GOWIN

PROGRAMMING FOR THE FUTURE

GowinSynthesis

User Guide

SUG550-1.7E, 05/09/2024

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.

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 1.9.9.03. 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

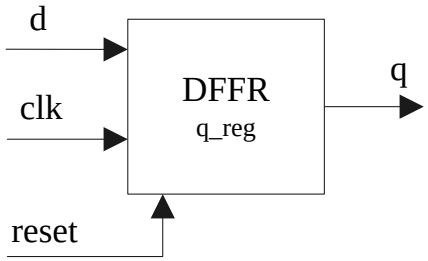
The only difference between CLK posedge flip-flop and CLK negedge flip-flop is that CLK triggers in different ways, so the following list is only the examples of CLK rising edge flip-flop.

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

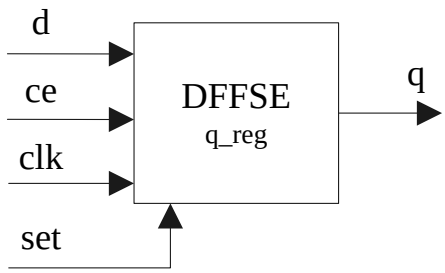


Example 2 can be synthesized as synchronous set flip-flop with clock enable.

```
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



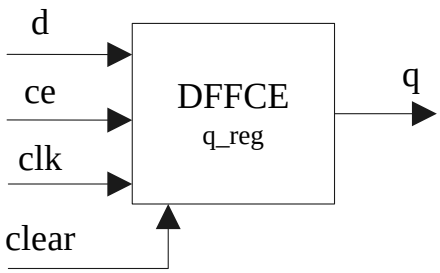
Example 3 can be synthesized as asynchronous reset flip-flop with clock enable.

```
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

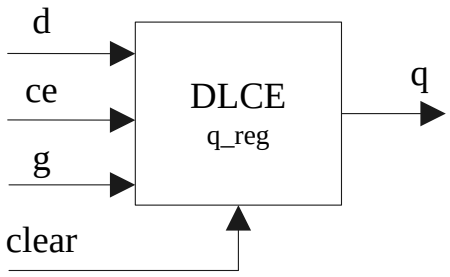


Example 4 can be synthesized as latch with reset and high level enable.

```
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



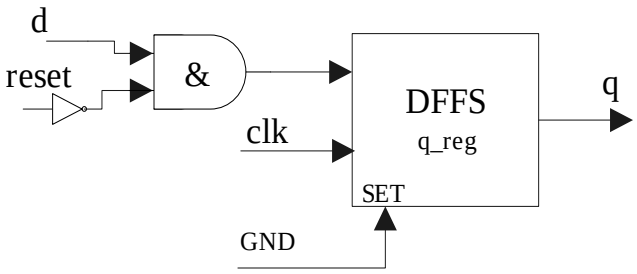
Specify the Initial Value of Flip-flop

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



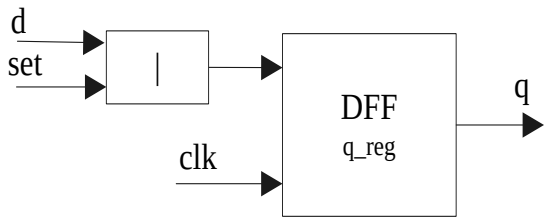
Example 6 is synchronous set clock flip-flop with an initial value of 1, but its initial value is 0 in RTL; and this synchronous set clock flip-flop will

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



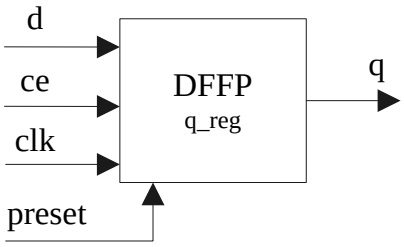
Example 7 is an asynchronous set flip-flop with an initial value of 1.

```
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

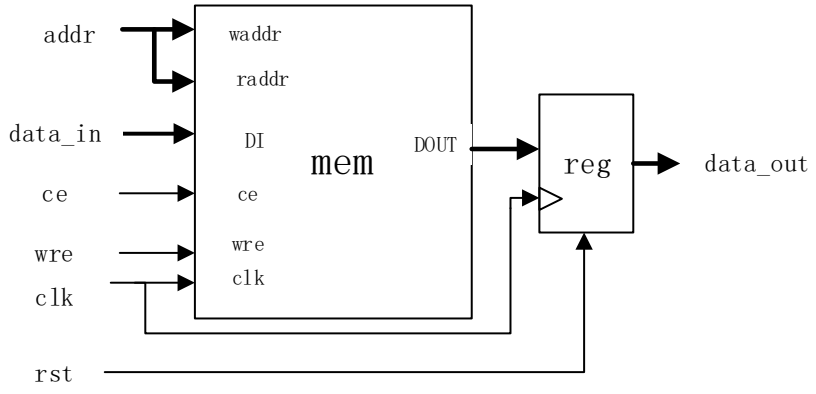
According to the different features of RAM, examples are as follows:

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



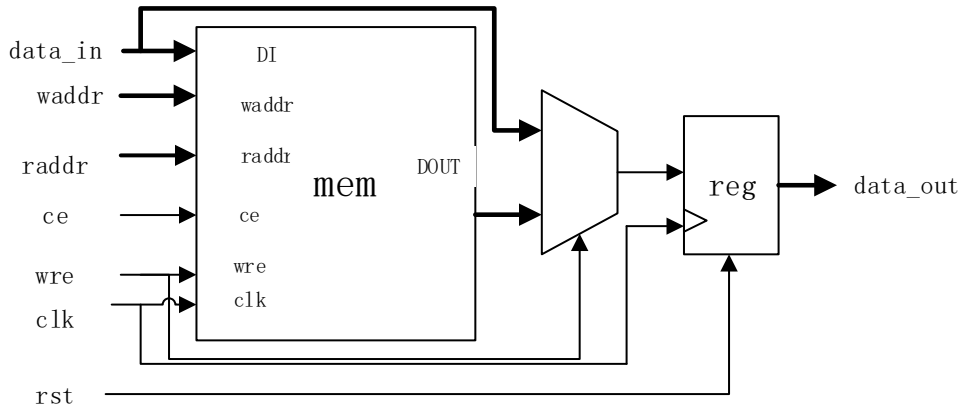
Example 2 is a memory with one write port, one read port and the

