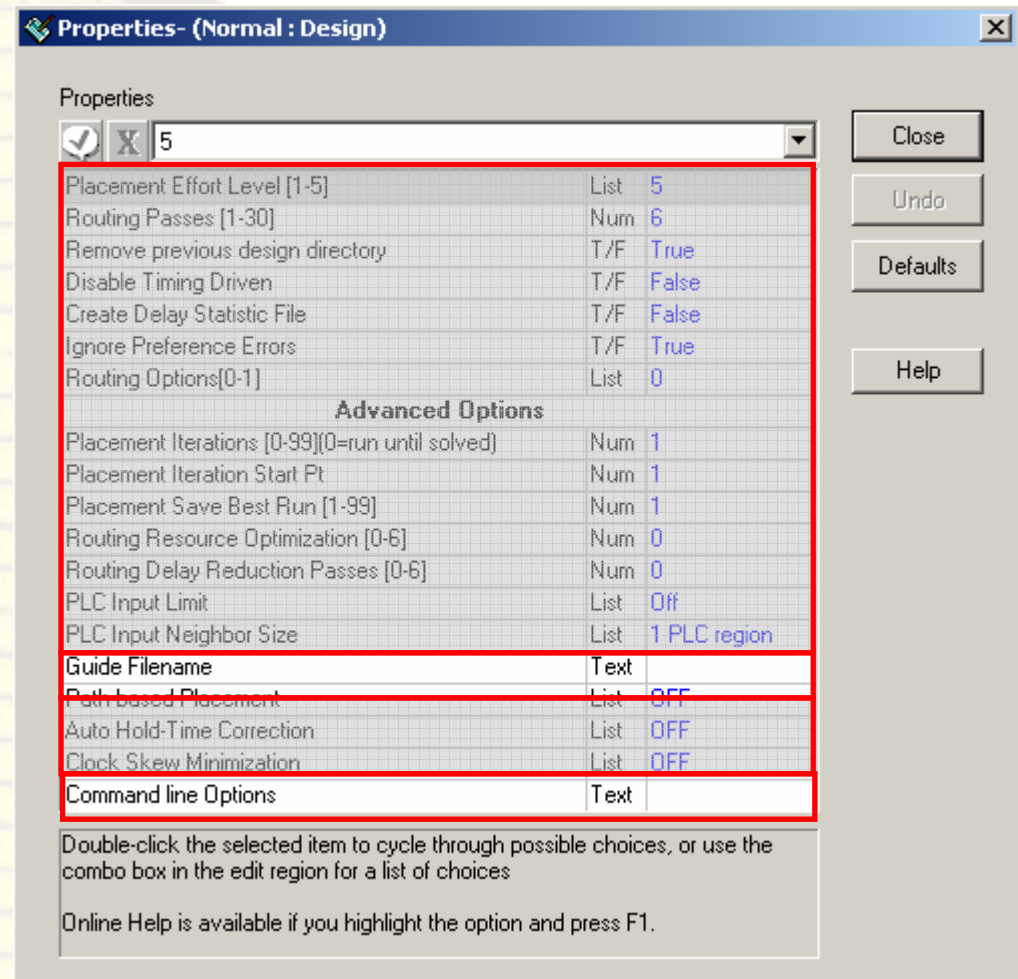
## 控制place and route

- 指定实施PLC输入控制的区域：  
1 PLC region (1, default)  
2X2 PLC region (2).



# 控制place and route

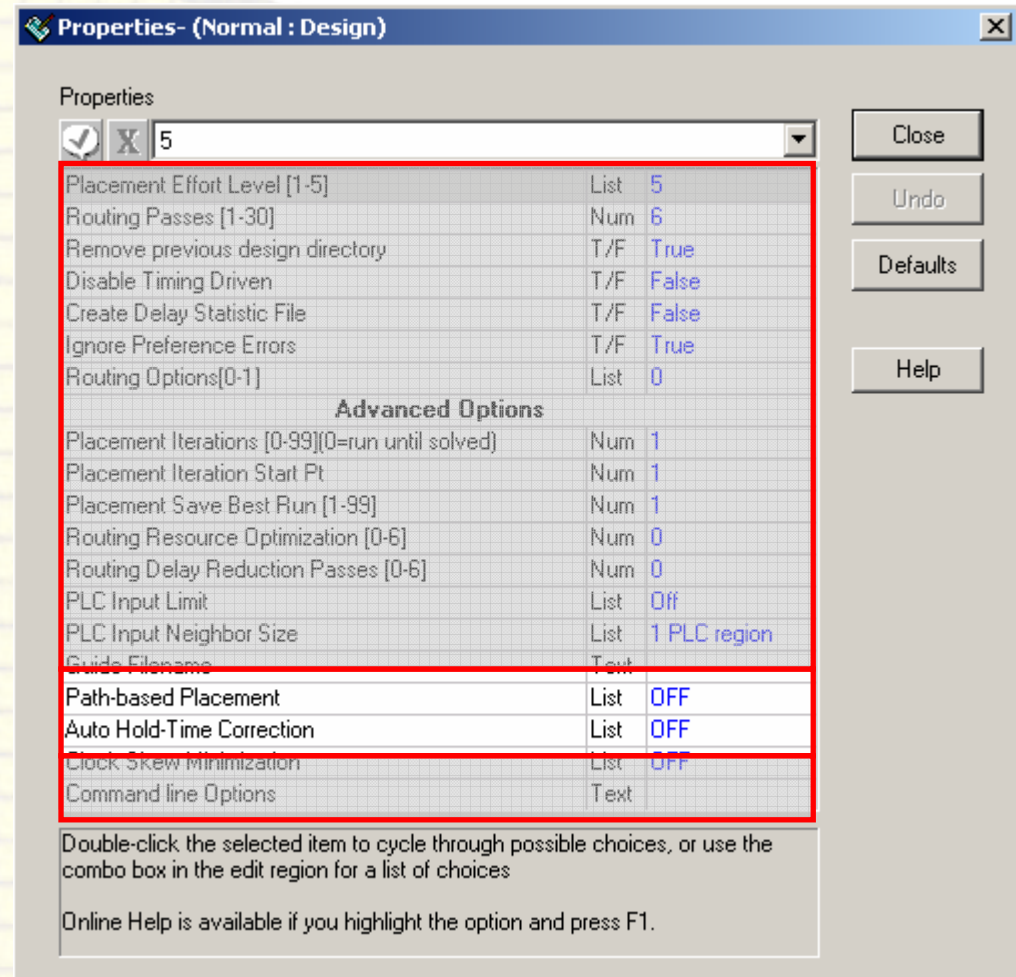
- 通过使用guide file 可以有效的减少PAR的运行时间：机理是监测logic database file (.ngd) 的改变，指导PAR 参照先前生成的physical database (.ncd) file 来进行新的Place & Route。
- 使用Guide file (例如prep\_1\_guide.ncd 是基于上次PAR结果重命名的NCD) 设计者应该同时在command line option输入-mf 80
- 所有的guided目标被placed and routed之后，PAR锁定guided components and macros的位置，然后在执行正常的布局布线操作。
- MF表示在guide PAR之前guiding objective 和guided objective之间最小的相同连接的百分比，MF只由net和component影响。





