Gowin HDL 编码风格

用户指南

SUG949-1.9, 2025-06-27

版权所有 © 2025 广东高云半导体科技股份有限公司

 、、Gowin、GowinSynthesis、晨熙以及小蜜蜂以及高云均为广东高云半导体科技股份有限公司注册商标，本手册中提到的其他任何商标，其所有权利属其拥有者所有。未经本公司书面许可，任何单位和个人都不得擅自摘抄、复制、翻译本文档内容的部分或全部，并不得以任何形式传播。

免责声明

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

版本信息

日期	版本	说明
2020/08/31	1.0	初始版本。
2021/06/10	1.1	添加单端 BUF 的 infer 案例。
2022/05/31	1.2	删除编码风格要求描述。
2023/04/20	1.3	新增第 7 章 Arora V DSP 编码规范。
2023/06/30	1.4	更新第 4 章 BSRAM 编码规范和第 5 章 SSRAM 编码规范。
2023/11/30	1.5	更新第 4 章 BSRAM 编码规范和第 5 章 SSRAM 编码规范。
2024/03/29	1.6	更新第 4 章 BSRAM 编码规范。
2024/06/28	1.7	更新 3.1 节 LUT 描述。
2024/10/25	1.7.1	更新 4.1.2 节读地址不经过 Register，输出经过 Register 中的案例。
2025/04/30	1.8	● 新增 VHDL 示例； ● 删除 SP 的 Decoder 形式。
2025/06/27	1.9	● 更新 4.2 SP/SPX9 中的注释，补充禁用 SP read-before-write 的器件。 ● 更新 3.1.1 查找表形式中 VHDL 示例。

目录

表目录

1 关于本手册

1.1 手册内容

本手册主要描述高云 HDL 编码风格要求及原语的 HDL 编码实现，旨在帮助用户快速熟悉高云 HDL 编码风格和原语实现，指导用户设计，提高设计效率。

1.2 相关文档

通过登录高云半导体网站 www.gowinsemi.com.cn 可下载、查看以下相关文档：

- SUG100, Gowin 云源软件用户指南
- SUG283, Gowin 原语用户指南

1.3 术语、缩略语

表 1-1 中列出了本手册中出现的相关术语、缩略语及相关释义。

表 1-1 术语、缩略语

术语、缩略语	全称	含义
BSRAM	Block Static Random Access Memory	块状静态随机存储器
CLU	Configurable Logic Unit	可配置逻辑单元
DSP	Digital Signal Processing	数字信号处理
FSM	Finite State Machine	有限状态机
HDL	Hardware Description Language	硬件描述语言
LUT	Look-up Table	查找表
SSRAM	Shadow Static Random Access Memory	分布式静态随机存储器

1.4 技术支持与反馈

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

网址：www.gowinsemi.com.cn

E-mail：support@gowinsemi.com

Tel：+86 755 8262 0391

2 Buffer 编码规范

Buffer 缓冲器，具有缓存功能。根据不同功能，可分为单端 buffer、模拟 LVDS（ELVDS）和真 LVDS（TLVDS）。模拟 LVDS 和真 LVDS 的原语实现需添加相应的属性约束，建议采用实例化方式编码。

2.1 IBUF

IBUF(Input Buffer)，输入缓冲器。其编码形式可采用如下 2 种方式：

方式 1：

Verilog 示例：

```
module rtl_ibuf (o, i);
input i ;
output o ;
assign o = i ;
endmodule
```

VHDL 示例：

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity rtl_ibuf is
    PORT (
        o: OUT std_logic;
        i : IN std_logic
    );
end rtl_ibuf;

architecture Behavioral of rtl_IBUF is
```

```
begin
    o <= i;
end Behavioral;
```

方式 2：

Verilog 示例：

```
module ibuf (o, i);
input i ;
output o ;
buf IB (o, i);
endmodule
```

2.2 TLVDS_IBUF

TLVDS_IBUF(True LVDS Input Buffer)，真差分输入缓冲器。该原语的实现需要添加属性约束，其编码形式可如下所示：

Verilog 示例：

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

VHDL 示例：

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity tlvds_ibuf_test is
    PORT (
        o : OUT std_logic;
        in1_p : IN std_logic;
        in1_n : IN std_logic
```

```
    );
attribute syn_tlvds_io:integer;
attribute syn_tlvds_io of in1_p,in1_n:signal is 1;
end tlvds_ibuf_test;

architecture Behavioral of tlvds_ibuf_test is
begin
    process(in1_n,in1_p)begin
        if(in1_p/=in1_n)then
            o<=in1_p;
        end if;
    end process;
end Behavioral;
```

2.3 ELVDS_IBUF

ELVDS_IBUF(Emulated LVDS Input Buffer)，模拟差分输入缓冲器。该原语的实现需要添加属性约束，其编码形式可如下所示：

Verilog 示例：

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

VHDL 示例：

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity elvds_ibuf_test is
    PORT (
        o : OUT std_logic;
```

```
        in1_p : IN std_logic;
        in1_n : IN std_logic
    );
attribute syn_elvds_io:integer;
attribute syn_elvds_io of in1_p,in1_n:signal is 1;
end elvds_ibuf_test;

architecture Behavioral of elvds_ibuf_test is
begin
  process(in1_p, in1_n)
    begin
      if (in1_p/=in1_n) then
          o <= in1_p ;
      end if;
  end process;

end Behavioral;
```

2.4 OBUF

OBUF(Output Buffer)，输出缓冲器。其编码形式可采用如下 2 种方式：

方式 1：

Verilog 示例：

```
module rtl_obuf (o, i);
input i ;
output o ;
assign o = i ;
endmodule
```

VHDL 示例：

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity rtl_obuf is
    PORT (
        o : OUT std_logic;
```

```
        i : IN std_logic
    );
end rtl_obuf;

architecture Behavioral of rtl_obuf is
begin
        o <= i;
end Behavioral;
```

方法 2：

Verilog 示例：

```
module obuf (o, i);
input i ;
output o ;
buf OB (o, i);
endmodule
```

2.5 TLVDS_OBUF

TLVDS_OBUF(True LVDS Output Buffer)，真差分输出缓冲器。该原语的实现需要添加属性约束，其编码形式可如下所示：

Verilog 示例：

```
module tlvds_obuf_test(i,out1,out2);
input i;
output out1/* synthesis syn_tlvds_io = 1*/;
output out2/* synthesis syn_tlvds_io = 1*/;
assign  out1 = i;
assign out2 = ~out1;
endmodule
```

VHDL 示例：

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity tlvds_obuf_test is
    PORT (
        out1 : out std_logic;
```

```
        out2 : out std_logic;
        i : IN std_logic
    );
attribute syn_tlvds_io:integer;
attribute syn_tlvds_io of out1,out2:signal is 1;
end tlvds_obuf_test;

architecture Behavioral of tlvds_obuf_test is
begin
    out1<=i;
    out2<=not(i);
end Behavioral;
```

2.6 ELVDS_OBUF

ELVDS_OBUF(Emulated LVDS Output Buffer)，模拟差分输出缓冲器。该原语的实现需要添加属性约束，其编码形式可如下所示：

Verilog 示例：

```
module elvds_obuf_test(i,out1,out2);
input i;
output out1/* synthesis syn_elvds_io = 1*/;
output out2/* synthesis syn_elvds_io = 1*/;
assign out1= i;
assign out2 = ~out1;
endmodule
```

VHDL 示例：

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity elvds_obuf_test is
    PORT (
        out1 : out std_logic;
        out2 : out std_logic;
        i : IN std_logic
    );
attribute syn_elvds_io:integer;
```

```
attribute syn_elvds_io of out1,out2:signal is 1;
end elvds_obuf_test;

architecture Behavioral of elvds_obuf_test is
begin
    out1<=i;
    out2<=not(i);
end Behavioral;
```

2.7 TBUF

TBUF(Output Buffer with Tri-state Control)，三态缓冲器，低电平使能。其编码形式可采用如下 2 种方式：

方式 1：

Verilog 示例：

```
module tbuf (i, oen, o);
input i, oen;
output o;
assign o= ~oen ? i:1'bz;
endmodule
```

VHDL 示例：

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity rtl_tbuf is
    PORT (
        o : OUT std_logic;
        i : IN std_logic;
        oen : IN std_logic
    );
end rtl_tbuf;

architecture Behavioral of rtl_tbuf is
begin
  process(i, oen)
```

```
    begin
      if ((oen = '0') or (oen = 'L')) then
          o <= TO_X01(i);
      elsif ((oen = '1') or (oen = 'H')) then
        o <= 'Z';
      else
        o <= 'X';
      end if;
  end process;

end Behavioral;
```

方式 2：

Verilog 示例：

```
module tbuf (out, in, oen);
input in, oen;
output out;
bufif0 TB (out, in, oen);
endmodule
```

2.8 TLVDS_TBUF

TLVDS_TBUF(True LVDS Tristate Buffer)，真差分三态缓冲器，低电平使能。该原语的实现需要添加属性约束，其编码形式可如下所示：

Verilog 示例：

```
module tlvds_tbuf_test(i, oen, out1,out2);
input i;
input oen;
output out1/* synthesis syn_tlvds_io = 1*/;
output out2/* synthesis syn_tlvds_io = 1*/;
assign out1 = ~oen ? i : 1'bz;
assign out2 = ~oen ? ~i : 1'bz;
endmodule
```

VHDL 示例：

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
entity   tlvds_tbuf_test   is
    PORT (
        out1 : out std_logic;
        out2 : out std_logic;
        oen : IN std_logic;
        i : IN std_logic
    );
attribute syn_tlvds_io:integer;
attribute syn_tlvds_io of out1,out2:signal is 1;
end tlvds_tbuf_test;

architecture Behavioral of tlvds_tbuf_test is
begin
    process(i,oen)begin
        if(oen='0')then
            out1<=i;
            out2<=not(i);
        else
            out1<='Z';
            out2<='Z';
        end if;

    end process;
end Behavioral;
```

2.9 ELVDS _TBUF

ELVDS_TBUF(Emulated LVDS Tristate Buffer)，模拟差分三态缓冲器，低电平使能。该原语的实现需要添加属性约束，其编码形式可如下所示：

Verilog 示例：

```
module elvds_tbuf_test(i, oen, out1,out2);
input i, oen;
output out1/* synthesis syn_elvds_io = 1*/;
output out2/* synthesis syn_elvds_io = 1*/;
assign out1 = ~oen ? i : 1'bz;
```

```
assign out2 = ~oen ? ~i : 1'bz;
endmodule
```

VHDL 示例：

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity elvds_tbuf_test is
    PORT (
        out1 : out std_logic;
        out2 : out std_logic;
        oen : IN std_logic;
        i : IN std_logic
    );
attribute syn_elvds_io:integer;
attribute syn_elvds_io of out1,out2:signal is 1;
end elvds_tbuf_test;

architecture Behavioral of elvds_tbuf_test is
begin
    process(i,oen)begin
        if(oen='0')then
            out1<=i;
            out2<=not(i);
        else
            out1<='Z';
            out2<='Z';
        end if;

    end process;
end;
```

2.10 IOBUF

IOBUF(Bi-Directional Buffer)，双向缓冲器。当 OEN 为高电平时，作为输入缓冲器；OEN 为低电平时，作为输出缓冲器。其编码形式可采用如下 2 种方式：

方式 1：

Verilog 示例：

```
module rtl_iobuf (i, oen,    io, o);
input i,oen;
output o;
inout io;
assign io= ~oen ? i :1'bz;
assign o = io;
endmodule
```

VHDL 示例：

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity rtl_iobuf is
      PORT (
          o    : OUT     std_logic;
          io : INOUT std_logic;
          i    : IN       std_logic;
          oen : IN       std_logic
      );
end rtl_iobuf;

architecture    Behavioral of rtl_iobuf is
begin
      process(io, i, oen)
      begin
         o <= TO_X01(io);
    if ((oen = '0') or (oen = 'L')) then
          io <= i;
    elsif ((oen = '1') or (oen = 'H')) then
          io <= 'Z';
    else
          io <= 'X';
    end if;
```

```
    end process;

end Behavioral;
```

方式 2：

Verilog 示例：

```
module iobuf (out, io, i, oen);
input i,oen;
output out;
inout io;
buf OB (out, io);
bufif0 IB (io,i,oen);
endmodule
```

2.11 TLVDS_IOBUF

TLVDS_IOBUF(True LVDS Bi-Directional Buffer)，真差分双向缓冲器，当 OEN 为高电平时，作为真差分输入缓冲器；OEN 为低电平时，作为真差分输出缓冲器。该原语的实现需要添加属性约束，其编码形式可如下所示：

Verilog 示例：

```
module rtl_tlvds_iobuf(o, io, iob, i, oen);
output reg o;
inout io /* synthesis syn_tlvds_io = 1 */;
inout iob /* synthesis syn_tlvds_io = 1 */;
input i, oen;
bufif0 ib(io, i, oen);
notif0 yb(iob, i, oen);
always @(io or iob)begin
    if (io != iob)begin
        o <= io;
    end
end
endmodule
```

VHDL 示例：

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
entity rtl_tlvds_iobuf is
    PORT (
        o : out std_logic;
        io : inout std_logic;
        iob : inout std_logic;
        i : IN std_logic;
        oen : IN std_logic
    );
attribute syn_tlvds_io:integer;
attribute syn_tlvds_io of io,iob:signal is 1;
end rtl_tlvds_iobuf;

architecture Behavioral of rtl_tlvds_iobuf is
begin
  process(i, oen)
    begin
      if (oen='1') then
          io <= 'Z' ;
        else
              io<=i;
      end if;
  end process;
  process(i, oen)
    begin
      if (oen='1') then
          iob <= 'Z' ;
        else
              iob<=not(i);
      end if;
  end process;

  process(io,iob)
    begin
      if (io/=iob) then
```

```
            o<=io;
        end if;
    end process;
end Behavioral;
```

2.12 ELVDS _IOBUF

ELVDS_IOBUF(Emulated LVDS Bi-Directional Buffer)，模拟差分双向缓冲器，当 OEN 为高电平时，作为模拟差分输入缓冲器；OEN 为低电平时，作为模拟差分输出缓冲器。该原语的实现需要添加属性约束，其编码形式可如下所示：

Verilog 示例：

```
module elvds_iobuf(o, io, iob, i, oen);
output o;
inout io /* synthesis syn_elvds_io = 1 */;
inout iob /* synthesis syn_elvds_io = 1 */;
input i, oen;
reg o;
bufif0 ib(io, i, oen);
notif0 yb(iob, i, oen);
always @(io or iob)begin
    if (io != iob)begin
        o <= io;
    end
end
endmodule
```

VHDL 示例：

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity rtl_elvds_iobuf is
    PORT (
        o : out std_logic;
        io : inout std_logic;
        iob : inout std_logic;
        i : IN std_logic;
        oen : IN std_logic
```

```
    );
attribute syn_elvds_io:integer;
attribute syn_elvds_io of io,iob:signal is 1;
end rtl_elvds_iobuf;
architecture Behavioral of rtl_elvds_iobuf is
begin
  process(i, oen)
    begin
      if (oen='1') then
          io <= 'Z' ;
        else
            io<=i;
      end if;
  end process;
  process(i, oen)
    begin
      if (oen='1') then
          iob <= 'Z' ;
        else
            iob<=not(i);
      end if;
  end process;

  process(io,iob)
    begin
      if (io/=iob) then
        o<=io;
      end if;
  end process;
end Behavioral;
```

3 CLU 编码规范

可配置逻辑单元 CLU（Configurable Logic Unit）是构成 FPGA 产品的基本单元，CLU 模块可实现 MUX/LUT/ALU/FF/LATCH 等模块的功能。

3.1 LUT

输入查找表 LUT，常用的 LUT 原语有 LUT2、LUT3、LUT4，其区别在于查找表输入位宽的不同，其实现方式可采用如下几种。

3.1.1 查找表形式

Verilog 示例：

```
module rtl_LUT4 (f, i0, i1,i2,i3);
parameter INIT = 16'h2345;
input i0, i1, i2,i3;
output f;
assign f=INIT[{i3,i2,i1,i0}];
endmodule
```