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

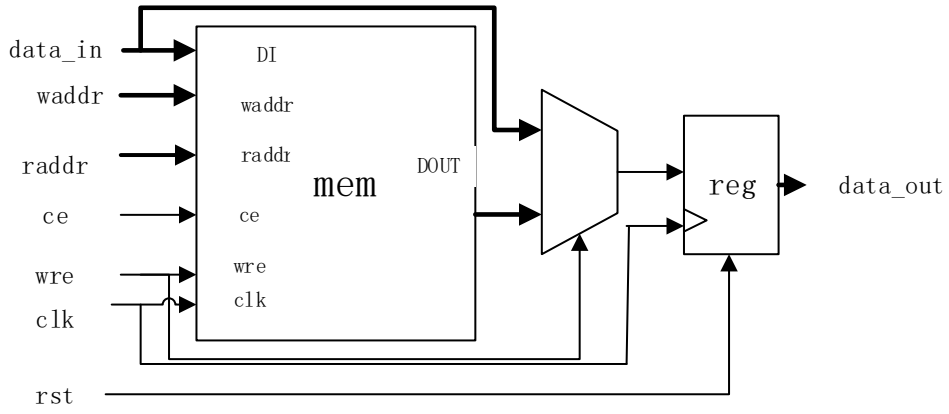


Example 3 is a memory with one write port, one read port and the same read and write address. When wre is 1, input data is written to memory, which can be synthesized to single-port BSRAM in read-before-write mode.

```
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



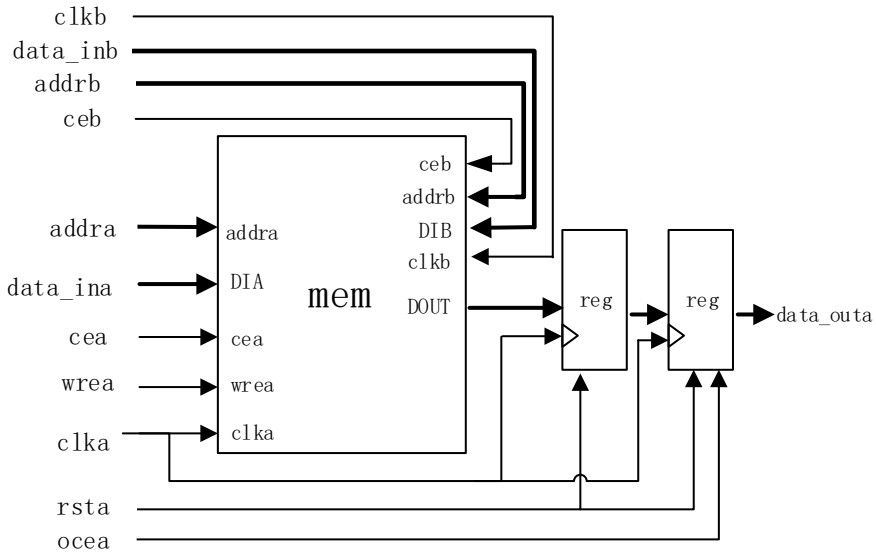
Example 4 is a memory with two write ports and one read port. One of the two write ports has a wre signal and the other does not. The read port absorbs asynchronous reset register. This example can be synthesized to asynchronous reset dual-port BSRAM with A port in normal write mode and B port in read-before-write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

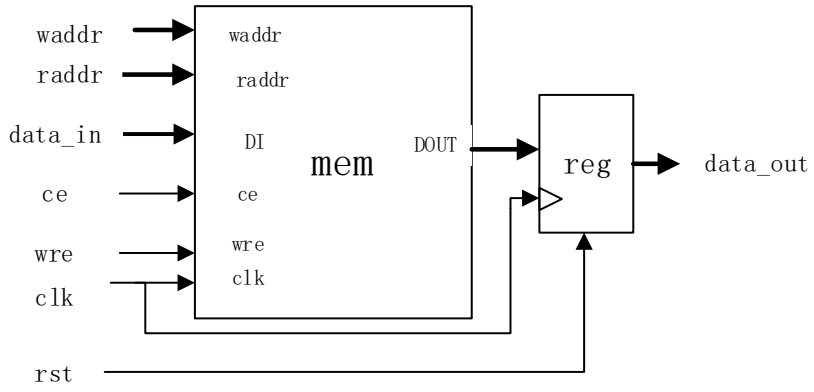


Example 5 is a memory with one read port and one write port and different read and write addresses, which can be synthesized to semi-dual-port BSRAM in normal write mode or in bypass read mode.