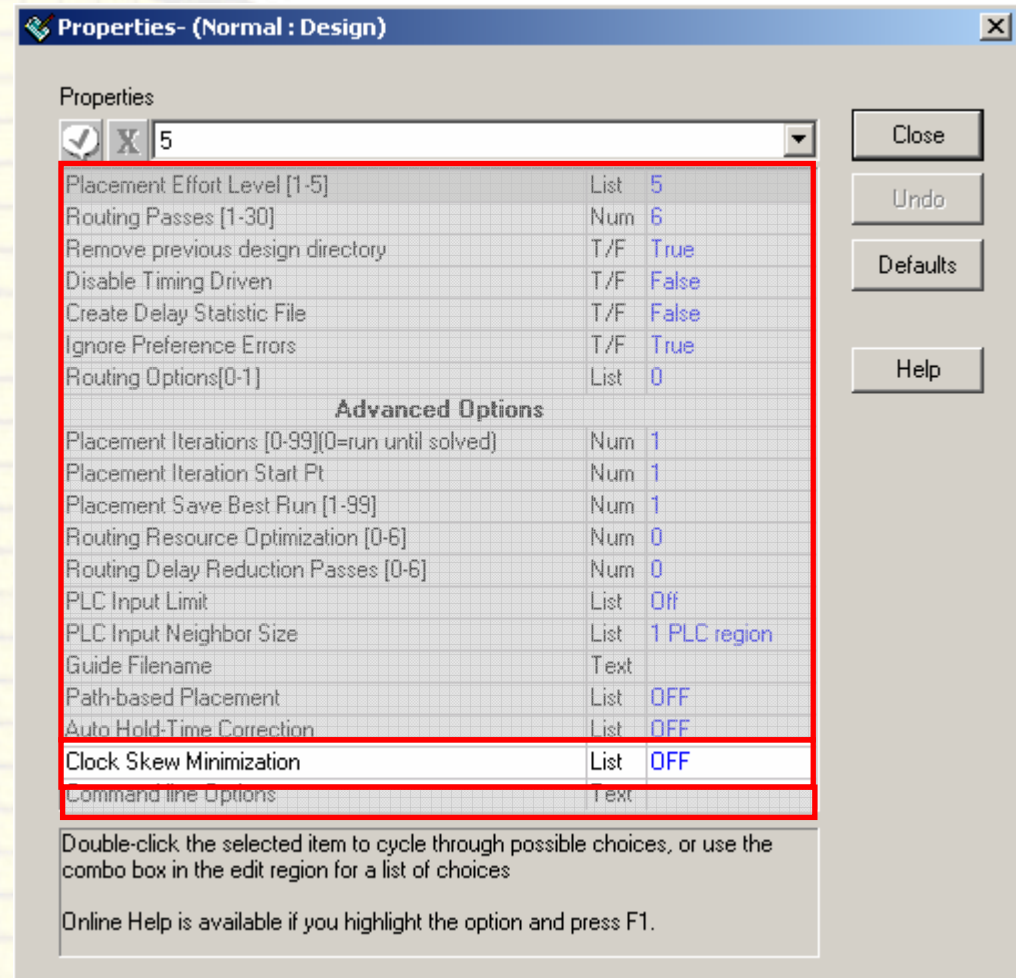
# 控制place and route

- 控制软件运行path-based Placement. Path-based Placement可以获得更好的性能和可预知的结果. 经测试平均能提高5%的性能, 同时布线时间显著增长.
- 选项选为ON: 直接连接input pad的寄存器如果出现hold time violation的情况布线器会自动插入额外的布线来进行延时补偿, 这样做的会对建立时间产生负面影响。



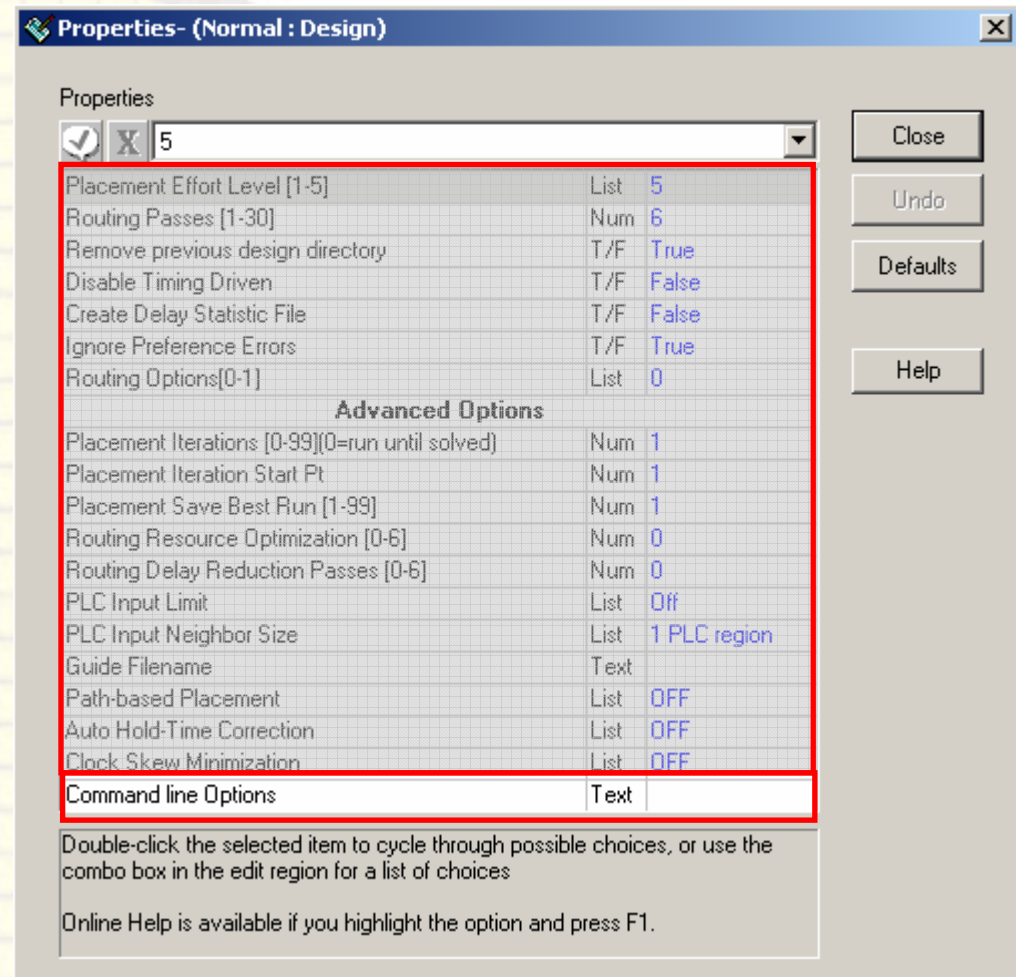
## 控制place and route

- 对那些未分配到全局时钟网络上的时钟信号进行均衡布线以减小时钟的 skew. 选项: Off (default), 1, and 2.
- 设为1: 根据预先计算的延时从时钟驱动管脚到时钟负载进行布线, 为了布线均衡时钟网络跨度不能太大。该设置对于时钟跨度不大同时要求 skew 较小的应用场合比较有效(例如 the bounding box span is within 2 to 3 PLCs) .
- 设为2: 先为 clock trunk 布线, 然后根据预先计算的延时从时钟 trunk 到时钟负载布线, 同样第二次布线该时钟网络跨度不能太大。此选项的时钟网络跨度较设置1大很多。