VHDL 示例：

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;
entity rtl_LUT4 is
    GENERIC ( INIT : std_logic_vector (15 DOWNTO 0) := X"2345" );
    PORT (
        f : out std_logic;
          i0 : in std_logic;
          i1 : in std_logic;
        i2 : in std_logic;
```

```
        i3 : in std_logic
    );
end rtl_LUT4;

architecture Behavioral of rtl_LUT4 is
    SIGNAL addr:STD_LOGIC_VECTOR (3 DOWNTO 0);
begin
    addr<=I3&I2&I1&I0;
    f<=INIT(conv_integer(addr));
end;
```

3.1.2 选择器形式

Verilog 示例：

```
module rtl_LUT3 (f,a,b,sel);
input a,b,sel;
output f;
assign f=sel?a:b;
endmodule
```

VHDL 示例：

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity rtl_LUT3 is
    PORT (
        f : out std_logic;
        a : in std_logic;
        b : in std_logic;
        sel : in std_logic
    );
end rtl_LUT3;

architecture Behavioral of rtl_LUT3 is
begin
    process(a,b,sel)begin
        if(sel='1')then
```

```
            f<=a;
        else
            f<=b;
        end if;
    end process;
end;
```

3.1.3 逻辑运算形式

Verilog 示例：

```
module top(a,b,c,d,f);
input [3:0]a,b,c,d;
output [3:0]f;
assign f=a&b&c|d;
endmodule
```

VHDL 示例：

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity top is
    PORT (
        f : out std_logic;
        a : in std_logic;
        b : in std_logic;
        c : in std_logic;
        d : in std_logic
    );
end top;

architecture Behavioral of top is
begin
    f <= (a and b and c) or d;
end;
```

3.2 ALU

ALU（2-input Arithmetic Logic Unit）2 输入算术逻辑单元，综合工具可以综合出 ADD/SUB/ADDSUB/NE 等功能。

3.2.1 ADD 功能

以 4 位全加器和 4 位半加器为例介绍 ALU 的 ADD 功能：

4 位全加器

4 位全加器其编码形式可如下所示：

Verilog 示例：

```
module add(a,b,cin,sum,cout);
input [3:0] a,b;
input cin;
output [3:0] sum;
output cout;
assign {cout,sum}=a+b+cin;
endmodule
```

VHDL 示例：

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity add is
    Port ( a : in   STD_LOGIC_VECTOR (3 downto 0);
           b : in   STD_LOGIC_VECTOR (3 downto 0);
           cin : in   STD_LOGIC;
           sum : out   STD_LOGIC_VECTOR (3 downto 0);
           cout : out   STD_LOGIC);
end add;

architecture Behavioral of add is
    signal temp_sum : STD_LOGIC_VECTOR (4 downto 0);
begin
    temp_sum <= ('0' & a) + ('0' & b) + cin;
    sum <= temp_sum(3 downto 0);
```

```
    cout <= temp_sum(4);
end Behavioral;
```

4 位半加器

4 位半加器其编码形式可如下所示：

Verilog 示例：

```
module add(a,b,sum,cout);
input [3:0] a,b;
output [3:0] sum;
output cout;
assign {cout,sum}=a+b;
endmodule
```

VHDL 示例：

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity add is
    Port ( a : in   STD_LOGIC_VECTOR (3 downto 0);
           b : in   STD_LOGIC_VECTOR (3 downto 0);
           sum : out   STD_LOGIC_VECTOR (3 downto 0);
           cout : out   STD_LOGIC);
end add;

architecture Behavioral of add is
    signal temp_sum : STD_LOGIC_VECTOR (4 downto 0);
begin
    temp_sum <= ('0' & a) + ('0' & b);
    sum <= temp_sum(3 downto 0);
    cout <= temp_sum(4);
end Behavioral;
```

3.2.2 SUB 功能

Verilog 示例：

```
module sub(a,b,sub);
input [3:0] a,b;
output [3:0] sub;
assign sub=a-b;
endmodule
```

VHDL 示例：

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity sub is
    Port ( a : in    STD_LOGIC_VECTOR (3 downto 0);
           b : in    STD_LOGIC_VECTOR (3 downto 0);
           sub : out    STD_LOGIC_VECTOR (3 downto 0)
    );
end sub;

architecture Behavioral of sub is
begin
    sub<=a-b;
end Behavioral;
```

3.2.3 ADDSUB 功能

Verilog 示例：

```
module addsub(a,b,c,sum);
input [3:0] a,b;
input c;
output [3:0] sum;
assign sum=c?(a-b):(a+b);
endmodule
```

VHDL 示例：

```
library IEEE;
```

```
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity addsub is
    Port (
        a : in   STD_LOGIC_VECTOR (3 downto 0);
        b : in   STD_LOGIC_VECTOR (3 downto 0);
        c : in   STD_LOGIC;
        sum : out   STD_LOGIC_VECTOR (3 downto 0)
    );
end addsub;

architecture Behavioral of addsub is
begin
    process(a,b,c)begin
        if(c='1')then
            sum<=a-b;
        else
            sum<=a+b;
        end if;
    end process;
end Behavioral;
```

3.2.4 NE 功能

Verilog 示例：

```
module ne(a, b, cout);
input [11:0] a, b;
output cout;
assign cout = (a != b) ? 1'b1 : 1'b0;
endmodule
```

VHDL 示例：

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
entity ne is
    Port (
        a : in    STD_LOGIC_VECTOR (11 downto 0);
        b : in    STD_LOGIC_VECTOR (11 downto 0);
        cout : out    STD_LOGIC
    );
end ne;

architecture Behavioral of ne is
begin
    process(a,b)begin
        if(a=b)then
            cout<='0';
        else
            cout<='1';
        end if;
    end process;
end Behavioral;
```

3.3 FF

触发器是时序电路中常用的基本元件，FPGA 内部的时序逻辑都可通过 FF 结构实现，常用的 FF 有 DFF、DFFE、DFFS、DFFSE 等，其区别在于复位方式、触发方式等方面。Arora V 的触发器原语仅只有 DFFSE、DFFRE、DFFPE、DFFCE。本节以 DFFSE、DFFRE、DFFPE、DFFCE 为例介绍寄存器的实现，其他寄存器类型原语的实现可参考这几类寄存器。编码定义 reg 信号时，建议添加初始值，不同类型寄存器的初始值可参考 *UG288，Gowin 可配置功能单元(CFU)用户指南*。

3.3.1 DFFSE

Verilog 示例：

```
module dffse_init1 (clk, d, ce, set, q );
input clk, d, ce, set;
output    reg q=1'b1;
always @(posedge clk)begin
    if (set)begin
        q <= 1'b1;
```

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

VHDL 示例:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity dffse_init1 IS
   PORT(q:OUT std_logic;
        d :IN    std_logic;
        set:IN    std_logic;
        ce:IN    std_logic;
        clk:IN std_logic
);
end dffse_init1;
ARCHITECTURE rtl OF dffse_init1 IS
begin
    process(clk)
    begin
        if(clk 'event and clk ='1') then
            if(set='1')then
                q<='1';
            elsif(ce='1')then
                q<=d;
            end if;
        end if;
    end process;
end;
```

3.3.2 DFFRE

Verilog 示例：

```
module dffre_init1 (clk, d, ce, rst, q );
input clk, d, ce, rst;
output    reg q= 1'b0;
always @(posedge clk)begin
      if (rst)begin
            q <= 1'b0;
      end
      else begin
            if (ce)begin
                  q <= d;
            end
      end
 end
endmodule
```

VHDL 示例：

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity dffre_init1 IS
   PORT(q:OUT std_logic;
          d :IN    std_logic;
          rst:IN    std_logic;
          ce: IN    std_logic;
          clk:IN std_logic
);
end dffre_init1;
ARCHITECTURE rtl OF dffre_init1 IS
begin
      process(clk,rst,ce)
      begin
            if(clk 'event and clk ='1') then
                  if(rst='1')then
                        q<='0';
```

```
            elsif(ce='1')then
                q<=D;
            end if;
        end if;
    end process;
end;
```

3.3.3 DFFPE

Verilog 示例：

```
module dffpe_test (clk, d, ce, preset, q );
input clk, d, ce, preset;
output   reg q= 1'b1;
always @(posedge clk or posedge preset )begin
    if (preset)begin
        q <= 1'b1;
    end
    else begin
        if (ce)begin
            q <= d;
        end
    end
 end
endmodule
```

VHDL 示例：

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

entity dffpe_test IS

PORT(
        q:OUT std_logic;
        d: IN   std_logic;
        clk:IN std_logic;
        ce: IN std_logic;
        preset:IN std_logic
```

```
);
end dffpe_test;
ARCHITECTURE rtl OF dffpe_test IS
begin
    process(clk,preset,ce)
    begin
        if(preset='1')then
            q<='1';
        elsif(clk 'event and clk ='1') then
            if(ce='1') then
                q<=D;
            end if;
        end if;
    end process;
end;
```

3.3.4 DFFCE

Verilog 示例：

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

VHDL 示例：

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
entity dffce_test IS
   PORT(q:OUT std_logic;
          d :   std_logic;
          clk:IN std_logic;
          ce: IN std_logic;
          clear:IN std_logic
);
end dffce_test;
ARCHITECTURE rtl OF dffce_test IS
begin
      process(clk,clear,ce)
      begin
           if(clear='1')then
                q<='0';
           elsif(clk 'event and clk ='1') then
                if(ce='1') then
                     q<=d;
                end if;
           end if;
      end process;
 end;
```

3.4 LATCH

锁存器是一种对电平触发的存储单元电路，其可在特定输入电平作用下改变状态。Arora V 的锁存器原语只有 DLCE、DLPE。本节以 DLCE、DLPE 为例介绍锁存器的实现，其他锁存器类型原语的实现可参考这几类锁存器。编码定义 reg 信号时，建议添加初始值，不同类型锁存器的初始值可参考 *UG288，Gowin 可配置功能单元(CFU)用户指南*。

3.4.1 DLCE

Verilog 示例：

```
module rtl_DLCE (Q, D, G, CE, CLEAR);
input D, G, CLEAR, CE;
output reg Q=1'b0;
always @(D or G or CLEAR or CE ) begin
    if (CLEAR)begin
        Q <= 1'b0;
    end
    else begin
        if (G && CE)begin
            Q <= D;
        end
    end
end
endmodule
```

VHDL 示例：

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY rtl_DLCE IS
PORT(D:IN STD_LOGIC;
     CLEAR:IN STD_LOGIC;
     G:IN STD_LOGIC;
     Q:OUT STD_LOGIC;
     CE:IN STD_LOGIC
);
END rtl_DLCE;
ARCHITECTURE rtl OF rtl_DLCE IS
```

```
BEGIN
    PROCESS(G,D,CE,CLEAR)
    BEGIN
        IF (CLEAR='1')THEN
            Q <= '0';
        ELSIF (G='1'AND CE='1')THEN
            Q<= D;
        END IF;
    END PROCESS;
END;
```

3.4.2 DLPE

Verilog 示例：

```
module rtl_DLPE (Q, D, G, CE, PRESET);
input D, G, PRESET, CE;
output reg Q= 1'b1;
always @(D or G or PRESET or CE ) begin
    if(PRESET)begin
        Q <= 1'b1;
    end
    else begin
        if (G && CE)begin
            Q <= D;
        end
    end
end
endmodule
```

VHDL 示例：

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY rtl_DLPE IS
PORT(D:IN STD_LOGIC;
     G:IN STD_LOGIC;
     PRESET:IN STD_LOGIC;
     Q:OUT STD_LOGIC;
```

```
        CE:IN STD_LOGIC
);
END rtl_DLPE;
ARCHITECTURE BEH OF rtl_DLPE IS
BEGIN
    PROCESS(G,D,PRESET,CE)
    BEGIN
        IF (PRESET='1')THEN
            Q <= '1';
        ELSIF(G='1'AND CE='1')THEN
            Q<=D;
        END IF;
    END PROCESS;
END;
```

4 BSRAM 编码规范

BSRAM 块状静态随机存储器，具有静态存取功能。根据配置模式，可分为单端口模式（SP/SPX9）、双端口模式（DPB/DPX9B）、伪双端口（SDPB/SDPX9B）和只读模式（pROM/pROMX9）。

4.1 DPB/DPX9B

DPB/DPX9B 的存储空间分别为 16Kbit/18Kbit，其工作模式为双端口模式，端口 A 和端口 B 均可分别独立实现读/写操作，可支持 2 种读模式（bypass 模式和 pipeline 模式）和 2 种写模式（normal 模式、write-through 模式）。本节以读地址是否经过 register、初值读取等方面进行实现说明。

4.1.1 读地址经过 register

读地址经过 register 时，仅支持读地址 register 无控制信号控制的情况。以 write-through 、bypass、同步复位模式的 DPB 为例介绍其实现，其编码形式可如下所示：

Verilog 示例：

```
module top(data_outa, data_ina, addra, clka,cea, wrea,data_outb, data_inb, addrb, clkb,ceb, wreb);
output [7:0]data_outa,data_outb;
input [7:0]data_ina,data_inb;
input [10:0]addra,addrb;
input clka,wrea,cea;
input clkb,wreb,ceb;
reg [7:0] mem [2047:0];
reg [10:0]addra_reg,addrb_reg;

always@(posedge clka)begin
    addra_reg<=addra;
```

```
end
always @(posedge clka)begin
    if (cea & wrea) begin
        mem[addra] <= data_ina;
    end
end
assign data_outa = mem[addra_reg];

always@(posedge clkb)begin
    addrb_reg<=addrb;
end



always @(posedge clkb)begin
    if (ceb & wreb) begin
        mem[addrb] <= data_inb;
    end
end
assign data_outb = mem[addrb_reg];
endmodule
```

VHDL 示例：

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

entity top is
    generic(
        DATA_WIDTH : integer := 8;
        ADDRESS_WIDTH : integer := 11
    );
    port (
        data_ina, data_inb : in STD_LOGIC_VECTOR(DATA_WIDTH
- 1 downto 0);
        addra, addrb: in STD_LOGIC_VECTOR(ADDRESS_WIDTH
- 1 downto 0);
```

```
        clka, cea, wrea, clkb, ceb, wreb: in STD_LOGIC;
        data_outa, data_outb : out
STD_LOGIC_VECTOR(DATA_WIDTH - 1 downto 0)
    );
end top;

architecture rtl of top is
    type mem_type is array (2**ADDRESS_WIDTH - 1 downto 0) of
STD_LOGIC_VECTOR (DATA_WIDTH - 1 downto 0);
    signal mem : mem_type;
    signal
addra_reg,addrb_reg:STD_LOGIC_VECTOR(ADDRESS_WIDTH - 1
downto 0);
begin
    process(clka)
    begin
        if (rising_edge(clka)) then
            addra_reg<=addra;
        end if;
    end process;
    process(clka)
    begin
        if (rising_edge(clka)) then
            if(cea = '1' and wrea = '1') then
                mem(conv_integer(addra)) <= data_ina;
            end if;
        end if;
    end process;

    data_outa <= mem(conv_integer(addra_reg));


    process(clkb)
    begin
        if (rising_edge(clkb)) then
```

```
            addrb_reg<=addrb;
        end if;
    end process;


    data_outb <= mem(conv_integer(addrb_reg));


    process(clkb)
    begin
        if (rising_edge(clkb)) then
            if(ceb = '1' and wreb = '1') then
                mem(conv_integer(addrb)) <= data_inb;
            end if;
        end if;
    end process;
end rtl;
```

4.1.2 读地址不经过 Register，输出经过 Register

读地址不经过 register 时，输出必须经过 register，经过一级 register 时为 bypass 模式；经过两级 register 时为 pipeline 模式。以 normal、pipeline、同步复位模式的 DPB 为例介绍其实现，其编码形式可如下所示：