```
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



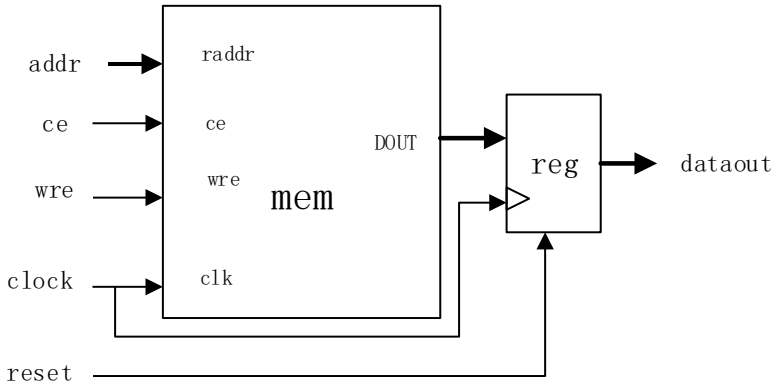
Example 6 is a memory with one read port and an initial value, which can be synthesized to asynchronous set read-only memory in bypass read mode.

```
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



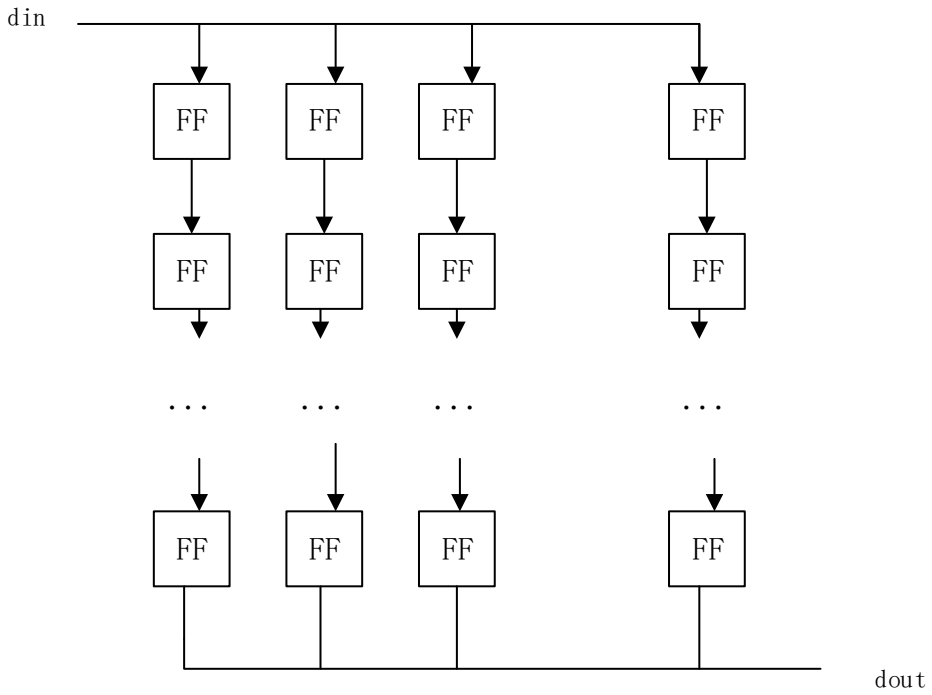
Example 7 is a memory with shift-register mode, which can be synthesized to simple-dual-port BSRAM in normal mode.

```
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.

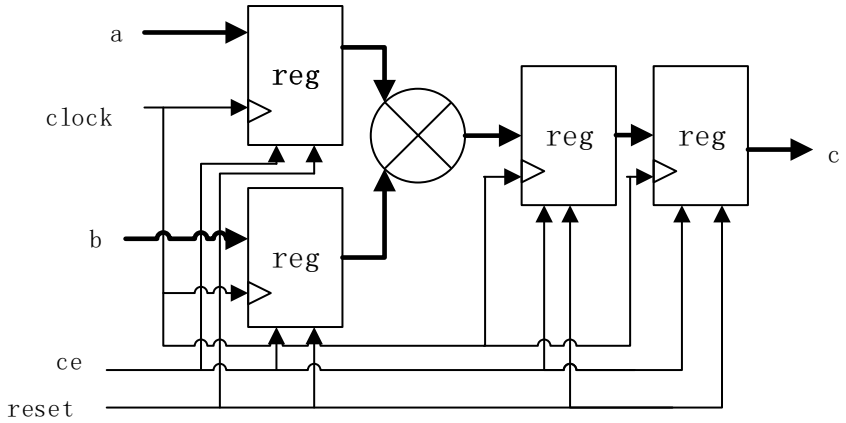
```
module top(a,b,c,clock,reset,ce).
parameter a_width = 18.
parameter b_width = 18.
parameter c_width = 36.
input signed [a_width-1:0] a.
input signed [b_width-1:0] b.
```

```
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.

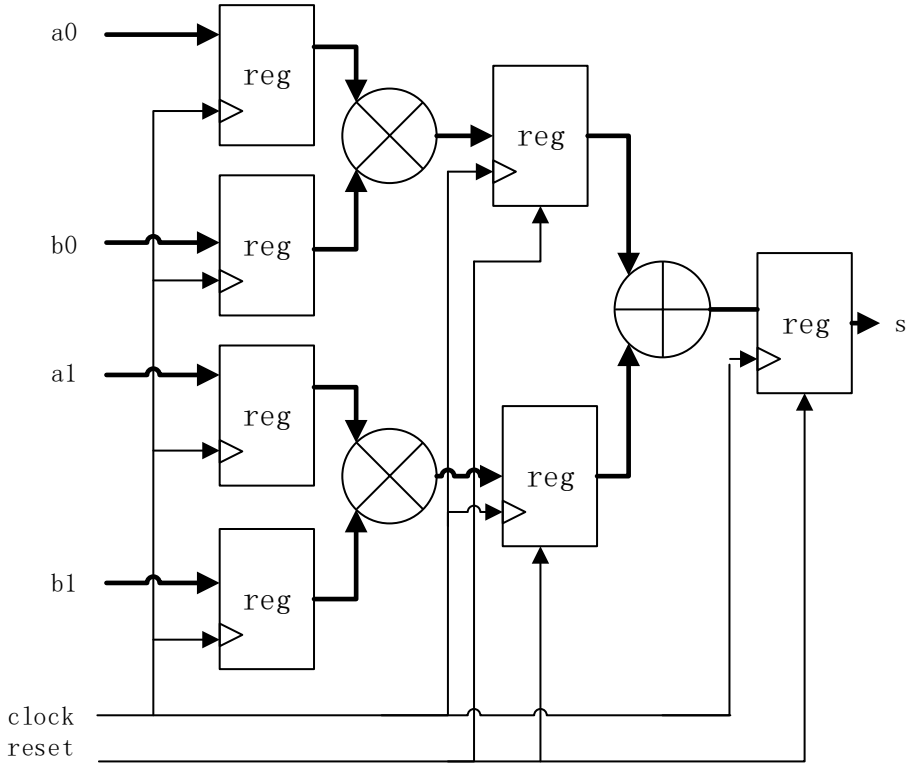
```
module top(a0, a1, b0, b1, s, reset, clock, ce).
parameter a0_width=18.
parameter a1_width=18.
parameter b0_width=18.
parameter b1_width=18.
parameter s_width=37.
input unsigned [a0_width-1:0] a0.
input unsigned [a1_width-1:0] a1.
input unsigned [b0_width-1:0] b0.
input unsigned [b1_width-1:0] b1.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