## 控制place and route

- Command line 允许键入命令行.
- The command line 是以上各种设置的备份方式。





## 控制place and route

---

- In the text box, type in the option and its value in the following format:
- **-exp *option=value***
- **-For example: -exp parPathBased=ON**

# Floorplanning the design

---

- 什么是 floorplanning?
- Floorplanning 是一种物理或逻辑的 partitioning 技术, 它可以控制和影响逻辑设计的物理布局 and 实现.
- 复杂FPGA 设计的管理
- 复杂的和大型的逻辑设计管理是非常困难的,但是性能的优化却更具挑战性, 当增加一个或修改一个模块时整个系统的优化需要多次叠代. 复杂的大型系统设计通常有如下需求:
  - 模块化、层次化和增量设计方法。
  - 使得设计管理和优化易于实现的软件。
  - IP的使用。
  - 前期优化模块的重复使用。

# Floorplanning the design

---

- **Floorplanning**和传统设计流程的比较:
- 1.传统设计流程中, 即使各个模块再集成之前经过优化已经满足了时序要求, 但是集成之后依然有可能不能满足系统**performance**要求。
- 2.即使整个模块已经满足的时序要求, 对一个模块的修改将会影响到其他的模块。
- 3.为了满足系统性能要求而重新进行优化需要实施多次叠代, 耗费大量时间。
- 4.**Floorplanning**协助进行各个独立的设计、测试和优化, 同时能否保持每个模块优化后的特性。
- 5. 模块集成到系统中时仅仅需要在模块之间进行系统优化。
- 6. **ispLEVER software**自动放置已经定义好的模块或是在用户控制下完成指定模块的放置。



# Floorplanning the design

---

- When to floorplan?
- 1.保持模块性能
  - Floorplanning 通过把相关的逻辑Group到一个特定的Region来保持该模块的性能。
- 2.设计重用
  - 对于经常使用的模块, 设计者可以基于验证后的设计生成库文件 , 库文件仅仅包括VHDL或Verilog source code 和 grouping attributes 和一些必要的注释, 例如performance和size。
- 提升设计性能的策略:
- 1.基于**design hierarchy**定义**region**。
- 2.如果**critical path**很长并且跨越多个**module**, 可以基于关键路径定义**region**。

# Floorplanning the design

---

- Design floorplanning方法:
- **1.PGROUP Physical Constrain**
  - 通过VHDL and Verilog HDL source code 中的attribute或通过.prf 文件中的 physical Preference。
  - PGROUP有严格的hierarchy限制,相同hierarchies的module才能PGROUP。
- **2.UGROUP Logic constrain**
  - 通过VHDL and Verilog HDL source code 中的attribute或通过.prf 文件中的 physical Preference。
  - UGROUP允许在不同级别hierarchies的block之间GROUP甚至没有任何 hierarchy限制。

# Floorplanning the design

## 3.Floorpanner 图形化用户界面

- 交互式的界面，非常方便的为模块指定Region的参数。
- 在HDL代码中键入PGROUP或UGROUP attribute相对于采用GUI方式要容易很多。因为GUI方式需要Floorplanner中Load大型网表，然后找到相应的模块并通过点击鼠标设置PGROUP或UGROUP。
- GUI不能把 grouping attributes反向标注到HDL代码, 因此GUI操作在每一个新的设计中都要重复进行。

### Floorplanner GUI在下列情形下使用比较适合:

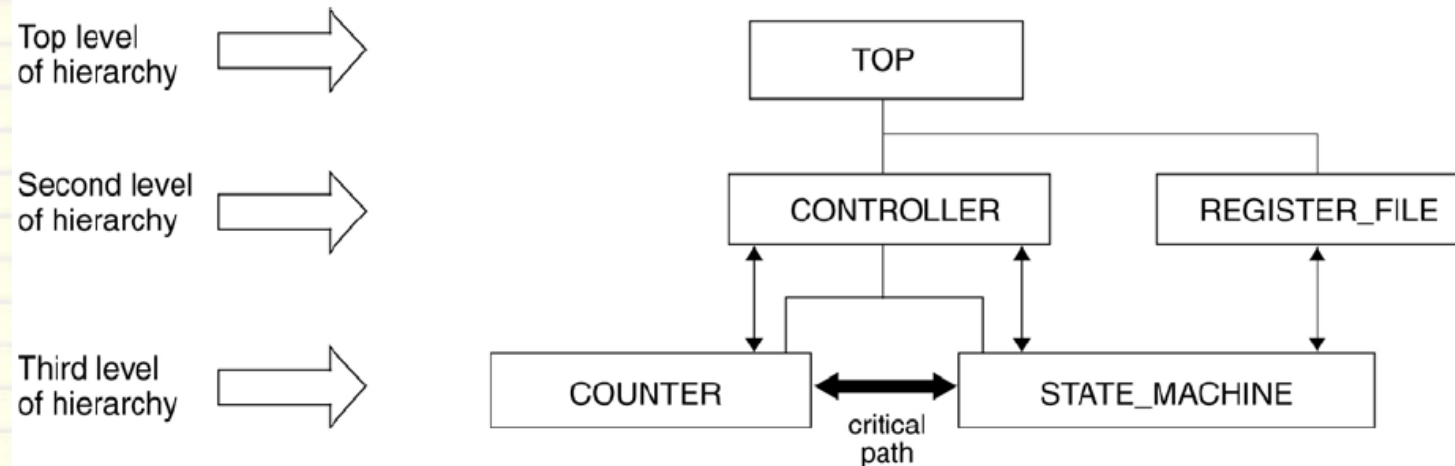
- 在图形化的界面下观察design's logical hierarchy .
- 观察已经存在的PGROUPs and UGROUPs.
- 重新定义Regions size和boundary boxes (BBOXes).
- 图形界面下的放置Regions, PGROUPs, and UGROUPs 然后运行 map, place, route和TRACE。