Verilog 示例：

```
module top(data_outa, data_ina, addra, clka, cea, ocea, wrea, rsta, data_outb, data_inb, addrb, clkb, ceb, oceb, wreb, rstb);
output reg [15:0]data_outa,data_outb;
input [15:0]data_ina,data_inb;
input [9:0]addra,addrb;
input clka,wrea,cea,ocea,rsta;
input clkb,wreb,ceb,oceb,rstb;
reg [15:0] mem [1023:0];
reg [15:0] data_outa_reg=16'h0000;
reg [15:0] data_outb_reg=16'h0000;
always@(posedge clka)begin
    if(rsta)begin
```

```
            data_outa <= 0;
        end
        else begin
            if (ocea)begin
                data_outa <= data_outa_reg;
            end
        end
    end
    always@(posedge clka)begin
        if(rsta)begin
            data_outa_reg <= 0;
        end
        else begin
            if(cea & !wrea)begin
                data_outa_reg <= mem[addra];
            end
        end
    end
    always @(posedge clka)begin
        if (cea & wrea) begin
            mem[addra] <= data_ina;
        end
    end

    always@(posedge clkb )begin
        if(rstb)begin
            data_outb <= 0;
        end
        else begin
            if (oceb)begin
                data_outb <= data_outb_reg;
            end
        end
    end
```

```
always@(posedge clkb )begin
    if(rstb)begin
        data_outb_reg <= 0;
    end
    else begin
        if(ceb & !wreb)begin
            data_outb_reg <= mem[addrb];
        end
    end
end

always @(posedge clkb)begin
    if (ceb & wreb) begin
        mem[addrb] <= data_inb;
    end
end
endmodule
```

VHDL 示例：

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

entity top is
    generic(
        DATA_WIDTH : integer := 16;
        ADDRESS_WIDTH : integer := 10
    );
    port (
        data_ina, data_inb : in STD_LOGIC_VECTOR(DATA_WIDTH
- 1 downto 0);
        addra, addrb: in STD_LOGIC_VECTOR(ADDRESS_WIDTH
- 1 downto 0);
        clka, cea, ocea,wrea,rsta, clkb, ceb, oceb,wreb,rstb: in
STD_LOGIC;
        data_outa, data_outb : out
```

```
STD_LOGIC_VECTOR(DATA_WIDTH - 1 downto 0)
        );
    end top;

    architecture rtl of top is
        type mem_type is array (2**ADDRESS_WIDTH - 1 downto 0) of
STD_LOGIC_VECTOR (DATA_WIDTH - 1 downto 0);
        signal mem : mem_type;
        signal data_outa_reg, data_outb_reg :
STD_LOGIC_VECTOR(DATA_WIDTH - 1 downto 0);
    begin
        process(clka)
        begin
            if (rising_edge(clka)) then
                if(cea = '1' and wrea = '1') then
                    mem(conv_integer(addra)) <= data_ina;
                end if;
            end if;
        end process;
        process(clka)
        begin
            if (rising_edge(clka)) then
                if(rsta='1')then
                    data_outa_reg<=(others=>'0');
                else
                    if(cea = '1' and wrea = '0') then
                        data_outa_reg <= mem(conv_integer(addra));
                    end if;
                end if;
            end if;
        end process;
        process(clka)
        begin
            if (rising_edge(clka)) then
                if(rsta='1')then
```

```
                data_outa<=(others=>'0');
            else
                if(ocea = '1') then
                    data_outa <= data_outa_reg;
                end if;
            end if;
        end if;
    end process;

    process (clkb)
    begin
        if (rising_edge(clkb)) then
            if(rstb='1')then
                data_outb <= (others=>'0');
            else
                if(oceb = '1') then
                    data_outb <= data_outb_reg;
                end if;
            end if;
        end if;
    end process;
    process (clkb)
    begin
        if (rising_edge(clkb)) then
            if(rstb='1')then
                data_outb_reg <= (others=>'0');
            else
                if(ceb = '1' and wreb = '0') then
                    data_outb_reg <= mem(conv_integer(addrb));
                end if;
            end if;
        end if;
    end process;
    process(clkb)
```

```
    begin
        if (rising_edge(clkb)) then
            if(ceb = '1' and wreb = '1') then
                mem(conv_integer(addrb)) <= data_inb;
            end if;
        end if;
    end process;
end rtl;
```

4.1.3 memory 定义时赋初值

memory 定义时赋初值仅 GowinSynthesis 支持，且 Verilog language 需选择 system Verilog。以 normal、bypass、同步复位模式的 DPB 为例介绍其实现，其编码形式可如下所示：

Verilog 示例：

```
module top(data_outa, data_ina, addra, clka,cea, wrea,rsta,data_outb, data_inb, addrb, clkb,ceb, wreb,rstb);
output [3:0]data_outa,data_outb;
input [3:0]data_ina,data_inb;
input [2:0]addra,addrb;
input clka,wrea,cea,rsta;
input clkb,wreb,ceb,rstb;
reg [3:0] mem [7:0]={4'h1,4'h2,4'h3,4'h4,4'h5,4'h6,4'h7,4'h8};

reg [3:0] data_outa=4'h0;
reg [3:0] data_outb=4'h0;
always@(posedge clka)begin
    if(rsta)begin
        data_outa <= 0;
    end
    else begin
        if(cea & !wrea)begin
            data_outa <= mem[addra];
        end
    end
end
```

```
always @(posedge clka)begin
    if (cea & wrea) begin
        mem[addra] <= data_ina;
    end
end

always@(posedge clkb)begin
    if(rstb)begin
        data_outb <= 0;
    end
    else begin
        if(ceb & !wreb)begin
            data_outb <= mem[addrb];
        end
    end
end

always @(posedge clkb)begin
    if (ceb & wreb) begin
        mem[addrb] <= data_inb;
    end
end
endmodule
```

VHDL 示例：

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

entity top is
    generic(
        DATA_WIDTH : integer := 4;
        ADDRESS_WIDTH : integer := 3
    );
```

```
        port (
            data_ina, data_inb : in STD_LOGIC_VECTOR(DATA_WIDTH
- 1 downto 0);
            addra, addrb: in STD_LOGIC_VECTOR(ADDRESS_WIDTH
- 1 downto 0);
            clka, cea, wrea,rsta, clkb, ceb, wreb,rstb: in STD_LOGIC;
            data_outa, data_outb : out
STD_LOGIC_VECTOR(DATA_WIDTH - 1 downto 0)
        );
    end top;

    architecture rtl of top is
        type mem_type is array (2**ADDRESS_WIDTH - 1 downto 0) of
STD_LOGIC_VECTOR (DATA_WIDTH - 1 downto 0);
        signal mem :
mem_type:=("0001","0010","0011","0100","0101","0110","0111","1000");
    begin
        process(clka)
        begin
            if (rising_edge(clka)) then
                if(cea = '1' and wrea = '1') then
                    mem(conv_integer(addra)) <= data_ina;
                end if;
            end if;
        end process;
        process(clka)
        begin
            if (rising_edge(clka)) then
                if(rsta='1')then
                    data_outa<=(others=>'0');
                else
                    if(cea = '1' and wrea = '0') then
                        data_outa <= mem(conv_integer(addra));
                    end if;
                end if;
            end if;
```

```
    end process;


    process (clkb)
    begin
        if (rising_edge(clkb)) then
            if(rstb='1')then
                data_outb <= (others=>'0');
            else
                if(ceb = '1' and wreb = '0') then
                    data_outb <= mem(conv_integer(addrb));
                end if;
            end if;
        end if;
    end process;
    process(clkb)
    begin
        if (rising_edge(clkb)) then
            if(ceb = '1' and wreb = '1') then
                mem(conv_integer(addrb)) <= data_inb;
            end if;
        end if;
    end process;
end rtl;
```

4.1.4 readmemb/readmemh 方式赋初值

readmemb/ readmemh 方式赋值在进行使用时需注意路径的书写，请以‘/’作为路径分隔符。以 normal、bypass、同步复位模式的 DPB 为例介绍其实现，其编码形式可如下所示：

Verilog 示例：

```
module top(data_outa, data_ina, addra, clka,cea, wrea,rsta,data_outb, data_inb, addrb, clkb,ceb, wreb,rstb);
output [3:0]data_outa,data_outb;
input [3:0]data_ina,data_inb;
input [2:0]addra,addrb;
```

```
input clka,wrea,cea,rsta;
input clkb,wreb,ceb,rstb;
reg [3:0] mem [7:0];
reg [3:0] data_outa=4'h0;
reg [3:0] data_outb=4'h0;

initial begin
    $readmemb ("E:/dpb.mi", mem);
end

always@(posedge clka)begin
    if(rsta)begin
        data_outa <= 0;
    end
    else begin
        if(cea & !wrea)begin
            data_outa <= mem[addra];
        end
    end
end

always @(posedge clka)begin
    if (cea & wrea) begin
        mem[addra] <= data_ina;
    end
end

always@(posedge clkb)begin
    if(rstb)begin
        data_outb <= 0;
    end
    else begin
        if(ceb & !wreb)begin
            data_outb <= mem[addrb];
```

```
            end
        end
    end

    always @(posedge clkb)begin
        if (ceb & wreb) begin
            mem[addrb] <= data_inb;
        end
    end
    endmodule
```

dpb.mi 书写形式如下所示：

```
0001
0010
0011
0100
0101
0110
0111
1000
```

注！

如果以 windows 系统支持的路径分隔符‘\’，需要加转义字符，如 E:\\dpb.mi。

4.1.5 byte-enable 功能

byte-enable 功能的推断仅 Arora V BSRAM 支持，其编码形式可如下所示：

Verilog 示例：

```
module Gowin_DPB (douta, doutb, clka, cea, reseta, wrea, clkb, ceb, resetb, wreb, ada, dina, adb, dinb, byte_ena, byte_enb);
    output reg [15:0] douta;
    output reg [15:0] doutb;
    input clka,cea,reseta,wrea;
    input clkb,ceb,resetb,wreb;
    input [9:0] ada,adb;
    input [15:0] dina,dinb;
```

```
input [1:0] byte_ena,byte_enb;

reg[15:0]mem[2**10-1:0];

always@(posedge clka)begin
    if(cea & wrea)begin
        if(byte_ena[0])begin
            mem[ada][7:0]<=dina[7:0];
        end
        if(byte_ena[1])begin
            mem[ada][15:8]<=dina[15:8];
        end
    end
end
always@(posedge clka)begin
    if(reseta)begin
        douta<=0;
    end
    else begin
        if(cea & !wrea)begin
            douta<=mem[ada];
        end
    end
end


always@(posedge clkb)begin
    if(ceb & wreb)begin
        if(byte_enb[0])begin
            mem[adb][7:0]<=dinb[7:0];
        end
        if(byte_enb[1])begin
            mem[adb][15:8]<=dinb[15:8];
        end
```

```
        end
    end
    always@(posedge clkb)begin
        if(resetb)begin
            doutb<=0;
        end
        else begin
            if(ceb & !wreb)begin
                doutb<=mem[adb];
            end
        end
    end

    endmodule
```

VHDL 示例：

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

    entity ram is
        generic(
            DATA_WIDTH : integer := 16;
            ADDRESS_WIDTH : integer := 10
        );
        port (
            dina, dinb : in STD_LOGIC_VECTOR(DATA_WIDTH - 1
downto 0);
            addra, addrb: in STD_LOGIC_VECTOR(ADDRESS_WIDTH
- 1 downto 0);
            clka, cea, wrea,rsta, clkb, ceb, wreb,rstb: in STD_LOGIC;
            byte_ena,byte_enb : in STD_LOGIC_VECTOR(1 downto 0);
            douta, doutb : out STD_LOGIC_VECTOR(DATA_WIDTH - 1
downto 0)
        );
    end ram;
```

```
    architecture rtl of ram is
        type mem_type is array (2**ADDRESS_WIDTH - 1 downto 0) of
STD_LOGIC_VECTOR (DATA_WIDTH - 1 downto 0);
        signal mem : mem_type;
    begin
        process(clka)
        begin
            if (rising_edge(clka)) then
                if(cea = '1' and wrea = '1') then
                    if(byte_ena(0)='1')then
                        mem(conv_integer(addra))(7 downto 0) <=
dina(7 downto 0) ;
                    end if;
                    if(byte_ena(1)='1')then
                        mem(conv_integer(addra))(15 downto 8) <=
dina(15 downto 8) ;
                    end if;
                end if;
            end if;
        end process;
        process(clka)
        begin
            if (rising_edge(clka)) then
                if(rsta='1')then
                    douta<=(others=>'0');
                else
                    if(cea = '1' and wrea = '0') then
                        douta <= mem(conv_integer(addra));
                    end if;
                end if;
            end if;
        end process;

        process (clkb)
```

```
    begin
        if (rising_edge(clkb)) then
            if(rstb='1')then
                doutb <= (others=>'0');
            else
                if(ceb = '1' and wreb = '0') then
                    doutb <= mem(conv_integer(addrb));
                end if;
            end if;
        end if;
    end process;

    process(clkb)
    begin
        if (rising_edge(clkb)) then
            if(ceb = '1' and wreb = '1') then
                if(byte_enb(0)='1')then
                    mem(conv_integer(addrb))(7 downto 0) <=
dinb(7 downto 0);
                end if;
                if(byte_enb(1)='1')then
                    mem(conv_integer(addrb))(15 downto 8) <=
dinb(15 downto 8);
                end if;
            end if;
        end if;
    end process;
end rtl;
```

4.2 SP/SPX9

SP/SPX9 存储空间为 16Kbit/18Kbit，其工作模式为单端口模式，由一个时钟控制单端口的读/写操作，可支持 2 种读模式（bypass 模式和 pipeline 模式）和 3 种写模式（normal 模式、write-through 模式和 read-before-write 模式）。若综合出 SP/SPX9，memory（除移位寄存器形式）需满足至少满足下述条件之一：

1. 数据位宽*地址深度>=1024

2. 使用 syn_ramstyle = "block_ram"。

本节以读地址是否经过 register、初值读取等方面进行实现说明。

注！

所有 Arora V 器件不支持 read-before-write 写模式编码风格。

4.2.1 读地址经过 register

读地址经过 register 时，仅支持读地址 register 无控制信号控制的情况，该类形式会综合出 write-through 模式的 SP。以输出不经过 register 为例介绍其实现，其编码形式可如下所示：

Verilog 示例：

```
module top(data_out, data_in, addr, clk,ce, wre);
output [9:0]data_out;
input [9:0]data_in;
input [9:0]addr;
input clk,wre,ce;
reg [9:0] mem [1023:0];
reg [9:0]addr_reg=10'h000;

always@(posedge clk)begin
    addr_reg<=addr;
end
always @(posedge clk)begin
    if (ce & wre) begin
        mem[addr] <= data_in;
    end
end
assign data_out = mem[addr_reg];
endmodule
```

VHDL 示例：

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

entity top is
generic( DATA_WIDTH : integer := 10;
```

```
    ADDRESS_WIDTH : integer := 10);

        port (
            data_in : in STD_LOGIC_VECTOR(DATA_WIDTH - 1 downto
0);
            addr: in STD_LOGIC_VECTOR(ADDRESS_WIDTH - 1
downto 0);
            clk,ce, wre: in STD_LOGIC;
            data_out : out STD_LOGIC_VECTOR(DATA_WIDTH - 1
downto 0));
    end top;

    architecture rtl of top is
        type mem_type is array (2**ADDRESS_WIDTH - 1 downto 0) of
        STD_LOGIC_VECTOR (DATA_WIDTH - 1 downto 0);
        signal mem : mem_type;
        signal addr_reg : STD_LOGIC_VECTOR(ADDRESS_WIDTH - 1
downto 0);
    begin

        data_out <= mem(conv_integer(addr_reg));

        process (clk)
        begin
            if (rising_edge(clk)) then
                addr_reg<=addr;
            end if;
        end process;

        process (clk)
        begin
            if (rising_edge(clk)) then
                if(ce ='1'and wre = '1')then
                    mem(conv_integer(addr)) <= data_in;
                end if;
```

```
        end if;
    end process;
end rtl;
```

