GOWIN

PROGRAMMING FOR FUTURE

GowinSynthesis

User Guide

SUG550-1.0E, 08/02/2019

Disclaimer

Revision History

Date	Version	Description
08/02/2019	1.0E	Initial version published.

Contents

List of Figures

List of Tables

1 About This Guide

1.1 Purpose

It mainly describes the function and operation of GowinSynthesis and aims to help users quickly learn this software to guide user design and improve design efficiency. The software screenshots and the supported products listed in this manual are based on Gowin YunYuan Software 1.9.2Beta. 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 Supported Products

The information presented in this guide applies to the following products:

- GW1N series of FPGA products: GW1N-1, GW1N-1S, GW1N-2, GW1N-2B, GW1N-4, GW1N-4B, GW1N-6, GW1N-9
- GW1NR series of FPGA products: GW1NR-4, GW1NR-4B, GW1NR-9
- GW1NS series FPGA products: GW1NS-2
- GW2A series FPGA products: GW2A-55, GW2A-18
- GW2AR series FPGA products: GW2AR-18
- GW1NZ series FPGA products: GW1NZ-1
- GW1NSR series of FPGA products: GW1NSR-2

1.3 Related Documents

The user guides are available on the GOWINSEMI Website. You can find the related documents at www.gowinsemi.com:

- DS100, GW1N series of FPGA Products Data Sheet
- DS117, GW1NR series of FPGA Products Data Sheet
- DS821, GW1NS series of FPGA Products Data Sheet
- DS102, GW2A series of FPGA Products Data Sheet
- DS226, GW2AR series of FPGA Products Data Sheet

- DS841, GW1NZ series of FPGA Products Data Sheet
- DS861, GW1NSR series of FPGA Products Data Sheet

1.4 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
FPGA	Field Programmable Gate Array
SSRAM	Shadow SRAM
B-SRAM	Block Static Random Access Memory
DSP	Digital Signal Processing
FSM	Finite State Machine
GSC	Gowin Synthesis Constraint

1.5 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

+Tel: +86 755 8262 0391

2 Overview

2.1 Introduction

It is Gowin RTL design synthesis tool GowinSynthesis user guide.

GowinSynthesis is a synthesis tool independently developed by Gowin. With project documents input, it supports the synthesis of FSM, Register, Add/Sub and Gowin device primitive RAM/DSP. The synthesis supports common attribute constraints to meet the results requirements in different application conditions. GowinSynthesis is generated based on a synthesized netlist of Gowin device primitive library, which can be used as the input file of the Gowin place and route tool.

2.2 Environment Requirement

Windows: Win7/10（64bit/32bit）

Linux: CentOS6.8/7/7.3, Ubuntu（64bit/32bit）

2.3 Version

This manual is applicable to V1.9.2Beta.

3 An Introduction to GowinSynthesis

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 YunYuan software. In addition to the specified user RTL file, GowinSynthesis project file also specifies the synthesis device, the user constraints file, including attribute constraints file, timing constraints file, netlists file. vg after synthesis, and part of synthesis options, such as top module, file include paths, etc.

3.2 Use GowinSynthesis for Synthesis

Right-click Synthesize in Process view of YunYuan software, select Configuration->GowinSynthesis, then GowinSynthesis tool can be used. The Configurations page can also specify top module, set include path, and select supported language versions, and the configurations will be written to the project file.

Double-click Synthesize in Process view of YunYuan 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 details, please see *Gowin YunYuan Software User Guide* 5.4.3 Synthesis

4 HDL Code Support

4.1 Register HDL Code Support

4.1.1 An Introduction to Register Features

The Gowin register contains flip-flops and latches

Flip-flop

Gowin flip-flops are all D flip-flops. 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 and set signal will lead to the output change immediately, but not controlled by CLK.

Latch

Gowin latch trigger includes high level trigger and low level trigger. 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 Register

Users can constrain register by the preserve attribute. When this constraint exists, except the registers that are suspended will be deleted, all the other registers will be preserved in the synthesis results. Please see 5.8 section 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 user sets the initial value of the synchronous clock flip-flop in the RTL is different from the initial value of Gowin synchronous clock, the synthesis tool will convert the type of synchronous clock flip-flop based on the initial value in the RTL. Asynchronous clock flip-flop do not be handled. Specific conversion strategies are:

1. RTL is designed as synchronous reset clock flip-flop. When the initial value is set to 1, the synthesis tool will replace it with synchronous set

clock flip-flop. Add related logic to the original synchronous reset signal to realize synchronous set function.

2. RTL is designed as synchronous set clock flip-flop. When the initial value is set to 0, the synthesis tool 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

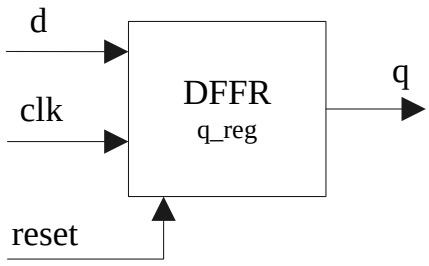
The only difference between CLK rising edge flip-flop and CLK falling edge flip-flop is that CLK triggers in different ways, so the following list is only the examples of CLK rising edge flip-flop.

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



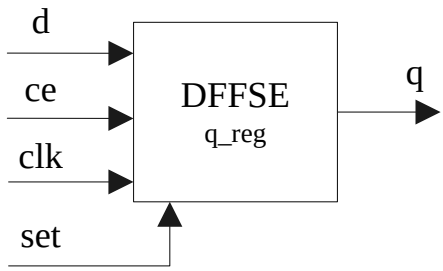
Example 2 can be synthesized as synchronous set flip-flop with clock enable

```
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



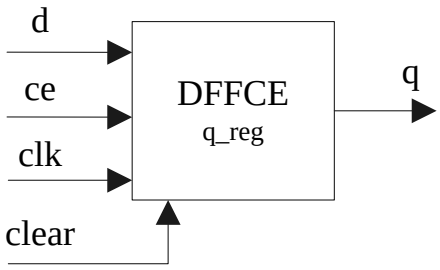
Example 3 can be synthesized as asynchronous reset flip-flop with clock enable