# Floorplanning the design

## 4.简单的例子: PGROUP和UGROUP

Figure 99: PGROUP Same Hierarchy Example, PGROUP CONTROLLER



For example, if the following Synplify attribute is in the Verilog HDL file,

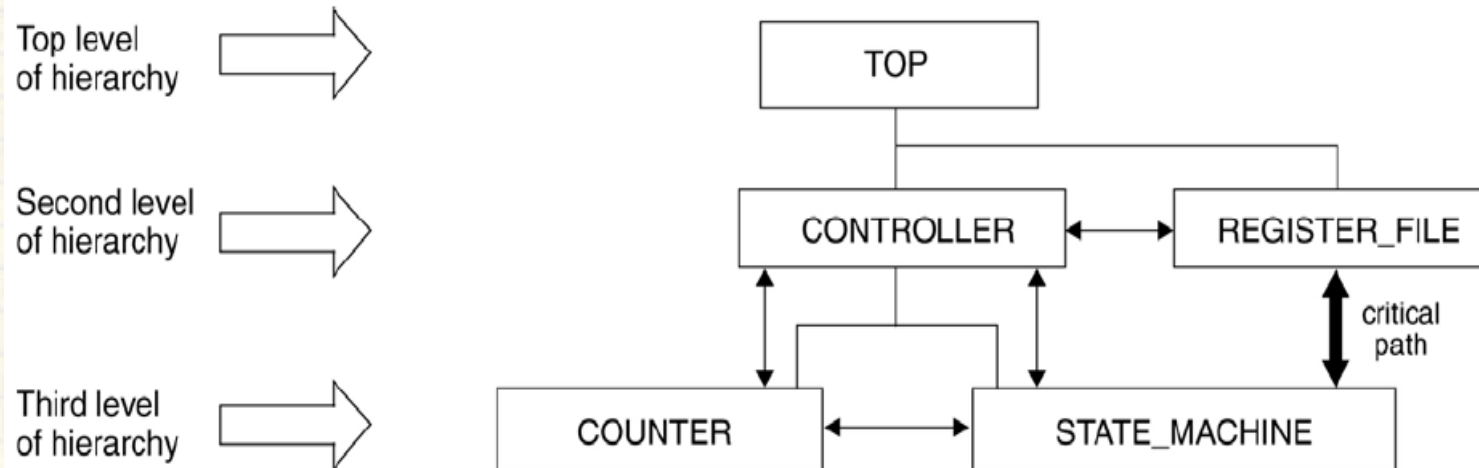
```
module CONTROLLER (<port_list>)  
/* synthesis pgroup="CONTROL_GROUP" */;
```

## Floorplanning the design

**COUNTER**和**STATE\_MACHINE**在FPGA将被**Grouped** 成一个**Boundary Box**。  
现在假定**COUNTER** MAP到**PFU\_0** and **PFU\_1**，**STATE\_MACHINE**被MAP到**PFU\_2**。在**MAP**过程生成的**.prf**文件中**preference** 将是：

- PGROUP "TOP/CONTROLLER/CONTROL\_GROUP"
- COMP "PFU\_0"
- COMP "PFU\_1"
- COMP "PFU\_2"

**Figure 100: UGROUP Different Hierarchy Example, UGROUP REGISTER\_FILE and STATE MACHINE**



# Floorplanning the design

For example, if the following Synplify attributes are in the Verilog HDL file:

```
module REGISTER_FILE (<port_list>) /*synthesis  
ugroup="CRITICAL_GROUP" */;
```

and

```
module STATE_MACHINE (<port_list>) /*synthesis  
ugroup="CRITICAL_GROUP" */;
```

**REGISTER\_FILE**和**STATE\_MACHINE** 将在FPGA中被grouped 到一个 **boundary box**.

现在假定 **REGISTER\_FILE** 被映射到 **PFU\_4 and PFU\_5**, **STATE\_MACHINE**被映射到**PFU\_2**.在**MAP**过程生成的**.prf**文件中**preference** 将是:

- PGROUP "TOP/CONTROLLER/CONTROL\_GROUP"
- COMP "PFU\_0"
- COMP "PFU\_1"
- COMP "PFU\_2";



# Floorplanning the design

如果**PGROUP attributes**替代**UGROUP attributes**.

```
module REGISTER_FILE (<port_list>) /*synthesis  
pgroup="CRITICAL_GROUP" */;
```

and

```
module STATE_MACHINE (<port_list>) /*synthesis  
pgroup="CRITICAL_GROUP" */;
```

then the resulting preference generated by map in the .prf file would be:

```
PGROUP "TOP/CONTROLLER/STATE_MACHINE/CRITICAL_GROUP"  
COMP "PFU_3"  
PGROUP "TOP/REGISTER_FILE/CRITICAL_GROUP"  
COMP "PFU_4"  
COMP "PFU_5";
```

在这个例子中使用 **PGROUP attributes**, **STATE\_MACHINE** 模块被Group到一个**bounding box**, **REGISTER\_FILE** 模块被Group到另外一个单独的**bounding box**。  
图100中的关键路径并没有被优化。

# Floorplanning the design

## Design floorplanning过程

---一个例子在 ispLever 安装目录下:

\ispTOOLS6\_1\examples\Tutorial\timing\_closure\_tutor\or1200

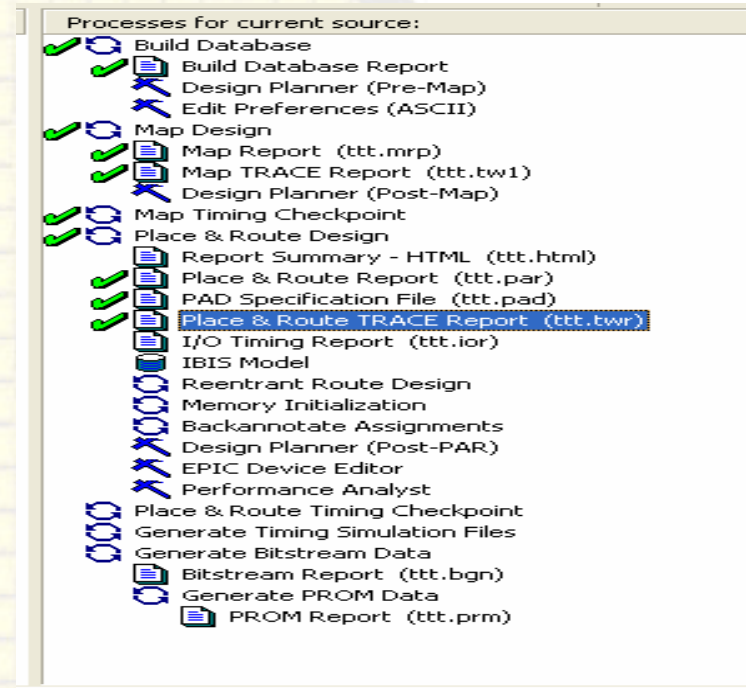
### 1. 检查TRACE report 找到需要提升性能的关键路径和模块.