4.2.2 读地址不经过 Register，输出经过 Register

读地址不经过 register 时，输出必须经过 register，经过一级 register 时为 bypass 模式；经过两级 register 时为 pipeline 模式。以 write-through、bypass、同步复位模式的 SPX9 为例介绍其实现，其编码形式可如下所示：

Verilog 示例：

```
module top(data_out, data_in, addr, clk,ce, wre,rst);
output reg [17:0]data_out=18'h00000;
input [17:0]data_in;
input [9:0]addr;
input clk,wre,ce,rst;
reg [17:0] mem [1023:0];
always@(posedge clk )begin
    if(rst)begin
        data_out <= 0;
    end
    else begin
        if(ce & wre)begin
            data_out <= data_in;
        end
        else begin
            if (ce & !wre)begin
                data_out <= mem[addr];
            end
        end
    end
end
always @(posedge clk)begin
    if (ce & wre)begin
        mem[addr] <= data_in;
    end
end
```

```
endmodule
```

VHDL 示例：

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

entity ram is

generic( DATA_WIDTH : integer := 18;
ADDRESS_WIDTH : integer := 10);

    port (
        data_in : in STD_LOGIC_VECTOR(DATA_WIDTH - 1 downto
0);
        addr: in STD_LOGIC_VECTOR(ADDRESS_WIDTH - 1
downto 0);
        clk, ce, wre,rst: in STD_LOGIC;
        data_out : out STD_LOGIC_VECTOR(DATA_WIDTH - 1
downto 0));
end ram;

architecture rtl of ram is
    type mem_type is array (2**ADDRESS_WIDTH - 1 downto 0) of
    STD_LOGIC_VECTOR (DATA_WIDTH - 1 downto 0);
    signal mem : mem_type;
begin
    process (clk)
    begin
        if (rising_edge(clk)) then
            if(ce = '1'and wre='1')then
                mem(conv_integer(addr)) <= data_in;
            end if;
        end if;
    end process;
```

```
process (clk)
begin
    if (rising_edge(clk)) then
        if(rst = '1')then
            data_out<=(others=>'0');
        else
            if(ce = '1')then
                if(wre='1')then
                    data_out <= data_in;
                else
                    data_out <= mem(conv_integer(addr));
                end if;
            end if;
        end if;
    end process;

end rtl;
```

4.2.3 memory 定义时赋初值

memory 定义时赋初值需 Verilog language 选择 system Verilog。以 normal、bypass、同步复位模式的 SP 为例介绍其实现，其编码形式可如下所示：

Verilog 示例：

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

```
        end
        else begin
            if(ce & !wre)begin
                data_out <= mem[addr];
            end
        end
    end
    always @(posedge clk)begin
        if (ce & wre)begin
            mem[addr] <= data_in;
        end
    end
    endmodule
```

VHDL 示例：

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

    entity top is
    generic( DATA_WIDTH : integer := 16;
    ADDRESS_WIDTH : integer := 3);

        port (
            data_in : in STD_LOGIC_VECTOR(DATA_WIDTH - 1 downto
0);
            addr: in STD_LOGIC_VECTOR(ADDRESS_WIDTH - 1
downto 0);
            clk,ce, wre,rst: in STD_LOGIC;
            data_out : out STD_LOGIC_VECTOR(DATA_WIDTH - 1
downto 0));
    end top;

    architecture rtl of top is
        type mem_type is array (2**ADDRESS_WIDTH - 1 downto 0) of
        STD_LOGIC_VECTOR (DATA_WIDTH - 1 downto 0);
```

```
        signal mem :
mem_type :=("0000000100100011","0100010101100111","100010011010
1011","1100110111101111","0000000101000111","0000001001011000","01
11100010011010","0101011001111000");
        attribute syn_ramstyle:string;
        attribute syn_ramstyle of mem : signal is "block_ram";
    begin
        process (clk)
        begin
            if (rising_edge(clk)) then
                if(rst='1')then
                    data_out <=(others=>'0');
                else
                    if(ce ='1' and wre = '0')then
                        data_out <= mem(conv_integer(addr));
                    end if;
                end if;
            end if;
        end process;

        process (clk)
        begin
            if (rising_edge(clk)) then
                if(ce ='1'and wre = '1')then
                    mem(conv_integer(addr)) <= data_in;
                end if;
            end if;
        end process;
    end rtl;
```

4.2.4 readmemb/readmemh 方式赋初值

readmemb/ readmemh 方式赋值使用时需注意路径的书写，请以‘/’作为路径分隔符。以 normal、pipeline、同步复位模式的 SP 为例介绍其实现，其编码形式可如下所示：

Verilog 示例：

```
module top(data_out, data_in, addr, clk, ce, oce, wre, rst)/*synthesis
```

```
syn_ramstyle="block_ram"*/;
    output reg [7:0]data_out=8'h00;
    input [7:0]data_in;
    input [2:0]addr;
    input clk,wre,ce,oce,rst;
    reg [7:0] mem [7:0];
    reg [7:0] data_out_reg=8'h00;
    initial begin
        $readmemh ("E:/sp.mi", mem);
    end

    always@(posedge clk )begin
        if(rst)begin
            data_out <= 0;
        end
        else begin
            if(oce)begin
                data_out <= data_out_reg;
            end
        end
    end
    always@(posedge clk)begin
        if(rst)begin
            data_out_reg <= 0;
        end
        else begin
            if(ce & !wre)begin
                data_out_reg <= mem[addr];
            end
        end
    end
    always @(posedge clk)begin
        if (ce & wre) begin
            mem[addr] <= data_in;
```

```
    end
end
endmodule
```

sp.mi 书写格式如下：

```
12
34
56
78
9a
bc
de
ff
```

4.2.5 byte-enable 功能

byte-enable 功能的推断仅 Arora V BSRAM 支持，其编码形式可如下所示:

Verilog 示例：

```
module Gowin_SP (dout, clk, ce, reset, wre, ad, din, byte_en);
output reg[35:0] dout;
input clk,ce,reset,wre;
input [7:0] ad;
input [35:0] din;
input [3:0] byte_en;

reg[35:0]mem[2**8-1:0];

always@(posedge clk)begin
    if(ce & wre)begin
        if(byte_en[0])begin
            mem[ad][8:0]<=din[8:0];
        end
        if(byte_en[1])begin
            mem[ad][17:9]<=din[17:9];
        end
```

```
            if(byte_en[2])begin
                mem[ad][26:18]<=din[26:18];
            end
            if(byte_en[3])begin
                mem[ad][35:27]<=din[35:27];
            end
        end
    end

    always@(posedge clk)begin
        if(reset)begin
            dout<=0;
        end
        else begin
            if(ce & !wre)begin
                dout<=mem[ad];
            end
        end
    end
    endmodule
```

VHDL 示例：

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

entity Gowin_SP is
    generic(
        DATA_WIDTH : integer := 36;
        ADDRESS_WIDTH : integer := 8
    );
    port (
        din : in STD_LOGIC_VECTOR(DATA_WIDTH - 1 downto 0);
        ad: in STD_LOGIC_VECTOR(ADDRESS_WIDTH - 1 downto
0);
```

```
        clk,ce, wre,reset: in STD_LOGIC;
        byte_en : in STD_LOGIC_VECTOR(3 downto 0);
        dout : out STD_LOGIC_VECTOR(DATA_WIDTH - 1 downto
0)
    );
end Gowin_SP;

architecture rtl of Gowin_SP is
    type mem_type is array (2**ADDRESS_WIDTH - 1 downto 0) of
    STD_LOGIC_VECTOR (DATA_WIDTH - 1 downto 0);
    signal mem : mem_type ;
begin
    process (clk)
    begin
        if (rising_edge(clk)) then
            if(reset='1')then
                dout <=(others=>'0');
            else
                if(ce ='1' and wre = '0')then
                    dout <= mem(conv_integer(ad));
                end if;
            end if;
        end if;
    end process;

    process (clk)
    begin
        if (rising_edge(clk)) then
            if(ce ='1'and wre = '1')then
                if(byte_en(0)='1')then
                    mem(conv_integer(ad))(8 downto 0) <= din(8
downto 0);
                end if;
                if(byte_en(1)='1')then
                    mem(conv_integer(ad))(17 downto 9) <= din(17
```

```
downto 9);
                        end if;
                        if(byte_en(2)='1')then
                            mem(conv_integer(ad))(26 downto 18) <=
din(26 downto 18);
                        end if;
                        if(byte_en(3)='1')then
                            mem(conv_integer(ad))(35 downto 27) <=
din(35 downto 27);
                        end if;
                    end if;
                end if;
            end process;
        end rtl;
```

4.3 SDPB/SDPX9B

SDPB/SDPX9B 存储空间分别为 16Kbit/18Kbit，工作模式为伪双端口模式，可支持 2 种读模式（bypass 模式和 pipeline 模式）和 1 种写模式（normal 模式）。若综合出 SDPB/SDPX9B，memory（除移位寄存器形式）需满足下述条件之一：

1. 数据位宽*地址深度>=1024；
2. 使用 syn_ramstyle = "block_ram"。

4.3.1 memory 无初值

以 bypass、同步复位模式的 SDPB 为例介绍其实现，其编码形式可如下所示：

Verilog 示例：

```
module top(dout, din, ada, adb, clka, cea, clkb, ceb, resetb);
output reg[15:0]dout=16'h0000;
input [15:0]din;
input [9:0]ada, adb;
input clka, cea,clkb, ceb, resetb;
reg [15:0] mem [1023:0];

always @(posedge clka)begin
    if (cea )begin
```

```
            mem[ada] <= din;
        end
    end

    always@(posedge clkb)begin
        if(resetb)begin
            dout <= 0;
        end
        else if(ceb)begin
            dout <= mem[adb];
        end
    end

    endmodule
```

VHDL 示例：

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

    entity top is
    generic(
        DATA_WIDTH : integer := 16;
        ADDRESS_WIDTH : integer := 10
        );
        port (
            din : in STD_LOGIC_VECTOR(DATA_WIDTH - 1 downto 0);
            ada, adb: in STD_LOGIC_VECTOR(ADDRESS_WIDTH - 1
downto 0);
            clka, cea, clkb, ceb,resetb: in STD_LOGIC;
            dout : out STD_LOGIC_VECTOR(DATA_WIDTH - 1 downto
0));
    end top;

    architecture rtl of top is
        type mem_type is array (2**ADDRESS_WIDTH - 1 downto 0) of
```

```
        STD_LOGIC_VECTOR (DATA_WIDTH - 1 downto 0);
        signal mem : mem_type;
    begin
        process(clkb)
        begin
            if (rising_edge(clkb)) then
                if (resetb='1')then
                    dout <=(others=>'0');
                else
                    if(ceb = '1')then
                        dout <= mem(conv_integer(adb));
                    end if;
                end if;
            end if;
        end process;

        process (clka)
        begin
            if (rising_edge(clka)) then
                if(cea = '1')then
                    mem(conv_integer(ada)) <= din;
                end if;
            end if;
        end process;

    end rtl;
```

4.3.2 memory 定义时赋初值

以 pipeline、异步复位模式的 SDPB 为例介绍其实现形式，其编码形式可如下所示：

Verilog 示例：

```
module top(data_out, data_in, addra, addrb, clka, cea, clkb, ceb,oce, rstb)/*synthesis syn_ramstyle="block_ram"*/;
output reg[15:0]data_out=16'h0000;
input [15:0]data_in;
```

```
input [2:0]addra, addrb;
input clka, cea, clkb, ceb, rstb,oce;
reg [15:0] mem [7:0]={16'h0123,16'h4567,16'h89ab,16'hcdef,
16'h0147, 16'h0258,16'h789a,16'h5678};
reg [15:0] data_out_reg=16'h0000;

always @(posedge clka)begin
    if (cea )begin
        mem[addra] <= data_in;
    end
end

always@(posedge clkb or posedge rstb)begin
    if(rstb)begin
        data_out_reg <= 0;
    end
    else if(ceb)begin
        data_out_reg<= mem[addrb];
    end
end
always@(posedge clkb or posedge rstb)begin
    if(rstb)begin
        data_out <= 0;
    end
    else if(oce)begin
        data_out<= data_out_reg;
    end
end
endmodule
```

VHDL 示例：

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

```
entity top is
    generic(
        DATA_WIDTH : integer := 16;
        ADDRESS_WIDTH : integer := 3
    );
    port (
        data_in : in STD_LOGIC_VECTOR(DATA_WIDTH - 1 downto
0);
        addra, addrb: in STD_LOGIC_VECTOR(ADDRESS_WIDTH
- 1 downto 0);
        clka, cea, clkb, ceb,rstb: in STD_LOGIC;
        data_out : out STD_LOGIC_VECTOR(DATA_WIDTH - 1
downto 0));
end top;

architecture rtl of top is
    type mem_type is array (2**ADDRESS_WIDTH - 1 downto 0) of
    STD_LOGIC_VECTOR (DATA_WIDTH - 1 downto 0);
    signal mem :
mem_type:=("0000000100100011","0100010101100111","1000100110101
011","1100110111101111","0000000101000111","0000001001011000","011
1100010011010","0101011001111000");
    attribute syn_ramstyle:string;
    attribute syn_ramstyle of mem : signal is "block_ram";
begin
    process(clkb,rstb)
    begin
        if (rstb='1')then
            data_out <=(others=>'0');
        else
            if (rising_edge(clkb)) then
                if(ceb = '1')then
                    data_out <= mem(conv_integer(addrb));
                end if;
            end if;
        end if;
```

```
    end process;

    process (clka)
    begin
        if (rising_edge(clka)) then
            if(cea = '1')then
                mem(conv_integer(addra)) <= data_in;
            end if;
        end if;
    end process;

end rtl;
```

4.3.3 readmemb/readmemh 方式赋初值

readmemb/ readmemh 方式赋值在进行使用时需注意路径的书写，请以‘/’作为路径分隔符。以 bypass、异步复位模式的 SDPB 为例介绍其实现，其编码形式可如下所示：

Verilog 示例：

```
module top(dout, din, ada, adb, clka, cea, clkb, ceb, resetb)/*synthesis syn_ramstyle="block_ram"*/;
output reg[7:0]dout=8'h00;
input [7:0]din;
input [2:0]ada, adb;
input clka, cea,clkb, ceb, resetb;
reg [7:0] mem [7:0];
initial begin
    $readmemh ("E:/sdpb.mi", mem);
end

always @(posedge clka)begin
    if (cea )begin
        mem[ada] <= din;
    end
end
```

```
always@(posedge clkb or posedge resetb)begin
    if(resetb)begin
        dout <= 0;
    end
    else if(ceb)begin
        dout <= mem[adb];
    end
end

endmodule
```

sdpb.mi 书写格式如下：

```
12
34
56
78
9a
bc
de
ff
```

4.3.4 移位寄存器形式

移位寄存器形式综合出 BSRAM 需满足下述条件之一：

1. memory 深度>=5 且 memory 深度*数据位宽>256，且 memory 深度！=2 的 n 次方+1；
2. 添加属性约束 syn_srlstyle="block_ram"，且 memory 深度！=2 的 n 次方+1, memory 深度>=5。

以 bypass、同步复位模式的 SDPX9B 介绍其实现，其编码形式可如下所示：

Verilog 示例：

```
module   p_seqshift(clk, we, din, dout);
parameter width=18;
parameter depth=16;
input clk, we;
input [width-1:0] din;
```

```
output [width-1:0] dout;

reg [width-1:0] regBank[depth-1:0];

always @(posedge clk) begin
    if (we) begin
        regBank[depth-1:1] <= regBank[depth-2:0];
        regBank[0] <= din;
    end
end

assign dout = regBank[depth-1];
endmodule
```

VHDL 示例：

```
Library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity p_seqshift is
    generic(
        width: integer := 18;
        depth: integer := 16
    );
    port(
        clk : in std_logic;
        we : in std_logic;
        din : in std_logic_vector(width-1 downto 0);
        dout : out std_logic_vector(width-1 downto 0)
    );

end entity;
architecture Behavioral of p_seqshift is
    TYPE matrix_index is array (depth-1 downto 0) of
std_logic_vector(width-1 downto 0);
```

```
    signal regBank: matrix_index;
begin
    PROCESS(clk) begin
    IF(clk'event and clk = '1') then
        IF (we = '1') then
            for i in 1 to depth-1 loop
                regBank(i) <= regBank(i-1);
            end loop;
            regBank(0) <= din;
        end IF;
    end IF;
    end PROCESS;
    dout <= regBank(depth-1);
END Behavioral;
```