```
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

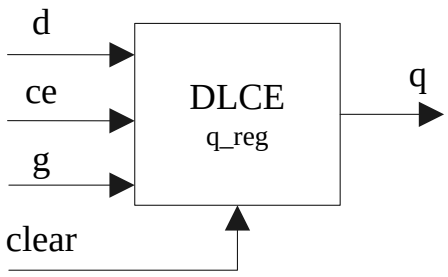


Example 4 can be synthesized as latch with reset and high level enable

```
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



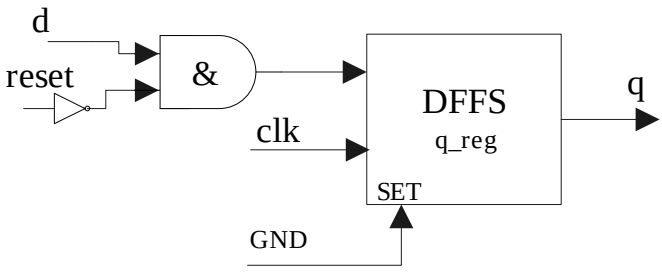
Specify the Initial Value of Flip-flop

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



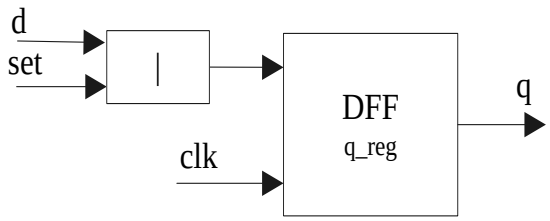
Example 6 is synchronous set clock flip-flop with an initial value of 1.

Set an initial value of 0 in RTL, which 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



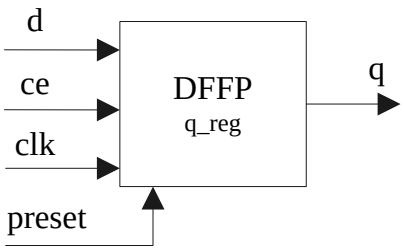
Example 7 is an asynchronous set flip-flop with an initial value of 1.

```
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 Inferencing Function

RAM inferencing is one step in RTL synthesis to infer block memory primitives (BSRAM and SSRAM) in FPGA to implement memory functions in customer design so that customer can write device-independent RTL and still take advantage of built-in block ram functionality in FPGA. For RTL memory blocks, GowinSynthesis will infer the RTL description that meets the corresponding conditions to corresponding RAM module according to RTL description.

If the design needs to implement by B-SRAM, 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. The synthesis tool 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

B-SRAM

There are four configuration modes for B-SRAM: single-port, dual-port, semi-dual-port and read-only. Read mode is divided into pipeline and bypass. Write mode is divided into normal, write-through and read-before-write.

SSRAM

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

4.2.3 Constraints related 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 B-SRAM, please use ram_style or rom_style constraint statement.

How to use constraint syntax, please see 5.2 and 5.3section.

4.2.4 RAM Inference Code Example

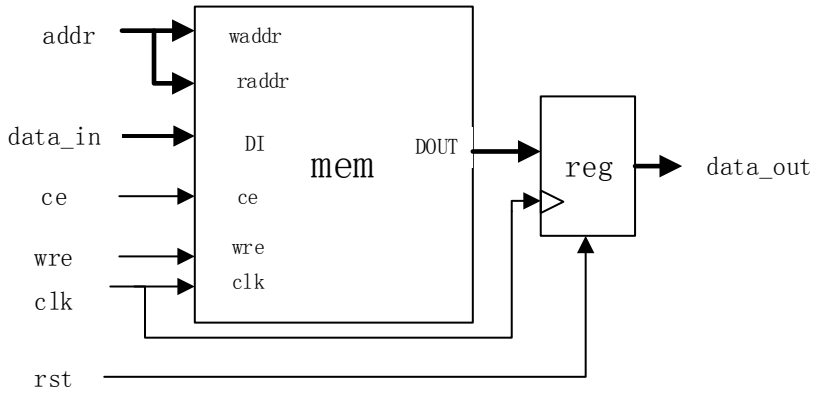
According to the different features of RAM, examples are as follows:

Example 1 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 B-SRAM 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 [127:0];
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 B-SRAM circuit diagram is shown in Figure 4-8.

Figure 4-8 RAM Circuit Diagram in Example 1

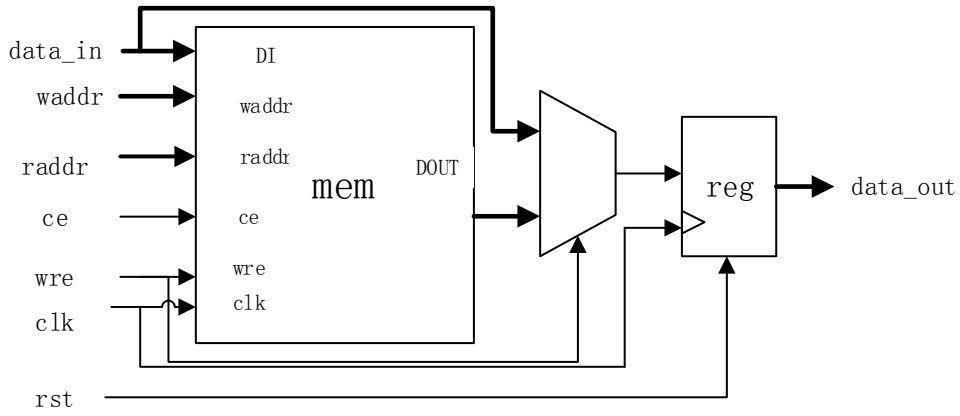


Example 2 is a memory with one write port, one read port and the same read and write address. When wre is 1, input data can be transferred directly to output, which can be synthesized to single-port B-SRAM in normal mode.

```
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 ==1)
          data_out <= 0;
 else
    if(wre == 1)
        data_out <= data_in;
    else
        data_out <= mem[addr];
always @(posedge clk)
    if (wre) mem[addr] <= data_in;
endmodule
```

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

Figure 4-9 RAM Circuit Diagram in Example 2

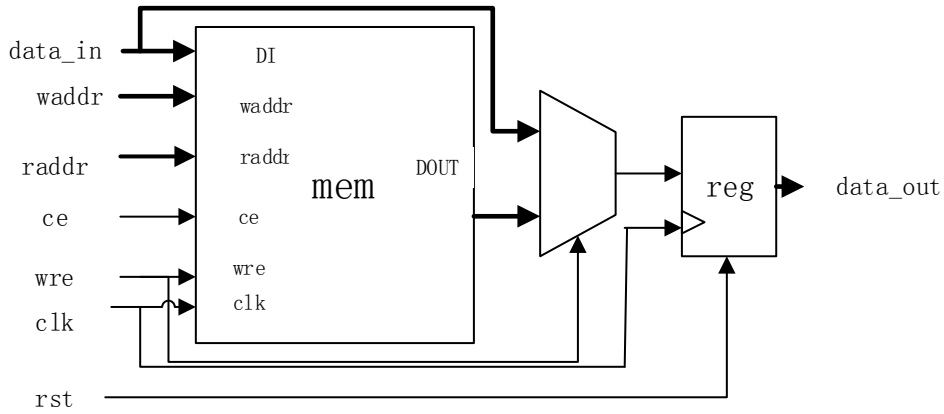


Example 3 is a memory with one write port, one read port and the same read and write address. When wre is 1, input data is written to memory, which can be synthesized to single-port B-SRAM in read-before-write mode.

```
module wt11_2(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)
    if(!wre)
        data_out <= mem[addr];
    else
        data_out <= data_in;
always @(posedge clk)
    if (!wre) mem[addr] <= data_in;
endmodule
```

