GOWIN高云

智慧逻辑 定制未来

GowinSynthesis

用户指南

SUG550-2.0, 02/28/2025

免责声明

本文档并未授予任何知识产权的许可，并未以明示或暗示，或以禁止反言或其它方式授予任何知识产权许可。除高云半导体在其产品的销售条款和条件中声明的责任之外，高云半导体概不承担任何法律或非法律责任。高云半导体对高云半导体产品的销售和／或使用不作任何明示或暗示的担保，包括对产品的特定用途适用性、适销性或对任何专利权、版权或其它知识产权的侵权责任等，均不作担保。高云半导体对文档中包含的文字、图片及其它内容的准确性和完整性不承担任何法律或非法律责任，高云半导体保留修改文档中任何内容的权利，恕不另行通知。高云半导体不承诺对这些文档进行适时的更新。

版本信息

日期	版本	说明
2019/08/02	1.0	初始版本。
2019/12/05	1.1	增加综合前后对象命名规则说明（适用于高云半导体云源软件 1.9.3 及以后版本）。
2020/03/03	1.2	增加支持 VHDL 语法的设计（适用于高云半导体云源软件 1.9.5 及以后版本）。
2020/05/29	1.3	● 修改综合命名规则； ● 增加 syn_srlstyle、syn_noprune 属性。
2020/09/14	1.4	增加 black_box_pad_pin 属性。
2021/10/28	1.5	● 增加 parallel_case 、syn_black_box 属性； ● 修改第 6 章 Report 用户文档。
2023/06/30	1.6	更新第 5 章综合约束支持。
2024/05/09	1.7	● 更新第 4 章硬件描述语言代码支持； ● 更新第 5 章综合约束支持； ● 更新第 6 章 Report 用户文档中的截图。
2024/08/09	1.8	● 新增综合属性约束对象数量说明； ● 更新图 6-4 Timing。
2024/12/31	1.9	更新第 5 章综合约束支持。
2025/02/28	2.0	● 新增 syn_radhardlevel 属性约束； ● 更新 syn_maxfan 描述； ● 更新 syn_looplimit 作用对象。

目录

图目录

表目录

1 关于本手册

1.1 手册内容

本手册主要描述高云半导体综合工具（GowinSynthesis）的功能及操作，旨在帮助用户快速熟悉 GowinSynthesis 软件的相关功能，指导用户设计，提高设计效率。本手册中的软件界面截图参考高云半导体云源软件（以下简称云源）1.9.11.01 版本，因软件版本升级，部分信息可能会略有差异，具体以用户软件版本的信息为准。

1.2 相关文档

通过登录高云半导体网站 www.gowinsemi.com.cn 可以下载、查看以下相关文档：SUG100，Gowin 云源软件用户指南。

1.3 术语、缩略语

本手册中的相关术语、缩略语及相关释义请参见表 1-1。

表 1-1 术语、缩略语

术语、缩略语	全称	含义
BSRAM	Block Static Random Access Memory	块状静态随机存储器
DSP	Digitial Signal Processing	数字信号处理
FPGA	Field Programmable Gate Array	现场可编程门阵列
FSM	Finite State Machine	有限状态机
GSC	Gowin Synthesis Constraint	GowinSynthesis综合约束文件
SSRAM	Shadow Static Random Access Memory	分布式静态随机存储器

1.4 技术支持与反馈

高云半导体提供全方位技术支持，在使用过程中如有任何疑问或建议，可直接与公司联系：

网址：www.gowinsemi.com.cn

E-mail：support@gowinsemi.com

Tel：+86 755 8262 0391

2 概述

本手册是高云半导体 RTL 设计综合工具 GowinSynthesis 的用户使用说明。

GowinSynthesis 是高云半导体（以下简称高云）自主研发的综合工具，采用了高云原创的 EDA 算法，基于产品硬件特性及硬件电路资源情况，实现 RTL 设计提取、算数优化、推导置换、资源共享、并行综合以及映射等技术，可快速对用户的 RTL 设计进行优化处理、资源检查、时序分析。

GowinSynthesis 针对高云 FPGA 芯片，为 FPGA 设计人员提供了最有效的设计实现方法。在设计实现方面，提供了时序分析、资源检查、原语逻辑分析等功能；以及提供了详细的综合信息。GowinSynthesis 生成基于高云器件原语库的综合后网表，可作为高云布局布线工具的输入文件，实现了面积和速度最优平衡结果，提高了软件编译效率，以及布通率。该软件具有如下特点：

- 支持 Verilog/SystemVerilog、VHDL 设计以及混合设计输入
- 支持超大规模设计，可为复杂可编程逻辑设计提供优秀的综合解决方案
- 支持查找表、寄存器、锁存器、算术逻辑单元的推断映射
- 支持 memory 推断映射以及与逻辑资源平衡
- 支持 DSP 的推断映射以及与逻辑资源平衡
- 支持 FSM 的综合优化
- 支持综合属性及综合指令，以满足不同应用条件下综合结果的要求

3 GowinSynthesis 的使用方法介绍

3.1 GowinSynthesis 输入输出

GowinSynthesis 以工程文件（.prj）格式读入用户 RTL 文件，工程文件由云源自动生成。GowinSynthesis 工程文件中除指定用户 RTL 文件外，同时还指定了综合器件信息、用户约束文件信息（综合属性约束文件）、综合后网表文件（.vg）信息及部分综合选项，如综合顶层模块（top module）指定、文件包含路径（include path）指定等。

3.2 使用 GowinSynthesis 进行综合

在云源 Process 窗口右键单击 Synthesize，选择 Configuration，该页面可以指定 top module，设置 include path，选择支持语言版本，配置相关综合选项。

在云源 Process 窗口双击 Synthesize 执行综合，Output 窗口会输出综合执行过程中的信息。GowinSynthesis 综合后生成综合报告及门级网表文件，双击 Process 窗口中的 Synthesis Report 和 Netlist File 即可查看具体内容。

具体操作流程请参考文档 *SUG100，Gowin 云源软件用户指南* > 4.4.3 Synthesize 章节。

3.3 综合前后的对象命名规则

为便于用户的验证与调试，GowinSynthesis 在整个综合过程中将最大程度的保留用户原始 RTL 设计信息，如用户设计中的 module 模块信息、primitive/module instance 例化名称信息、用户定义的 wire/reg 连线名称等。对必须要经过优化或转化重新生成的对象，这些对象的名称也将由用户定义的连线名称通过一些衍生规则来生成，具体规则如下。

3.3.1 综合后网表文件命名

综合后网表文件名称取决于工程文件（*.prj）中指定的输出网表的文件名称。

若工程文件中未指定综合后网表名称，默认生成名称同工程文件名后缀

为.vg 的综合后网表文件。

3.3.2 综合后网表 module 命名

综合后网表的 module 名称与 RTL 设计命名一致，多次例化的模块会以_idx 后缀区分，module 的例化名与 RTL 设计一致。

3.3.3 综合后网表 Instance 命名

用户 RTL 设计中存在 Instance，若该 Instance 在综合过程中不被优化，则在综合后网表中保持 Instance 名称不变。

综合过程中生成的 Instance，Instance 名称来源于该 Instance 在用户 RTL 设计中所表示的功能设计块的外部输出信号名，若该功能设计块有多个输出信号，则该 Instance 名取决于第一个输出信号的名称。

对综合过程中生成的 Instance，Instance 名组成除上述所述的信号名外其后还会根据类型添加后缀，buf 类型后缀为_ibuf、_obuf 及_iobuf，其他为_s 的后缀，s 后的数字为名称被节点引用的次数。

在指定 flatten 输出时，原有的子模块 Instance 名称需要层级表示时，斜线“/”将作为层级分隔符。

3.3.4 综合后网表连线命名

用户在 RTL 设计中定义的 wire/reg 信号，若该信号在综合过程中未被优化，则综合后网表的相关模块仍将保留该信号名。

GowinSynthesis 的综合过程中，会针对某些 RTL 设计中的一整块功能设计模块进行置换或优化，综合后，网表中这些功能设计模块的输出信号名会被保留，对模块内部的信号，将根据输出信号名称做衍化，具体为在原始信号名称基础上增加相关数字后缀（_idx）来形成衍化后的信号名。

当多位宽的信号（bus 格式）名作为衍生信号名，衍生出其他信号名或 Instance 名称时，该信号名称中的 bus 位信息将以下划线加位信息（_idx）的形式被保留。

在指定 flatten 输出时，下划线“_”将作为层级分隔符。

4 硬件描述语言代码支持

4.1 寄存器的硬件描述语言代码支持

4.1.1 寄存器特征介绍

寄存器包含触发器和锁存器。

触发器

触发器全部为 D 触发器，定义时赋初值。其复位/置位方式有两种：同步复位/置位和异步复位/置位。同步复位/置位指只有当时钟信号 CLK 的上升沿或下降沿到来，且 reset/set 为高电平时，复位/置位才能完成；异步复位/置位指 reset 和 set 信号值的变化会激发过程进入到执行状态，只要 reset/set 由低电平变为高电平，复位/置位即可完成，不受时钟信号 CLK 的控制。

锁存器

锁存器触发方式含高电平触发及低电平触发，定义时赋初值，FPGA 设计最好规避锁存器。高电平触发是当控制信号为高电平时，锁存器允许数据端信号通过；低电平触发是当控制信号为低电平时，锁存器允许数据端信号通过。

4.1.2 寄存器相关的约束

用户可使用 preserve 属性对寄存器进行约束。当有此约束时，除了输出悬空的寄存器会被优化外，其他寄存器都会原样保留到综合结果中，详细请参考 syn_preserve 章节。

4.1.3 寄存器代码示例

高云芯片设计同步复位时钟触发器的初始值仅可置为 0，同步置位时钟触发器的初始值仅可置为 1，故当用户在 RTL 中设置的同步时钟触发器的初始值与同步时钟触发器的初始值不同时，GowinSynthesis 将优先依据 RTL 中的初始值，转换同步时钟触发器的类型。异步时钟触发器不做处理。具体转换策略是：

1. RTL 设计为同步复位时钟触发器，但指定初始值为 1 时，GowinSynthesis 将其替换为同步置位的时钟触发器，将原同步复位信

号添加相关逻辑以实现同步置位功能。

2. RTL 设计为同步置位时钟触发器，但指定初始值为 0 时，GowinSynthesis 将其替换为普通触发器，并在原同步置位信号上添加相关逻辑作为触发器数据端输入。

不指定触发器初始值

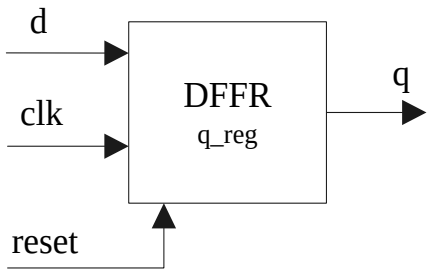
CLK 上升沿触发的触发器和 CLK 下降沿触发的触发器的区别仅仅是 CLK 触发方式不同，所以下面只列出能综合出 CLK 上升沿触发的触发器的示例。

示例 1 可被综合为同步复位的时钟触发器

```
module top (q, d, clk, reset);
input d;
input clk;
input reset;
output q;
reg q_reg;
always @(posedge clk)begin
    if(reset)
        q_reg<=1'b0;
    else
        q_reg<=d;
end
assign q = q_reg;
endmodule
```

同步复位时钟触发器示意如图 4-1 所示。

图 4-1 示例 1 同步复位时钟触发器示意图



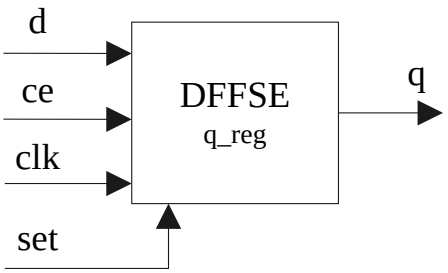
示例 2 可被综合为同步置位且带有时钟使能功能的触发器

```
module top (q, d, clk, ce, set);
```

```
input d;
input clk;
input ce;
input set;
output q;
reg q_reg;
always @(posedge clk)begin
    if(set)
        q_reg<=1'b1;
    else if(ce)
        q_reg<=d;
end
assign q = q_reg;
endmodule
```

同步置位且带有时钟使能功能的触发器示意如图 4-2 所示。

图 4-2 示例 2 同步置位且带有使能功能的触发器示意图



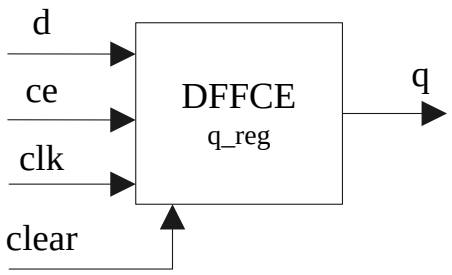
示例 3 可被综合为异步复位且带有时钟使能功能的触发器

```
module top (q, d, clk, ce, clear);
input d;
input clk;
input ce;
input clear;
output q;
reg q_reg;
always @(posedge clk or posedge clear)begin
    if(clear)
        q_reg<=1'b0;
```

```
    else if(ce)
        q_reg<=d;
end
assign q = q_reg;
endmodule
```

异步复位且带有时钟使能功能的触发器示意如图 4-3 所示。

图 4-3 示例 3 异步复位且带有时钟使能功能的触发器示意图

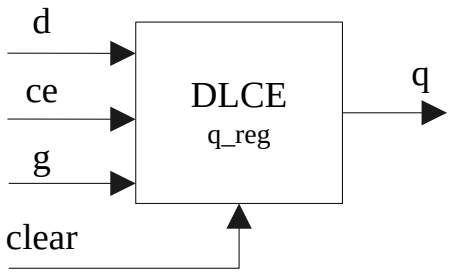


示例 4 可被综合为带有复位和高电平使能功能的锁存器

```
module top(d,g,clear,q,ce);
input d,g,clear,ce;
output q;
reg q_reg;
always @(g or d or clear or ce) begin
    if(clear)
        q_reg <= 0;
    else if(g && ce)
        q_reg <= d;
end
assign q = q_reg;
endmodule
```