4.3.5 不对称类型

不对称类型综合出 SDPB。以 bypass、同步复位模式的 SDPB 为例介绍其实现，其编码形式可如下所示：

Verilog 示例：

```
module top(dout, din, ada, clka, cea, adb, clkb, ceb, rstb);
parameter adawidth = 8;
parameter diwidth = 6;
parameter adbwidth = 7;
parameter dowidth = 12;

output [dowidth-1:0]dout;
input [diwidth-1:0]din;
input [adawidth-1:0]ada;
input [adbwidth-1:0]adb;
input clka,cea,clkb,ceb,rstb;
reg [diwidth-1:0]mem [2**adawidth-1:0];
reg [dowidth-1:0]dout_reg;
localparam b = 2**adawidth/2**adbwidth ;
integer j ;
```

```
always @(posedge clka)begin
    if (cea)begin
        mem[ada] <= din;
    end
end

always@(posedge clkb )begin
    if(rstb)begin
        dout_reg <= 0;
    end
    else begin
        if(ceb)begin
            for(j = 0;j < b;j = j+1)
                dout_reg[((j+1)*diwidth-1)-: diwidth]<=
mem[adb*b+j];
        end
    end
end
assign dout = dout_reg;
endmodule
```

VHDL 示例：

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

entity top is

    generic( DATA_WIDTHB : integer := 12;
        ADDRESS_WIDTHB : integer := 7;
        ADDRESS_WIDTHA : integer := 8;
        DATA_WIDTHA : integer := 6
);

    port (
```

```
        din : in STD_LOGIC_VECTOR(DATA_WIDTHA - 1 downto 0);
        ada: in STD_LOGIC_VECTOR(ADDRESS_WIDTHA - 1 downto
0);
        clka, cea, clkb, ceb, rstb: in STD_LOGIC;
        adb: in STD_LOGIC_VECTOR(ADDRESS_WIDTHB - 1 downto
0);
        dout : out STD_LOGIC_VECTOR(DATA_WIDTHB - 1 downto 0));
    end top;

    architecture rtl of top is
    type mem_type is array (2**ADDRESS_WIDTHA - 1 downto 0) of
STD_LOGIC_VECTOR (DATA_WIDTHA - 1 downto 0);

        signal mem : mem_type;
    begin
        process (clka)
        begin
            if (rising_edge(clka)) then
                if(cea = '1')then
                    mem(conv_integer(ada)) <= din;
                end if;
            end if;
        end process;

        process(clkb)
        begin
            if (rising_edge(clkb)) then
                if (rstb='1')then
                    dout <=(others=>'0');
                else
                    if(ceb = '1')then
                    dout(DATA_WIDTHA*1 - 1 downto DATA_WIDTHA*0)
<= mem(conv_integer(adb&'0'));
                    dout(DATA_WIDTHA*2 - 1 downto DATA_WIDTHA*1)
<= mem(conv_integer(adb&'1'));
                    end if;
```

```
            end if;
        end if;
    end process;
end rtl;
```

4.3.6 Decoder 形式

以 bypass、同步复位模式的 SDPB 为例介绍其实现，其编码形式可如下所示：

Verilog 示例：

```
module top (data_out, data_in, wad, rad,rst, clk,wre)/*synthesis syn_ramstyle="block_ram"*/;;
parameter init0 = 16'h1234;
parameter init1 = 16'h5678;
parameter init2 = 16'h9abc;
parameter init3 = 16'h0147;

output reg[3:0] data_out;
input [3:0]data_in;
input [3:0]wad,rad;
input clk,wre,rst;

reg [15:0] mem0=init0;
reg [15:0] mem1=init1;
reg [15:0] mem2=init2;
reg [15:0] mem3=init3;
always @(posedge clk)begin
    if (wre) begin
      mem0[wad] <= data_in[0];
      mem1[wad] <= data_in[1];
      mem2[wad] <= data_in[2];
      mem3[wad] <= data_in[3];
   end
end
always @(posedge clk)begin
    if(rst)begin
```

```
            data_out<=16'h00;
        end
        else begin
            data_out[0] <= mem0[rad];
            data_out[1] <= mem1[rad];
            data_out[2] <= mem2[rad];
            data_out[3] <= mem3[rad];
        end
    end
    endmodule
```

VHDL 示例：

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

    entity top is
        port (
            data_in : in STD_LOGIC_VECTOR(3 downto 0);
            wad: in STD_LOGIC_VECTOR(3 downto 0);
            clk, wre,rst: in STD_LOGIC;
            rad: in STD_LOGIC_VECTOR(3 downto 0);
            data_out : out STD_LOGIC_VECTOR(3 downto 0)
        );
        attribute syn_ramstyle : string;
        attribute syn_ramstyle of top : entity is "block_ram";
    end top;

    architecture rtl of top is
        signal mem0 :   STD_LOGIC_VECTOR(15 downto
0) :="0001001000110100";
        signal mem1 :   STD_LOGIC_VECTOR(15 downto
0) :="0101011001111000";
        signal mem2 :   STD_LOGIC_VECTOR(15 downto
0) :="1001101010111100";
        signal mem3 :   STD_LOGIC_VECTOR(15 downto
```

```
0) :="0000000101000111";
    begin
        process (clk)
        begin
            if (rising_edge(clk)) then
                if(wre = '1')then
                    mem0(conv_integer(wad)) <= data_in(0);
                    mem1(conv_integer(wad)) <= data_in(1);
                    mem2(conv_integer(wad)) <= data_in(2);
                    mem3(conv_integer(wad)) <= data_in(3);
                end if;
            end if;
        end process;

        process(clk)
        begin
            if (rising_edge(clk)) then
                if (rst='1')then
                    data_out <=(others=>'0');
                else
                    data_out(0) <= mem0(conv_integer(rad));
                    data_out(1) <= mem1(conv_integer(rad));
                    data_out(2) <= mem2(conv_integer(rad));
                    data_out(3) <= mem3(conv_integer(rad));
                end if;
            end if;
        end process;
    end rtl;
```

4.3.7 byte-enable 功能

byte-enable 功能的推断仅 Arora V BSRAM 支持，其编码形式可如下所示:

Verilog 示例：

```
module Gowin_SDPB (dout, clka, cea, clkb, ceb, reset, ada, din, adb,
byte_ena);
```

```
output reg[31:0] dout;
input clka,cea;
input clkb,ceb;
input reset;
input [7:0] ada,adb;
input [31:0] din;
input [3:0] byte_ena;

reg[31:0]mem[2**8-1:0];

always@(posedge clka)begin
    if(cea)begin
        if(byte_ena[0])begin
            mem[ada][7:0]<=din[7:0];
        end
        if(byte_ena[1])begin
            mem[ada][15:8]<=din[15:8];
        end
        if(byte_ena[2])begin
            mem[ada][23:16]<=din[23:16];
        end
        if(byte_ena[3])begin
            mem[ada][31:24]<=din[31:24];
        end
    end
end
always@(posedge clkb)begin
    if(reset)begin
        dout<=0;
    end
    else begin
        if(ceb)begin
            dout<=mem[adb];
```

```
            end
        end
    end
    endmodule
```

VHDL 示例：

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

entity ram is
    generic(
        ADDRESS_WIDTH : integer := 8;
        DATA_WIDTH : integer := 32
    );
    port (
    din : in STD_LOGIC_VECTOR(DATA_WIDTH - 1 downto 0);
    ada: in STD_LOGIC_VECTOR(ADDRESS_WIDTH - 1 downto 0);
    clka, cea, clkb, ceb,reset: in STD_LOGIC;
    adb: in STD_LOGIC_VECTOR(ADDRESS_WIDTH - 1 downto 0);
    byte_ena : in STD_LOGIC_VECTOR(3 downto 0);
    dout : out STD_LOGIC_VECTOR(DATA_WIDTH - 1 downto 0));
end ram;

architecture rtl of ram is
    type mem_type is array (2**ADDRESS_WIDTH - 1 downto 0) of
STD_LOGIC_VECTOR (DATA_WIDTH - 1 downto 0);
    signal mem : mem_type;
begin
    process(clkb)
    begin
        if (rising_edge(clkb)) then
            if (reset='1')then
                dout <=(others=>'0');
            else
```

```
                    if(ceb = '1')then
                        dout <= mem(conv_integer(adb));
                    end if;
                end if;
            end if;
        end process;

        process (clka)
        begin
            if (rising_edge(clka)) then
                if(cea = '1')then
                    if(byte_ena(0)='1')then
                        mem(conv_integer(ada))(7 downto 0) <= din(7
downto 0);
                    end if;

                    if(byte_ena(1)='1')then
                        mem(conv_integer(ada))(15 downto 8) <=
din(15 downto 8);
                    end if;
                    if(byte_ena(2)='1')then
                        mem(conv_integer(ada))(23 downto 16) <=
din(23 downto 16);
                    end if;
                    if(byte_ena(3)='1')then
                        mem(conv_integer(ada))(31 downto 24) <=
din(31 downto 24);
                    end if;
                end if;
            end if;
        end process;
    end rtl;
```

4.4 pROM/pROMX9

pROM/pROMX9(16K/18K Block ROM)，16Kbit/18Kbit 块状只读储存器。其工作模式为只读模式，可支持 2 种读模式（bypass 模式和 pipeline 模式）。pROM/pROMX9 的赋值方式有 case 语句、readmemb/readmemh、memory 定义时赋值等方式赋值。若综合出 pROM，需至少满足下述条件之一：

1.数据位宽*地址深度>=1024，且地址深度>32

2.使用 syn_romstyle = "block_rom"

4.4.1 case 语句赋初值

以 bypass、同步复位模式的 pROM 为例介绍其实现，其编码形式可如下所示：

Verilog 示例：

```
module top(clk,rst,ce,addr,dout)/*synthesis
syn_romstyle="block_rom"*/ ;
input clk;
input rst,ce;
input [4:0] addr;
output reg [31:0] dout=32'h00000000;

always @(posedge clk )begin
    if (rst) begin
        dout <= 0;
    end
    else begin
        if(ce)begin
            case(addr)
                    5'h00: dout <= 32'h52853fd5;
                      5'h01: dout <= 32'h38581bd2;
                      5'h02: dout <= 32'h040d53e4;
                      5'h03: dout <= 32'h22ce7d00;
                      5'h04: dout <= 32'h73d90e02;
                      5'h05: dout <= 32'hc0b4bf1c;
                      5'h06: dout <= 32'hec45e626;
                      5'h07: dout <= 32'hd9d000d9;
                      5'h08: dout <= 32'haacf8574;
```

```
                5'h09: dout <= 32'hb655bf16;
                5'h0a: dout <= 32'h8c565693;
                5'h0b: dout <= 32'hb19808d0;
                5'h0c: dout <= 32'he073036e;
                5'h0d: dout <= 32'h41b923f6;
                5'h0e: dout <= 32'hdce89022;
                5'h0f: dout <= 32'hba17fce1;
                5'h10: dout <= 32'hd4dec5de;
                5'h11: dout <= 32'ha18ad699;
                5'h12: dout <= 32'h4a734008;
                5'h13: dout <= 32'h5c32ac0e;
                5'h14: dout <= 32'h8f26bdd4;
                5'h15: dout <= 32'hb8d4aab6;
                5'h16: dout <= 32'hf55e3c77;
                5'h17: dout <= 32'h41a5d418;
                5'h18: dout <= 32'hba172648;
                5'h19: dout <= 32'h5c651d69;
                5'h1a: dout <= 32'h445469c3;
                5'h1b: dout <= 32'h2e49668b;
                5'h1c: dout <= 32'hdc1aa05b;
                5'h1d: dout <= 32'hcebfe4cd;
                5'h1e: dout <= 32'h1e1f0f1e;
                5'h1f: dout <= 32'h86fd31ef;
                default: dout <= 32'h8e9008a6;
            endcase
        end
    end
end
endmodule
```

VHDL 示例：

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

ENTITY top is
```

```
    PORT (
        clk:IN STD_LOGIC;
        ce,rst:IN STD_LOGIC;
        addr : STD_LOGIC_VECTOR(4 DOWNTO 0);
        dout:OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
    );
attribute syn_romstyle : string;
attribute syn_romstyle of top : entity is "block_rom";
end top;

ARCHITECTURE rtl OF top IS
BEGIN
    PROCESS(clk)
    BEGIN
        IF(clk'EVENT AND clk='1') THEN
            IF(rst='1')THEN
                dout<=(OTHERS=>'0');
            ELSE
                IF(ce='1')THEN
                    CASE addr IS
                  WHEN b"00000"=> dout <= x"52853fd5";
                  WHEN b"00001"=> dout <= x"38581bd2";
                  WHEN b"00010"=> dout <= x"040d53e4";
                  WHEN b"00011"=> dout <= x"22ce7d00";
                  WHEN b"00100"=> dout <= x"73d90e02";
                  WHEN b"00101"=> dout <= x"c0b4bf1c";
                  WHEN b"00110"=> dout <= x"ec45e626";
                  WHEN b"00111"=> dout <= x"d9d000d9";
                  WHEN b"01000"=> dout <= x"aacf8574";
                  WHEN b"01001"=> dout <= x"b655bf16";
                  WHEN b"01010"=> dout <= x"8c565693";
                  WHEN b"01011"=> dout <= x"b19808d0";
                  WHEN b"01100"=> dout <= x"e073036e";
                  WHEN b"01101"=> dout <= x"41b923f6";
```

```
                WHEN b"01110"=> dout <= x"dce89022";
                WHEN b"01111"=> dout <= x"ba17fce1";
                WHEN b"10000"=> dout <= x"d4dec5de";
                WHEN b"10001"=> dout <= x"a18ad699";
                WHEN b"10010"=> dout <= x"4a734008";
                WHEN b"10011"=> dout <= x"5c32ac0e";
                WHEN b"10100"=> dout <= x"8f26bdd4";
                WHEN b"10101"=> dout <= x"b8d4aab6";
                WHEN b"10110"=> dout <= x"f55e3c77";
                WHEN b"10111"=> dout <= x"41a5d418";
                WHEN b"11000"=> dout <= x"ba172648";
                WHEN b"11001"=> dout <= x"5c651d69";
                WHEN b"11010"=> dout <= x"445469c3";
                WHEN b"11011"=> dout <= x"2e49668b";
                WHEN b"11100"=> dout <= x"dc1aa05b";
                WHEN b"11101"=> dout <= x"cebfe4cd";
                WHEN b"11110"=> dout <= x"1e1f0f1e";
                WHEN b"11111"=> dout <= x"86fd31ef";
                WHEN OTHERS => dout <= x"8e9008a6";
                  END CASE;
              END IF;
            END IF;
          END IF;
        END PROCESS;
    END rtl;
```

4.4.2 memory 定义时赋初值

memory 定义时赋初值需 Verilog language 选择 system Verilog。以 bypass、同步复位模式的 pROM 为例介绍其实现形式，其编码形式可如下所示：

Verilog 示例

```
module top ( clk, addr,rst, data_out)/* synthesis syn_romstyle = "block_rom" */;
input clk;
input rst;
```

```
input [3:0] addr;
output reg[3:0] data_out;
reg [3:0] mem [15:0]={4'h1,4'h2,4'h3,4'h4,4'h5,4'h6,4'h7,4'h8,4'h9,4'ha,
4'hb, 4'hc,4'hd,4'he,4'hf,4'hd};

always @(posedge clk)begin
    if (rst) begin
      data_out <= 0;
   end
    else begin
        data_out <= mem[addr];;
    end
end
endmodule
```