如果很少 (< 5)的关键路径不能满足Fmax要求, 可以考虑使用 (UGROUP)沿关键路径创建 GROUP.

在“Processes for current source”窗口中双击Place & Route TRACE Report , ttt.twr will be generated automatically.

在Window菜单中选择Text Editor, choose **File Open**. 文件类型选择 **All Files**, 找到ttt.twr点击**OPEN**。

你会发现和关键路径相关的几个模块:  
**or1200\_except** and **or1200\_freeze**.



# Floorplanning the design

## 2. Use Design Planning and Grouping

-检查关键路径

- a. 运行ispLEVER post-map Design Planner 然后open **Floorplan View**, **Post-Mapped View**, **Package View**, and **Pre-Mapped View**.
- b. 最大化Floorplan View window 选择**Tools > Path Tracer**.
- c. 在**Path Tracer**中点击**Query**.

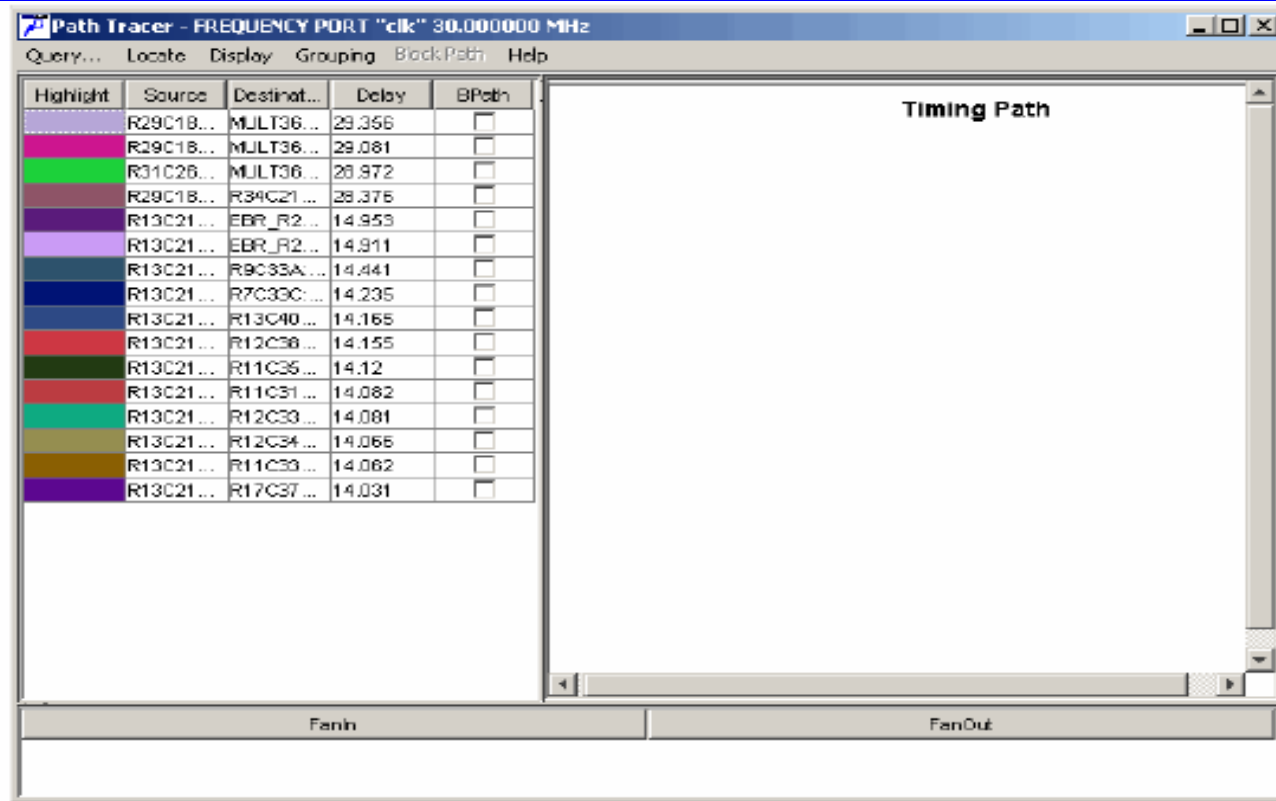


- d. 在Timing Query 对话框, 在Preference list中选择**FREQUENCY PORT “clk” 30.000000 MHz** 然后点击**Query**.

