GOWIN

PROGRAMMING FOR THE FUTURE

GowinSynthesis

User Guide

SUG550-1.9E, 12/31/2024

Disclaimer

GOWINSEMI assumes no liability and provides no warranty (either expressed or implied) and is not responsible for any damage incurred to your hardware, software, data, or property resulting from usage of the materials or intellectual property except as outlined in the GOWINSEMI Terms and Conditions of Sale. GOWINSEMI may make changes to this document at any time without prior notice. Anyone relying on this documentation should contact GOWINSEMI for the current documentation and errata.

Revision History

Date	Version	Description
08/02/2019	1.0E	Initial version published.
12/09/2019	1.1E	"Naming Rules of Objects before/after Synthesis" added. (Gowin Software V1.9.3 and above)
03/03/2020	1.2E	VHDL syntax design supported. (Gowin Software V1.9.5 and above)
05/29/2020	1.3E	● "Synthesis Naming Rules" modified. ● "syn_srlstyle" and "syn_noprune" attribute added.
09/14/2020	1.4E	"black_box_pad_pin" attribute added.
10/28/2021	1.5E	● "parallel_case" and "syn_black_box" attributes added; ● Chapter 6 Report Document modified.
06/30/2023	1.6E	Chapter 5 Synthesis Constraints Support updated.
05/09/2024	1.7E	● Chapter 4 HDL Code Support updated. ● Chapter 5 Synthesis Constraints Support updated. ● Screenshots in Chapter 6 Report Document updated.
08/09/2024	1.8E	● Descriptions of the number of synthesis attribute constraint objects added. ● Figure 6-4 Timing updated.
12/31/2024	1.9E	Chapter 5 Synthesis Constraints Support updated.

Contents

List of Figures

List of Tables

1 About This Guide

1.1 Purpose

It mainly describes functions and operations of GowinSynthesis and aims to help you learn how to use this software and improve design efficiency. The software screenshots in this manual are based on Gowin Software 1.9.11. As the software is subject to change without notice, some information may not remain relevant and may need to be adjusted according to the software that is in use.

1.2 Related Documents

The user guides are available on the GOWINSEMI Website. You can find the related documents at www.gowinsemi.com: SUG100, Gowin Software User Guide

1.3 Terminology and Abbreviation

Table 1-1 shows the abbreviations and terminology that are used in this guide.

Table 1-1 Abbreviations and Terminology

Terminology and Abbreviation	Meaning
BSRAM	Block Static Random Access Memory
DSP	Digital Signal Processing
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
GSC	Gowin Synthesis Constraint File
SSRAM	Shadow Static Random Access Memory

1.4 Support and Feedback

Gowin Semiconductor provides customers with comprehensive technical support. If you have any questions, comments, or suggestions, please feel free to contact us directly by the following ways.

Website: www.gowinsemi.com

E-mail: support@gowinsemi.com

2 Overview

This is the user guide of Gowin RTL design synthesis tool GowinSynthesis.

GowinSynthesis is designed in-house by GOWINSEMI, and it uses Gowin original EDA algorithm to realize RTL design extraction, arithmetic optimization, inference, resource sharing, parallel synthesis and mapping based on product hardware characteristics and hardware circuit resources, which can quickly optimize your RTL design, resource check, and timing analysis.

GowinSynthesis, targeting Gowin FPGA chips, provides the most efficient design implementation method for FPGA designers. GowinSynthesis can generate a post-synthesis netlist based on the Gowin device primitive library, which can be used as an input file for the PnR, achieving the optimal balance of area and speed, improving software compilation efficiency, and routing rate. The software has the following features:

- Supports Verilog/SystemVerilog, VHDL and mixed design input.
- Supports ultra-large-scale design, providing an excellent synthesis solution for complex programmable logic designs.
- Supports inferred mapping of look-up tables, registers, latches, and arithmetic logic units.
- Supports memory inferred mapping and logic resources balancing.
- Supports DSP inferred mapping and logic resources balancing.
- Supports synthesis optimization of FSM
- Supports synthesis attributes and instructions to meet the synthesis requirements in different applications.

3 GowinSynthesis Usage

3.1 Input and Output of GowinSynthesis

GowinSynthesis reads user's RTL file in the format of project file (.prj), and the project file is automatically generated by Gowin Software. In addition to the specified user RTL file, GowinSynthesis project file also specifies the synthesis device, the user constraints file including attribute constraints, post-synthesis netlists file (.vg), and some synthesis options, such as top module, files including paths, etc.

3.2 Use GowinSynthesis for Synthesis

Right-click "Synthesize" in the "Process" view of Gowin Software, select "Configuration ". In the Configuration page, you can specify top module, set include path, and select language versions, and configure options.

Double-click "Synthesize" in "Process" view of Gowin Software to perform synthesis, and the "Output" view will output the synthesis information. The synthesis report and gate level netlist file will be generated after synthesis. Double-click the "Synthesis Report" and "Netlist File" in the "Process" view to check the specific contents.

For the further detailed operation, see *SUG100, Gowin Software User Guide* > 4.4.3 Synthesize.

3.3 Naming Rules of Objects Pre/Post Synthesis

For user-friendly verification and debugging, GowinSynthesis synthesis tool will reserve the original user RTL design info. in maximum, such as the module info., primitive/module instance name, the user-defined wire/reg name in user designs, etc. For the objects that must be optimized/converted and to be regenerated, the name will be created according to the user-defined wiring name and some derivative rules. The rules are described in the sections below.

3.3.1 Naming of the Post-synthesis Netlist File

The post-synthesis netlist file name depends on the specified output netlist file name in project file (*.prj).

If the post-synthesis netlist file name is not specified in project file, the name is created as the same as the project file with a .vg suffix.

3.3.2 Naming of the Post-synthesis Netlist Module

The post-synthesis netlist module name is consistent with RTL design name. Modules that are instantiated multiple times are distinguished by the suffix "_idx", and the module's instantiation name is consistent with the RTL design.

3.3.3 Naming of the Post-synthesis Netlist Instance

If there is Instance in user RTL design and it is not optimized in the process of synthesis, the Instance name stays the same in the post-synthesis netlist.

For the Instance generated in the process of synthesis, the Instance name comes from the the external output signal name of the function design module in the user RTL. If the function design module has multiple output signal, the Instance name depends on the first output signal name.

For the Instance generated in the process of synthesis, the Instance name contains the names described above, and they are also followed by a suffix based on type. The "buf" types are suffixed with "_ibuf", "_obuf", "_iobuf", and for others with "_s", the number after "s" is the number of times the name is referenced by the node.

When flatten is specified to output, if the original submodule Instance name needs hierarchy, "/" can be used as hierarchical separator.

3.3.4 Naming of the Post-synthesis Netlist Wiring

For the user-defined wire/reg signal in RTL design, if the signal is not optiminzed in the process of synthesis, the related module name of the post-synthesis netlist stays the same.

In the process of synthesis using GowinSynthesis, some whole functional design modules in some RTL designs will be replaced or optimized. After synthesis, the output signal name of these functional design modules in netlist will be kept. For the internal signal of these modules, the name will be derived from the name of the output signal. The related digital suffix (_idx) will be added on the basis of the original signal name.

When the multi-bit signal name (the bus format) is used as the derived signal name and other signal names or Instance name is derived, the bus bit in the singal name will be kept in the form of "_idx".

When flatten is specified to output, "_" can be used as hierarchical separator.

4 HDL Code Support

4.1 Register HDL Code Support

4.1.1 An Introduction to Register Features

The registers contain flip-flops and latches.

Flip-flop

The flip-flops are all D flip-flops, and the initial value is assigned when defined. There are two types of reset/set: Synchronous and asynchronous. Synchronous reset/set means that only when the posedge or negedge of CLK arrives, and reset/set is at high level, can the reset/set be completed. Asynchronous reset/set means the level change from low to high of reset/set will lead to the output change immediately, but not controlled by CLK.

Latch

The latches trigger includes high-level trigger and low-level trigger, and the initial value is assigned when defined. In FPGA design, it is best to avoid latches. High level trigger is when the control signal is high; the latch allows the data signal to pass through. Low level trigger is when the control signal is low; latch allows the data signal to pass through.