带有复位和高电平使能功能的锁存器示意如图 4-4 所示。

图 4-4 示例 4 带有复位和高电平使能功能的锁存器示意图



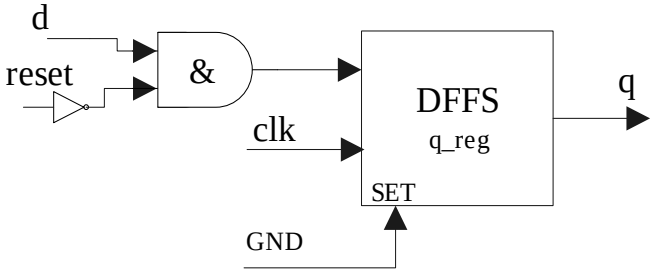
指定触发器初始值

示例 5 为同步复位的时钟触发器，其初始值应为 0，但 RTL 中设置初始值为 1，将被综合为初始值为 1 的同步置位的时钟触发器及用于实现同步复位功能的逻辑电路。

```
module top (q, d, clk, reset);
input d;
input clk;
input reset;
output q;
reg q_reg = 1'b1;
always @(posedge clk)begin
    if(reset)
        q_reg<=1'b0;
    else
        q_reg<=d;
end
assign q = q_reg;
endmodule
```

如上的同步复位时钟触发器示意如图 4-5 所示。

图 4-5 示例 5 同步复位时钟触发器及逻辑电路示意图

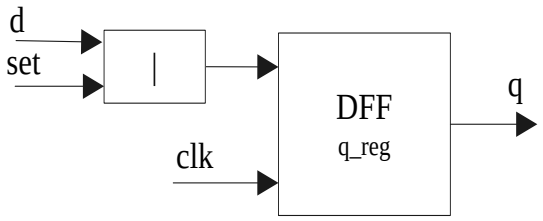


示例 6 为同步置位的时钟触发器，其初始值应为 1，但 RTL 中设置初始值为 0，将被综合为初始值为 0 的普通时钟触发器及一个用于实现同步置位功能的逻辑电路。

```
module top (q, d, clk, set);
input d;
input clk;
input set;
output q;
reg q_reg = 1'b0;
always @(posedge clk)begin
    if(set)
        q_reg<=1'b1;
    else
        q_reg<=d;
end
assign q = q_reg;
endmodule
```

如上初始值为 0 的基本时钟触发器及逻辑电路示意如图 4-6 所示。

图 4-6 示例 6 初始值为 0 的基本时钟触发器及逻辑电路示意图



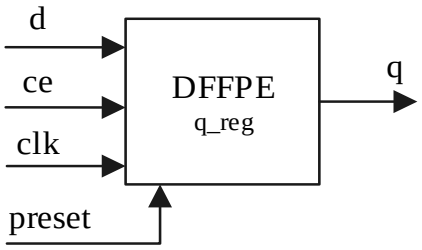
示例 7 为设置初值为 1 的异步置位触发器。

```
module top (q, d, clk, ce, preset);
input d;
input clk;
input ce;
input preset;
output q;
reg q_reg = 1'b1;
always @(posedge clk or posedge preset)begin
    if(preset)
```

```
        q_reg<=1'b1;
    else if(ce)
        q_reg<=d;
end
assign q = q_reg;
endmodule
```

如上的异步置位触发器示意如图 4-7 所示。

图 4-7 示例 7 异步置位触发器示意图



4.2 RAM 的硬件描述语言代码支持

4.2.1 RAM 置换基本功能介绍

RAM 置换是 RTL 综合过程中将用户设计中的存储功能部分推导置换为块状静态随机存储器（BSRAM）或分布式随机存储器（SSRAM）的过程。用户在设计 RTL 时既可以直接实例化 BSRAM 或 SSRAM 原语，也可以写不依赖器件的 RTL 格式的存储器描述。对 RTL 格式的存储器块，GowinSynthesis 将依据 RTL 描述，将符合相应条件的 RTL 描述置换为相应的 RAM 模块。

当逻辑模块需要使用 BSRAM 来实现时，需要满足以下原则：

1. 所有的输出寄存器有相同的控制信号；
2. RAM 必须为同步存储器，不可以有异步的控制信号相连，GowinSynthesis 不支持异步 RAM；
3. 需要在读地址或者输出端连接寄存器。

4.2.2 RAM 特征介绍

BSRAM

BSRAM 的配置模式分为单端口模式，双端口模式，伪双端口模式，只读模式；读模式分为寄存器输出模式（pipeline）及旁路模式(bypass)两种；写模式支持普通模式（normal Mode）、通写模式(write-through Mode)及先读后写模式(read-before-write Mode)三种。

SSRAM

配置模式分为单端口模式、伪双端口模式及只读模式三种，SSRAM 不支持双端口模式。

4.2.3 RAM 推导相关的约束

syn_ramstyle 指定存储器的实现方式，syn_romstyle 指定只读存储器的实现方式。

如果设计中指定要生成 SSRAM 或者 BSRAM，请使用约束语句 ram_style、rom_style 或 syn_srlstyle 来控制。

约束语句具体使用方式请参考 syn_ramstyle 和 syn_srlstyle。

4.2.4 RAM 推导代码示例

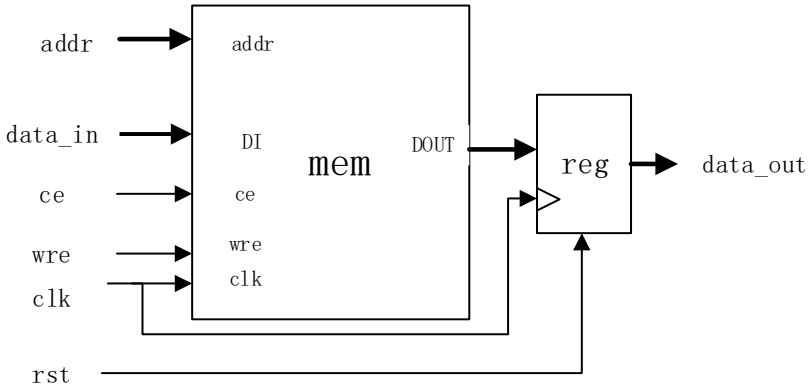
按照 RAM 的不同特征举例如下：

示例 1 为 1 个写端口，1 个读端口并且读写端口地址相同的存储器，可以被综合为普通模式的单端口 BSRAM。

```
module normal(data_out, data_in, addr, clk,ce, wre,rst);
output [7:0]data_out;
input [7:0]data_in;
input [7:0]addr;
input clk,wre,ce,rst;
reg [7:0] mem [255:0];
reg [7:0] data_out;
always@(posedge clk or posedge rst)
if(rst)
    data_out <= 0;
else
    if(ce & !wre)
        data_out <= mem[addr];
always @(posedge clk)
  if (ce & wre)
        mem[addr] <= data_in;
endmodule
```

如上的单端口 BSRAM 电路描述示意图如图 4-8 所示。

图 4-8 示例 1 RAM 电路图

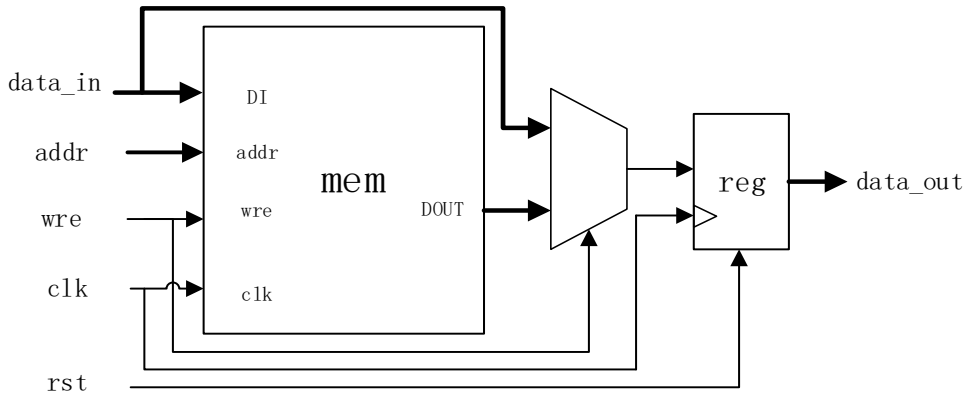


示例 2 为 1 个写端口，1 个读端口并且读写端口地址相同的存储器，当 wre 为 1 时，输入数据可以直接传输给输出，此案例被综合为普通写模式的单端口 BSRAM。

```
module wt11(data_out, data_in, addr, clk, wre,rst);
output [31:0]data_out;
input [31:0]data_in;
input [6:0]addr;
input clk,wre,rst;
reg [31:0] mem [127:0];
reg [31:0] data_out;
always@(posedge clk or posedge rst)
if(rst)
    data_out <= 0;
else if(wre)
     data_out <= data_in;
else
     data_out <= mem[addr];
always @(posedge clk)
if (wre)
    mem[addr] <= data_in;
endmodule
```

如上的单端口 BSRAM 电路描述示意图如图 4-9 所示。

图 4-9 示例 2 RAM 电路图

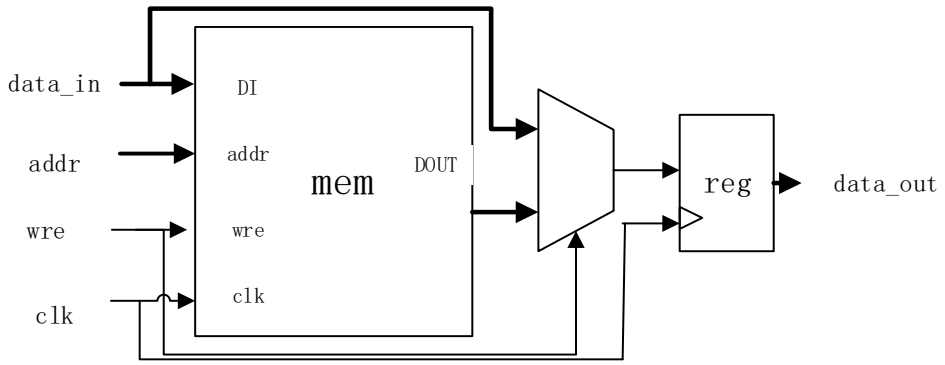


示例 3 为 1 个写端口，1 个读端口并且读写端口地址相同的存储器，当 wre 为 1 时，输入数据写入存储器中，此案例被综合为先读后写模式的单端口 BSRAM。

```
module read_first_01(data_out, data_in, addr, clk, wre);
output [31:0]data_out;
input [31:0]data_in;
input [6:0]addr;
input clk,wre;
reg [31:0] mem [127:0];
reg [31:0] data_out;
always @(posedge clk)
begin
    if (wre)
        mem[addr] <= data_in;
    data_out <= mem[addr];
 end
endmodule
```

如上的单端口 BSRAM 电路描述示意图如图 4-10 所示。

图 4-10 示例 3 RAM 电路图



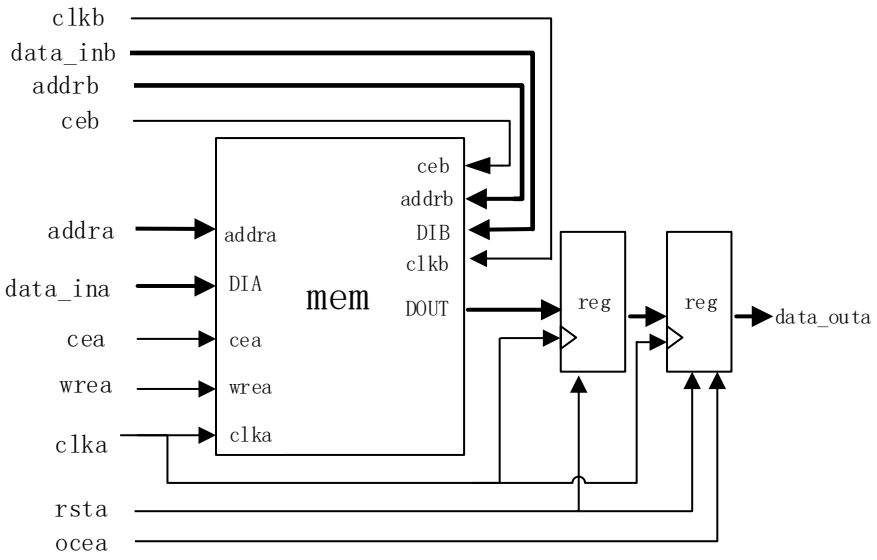
示例 4 为 2 个写端口，1 个读端口的存储器，一个写端口没有 wre 信号，另外一个写端口有 wre 信号，1 个读端口吸收异步复位的寄存器。此案例被综合为写模式 A 端为普通模式，B 端为先读后写模式，读模式为寄存器输出模式的异步复位双端口 BSRAM。

```
module read_first_02_1(data_outa, data_ina, addra, clka, rsta,cea,
wrea,ocea, data_inb, addrb, clkb, ceb );
output [17:0]data_outa;
input [17:0]data_ina,data_inb;
input [6:0]addra,addrb;
input clka, rsta,cea, wrea,ocea;
input clkb, ceb;
reg [17:0] mem [127:0];
reg [17:0] data_outa;
reg [17:0] data_out_rega,data_out_regb;
always @(posedge clkb)
if (ceb)
    mem[addrb] <= data_inb;
always@(posedge clka or posedge rsta)
if(rsta)
     data_out_rega <= 0;
else begin
     data_out_rega <= mem[addra];
end
always@(posedge clka or posedge rsta)
if(rsta)
```

```
        data_outa <= 0;
    else if (ocea)
        data_outa <= data_out_rega;
    always @(posedge clka)
    if (cea & wrea)
       mem[addra] <= data_ina;
    endmodule
```

如上的双端口 BSRAM 电路描述示意图如图 4-11 所示。

图 4-11 示例 4 RAM 电路图



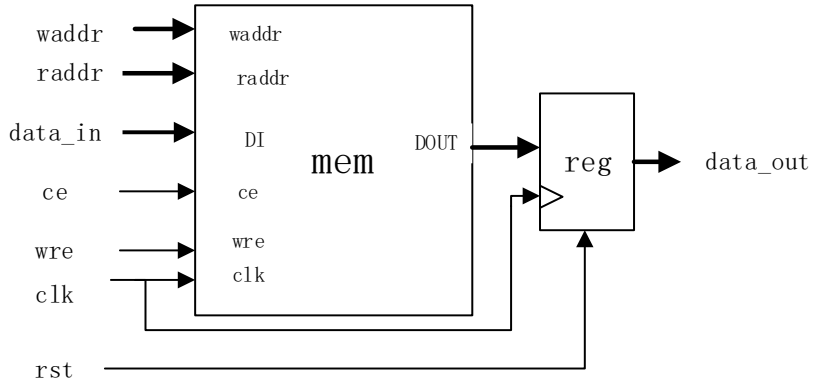
示例 5 为 1 个读端口，1 个写端口且读写地址不同的存储器，被综合为写模式为普通模式，读模式为旁路模式的伪双端口 BSRAM。

```
module read_first_wp_pre_1(data_out, data_in, waddr, raddr,clk, rst,ce);
output [10:0]data_out;
input [10:0]data_in;
input [6:0]raddr,waddr;
input clk, rst,ce;
reg [10:0] mem [127:0];
reg [10:0] data_out;
always@(posedge clk or posedge rst)
if(rst)
    data_out <= 0;
else if(ce)
    data_out <= mem[raddr];
```

```
always @(posedge clk)
if (ce) mem[waddr] <= data_in;
endmodule
```