VHDL 示例：

```
library ieee;
use ieee.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
ENTITY top is
    generic(
        Width : integer := 32;
        Depth : integer := 5
    );
    PORT (clk:IN STD_LOGIC;
        ce,rst:IN STD_LOGIC;
        addr : STD_LOGIC_VECTOR(Depth-1 DOWNTO 0);
        dout:OUT STD_LOGIC_VECTOR(Width-1 DOWNTO 0)
    );
attribute syn_romstyle:string;
attribute syn_romstyle of top : entity is "block_rom";
end top;

ARCHITECTURE rtl OF top IS
    signal data_out :STD_LOGIC_VECTOR(Width-1 DOWNTO 0);
```

```
        TYPE vector_array IS ARRAY(2**Depth-1 DOWNTO 0) OF
STD_LOGIC_VECTOR(Width-1 DOWNTO 0);
      CONSTANT mem:vector_array:=(
        "01110010010011000010110011001010",

 "01110100111100101111010111001010","1010000100101000001101001
1101011","1101010000111100111101000000010","0001000010000001110
0000010010110",

 "00100100101000010010000011111011","0000101001010100011101111
0111110","1110000101110111011101010111110101","1111100011111111011
0000000001111",

 "01011001100101100001010110010010","1000111011010110000011100
1011011","1010110110001111010001111011010","0000000101001111001
1011100100111",

 "00111010101001100100101011011010","1111011100111011101001010
111111","0100010010111000111010100110000 1","1011011101110101101
1111000100111",

 "11010100101101011001101101101111","1100101010000001100100010
1001011","1011110111010000000110101101000 1","0011001011100110010
0110010011011",

 "11101011101010000010000101101100","1000110101111010100010110
1010010","0000000110111001011011101110001 1","0101111110011010110
1011000000110",

 "01100110001100100010001101100111","0011101001011011110101001
1101000","0100100101011111011110011101100 1","0100001000100110111
0100110010000",

 "10001111001010011111101000000001","0100111100111010110111011
1001110","1101001100111101101011111011010 0");

    BEGIN
        PROCESS(clk)
        BEGIN
            IF(clk'EVENT AND clk='1') THEN
                IF(rst='1')THEN
```

```
            data_out<=(OTHERS=>'0');
        ELSE
            IF(ce='1') THEN
                data_out<=mem(conv_integer(addr));
            END IF;
        END IF;
    END IF;
  END PROCESS;
  dout<=data_out;
END rtl;
```

4.4.3 readmemb/readmemh 方式赋初值

readmemb/ readmemh 方式赋值在进行使用时需注意路径的书写，请以‘/’作为路径分隔符。以 pipeline、异步复位模式的 pROM 为例介绍其实现，其编码形式可如下所示：

Verilog 示例：

```
module top ( clk, addr, rst ,data_out);
input clk;
input rst;
input [4:0] addr;
output reg   [31:0] data_out;
reg [31:0] mem [31:0] /* synthesis syn_romstyle = "block_rom" */;
reg [31:0] data_out_reg;
initial begin
  $readmemh ("E:/prom.ini", mem);
end
always @(posedge clk or posedge rst)begin
  if(rst)begin
    data_out_reg <=0;
  end
  else begin
    data_out_reg <= mem[addr];
  end
end
```

```
always @(posedge clk or posedge rst)begin
    if(rst)begin
        data_out <=0;
    end
    else begin
        data_out <= data_out_reg;
    end
end
endmodule
```

prom.ini 数据如下：

```
11001100
11001100
11001100
11001100
17001100
11001100
16001100
1f001100
11111100
11111100
11001110
11000111
11000111
11001110
11011100
11001110
```

5 SSRAM 编码规范

SSRAM 是分布式静态随机存储器，可配置成单端口模式，伪双端口模式和只读模式。

5.1 RAM16S 类型

RAM16S 类型包含 RAM16S1、RAM16S2、RAM16S4，其区别在于输出位宽宽度。RAM16S 类型可以采用 Decoder、Memory、移位寄存器等形式进行书写，若综合出 RAM16S，memory（除移位寄存器形式）需满足以下其中一个条件：

1.读地址和输出不经过 register；

2.输出经过 register，地址深度*数据位宽<1024；

3.使用属性约束 syn_ramstyle="distributed_ram"。

5.1.1 Decoder 形式

以 RAM16S4 为例，介绍 Decoder 形式的实现，其编码形式可如下所示：

Verilog 示例：

```
module top (data_out, data_in, addr, clk,wre);
parameter init0 = 16'h1234;
parameter init1 = 16'h5678;
parameter init2 = 16'h9abc;
parameter init3 = 16'h0147;

output [3:0]data_out;
input [3:0]data_in;
input [3:0]addr;
input clk,wre;
reg [15:0] mem0=init0;
```

```
reg [15:0] mem1=init1;
reg [15:0] mem2=init2;
reg [15:0] mem3=init3;

always @(posedge clk)begin
    if (wre) begin
      mem0[addr] <= data_in[0];
      mem1[addr] <= data_in[1];
      mem2[addr] <= data_in[2];
      mem3[addr] <= data_in[3];
  end
end

assign data_out[0] = mem0[addr];
assign data_out[1] = mem1[addr];
assign data_out[2] = mem2[addr];
assign data_out[3] = mem3[addr];
endmodule
```

VHDL 示例：

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

entity top is
    port (
        data_in : in STD_LOGIC_VECTOR(3 downto 0);
        addr: in STD_LOGIC_VECTOR(3 downto 0);
        clk, wre: in STD_LOGIC;
        data_out : out STD_LOGIC_VECTOR(3 downto 0)
    );
end top;

architecture rtl of top is
    signal mem0 :   STD_LOGIC_VECTOR(15 downto
```

```
0) :="0001001000110100";
        signal mem1 :   STD_LOGIC_VECTOR(15 downto
0) :="0101011001111000";
        signal mem2 :   STD_LOGIC_VECTOR(15 downto
0) :="1001101010111100";
        signal mem3 :   STD_LOGIC_VECTOR(15 downto
0) :="0000000101000111";
    begin
        process (clk)
        begin
            if (rising_edge(clk)) then
                if(wre = '1')then
                    mem0(conv_integer(addr)) <= data_in(0);
                    mem1(conv_integer(addr)) <= data_in(1);
                    mem2(conv_integer(addr)) <= data_in(2);
                    mem3(conv_integer(addr)) <= data_in(3);
                end if;
            end if;
        end process;

        data_out(0) <= mem0(conv_integer(addr));
        data_out(1) <= mem1(conv_integer(addr));
        data_out(2) <= mem2(conv_integer(addr));
        data_out(3) <= mem3(conv_integer(addr));

    end rtl;
```

5.1.2 Memory 形式

memory 形式可根据初值形式分为 memory 无初值形式、memory 定义时赋初值以及 readmemh/readmemb 形式。memory 定义时赋初值以及 readmemh/readmemb 形式请参考 4.1.4 readmemb/readmemh 方式赋初值，本节不再赘述。

以 RAM16S4 为例，介绍 memory 无初值形式的实现，其编码形式可如下所示：

Verilog 示例：

```
module top(data_out, data_in, addr, clk, wre);
```

```
output [3:0]data_out;
input [3:0]data_in;
input [3:0]addr;
input clk,wre;
reg [3:0] mem [15:0];
always @(posedge clk)begin
    if ( wre) begin
        mem[addr] <= data_in;
    end
end
assign data_out = mem[addr];
endmodule
```

VHDL 示例：

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

entity top is
    generic(
        DATA_WIDTH : integer := 4;
        ADDRESS_WIDTH : integer := 4
    );

    port (
        data_in : in STD_LOGIC_VECTOR(DATA_WIDTH - 1 downto
0);
        addr: in STD_LOGIC_VECTOR(ADDRESS_WIDTH - 1
downto 0);
        clk, wre: in STD_LOGIC;
        data_out : out STD_LOGIC_VECTOR(DATA_WIDTH - 1
downto 0)
    );
end top;

architecture rtl of top is
```

```
    type mem_type is array (2**ADDRESS_WIDTH - 1 downto 0) of
    STD_LOGIC_VECTOR (DATA_WIDTH - 1 downto 0);
    signal mem : mem_type;
begin
    data_out <= mem(conv_integer(addr));
    process (clk)
    begin
        if (rising_edge(clk)) then
            if( wre = '1')then
                mem(conv_integer(addr)) <= data_in;
            end if;
        end if;
    end process;
end rtl;
```

5.1.3 移位寄存器形式

需满足以下条件之一：

1. memory 深度>3 且 8<memory 深度*数据位宽<=256，且 memory 深度=2 的 n 次方；
2. 添加属性约束 syn_srlstyle= "distributed_ram"，且 memory 深度=2 的 n 次方, memory 深度>3 且 memory 深度*数据位宽>8。

以 GowinSynthesis 综合出 RAM16S4 为例，介绍移位寄存器形式的实现，其编码形式可如下所示：

Verilog 示例：

```
module    p_seqshift(clk, we, din, dout);
parameter width=18;
parameter depth=4;
input clk, we;
input [width-1:0] din;
output [width-1:0] dout;

reg [width-1:0] regBank[depth-1:0];

always @(posedge clk) begin
    if (we) begin
```

```
            regBank[depth-1:1] <= regBank[depth-2:0];
            regBank[0] <= din;
        end
    end

    assign dout = regBank[depth-1];
    endmodule
```

VHDL 示例：

```
    Library IEEE;
    use IEEE.std_logic_1164.all;
    use IEEE.std_logic_unsigned.all;

    entity p_seqshift is
        generic(
            width: integer := 18;
            depth: integer := 4
        );
        port(
            clk : in std_logic;
            we : in std_logic;
            din : in std_logic_vector(width-1 downto 0);
            dout : out std_logic_vector(width-1 downto 0)
        );

    end entity;
    architecture Behavioral of p_seqshift is
        TYPE matrix_index is array (depth-1 downto 0) of
std_logic_vector(width-1 downto 0);
        signal regBank: matrix_index;
    begin
        PROCESS(clk) begin
        IF(clk'event and clk = '1') then
            IF (we = '1') then
                for i in 1 to depth-1 loop
```

```
                    regBank(i) <= regBank(i-1);
                end loop;
                regBank(0) <= din;
            end IF;
        end IF;
        end PROCESS;
        dout <= regBank(depth-1);
    END Behavioral;
```

5.2 RAM16SDP 类型

RAM16SDP 类型包含 RAM16SDP1、RAM16SDP2、RAM16SDP4，其区别在于输出位宽宽度。RAM16SDP 类型可以采用 Decoder、Memory、移位寄存器等形式进行书写，若综合出 RAM16SDP，memory（除移位寄存器形式）需满足以下其中一个条件：

1.读地址和输出不经过 register；

2.读地址或输出经过 register，地址深度*数据位宽<1024；

3.使用属性约束 syn_ramstyle="distributed_ram"。

5.2.1 Decoder 形式

以 RAM16SDP4 为例，介绍 Decoder 形式的实现，其编码形式可如下所示：

Verilog 示例：

```
module top (data_out, data_in, wad, rad, clk,wre);
parameter init0 = 16'h1234;
parameter init1 = 16'h5678;
parameter init2 = 16'h9abc;
parameter init3 = 16'h0147;

output [3:0]data_out;
input [3:0]data_in;
input [3:0]wad,rad;
input clk,wre;
reg [15:0] mem0=init0;
reg [15:0] mem1=init1;
reg [15:0] mem2=init2;
reg [15:0] mem3=init3;
```

```
always @(posedge clk)begin
    if (wre) begin
      mem0[wad] <= data_in[0];
      mem1[wad] <= data_in[1];
      mem2[wad] <= data_in[2];
      mem3[wad] <= data_in[3];
   end
end

assign data_out[0] = mem0[rad];
assign data_out[1] = mem1[rad];
assign data_out[2] = mem2[rad];
assign data_out[3] = mem3[rad];
endmodule
```

VHDL 示例：

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

entity top is
    port (
        data_in : in STD_LOGIC_VECTOR(3 downto 0);
        wad: in STD_LOGIC_VECTOR(3 downto 0);
        clk, wre: in STD_LOGIC;
        rad: in STD_LOGIC_VECTOR(3 downto 0);
        data_out : out STD_LOGIC_VECTOR(3 downto 0)
    );
end top;

architecture rtl of top is
    signal mem0 :   STD_LOGIC_VECTOR(15 downto
0) :="0001001000110100";
    signal mem1 :   STD_LOGIC_VECTOR(15 downto
0) :="0101011001111000";
```

```
        signal mem2 :   STD_LOGIC_VECTOR(15 downto
0) :="1001101010111100";
        signal mem3 :   STD_LOGIC_VECTOR(15 downto
0) :="0000000101000111";
    begin
        process (clk)
        begin
            if (rising_edge(clk)) then
                if(wre = '1')then
                    mem0(conv_integer(wad)) <= data_in(0);
                    mem1(conv_integer(wad)) <= data_in(1);
                    mem2(conv_integer(wad)) <= data_in(2);
                    mem3(conv_integer(wad)) <= data_in(3);
                end if;
            end if;
        end process;

        data_out(0) <= mem0(conv_integer(rad));
        data_out(1) <= mem1(conv_integer(rad));
        data_out(2) <= mem2(conv_integer(rad));
        data_out(3) <= mem3(conv_integer(rad));

    end rtl;
```

5.2.2 Memory 形式

memory 形式可根据初值形式分为 memory 无初值形式、memory 定义时赋初值以及 readmemh/readmemb 形式。memory 定义时赋初值以及 readmemh/readmemb 形式请参考 4.1.4 readmemb/readmemh 方式赋初值，本节不再赘述。

以 RAM16SDP4 为例，介绍 Memory 形式的实现，其编码形式可如下所示：

Verilog 示例：

```
module top(data_out, data_in, addra, clk, wre, addrb);
output [3:0]data_out;
input [3:0]data_in;
input [3:0] addra ,addrb;
```

```
input clk,wre;

reg [3:0] mem [15:0];

always @(posedge clk)begin
    if (wre)begin
        mem[addra] <= data_in;
    end
end

assign data_out = mem[addrb];

endmodule
```

VHDL 示例：

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

entity top is
    generic(
        DATA_WIDTH : integer := 4;
        ADDRESS_WIDTH : integer := 4
    );
    port (
        data_in : in STD_LOGIC_VECTOR(DATA_WIDTH - 1 downto
0);
        ada, adb: in STD_LOGIC_VECTOR(ADDRESS_WIDTH - 1
downto 0);
        clk, wre: in STD_LOGIC;
        data_out : out STD_LOGIC_VECTOR(DATA_WIDTH - 1
downto 0));
end top;

architecture rtl of top is
    type mem_type is array (2**ADDRESS_WIDTH - 1 downto 0) of
```

```
        STD_LOGIC_VECTOR (DATA_WIDTH - 1 downto 0);
        signal mem : mem_type;
begin
        data_out <= mem(conv_integer(adb));
        process (clk)
        begin
            if (rising_edge(clk)) then
                if(wre = '1')then
                    mem(conv_integer(ada)) <= data_in;
                end if;
            end if;
        end process;

end rtl;
```

5.2.3 移位寄存器形式

移位寄存器若综合为 RAM16SDP 类型，需满足以下条件之一：

1.memory 深度>3 且 memory 深度*数据位宽<=256，且 memory 深度!=2 的 n 次方；

2.添加属性约束 syn_srlstyle= " distributed_ram"，且 memory 深度!=2 的 n 次方，memory 深度>3 且 memory 深度*数据位宽>8。

以 GowinSynthesis 综合出 RAM16SDP4 为例，介绍移位寄存器形式的实现，其编码形式可如下所示：

Verilog 示例：

```
module    p_seqshift(clk, we, din, dout);
parameter width=18;
parameter depth=7;
input clk, we;
input [width-1:0] din;
output [width-1:0] dout;

reg [width-1:0] regBank[depth-1:0];

always @(posedge clk) begin
    if (we) begin
```

```
            regBank[depth-1:1] <= regBank[depth-2:0];
            regBank[0] <= din;
        end
    end

    assign dout = regBank[depth-1];
    endmodule
```