The above single-port B-SRAM 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 has register for absorbing asynchronous set. This example can be synthesized to asynchronous set dual-port B-SRAM with A port in read-before-write mode and B port in normal mode.

```
module read_first_01(data_outa, data_ina, addra, clka, rsta,cea,
wrea,ocea,data_outb, data_inb, addrb, clkb, rstb,ceb, wreb,oceb);
output [17:0]data_outa,data_outb;
input [17:0]data_ina,data_inb;
input [6:0]addra,addrb;
input clka, rsta,cea, wrea,ocea;
input clkb, rstb,ceb, wreb,oceb;
reg [17:0] mem [127:0];
reg [17:0] data_outa,data_outb;
reg [17:0] data_out_rega,data_out_regb;
always@(posedge clkb or posedge rstb)
if(rstb == 1)
    data_out_regb <= 0;
else begin
    if(ceb)
    data_out_regb <= mem[addrb];
end
always@(posedge clkb or posedge rstb)
if(rstb == 1)
    data_outb <= 0;
else if (oceb)
```

data_outb <= data_out_regb;

always @(posedge clkb)

if (ceb & wreb) mem[addrb] <= data_inb;

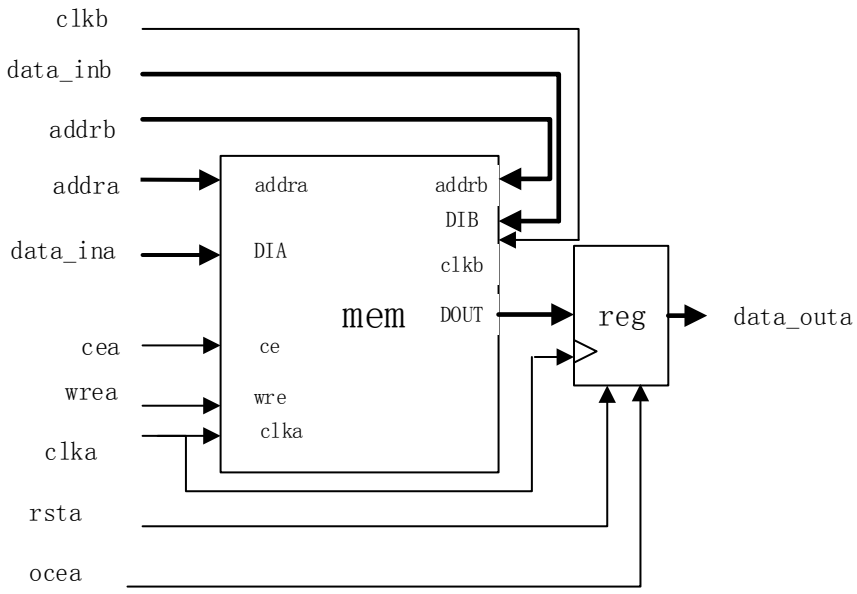
always @(posedge clka)

if (cea & wrea) mem[addra] <= data_ina;

endmodule

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

Figure 4-11 RAM Circuit Diagram in Example 4



Example 5 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 has register for absorbing asynchronous set. This example can be synthesized to asynchronous set dual-port B-SRAM with A port in normal mode and B port in read-before-write mode or in pipeline mode.

module read_first_02_1(data_outa, data_ina, addra, clka, rsta,cea, wrea,ocea,data_outb, data_inb, addrb, clkb, rstb,ceb, wreb,oceb);

output [17:0]data_outa,data_outb;

input [17:0]data_ina,data_inb;

input [6:0]addra,addrb;

input clka, rsta,cea, wrea,ocea;

input clkb, rstb,ceb, wreb,oceb;

reg [17:0] mem [127:0];

reg [17:0] data_outa,data_outb;

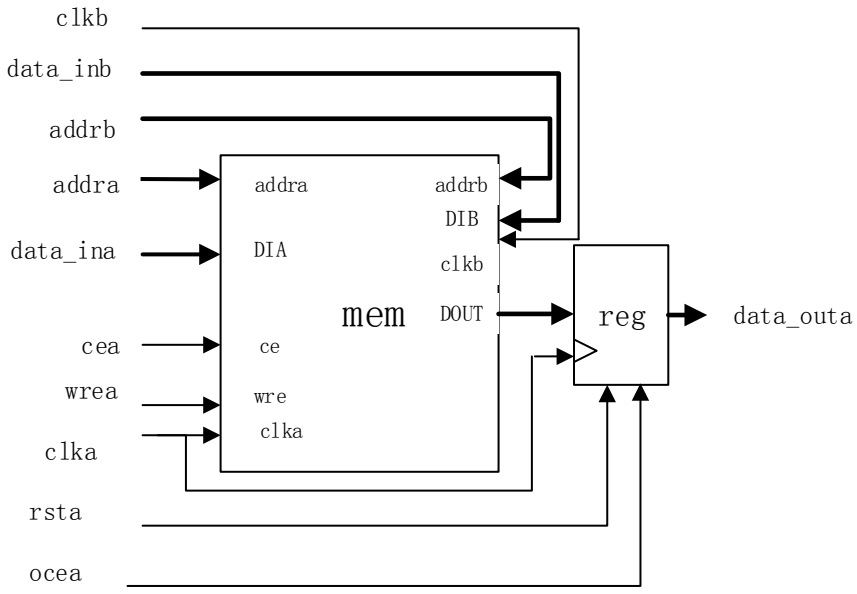
reg [17:0] data_out_rega,data_out_regb;