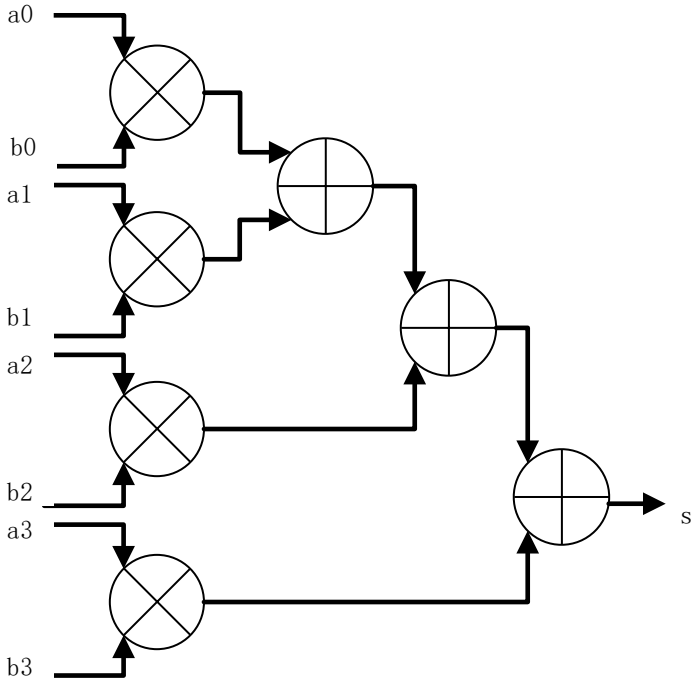
output unsigned [s_width-1:0] s.

*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.

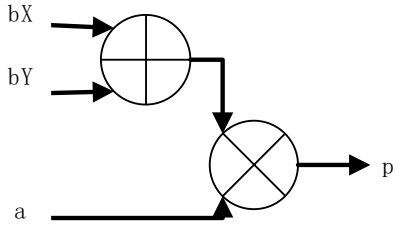
assign b = bX + bY.

*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

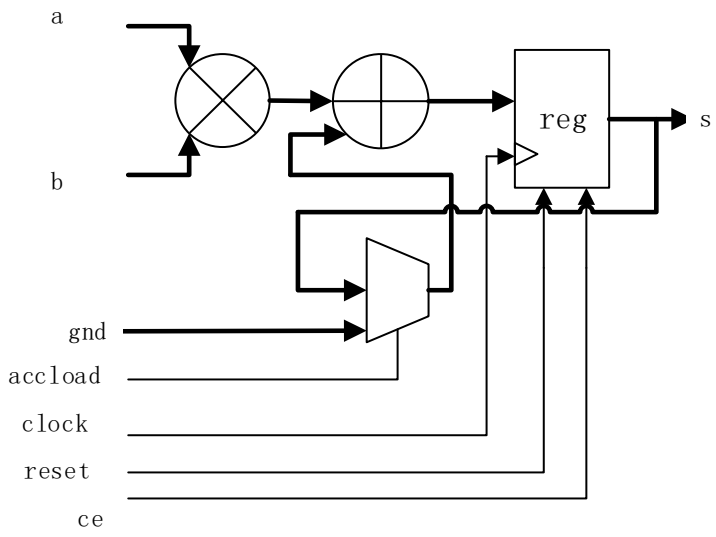


Example 5 can be synthesized to a multiplier accumulator with sign-bit 0, and the output register is s.

```
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 for 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 separate by breaks, and a semicolon must be added at the end of the statement.

Constraints in RTL File

Constraints in the RTL file must be added in the definition statement of the constraint object before the semicolon. If the constraint setting_value in the statement is a string, then double quotes should be added before and after the setting_value. If the setting_value is a number, then no double quotes are required.

GSC

GSC constraints include Instance constraints, Net constraints, Port constraints and global objects constraints. In order to distinguish the types, there are different syntaxes in writing. The constraint object must be enclosed by double quotation marks. Attribute name and the setting_value need not be identified by double quotation marks or other sligns, and there can be spaces before and after the equal sign. GSC constraints support notes with "//". The examples are as follows:

INS "object" attributeName=setting_value;

NET "object" attributeName=setting_value;

PORT "object" attributeName=setting_value;

GLOBAL attributeName=setting_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 name of instance does not contain parenthesis. In other

words, don't write temp[15:0] when bus, just write temp instead.

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.

The name of the object in the constraint must match the one in the netlist. There can be no spaces in the name. Wildcards are supported in the name of the object. Use "/" to distinguish the hierarchy of names. Add w before object to distinguish when using wildcards, such as w "object".

The setting_value may be the value specified directly by the user, inherited from the super-structure, or the default. The priority of values is direct value in GSC > direct value in RTL > inherited value in GSC > inherited value in RTL > default value. When there are multiple inherited values, take the value closest to the specified name (lowest level). For example, check the MULT_STYLE attribute of A/D/C/mult1 ("/" indicates the hierarchy between module names), which has direct value of DSP. Check MULT_STYLE attribute of "A/D/C", which has no direct value, and the MULT_STYLE attribute can be inherited, so find and inherit the attribute value logic of "A/D". Check MULT_STYLE of "A/D/C/mult1", which has both the inherited value and the direct value. The direct value DSP is finally taken because of the priority.

5.1 black_box_pad_pin

Description

Specify that the IO pads of the black box are visible to the external environment. This attribute only works for the black box IO pads.

This attribute can only be specified in the RTL file.

Syntax

Verilog Constraint Syntax

Verilog object /* synthesis black_box_pad_pin=portList */;

VHDL Constraint Syntax

attribute black_box_pad_pin : string;

attribute black_box_pad_pin of object: objectType is portList;

Note!

- object: It can be module or component defined by a black box.
- objectType: The type of object, such as component.
- The PortList enclosed in double quotes is a space-less, comma-separated list of the ports in 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;
……
black_box_add U2 (in1, in2, out2,D,E);
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(width1 downto 0);
    clk : in std_logic;
    q : out std_logic_vector (width1 downto 0)
);
```