VHDL 示例：

```
    Library IEEE;
    use IEEE.std_logic_1164.all;
    use IEEE.std_logic_unsigned.all;

    entity p_seqshift is
        generic(
            width: integer := 18;
            depth: integer := 7
        );
        port(
            clk : in std_logic;
            we : in std_logic;
            din : in std_logic_vector(width-1 downto 0);
            dout : out std_logic_vector(width-1 downto 0)
        );

    end entity;
    architecture Behavioral of p_seqshift is
        TYPE matrix_index is array (depth-1 downto 0) of
std_logic_vector(width-1 downto 0);
        signal regBank: matrix_index;
    begin
        PROCESS(clk) begin
        IF(clk'event and clk = '1') then
            IF (we = '1') then
                for i in 1 to depth-1 loop
```

```
                regBank(i) <= regBank(i-1);
            end loop;
            regBank(0) <= din;
        end IF;
    end IF;
    end PROCESS;
    dout <= regBank(depth-1);
END Behavioral;
```

5.3 ROM16

ROM16 是地址深度为 16，数据位宽为 1 的只读存储器，存储器的内容通过 INIT 进行初始化，ROM16 可以采用 Memory、Decoder 等形式进行书写，ROM16 的综合需要添加属性约束 syn_romstyle ="distributed_rom"。

5.3.1 Decoder 形式

Verilog 示例：

```
module top(addr,dataout)/*synthesis
syn_romstyle="distributed_rom"*/ ;
input [3:0] addr;
output reg    dataout=1'h0;
always @(*)begin
    case(addr)
        4'h0: dataout <= 1'h0;
        4'h1: dataout <= 1'h0;
        4'h2: dataout <= 1'h1;
        4'h3: dataout <= 1'h0;
        4'h4: dataout <= 1'h1;
        4'h5: dataout <= 1'h1;
        4'h6: dataout <= 1'h0;
        4'h7: dataout <= 1'h0;
        4'h8: dataout <= 1'h0;
        4'h9: dataout <= 1'h1;
        4'ha: dataout <= 1'h0;
        4'hb: dataout <= 1'h0;
        4'hc: dataout <= 1'h1;
        4'hd: dataout <= 1'h0;
```

```
        4'he: dataout <= 1'h0;
        4'hf: dataout <= 1'h0;
        default: dataout <=  1'h0;
    endcase
end
endmodule
```

VHDL 示例：

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

ENTITY top is
    PORT (
        addr : STD_LOGIC_VECTOR(3 DOWNTO 0);
        dout:OUT STD_LOGIC
    );
attribute syn_romstyle : string;
attribute syn_romstyle of top : entity is "distributed_rom";
end top;

ARCHITECTURE rtl OF top IS
BEGIN
    PROCESS(addr)
    BEGIN
        CASE addr IS
            WHEN b"0000"=> dout <= '0';
            WHEN b"0001"=> dout <= '0';
            WHEN b"0010"=> dout <= '1';
            WHEN b"0011"=> dout <= '0';
            WHEN b"0100"=> dout <= '1';
            WHEN b"0101"=> dout <= '1';
            WHEN b"0110"=> dout <= '0';
            WHEN b"0111"=> dout <= '0';
            WHEN b"1000"=> dout <= '0';
            WHEN b"1001"=> dout <= '1';
```

```
            WHEN b"1010"=> dout <= '0';
            WHEN b"1011"=> dout <= '0';
            WHEN b"1100"=> dout <= '1';
            WHEN b"1101"=> dout <= '0';
            WHEN b"1110"=> dout <= '0';
            WHEN b"1111"=> dout <= '0';
            WHEN OTHERS => dout <= '0';
        END CASE;
    END PROCESS;
END rtl;
```

5.3.2 Memory 形式

memory 形式可根据初值形式分为 memory 无初值形式、memory 定义时赋初值以及 readmemh/readmemb 形式。memory 定义时赋初值以及 readmemh/readmemb 形式请参考 4.1.4，本节不再赘述。

Verilog 示例：

```
module top (addr,dataout)/*synthesis
syn_romstyle="distributed_rom"*/;
input [3:0] addr;
output reg dataout=1'b0;

parameter init0 = 16'h117a;
reg [15:0] mem0=init0;
always @(*)begin
    dataout <= mem0[addr];
    end
endmodule
```

VHDL 示例：

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
ENTITY top is
    PORT (
        addr : STD_LOGIC_VECTOR(3 DOWNTO 0);
        dout:OUT STD_LOGIC
```

```
        );
    attribute syn_romstyle : string;
    attribute syn_romstyle of top : entity is "distributed_rom";
    end top;

    ARCHITECTURE rtl OF top IS
        constant mem0 :    STD_LOGIC_VECTOR(15 downto
0) :=x"117a";
    BEGIN
        PROCESS(addr)
        BEGIN
            dout<=mem0(conv_integer(addr));
        END PROCESS;
    END rtl;
```

6 DSP 编码规范[1]

注！

[1] Arora V 产品 DSP 编码规范参见第 7 章 Arora V DSP 编码规范。

DSP（Digital Signal Processing）数字信号处理包含预加器（Pre-Adder），乘法器（MULT）和 54 位算术逻辑单元（ALU54D）。

6.1 Pre-adder

Pre-adder 是预加器，实现预加、预减和移位功能。Pre-adder 按照位宽分为两种，分别是 9 位宽的 PADD9 和 18 位宽的 PADD18。Pre-adder 需与 Multiplier 相配合才可推断出来。

6.1.1 预加功能

以经过 AREG、BREG、同步复位模式的 PADD9-MULT9X9 为例介绍 PADD 预加功能的实现，其编码形式可如下所示：

Verilog 示例：

```
module top(a0, b0, b1, dout, rst, clk, ce);
input [7:0] a0;
input [7:0] b0;
input [7:0] b1;
input rst, clk, ce;
output [17:0] dout;
reg [8:0] p_add_reg=9'h000;
reg [7:0] a0_reg=8'h00;
reg [7:0] b0_reg=8'h00;
reg [7:0] b1_reg=8'h00;
reg [17:0] pipe_reg=18'h00000;
reg [17:0] s_reg=18'h00000;
```

```
always @(posedge clk)begin
    if(rst)begin
        a0_reg <= 0;
        b0_reg <= 0;
        b1_reg <= 0;
    end else begin
        if(ce)begin
            a0_reg <= a0;
            b0_reg <= b0;
            b1_reg <= b1;
        end
    end
end
always @(posedge clk)begin
    if(rst)begin
            p_add_reg <= 0;
    end else begin
        if(ce)begin
                p_add_reg <= b0_reg+b1_reg;
        end
    end
end


always @(posedge clk)
begin
    if(rst)begin
            pipe_reg <= 0;
    end else begin
        if(ce) begin
            pipe_reg <= a0_reg*p_add_reg;
        end
    end
```

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

assign dout = s_reg;

endmodule
```

VHDL 示例：

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity top is
      generic(
        b_width: integer := 8;
        a_width : integer := 8;
        dout_width : integer := 18
      );
      port (
            dout: out std_logic_vector(dout_width-1 downto 0);
            a0: in std_logic_vector(a_width-1 downto 0);
            b0,b1: in std_logic_vector(b_width-1 downto 0);
            ce: in std_logic;
            clk: in std_logic;
            rst: in std_logic
      );
```

```
end top;
architecture Behavioral of top is
    signal a0_reg: std_logic_vector(a_width-1 downto 0);
    signal b0_reg,b1_reg: std_logic_vector(b_width-1 downto 0);
    signal p_add_reg : std_logic_vector(b_width downto 0);
    signal pipe_reg,s_reg: std_logic_vector(dout_width-1 downto 0);
begin
    process(clk)begin
        if(rising_edge(clk))then
            if(rst='1')then
                a0_reg<=(others=>'0');
                b0_reg<=(others=>'0');
                b1_reg<=(others=>'0');
            else
                if(ce='1')then
                    a0_reg<=a0;
                    b0_reg<=b0;
                    b1_reg<=b1;
                end if;
            end if;
        end if;
    end process;

    process(clk)begin
        if(rising_edge(clk))then
            if(rst='1')then
                p_add_reg<=(others=>'0');
            else
                if(ce='1')then
                    p_add_reg<= ('0' & b0_reg) + ('0' & b1_reg);
                end if;
            end if;
        end if;
    end process;
```

```
    process(clk)begin
        if(rising_edge(clk))then
            if(rst='1')then
                pipe_reg<=(others=>'0');
            else
                if(ce='1')then
                    pipe_reg<=('0' & a0_reg)*p_add_reg;
                end if;
            end if;
        end if;
    end process;

    process(clk)begin
        if(rising_edge(clk))then
            if(rst='1')then
                s_reg<=(others=>'0');
            else
                if(ce='1')then
                    s_reg<=pipe_reg;
                end if;
            end if;
        end if;
    end process;
    dout<=s_reg;
end Behavioral;
```

6.1.2 预减功能

以经过 AREG、BREG、同步复位模式的 PADD9-MULT9X9 为例介绍 PADD 预减功能的实现，其编码形式可如下所示：

Verilog 示例：

```
module top(a0, b0, b1, dout, rst, clk, ce);
input [7:0] a0;
input [7:0] b0;
input [7:0] b1;
```

```
input rst, clk, ce;
output [17:0] dout;
reg [8:0] p_add_reg=9'h000;
reg [7:0] a0_reg=8'h00;
reg [7:0] b0_reg=8'h00;
reg [7:0] b1_reg=8'h00;
reg [17:0] pipe_reg=18'h00000;
reg [17:0] s_reg=18'h00000;

always @(posedge clk)begin
    if(rst)begin
        a0_reg <= 0;
        b0_reg <= 0;
        b1_reg <= 0;
    end else begin
        if(ce)begin
            a0_reg <= a0;
            b0_reg <= b0;
            b1_reg <= b1;
        end
    end
end
always @(posedge clk)begin
    if(rst)begin
            p_add_reg <= 0;
    end else begin
        if(ce)begin
                p_add_reg <= b0_reg-b1_reg;
        end
    end
end


always @(posedge clk)
```

```
begin
    if(rst)begin
            pipe_reg <= 0;
    end else begin
        if(ce) begin
            pipe_reg <= a0_reg*p_add_reg;
        end
    end
end
always @(posedge clk)
begin
    if(rst)begin
        s_reg <= 0;
    end else begin
        if(ce) begin
            s_reg <= pipe_reg;
        end
    end
end

assign dout = s_reg;

endmodule
```

VHDL 示例：

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity top is
      generic(
        b_width: integer := 8;
        a_width : integer := 8;
        dout_width : integer := 18
      );
```

```
    port (
        dout: out std_logic_vector(dout_width-1 downto 0);
        a0: in std_logic_vector(a_width-1 downto 0);
        b0,b1: in std_logic_vector(b_width-1 downto 0);
        ce: in std_logic;
        clk: in std_logic;
        rst: in std_logic
    );
end top;
architecture Behavioral of top is
    signal a0_reg: std_logic_vector(a_width-1 downto 0);
    signal b0_reg,b1_reg: std_logic_vector(b_width-1 downto 0);
    signal p_add_reg : std_logic_vector(b_width downto 0);
    signal pipe_reg,s_reg: std_logic_vector(dout_width-1 downto 0);
begin
    process(clk)begin
        if(rising_edge(clk))then
            if(rst='1')then
                a0_reg<=(others=>'0');
                b0_reg<=(others=>'0');
                b1_reg<=(others=>'0');
            else
                if(ce='1')then
                    a0_reg<=a0;
                    b0_reg<=b0;
                    b1_reg<=b1;
                end if;
            end if;
        end if;
    end process;

    process(clk)begin
        if(rising_edge(clk))then
            if(rst='1')then
```

```
                p_add_reg<=(others=>'0');
            else
                if(ce='1')then
                    p_add_reg<= ('0' & b0_reg) - ('0' & b1_reg);
                end if;
            end if;
        end if;
    end process;

    process(clk)begin
        if(rising_edge(clk))then
            if(rst='1')then
                pipe_reg<=(others=>'0');
            else
                if(ce='1')then
                    pipe_reg<=('0' & a0_reg)*p_add_reg;
                end if;
            end if;
        end if;
    end process;

    process(clk)begin
        if(rising_edge(clk))then
            if(rst='1')then
                s_reg<=(others=>'0');
            else
                if(ce='1')then
                    s_reg<=pipe_reg;
                end if;
            end if;
        end if;
    end process;
    dout<=s_reg;
end Behavioral;
```

6.1.3 移位功能

以经过 AREG、BREG、异步复位模式的 PADD18-MULT18X18 介绍 PADD 移位功能的实现，其编码形式可如下所示：

Verilog 示例：

```
module top(a0, a1, b0, b1, p0, p1, clk, ce, reset);
parameter a_width=18;
parameter b_width=18;
parameter p_width=36;
input [a_width-1:0] a0, a1;
input [b_width-1:0] b0, b1;
input clk, ce, reset;
output [p_width-1:0] p0, p1;
wire [b_width-1:0] b0_padd, b1_padd;
reg [b_width-1:0] b0_reg=18'h00000;
reg [b_width-1:0] b1_reg=18'h00000;
reg [b_width-1:0] bX1=18'h00000;
reg [b_width-1:0] bY1=18'h00000;

always @(posedge clk or posedge reset)
begin
    if(reset)begin
        b0_reg <= 0;
        b1_reg <= 0;
        bX1 <= 0;
        bY1 <= 0;
    end else begin
        if(ce)begin
            b0_reg <= b0;
            b1_reg <= b1;
            bX1 <= b0_reg;
            bY1 <= b1_reg;
        end
    end
end
```

```
assign b0_padd = bX1 + b1_reg;
assign b1_padd= b0_reg + bY1;

assign p0 = a0 * b0_padd;
assign p1 = a1 * b1_padd;

endmodule
```

VHDL 示例：

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity top is
    generic(
      b_width: integer := 18;
      a_width : integer := 18;
      p_width : integer := 36
    );
    port (
        p0,p1: out std_logic_vector(p_width-1 downto 0);
        a0,a1: in std_logic_vector(a_width-1 downto 0);
        b0,b1: in std_logic_vector(b_width-1 downto 0);
        ce: in std_logic;
        clk: in std_logic;
        reset: in std_logic
    );
end top;
architecture Behavioral of top is
    signal b0_reg,b1_reg,bX1,bY1: std_logic_vector(b_width-1
downto 0);
    signal b0_padd,b1_padd: std_logic_vector(b_width-1 downto 0);
begin
    process(clk,reset)begin
        if(reset='1')then
```

```
            b0_reg<=(others=>'0');
            b1_reg<=(others=>'0');
            bX1<=(others=>'0');
            bY1<=(others=>'0');
        else
            if(rising_edge(clk))then
                if(ce='1')then
                    b0_reg<=b0;
                    b1_reg<=b1;
                    bX1<=b0_reg;
                    bY1<=b1_reg;
                end if;
            end if;
        end if;
    end process;
    b0_padd<=bX1+b1_reg;
    b1_padd<=b0_reg+bY1;

    p0<=a0*b0_padd;
    p1<=a1*b1_padd;

end Behavioral;
```

6.2 Multiplier

Multiplier 是 DSP 的乘法器单元，乘法器的乘数输入信号定义为 MDIA 和 MDIB，乘积输出信号定义为 MOUT，可实现乘法运算：DOUT=A*B。

Multiplier 根据数据位宽可配置成 9x9，18x18，36x36 等乘法器，分别对应原语 MULT9X9，MULT18X18，MULT36X36。以经过 AREG、BREG、OUT_REG、PIPE_REG、异步复位模式的 MULT18X18 为例介绍其实现，其编码形式可如下所示：

Verilog 示例：

```
module top(a,b,c,clock,reset,ce);
input signed [17:0] a;
input signed [17:0] b;
input clock;
```

```
input reset;
input ce;
output signed [35:0] c;

reg signed [17:0] ina=18'h00000;
reg signed [17:0] inb=18'h00000;
reg signed [35:0] pp_reg=36'h000000000;
reg signed [35:0] out_reg=36'h000000000;
wire signed [35:0] mult_out;

always @(posedge clock or posedge reset)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 or posedge reset)begin
    if(reset)begin
        pp_reg<=0;
    end else begin
        if(ce)begin
            pp_reg<=mult_out;
        end
    end
end

always @(posedge clock or posedge reset)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
```