4.1.2 Constraints Related with Register

You can constrain register by the preserve attribute. When this constraint exists, except the registers that are suspended will be optimized, all the other registers will be preserved in the synthesis results. Please see syn_preserve for details.

4.1.3 Register Code Example

The initial value of Gowin synchronous reset clock flip-flop can only be set to 0. The initial value of synchronous set clock flip-flop can only be set to 1. When the initial value of the synchronous clock flip-flop in the RTL is different from the initial value of Gowin synchronous clock, GowinSynthesis will convert the type of synchronous clock flip-flop based on the initial value in the RTL. Asynchronous clock flip-flop does not be handled. Specific conversion strategies are:

RTL is designed as synchronous reset clock flip-flop. When the initial value is set to 1, GowinSynthesis will replace it with synchronous set clock flip-flop. Add related logic to the original synchronous reset signal to realize synchronous set function.

RTL is designed as synchronous set clock flip-flop. When the initial value is set to 0, GowinSynthesis will replace it with normal flip-flop. Add related logic to the original synchronous set signal as the input of the flip-flop data end.

Not specify the Initial Value of Flip-flop

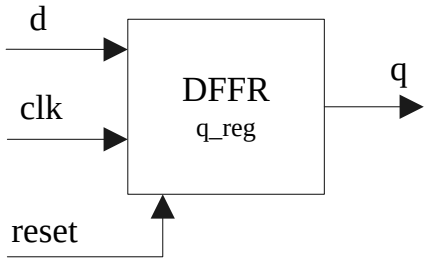
The only difference between CLK posedge flip-flop and CLK negedge flip-flop is that CLK triggers in different ways, so the following list is only the examples of CLK rising edge flip-flop.

Example 1 can be synthesized as synchronous reset clock flip-flop.

```
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
```

Synchronous reset clock flip-flop is as shown in Figure 4-1.

Figure 4-1 Synchronous Reset Clock Flip-flop in Example 1

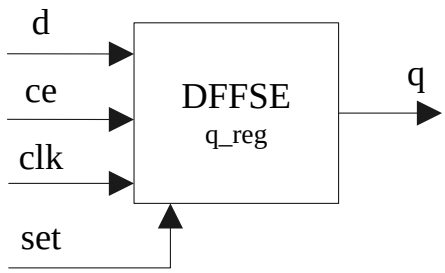


Example 2 can be synthesized as synchronous set flip-flop with clock enable.

```
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
```

Synchronous set flip-flop with clock enable is as shown in Figure 4-2.

Figure 4-2 Synchronous Set Flip-flop with Clock Enable in Example 2



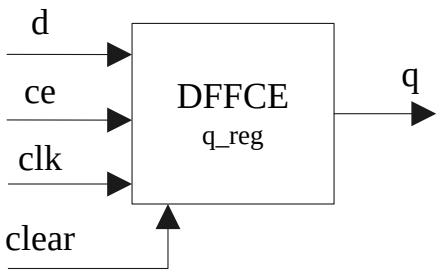
Example 3 can be synthesized as asynchronous reset flip-flop with clock enable.

```
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
```

Asynchronous reset flip-flop with clock enable is as shown in Figure 4-3.

Figure 4-3 Asynchronous Reset Flip-flop with Clock Enable in Example 3

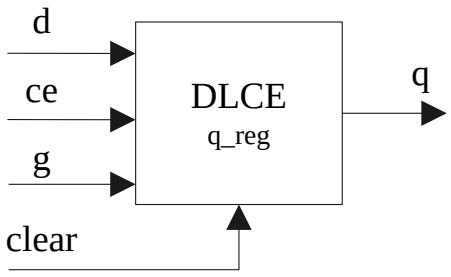


Example 4 can be synthesized as latch with reset and high level enable.

```
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
```

The latch with reset and high level enable is shown as in Figure 4-4.

Figure 4-4 Latch with Reset and High Level Enable in Example 4



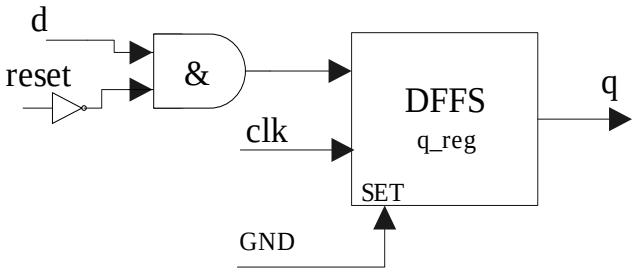
Specify the Initial Value of Flip-flop

Example 5 is synchronous reset clock flip-flop with an initial value of 0. Set an initial value of 1 in RTL, which will be synthesized as synchronous set clock flip-flop with an initial value of 1 and a logic circuit for synchronous reset.

```
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
```

Synchronous reset clock flip-flop above is as shown in Figure 4-5.

Figure 4-5 Synchronous Reset Clock Flip-flop and Logic Circuit in Example 5



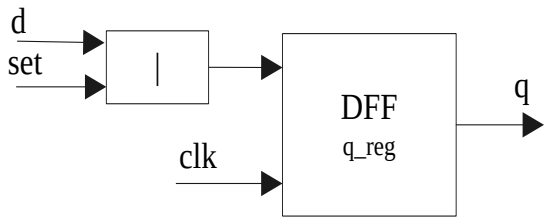
Example 6 is synchronous set clock flip-flop with an initial value of 1, but its initial value is 0 in RTL; and this synchronous set clock flip-flop will

be synthesized as common clock flip-flop with an initial value of 0 and a logic circuit for synchronous set.

```
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
```

The common clock flip-flop with the initial value of 0 and logic circuit are shown in Figure 4-6.

Figure 4-6 Common Clock Flip-flop with an Initial Value of 0 and Logic Circuit in Example 6



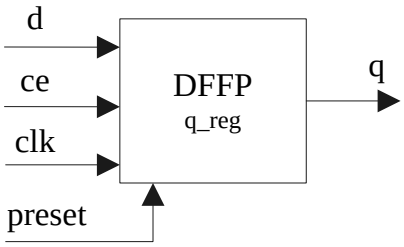
Example 7 is an asynchronous set flip-flop with an initial value of 1.

```
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
```

Asynchronous set flip-flop above is as shown in Figure 4-7.

Figure 4-7 Asynchronous Set Flip-flop in Example 7



4.2 RAM HDL Code Support

4.2.1 An Introduction to RAM Inference Function

RAM inference is one step in RTL synthesis to infer block memory primitives (BSRAM and SSRAM) in FPGA to implement memory functions in user design so that you can write device-independent RTL or use embedded block ram functionality in FPGA. For RTL memory blocks, GowinSynthesis will infer the RTL that meets the corresponding conditions to RAM module according to RTL description.

If the design needs to implement by BSRAM, the following principles need to be met:

1. All output registers have the same control signal.
2. RAM must be synchronous memory and can not connect to asynchronous control signal. GowinSynthesis does not support asynchronous RAM.
3. It needs to connect registers at read address or output port.

4.2.2 An Introduction to RAM Features

BSRAM

There are four configuration modes for BSRAM: single-port, dual-port, semi-dual-port and read-only. Read mode includes pipeline and bypass. Write mode includes normal, write-through and read-before-write.

SSRAM

There are three configuration modes for SSRAM: Single-port, semi-dual-port and read-only. SSRAM does not support dual-port mode.

4.2.3 Constraints Related with RAM Inference

Syn_ramstyle specifies how memory is inferenced, and syn_romstyle

specifies how read-only memory is implemented.

If the design needs to generate SSRAM or BSRAM, please use ram_style, rom_style or syn_srlstyle constraint statement.

For constraint syntax use, please see syn_ramstyle and syn_romstyle.