如上的伪双端口 BSRAM 电路描述示意图如图 4-12 所示。

图 4-12 示例 5 RAM 电路图



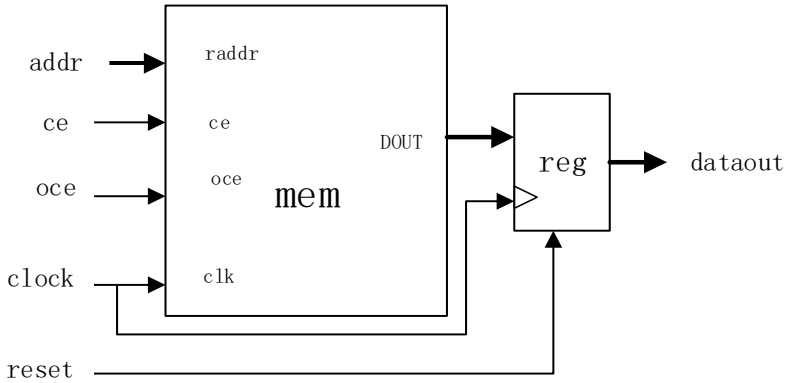
示例 6 为具有初值的只有 1 个读端口的存储器，被综合为读模式为旁路模式的异步置位只读存储器。

```
module test_invce (clock,ce,oce,reset,addr,dataout) ;
input clock,ce,oce,reset;
input [5:0] addr;
output [7:0] dataout;
reg [7:0] dataout;
always @(posedge clock or posedge reset)
if(reset) begin
    dataout <= 0;
end else begin
if (ce & oce) begin
case (addr)
6'b000000: dataout <= 32'h87654321;
6'b000001: dataout <= 32'h18765432;
6'b000010: dataout <= 32'h21876543;
…….
6'b111110: dataout <= 32'hdef89aba;
6'b111111: dataout <= 32'hef89abce;
default: dataout <= 32'hf89abcde;
endcase
```

```
end
end
endmodule
```

如上的只读存储器电路描述示意图如图 4-13 所示。

图 4-13 示例 6 pROM 电路图

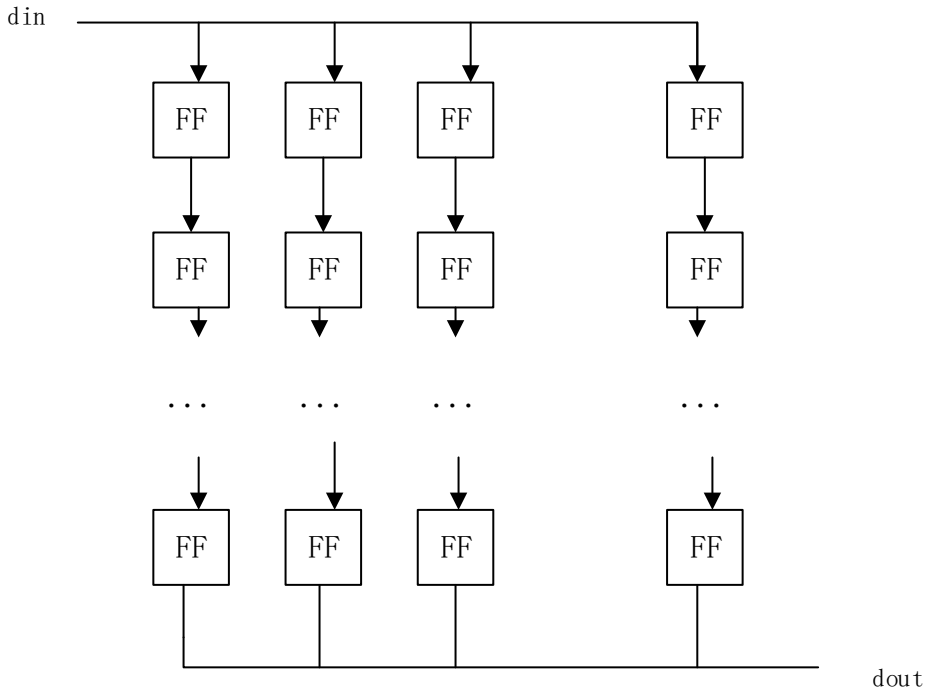


示例 7 为 shift register 模式的存储器，被综合为普通模式的伪双端口 BSRAM。

```
module seqshift_bsram (clk, din, dout) ;
parameter SRL_WIDTH = 65;
parameter SRL_DEPTH = 16;
input clk;
input [SRL_WIDTH-1:0] din;
output [SRL_WIDTH-1:0] dout;
reg [SRL_WIDTH-1:0] regBank[SRL_DEPTH-1:0] ;
integer i;
always @(posedge clk) begin
    for (i=SRL_DEPTH-1; i>0; i=i-1) begin
        regBank[i] <= regBank[i-1];
    end
    regBank[0] <= din;
end
assign dout = regBank[SRL_DEPTH-1];
endmodule
```

如上的伪双端口 BSRAM 电路描述示意图如图 4-14 所示。

图 4-14 示例 7 RAM 电路图



注！

更多示例，请参见高云官网 GowinSynthesis 推导置换编码模板文档“GowinSynthesis_Inference_Coding_Template”。

4.3 DSP 的硬件描述语言代码支持

4.3.1 DSP 推导的基本功能介绍

DSP 推导是综合过程中将用户设计中的乘法及部分加法推导置换为 DSP 的算法。用户在设计 RTL 时既可以实例化 DSP 也可以写 RTL 格式的 DSP 描述，GowinSynthesis 将依据 RTL 描述，将符合相应条件的 RTL 描述推导置换为相应的 DSP 模块。

DSP 模块具有乘法以及加法和寄存器的功能。当用户当前使用的器件不支持 DSP 模块时，GowinSynthesis 会使用逻辑电路来实现乘法器的功能。

4.3.2 DSP 特征介绍

高云 DSP 分为乘法器、乘加器、预加器及累加器。具有以下功能：

1. 支持输入符号位不同的乘法置换；
2. 支持同步或异步模式；
3. 支持乘法的级联；
4. 支持乘法的累加；
5. 支持预加功能；
6. 支持寄存器的吸收，包括输入寄存器，输出寄存器，旁路寄存器的吸

收。

4.3.3 DSP 相关的约束

syn_dspstyle 用来控制具体对象或者全局范围的乘法器使用 DSP 还是逻辑电路实现。

syn_perserve 用来保留寄存器，当 DSP 周围的寄存器有此属性时，DSP 不可以吸收此寄存器。

约束语句的具体使用方式请参考 syn_dspstyle、syn_preserve 章节。

4.3.4 DSP 推导代码示例

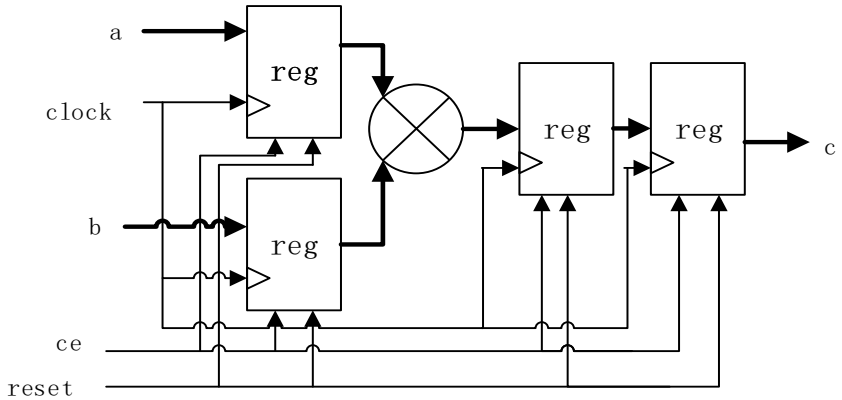
示例 1 可综合出带符号位的同步复位的乘法器，该乘法器的输入寄存器为 ina 和 inb，输出寄存器为 out_reg，旁路寄存器为 pp_reg。

```
module top(a,b,c,clock,reset,ce);
parameter a_width = 18;
parameter b_width = 18;
parameter c_width = 36;
input signed [a_width-1:0] a;
input signed [b_width-1:0] b;
input clock;
input reset;
input ce;
output signed [c_width-1:0] c;
reg signed [a_width-1:0] ina;
reg signed [b_width-1:0] inb;
reg signed [c_width-1:0] pp_reg;
reg signed [c_width-1:0] out_reg;
wire signed [c_width-1:0] mult_out;
always @(posedge clock) begin
    if(reset)begin
        ina<=0;
        inb<=0;
    end else begin
        if(ce)begin
            ina<=a;
            inb<=b;
        end
```

```
    end
end
assign mult_out=ina*inb;
always @(posedge clock) begin
    if(reset)begin
        pp_reg<=0;
    end else begin
        if(ce)begin
            pp_reg<=mult_out;
        end
    end
end
always @(posedge clock) begin
    if(reset)begin
        out_reg<=0;
    end else begin
        if(ce)begin
            out_reg<=pp_reg;
        end
    end
end
assign c=out_reg;
endmodule
```

如上的乘法器电路描述示意图如图 4-15 所示。

图 4-15 示例 1 DSP 电路图



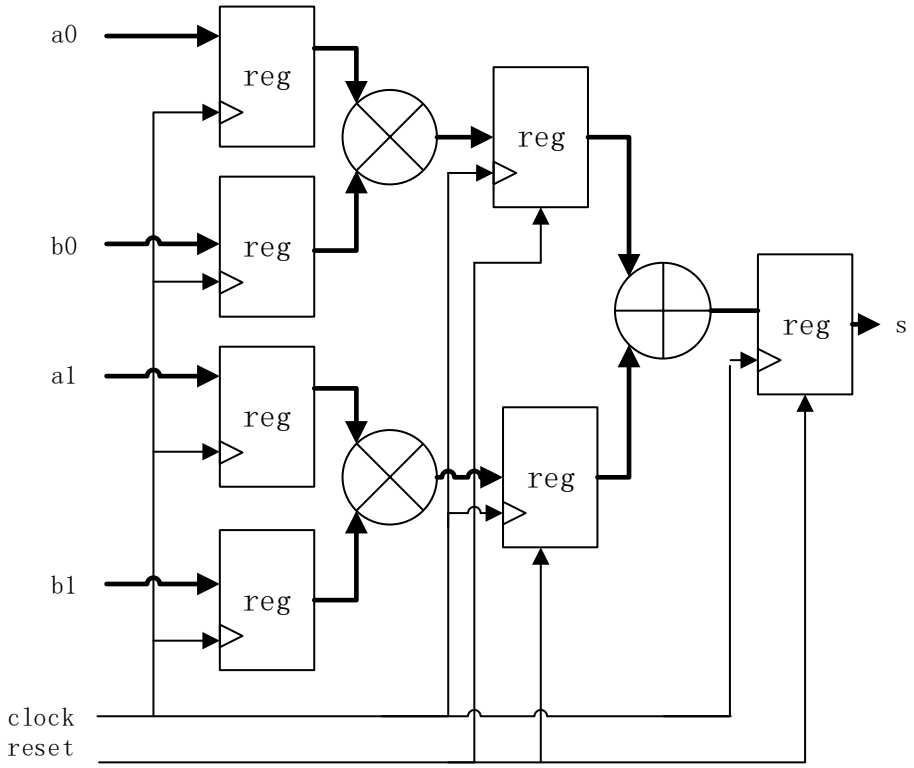
示例 2 可综合出异步模式的乘加器，该乘加器的输入端寄存器为 a0_reg、a1_reg、b0_reg 和 b1_reg，输出端寄存器为 s_reg，旁路寄存器为 p0_reg 和 p1_reg。

```
module top(a0, a1, b0, b1, s, reset, clock, ce);
parameter a0_width=18;
parameter a1_width=18;
parameter b0_width=18;
parameter b1_width=18;
parameter s_width=37;
input unsigned [a0_width-1:0] a0;
input unsigned [a1_width-1:0] a1;
input unsigned [b0_width-1:0] b0;
input unsigned [b1_width-1:0] b1;
input reset, clock, ce;
output unsigned [s_width-1:0] s;
wire unsigned [s_width-1:0] p0, p1, p;
reg unsigned [a0_width-1:0] a0_reg;
reg unsigned [a1_width-1:0] a1_reg;
reg unsigned [b0_width-1:0] b0_reg;
reg unsigned [b1_width-1:0] b1_reg;
reg unsigned [s_width-1:0] p0_reg, p1_reg, s_reg;
always @(posedge clock or posedge reset)
begin
    if(reset)begin
        a0_reg <= 0;
        a1_reg <= 0;
        b0_reg <= 0;
        b1_reg <= 0;
    end else begin
        if(ce)begin
            a0_reg <= a0;
            a1_reg <= a1;
            b0_reg <= b0;
            b1_reg <= b1;
```

```
        end
    end
end
assign p0 = a0_reg*b0_reg;
assign p1 = a1_reg*b1_reg;
always @(posedge clock or posedge reset)
begin
    if(reset)begin
        p0_reg <= 0;
        p1_reg <= 0;
    end else begin
        if(ce)begin
            p0_reg <= p0;
            p1_reg <= p1;
        end
    end
end
assign p = p0_reg - p1_reg;
always @(posedge clock or posedge reset)
begin
    if(reset)begin
        s_reg <= 0;
    end else begin
        if(ce) begin
            s_reg <= p;
        end
    end
end
assign s = s_reg;
endmodule
```