always @(posedge clkb)

```
if (ceb & wreb) mem[addrb] <= data_inb;
always@(posedge clka or posedge rsta)
if(rsta == 1)
    data_out_rega <= 0;
else begin
    data_out_rega <= mem[addra];
end
always@(posedge clka or posedge rsta)
if(rsta == 1)
    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 B-SRAM circuit diagram is shown in Figure 4-12.

Figure 4-12 RAM Circuit Diagram in Example 5



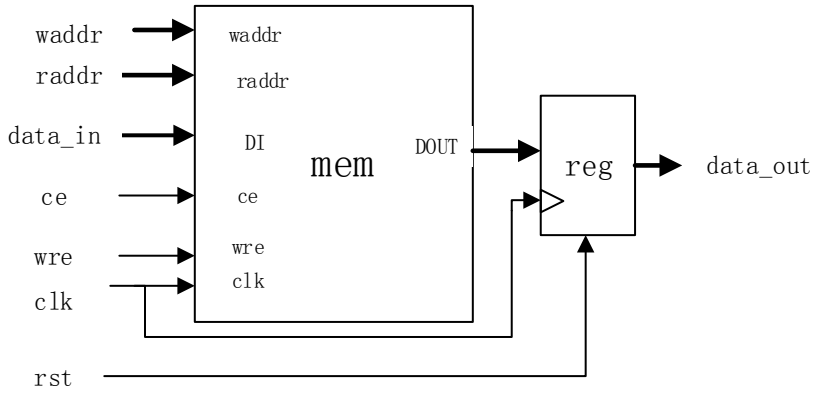
Example 6 is a memory with one read port and one write port and different read and write addresses, which can be synthesized to semi-dual-port B-SRAM in normal mode or in bypass mode.

```
module read_first_wp_pre_1(data_out, data_in, waddr, raddr,clk,
rst,ce, wre);
output [10:0]data_out;
```

```
input [10:0]data_in;
input [6:0]raddr,waddr;
input clk, rst,ce, wre;
reg [10:0] mem [127:0];
reg [10:0] data_out;
always@(posedge clk)
if(ce |wre)
    data_out <= mem[raddr];
always @(posedge clk)
if (rst)
   mem[waddr] <= data_in;
else if (ce | !wre) mem[waddr] <= data_in;
endmodule
```

The above semi-dual-port B-SRAM circuit diagram is shown in Figure 4-13.

Figure 4-13 RAM Circuit Diagram in Example 6



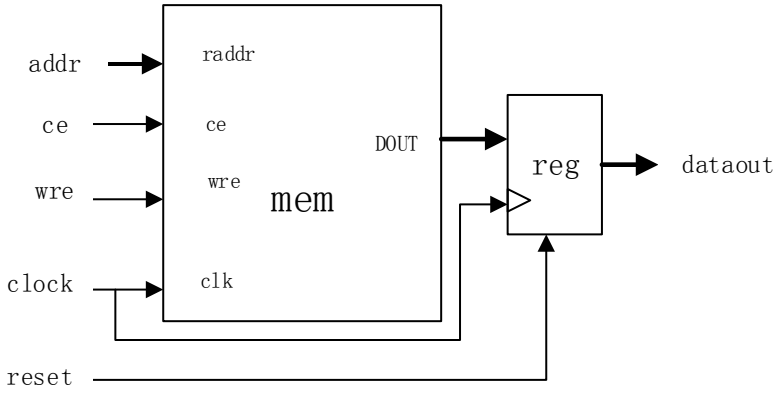
Example 7 is a memory with an initial value of one read port, which can be synthesized to asynchronous set read-only memory in bypass mode.

```
module test_invce (clock,ce,wre,reset,addr,dataout) ;
input clock,ce,wre,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&(!wre)) 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 4-14.

Figure 4-14 RAM Circuit Diagram in Example 7



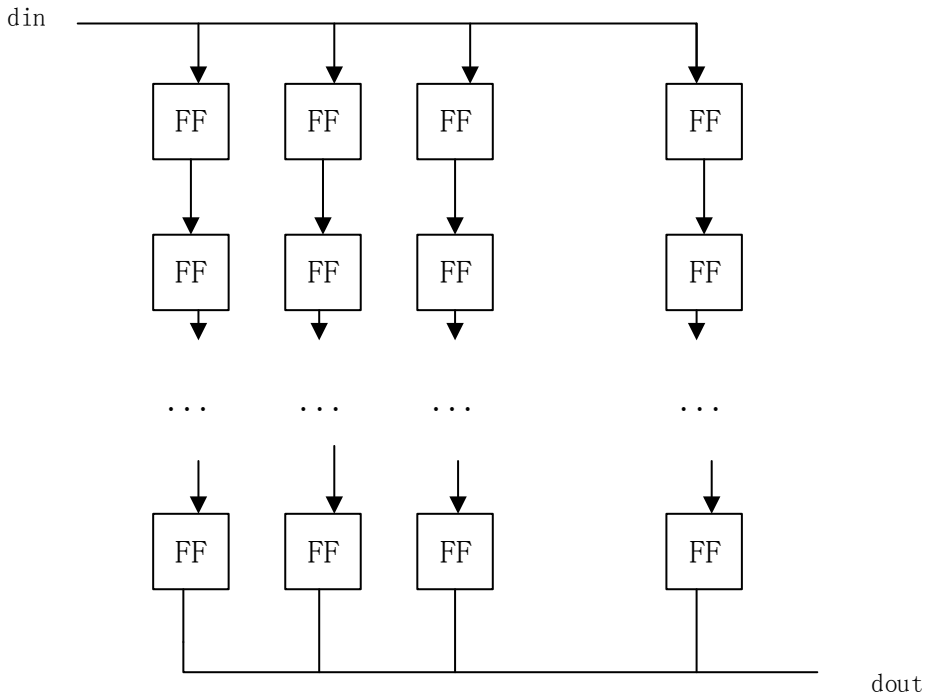
Example 8 is memory with shift-register mode, which can be synthesized to simple-dual-port B-SRAM in normal mode.

```
module seqshift_bsram (clk, din, dout) ;
parameter SRL_WIDTH = 65;
parameter SRL_DEPTH = 4;
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 B-SRAM circuit diagram is shown in Figure 4-15.

Figure 4-15 RAM Circuit Diagram in Example 8



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

4.3.3 Constraints Related 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.

How to use constraint statements, please see 5.1 and 5.8section.

4.3.4 DSP Inference Code Example

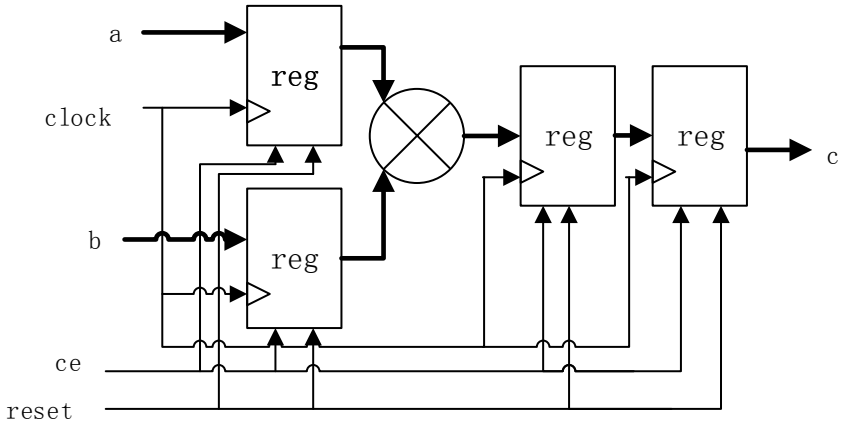
Example 1 can be synthesized to synchronous set multiplier with sign bit. The input registers are ina and inb; The output register is out_reg, and the bypass register is pp_reg.