4.2.4 RAM Inference Code Example

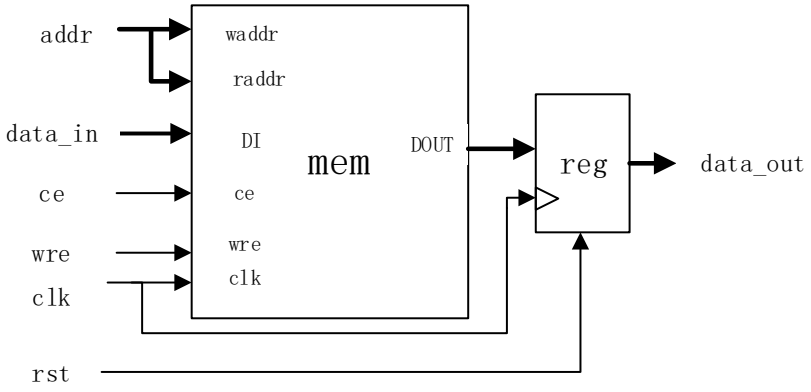
According to the different features of RAM, examples are as follows:

Example1 is a memory with one write port, one read port and the same read and write address, which can be synthesized to a single port BSRAM in normal mode.

```
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
```

The above single-port BSRAM circuit diagram is shown in Figure 4-8.

Figure 4-8 RAM Circuit Diagram in Example 1

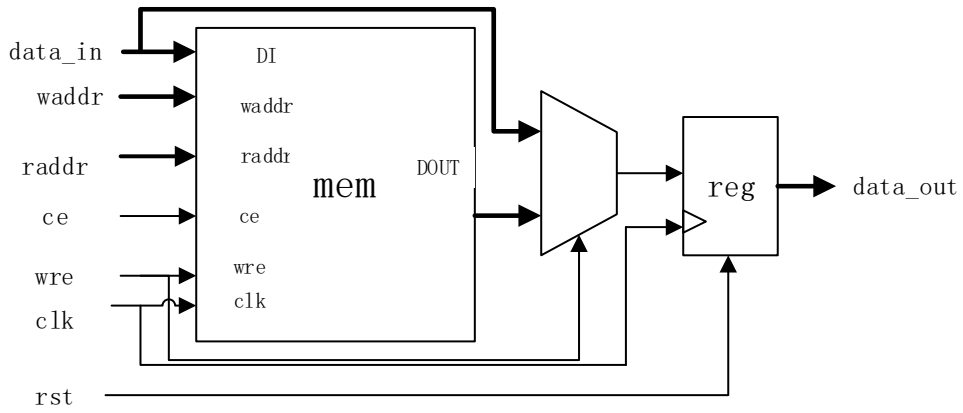


Example 2 is a memory with one write port, one read port and the same read and write address. When wre is 1, input data can be transferred directly to output, which can be synthesized to single-port BSRAM in normal write mode.

```
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
```

The above single-port BSRAM circuit diagram is shown in Figure 4-9.

Figure 4-9 RAM Circuit Diagram in Example 2

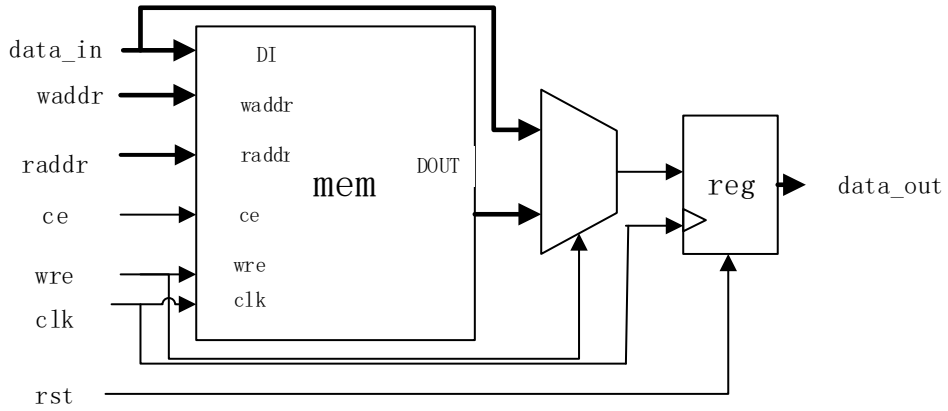


Example 3 is a memory with one write port, one read port and the same read and write address. When wre is 1, input data is written to memory, which can be synthesized to single-port BSRAM in read-before-write mode.

```
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
```

The above single-port BSRAM circuit diagram is shown in Figure 4-10.

Figure 4-10 RAM Circuit Diagram in Example 3



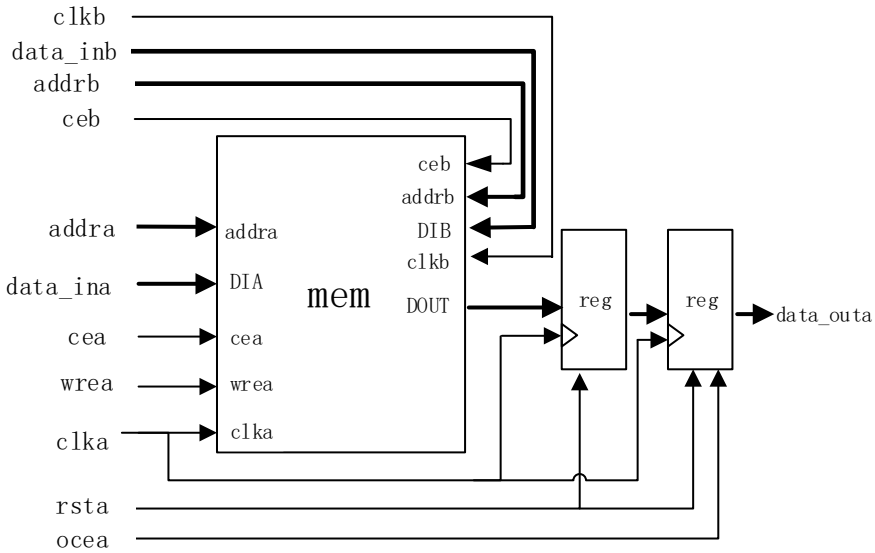
Example 4 is a memory with two write ports and one read port. One of the two write ports has a wre signal and the other does not. The read port absorbs asynchronous reset register. This example can be synthesized to asynchronous reset dual-port BSRAM with A port in normal write mode and B port in read-before-write mode or in register output read mode.

```
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
```

The above dual-port BSRAM circuit diagram is shown in Figure 4-11.

Figure 4-11 RAM Circuit Diagram in Example 4

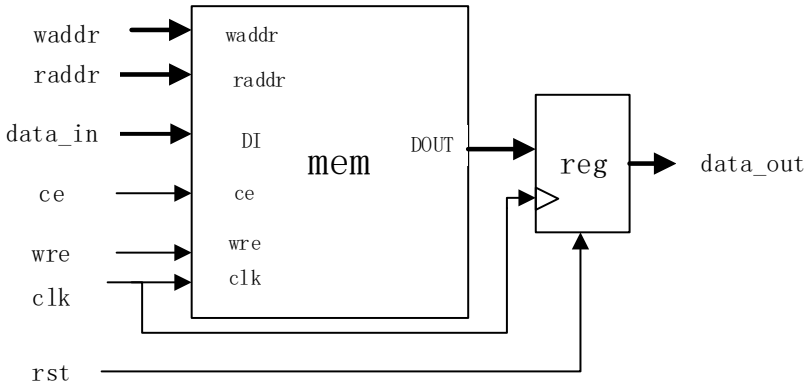


Example 5 is a memory with one read port and one write port and different read and write addresses, which can be synthesized to semi-dual-port BSRAM in normal write mode or in bypass read mode.

```
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
```

The above semi-dual-port BSRAM circuit diagram is shown in Figure 4-12.