如上的乘加器电路描述示意图如图 4-16 所示。

图 4-16 示例 2 DSP 电路图

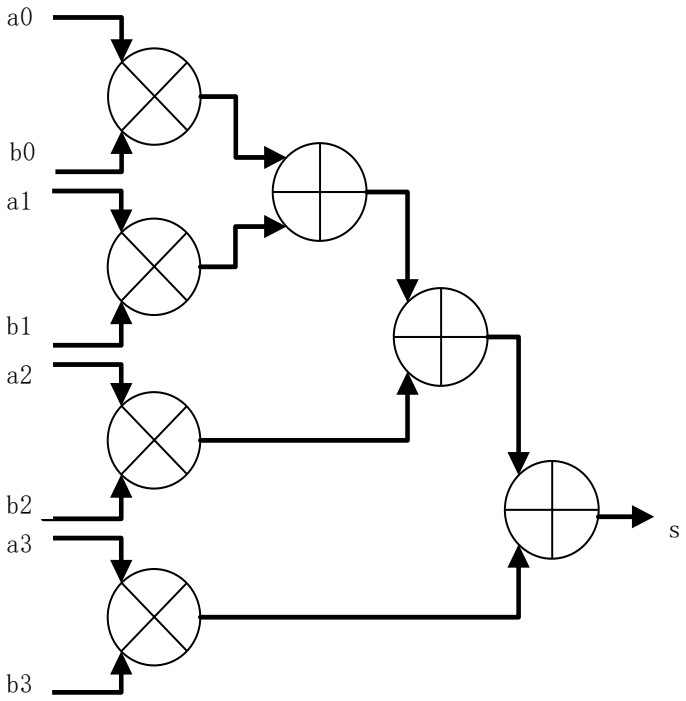


示例 3 可以综合出两个无符号位的乘加器，这两个乘加器为级联关系。

```
module top(a0, a1, a2, b0, b1, b2, a3, b3, s);
parameter a_width=18;
parameter b_width=18;
parameter s_width=36;
input unsigned [a_width-1:0] a0, a1, a2, b0, b1, b2, a3, b3;
output unsigned [s_width-1:0] s;
assign s=a0*b0+a1*b1+a2*b2+a3*b3;
endmodule
```

如上的乘加器电路描述示意图如图 4-17 所示。

图 4-17 示例 3 DSP 电路图

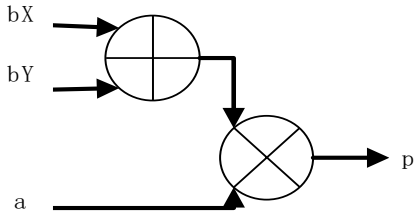


示例 4 可以综合出一个符号位为 0 的乘法器和符号位为 0 的预加器，该乘法器的一个输入端与预加器的输出端 b 相互连接。

```
module top(a, bX, bY, p);
parameter a_width=36;
parameter b_width=18;
parameter p_width=54;
input [a_width-1:0] a;
input [b_width-1:0] bX, bY;
output [p_width-1:0] p;
wire [b_width-1:0] b;
assign b = bX + bY;
assign p = a*b;
endmodule
```

如上的乘加器电路描述示意图如图 4-18 所示。

图 4-18 示例 4 DSP 电路图

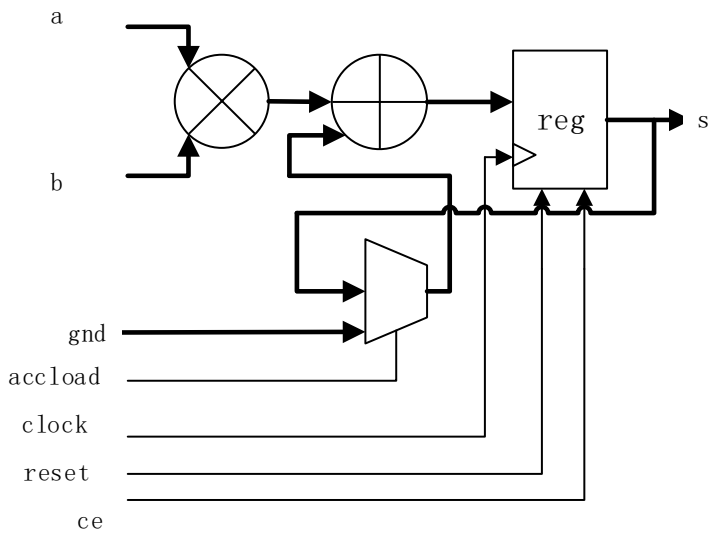


示例 5 可综合出一个符号位为 0 的乘法累加器，该乘法累加器的输出寄存器为 s。

```
module acc(a, b, s, accload, reset, ce, clock);
parameter a_width=36; //18 36
parameter b_width=18; //18 36
parameter s_width=54; //54
input unsigned [a_width-1:0] a;
input unsigned [b_width-1:0] b;
input accload, reset, ce, clock;
output unsigned [s_width-1:0] s;
wire unsigned [s_width-1:0] s_sel;
wire unsigned [s_width-1:0] p;
reg [s_width-1:0] s;
assign p = a*b ;
assign s_sel = (accload == 1'b1) ? s : 54'h0000000;
always @(posedge clock)
begin
    if(reset)begin
        s <= 0;
    end else begin
        if(ce)begin
            s <= s_sel + p;
        end
    end
end
endmodule
```

如上的乘法累加器电路描述示意图如图 4-19 所示。

图 4-19 示例 5 DSP 电路图



注！

更多示例，请参见 GowinSynthesis 推导置换编码模板文档“GowinSynthesis_Inference_Coding_Template”。

4.4 有限状态机的综合实现规则

4.4.1 有限状态机的综合规则

GowinSynthesis 支持有限状态机（Finate State Machine,FSM）的综合，编码方式支持独热码、格雷码、二进制码等。有限状态机的综合结果与状态机的编码方式、编码个数、编码位宽、编码约束等信息有关。在不设定编码约束的情况下，GowinSynthesis 自动选用独热码、格雷码或者二进制实现状态机；若有编码约束的情况下优先根据约束指定的编码方式进行实现，有关状态机的编码约束请参考 syn_encoding 章节。

注！

有限状态机的输出若直接驱动输出端口，GowinSynthesis 不将其作为状态机进行综合，此时状态机上的编码约束将被忽略。

4.4.2 有限状态机代码示例

有限状态机的综合规则介绍如下。

独热码状态机

若 RTL 设计中状态机采用独热码进行编码，在不设定编码约束的情况下，GowinSynthesis 默认选择独热码实现状态机的功能，在有编码约束的情况下则依据约束指定的编码方式实现状态机功能。独热码的编码方式举例如下。

```
reg [3:0] state,next_state;
parameter state0=4'b0001;
parameter state1=4'b0010;
parameter state2=4'b0100;
```

parameter state3=4'b1000;

上述案例中 RTL 采用独热码编码，GowinSynthesis 采用独热码进行实现。

格雷码状态机

若 RTL 设计中状态机采用格雷码进行编码，在不设定编码约束的情况下，GowinSynthesis 默认选择格雷码实现状态机的功能，在有编码约束的情况下则依据约束指定的编码方式实现状态机功能。格雷码的编码方式举例如下。

reg [3:0] state,next_state;

parameter state0=2'b00;

parameter state1=2'b01;

parameter state2=2'b11;

parameter state3=2'b10;

上述案例中 RTL 采用格雷码编码，GowinSynthesis 采用格雷码进行实现。

二进制码或其它编码状态机

若 RTL 设计中状态机采用二进制码进行编码，既不是独热码又不是格雷码，在不设定编码约束的情况下，GowinSynthesis 将根据编码的个数和位宽选择相应的编码进行实现。选取原则如下所示，若编码的个数大于编码的有效位宽则选用二进制码进行实现，若编码的个数小于等于编码的有效位宽则选用独热码进行实现；在有编码约束的情况下则依据约束指定的编码方式实现状态机功能。

示例 1

reg [5:0] state,next_state;

parameter state0= 6'b000001;

parameter state1= 6'b000011;

parameter state2= 6'b000000;

parameter state3= 6'b010101;

上述案例中编码个数为 4，编码的位宽为 6，其中有效位宽为 5，可见编码的个数小于编码的有效位宽，采用独热码进行实现。

示例 2

reg [2:0] state,next_state;

parameter state0=3'b001;

parameter state1=3'b010;

parameter state2=3'b011;

parameter state3=3'b100;

上述案例中编码个数为 4，编码的有效位宽为 3，可见编码的个数大于

编码的有效位宽，采用二进制码进行实现。

示例 3

```
reg [5:0] state,next_state;
parameter state0= 1;
parameter state1= 3;
parameter state2= 6;
parameter state3= 15;
```

上述案例中编码的个数为 4，编码采用 10 进制，有效位宽换算为二进制后是 4 位，可见编码的个数等于编码的有效位宽，采用独热码进行实现。

5 综合约束支持

属性约束用于设置综合项目中优化选择、功能实现方式、输出网表格式等的各种属性，使综合结果更好地满足用户的设计功能和使用场景。属性设置可以写在约束文件中，也可以嵌入在源码中。

本章描述了 RTL 文件中的约束和 GowinSynthesis 约束文件 GSC（Gowin Synthesis Constraint）约束的语法，作用范围和优先级。Verilog 文件是大小写敏感的，因此指令及属性必须严格按照语法描述的进行输入。约束语句中一条属性约束须写在一行中，中间不可以有换行符等进行分隔。

RTL 文件中属性约束的语法

RTL 文件中的约束必须在约束对象（object）的定义语句中添加，添加在定义语句结束的分号之前。具体语句语法如下：

object /*synthesis attributeName=value*/;

若约束属性值（value）如果为字符串，则 value 值前后需添加双引号。

GSC 中属性约束的语法

GSC 约束分为 Instance 类型的约束、Net 类型的约束、Port 类型的约束及全局对象约束。在 GSC 中,若属性值为字符串，属性值不需要使用双引号。GSC 约束中支持注释，注释使用“//”。具体语句语法如下：

INS "object" attributeName=value;

NET "object" attributeName=value;

PORT "object" attributeName=value;

GLOBAL attributeName=value;

约束语句开头为 INS，object 必须为 instance 名称，instance 包括 module/entity instance 及 primitive instance。

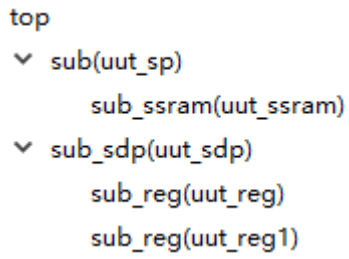
约束语句开头为 NET，约束对象必须为 net 名称。

约束语句开头为 PORT，约束对象必须为 port 名称。

约束语句开头为 GLOBAL，说明该属性约束是全局属性约束，约束对象为全局。

在 GSC 中使用“/”来区分名字之间的层级关系，举例说明，设计层级如图 5-1 所示：若对 sub_ssram 模块下的 mem 信号添加属性约束可以采用如下形式：INS "uut_sp/uut_ssram/mem" syn_ramstyle = block_ram

图 5-1 设计层级关系



属性约束范围

- 若对 top module 定义处设置属性约束，则该属性值的约束范围为整个工程；
- 若对 sub module 定义处设置属性约束，则该属性值的约束范围为当前 sub 层以及 sub 的子模块；
- 若对 sub module 的例化设置属性约束，则该属性值的约束范围为该 sub 层的例化以及 sub 的子模块；
- 若对 reg/wire 设置属性约束，则该属性值的约束范围为该 reg/wire；

每个属性约束只能作用于单个约束对象，如需约束多个对象需多次添加属性约束。

例如对 RTL 设计中 reg dout1, dout2 添加属性约束，可采用以下方法：

- 方法 1：reg dout1 /*synthesis syn_preserve=1*/, dout2 /*synthesis syn_preserve=1*/;
- 方法 2：reg dout1 /*synthesis syn_preserve=1*/; reg dout2 /*synthesis syn_preserve=1*/;

属性优先级

同一属性在工程的不同位置同时使用不同属性值进行属性设置，会存在优先级的问题，属性优先级具体如下：

- 同时对同一位置（如同时约束 top module）使用不同的属性约束值时，属性约束优先级如下：

  GSC 中设置的属性约束 > RTL 中设置的属性约束 > IDE 界面配置

- 在设计不同位置使用不同的属性约束值时，属性约束优先级如下：

对 reg/wire 使用的属性约束 > GSC 中对子模块使用的属性约束 > RTL 中子模块定义时使用的属性约束 > RTL 中子模块例化时使用的属性约束 > top module 定义时使用的属性约束。

5.1 syn_black_box

描述

指定 module 或 component 为黑盒。在综合时，黑盒模块只定义它的接口，其内容是不可访问和优化的。一个模块无论是否为空，均可作为黑盒。

属性值

表 5-1 属性值

Default	Global Attribute	Object
N/A	No	module/component

语法

表 5-2 语法示例

Verilog 约束语法	object /* synthesis syn_black_box */; Verilog Example
VHDL 约束语法	attribute syn_black_box: boolean; attribute syn_black_box of object: objectType is true; VHDL Example

注!

object 只可以是 sub module/entity。

示例

- Verilog 约束示例

```
module top(clk, in1, in2, out1, out2);
input clk;
input [1:0]in1;
input [1:0]in2;
output [1:0]out1;
output [1:0]out2;
add U1(clk, in1, in2, out1);
black_box_add U2 (in1, in2, out2);
endmodule

module add (clk, in1, in2, out1);
input clk;
input [1:0]in1;
input [1:0]in2;
output [1:0]out1;
```

```
reg [1:0]out1;
always @(posedge clk)
begin
    out1 <= in1 + in2;
end
endmodule

module black_box_add(A, B, C)/* synthesis syn_black_box */;
input [1:0]A;
input [1:0]B;
output [1:0]C;
endmodule
```

- VHDL 约束示例

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity mux2_1_top is
port(
      dina : in bit;
      dinb : in bit;
      sel   : in bit;
      dout : out bit
);
end mux2_1_top;

architecture Behavioral of mux2_1_top is
component mux2_1
port(
      dina : in bit;
      dinb : in bit;
      sel   : in bit;
      dout : out bit
);
end component;
      attribute syn_black_box: boolean;
```