```
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 4-16.

Figure 4-16 DSP Circuit Diagram in Example 1



Example 2 can be synthesized to the MAC 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.

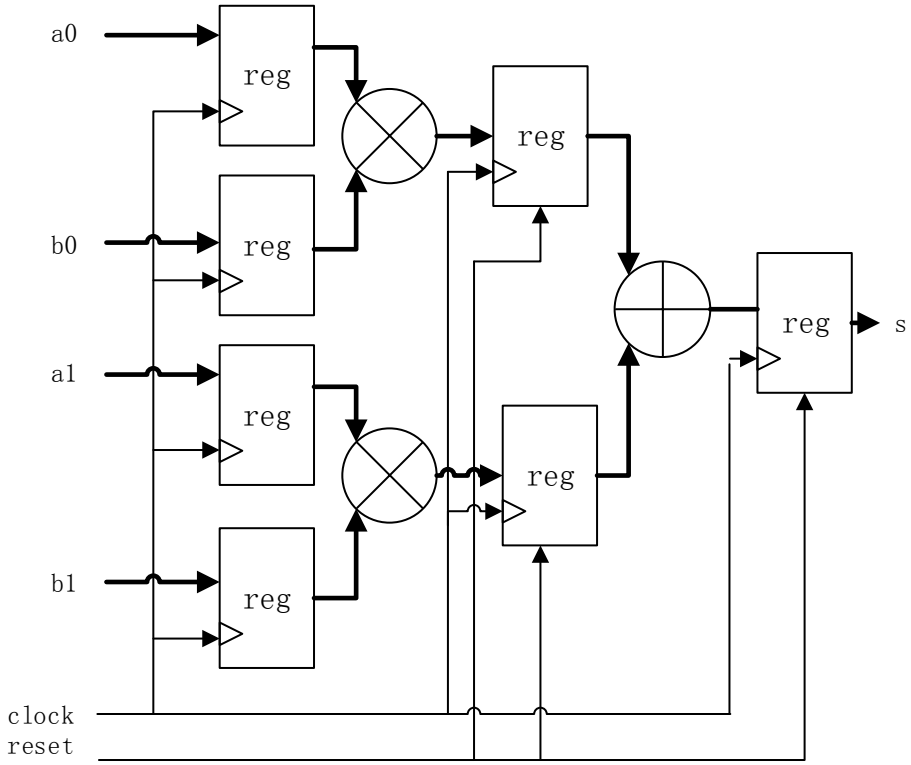
```
module top(a0, a1, b0, b1, s, reset, clock, ce);
parameter a0_width=18;
parameter a1_width=18;
parameter b0_width=18;
parameter b1_width=18;
parameter s_width=37;
input unsigned [a0_width-1:0] a0;
input unsigned [a1_width-1:0] a1;
input unsigned [b0_width-1:0] b0;
input unsigned [b1_width-1:0] b1;
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 multiply add accumulator circuit diagram is shown in Figure 4-17.

Figure 4-17 DSP Circuit Diagram in Example 2

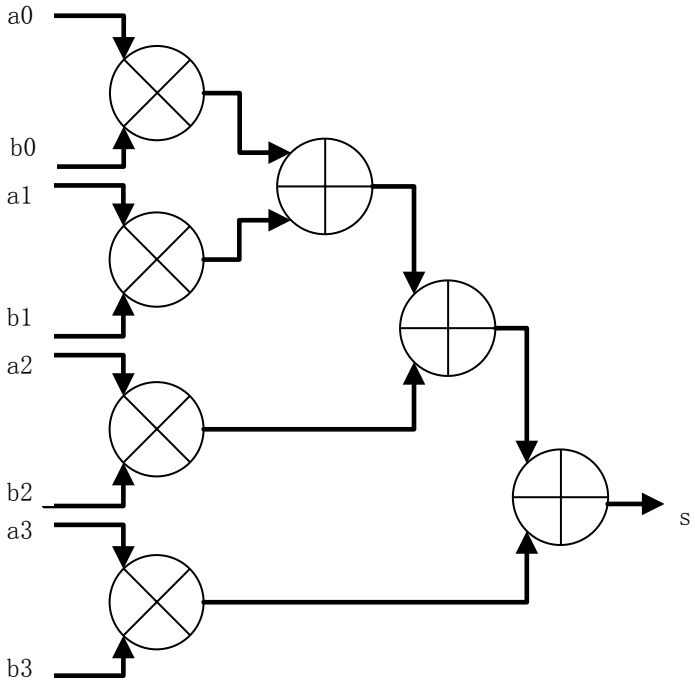


Example 3 can be synthesized to two unsigned bit multipliers, which is chain addition relation.

```
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 multiply add accumulator circuit diagram is shown in Figure 4-18.