end component;

attribute black_box_pad_pin : string;

attribute black_box_pad_pin of test : component is "q[4:0]";

begin

test123 : test generic map (width) port map (in1,in2,clk,q);

end top1_arch;

5.2 full_case

Description

full_case is only used in Verilog design. When this attribute is added after a case, casex or casez statement, it means that all possible values are given and no redundant hardware is required to preserve the signal values.

This attribute can only be specified in RTL files and only support Verilog.

Syntax

RTL Constraint Syntax

verilog case /* synthesis full_case*/

Example

RTL Constraint Example

Example 1 specifies that part of the circuit no longer requires redundant hardware to preserve signal values

module top(…);

……

always @(select or a or b or c or d)

begin

casez(select) / synthesis full_case*/*

4'b???1: out=a;

……

4'b1???: out=d;

endcase

end

endmodule

5.3 parallel_case

The parallel_case is an instruction, which forces to use a parallel-multiplexed structure instead of a priority-encoded structure.

This instruction can only be specified in the Verilog file.

Description

The case statement is defined in priority order by default, executing only the first statement that matches the selected value. Priority coding allows to have input signals at several inputs at the same time, and to code only the highest priority of the several signals input at the same time in the priority order.

If the bus selected is driven from outside of the current module and the current module does not have information about the legal selected value, the software must create a logical chain of disable in order to match the tag statement to disable all subsequent statements.

However, if the legal value of the selection is known, the parallel_case instruction can be used to eliminate the additional priority coding logic.

Syntax

Verilog Constraint Syntax

object /* synthesis parallel_case */;

Note!

- global support: No
- object: case, casex, and casez statements.
- setting_value: No value.

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 = 'bx;
```

endcase

end

endmodule

5.4 syn_black_box

Description

Specify a module or component as a black box. In synthesis, a black box module defines only its interface, and its content can not be accessed or optimized. The module is treated as a black box regardless of whether it is empty or not.

This attribute can only be specified in the RTL file.

Syntax

Verilog Constraint Syntax

object /* synthesis syn_black_box */;

VHDLConstraint Syntax

attribute syn_black_box: boolean;

attribute syn_black_box of object : objectType is true;

Note!

- object: It can only be sub module/entity.
- objectType: The type of object, such as component.

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);

……

begin

out1 <= in1 + in2;

end

```
endmodule


module black_box_add(A, B, C)/* synthesis syn_black_box */;
……
assign C = A + B;
endmodule
```

Before the attribute used, the content of the module black_box_add is visible. After the attribute used, the content of the module black_box_add is not visible and becomes a black box.

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
……
    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;
```

5.5 syn_dspstyle

Description

The specified multiplier is implemented by a dedicated DSP hardware module or logic circuit, which can be applied to both specific module/entity instance and the global. This attribute can be specified in GSC and RTL files.

Syntax

GSC Constraint Syntax

INS "object" syn_dspstyle =setting_value;

GLOBAL syn_dspstyle =setting_value;

Verilog Constraint Syntax

Verilog object /* synthesis syn_dspstyle ="setting_value" */;

VHDL Constranit Syntax

attribute syn_dspstyle:string;

attribute syn_dspstyle of object:objectType is "setting_value";

Note!

- object: It can be wire, register, module/entity name or module/entity instance name.
- objectType: The type of object, such as signal.
- setting_value: Multiplier implementation, and currently DSP and logic are supported.
- setting_value is logic, inferring object to logic circuit.
- setting_value is DSP, inferring object to DSP, which is the default.

Examples

GSC Constraint Example

Example 1 specifies logic to implement instance

INS ‘temp" syn_dspstyle=logic;

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

Example 2 specifies logic to implement global multipliers

GLOBAL syn_dspstyle=logic;

Verilog Constraint Example

Example 1 specifies logic to implement all multipliers in mult module.

module mult(…) / synthesis syn_dspstyle = "logic" */;*

……

wire [15:0] temp;

*assign temp = a*b;*

……

Endmodule

Example 2 specifies logic to implement the multiplier temp.

```
module mult(…) ;
……
wire [15:0] temp/* synthesis syn_dspstyle = "logic" */ ;
assign temp = a*b;
……
endmodule
```

VHDLConstraint Example

Example 1 specifies logic to implement the multiplier result.

```
entity Mult is
port(
    ……
      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
    ……
      result <= x1 * y1;
end Behavior;
```

5.6 syn_encoding

Description

Specify the encoding mode of state machine, which can only be applied to specific objects.

This attribute can only be specified in RTL files.

Syntax

Verilog Syntax

```
verilog object /* synthesis syn_encoding = "setting_value" */;
```

VHDLConstraints Syntax

```
attribute syn_encoding : string;
attribute syn_encoding of object : objectType is "setting_value";
```

Note!

- object: It can be wire, register, module/entity name or module/entity instance name.
- objectType: The type of object, which is typically "type" here.
- setting_value: State machine encoding mode; verilog currently supports one hot; VHDL currently supports one hot and gray.

Examples

Verilog Example

This example specifies one hot code for the state machine

module test (…);

reg [2:0] ps / synthesis syn_encoding="onehot " */;*

……

endmodule

VHDL Example

This example specifies one-hot code for the state machine.