```
        attribute syn_black_box of mux2_1 : component is true;
begin
        u_mux2_1 : mux2_1
        port map(
                dina => dina,
                dinb => dinb,
                sel => sel,
                dout => dout
        );
end Behavioral;
```

5.2 black_box_pad_pin

描述

指定黑盒子的 IO pads 对外部环境是可见的。该属性与 syn_black_box 搭配使用，只对黑盒子的 IO pads 起作用。

属性值

表 5-3 属性值

Default	Global Attribute	Object
N/A	No	module/component

下表列出了可设置的属性值，以及属性值所对应的特定功能。

表 5-4 可设置的属性值及其对应功能

Value	功能
portList	指定黑盒的端口为 I/O pads

语法

表 5-5 语法示例

Verilog约束语法	object /* synthesis syn_black_box black_box_pad_pin = portList */; Verilog Example
VHDL约束语法	attribute syn_black_box: boolean; attribute syn_black_box of object : objectType is true; attribute black_box_pad_pin : string; attribute black_box_pad_pin of object: objectType is portList; VHDL Example

注!

用双引号括起来的 portList 是一个无空格、以逗号分隔的列表，该列表列出了黑盒上的端口名称。

示例

- Verilog 约束示例

```
module top(clk, in1, in2, out1, out2, D, E);
input clk;
input [1:0]in1;
input [1:0]in2;
output [1:0]out1;
output [1:0]out2;
output D,E;
add U1(clk, in1, in2, out1);
black_box_add U2 (in1, in2, out2, D, E);
endmodule

module add (clk, in1, in2, out1);
input clk;
input [1:0]in1;
input [1:0]in2;
output [1:0]out1;
reg [1:0]out1;
always @(posedge clk)
begin
    out1 <= in1 + in2;
end
endmodule

module black_box_add(A, B, C, D, E)/* synthesis syn_black_box
black_box_pad_pin="D,E" */;
input [1:0]A;
input [1:0]B;
output [1:0]C;
output D,E;
endmodule
```

- VHDL 约束示例

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

```
entity top is
generic (width : integer := 4);
port (in1,in2 : in std_logic_vector(width downto 0);
      clk : in std_logic;
      q : out std_logic_vector (width downto 0)
);
end top;
architecture top1_arch of top is
component test is
generic (width1 : integer := 2);
port (in1,in2 : in std_logic_vector(width downto 0);
      clk : in std_logic;
      q : out std_logic_vector (width downto 0)
);
end component;
attribute syn_black_box : boolean;
attribute syn_black_box of test : component is true;
attribute black_box_pad_pin : string;
attribute black_box_pad_pin of test : component is "q";
begin
     test123 : test generic map (width) port map (in1,in2,clk,q);
end top1_arch;
```

5.3 full_case

描述

仅适用于 Verilog 的 RTL 设计。在 case、casex 或 casez 语句后添加此属性，表明所有可能的取值都已涵盖，不需要额外的硬件来保留信号值。

语法

表 5-6 语法示例

Verilog 约束语法	object /*synthesis full_case*/ Verilog Example

示例

- Verilog 约束示例

```
module top(out, a, b, c, d, select);
```

```
output out;
input a,b,c,d;
input [3:0] select;
reg out;
always @(select or a or b or c or d)
begin
    casez(select) /*synthesis full_case*/
        4'b???1: out=a;
        4'b??1?: out=b;
        4'b?1??: out=c;
        4'b1???: out=d;
    endcase
end
endmodule
```

5.4 parallel_case

描述

仅适用于 Verilog 的 RTL 设计。在 case、casex 或 casez 语句后添加此属性，表明使用并行多路复用结构，而不是优先级编码结构。

case 语句默认按照优先级顺序工作，即仅对多个输入信号中优先级最高的一个进行编码。而 parallel_case 指令则强制采用并行多路复用结构，而非优先级编码结构。

语法

表 5-7 语法示例

Verilog 约束语法	object /*synthesis parallel_case*/ Verilog Example

示例

- Verilog 约束示例

```
module test (out, a, b, c, d, select);
output out;
input a, b, c, d;
input [3:0] select;
reg out;
always @(select or a or b or c or d)
begin
```

```
        casez (select) /* synthesis parallel_case */
            4'b???1: out = a;
            4'b??1?: out = b;
            4'b?1??: out = c;
            4'b1???: out = d;
            default: out = 1'b0;
        endcase
    end
    endmodule
```

5.5 syn_dspstyle

描述

指定乘法器以专用 DSP 硬件模块或逻辑电路的方式实现。默认情况下，乘法、乘法加/减、乘法累加、乘法级联、基于乘法的预加/减、基于乘法的移位、大加/减法等类型的结构会被推断为 DSP 硬件模块。

属性值

表 5-8 属性值

Global Attribute	Object
Yes	module/entity, wire/reg/signal

下表列出了所有可设置的属性值，以及每个属性值所对应的特定功能。

表 5-9 可设置的属性值及其对应功能

Value	功能
dsp	指定乘法器的实现方式为 DSP 硬件模块。
logic	指定乘法器的实现方式为逻辑电路。

语法

表 5-10 语法示例

GSC 约束语法	INS "object" syn_dspstyle = value GLOBAL syn_dspstyle = value GSC Example
Verilog 约束语法	object /* synthesis syn_dspstyle = "value" */; Verilog Example
VHDL 约束语法	attribute syn_dspstyle : string; attribute syn_dspstyle of object : objectType is "value"; VHDL Example

注！

若对不支持 DSP 硬件模块的器件使用属性值为 dsp 的属性约束，综合工具会给出 warning，提示属性约束无效。

示例

- GSC 约束示例

  示例 1：指定 instance 的实现方式为逻辑电路。

  ```
  INS "temp" syn_dspstyle=logic;
  INS "aa0/mult/c" syn_dspstyle=logic;
  ```

  示例 2：指定全局所有乘法器的实现方式为 DSP 硬件模块。

  ```
  GLOBAL syn_dspstyle=dsp;
  ```

- Verilog 约束示例

  示例 1：指定 module 中所有乘法器的实现方式为 logic。

  ```
  module mult(a,b,a0,b0,a1,b1,c,r,en,r0,r1)/*synthesis syn_dspstyle =
  "logic" */;
  input [7:0] a,b;
  input [7:0] a0,b0;
  input [7:0] a1,b1;
  output [15:0]r;
  output[15:0]r0,r1;
  input [15:0]c;
  input en;
  wire [15:0] temp;
  assign temp = a*b;
  assign r = en ? temp:c;
  test aa0(a0,b0,r0,a1,b1,r1);
  endmodule

  module test(a,b,c,d,e,f);
  input [7:0]a,b,d,e;
  output [15:0] c,f;
  assign c=a*b;
  test222 mult(d,e,f);
  endmodule

  module test222(a,b,c);
  input [7:0] a,b;
  ```

```
output [15:0]c;
assign c = a*b;
endmodule
```

示例 2：指定选中乘法器的实现方式为 logic。

```
module mult(a,b,a0,b0,a1,b1,c,r,en,r0,r1);
input [7:0] a,b;
input [7:0] a0,b0;
input [7:0] a1,b1;
output [15:0]r;
output[15:0]r0,r1;
input [15:0]c;
input en;
wire [15:0] temp/*synthesis syn_dspstyle = "logic" */;
assign temp = a*b;
assign r = en ? temp:c;
test aa0(a0,b0,r0,a1,b1,r1);
endmodule

module test(a,b,c,d,e,f);
input [7:0]a,b,d,e;
output [15:0] c,f;
assign c=a*b;
test222 mult(d,e,f);
endmodule

module test222(a,b,c);
input [7:0] a,b;
output [15:0]c;
assign c = a*b;
endmodule
```

- VHDL 约束示例

  示例 1：指定选中乘法器的实现方式为 logic。

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

```
entity Mult is
port(
      clk: in bit;
      ce: in bit;
      x: in signed(11 downto 0);
      y: in signed(11 downto 0);
      result: out signed(23 downto 0));
      attribute syn_dspstyle: string;
      attribute syn_dspstyle of result: signal is "logic";
end Mult;

architecture Behavior of Mult is
      signal x1 : signed(11 downto 0);
      signal y1 : signed(11 downto 0);
begin
       process(clk,ce)
       begin
            if(clk'event and clk = '1' and ce = '1')then
                 x1 <= x;
                 y1 <= y;
           end if;
       end process;
       result <= x1 * y1;
end Behavior;
```

5.6 syn_encoding

描述

指定有限状态机的编码方式。默认情况下，综合结果的编码方式与设计中的编码方式相同。

属性值

表 5-11 属性值

Global Attribute	Object
No	reg/type

下表列出了所有可设置的属性值，以及每个属性值所对应的特定功能。

表 5-12 可设置的属性值及其对应功能

Value	功能
onehot	指定有限状态机的编码方式为独热码（One-hot）。
gray	指定有限状态机的编码方式为格雷码（Gray）。

语法

表 5-13 语法示例

Verilog 约束语法	object /* synthesis syn_encoding = "value" */ ; Verilog Example
VHDL 约束语法	attribute syn_encoding : string; attribute syn_encoding of object : objectType is "value" ; VHDL Example

注！

value：状态机的编码方式，Verilog 当前支持的约束编码方式有 onehot；VHDL 当前支持的约束编码方式有 onehot、gray。

示例

- Verilog 示例

```
指定状态机按照 One-hot 码的方式来进行编码。
module test (clk,rst,data,out);
input clk,rst,data;
output out;
reg out;
reg [2:0]ps /*synthesis syn_encoding="onehot"*/;
reg [2:0]ns;
parameter S0=3'b000, S1=3'b001, S2=3'b010, S3=3'b011, S4=3'b100,
S5=3'b101, S6=3'b110;
always @(posedge clk or posedge rst) begin
if(rst)
    ps <= S0;
else
    ps <= ns; end
always @(ps or data)
begin
    ns = S0;
    case (ps)
        S0: if(data)
            ns = S1;
```

```
                else ns = S0;
                S1: if(data)
                      ns = S1;
                else ns = S2;
                S2: if(data)
                      ns = S1;
                else ns = S3;
                S3: if(data)
                      ns = S4;
                else ns = S0;
                S4: if(data)
                      ns = S5;
                else ns = S2;
                S5: if(data)
                      ns = S1;
                else ns = S6;
                S6: if(data)
                      ns = S1;
                 else ns = S3;
                 default: ns = S0;
        endcase
end
always @(ps or data) begin
if ((ps == S6) && (data == 1))
      out=1;
else
      out=0;
end
endmodule
```

- VHDL 示例

  指定状态机按照 Gray 码的方式来进行编码。

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

```
entity top is
    port(
        clk : in std_logic;
        rst : in std_logic;
        sel : in std_logic_vector(1 downto 0);
        a : in std_logic_vector(7 downto 0);
        b : in std_logic_vector(7 downto 0);
        qout: out std_logic_vector(7 downto 0)
    );
end entity;

architecture rtl of top is
    type state_value is (S5,S4,S3,S2,S1,S0);
    signal curr_state:state_value;
    signal next_state:state_value;
    attribute syn_encoding : string;
    attribute syn_encoding of state_value : type is "gray";
begin

    process(clk,rst)begin
        if(rst='1')then
            curr_state<=S0;
        else
            if(rising_edge(clk))then
                curr_state<=next_state;
            end if;
        end if;
    end process;

    process(sel,curr_state)begin
        next_state <= S0;
        case curr_state is
            when S0 =>
                if(sel ="00")then
```

```
            next_state <= S1;
        else
            next_state <= S0;
        end if;
    when S1 =>
        if(sel ="01")then
            next_state <= S2;
        else
            next_state <= S1;
        end if;
    when S2 =>
        if(sel ="10")then
            next_state <= S3;
        else
            next_state <= S2;
        end if;
    when S3 =>
        if(sel ="11")then
            next_state <= S4;
        else
            next_state <= S3;
        end if;
    when S4 =>
        if(sel ="00")then
            next_state <= S4;
        else
            next_state <= S5;
        end if;
    when S5 =>
        if(sel ="01")then
            next_state <= S5;
        else
            next_state <= S0;
        end if;
    when others=>next_state <= S0;
```

```
        end case;
    end process;

    process(clk,rst)begin
        if(rst='1')then
            qout<=(others=>'0');
        else
            if(rising_edge(clk))then
                case curr_state is
                    when S0 => qout <= a+'1';
                    when S1 => qout <= b-'1';
                    when S2 => qout <= a;
                    when S3 => qout <= b;
                    when S4 => qout <= a+b;
                    when S5 => qout <= "10101010";
                    when others=>qout <= a-b;
                end case;
            end if;
        end if;
    end process;
end rtl;
```

5.7 syn_insert_pad

描述

指定是否插入 I/O buffer，该属性只可以在 gsc 中使用。

属性值

下表列出了所有可设置的属性值，以及每个属性值所对应的特定功能。

表 5-14 可设置的属性值及其对应的功能

Value	功能
0	移除 I/O buffer
1	插入 I/O buffer

语法

表 5-15 语法示例

GSC 约束语法	PORT "object" syn_insert_pad=value; GSC Example

注！

object 只可以为 port，此约束只对 Input port 或 Output port 起作用，对 Inout port 不起作用。

示例

- GSC 约束示例

  示例 1：指定插入 I/O buffer。

  PORT "out" syn_insert_pad = 1;

  示例 2：指定移除 I/O buffer。

  PORT "out" syn_insert_pad = 0;

5.8 syn_keep

描述

指定 net 作为占位符而将其保留，不进行优化。此属性可保留在综合过程中可能被移除的 net，防止在优化过程中合并重复的单元。

属性值

表 5-16 属性值

Default	Global Attribute	Object
N/A	No	wire/signal, port, 组合逻辑

下表列出了可设置的属性值，以及所对应的特定功能。

表 5-17 可设置的属性值及其对应的功能

Value	功能
0	允许优化 net
1	保留 net 不被优化

语法

表 5-18 语法示例

Verilog 约束语法	object /* synthesis syn_keep = value */ ; Verilog Example
VHDL 约束语法	attribute syn_keep : integer; attribute syn_keep of object : objectType is value; VHDL Example

syn_keep 与 syn_preserve 的区别

表 5-19 syn_keep 与 syn_preserve 的区别

syn_keep	仅适用于 net 和组合逻辑。此属性可在综合期间保留指定的 net，防止不必要的优化。
syn_preserve	仅适用于 register。此属性可以防止 register 被优化或者被吸收。

示例

- Verilog 约束示例

  指定 wire top1, top2 不被优化掉。

```
module test (out1, out2, clk, in1, in2);
output out1, out2;
input clk;
input in1, in2;
wire and_out;
wire top1 /*synthesis syn_keep=1*/;
wire top2 /*synthesis syn_keep=1*/;
reg out1, out2;
assign and_out=in1&in2;
assign top1=and_out;
assign top2=and_out | in1;
always @(posedge clk)begin
    out1<=top1;
    out2<=top2;
end
endmodule
```

- VHDL 约束示例

  指定 signal tmp0, tmp1 不被优化掉。