Figure 4-12 RAM Circuit Diagram in Example 5



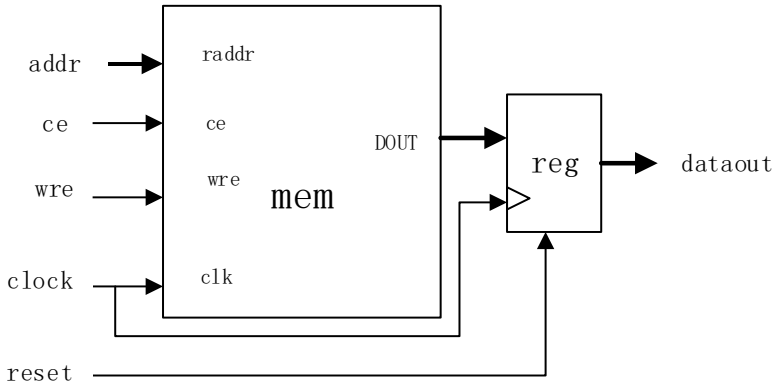
Example 6 is a memory with one read port and an initial value, which can be synthesized to asynchronous set read-only memory in bypass read mode.

```
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
```

The above read-only memory circuit diagram is shown in Figure 4-13.

Figure 4-13 pROM Circuit Diagram in Example 6



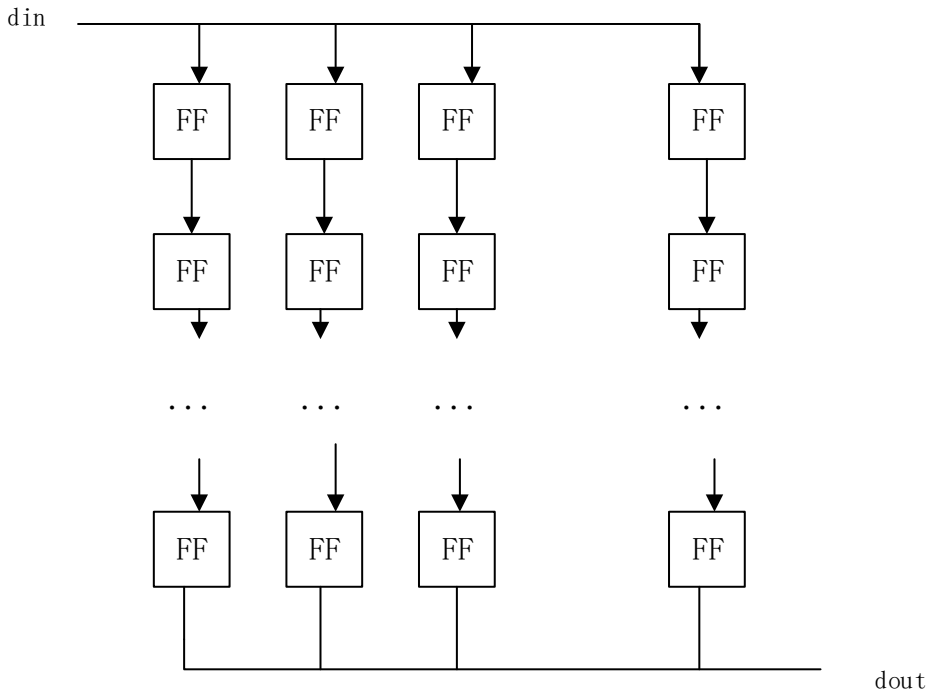
Example 7 is a memory with shift-register mode, which can be synthesized to simple-dual-port BSRAM in normal mode.

```
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
```

The above semi-dual-port BSRAM circuit diagram is shown in Figure 4-14.

Figure 4-14 RAM Circuit Diagram in Example 7



Note!

For more examples, please see GowinSynthesis_Inference_Coding_Template at Gowinsemi official website.

4.3 DSP HDL Code Support

4.3.1 Basic Introduction to DSP Inference

DSP inference is an algorithm that infers and permutes multiplication and partial addition in user design to DSP in RTL synthesis. When designing RTL, you can either instantiate DSP or write DSP description in device-independent RTL. For the multiplication and addition module of RTL, GowinSynthesis will permute RTL description meeting corresponding conditions with corresponding DSP module.

DSP module has functions of multiplication, addition and register. GowinSynthesis uses logic circuits to realize multiplier functions when the current device does not support DSP modules.

4.3.2 Introduction to DSP Features

Gowin DSP includes multiplier, multiply add accumulator and preadder. The following functions are supported:

1. Supports multiplication permutation of different sign bits input
2. Supports synchronous or asynchronous mode
3. Supports multiplication chain addition
4. Supports multiplication accumulation
5. Supports pre-add function
6. Supports register absorption, including input register, output register, bypass register

4.3.3 Constraints Related with DSP

Syn_dspstyle is used to control the multipliers or specific objects using DSP or logic circuits.

Syn_perserve is used to reserve registers. When register around the DSP has this property, the DSP cannot absorb this register.

For the constraint statements, please see syn_dspstyle and syn_preserve.

4.3.4 DSP Inference Code Example

Example 1 can be synthesized to synchronous set multiplier with sign bit. The input registers are ina and inb. The output register is out_reg, and the bypass register is 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.

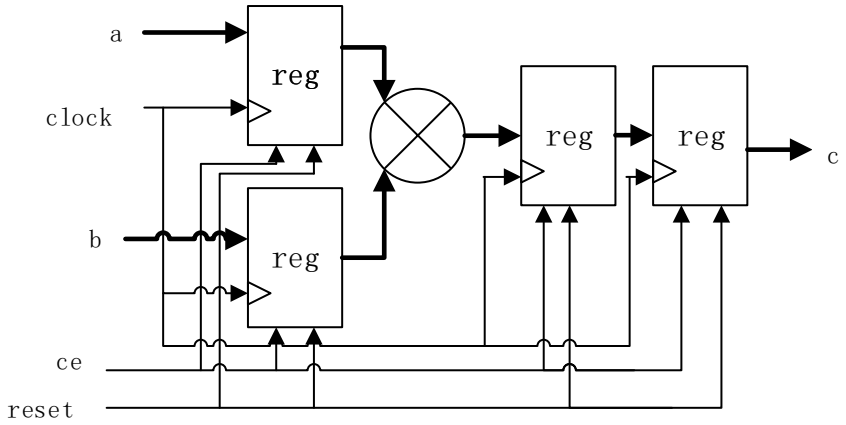
input signed [b_width-1:0] b.

```
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
```

The above multiplier circuit diagram is shown in Figure 4-15.

Figure 4-15 DSP Circuit Diagram in Example 1



Example 2 can be synthesized to a multiplier accumulator in asynchronous mode, which has input registers a0_reg, a1_reg, b0_reg and b1_reg, output register s_reg and bypass registers p0_reg and p1_reg.

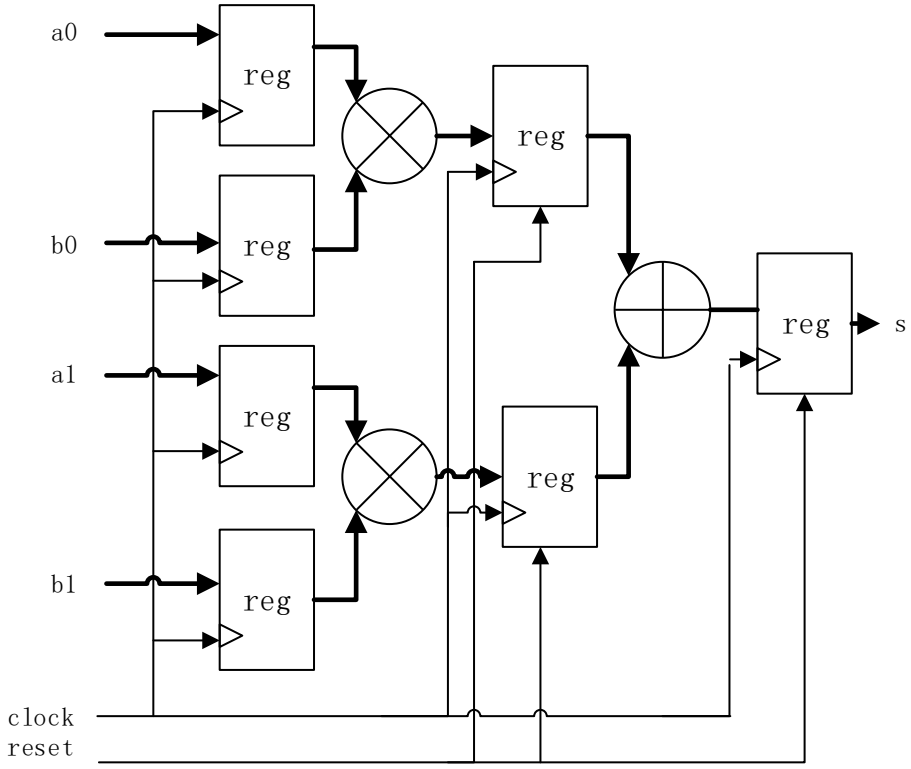
```
module top(a0, a1, b0, b1, s, reset, clock, ce).
parameter a0_width=18.
parameter a1_width=18.
parameter b0_width=18.
parameter b1_width=18.
parameter s_width=37.
input unsigned [a0_width-1:0] a0.
input unsigned [a1_width-1:0] a1.
input unsigned [b0_width-1:0] b0.
input unsigned [b1_width-1:0] b1.
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
```

The above multiplier accumulator circuit diagram is shown in Figure 4-16.