Figure 4-18 DSP Circuit Diagram in Example 3

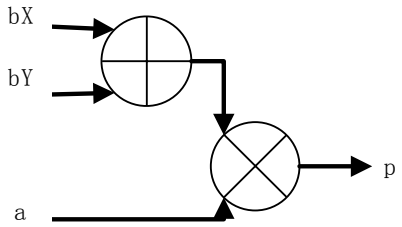


Example 4 can be synthesized to a multiplier with sign-bit 0 and a preadder (MAC) with sign-bit 0. An input port of this MAC is connected to the output port b of the preadder.

```
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 multiply add accumulator circuit diagram is shown in Figure 4-19.

Figure 4-19 DSP Circuit Diagram in Example 4

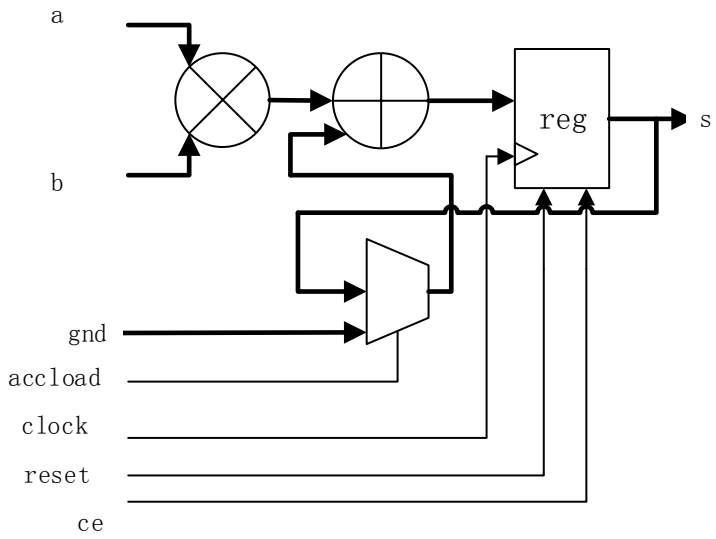


Example 5 can be synthesized to a MAC with an accumulator function and sign-bit 0, and the output register is s.

```
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 multiply add accumulator circuit diagram is shown in Figure 4-20.

Figure 4-20 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

The synthesis tool supports the synthesis of Finate State Machine (FSM), and the encode mode supports one-hot code, gray code, binary code, etc. The synthesis results of finite state machine is related to its the encode mode, code number, code bit width and code constraints. If not setting the code constraints, the synthesis tool automatically selects the one-hot code or gray code to realize state machine. If setting code constraints, the code method specified by the constraints should be implemented first. For the code constraints of state machine, please see 5.5 section.

It should be noted that if the output of the finite state machine drives the output port directly, the synthesis tool 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, the synthesis tool 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 encode mode specified by the constraints. Examples of one-hot encode mode are 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 examples, RTL and synthesis tool are implemented by one-hot code.

Gray Code State Machine

If the state machine adopts gray code for coding in the RTL design, the synthesis tool 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 encode mode specified by the constraints. Examples of gray code are 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 examples, RTL and synthesis tool are implemented by gray code.

Binary Code or other Codes State Machine

If the state machine adopts binary code in RTL design, which is neither one-hot code nor gray code, the synthesis tool 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, gray 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 encode 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 examples, 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 examples, 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 gray code.

Example 3

reg [5:0] state,next_state;

parameter state0= 1,

parameter state1= 3,

parameter state2= 6,

parameter state3= 15;

In the above examples, 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 mode, output netlist format in synthesis so that the synthesis results can better meet the design function and usage. Attribute settings can be written in constraint files or embedded in source code.

This chapter describes the syntax for constraints in RTL files and GowinSynthesis Constraint files. Verilog files are case-sensitive, so directives and attributes must be entered 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 Files

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 can be divided into 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 and use "//". Syntax:

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 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 the direct value > inherited value > default value. When there are multiple inherited values, take the value closest to the specified name (lowest level). For example, query the MULT_STYLE attribute of A/D/C/mult1 ("/" indicates the hierarchy between module names), which has direct value of DSP; Query 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"; Query 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 syn_dspstyle

Description

The specified multiplier is implemented by a dedicated DSP hardware module or logic circuit, 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_dspstyle =setting_value;

GLOBAL syn_dspstyle =setting_value;

RTL Constraint Syntax

verilog object /* synthesis syn_dspstyle =”setting_value” */;

object: Specify the object, which can be either a module name, a module instance name, or a primitive instance name.

Note!

- setting_value: Multiplier implementation currently supports DSP and logic.
- Setting_value is logic, inferring object to logic circuit;
- Setting_value is DSP inferring object to DSP, which is the default.

Examples

GSC Constraints Examples

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;

RTL Constraints Examples

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 temp

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

5.2 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 Constraints Syntax

INS “object” syn_ramstyle =setting_value;

GLOBAL syn_ramstyle =setting_value;

RTL Constraints Syntax

verilog object /* synthesis syn_ramstyle = “setting_value” */ ;

object: Specify the object, which can be either a module name, a module instance name, or a primitive instance name.

Note!

- setting_value: The implementation of memory currently supports block_ram, distributed_ram, registers, rw_check, no_rw_check.

- Setting_value is registers, mapping inferred RAM to registers (flip-flop and logic circuit) rather than dedicated RAM resources;
- Setting_value is block_ram, mapping inferred RAM to dedicated memory, which uses FPGA dedicated memory resources;
- Setting_value is distributed_ram, mapping inferred RAM to dedicated memory (shadow memory block);
- Setting_value is rw_check/no_rw_check, and the default is rw_check. When the read and write are in the same address, uncertain external inputs can cause simulation mismatches. The synthesis tool will insert bypass logic around the infered RAM to avoid mismatches. If the attribute is set to no_rw_check, there is no read/write check and bypass logic will not be inserted around the inferred RAM.

Examples

GSC Constraints Examples

Example 1 specifies B-SRAM to implement instance

INS “mem” syn_ramstyle=block_ram;

Example 2 specifies SSRAM to implement global memory

GLOBAL syn_ramstyle=distributed_ram;

RTL Constraints Examples

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

module test (…) / synthesis syn_ramstyle = “no_rw_check,block_ram” */;*

……

endmodule

Example 2 specifies B-SRAM to implement instance

module test (…) ;

……

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

……

endmodule

5.3 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 Constraints Syntax

INS “object” syn_romstyle =setting_value;

GLOBAL syn_romstyle =setting_value;

RTL Constraints Syntax

verilog object /* synthesis syn_romstyle = “setting_value” */ ;

object: Specify the object, which can be either a module name, a module instance name, or a primitive instance name.

Note!

setting_value: The implementation of read-only memory currently supports block_rom, distributed_rom, logic.

Examples

GSC Constraints 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;

RTL Constraints Examples

Example 1 specifies SSRAM to implement memory in module

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

……

endmodule

5.4 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;

RTL Constraints Syntax

verilog object /* synthesis syn_maxfan = setting_value */ ;

object: Specify the object, which can be either a module name, a module instance name, or a primitive instance name.

Note!

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;

RTLExamples

Example 1 specifies that the max.fanout of instance in module is 3

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

5.5 syn_encoding

Description

Specify the encode mode of state machine, which can be applied to both specific objects and the global.

This attribute can only be specified in RTL files.

Syntax

RTL Syntax

verilog object /* synthesis syn_encoding = “setting_value” */ ;

Note!

- object: Specify the object, which can only be a register name;
- setting_value: One-hot code and gray code are currently supported for state machine.