```
library ieee;
use ieee.std_logic_1164.all;
entity mux2_1 is
    port(
        dina : in bit;
        dinb : in bit;
        sel  : in bit;
```

```
        dout0 : out bit;
        dout1 : out bit
    );
end mux2_1;

architecture Behavioral of mux2_1 is
    signal tmp0 : bit;
    signal tmp1 : bit;
    attribute syn_keep : integer;
    attribute syn_keep of tmp0 : signal is 1;
    attribute syn_keep of tmp1 : signal is 1;
begin
    tmp0 <= dina when sel = '0' else dinb;
    tmp1 <= dinb when sel = '0' else dina;
    dout0 <= tmp0;
    dout1 <= tmp1;
end Behavioral;
```

5.9 syn_looplimit

描述

指定设计中循环迭代次数的最大值。默认情况下，此限制值设定为2000。

语法

表 5-20 语法示例

GSC 约束语法	GLOBAL syn_looplimit=value GSC Example

注！

GSC 约束语法中不支持 vhdl 的 for loop，可对 while loop 进行约束。

示例

GSC 约束示例

```
GLOBAL syn_looplimit=3000
```

5.10 syn_maxfan

描述

指定最大扇出值,该属性可作用于模块、端口、信号(wire/reg/signal)，但该属性约束对时钟信号、控制使能信号、复位/置位信号不起作用。为了实现设定的最大扇出值，可能会复制寄存器或其他逻辑资源。

属性值

表 5-21 属性值

Default	Global Attribute	Object
N/A	Yes	module/entity, wire/reg/signal, port

语法

表 5-22 语法示例

GSC 约束语法	INS "object" syn_maxfan=value; NET "object" syn_maxfan=value; GLOBAL syn_maxfan=value; GSC Example
Verilog 约束语法	object /* synthesis syn_maxfan = value */ ; Verilog Example
VHDL 约束语法	attribute syn_maxfan : integer; attribute syn_maxfan of object : objectType is value; VHDL Example

注！

GSC 约束语法中不支持 module。

示例

- GSC 约束示例

  示例 1：指定 instance 的最大扇出值为 10。

  INS "d" syn_maxfan=10;

  示例 2：指定全局的最大扇出值为 100。

  GLOBAL syn_maxfan=100;

  示例 3：指定 instance 的最大扇出值为 10。

  INS "aa0/mult/d" syn_maxfan=10;

  示例 4：指定 net 的最大扇出值为 10。

  NET "aa0/mult/d" syn_maxfan=10;

- Verilog 示例

  示例 1：指定 module 内除 clk 外所有 instance 的最大扇出值为 3。

  ```
  module test(a, b,sel,clk,c) /*synthesis syn_maxfan=3*/;
  input [7:0] a;
  input [7:0] b;
  input sel;
  input clk;
  output reg [7:0] c;
  ```

```
always @ (posedge clk) begin
    c <= sel ? a : b;
end
endmodule
```

示例 2：指定 sel 的最大扇出值为 3。

```
module test(a, b,sel,clk,c) ;
input [7:0] a;
input [7:0] b;
input sel/*synthesis syn_maxfan=3*/;
input clk;
output reg [7:0] c;

always @ (posedge clk) begin
    c <= sel ? a : b;
end
endmodule
```

- VHDL 示例

示例 1：指定 sel 的最大扇出值为 3。

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

entity test is
    generic (
        n : integer := 10;
        m : integer := 7
    );
    port (
        a : in std_logic_vector(7 downto 0);
        b : in std_logic_vector(7 downto 0);
        sel : in std_logic;
        clk : in std_logic;
        c : out std_logic_vector(7 downto 0)
    );
    attribute syn_maxfan : integer;
```

```
    attribute syn_maxfan of sel : signal is (n-m);
end test;
architecture rtl of test is

begin
process (clk) begin
    if (clk'event and clk = '1') then
        if (sel = '1') then
            c <= a;
        else
            c <= b;
        end if;
    end if;
end process;
end rtl;
```

5.11 syn_netlist_hierarchy

描述

指定是否生成层级结构的网表。默认情况，综合工具生成层级结构网表。

属性值

表 5-23 属性值

Default	Global Attribute	Object
1	Yes	module/entity

下表列出了可设置的属性值，以及每个属性值所对应的特定功能。

表 5-24 可设置的属性值及其对应功能

Value	功能
0	将层级结构网表进行扁平化输出。
1	生成层级结构网表。

语法

表 5-25 语法示例

GSC 约束语法	GLOBAL syn_netlist_hierarchy=value; GSC Example
Verilog 约束语法	object /* synthesis syn_netlist_hierarchy = value */; Verilog Example
VHDL 约束语法	attribute syn_netlist_hierarchy: integer; attribute syn_netlist_hierarchy of object : objectType is value; VHDL Example

注！

指定作用的对象只能是 top module/entity。

示例

- GSC 约束示例

```
GLOBAL syn_netlist_hierarchy=0;
```

- Verilog 约束示例

```
module fu_add (a,b,cin,su,cy);
input a,b,cin;
output su,cy;

assign su = a ^ b ^ cin;
assign cy = (a & b) | ((a ^ b) & cin);
endmodule

module rca_adder (A,B,CIN,SU,COUT);
input [1:0] A,B;
input CIN;
output [1:0] SU;
output COUT;

wire CY;
fu_add FA0(.su(SU[0]),.cy(CY),.cin(CIN),.a(A[0]),.b(B[0]));
fu_add FA1(.su(SU[1]),.cy(COUT),.cin(CY),.a(A[1]),.b(B[1]));
endmodule

module rp_top (A1, B1,CIN,SUM,COUT)/*synthesis
syn_netlist_hierarchy=1 */;
```

```
input [7:0] A1, B1;
input CIN;
output [7:0] SUM;
output COUT;

wire [2:0] CY1;
rca_adder RA0
(.SU(SUM[1:0]),.COUT(CY1[0]),.CIN(CIN),.A(A1[1:0]),.B(B1[1:0]));
rca_adder RA1
(.SU(SUM[3:2]),.COUT(CY1[1]),.CIN(CY1[0]),.A(A1[3:2]),.B(B1[3:2]));
rca_adder RA2
(.SU(SUM[5:4]),.COUT(CY1[2]),.CIN(CY1[1]),.A(A1[5:4]),.B(B1[5:4]));
rca_adder RA3
(.SU(SUM[7:6]),.COUT(COUT),.CIN(CY1[2]),.A(A1[7:6]),.B(B1[7:6]));
endmodule
```

- VHDL 约束示例

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity mux4_1_top is
port(dina : in bit;
     dinb : in bit;
     sel :in bit;
     dout : out bit);
attribute syn_netlist_hierarchy: integer;
attribute syn_netlist_hierarchy of mux4_1_top: entity is 0;
end mux4_1_top;

architecture Behavioral of mux4_1_top is
component mux2_1
port(dina : in bit;
     dinb : in bit;
     sel : in bit;
     dout : out bit);
end component;
```

```
begin
    u_mux2_1 : mux2_1
    port map(dina => dina,
             dinb => dinb,
             sel => sel,
             dout => dout);
end Behavioral;
entity mux2_1 is
    port(dina : in bit;
         dinb : in bit;
         sel : in bit;
         dout : out bit);
end mux2_1;
architecture Behavioral of mux2_1 is
begin
    dout <= dina when sel = '0' else dinb;
end Behavioral;
```

5.12 syn_preserve

描述

指定是否优化 register。

属性值

表 5-26 属性值

Default	Global Attribute	Object
0	Yes	module/entity, module/entity instance, reg/signal

下表列出了可设置的属性值，以及其所对应的特定功能。

表 5-27 可设置的属性值及其对应功能

Value	功能
0	允许优化 register。
1	保留 register 不被优化。

语法

表 5-28 语法示例

GSC 约束语法	INS "object" syn_preserve=value; GLOBAL syn_preserve=value; GSC Example
Verilog 约束语法	object /* synthesis syn_preserve = value */; Verilog Example
VHDL 约束语法	attribute syn_preserve : integer; attribute syn_preserve of object : objectType is value; VHDL Example

示例

- GSC 约束示例

  示例 1：指定保留 reg1 不被优化。

  ```
  INS "reg1" syn_preserve = 1;
  ```

  示例 2：指定保留设计中的所有寄存器。

  ```
  GLOBAL syn_preserve = 1;
  ```

- Verilog 约束示例

  示例 1：指定保留 module 中的所有寄存器。

  ```
  module test (out1,out2,clk,in1,in2)/*synthesis syn_preserve = 1*/;
  output out1, out2;
  input clk;
  input in1, in2;
  reg out1;
  reg out2;
  reg reg1;
  reg reg2;
  always @(posedge clk)begin
      reg1 <= in1 & in2;
      reg2 <= in1& in2;
      out1 <= !reg1;
      out2 <= !reg1 & reg2;
  end
  endmodule
  ```

  示例 2：指定保留 reg1 不被优化。

  ```
  module test (out1,out2,clk,in1,in2);
  output out1, out2;
  ```

```
input clk;
input in1, in2;
reg out1;
reg out2;
reg reg1/*synthesis syn_preserve = 1*/;
reg reg2;
always @(posedge clk)begin
    reg1 <= in1 & in2;
    reg2 <= in1& in2;
    out1 <= !reg1;
    out2 <= !reg1 & reg2;
end
endmodule
```

- VHDL 约束示例

  示例 1：指定保留寄存器 reg1。

```
library ieee;
use ieee.std_logic_1164.all;
entity syn_test is
    port (out1 : out std_logic;
          out2 : out std_logic;
          in1,in2,clk : in std_logic);
end syn_test;
architecture behave of syn_test is
    signal reg1 : std_logic;
    signal reg2 : std_logic;
    attribute syn_preserve : integer;
    attribute syn_preserve of reg1 : signal is 1;
begin
process
begin
    wait until clk'event and clk = '1';
    reg1 <= in1 and in2;
    reg2 <= in1 and in2;
    out1 <= reg1;
    out2 <= reg1 and reg2;
```

```
end process;
end behave;
```

5.13 syn_probe

描述

该属性约束通过插入探测点对设计中的内部信号进行测试和调试。被指定的探测点会以 port 的形式出现在顶层端口列表中。

属性值

表 5-29 属性值

Default	Global Attribute	Object
N/A	No	wire/reg/signal

下表列出了所有可设置的属性值，以及每个属性值所对应的特定功能。

表 5-30 可设置的属性值及其对用功能

Value	功能
0	不允许探测
1	插入探测点，根据 net 的名称自动得到探测 port 的名称。
portName	插入一个指定名字的探测点

语法

表 5-31 语法示例

Verilog 约束语法	object /* synthesis syn_probe = "value" */; Verilog Example
VHDL 约束语法	attribute syn_probe: string; attribute syn_probe of object: objectType is "value"; VHDL Example

注！

不支持 value 的值与 object 名字或者 module 的 port 名相同。

示例

- Verilog 约束示例

  设置该属性约束后，probe_alu_tmp 将被列在顶层输出端口列表中。

```
module alu(out1, opcode, clk, a, b, sel);
output [7:0] out1;
input [2:0] opcode;
input [7:0] a, b;
input clk, sel;
reg [7:0] alu_tmp/*synthesis syn_probe=1*/;
```

```
reg [7:0] out1;
always @(opcode or a or b or sel)
begin
    case (opcode)
        3'b000: alu_tmp <= a+b;
        3'b001: alu_tmp <= a-b;
        3'b010: alu_tmp <= a^b;
        3'b100: alu_tmp <= sel ? a:b;
        default: alu_tmp<= a | b;
    endcase
end
always @(posedge clk)
    out1 <= alu_tmp;
endmodule
```

- VHDL 约束示例

  设置该属性约束后，probe_string 会被列在顶层输出端口列表中。

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

entity halfadd is
port(a,b : in std_logic;
    s,c : out std_logic);
end halfadd;

architecture add of halfadd is
    signal probe_tmp: std_logic;
    attribute syn_probe: string;
    attribute syn_probe of probe_tmp: signal is "probe_string";
begin
    s <= a xor b;
    probe_tmp <= a and b;
    c <= probe_tmp;
end;
```

5.14 syn_radhardlevel

描述

指定用三模冗余技术来实现寄存器的功能。三模冗余（Triple modular redundancy, TMR）是多模块冗余的一种形式，基本思想是将模块复制为 3 份，为每个模块提供相同的输入，三个模块的输出被连接到多数表决器，以少数服从多数的方式屏蔽错误，可以对大多数实际电路中的软错误进行监测和屏蔽。

属性值

表 5-32 属性值

Default	Global Attribute	Object
none	Yes	module/entity, reg/signal, module/entity instance

下表列出了所有可设置的属性值，以及每个属性值所对应的特定功能。

表 5-33 可设置的属性值及其对应功能

Value	功能
none（默认值）	使用标准设计技术，不使用三模冗余技术。
tmr	使用三模块冗余技术。每个寄存器由三个触发器实现，它们通过“投票”来确定寄存器的状态。由于插入了额外的逻辑，此选项可能会影响面积和时序，因此在使用此选项时，请务必检查面积和时序目标。

语法

表 5-34 语法示例

GSC 约束语法	GLOBAL syn_radhardlevel=setting_value INS “object” syn_radhardlevel=setting_value GSC Example
Verilog 约束语法	object/* synthesis syn_radhardlevel= setting_value */; Verilog Example
VHDL 约束语法	attribute syn_radhardlevel : string; attribute syn_radhardlevel of object: objectType is setting_value; VHDL Example

示例

- GSC 约束示例

  示例 1：指定用三模冗余技术来实现寄存器 reg1 的功能。

  INS “reg1” syn_radhardlevel=tmr

  示例 2：指定全局所有寄存器均不用三模冗余技术来实现。

  GLOBAL syn_radhardlevel=none

- Verilog 约束示例

示例 1：如下案例对 reg out_r 添加属性约束，out_r 将被复制成 3 份。

```
module top(clk,rst,din,out);
input clk;
input rst;
input din;
output out;

reg out_r/*synthesis syn_radhardlevel=tmr*/;
assign out=out_r;
always@(posedge clk or posedge rst)begin
    if(rst)begin
        out_r<=0;
    end
    else begin
        out_r<=din;
    end
end
endmodule
```

示例 2：如下案例对 module 添加属性约束，该模块的所有的 reg 都会被复制成 3 份。

```
module top(clk,rst,ina,inb,douta,doutb)/*synthesis
syn_radhardlevel="tmr"*/;
input clk;
input rst;
input ina,inb;
output douta,doutb;

reg douta,doutb;

always@(posedge clk)begin
    if(rst)begin
        douta<=0;
    end
    else begin
```

```
            douta<=ina;
        end
    end

    always@(posedge clk)begin
        doutb<=inb;
    end
    endmodule
```