Figure 4-16 DSP Circuit Diagram in Example 2



Example 3 can be synthesized to two unsigned bit multipliers, which are in chain addition relation.

```
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.
```

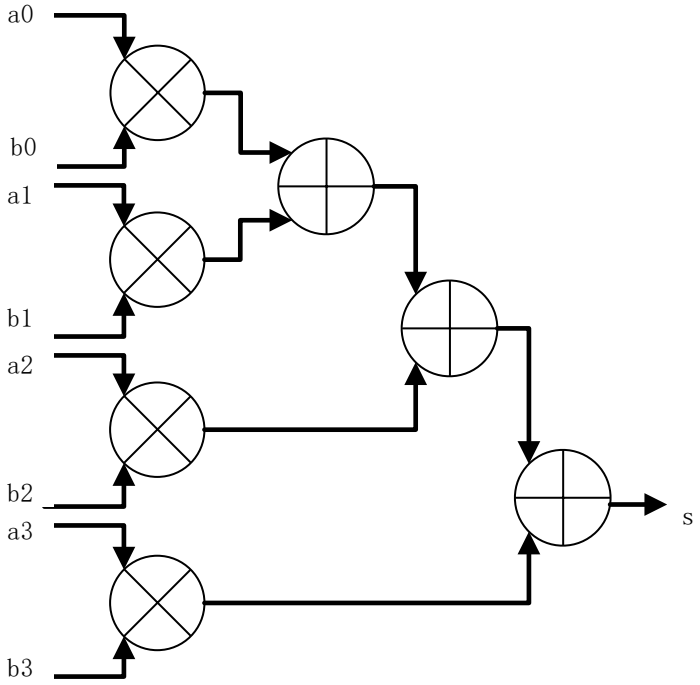
output unsigned [s_width-1:0] s.

*assign s=a0*b0+a1*b1+a2*b2+a3*b3.*

endmodule

The above multiplier accumulator circuit diagram is shown in Figure 4-17.

Figure 4-17 DSP Circuit Diagram in Example 3



Example 4 can be synthesized to a multiplier with sign-bit 0 and a preadder with sign-bit 0. An input port of this multiplier is connected to the output port b of the preadder.

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.

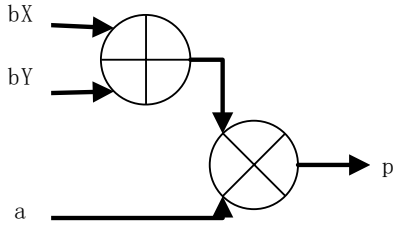
assign b = bX + bY.

*assign p = a*b.*

endmodule

The above multiplier accumulator circuit diagram is shown in Figure 4-18.

Figure 4-18 DSP Circuit Diagram in Example 4

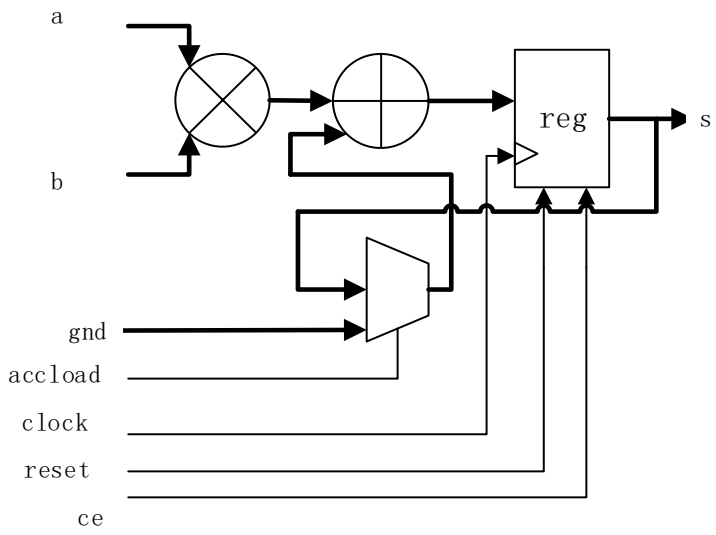


Example 5 can be synthesized to a multiplier accumulator with sign-bit 0, and the output register is s.

```
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
```

The above multiplier accumulator circuit diagram is shown in Figure 4-19.

Figure 4-19 DSP Circuit Diagram in Example 5



Note!

For more examples, please see GowinSynthesis_Inference_Coding_Template at Gowinsemi official website.

4.4 Synthesis Implementation Rules for Finite State Machine

4.4.1 Synthesis Rules for Finite State Machine

GowinSynthesis supports the synthesis of Finate State Machine (FSM), and the encoding mode supports one-hot code, gray code, binary code, etc. The synthesis results of finite state machine are related to its encoding mode, code number, code bit width and code constraints. Without specifying encoding constraints, GowinSynthesis automatically selects one-hot code, Gray code, or binary code to implement state machine. With specifying encoding constraints, the code method specified by the constraints should be implemented first. For the code constraints of state machine, please see syn_encoding.

Note!

It should be noted that if the output of the finite state machine drives the output port directly, GowinSynthesis will not synthesize it as a state machine, and the code constraints of the state machine will be ignored.

4.4.2 Finite State Machine Code Example

The synthesis rules for finite state machine are described below.

One-hot Code State Machine

If the state machine adopts one-hot code for coding in RTL design, GowinSynthesis will select one-hot code by default to realize the functions of the state machine with no code constraints. In the case of code constraints, the functions of the state machine are realized according to the encoding mode specified by the constraints. One-hot encoding mode example is as follows:

reg [3:0] state,next_state;

parameter state0=4'b0001;

parameter state1=4'b0010;

parameter state2=4'b0100;

parameter state3=4'b1000;

In the above example, RTL uses one-hot code and GowinSynthesis uses one-hot code to implement.

Gray Code State Machine

If the state machine adopts gray code for coding in the RTL design, GowinSynthesis will select gray code by default to realize the functions of the state machine with no code constraints. In the case of code constraints, the functions of the state machine are realized according to the encoding mode specified by the constraints. Gray code example is as follows:

reg [3:0] state,next_state;

parameter state0=2'b00;

parameter state1=2'b01;

parameter state2=2'b11;

parameter state3=2'b10;

In the above example, RTL uses gray code and GowinSynthesis uses gray code to implement.

Binary Code or other Codes State Machines

If the state machine adopts binary code in RTL design, which is neither one-hot code nor gray code, GowinSynthesis will select the corresponding code according to the number of codes and bit width for implementation with no constraints. The selection principle is as follows: If the number of code is greater than the effective bit width of code, binary code will be used for implementation. If the number of code is less than or equal to the effective bit width of code, one-hot code will be used for implementation. In the case of code constraints, the functions of state machine are realized according to the encoding mode specified by the constraints.

Example 1

reg [5:0] state,next_state;

parameter state0= 6'b000001;

parameter state1= 6'b000011;

parameter state2= 6'b000000;

parameter state3= 6'b010101;

In the above example, the number of code is 4, the bit width of code is 6, and the effective bit width is 5, so the number of code is less than the effective bit width of code, and the implementation is carried out by one-hot code.