*Path Tracer*会显示一系列的组合逻辑路径。

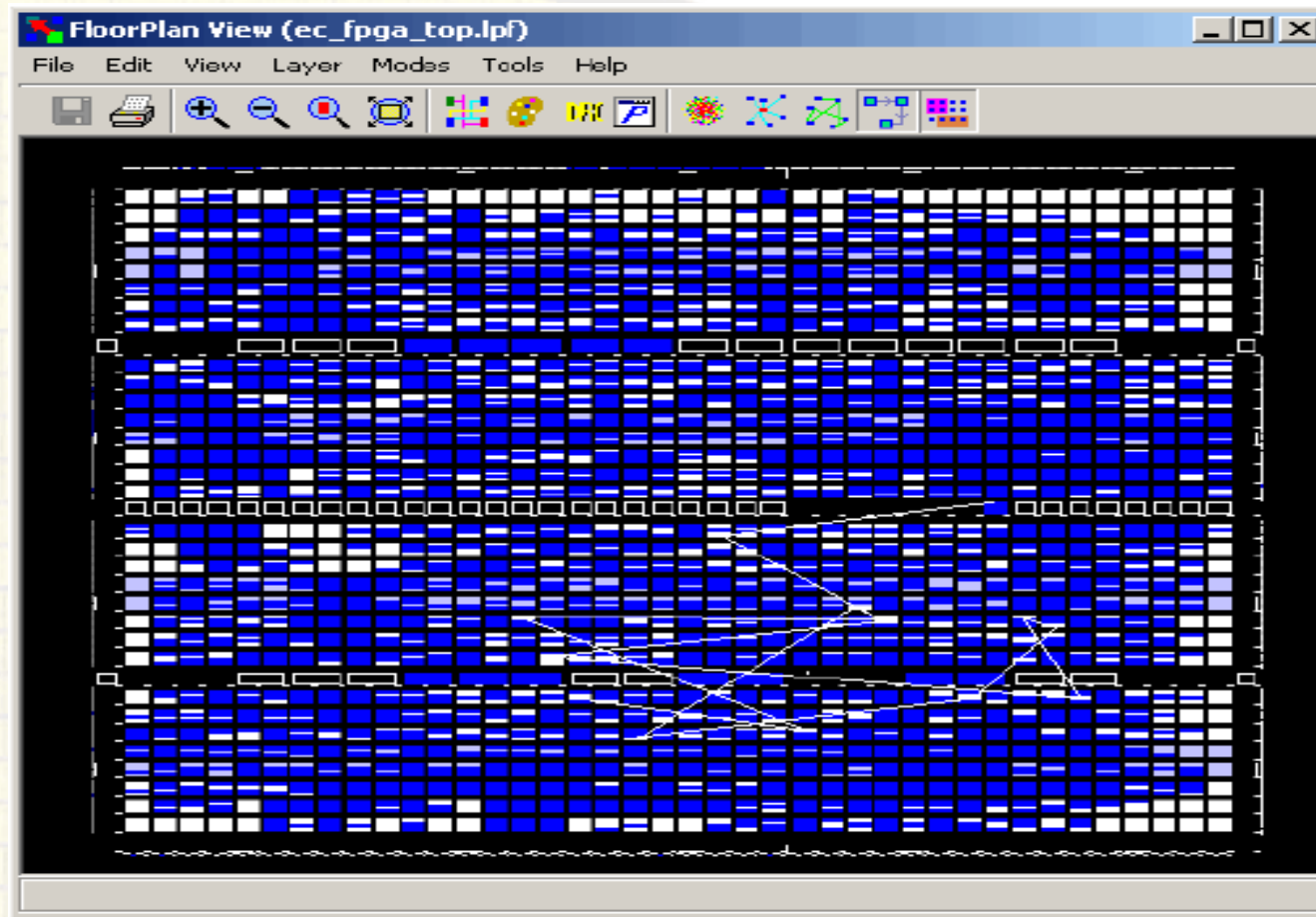


# Floorplanning the design




- e. 点击最顶行的highlight color 打开 Change Display Color 对话框。
- f. 选择白色, 将会提供最好的对比度, 然后点击 OK。
- g. 依然选中最顶行, 点击 **Locate**, 如果Floorplan View window依然不可见, 选择 Floorplan View window。

# Floorplanning the design

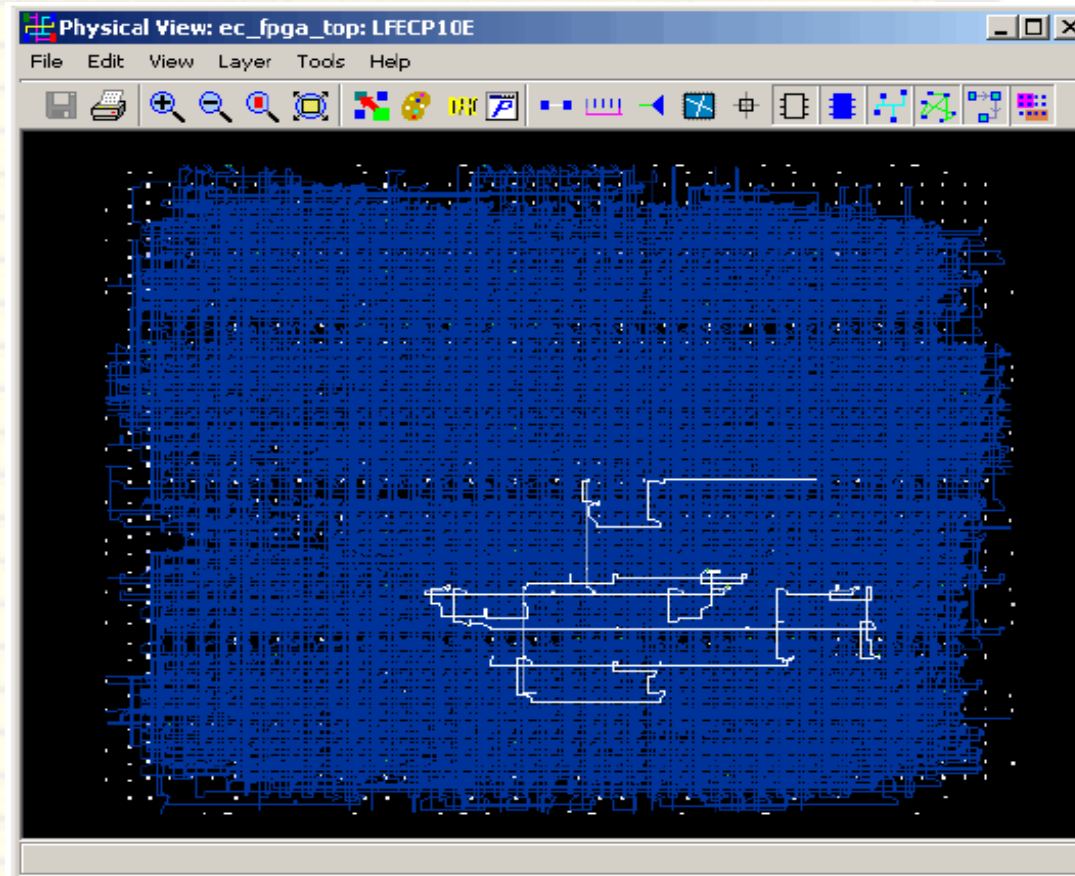


沿着关键路径的slices被高亮为白色，飞线表示逻辑连接。.

# Floorplanning the design

- h. From Floorplan View, click the **Physical View**  button on the toolbar.

*Physical View* 被打开并显示相关的Path和物理布线。





# Floorplanning the design

---

i. 为了提高显示对比度,可以设置如下选项:

- >选择 **Tools > Color Legend**.