示例 3：如下案例对 top module 添加属性约束，top module、sub module 中的 reg 都会进行复制。

```
    module top(clk,ce,rst,ina,inb,douta,doutb)/*synthesis
syn_radhardlevel="tmr"*/;
    input clk;
    input rst,ce;
    input[1:0] ina;
    input inb;
    output reg[1:0] douta;
    output doutb;

    always@(posedge clk)begin
        if(rst)begin
            douta<=0;
        end
        else begin
            if(ce)begin
                douta<=ina;
            end
        end
    end

    sub uut(
        .clk(clk),
        .ce(ce),
        .inb(inb),
        .doutb(doutb)
```

```
);
endmodule

module sub(clk,ce,inb,doutb);
input clk;
input ce;
input inb;
output reg doutb;
always@(posedge clk)begin
    if(ce)begin
        doutb<=inb;
    end
end
endmodule
```

- VHDL 约束示例

示例 1：如下案例对 entity 添加属性约束，该实体中所有的寄存器都会被复制成 3 份。

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

entity top is
port(
    clk : in std_logic;
    rst : in std_logic;
    din : in std_logic_vector(1 downto 0);
    dout: out std_logic_vector(1 downto 0)
);
    attribute syn_radhardlevel :string;
    attribute syn_radhardlevel of top:   entity is "tmr";
end;
architecture rtl of top is
begin
process(clk)begin
```

```
        if(rising_edge(clk))then
            if(rst='1')then
                dout<="00";
            else
                dout<=din;
            end if;
        end if;
    end process;
    end ;
```

示例 2：如下案例对 dout_reg 信号添加属性约束，该信号对应的寄存器会被复制成 3 份。

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

    entity top is
    port(
        clk : in std_logic;
        rst : in std_logic;
        din : in std_logic_vector(1 downto 0);
        dout: out std_logic_vector(1 downto 0)
    );
    end;
    architecture rtl of top is
        signal dout_reg :   std_logic_vector(1 downto 0);
        attribute syn_radhardlevel :string;
        attribute syn_radhardlevel of dout_reg :   signal   is "tmr";
    begin
    process(clk)begin
        if(rising_edge(clk))then
            if(rst='1')then
                dout_reg<="00";
            else
                dout_reg<=din;
            end if;
```

```
        end if;
    end process;
    dout<=dout_reg;
    end ;
```

5.15 syn_ramstyle

描述

指定 RAM 的实现方式。默认情况下，综合工具会根据 RAM 的大小提供最优推断，推断规则请参考 *SUG949，Gowin_HDL 编码风格用户指南*的 BSRAM 编码规范章节和 SSRAM 编码规范章节。

属性值

表 5-35 属性值

Default	Global Attribute	Object
block_ram	Yes	module/entity, reg/signal, module/entity instance

下表列出了所有可设置的属性值，以及每个属性值所对应的特定功能。

表 5-36 可设置的属性值及其对应功能

Value	功能
registers	指定 RAM 的实现方式为 registers。
block_ram	指定 RAM 的实现方式为 FPGA 中的块状静态随机存储器(BSRAM)。
distributed_ram	指定 RAM 的实现方式为 FPGA 中的分布式静态随机存储器(SSRAM)。
rw_check	指定在 RAM 周围插入逻辑，以防止对同一地址进行读写时输出不确定，从而造成 RTL 功能仿真与综合网表的功能仿真结果不匹配。
no_rw_check	指定不在 RAM 周围插入逻辑。默认情况下，此功能启用。

语法

表 5-37 语法示例

GSC 约束语法	INS " object " syn_ramstyle = value GLOBAL syn_ramstyle = value GSC Example
Verilog 约束语法	object /* synthesis syn_ramstyle = "value" */; Verilog Example
VHDL 约束语法	attribute syn_ramstyle : string; attribute syn_ramstyle of object : objectType is "value"; VHDL Example

注!

若对不支持 SSRAM 的器件使用属性值为 distributed_ram 的属性约束，综合工具会给出 warning，提示属性约束无效。

示例

- GSC 约束示例

  示例 1：指定 instance 的实现方式为 BSRAM。

  INS " object " syn_ramstyle=block_ram

  示例 2：指定全局所有 RAM 的实现方式为 SSRAM。

  GLOBAL syn_ramstyle=distributed_ram

- Verilog 约束示例

  示例 1：指定 module 内所有 RAM 的实现方式为 BSRAM。

```
module top(data_out, data_in, addr, clk,ce, wre,rst) /*synthesis
syn_ramstyle="block_ram"*/;
output [15:0]data_out;
input [15:0]data_in;
input [2:0]addr;
input clk,wre,ce,rst;
reg [15:0] mem [7:0]={16'h0123,16'h4567,16'h89ab,16'hcdef,
16'h0147,16'h0258,16'h789a,16'h5678};
reg [15:0] data_out=16'h0000;
always@(posedge clk )begin
    if(rst)begin
        data_out <= 0;
    end
    else begin
        if(ce & !wre)begin
            data_out <= mem[addr];
        end
    end
end
always @(posedge clk)begin
    if (ce & wre)begin
        mem[addr] <= data_in;
    end
end
endmodule
```

示例 2：指定 RAM 的实现方式为 registers。

```
module top(dout, din, ada, adb, clka, cea, clkb, ceb, resetb);
output reg[15:0]dout=16'h0000;
input [15:0]din;
input [9:0]ada, adb;
input clka, cea,clkb, ceb, resetb;
reg [15:0] mem [1023:0] /*synthesis syn_ramstyle="registers"*/;
always @(posedge clka)begin
    if (cea)begin
        mem[ada] <= din;
    end
end
always@(posedge clkb)begin
    if(resetb)begin
        dout <= 0;
    end
    else if(ceb)begin
        dout <= mem[adb];
    end
end
endmodule
```

示例 3：rw_check 示例。

```
module normal(data_out, data_in, addra, addrb, clka, cea, clkb);
parameter DATA_WIDTH = 8;
parameter ADDRESS_WIDTH = 8;

output [DATA_WIDTH-1:0]data_out;
input [DATA_WIDTH-1:0]data_in;
input [ADDRESS_WIDTH-1:0]addra, addrb;
input clka, cea,clkb;
reg [DATA_WIDTH-1:0] mem [2**ADDRESS_WIDTH-1:0] /*synthesis
syn_ramstyle=" rw_check "*/;
reg[ADDRESS_WIDTH-1:0]addrb_reg;
always @(posedge clka)begin
    if (cea)begin
```

```
            mem[addra] <= data_in;
        end
    end
    always @(posedge clkb)begin
        addrb_reg<=addrb;
    end
    assign data_out=mem[addrb_reg];

    endmodule
```

- VHDL 约束示例

  示例 1：指定 RAM 的实现方式为 SSRAM。

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity top is
  port(
    dout: out std_logic_vector(15 downto 0);
    din: in std_logic_vector(15 downto 0);
    ada: in std_logic_vector(9 downto 0);
    adb: in std_logic_vector(9 downto 0);
    clka: in std_logic;
    cea: in std_logic;
    clkb: in std_logic;
    ceb: in std_logic;
    resetb: in std_logic
  );
end top;

architecture behavioral of top is
  type memory_array is array(0 to 1023) of std_logic_vector(15
downto 0);
  signal mem : memory_array := (others => (others => '0'));
    attribute syn_ramstyle:string;
    attribute syn_ramstyle of mem: signal is " distributed_ram";
  signal dout_reg : std_logic_vector(15 downto 0) := (others => '0');
```

```
begin
  process(clka)
  begin
    if rising_edge(clka) then
      if cea = '1' then
        mem(to_integer(unsigned(ada))) <= din;
      end if;
    end if;
  end process;
  process(clkb)
  begin
    if rising_edge(clkb) then
      if resetb = '1' then
        dout_reg <= (others => '0');
      elsif ceb = '1' then
        dout_reg <= mem(to_integer(unsigned(adb)));
      end if;
    end if;
  end process;
  dout <= dout_reg;
end behavioral;
```

5.16 syn_romstyle

描述

指定 ROM 的实现方式。默认情况下，综合工具会根据 ROM 的大小提供最优推断，推断规则请参考 *SUG949，Gowin_HDL 编码风格用户指南*的 BSRAM 编码规范章节。

属性值

表 5-38 属性值

Global Attribute	Object
Yes	module/entity, reg/signal, module/entity instance

下表列出了所有可设置的属性值，以及每个属性值所对应的特定功能。

表 5-39 可设置的属性值及其对应功能

Value	功能
logic	指定 ROM 的实现方式为逻辑电路。
block_rom	指定 ROM 的实现方式为 FPGA 中的块状静态随机存储器(BSRAM)。
distributed_rom	指定 ROM 的实现方式为 ROM16。

语法

表 5-40 语法示例

GSC 约束语法	INS "object" syn_romstyle = value GLOBAL syn_romstyle = value GSC Example
Verilog 约束语法	object /* synthesis syn_romstyle = "value" */; Verilog Example
VHDL 约束语法	attribute syn_romstyle : string; attribute syn_romstyle of object : objectType is "value"; VHDL Example

注！

若对不支持 ROM16 的器件使用属性值为 distributed_rom 的属性约束，综合工具会给出 warning，提示属性约束无效。

示例

- GSC 约束示例

  示例 1：指定 instance 的实现方式为 BSRAM。

  INS " object " syn_romstyle=block_rom

  示例 2：指定全局 ROM 的实现方式为 ROM16。

  GLOBAL syn_romstyle=distributed_rom

- Verilog 约束示例

  示例 1：指定 module 内所有 ROM 的实现方式为 ROM16。

```
module top(addr,dataout)/*synthesis
syn_romstyle="distributed_rom"*/ ;
input [3:0] addr;
output reg dataout=1'h0;
always @(*)begin
    case(addr)
        4'h0: dataout <= 1'h0;
        4'h1: dataout <= 1'h0;
        4'h2: dataout <= 1'h1;
        4'h3: dataout <= 1'h0;
```

```
            4'h4: dataout <= 1'h1;
            4'h5: dataout <= 1'h1;
            4'h6: dataout <= 1'h0;
            4'h7: dataout <= 1'h0;
            4'h8: dataout <= 1'h0;
            4'h9: dataout <= 1'h1;
            4'ha: dataout <= 1'h0;
            4'hb: dataout <= 1'h0;
            4'hc: dataout <= 1'h1;
            4'hd: dataout <= 1'h0;
            4'he: dataout <= 1'h0;
            4'hf: dataout <= 1'h0;
            default: dataout <= 1'h0;
        endcase
    end
    endmodule
```

- VHDL 约束示例

  示例 1：指定 module 内所有的存储器实现方式为 BSRAM。

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity top is
    port (
        clk: in std_logic;
        rst: in std_logic;
        ce: in std_logic;
        addr: in std_logic_vector(4 downto 0);
        dout: out std_logic_vector(31 downto 0)
    );
end top;
architecture Behavioral of top is
    signal dout_reg : std_logic_vector(31 downto 0) := (others => '0');
    attribute syn_romstyle:string;
    attribute syn_romstyle of dout_reg : signal is "block_rom";
```

```
begin
    dout <= dout_reg;
    process(clk)
    begin
        if rising_edge(clk) then
            if rst = '1' then
                dout_reg <= (others => '0');
            elsif ce = '1' then
                case addr is
                    when "00000" => dout_reg <= x"52853FD5";
                    when "00001" => dout_reg <= x"38581BD2";
                    when "00010" => dout_reg <= x"040D53E4";
                    when "00011" => dout_reg <= x"22CE7D00";
                    when "00100" => dout_reg <= x"73D90E02";
                    when "00101" => dout_reg <= x"C0B4BF1C";
                    when "00110" => dout_reg <= x"EC45E626";
                    when "00111" => dout_reg <= x"D9D000D9";
                    when "01000" => dout_reg <= x"AACF8574";
                    when "01001" => dout_reg <= x"B655BF16";
                    when "01010" => dout_reg <= x"8C565693";
                    when "01011" => dout_reg <= x"B19808D0";
                    when "01100" => dout_reg <= x"E073036E";
                    when "01101" => dout_reg <= x"41B923F6";
                    when "01110" => dout_reg <= x"DCE89022";
                    when "01111" => dout_reg <= x"BA17FCE1";
                    when "10000" => dout_reg <= x"D4DEC5DE";
                    when "10001" => dout_reg <= x"A18AD699";
                    when "10010" => dout_reg <= x"4A734008";
                    when "10011" => dout_reg <= x"5C32AC0E";
                    when "10100" => dout_reg <= x"8F26BDD4";
                    when "10101" => dout_reg <= x"B8D4AAB6";
                    when "10110" => dout_reg <= x"F55E3C77";
                    when "10111" => dout_reg <= x"41A5D418";
                    when "11000" => dout_reg <= x"BA172648";
                    when "11001" => dout_reg <= x"5C651D69";
```

```
                when "11010" => dout_reg <= x"445469C3";
                when "11011" => dout_reg <= x"2E49668B";
                when "11100" => dout_reg <= x"DC1AA05B";
                when "11101" => dout_reg <= x"CEBFE4CD";
                when "11110" => dout_reg <= x"1E1F0F1E";
                when "11111" => dout_reg <= x"86FD31EF";
                when others => dout_reg <= x"8E9008A6";
            end case;
        end if;
    end if;
end process;
end Behavioral;
```

5.17 syn_srlstyle

描述

指定 shift registers 的实现方式。默认情况下，综合工具会根据 shift registers 的大小提供最优推断，推断规则请参考 *SUG949，Gowin_HDL 编码风格用户指南*的 BSRAM 编码规范章节和 SSRAM 编码规范章节。

属性值

表 5-41 属性值

Global Attribute	Object
Yes	module/entity, reg/signal, primitive instance

下表列出了所有可设置的属性值，以及每个属性值所对应的特定功能。

表 5-42 可设置的属性值及其对应功能

value	功能
registers	指定 shift registers 的实现方式为 registers。
block_ram	指定 shift registers 的实现方式为 FPGA 中的块状静态随机存储器(BSRAM)。
distributed_ram	指定 shift registers 的实现方式为 FPGA 中的分布式静态随机存储器(SSRAM)。
bsram_sdp	指定 shift registers 的实现方式为伪双端口模式的 BSRAM。