ENTITY fsm IS

……

END fsm;

ARCHITECTURE behaviour OF fsm IS

TYPE state_type IS (s0,s1,s2,s3);

SIGNAL present_state,next_state : state_type;

attribute syn_encoding:string;

attribute syn_encoding of state_type:type is "onehot";

BEGIN

……

END behaviour;

5.7 syn_insert_pad

Description

Specify whether to insert an I/O buffer. Insert the I/O buffer when the attribute value is 1

This attribute can only be specified in GSC files.

Syntax

GSC Constraints Syntax

PORT "object" syn_insert_pad=setting_value;

Note!

- setting_value: 0 or 1. Remove I/O buffer at 0. Insert I/O buffer at 1.
- object: It can only be port. This constraint only applies to input port or output port, not inout port.

Examples

GSC Examples

Example 1 specifies the inserted I/O buffer

PORT "out" syn_insert_pad=1;

Example 2 specifies the removed I/O buffer

PORT "out" syn_insert_pad=0;

5.8 syn_keep

Description

Specify wire as a placeholder and preserve it without optimization; to preserve "reg," please use "syn_preserve.

This attribute can only be specified in RTL files.

Syntax

Verilog Constraints Syntax

Verilog object /* synthesis syn_keep= setting_value */;

VHDL Constraints Syntax

attribute syn_keep : integer;

attribute syn_keep of object : objectType is 1;

Note!

- object: It can only be wire, port and combination logic.
- ObjectType: The type of object, such as signal.
- setting_value: The setting_value can only be 0 or 1. When it is 1, the net is preserved without optimization.

Examples

Verilog Constraint Example

This example specifies mywire and leaves it unoptimized.

```
module test (…);
……
wire mywire /* synthesis syn_keep=1 */;
……
endmodule
```

VHDL Constraint Example

This example specifies tmp0 and leaves it unoptimized.

```
entity mux2_1 is
    port(
        ……
    );
```

```
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;
……
end Behavioral;
```

5.9 syn_looplimit

Description

Specify the limited value of loop iteration. The default is 2000 times. If it does not specify the time, exceeding the default, there will be an error in synthesis.

This attribute can only be set in GSC.

Syntax

GSC Constraints Syntax

GLOBAL syn_looplimit=setting_value

Note!

setting_value: It can only be number, meaning the upper limited times in loop iteration.

Example

GSC Constraints Example

GLOBAL syn_looplimit=3000

5.10 syn_maxfan

Description

Specify max. fanout, which can be applied to both specific objects and the global.

This attribute can be specified in GSC and RTL files.

Syntax

GSC Constraints Syntax

INS "object" syn_maxfan=setting_value;

NET "object" syn_maxfan=setting_value;

GLOBAL syn_maxfan=setting_value;

Verilog Constraints Syntax

Verilog object /* synthesis syn_maxfan = setting_value */;

VHDL Constraints Syntax

attribute syn_maxfan : integer;

attribute syn_maxfan of object : objectType is setting_value;

Note!

- object: It can be wire, register, input, output, module/entity name, module/entity instance name; it does not work for the inpin related with CLK, CE, LSR.
- objectType: The type of object, such as signal.
- setting_value: Integer greater than 0.

Examples

GSC Examples

Example 1 specifies that the max.fanout of instance is 10.

INS "d" syn_maxfan=10;

Example 2 specifies that the max.fanout of global is 100.

GLOBAL syn_maxfan=100;

Example 3 specifies that the max.fanout of instance is 10.

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

Example 4 specifies that the max.fanout of net is 10.

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

Verilog Examples

Example 1 specifies a maximum fanout value of 3 for all instances in the module except CLK.

module test (…) / synthesis syn_maxfan = 3*/;*

……

endmodule

Example 2 specifies that the max.fanout of instance is 3.

module test (…);

reg [7:0] d / synthesis syn_maxfan = 3*/;*

……

endmodule

VHDL Example

entity test is

……

end test;

architecture rtl of test is

signal d : std_logic;

attribute syn_maxfan : integer;

attribute syn_maxfan of d : signal is 5;

……

end rtl;

5.11 syn_netlist_hierarchy

Description

Specify whether to generate hierarchy netlists. The default 1 indicates the generation of hierarchy netlists. When it is set to 0, the hierarchy netlists will be flattened for output.

This attribute can be specified in GSC and RTL files.

Syntax

GSC Constraint Syntax

GLOBAL syn_netlist_hierarchy=setting_value;

Verilog Constraint Syntax

Verilog object /* synthesis syn_netlist_hierarchy=setting_value */;

VHDL Constraint Syntax

attribute syn_netlist_hierarchy: integer;

attribute syn_netlist_hierarchy of object : objectType is setting_value;

Note!

- object: It can only be the top module/entity.
- objectType: The type of object, such as entity.
- setting_value: The setting_value is 0 or 1. If it is 1, hierarchy is allowed to be generated. If it is 0, the hierarchy netlists will be flattened for output.

Examples

GSC Example

GLOBAL syn_netlist_hierarchy=0;

Verilog Example

module rp_top (…) / synthesis syn_netlist_hierarchy=1 */;*

……

endmodule

VHDL Example

```
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;

5.12 syn_noprune

Description

Ensure that whether the outputs of module/entity instances, primitive instances, or black boxes (including primitives) are optimized or not when left floating. This can be applied to specific objects or globally.

This attribute can only be specified in RTL file.

Syntax

Verilog Constraints Syntax

Verilog object /* synthesis syn_noprune = setting_value */;

VHDL Constraints Syntax

attribute syn_noprune : integer;

attribute syn_noprune of object: objectType is 1;

Note!

- object: It can be module/entity instance name, primitive instance name or black box.
- objectType: The type of object, such as signal.
- setting_value: It can only be 0 or 1. If it is 1, retain instance and black box. If it is 0, optimize the corresponding instance and black box as needed.

Example

Verilog Constraint Example

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

……

*noprune_bb u1(out1,in1)/*synthesis syn_noprune=1*/;*

……

endmodule

module noprube_bb(din,dout);

input din;

output dout;

endmodule

VHDL Constraint Example

library ieee;

use ieee.std_logic_1164.all;

entity top is

……

end entity top;

architecture arch of top is

component noprune_bb