- >在对话框中选择 **General Routing Display** tab

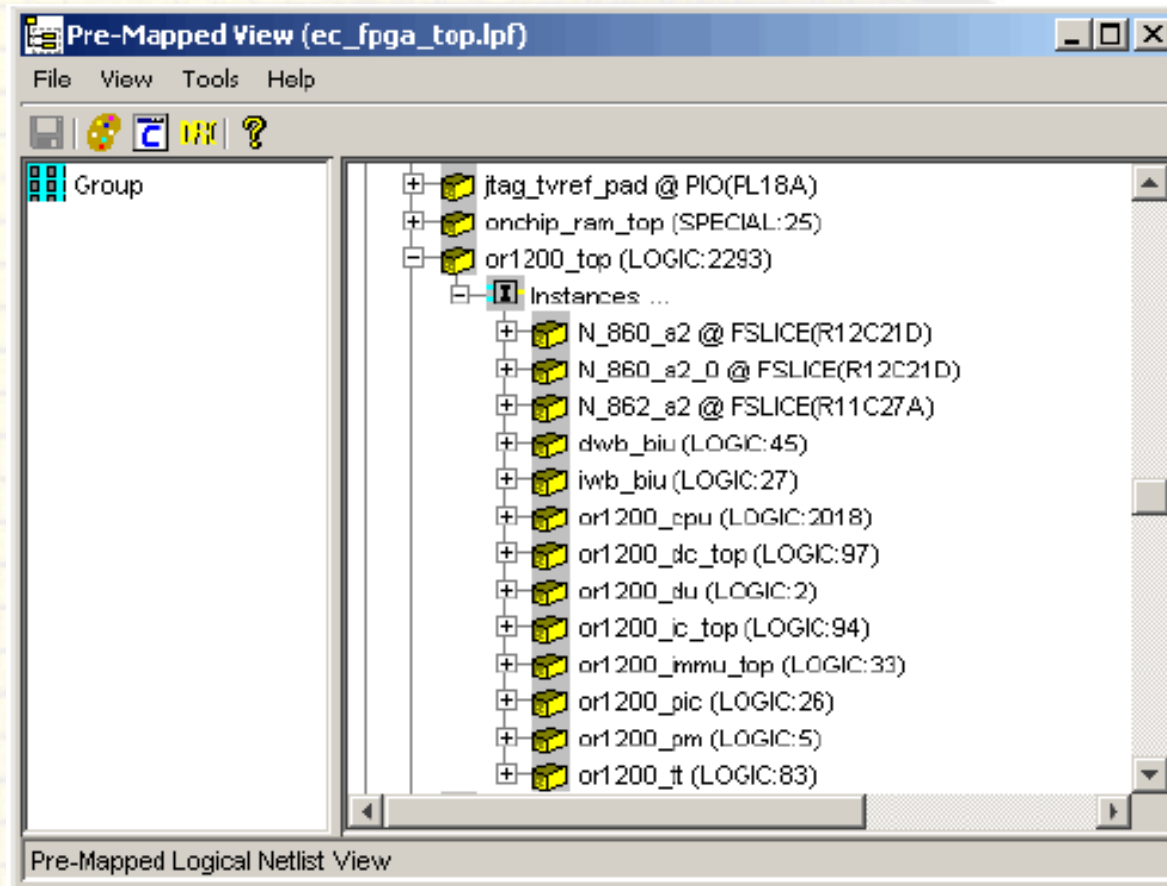
- >为**Routed Connection**选择颜色框 选择**dark color**.

- >点击 **OK**, 然后点击**Apply**.

# Floorplanning the design

## -检查design hierarchy

a. 在Design Planner Control选择 **View > Pre-Mapped View**.



# Floorplanning the design

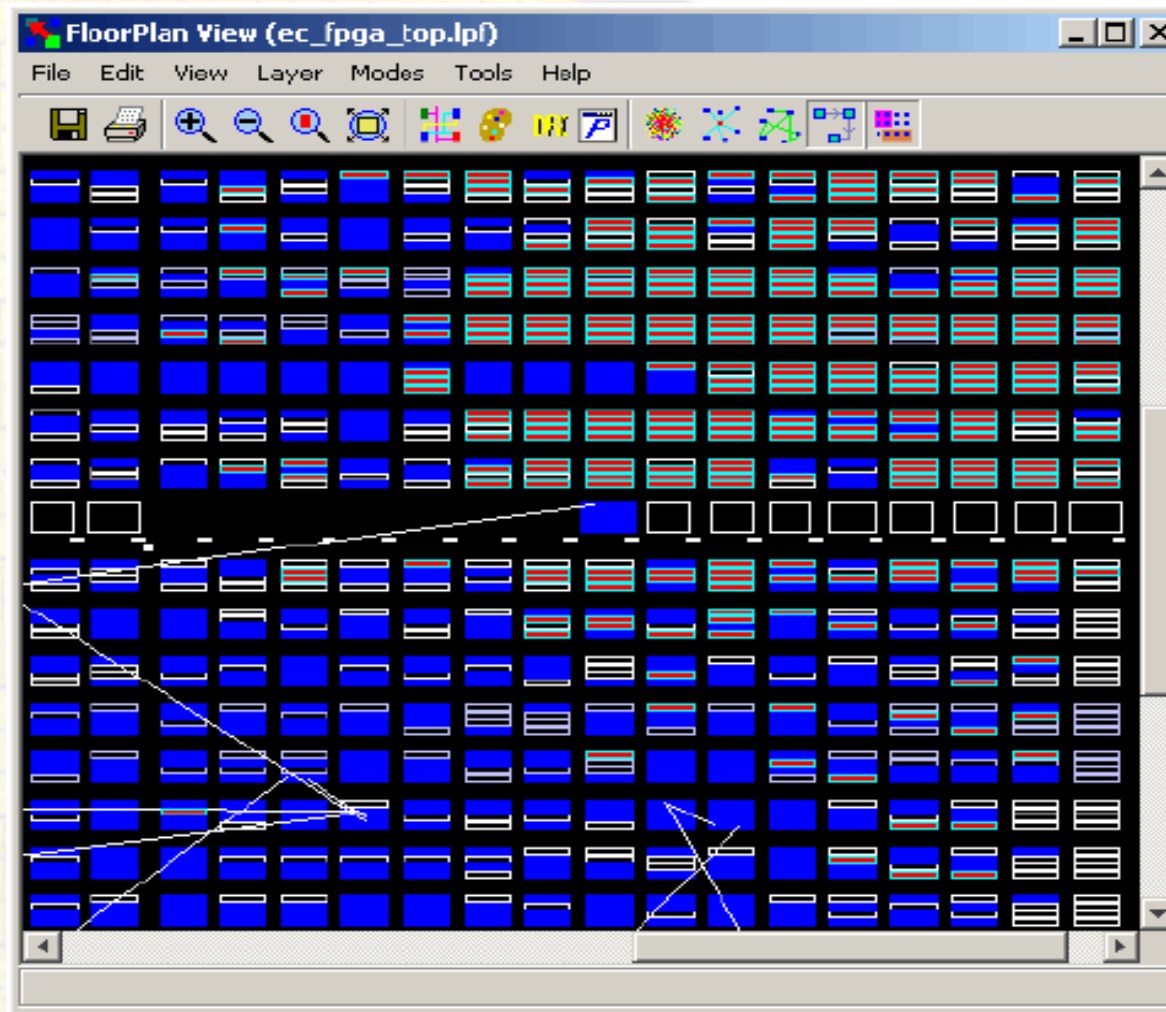
---

- b. 展开 Instances folder, 然后锁定并展开 **or1200\_top (LOGIC: 2293)** folder. 注意 LOGIC: 2293 指出了这个逻辑模块的资源使用情况。
- c. 在 or1200\_top (LOGIC: 2293) folder 展开 **Instances** folder。
- d. 锁定 **or1200\_cpu (LOGIC: 2018)** folder, 展开 Instances。
- e. 锁定 **or1200\_except (LOGIC: 308)** folder, 点击鼠标右键, 选择 **Locate > Floorplan View**.

选定的模块在 *Floorplan View* 被高亮。



# Floorplanning the design



# Floorplanning the design

---

f. 在Floorplan View中 **View > Highlight > Light Blue.** 和or1200\_except branch 相关的element都被高亮。

g. 在Pre-Mapped view, 锁定 **or1200\_freeze (LOGIC: 29)** folder 点击鼠标右键 选择**Locate > Floorplan View.**选中的模块在Floorplan View中被高亮。

h.在Floorplan View中选择**View > Highlight > Green.**

>通过这些view, 你可以看到逻辑被布局在什么地方, 每个模块的逻辑资源使用量, 关键路径相对于这些资源的位置。.