Example 2

reg [2:0] state,next_state;

parameter state0=3'b001;

parameter state1=3'b010;

parameter state2=3'b011;

parameter state3=3'b100;

In the above example, the number of code is 4, and the effective bit width of code is 3. The number of code is larger than the effective bit width of code, and the implementation is carried out by binary code.

Example 3

reg [5:0] state,next_state.

parameter state0= 1,

parameter state1= 3,

parameter state2= 6,

parameter state3= 15.

In the above example, the number of code is 4 in decimal, and the effective bit width converted into binary is 4 bits. The number of code is equal to the effective bit width of code, and the implementation is carried out by one-hot code.

5 Synthesis Constraints Support

Attribute constraints is used to set various attributes of optimization selection, function implement, output netlist format in synthesis so that the synthesis results can better meet the design function and usage. Attributes can be written in constraint files or embedded in source code.

This chapter describes the syntax, scope, and priority of constraints in RTL files and GowinSynthesis Constraint (GSC) files. Verilog files are case-sensitive, so instructions and attributes must be typed exactly as described in the syntax. An attribute constraint must be written in the same line in a constraint statement and can not be separated by breaks.

Syntax of Attribute Constraints in RTL File

Constraints in the RTL file must be added in the definition statement of the constraint object, and placed before the terminating semicolon. The specific syntax is as follows:

object /*synthesis attributeName=value*/;

If the attribute value (value) is a string, it must be enclosed in double quotation marks.

Syntax of Attribute Constraints in GSC

GSC constraints include Instance constraints, Net constraints, Port constraints and global objects constraints. In GSC, if the attribute value is a string, quotation marks are not required. GSC constraints support comments, which use //. The specific syntax is as follows:

INS "object" attributeName=value;

NET "object" attributeName=value;

PORT "object" attributeName=value;

GLOBAL attributeName=value;

The constraint statement begins with INS, and the object must be the name of instance. Instance includes module/entity instance and primitive instance.

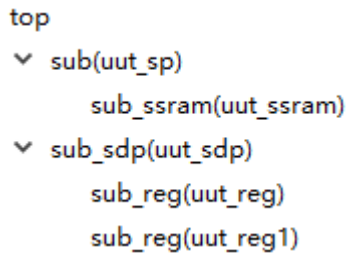
The constraint statement begins with NET, and the constraint object must be the NET name.

The constraint statement begins with PORT, and the constraint object must be the PORT name.

The constraint statement begins with GLOBAL, indicating that the attribute constraint is global.

In GSC, the / is used to distinguish hierarchical levels in names. For example, as shown in Figure 5-1, if you want to add an attribute constraint to the mem signal in the sub_ssram module, it can be expressed as: INS "uut_sp/uut_ssram/mem" syn_ramstyle = block_ram

Figure 5-1 Design Hierarchy



Scope of Attribute Constraints

- If an attribute constraint is set at the definition of a top module, the scope of the attribute value is the entire project.
- If an attribute constraint is set at the definition of a sub module, the scope of the attribute value is from the current sub layer to its sub modules.
- If an attribute constraint is set at the instantiation of a sub module, the scope of the attribute value is from the instantiation of the sub layer to the its sub modules.
- If an attribute constraint is set on reg/wire, the scope of the attribute value is that reg/wire.

Each attribute constraint can only be applied to a single constraint object. To constrain multiple objects, the attribute constraint must be added multiple times.

For example, to add attribute constraints to reg dout1 and dout2 in an RTL design, you can use the following methods:

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

Attribute Priority

When the same attribute is used with different values in different locations within the project, priority rules are applied as follows:

- When different attribute constraint values are applied to the same location (e.g., constraining the top module at the same time), the priority of the attribute constraints is as follows:

Constraints set in GSC > Constraints set in RTL > Configuration in the IDE interface.

- When different attribute constraint values are used at different locations in the design, the priority of the attribute constraints is as follows:

Constraints on registers/wires > Constraints on submodules in GSC > Constraints on submodules in RTL during its definition > Constraints on submodules in RTL during its instantiation > Constraints when defining the top module.

5.1 syn_black_box

Description

Specify a module or component as a black box. During synthesis, a black box module only defines its interface, while its contents are not accessible or optimized. A module can be designated as a black box regardless of whether it is empty or not.

Attribute Value

Table 5-1 Attribute Value

Default	Global Attribute	Object
N/A	No	module/component

Syntax

Table 5-2 Syntax Example

Verilog Constraint Syntax	object /* synthesis syn_black_box */; Verilog Example
VHDL Constraint Syntax	attribute syn_black_box: boolean; attribute syn_black_box of object: objectType is true; VHDL Example

Note!

object can only be a sub module/entity.

Examples

- Verilog Constraint Example

```
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 Constraint Example

```
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

Description

The specified IO pads of black box are visible to the external environment. This attribute is used in conjunction with syn_black_box and only applies to the IO pads of the black box.

Attribute Value

Table 5-3 Attribute Value

Default	Global Attribute	Object
N/A	No	module/component

The table below lists the configurable attribute value and its corresponding function.

Table 5-4 Configurable Attribute Value and its Corresponding Function

Value	Function
portList	Specifies the ports of the black box as I/O pads.

Syntax

Table 5-5 Syntax Example

Verilog Constraint Syntax	object /* synthesis syn_black_box black_box_pad_pin = portList */; Verilog Example

VHDL Constraint Syntax	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

Note!

The portList enclosed in double quotation marks is a comma-separated list without spaces, specifying the names of the ports on the black box.

Examples

- Verilog Constraint Example

```
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 Constraint Example

```
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

Description

full_case is only used in Verilog RTL designs. Adding this attribute after a case, casex, or casez statement indicates that all possible values are covered, and no additional hardware is required to retain signal values.

Syntax

Table 5-6 Syntax Example

Verilog Constraint Syntax	object /*synthesis full_case*/ Verilog Example

Example

- Verilog Constraint Example

```
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

Description

It is only used in Verilog RTL design. Adding this attribute after a case, casex, or casez statement indicates the use of a parallel multiplexer structure instead of a priority-encoded structure.

By default, case statements operate in priority order, encoding only the highest-priority signal among multiple input signals. However, the parallel_case enforces the use of a parallel multiplexer structure rather than a priority-encoded structure.

Syntax

Table 5-7 Syntax Example

Verilog Constraint Syntax	object /*synthesis parallel_case*/ Verilog Example

Example

- Verilog Constraint Example

```
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

Description

Specify that the multiplier is implemented using either a dedicated DSP hardware module or logic circuits. By default, structures such as multiplication, multiply-add/subtract, multiply-accumulate, multiply-cascade, multiplier-based pre-add/subtract, multiplier-based shifting, and large add/subtract operations are inferred as DSP hardware modules.

Attribute Value

Table 5-8 Attribute Value

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

The table below lists the configurable attribute value and its corresponding function.

Table 5-9 Configurable Attribute Value and its Corresponding Function

Value	Function
dsp	Specifies the implementation of the multiplier as a DSP hardware module.
logic	Specifies the implementation of the multiplier as logic circuits.

Syntax

Table 5-10 Syntax Example

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

Note!

If the attribute constraint with a value of dsp is applied to a device that does not support DSP hardware modules, the synthesis tool will issue a warning indicating that the attribute constraint is invalid.

Examples

- GSC Constraint Example

Example 1: Specifies the implementation of the instance as logic circuits.

INS "temp" syn_dspstyle=logic;

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

Example 2: Specifies the implementation of all global multipliers as DSP hardware modules.