```
port(
din : in std_logic;
dout : out std_logic);
end component noprune_bb;
signal o1_noprunereg : std_logic;
signal o2_reg : std_logic;
attribute syn_noprune : integer;
attribute syn_noprune of U1: label is 1;
attribute syn_noprune of o1_noprunereg : signal is 1;
……
end architecture arch;
```

5.13 syn_preserve

Description

Specify register or whether to optimize the register logic, which can be applied to both specific objects and the global.

This attribute can be specified in RTL and GSC files.

Syntax

GSC Constraint Syntax

INS "object" syn_preserve=setting_value;

GLOBAL syn_preserve=setting_value;

Verilog Constraint Syntax

Verilog object /* synthesis syn_preserve = setting_value */;

VHDL Constraint Syntax

attribute syn_preserve : integer;

attribute syn_preserve of object : objectType is setting_value;

Note!

- Object: It can be the name of register, module/entity, module/entity instance.
- objectType: The type of object, such as signal.
- setting_value: 0 or 1. When it is 1, the corresponding register is preserved. When it is 0, the corresponding register is optimized as needed.

Examples

GSC Examples

Example 1 specifies reg1and leaves it unoptimized.

INS "reg1" syn_ preserve =1;

Example 2 specifies that all registers in the design are preserved.

GLOBAL syn_ preserve =1;

Verilog Examples

Example 1 specifies that all registers in the module are preserved.

```
module test (…) /* synthesis syn_preserve = 1 */;
……
endmodule
```

Example 2 specifies reg1and leaves it unoptimized.

```
module test (…).
……
reg reg1/* synthesis syn_preserve = 1 */;
……
endmodule
```

VHDL Example

Example 1 specifies to preserve reg 1.

```
entity syn_test is
    port (……
    );
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
……
end behave;
```

5.14 syn_probe

Description

This attribute tests and debugs the internal signals in the design by inserting probe points. The specified probe points appear as ports in the top-level port list.

This attribute can only be specified in RTL files.

Syntax

Verilog Constraints Syntax

Verilog object /* synthesis syn_probe = setting_value */;

VHDL Constraints Syntax

attribute syn_probe: string;

attribute syn_probe of object: objectType is " setting_value ";

Note!

- object: It can only be wire or register.
- objectType: The type of object, such as signal.
- setting_value is 1: Insert the probe point and automatically get the probe port name according to the net name.
- setting_value is 0: Probe is not allowed.
- setting_value is a string: Insert a specified probe point name. When the name specified by setting_value is bus, the number is automatically added after the inserted name.
- GowinSyn does not support the setting_value with the same value as the object name or the port name of the module.

Examples

Verilog Constraints Example

When probe_tmp is set, probe_tmp is listed in output port list of the top level.

module test (…);

……

reg [7:0] probe_tmp / synthesis syn_probe=1*/;*

……

endmodule

VHDL Constraints Example

entity halfadd is

port(……);

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";

……

End;

5.15 syn_ramstyle

Description

Specify the implementation of memory, which can be applied to both specific instances and the global.

This attribute can be specified in GSC and RTL files.

Syntax

GSC Constraint Syntax

INS "object" syn_ramstyle =setting_value;

GLOBAL syn_ramstyle =setting_value;

Verilog Constraint Syntax

Verilog object /* synthesis syn_ramstyle = "setting_value" */;

VHDL Constraint Syntax

attribute syn_ramstyle:string;

attribute syn_ramstyle of object : objectType is " setting_value";

Note!

- object: It can be module/entity name, module entity instance name, or register.
- objectType: The type of object, such as signal.
- setting_value: The implementation of memory currently supports block_ram, distributed_ram, registers, rw_check, no_rw_check.
- setting_value is registers: Maps inferred RAM to registers (flip-flop and logic circuit) rather than dedicated RAM resources.
- setting_value is block_ram: Maps inferred RAM to dedicated memory, which uses FPGA dedicated memory resources.

Examples

GSC Constraint Examples

Example 1 specifies BSRAM to implement instance.

INS "mem" syn_ramstyle=block_ram;

Example 2 specifies SSRAM to implement global memory.

GLOBAL syn_ramstyle=distributed_ram;

Verilog Constraint Examples

Example 1 specifies block_ram to implement memory in module and there are no read or write check.

module test (…) / synthesis syn_ramstyle = "block_ram" */*

…

endmodule

Example 2 specifies BSRAM to implement instance.

module test (…);

……

*reg [DATA_W - 1 : 0] mem [(2**ADDR_W) - 1 : 0] /* synthesis syn_ramstyle = "block_ram" */;*

……

endmodule

VHDL Constraint Example

Example 1 specifies BSRAM to implement memory.

entity ram is

GENERIC(bits:INTEGER:=8;

```
    words:INTEGER:=256);
PORT(……);
end ram;