# Floorplanning the design

## -生成 logic group (UGROUP)

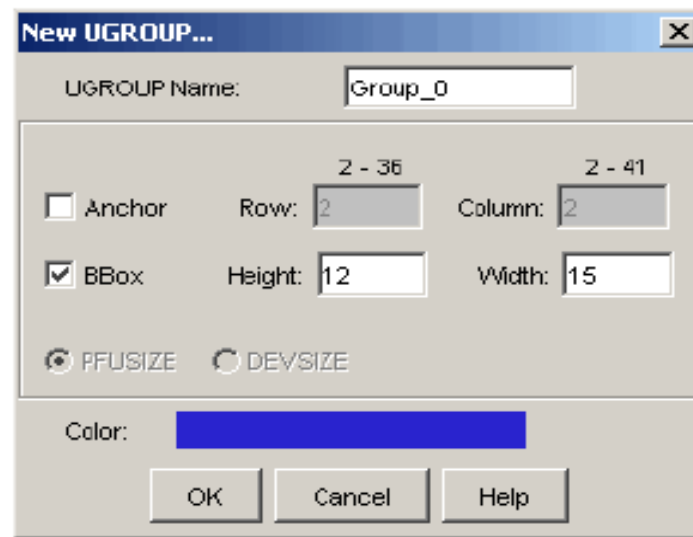
a. 在 Pre-Mapped View, 使用Ctrl+Click 在or1200\_cpu (LOGIC: 2018) folder选中系列模块:

>or1200\_except

>or1200\_freeze

b. 右击鼠标选择 **new UGROUP**。

c. 在 New UGROUP 对话框中选择**BBox**。





# Floorplanning the design

---

d. 指定Height **12** , Width**15** 然后点击 **OK**.Group\_0 将会出现在Pre-Map View的Group窗口。

e. 在Pre-Mapped View,选择 **File > Save**.

Design Planner 保存UGROUP 到 logical preference file. 在下次Mapping时会被写入physical preference file。

f. 在 Design Planner Control, 选择**File > Exit**.

---

Tips:

1.Group\_0指导 PAR 把选定模块的逻辑放置在一个 15 x 12 area (180 PFUs = 720 slices) 。

2.720个slices足够容纳337logic components(or1200\_except (LOGIC: 308) + or1200\_freeze (LOGIC:29)).

3.Lattice 建议添加的bounding box (BBox) 的资源要超出Grouped modules资源20% 。

---

# Floorplanning the design

## -重新运行**MAP, place & route**

- a. 在“Processes for current source” 窗口双击 **Map design**。
- b. 在window菜单选择Text Editor，选择 **File > Open**. 打开**ec\_fpga\_top.prf** 。
- c. 注意到 group\_0 physical group (PGROUP) 已经产生了：

```
>PGROUP "Group_0" BBOX 12 15 PFUSIZE
```

```
>COMP "or1200_top/or1200_cpu/SLICE_405"
```

```
>COMP "or1200_top/or1200_cpu/or1200_except/SLICE_473"
```

```
.
```

```
.
```

- d. 双击 **Place & route design** 重新运行布局布线。
- e. 双击 **Place & route Trace Report** 重新生成布线后的静态时序报告。

# Floorplanning the design

---

## f. 查看 **TRACE report**.

如果性能没有得到提升，可以尝试额外的 **grouping**和**anchoring**，这会让工具自动放置 **Bounding Box**效果好很多。

**g.** 如果性能已经得到改善，可以把**UGROUP attribute**移植到**HDL**源代码中了。



# Floorplanning the design

```
//  
// Instantiation of freeze logic  
//  
or1200_freeze or1200_freeze(  
    .clk(clk),  
    .rst(rst),  
    .multicycle(multicycle),  
    .flushpipe(flushpipe),  
    .extend_flush(extend_flush),  
    .lsu_stall(lsu_stall),  
    .if_stall(if_stall),  
    .lsu_unstall(lsu_unstall),  
    .force_dslot_fetch(force_dslot_fetch),  
    .abort_ex(abort_ex),  
    .du_stall(du_stall),  
    .mac_stall(mac_stall),  
    .genpc_freeze(genpc_freeze),  
    .if_freeze(if_freeze),  
    .id_freeze(id_freeze),  
    .ex_freeze(ex_freeze),  
    .wb_freeze(wb_freeze),  
    .icpu_ack_i(icpu_ack_i),  
    .icpu_err_i(icpu_err_i)  
); /* synthesis UGroup="group_0" PBBox="12,15" */;
```

# Floorplanning the design

```
//  
// Instantiation of exception block  
//  
or1200_except or1200_except(  
    .clk(clk),  
    .rst(rst),  
    .sig_ibuserr(except_ibuserr),  
    .sig_dbuserr(except_dbuserr),  
    .sig_illegal(except_illegal),  
    .sig_align(except_align),  
    .sig_range(1'b0),  
    .sig_dtlbmiss(except_dtlbmiss),  
    .sig_dmmufault(except_dmmufault),  
    .sig_int(sig_int),  
    .sig_syscall(sig_syscall),  
    .sig_trap(sig_trap),  
    .sig_itlbmiss(except_itlbmiss),  
    .sig_immufault(except_immufault),  
    .sig_tick(sig_tick),  
    .branch_taken(branch_taken),  
    .icpu_ack_i(icpu_ack_i),  
    .icpu_err_i(icpu_err_i),  
    .dcpu_ack_i(dcpu_ack_i),  
    .dcpu_err_i(dcpu_err_i),  
    .genpc_freeze(genpc_freeze),  
    .id_freeze(id_freeze),  
    .ex_freeze(ex_freeze),  
    .wb_freeze(wb_freeze),  
    .if_stall(if_stall),
```

# Floorplanning the design

```
.if_pc(if_pc),  
.lr_sav(lr_sav),  
.flushpipe(flushpipe),  
.extend_flush(extend_flush),  
.except_type(except_type),  
.except_start(except_start),  
.except_started(except_started),  
.except_stop(except_stop),  
.ex_void(ex_void),  
.spr_dat_ppc(spr_dat_ppc),  
.spr_dat_npc(spr_dat_npc),  
  
.datain(operand_b),  
.du_dsr(du_dsr),  
.epcr_we(epcr_we),  
.eear_we(eear_we),  
.esr_we(esr_we),  
.pc_we(pc_we),  
.epcr(epcr),  
.eear(eear),  
.esr(esr),  
  
.lsu_addr(dcpu_adr_o),  
.sr_we(sr_we),  
.to_sr(to_sr),  
.sr(sr),  
.abort_ex(abort_ex)  
);/* synthesis UGroup="group_0" PBox="12,15" */;
```