语法

表 5-43 语法示例

GSC 约束语法	INS "object" syn_ srlstyle = value GLOBAL syn_ srlstyle = value GSC Example
Verilog 约束语法	object /* synthesis syn_ srlstyle = "value" */; Verilog Example
VHDL 约束语法	attribute syn_ srlstyle: string; attribute syn_ srlstyle of object : objectType is "value"; VHDL Example

注！

若对不支持 SSRAM 的器件使用属性值为 distributed_ram 的属性约束，综合工具会给出 warning，提示属性约束无效。

示例

- GSC 约束示例

  示例 1：指定 instance 的实现方式为 BSRAM。

  INS "mem" syn_srlstyle=block_ram

  示例 2：指定全局所有 shift registers 的实现方式为 SSRAM。

  GLOBAL syn_srlstyle=distributed_ram

- Verilog 约束示例

  示例 1：指定 module 内 shift registers 的实现方式为 BSRAM。

```
module p_seqshift(clk, we, din, dout) /* synthesis syn_srlstyle =
"block_ram" */;
parameter width=18;
parameter depth=4;
input clk, we;
input [width-1:0] din;
output [width-1:0] dout;
reg [width-1:0] regBank[depth-1:0];
always @(posedge clk) begin
    if (we) begin
        regBank[depth-1:1] <= regBank[depth-2:0];
        regBank[0] <= din;
    end
end
assign dout = regBank[depth-1];
endmodule
```

示例 2：指定 instance 的实现方式为 SSRAM。

```
module p_seqshift(clk, we, din, dout);
parameter width=18;
parameter depth=16;
input clk, we;
input [width-1:0] din;
output [width-1:0] dout;
reg [width-1:0] regBank[depth-1:0] /* synthesis syn_srlstyle =
"distributed_ram" */;
always @(posedge clk) begin
    if (we) begin
        regBank[depth-1:1] <= regBank[depth-2:0];
        regBank[0] <= din;
    end
end
assign dout = regBank[depth-1];
endmodule
```

- VHDL 约束示例

示例 1：指定 module 内 shift registers 的实现方式为 registers。

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity p_seqshift is
    generic (
        width : integer := 18;
        depth : integer := 1024
    );
    port (
        clk   : in std_logic;
        we    : in std_logic;
        din   : in std_logic_vector(width-1 downto 0);
        dout : out std_logic_vector(width-1 downto 0)
    );
end p_seqshift;
```

```
architecture Behavioral of p_seqshift is
    type regBank_type is array (0 to depth-1) of
std_logic_vector(width-1 downto 0);
    signal regBank : regBank_type := (others => (others => '0'));
    attribute syn_srlstyle:string;
    attribute syn_srlstyle of regBank : signal is "registers";
begin
    process(clk)
    begin
        if rising_edge(clk) then
            if we = '1' then
                for i in depth-1 downto 1 loop
                    regBank(i) <= regBank(i-1);
                end loop;
                regBank(0) <= din;
            end if;
        end if;
    end process;
    dout <= regBank(depth-1);
end Behavioral;
```

5.18 syn_tlvds_io/syn_elvds_io

描述

指定差分 I/O buffer 映射的属性。

属性值

表 5-44 属性值

Default	Global Attribute	Object
0	Yes	module/entity, port, signal

下表列出了所有可设置的属性值，以及每个属性值所对应的特定功能。

表 5-45 可设置的属性值及其对应功能

Value	功能
0	禁用自动 infer 差分 I/O buffer。
1	启用自动 infer 差分 I/O buffer。

语法

表 5-46 语法示例

GSC 约束语法	PORT "object" syn_tlvds_io/syn_elvds_io = value GLOBAL syn_tlvds_io/syn_elvds_io = value GSC Example
Verilog 约束语法	object /* synthesis syn_tlvds_io/syn_elvds_io = value */; Verilog Example
VHDL 约束语法	attribute syn_tlvds_io/syn_elvds_io: integer; attribute syn_tlvds_io/syn_elvds_io of object : objectType is value; VHDL Example

示例

- GSC 约束示例

  示例 1：指定 buffer 的实现方式为 TLVDS。

  ```
  PORT "io" syn_tlvds_io =1;
  PORT "iob" syn_tlvds_io =1;
  ```

  示例 2：指定全局所有 buffer 的实现方式为 ELVDS。

  ```
  GLOBAL syn_elvds_io =1;
  ```

- Verilog 约束示例

  示例 1：指定 buffer 的实现方式为 TLVDS。

  ```
  module tlvds_ibuf_test(in1_p, in1_n, out);
  input in1_p /* synthesis syn_tlvds_io = 1*/;
  input in1_n /* synthesis syn_tlvds_io = 1*/;
  output reg out;
  always@(in1_p or in1_n) begin
      if (in1_p != in1_n) begin
          out = in1_p;
      end
  end
  endmodule
  ```

- VHDL 约束示例

  示例 1：指定 buffer 的实现方式为 ELVDS。

  ```
  library IEEE;
  use IEEE.STD_LOGIC_1164.ALL;
  entity elvds_obuf_test is
        port (
  ```

```
        in0 : in std_logic;
        out1 : buffer std_logic;
        out2 : buffer std_logic
    );
attribute syn_elvds_io: integer;
attribute syn_elvds_io of out1,out2: signal is 1;
end elvds_obuf_test;
architecture Behavioral of elvds_obuf_test is
begin
    out1 <= in0;
    out2 <= not out1;
end Behavioral;
```

5.19 translate_off/translate_on

描述

translate_off 之后的语句将在综合过程中被跳过，直到 translate_on 出现，常用于在综合时自动屏蔽一些语句。translate_off/translate_on 必须成对出现。

语法

表 5-47 语法示例

Verilog 约束语法	/* synthesis translate_off*/ 综合过程中被忽略的语句 /* synthesis translate_on*/ Verilog Example
VHDL 约束语法	-- synthesis translate_off 综合过程中被忽略的语句 -- synthesis translate_on VHDL Example

示例

- Verilog 约束示例

  /*synthesis translate_off*/与/*synthesis translate_on*/之间的 assign Nout =a*b 语句在综合过程中被忽略。

```
module top(a, b, dout, Nout);
input [1:0] a;
input [1:0] b;
output [1:0] dout;
output [3:0] Nout;
```

```
assign dout = a+b;
/*synthesis translate_off*/
assign Nout = a*b;
/*synthesis translate_on*/
endmodule
```

- VHDL 约束示例

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity top is
port(
    a : in std_logic_vector(1 downto 0);
    b : in std_logic_vector(1 downto 0);
    dout : out std_logic_vector(1 downto 0);
    Nout : out std_logic_vector(3 downto 0)
);
end top;
architecture rtl of top is
begin
dout <= a + b;
--synthesis translate_off
Nout <= a * b;
--synthesis translate_on
end rtl;
```

6 Report 用户文档

report 文档是在执行综合之后，生成的统计报告文件，文件名为*_syn.rpt.html（*为指定输出网表 vg 文件的名称)，包含 Synthesis Message、Synthesis Details、Resource、Timing。

6.1 Synthesis Message

Synthesis Message，即综合基本信息。主要包括综合的设计文件、当前 GowinSynthesis 版本、设计配置信息和运行时间等信息，如图 6-1 所示。

图 6-1 Synthesis Message

Synthesis Messages

Report Title	GowinSynthesis Report
Design File	/n9k/share/gwsw/sw_pub/testcase/gw1nsr-2/SYN/rom16_case/src/rom_bp_async_rst_addr_5_dout_35.v
GowinSynthesis Constraints File	---
Tool Version	V1.9.9.03
Part Number	GW1NSR-LV4CQN48GC6/I5
Device	GW1NSR-4C
Created Time	Tue Apr 23 15:20:27 2024
Legal Announcement	Copyright (C)2014-2024 Gowin Semiconductor Corporation. ALL rights reserved.

6.2 Synthesis Details

Synthesis Details，该标题下打印设计文件的顶层模块、综合各阶段的运行时间和占用内存、总运行时间和总占用内存，如图 6-2 所示。

图 6-2 Synthesis Details

Synthesis Details

Top Level Module	top
Synthesis Process	Running parser: CPU time = 0h 0m 0.109s, Elapsed time = 0h 0m 0.123s, Peak memory usage = 52.926MB Running netlist conversion: CPU time = 0h 0m 0s, Elapsed time = 0h 0m 0s, Peak memory usage = 52.992MB Running device independent optimization: Optimizing Phase 0: CPU time = 0h 0m 0s, Elapsed time = 0h 0m 0s, Peak memory usage = 53.133MB Optimizing Phase 1: CPU time = 0h 0m 0s, Elapsed time = 0h 0m 0s, Peak memory usage = 53.191MB Optimizing Phase 2: CPU time = 0h 0m 0s, Elapsed time = 0h 0m 0s, Peak memory usage = 53.254MB Running inference: Inferring Phase 0: CPU time = 0h 0m 0s, Elapsed time = 0h 0m 0s, Peak memory usage = 53.344MB Inferring Phase 1: CPU time = 0h 0m 0s, Elapsed time = 0h 0m 0s, Peak memory usage = 53.402MB Inferring Phase 2: CPU time = 0h 0m 0s, Elapsed time = 0h 0m 0s, Peak memory usage = 53.430MB Inferring Phase 3: CPU time = 0h 0m 0s, Elapsed time = 0h 0m 0s, Peak memory usage = 53.441MB Running technical mapping: Tech-Mapping Phase 0: CPU time = 0h 0m 0s, Elapsed time = 0h 0m 0s, Peak memory usage = 53.465MB Tech-Mapping Phase 1: CPU time = 0h 0m 0s, Elapsed time = 0h 0m 0s, Peak memory usage = 53.465MB Tech-Mapping Phase 2: CPU time = 0h 0m 0s, Elapsed time = 0h 0m 0s, Peak memory usage = 53.473MB Tech-Mapping Phase 3: CPU time = 0h 0m 0.039s, Elapsed time = 0h 0m 0.028s, Peak memory usage = 54.289MB Tech-Mapping Phase 4: CPU time = 0h 0m 0s, Elapsed time = 0h 0m 0s, Peak memory usage = 54.289MB Generate output files: CPU time = 0h 0m 0s, Elapsed time = 0h 0m 0.008s, Peak memory usage = 57.043MB
Total Time and Memory Usage	CPU time = 0h 0m 0.148s, Elapsed time = 0h 0m 0.159s, Peak memory usage = 57.043MB

6.3 Resource

Resource，即资源信息。主要包括资源使用统计和芯片占用统计。

资源使用表会统计用户设计中 I/O Port、I/O Buf、REG、LUT 等的数量。资源利用率表用于预估用户设计中 CFU Logics、Register、BSRAM、DSP 等在当前器件的资源占用率，如图 6-3 所示。

图 6-3 Resource

Resource

Resource Usage Summary

Resource	Usage
I/O Port	32
I/O Buf	32
IBUF	24
OBUF	8
Register	1032
DFFE	1032
LUT	680
LUT2	128
LUT3	512
LUT4	40
INV	1
INV	1

Resource Utilization Summary

Resource	Usage	Utilization
Logic	681(681 LUT, 0 ALU) / 4608	15%
Register	1032 / 4020	26%
--Register as Latch	0 / 4020	0%
--Register as FF	1032 / 4020	26%
BSRAM	0 / 10	0%

6.4 Timing

Timing，即时序统计。主要包括 Clock Summary、Max Frequency Summary、Detail Timing Paths Informations 等信息。

Clock Summary 主要描述网表的时钟信号信息，如下图 6-4 所示，给出一个时钟，频率为 100MHz，周期为 10ns，0ns 时为上升沿，5ns 时为下降沿。

图 6-4 Timing

Timing

Clock Summary:

NO.	Clock Name	Type	Period	Frequency(MHz)	Rise	Fall	Source	Master	Object
1	clk	Base	10.000	100.0	0.000	5.000			clk_ibuf/I

Max Frequency Summary 主要统计网表文件可以达到的时钟频率，并以此来衡量整个网表文件的时序是否达到要求。如下图 6-5 中所示。要求频率为 100MHz，实际时钟频率为 747.2MHz，满足时序要求。如果实际频率未达到要求频率，则不满足时序要求，需要进一步查看具体时序路径。

图 6-5 Max Frequency Summary

Max Frequency Summary:

No.	Clock Name	Constraint	Actual Fmax	Logic Level	Entity
1	clk	100.0(MHz)	747.2(MHz)	1	TOP

Detail Timing Paths Information 显示时序路径详细信息，默认为 5 条，所有时间值的默认单位均为纳秒。Path Summary 主要描述网表文件中的关键时序路径，起始节点及相关时延信息，如图 6-6 所示。Data Arrival Path 和 Data Require Path 主要描述关键时序路径，给出详细的连接关系，扇出信息如图 6-7 所示。Path Statistics 描述路径的时延信息，如图 6-8 所示。

图 6-6 Path Summary

Detail Timing Paths Information

Path 1

Path Summary:

Slack	8.662
Data Arrival Time	2.283
Data Required Time	10.945
From	reg2_s0
To	out2
Launch Clk	clk[R]
Latch Clk	clk[R]

图 6-7 连接关系、时延及扇出信息

Data Arrival Path:

AT	DELAY	TYPE	RF	FANOUT	NODE
0.000	0.000				clk
0.000	0.000	tCL	RR	1	clk_ibuf/I
0.982	0.982	tINS	RR	3	clk_ibuf/O
1.345	0.363	tNET	RR	1	reg2_s0/CLK
1.803	0.458	tC2Q	RF	3	reg2_s0/Q
2.283	0.480	tNET	FF	1	out2/D

Data Required Path:

AT	DELAY	TYPE	RF	FANOUT	NODE
10.000	0.000				clk
10.000	0.000	tCL	RR	1	clk_ibuf/I
10.982	0.982	tINS	RR	3	clk_ibuf/O
11.345	0.363	tNET	RR	1	out2/CLK
10.945	-0.400	tSu		1	out2

图 6-8 Path Statistics

Path Statistics:

Clock Skew:	0.000
Setup Relationship:	10.000
Logic Level:	1
Arrival Clock Path Delay:	cell: 0.982, 73.009%; route: 0.363, 26.991%
Arrival Data Path Delay:	cell: 0.000, 0.000%; route: 0.480, 51.155%; tC2Q: 0.458, 48.845%
Required Clock Path Delay:	cell: 0.982, 73.009%; route: 0.363, 26.991%


GOWIN高云
智慧逻辑 定制未来