GLOBAL syn_dspstyle=dsp;

- Verilog Constraint Example

Example 1: Specifies the implementation of all multipliers in the module as 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
```

Example 2: Specifies the implementation of the selected multiplier as 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 Constraint Example

Example 1: Specifies the implementation of the selected multiplier as 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

Description

Specify the encoding method for the finite state machine. By default, the encoding method of the synthesis result matches the encoding method used in the design.

Attribute Value

Table 5-11 Attribute Value

Global Attribute	Object
No	reg/type

The table below lists the configurable attribute value and its corresponding function.

Table 5-12 Configurable Attribute Value and its Corresponding Function

Value	Function
onehot	Specifies the encoding method for the finite state machine as

Value	Function
	one hot.
gray	Specifies the encoding method for the finite state machine as Gray.

Syntax

Table 5-13 Syntax Example

Verilog Constraint Syntax	object /* synthesis syn_encoding = "value" */ ; Verilog Example
VHDL Constraint Syntax	attribute syn_encoding : string; attribute syn_encoding of object : objectType is "value" ; VHDL Example

Note!

value: The encoding method for the state machine. The encoding methods currently supported by Verilog are onehot, while VHDL supports onehot and gray encoding methods.

Examples

- Verilog Example

  Specify the encoding method for the finite state machine as one hot.

```
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 Example

  Specify the encoding method for the finite state machine as Gray.

```
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

Description

Specify whether to insert an I/O buffer, and this attribute can only be specified in GSC.

Attribute Value

The table below lists the configurable attribute value and its corresponding function.

Table 5-14 Configurable Attribute Value and its Corresponding Function

Value	Function
0	Remove I/O buffer
1	Insert I/O buffer

Syntax

Table 5-15 Syntax Example

GSC Constraint Syntax	PORT "object" syn_insert_pad=value; GSC Example

Note!

The object can only be a port. This constraint only applies to input ports or output ports and does not apply to inout ports.

Example

- GSC Constraint Example

  Example 1: Insert the I/O buffer.

  PORT "out" syn_insert_pad = 1;

  Example 2: Remove the I/O buffer.

  PORT "out" syn_insert_pad = 0;

5.8 syn_keep

Description

Specify the net as a placeholder and retains it without optimization. This attribute can preserve nets that may be removed during the synthesis, preventing the merging of duplicate cells during optimization.

Attribute Value

Table 5-16 Attribute Value

Default	Global Attribute	Object
N/A	No	wire/signal, port, combinational logic

The table below lists the configurable attribute value and its corresponding function.

Table 5-17 Configurable Attribute Value and its Corresponding Function

Value	Function
0	Allows net optimization
1	Preserve net without optimization

Syntax

Table 5-18 Syntax Example

Verilog Constraint Syntax	object /* synthesis syn_keep = value */ ; Verilog Example
VHDL Constraint Syntax	attribute syn_keep : integer; attribute syn_keep of object : objectType is value; VHDL Example

Difference between syn_keep and syn_preserve

Table 5-19 Difference between syn_keep and syn_preserve

syn_keep	Only applies to nets and combinational logic. This attribute can preserve the specified net during synthesis without optimization.
syn_preserve	Only applies to registers. This attribute prevents the register from being optimized or absorbed.

Examples

- Verilog Constraint Example

  Specifies that the wires top1 and top2 are not be optimized.

```
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 Constraint Example

  Specifies that signal tmp0, tmp1 are not be optimized.

```
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

Description

Specify the maximum number of loops in the design. By default, this limit is set to 2000.

Syntax

Table 5-20 Syntax Example

GSC Constraint Syntax	GLOBAL syn_looplimit=value GSC Example

Example

GSC Constraint Example

GLOBAL syn_looplimit=3000

5.10 syn_maxfan

Description

Specify the maximum fanout. This attribute can be applied to modules, ports, and signals (wire/reg/signal), but it does not apply to clock signals, control enable signals, or reset/set signals.

Attribute Value

Table 5-21 Attribute Value

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

Syntax

Table 5-22 Syntax Example

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

Note!

The GSC constraint syntax does not support module.

Examples

- GSC Constraint Example

  Example 1: Specifies the maximum fanout of the instance as 10.

  INS "d" syn_maxfan=10;

  Example 2: Specifies the global maximum fanout as 100.

  GLOBAL syn_maxfan=100;

  Example 3: Specifies the maximum fanout of the instance as 10.

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

  Example 4: Specifies the maximum fanout of the net as 10.

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

- Verilog Constraint Example

  Example 1：Specifies the maximum fanout of all instances within the module, except for the clock (clk), as 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
```

Example 2：Specifies the maximum fanout of sel as 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 Constraint Example

Example 1：Specifies the maximum fanout of sel as 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

Description

Specify whether to generate a hierarchical netlist. By default, the synthesis tool generates a hierarchical netlist.

Attribute Value

Table 5-23 Attribute Value

Default	Global Attribute	Object
1	Yes	module/entity

The table below lists the configurable attribute value and its corresponding function.

Table 5-24 Configurable Attribute Value and its Corresponding Function

Value	Function
0	Flattens the hierarchical netlist for output.
1	Generates a hierarchical netlist.

Syntax

Table 5-25 Syntax Example

GSC Constraint Syntax	GLOBAL syn_netlist_hierarchy=value; GSC Example
Verilog Constraint Syntax	object /* synthesis syn_netlist_hierarchy = value */; Verilog Example
VHDL Constraint Syntax	attribute syn_netlist_hierarchy: integer; attribute syn_netlist_hierarchy of object : objectType is value; VHDL Example

Note!

The specified object can only be the top module/entity.

Examples

- GSC Constraint Example

```
GLOBAL syn_netlist_hierarchy=0;
```

- Verilog Constraint Example

```
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 Constraint Example

```
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

Description

Specify whether to optimize the register.

Attribute Value

Table 5-26 Attribute Value

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

The table below lists the configurable attribute value and its corresponding function.

Table 5-27 Configurable Attribute Value and its Corresponding Function

Value	Function
0	Allows to optimize register.
1	Preserves register without optimization

Syntax

Table 5-28 Syntax Example

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

Examples

- GSC Constraint Example

  Example 1: Specifies reg1 without optimization.

  INS "reg1" syn_preserve = 1;

  Example 2: Preserves all registers in the design.

  GLOBAL syn_preserve = 1;

- Verilog Constraint Example

  Example 1: Preserves all registers in the module.

```
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
```

  Example 2: Specifies reg1 without optimization.

```
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 Constraint Example

Example1: Preserves 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

Description

This attribute constraint is used to test and debug internal signals in the design by inserting probe points. The specified probe points will appear as ports in the top-level port list.

Attribute Value

Table 5-29 Attribute Value

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

The table below lists the configurable attribute value and its corresponding function.

Table 5-30 Configurable Attribute Value and its Corresponding Function

Value	Function
0	Probing is not allowed.
1	Inserts a probe point, automatically generating the probe port name based on the net name.
portName	Inserts a probe point with a specified name.

Syntax

Table 5-31 Syntax Example

Verilog Constraint Syntax	object /* synthesis syn_probe = "value" */; Verilog Example
VHDL Constraint Syntax	attribute syn_probe: string; attribute syn_probe of object: objectType is "value"; VHDL Example

note！

The value of "value" cannot be the same as the object name or the module's port name

Examples

- Verilog Constraint Example

After setting this attribute constraint, probe_alu_tmp will be listed in the top-level output port list.