Examples

RTL Examples

Example 1 specifies gray code for the state machine

module test (…);

reg [2:0] ps, ns/ synthesis syn_encoding="gray" */;*

……

endmodule

5.6 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 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;

5.7 syn_netlist_hierarchy

Description

Specify whether to generate hierarchy netlists. The default 1 indicates the generation of hierarchy netlists. When setting to 0, the hierarchy netlists are output flattened.

This attribute can be specified in GSC and RTL files.

Syntax

GSC Constraints Syntax

GLOBAL syn_netlist_hierarchy=setting_value;

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 are output flattened.

RTL Constraints Syntax

verilog object /* synthesis syn_netlist_hierarchy=setting_value */;

Note!

Object: The object to be specified can only be the top module

Examples

GSC Examples

GLOBAL syn_netlist_hierarchy=0;

RTL Examples

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

……

endmodule

5.8 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 Constraints Syntax

INS “object” syn_preserve=setting_value;

GLOBAL syn_preserve=setting_value;

RTL Constraints Syntax

verilog object /* synthesis syn_preserve = setting_value */;

Note!

Object: Specify the object, which can be either a register name, a module name, or a module instance name.
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 and keeps reg1

INS “reg1” syn_ preserve =1;

Example 2 specifies that all registers in the design are preserved

GLOBAL syn_ preserve =1;

RTL Examples

Example 1 specifies that all registers in the module are preserved

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

……

endmodule

Example 2 specifies and keeps reg1

module test (…);

……

reg reg1/ synthesis syn_preserve = 1 */;*

……

endmodule

5.9 syn_keep

Description

Specify net as a placeholder and leave it unoptimized

This attribute can only be specified in RTL files.

Syntax

RTL Constraints Syntax

verilog object /* synthesis syn_keep= setting_value */;

Note!

Object: The object to be specified can only be the net name.
setting_value: The setting_value can only be 0 or 1. When it is 1, the net is preserved without optimization.

Examples

RTL Constraints Examples

Example 1 specifies mywire and leaves it unoptimized

module test (…);

……

wire mywire / synthesis syn_keep=1 */;*

……

endmodule

5.10 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

RTL Constraints Syntax

verilog object /* synthesis syn_probe = setting_value */;

Note!

- Object: Specify the object, which can only be a net name;
- 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

RTL 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

5.11 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

RTL Constraints Syntax

/* synthesis translate_off*/ ;

Statements that are ignored in the synthesis

/* synthesis translate_on*/

Examples

RTL Constraints Examples

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

5.12 Full_case

Description

Full_case is only used in Verilog's design. When using case, casex, or casez, adding this attribute indicates that all possible values have been

given and that no redundant hardware is required to preserve the signal values.

This attribute can only be specified in RTL files.