ARCHITECTURE arch of ram IS
TYPE vector_array IS ARRAY(0 TO words-1) OF
STD_LOGIC_VECTOR(bits-1 DOWNTO 0);
SIGNAL memory:vector_array ;
    attribute syn_ramstyle:string;
    attribute syn_ramstyle of memory : signal is " block_ram";
BEGIN
……
end arch;
```

5.16 syn_romstyle

Description

Specify the implementation of read-only memory, which can be applied to both specific objects and the global.

This attribute can be specified in GSC and RTL files.

Syntax

GSC Constraint Syntax

INS "object" syn_romstyle =setting_value;

GLOBAL syn_romstyle =setting_value;

Verilog Constraints Syntax

Verilog object /* synthesis syn_romstyle = "setting_value" */;

VHDL Constraint Syntax

attribute syn_romstyle:string;

attribute syn_romstyle of object : objectType is " setting_value";

Note!

- object: It can be module/entity name, module entity instance name, or register.
- objectType: The type of object, such as signal.
- setting_value: The implementation of read-only memory, and it currently supports block_rom, distributed_rom, logic.

Examples

GSC Constraint Examples

Example 1 specifies BSRAM to implement instance.

INS "mem" syn_romstyle=block_rom;

Example 2 specifies SSRAM to implement global memory.

GLOBAL syn_romstyle=distributed_rom;

Verilog Constraint Example

This example specifies SSRAM to implement memory in module.

*module rom16_test(…)/*synthesis syn_romstyle="distributed_rom"*/;*

……

endmodule

VHDL Constraint Example

This example specifies SSRAM to implement all memories in the module.

ENTITY rom is

……

end rom;

ARCHITECTURE rom OF rom IS

signal data_out :STD_LOGIC_VECTOR(bits-1 DOWNTO 0);

attribute syn_romstyle:string;

attribute syn_romstyle of data_out : signal is "*block_rom*";

……

END rom;

5.17 syn_srlstyle

Description

Specify the implementation of shift registers, either acting on a specific object or the global. The shift register can be implemented by BSRAM, SSRAM, registers, and bsram_sdp. The number of registers in the shift register determines which implementation method to use by default. The default value can be changed by using syn_srlstyle.

This attribute can be specified in GSC and RTL files.

Syntax

GSC Constraint Syntax

NS "object" syn_srlstyle =setting_value;

GLOBAL syn_srlstyle =setting_value;

Verilog Constraint Syntax

Verilog object /* synthesis syn_srlstyle = "setting_value" */;

VHDL Constraint Syntax

attribute syn_srlstyle:string;

attribute syn_srlstyle of object : objectType is " setting_value";

Note!

- object: It can be module/entity, module/entity instance or register. Module/entity is not supported in the GSC syntax.
- objectType: The type of object, such as signal.
- setting_value: The memory implementation, and it supports block_ram, distributed_ram, registers, and bsram_sdp currently. When specified as bsram_sdp, shift registers are not inferred as SP.

Example

GSC Constraint Examples

Example 1 specifies that instance implementation is BSRAM.

INS "mem" syn_srlstyle=block_ram;

Example 2 specifies that the global memory implementation is SSRAM.

GLOBAL syn_srlstyle=distributed_ram;

Verilog Constraint Example

Example 1 specifies that the register implementation in the module is block_ram.

module test (…) / synthesis syn_srlstyle = "block_ram" */;*

……

endmodule

Example 2 specifies that instance implementation is BSRAM.

module test (…);

……

reg [SRL_WIDTH-1:0] regBank[SRL_DEPTH-1:0]/ synthesis syn_srlstyle = "block_ram" */;*

……

endmodule

VHDL Constraint Example

This example specifies that the register implementation in the module is register.

```
entity ram is
GENERIC(bits:INTEGER:=8;
    words:INTEGER:=256);
PORT(……);
attribute syn_srlstyle:string;
attribute syn_srlstyle of shiftreg : entity is "registers";
end ram;

ARCHITECTURE arch of ram IS
```

……

end ram;

5.18 syn_tlvds_io/syn_elvds_io

Description

Specify the attribute of the differential I/O buffer mapping, which can be applied to both specific objects and the global.

This attribute can be specified in GSC and RTL files.

Syntax

GSC Constraint Syntax

PORT "object" syn_tlvds_io =setting_value;

GLOBAL syn_tlvds_io =setting_value;

PORT "object" syn_elvds_io =setting_value;

GLOBAL syn_elvds_io =setting_value;

Verilog Constraint Syntax

Verilog object /* synthesis syn_tlvds_io = setting_value */;

VHDL Constraint Syntax

attribute syn_tlvds_io: integer;

attribute syn_tlvds_io of object: objectType is setting_value;

Note!

- object: It can be module/entity name or port name.
- objectType: The type of object, such as signal.
- setting_value: 0 or 1.

Examples

GSC Constraint Examples

Example 1 specifies that the buffer implementation is TLVDS.

PORT "io" syn_tlvds_io =1;

PORT "iob" syn_tlvds_io =1;

Example 2 specifies that the implementation of all buffers in the global is TLVDS.

GLOBAL syn_tlvds_io =1;

Verilog Constraint Example

module elvds_iobuf(io,iob…);

*inout io/*synthesis syn_elvds_io=1*/;*

*inout iob/*synthesis syn_elvds_io=1*/;*

.....

endmodule

VHDL Constraint Example

```
entity test is
port (in1_p : in std_logic;
in1_n : in std_logic;
clk : in std_logic;
out1 : out std_logic;
out2 : out std_logic);
attribute syn_tlvds_io: integer;
attribute syn_tlvds_io of in1_p,in1_n,out1,out2: signal is 1;
end test;
architecture arch of test is
……
end arch
```

5.19 translate_off/Translate_on

Description

translate_off /translate_on must occur in pairs, and statements after translate_off are skipped during the synthesis until translate_on occurs, which is often used to automatically mask some statements during synthesis.

This attribute can only be specified in RTL files.

Syntax

Verilog Constraint Syntax

/* synthesis translate_off*/;

Statements ignored in the synthesis.

/* synthesis translate_on*/;

VHDL Constraint Syntax

-- synthesis translate_off

Statements ignored in the synthesis

-- synthesis translate_on

Examples

Verilog Constraint Example

The assign Nout =a*b between /*synthesis translate_off*/ and /*synthesis translate_on*/ is ignored in the synthesis in example 1.

```
module test (…);
……
/*synthesis translate_off*/
```

```
assign my_ignore=a*b;
/* synthesis translate_on*/
……
endmodule
```

VHDL Constraint Example

```
entity top is
port (
……
);
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:

Clock Name	Type	Period	Frequency(MHz)	Rise	Fall	Source	Master	Object
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