```
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 Constraint Example

After setting this attribute constraint, probe_string will be listed in the top-level output port list.

```
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_ramstyle

Description

Specify the implementation for RAM. By default, the synthesis tool provides optimal inference based on the size of the RAM. For inference rules, see the sections of Guideline on BSRAM Coding and Guideline on SSRAM Coding in *SUG949, Gowin HDL Coding User Guide*.

Attribute Value

Table 5-32 Attribute Value

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

The table below lists the configurable attribute value and its corresponding function.

Table 5-33 Configurable Attribute Value and its Corresponding Function

Value	Function
registers	Specifies the implementation of RAM as registers.
block_ram	Specifies the implementation of RAM as BSRAM in the FPGA.
distributed_ram	Specifies the implementation of RAM as SSRAM in the FPGA.
rw_check	Inserts logic around the RAM to prevent undefined outputs during read and write operations on the same address, ensuring that the functional simulation of RTL matches the functional simulation of the netlist.
no_rw_check	Logic is not inserted around the RAM. By default, this function is enabled.

Syntax

Table 5-34 Syntax Example

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

Note!

If the attribute constraint distributed_ram is applied to a device that does not support SSRAM, the synthesis tool will issue a warning indicating that the attribute constraint is invalid.

Examples

- GSC Constraint Example

  Example 1: Specifies the implementation of the instance as BSRAM.

  INS " object " syn_ramstyle=block_ram

  Example 2: Specifies the implementation of all global RAM as SSRAM.

  GLOBAL syn_ramstyle=distributed_ram

- Verilog Constraint Example

  Example 1: Specifies the implementation of all RAMs in the module as 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
```

  Example 2: Specifies the implementation of the RAM as 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
```

Example 3: rw_check

```
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 Constraint Example

  Example 1: Specifies the implementation of the RAM as 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;
```

5.15 syn_romstyle

Description

Specify the implementation for ROM. By default, the synthesis tool provides optimal inference based on the size of the ROM. For inference rules, see the sections of Guideline on BSRAM Coding and Guideline on SSRAM Coding in *SUG949, Gowin HDL Coding User Guide*.

Attribute Value

Table 5-35 Attribute Value

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

The table below lists the configurable attribute value and its corresponding function.

Table 5-36 Configurable Attribute Value and its Corresponding Function

Value	Function
logic	Specifies the implementation of ROM as logic circuits.
block_rom	Specifies the implementation of ROM as BSRAM in the FPGA.

Value	Function
distributed_rom	Specifies the implementation of ROM as ROM16.

Syntax

Table 5-37 Syntax Example

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

Note!

If an attribute constraint distributed_rom is applied to a device that does not support ROM16, the synthesis tool will issue a warning indicating that the attribute constraint is invalid.

Examples

- GSC Constraint Example

  Example 1: Specifies the implementation of the instance as BSRAM.

  INS " object " syn_romstyle=block_rom

  Example 2: Specifies the implementation of all global ROM as ROM16.

  GLOBAL syn_romstyle=distributed_rom

- Verilog Constraint Example

  Example 1: Specifies the implementation of all ROMs in the module as 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 Constraint Example

Example 1: Specifies the implementation of all memories in the module as 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;
```

5.16 syn_srlstyle

Description

Specify the implementation for shift registers. By default, the synthesis tool provides optimal inference based on the size of the shift registers. For inference rules, see the sections of Guideline on BSRAM Coding and Guideline on SSRAM Coding in *SUG949, Gowin HDL Coding User Guide*.

Attribute Value

Table 5-38 Attribute Value

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

The table below lists the configurable attribute value and its corresponding function.

Table 5-39 Configurable Attribute Value and its Corresponding Function

value	Function
registers	Specifies the implementation of shift registers as registers.
block_ram	Specifies the implementation of shift registers as BSRAM in the FPGA.
distributed_ram	Specifies the implementation of shift registers as SSRAM in the FPGA.
bsram_sdp	Specifies the implementation of shift registers as BSRAM in semi-dual port mode.

Syntax

Table 5-40 Syntax Example

GSC Constraint Syntax	INS "object" syn_ srlstyle = value GLOBAL syn_ srlstyle = value GSC Example

Verilog Constraint Syntax	object /* synthesis syn_ srlstyle = "value" */; Verilog Example
VHDL Constraint Syntax	attribute syn_ srlstyle: string; attribute syn_ srlstyle of object : objectType is "value"; VHDL Example

Note!

If an attribute constraint distributed_ram is applied to a device that does not support SSRAM, the synthesis tool will issue a warning indicating that the attribute constraint is invalid.

Examples

- GSC Constraint Example

  Example 1: Specifies the implementation of the instance as BSRAM.

  INS "mem" syn_srlstyle=block_ram

  Example 2: Specifies the implementation of all global shift registers as SSRAM.

  GLOBAL syn_srlstyle=distributed_ram

- Verilog Constraint Example

  Example 1: Specifies the implementation of shift registers in the module as 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
```

  Example 2: Specifies the implementation of the instance as 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 Constraint Example

Example 1: Specifies the implementation of shift registers in the module as 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;
```

5.17 syn_tlvds_io/syn_elvds_io

Description

Specify the attribute of the differential I/O buffer mapping.

Attribute Value

Table 5-41 Attribute Value

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

The table below lists the configurable attribute value and its corresponding function.

Table 5-42 Configurable Attribute Value and its Corresponding Function

Value	Function
0	Disables automatic inference of differential I/O buffers.
1	Enables automatic inference of differential I/O buffers.

Syntax

Table 5-43 Syntax Example

GSC Constraint Syntax	PORT "object" syn_tlvds_io/syn_elvds_io = value GLOBAL syn_tlvds_io/syn_elvds_io = value

	GSC Example
Verilog Constraint Syntax	object /* synthesis syn_tlvds_io/syn_elvds_io = value */; Verilog Example
VHDL Constraint Syntax	attribute syn_tlvds_io/syn_elvds_io: integer; attribute syn_tlvds_io/syn_elvds_io of object : objectType is value; VHDL Example

Examples

- GSC Constraint Example

  Example 1: Specifies the implementation of the buffer as TLVDS.

  PORT "io" syn_tlvds_io =1;

  PORT "iob" syn_tlvds_io =1;

  Example 2: Specifies the implementation of all global buffers as ELVDS.

  GLOBAL syn_elvds_io =1;

- Verilog Constraint Example

  Example 1: Specifies the implementation of the buffer as 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 Constraint Example

  Example 1: Specifies the implementation of the buffer as 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;
```

5.18 translate_off/Translate_on

Description

The statements after translate_off will be skipped during the synthesis until translate_on appears. This is commonly used to automatically mask certain statements during synthesis. translate_off/translate_on must appear in pairs.

Syntax

Table 5-44 Syntax Example

Verilog Constraint Syntax	/* synthesis translate_off*/ Statements ignored during the synthesis. process./* synthesis translate_on*/ Verilog Example
VHDL Constraint Syntax	-- synthesis translate_off Statements ignored during the synthesis. -- synthesis translate_on VHDL Example

Examples

- Verilog Constraint Example

  The statement assign Nout =a*b between /*synthesis translate_off*/ and /*synthesis translate_on*/ will be ignored during the synthesis.

```
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 Constraint Example

```
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 Document

The report document is the statistical report generated after synthesis. The file name is *_syn.rpt.html (* is the name of specified output netlist vg file). It includes Synthesis Message, Synthesis Details, Resource, and Timing, etc.

6.1 Synthesis Message

Synthesis Message refers to the basic information of Synthesis. As shown in Figure 6-1. It mainly includes design file, GowinSynthesis version, configuration information, and created time, etc.

Figure 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 includes the information of top level module, synthesis process, total time and memory usage, as shown in Figure 6-2.

Figure 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 shows the resource information. It mainly includes resource and chip utilization statistics.

The resource usage summary table counts the number of I/O PORT, I/O BUF, REG, LUT, etc. The resource utilization summary table is used to estimate the resource utilization of CFU Logics, Register, BSRAM, DSP in the current device, as shown in Figure 6-3.

Figure 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 shows the timing statistics. It includes Clock Summary, Max. Frequency Summary, and Detail Timing Paths Information, etc.

Clock Summary mainly describes the clock signals of netlists, as shown in Figure 6-4. It shows a default clock with 100MHz and a period of 10ns, a rising edge at 0ns and a falling edge at 5ns.

Figure 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 mainly counts the time frequency of netlist file in order to measure whether the timing meets the requirements. As shown in Figure 6-5, the requested frequency is 100MHz and the actual frequency is 747.2MHz, and which meet the timing requirements. If the requirements can not be met, the specific timing path needs to be further checked.

Figure 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 shows the details of timing path. The default is 5 paths, and the unit is nanoseconds. Path Summary describes the key path, nodes and delays in the netlist file, as shown in Figure 6-6; Data Arrival Path and Data Require Path are key paths, and you can see from/to nodes and fanout in Figure 6-7. Path Statistics shows the path delay information, as shown in Figure 6-8.

Figure 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]

Figure 6-7 Connection Relation, Delay and Fanout Information

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

Figure 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
PROGRAMMING FOR THE FUTURE