Syntax

RTL Constraints Syntax

verilog case /* synthesis full_case*/

Examples

RTL Constraints Examples

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
```

5.13 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 Constraints 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;

RTL Constraints Syntax

verilog object /* synthesis syn_tlvds_io = setting_value */;

Note!

- Object: Specify the object, which can be either a module name or a port name.

- setting_value: 0 or 1.

Examples

GSC Constraints 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;

RTL Constraints Examples

module elvds_iobuf(io,iob…);

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

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

.....

endmodule

6 Report User 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, Design Settings, Resource, Timing, Message and Summary.

6.1 Synthesis Message

Synthesis Message refers to the basic information of Synthesis. As shown in Figure 6-1. It mainly includes design document, the current GowinSynthesis version, running time, etc.

Figure 6-1 Synthesis Message

Synthesis Messages

Report Title	GowinSynthesis Report
Design File	/gwsw/sw_pub/swfiles/Gowin/memory/src/ram_1.v
GowinSynthesis Constraints File	---
GowinSynthesis Verision	GowinSynthesis V1.9.2Beta
Created Time	Wed Aug 14 17:26:08 2019
Legal Announcement	Copyright (C)2014-2019 Gowin Semiconductor Corporation. ALL rights reserved.

6.2 Design Settings

Design Settings is the design configuration information. As shown in Figure 6-2, it mainly includes the top level module, the language setting, the chip type specified, etc.

Figure 6-2 Design Settings

Design Settings

Top Level Module:	RAM_test
Design Language:	verilog
Series:	GW2A
Device:	GW2A-55
Package:	PBGA484
Speed Grade:	7

6.3 Resource

Resource refers to the resource information. It mainly includes resource and chip utilization statistics.

The resource utilization table counts the number of IOPORT, IOBUF, REG, LUT, etc. The resource utilization table is used to estimate the resource utilization rate of CFU Logics, Register, BSRAM, DSP, etc. in the current device, as shown in Figure 6-3:

Figure 6-3 Resource

Resource

Resource Usage Summary

IOPORT Usage:	16
IOBUF Usage:	16
IBUF	12
OBUF	4
BSRAM Usage:	1
SP	1

Resource Utilization Summary

Target Device: GW2A-55-PBGA484

CFU Logics	0(0 LUTs, 0 ALUs) / 54720	0%
Registers	0 / 41040	0%
BSRAMs	1 / 140	1%
DSP Macros	0 / (10*2)	0%

6.4 Timing

Timing refers to timing statistics. It mainly includes Clock Summary, Timing Report, Performance Summary, Detail Timing Paths Informations and etc.

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

Timing Report mainly describes timing of netlists, including top level module, limited frequency and the maximum number of timing paths, and its default is 5. All time values are in nanoseconds.

Figure 6-4 Timing

Timing

Clock Summary:

Clock	Type	Frequency	Period	Rise	Fall	Source	Master	Object
DEFAULT_CLK	Base	100.0 MHz	10.000	0.000	5.000			

Timing Report:

Top View:	test_time
Requested Frequency:	100.0 MHz
Paths Requested:	5
Constraint File(ignored):	

All time values displayed in nanoseconds(ns).

Performance Summary mainly counts the maximum time delay and the time frequency that can be achieved of the netlists file in order to measure whether the timing sequence meets the requirements. As shown in Figure 6-5below. The slack is 8.462ns; The requested frequency is 100MHZ and the estimated frequency is 650MHZ; The requested clock period is 10ns and the estimated period is 1.538ns. They meet the timing sequence requirements. If the slack is negative, which can not meet, and the specific timing path needs to be further checked.

Figure 6-5 Performance Summary

Performance Summary:

Worst Slack in Design: 8.462

Start Clock	Slack	Requested Frequency	Estimated Frequency	Requested Period	Estimated Period	Clock Type
DEFAULT_CLK	8.462	100.0 MHz	650.1 MHz	10.000	1.538	Base

Detail Timing Paths Information mainly describes the key timing paths, starting and ending points, related delay information in netlist files, as shown in Figure 6-6 below. The detailed connection relation, delay and fanout information are as shown in Figure 6-7.

Figure 6-6 Detail Timing Paths Information

Detail Timing Paths Information

Path information for path number 1 :

Clock Skew:	0.000
Setup Relationship:	10.000
Slack(critical):	8.462
Data Arrival Time:	2.283
Data Required Time:	10.745
Number of Logic Level:	0
Starting Point:	dff1
Ending Point:	dff2
The Start Point Is Clocked By:	DEFAULT_CLK[rising]
The End Point Is Clocked By:	DEFAULT_CLK[rising]

Figure 6-7 Connection Relation, Delay and Fanout Information

Instance/Net Name	Type	Pin Name	Pin Dir	Delay	Arrival Time	Fanout
clk_ins12	IBUF	I	In	-	0.000	-
clk_ins12	IBUF	O	Out	0.982	0.982	-
clk_3	Net	-	-	0.363	-	4
dff1_ins1	DFF	CLK	In	-	1.345	-
dff1_ins1	DFF	Q	Out	0.458	1.803	-
b	Net	-	-	0.480	-	1
dff2_ins2	DFF	D	In	-	2.283	-

Total Path Delay: 2.283
Logic Delay: 1.440(63.1%)
Route Delay: 0.843(36.9%)

6.5 Message

Message refers to the synthesis output message. It is the log message in the synthesis, which includes Info, Warning and Error. Info is prompt information, Warning is warning information, and Error is error information. When Error information occurs during the operation, the synthesis flow exits and no output file is generated. The synthesis output message is as shown in Figure 6-8.

Figure 6-8 Message

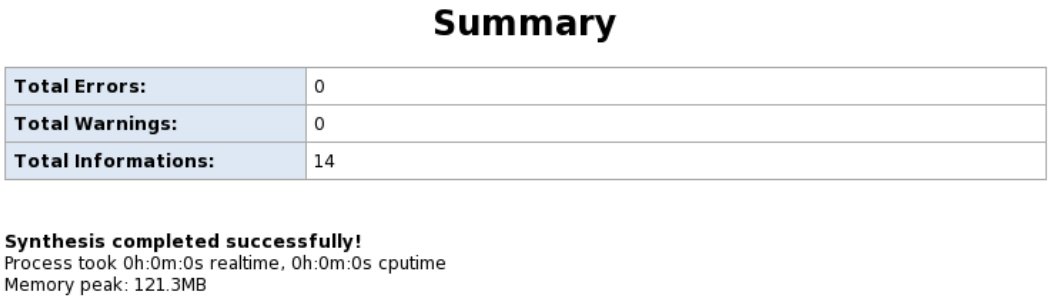
Message

Info (EXT0100) : Run analyzation & elaboration
Info (EXT1482) : Analyzing Verilog file '/share/gwsw/sw_pub/swfiles/Gowin/fpga_project/src/counter1.v'
Info (EXT1018) : Compiling module 'RAM_test'(/gwsw/sw_pub/swfiles/Gowin/memory/src/ram_1.v:2)
Info (EXT0101) : Current top module is "RAM_test"
Info (CVT0001) : Run conversion
Info (DIO0001) : Run device independent optimization
Info (DIO0006) : Register and gate optimizing before inferencing
Info (DSP0001) : DSP inferencing
Info (RAM0001) : RAM inferencing
Info (DIO0001) : Run device independent optimization
Info (DIO0007) : Register and gate optimizing before mapping
Info (MAP0001) : Run tech-mapping
Info (DIO0001) : Run device independent optimization
Info (SYN0009) : Write post-map netlist to file:
/gwsw/sw_pub/swfiles/Gowin/memory/impl/gwsynthesis/memory.vg

6.6 Summary

Summary table counts the number of warnings, errors and information of the output and displays the actual running time and CPU running time as well as the memory peak in the synthesis. As shown in Figure 6-9.

Figure 6-9 Summary



Summary

Total Errors:	0
Total Warnings:	0
Total Informations:	14

Synthesis completed successfully!
Process took 0h:0m:0s realtime, 0h:0m:0s cputime
Memory peak: 121.3MB

7 Hierarchy Resource Document

Hierarchy resource information document is the resource statistics file of each module generated after synthesis, whose file name is composed of * _syn_resource-html (* is the name of the specified output netlist vg file).

The Resource file counts the module hierarchy and displays the resource statistics of each module and prints them according to the hierarchy. As shown in Figure 7-1, the Resource file prints all modules in depth-first order from the top to the bottom. Each line includes module name, the file path, and the number of REG, ALU, LUT, DSP, BSRAM, SSRAM, etc.in each module. Users can see the details in hierarchy resource document.

Figure 7-1 Hierarchy Module Resource

Hierarchy Module Resource

MODULE NAME	REG NUMBER	ALU NUMBER	LUT NUMBER	DSP NUMBER	BSRAM NUMBER	SSRAM NUMBER
nflash_test_top (/share/gwsw/sw_pub/swfiles/Gowin/nflash_test/nflash_test_top.v)	2	-	-	-	-	-
|--pll_inst (/share/gwsw/sw_pub/swfiles/Gowin/nflash_test/nflash_test_top.v)	-	-	-	-	-	-
|--flash_data_cmd_inst (/share/gwsw/sw_pub/swfiles/Gowin/nflash_test/nflash_test_top.v)	36	-	85	-	-	-
|--mixBF (/share/gwsw/sw_pub/swfiles/Gowin/nflash_test/flash_data_cmd.v)	-	-	1	-	-	-
|--nfcm_top_inst (/share/gwsw/sw_pub/swfiles/Gowin/nflash_test/nflash_test_top.v)	32	-	80	-	-	-
|--dp_2k_d8 (/share/gwsw/sw_pub/swfiles/Gowin/nflash_test/nfcm_top.v)	-	-	-	-	1	-
|--addr_counter (/share/gwsw/sw_pub/swfiles/Gowin/nflash_test/nfcm_top.v)	12	-	22	-	-	-
|--tim_fsm (/share/gwsw/sw_pub/swfiles/Gowin/nflash_test/nfcm_top.v)	9	-	67	-	-	-
|--main_fsm (/share/gwsw/sw_pub/swfiles/Gowin/nflash_test/nfcm_top.v)	8	-	173	-	-	-
|--ecc_gen (/share/gwsw/sw_pub/swfiles/Gowin/nflash_test/nfcm_top.v)	7	-	3	-	-	-
|--ecc_err_loc (/share/gwsw/sw_pub/swfiles/Gowin/nflash_test/nfcm_top.v)	8	-	6	-	-	-


GOWIN
PROGRAMMING FOR FUTURE