VHDL 示例：

```
    library IEEE;
    use IEEE.std_logic_1164.all;
    use IEEE.STD_LOGIC_ARITH.ALL;
    use IEEE.STD_LOGIC_UNSIGNED.ALL;
    entity top is
        generic(
          b_width: integer := 18;
          a_width : integer := 18;
          dout_width : integer := 36
    );
        port (
            c: out std_logic_vector(dout_width-1 downto 0);
            a: in std_logic_vector(a_width-1 downto 0);
            b: in std_logic_vector(b_width-1 downto 0);
            ce: in std_logic;
            clock: in std_logic;
            reset: in std_logic
        );
    end top;
    architecture Behavioral of top is
        signal ina: std_logic_vector(a_width-1 downto 0);
        signal inb: std_logic_vector(b_width-1 downto 0);
        signal pp_reg,out_reg,mult_out: std_logic_vector(dout_width-1
downto 0);
```

```
begin
    process(clock,reset)begin
        if(reset='1')then
            ina<=(others=>'0');
            inb<=(others=>'0');
        else
            if(rising_edge(clock))then
                if(ce='1')then
                    ina<=a;
                    inb<=b;
                end if;
            end if;
        end if;
    end process;
    mult_out<=ina*inb;

    process(clock,reset)begin
        if(reset='1')then
            pp_reg<=(others=>'0');
        else
            if(rising_edge(clock))then
                if(ce='1')then
                    pp_reg<=mult_out;
                end if;
            end if;
        end if;
    end process;

    process(clock,reset)begin
        if(reset='1')then
            out_reg<=(others=>'0');
        else
            if(rising_edge(clock))then
                if(ce='1')then
```

```
                    out_reg<=pp_reg;
                end if;
            end if;
        end if;
    end process;
    c<=out_reg;
end Behavioral;
```

6.3 ALU54D

ALU54D（54-bit Arithmetic Logic Unit）是 54 位算术逻辑单元，实现 54 位的算术逻辑运算。若综合出 ALU54D，数据位宽 width 需在[48,54]区间，否则需添加属性约束 syn_dspstyle="dsp"。以经过 AREG、BREG、OUT_REG、异步复位模式的 ALU54D 介绍其实现，其编码形式可如下所示：

Verilog 示例：

```
module top(a, b, s, accload, clk, ce, reset);
parameter width=54;
input signed [width-1:0] a, b;
input accload, clk, ce, reset;
output signed [width-1:0] s;
wire signed [width-1:0] s_sel;
reg [width-1:0] a_reg=54'h00000000000000;
reg [width-1:0] b_reg=54'h00000000000000;
reg [width-1:0] s_reg=54'h00000000000000;
reg acc_reg=1'b0;

always @(posedge clk or posedge reset)
begin
    if(reset)begin
        a_reg <= 0;
        b_reg <= 0;
    end else begin
        if(ce)begin
            a_reg <= a;
            b_reg <= b;
        end
```

```
    end
end
always @(posedge clk)
begin
    if(ce)begin
        acc_reg <= accload;
    end
end

assign s_sel = (acc_reg == 1) ? s : 0;
always @(posedge clk or posedge reset)
begin
    if(reset)begin
        s_reg <= 0;
    end else begin
        if(ce)begin
            s_reg <= s_sel + a_reg + b_reg;
        end
    end
end
assign s = s_reg;
endmodule
```

VHDL 示例：

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity top is
      generic(
        b_width: integer := 54;
        a_width : integer := 54;
        dout_width : integer := 54
      );
      port (
```

```
            s: buffer std_logic_vector(dout_width-1 downto 0);
            a: in std_logic_vector(a_width-1 downto 0);
            b: in std_logic_vector(b_width-1 downto 0);
            accload : in std_logic;
            ce: in std_logic;
            clk: in std_logic;
            reset: in std_logic
        );
end top;
architecture Behavioral of top is
    signal a_reg: std_logic_vector(a_width-1 downto 0);
    signal b_reg: std_logic_vector(b_width-1 downto 0);
    signal s_sel,s_reg: std_logic_vector(b_width-1 downto 0);
    signal acc_reg : std_logic;
begin
    process(clk,reset)begin
        if(reset='1')then
            a_reg<=(others=>'0');
            b_reg<=(others=>'0');
        else
            if(rising_edge(clk))then
                if(ce='1')then
                    a_reg<=a;
                    b_reg<=b;
                end if;
            end if;
        end if;
    end process;

    process(clk)begin
        if(rising_edge(clk))then
            if(ce='1')then
                acc_reg <= accload;
            end if;
```

```
        end if;
    end process;

    process(acc_reg,s)begin
        if(acc_reg='1')then
            s_sel<=s;
        else
            s_sel<=(others=>'0');
        end if;
    end process;


    process(clk,reset)begin
        if(reset='1')then
            s_reg<=(others=>'0');
        else
            if(rising_edge(clk))then
                if(ce='1')then
                    s_reg<=s_sel + a_reg +b_reg;
                end if;
            end if;
        end if;
    end process;

    s<=s_reg;
end Behavioral;
```

6.4 MULTALU

MULTALU 模式实现一个乘法器输出经过 54-bit ALU 运算，包括 MULTALU36X18 和 MULTALU18X18，其中 MULTALU18X18 仅支持实例化形式。MULTALU36X18 有三种运算功能：DOUT=A*B±C、DOUT=∑(A*B)、DOUT=A*B+CASI。

6.4.1 A*B±C 功能

以经过 AREG、BREG、CREG、PIPE_REG、OUT_REG、异步复位模式的 MULTALU36X18 介绍 DOUT=A*B+C 功能的实现，其编码形式可如

下所示：

Verilog 示例：

```
module top(a0, b0, c,s, reset, ce, clock);
parameter a_width=36;
parameter b_width=18;
parameter s_width=54;
input signed [a_width-1:0] a0;
input signed [b_width-1:0] b0;
input signed [s_width-2:0] c;
input reset, ce, clock;
output signed [s_width-1:0] s;
wire signed [s_width-1:0] p0;
reg signed [a_width-1:0] a0_reg=36'h000000000;
reg signed [b_width-1:0] b0_reg=18'h00000;
reg signed [s_width-1:0] p0_reg=54'h00000000000000;
reg signed [s_width-1:0] o0_reg=54'h00000000000000;
reg signed [s_width-2:0] c_reg=54'h00000000000000;
always @(posedge clock or posedge reset)begin
    if(reset)begin
        a0_reg <= 0;
        b0_reg <= 0;
        c_reg <= 0;
    end else begin
        if(ce)begin
            a0_reg <= a0;
            b0_reg <= b0;
              c_reg <= c;
        end
    end
end

assign p0 = a0_reg * b0_reg;
always @(posedge clock or posedge reset)begin
    if(reset)begin
```

```
        p0_reg <= 0;
        o0_reg <= 0;
    end else begin
        if(ce)begin
            p0_reg <= p0;
            o0_reg <= p0_reg+c_reg;
        end
    end
end

assign s = o0_reg;
endmodule
```

VHDL 示例：

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity top is
    generic(
      b_width: integer := 36;
      a_width : integer := 18;
      s_width : integer := 54
    );
    port (
          s: out std_logic_vector(s_width-1 downto 0);
          a0: in std_logic_vector(a_width-1 downto 0);
          b0: in std_logic_vector(b_width-1 downto 0);
          c : in std_logic_vector(s_width-2 downto 0);
          ce: in std_logic;
          clock: in std_logic;
          reset: in std_logic
    );
end top;
architecture Behavioral of top is
```

```
    signal a0_reg: std_logic_vector(a_width-1 downto 0);
    signal b0_reg: std_logic_vector(b_width-1 downto 0);
    signal p0,p0_reg,o0_reg: std_logic_vector(s_width-1 downto 0);
    signal c_reg: std_logic_vector(s_width-2 downto 0);
begin
    process(clock,reset)begin
        if(reset='1')then
            a0_reg<=(others=>'0');
            b0_reg<=(others=>'0');
            c_reg <=(others=>'0');
        else
            if(rising_edge(clock))then
                if(ce='1')then
                    a0_reg<=a0;
                    b0_reg<=b0;
                    c_reg <=c;
                end if;
            end if;
        end if;
    end process;

    p0<=a0_reg * b0_reg;

    process(clock,reset)begin
        if(reset='1')then
            p0_reg<=(others=>'0');
            o0_reg<=(others=>'0');
        else
            if(rising_edge(clock))then
                if(ce='1')then
                    p0_reg<=p0;
                    o0_reg<=p0_reg+c_reg;
                end if;
            end if;
```

```
        end if;
    end process;

    s<=o0_reg;
end Behavioral;
```

6.4.2 ∑(A*B)功能

以经过 PIPE_REG、OUT_REG、异步复位模式的 MULTALU36X18 介绍 DOUT=∑(A*B)功能的实现，其编码形式可如下所示：

Verilog 示例：

```
module top(a,b,c,clock,reset,ce);
parameter a_width = 36;
parameter b_width = 18;
parameter c_width = 54;
input signed [a_width-1:0] a;
input signed [b_width-1:0] b;
input clock;
input reset,ce;
output signed [c_width-1:0] c;

reg signed [c_width-1:0] pp_reg=54'h00000000000000;
reg signed [c_width-1:0] out_reg=54'h00000000000000;
wire signed [c_width-1:0] mult_out,c_sel;
reg acc_reg0=1'b0;
reg acc_reg1=1'b0;

assign mult_out=a*b;
always @(posedge clock or posedge reset) begin
    if(reset)begin
        pp_reg<=0;
    end else begin
        if(ce)begin
            pp_reg<=mult_out;
        end
    end
```

```
end

always @(posedge clock or posedge reset)
begin
    if(reset) begin
        out_reg <= 0;
    end else if(ce) begin
        out_reg <= c + pp_reg;
    end
end

assign c=out_reg;
endmodule
```

VHDL 示例：

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity top is
      generic(
        b_width: integer := 36;
        a_width : integer := 18;
        c_width : integer := 54
      );
      port (
            c: buffer std_logic_vector(c_width-1 downto 0);
            a: in std_logic_vector(a_width-1 downto 0);
            b: in std_logic_vector(b_width-1 downto 0);
            ce: in std_logic;
            clock: in std_logic;
            reset: in std_logic
      );
end top;
architecture Behavioral of top is
```

```
        signal pp_reg,out_reg,mult_out: std_logic_vector(c_width-1
downto 0);
    begin

        mult_out<=a*b;
        process(clock,reset)begin
            if(reset='1')then
                pp_reg<=(others=>'0');
            else
                if(rising_edge(clock))then
                    if(ce='1')then
                        pp_reg<=mult_out;
                    end if;
                end if;
            end if;
        end process;

        process(clock,reset)begin
            if(reset='1')then
                out_reg<=(others=>'0');
            else
                if(rising_edge(clock))then
                    if(ce='1')then
                        out_reg<=c+pp_reg;
                    end if;
                end if;
            end if;
        end process;

        c<=out_reg;
    end Behavioral;
```

6.4.3 A*B+CASI 功能

以经过 PIPE_REG、OUT_REG、异步复位模式的 MULTALU36X18 介绍 DOUT=A*B+CASI 功能的实现，其编码形式可如下所示：

Verilog 示例：

```
module top(a0, a1, a2, b0, b1, b2, s, reset, ce, clock);
parameter a_width=36;
parameter b_width=18;
parameter s_width=54;
input signed [a_width-1:0] a0, a1, a2;
input signed [b_width-1:0] b0, b1, b2;
input reset, ce, clock;
output signed [s_width-1:0] s;
wire signed [s_width-1:0] p0, p1, p2, s0, s1;
reg signed [s_width-1:0] p0_reg=54'h00000000000000;
reg signed [s_width-1:0] p1_reg=54'h00000000000000;
reg signed [s_width-1:0] p2_reg=54'h00000000000000;
reg signed [s_width-1:0] s0_reg=54'h00000000000000;
reg signed [s_width-1:0] s1_reg=54'h00000000000000;
reg signed [s_width-1:0] o0_reg=54'h00000000000000;

assign p0 = a0 * b0;
assign p1 = a1 * b1;
assign p2 = a2 * b2;
always @(posedge clock or posedge reset)
begin
    if(reset)begin
        p0_reg <= 0;
        p1_reg <= 0;
        p2_reg <= 0;
        o0_reg <= 0;
    end else begin
        if(ce)begin
            p0_reg <= p0;
            p1_reg <= p1;
```

```
            p2_reg <= p2;
            o0_reg <= p0_reg;
        end
    end
end

assign s0 = o0_reg + p1_reg;
always @(posedge clock or posedge reset)
begin
    if(reset)begin
        s0_reg <= 0;
    end else begin
        if(ce)begin
            s0_reg <= s0;
        end
    end
end

assign s1 = s0_reg + p2_reg;
always @(posedge clock or posedge reset)
begin
    if(reset)begin
        s1_reg <= 0;
    end else begin
        if(ce)begin
            s1_reg <= s1;
        end
    end
end
assign s=s1_reg;
endmodule
```

VHDL 示例：

```
library IEEE;
use IEEE.std_logic_1164.all;
```

```
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity top is
    generic(
      b_width: integer := 36;
      a_width : integer := 18;
      s_width : integer := 54
    );
    port (
        s: out std_logic_vector(s_width-1 downto 0);
        a0,a1,a2: in std_logic_vector(a_width-1 downto 0);
        b0,b1,b2: in std_logic_vector(b_width-1 downto 0);
        ce: in std_logic;
        clock: in std_logic;
        reset: in std_logic
    );
end top;
architecture Behavioral of top is
    signal p0,p1,p2,s0,s1: std_logic_vector(s_width-1 downto 0);
    signal p0_reg,p1_reg,p2_reg: std_logic_vector(s_width-1 downto
0);
    signal s0_reg,s1_reg,o0_reg: std_logic_vector(s_width-1 downto
0);
begin

    p0<=a0*b0;
    p1<=a1*b1;
    p2<=a2*b2;

    process(clock,reset)begin
        if(reset='1')then
            p0_reg<=(others=>'0');
            p1_reg<=(others=>'0');
            p2_reg<=(others=>'0');
            o0_reg<=(others=>'0');
```

```
        else
            if(rising_edge(clock))then
                if(ce='1')then
                    p0_reg<=p0;
                    p1_reg<=p1;
                    p2_reg<=p2;
                    o0_reg<=p0_reg;
                end if;
            end if;
        end if;
    end process;

    s0 <= o0_reg+p1_reg;

    process(clock,reset)begin
        if(reset='1')then
            s0_reg<=(others=>'0');
        else
            if(rising_edge(clock))then
                if(ce='1')then
                    s0_reg<=s0;
                end if;
            end if;
        end if;
    end process;

    s1<=s0_reg+p2_reg;

    process(clock,reset)begin
        if(reset='1')then
            s1_reg<=(others=>'0');
        else
            if(rising_edge(clock))then
                if(ce='1')then
```

```
                    s1_reg<=s1;
                end if;
            end if;
        end if;
    end process;

    s<=s1_reg;
end Behavioral;
```

6.5 MULTADDALU

MULTADDALU（The Sum of Two Multipliers with ALU）是带 ALU 功能的乘加器，实现乘法求和后累加或 reload 运算，对应的原语为 MULTADDALU18X18。有三种运算功能：DOUT=A0*B0±A1*B1±C、DOUT=∑(A0*B0±A1*B1)、DOUT=A0*B0±A1*B1+CASI。

6.5.1 A0*B0±A1*B1±C 功能

以经过 A0REG、A1REG、B0REG、B1REG、PIPE0_REG、PIPE1_REG、OUT_REG、异步复位模式的 MULTADDALU18X18 介绍 DOUT= A0*B0 ± A1*B1±C 功能的实现，其编码形式可如下所示：

Verilog 示例：

```
module top(a0, a1, b0, b1, c,s, reset, clock, ce);
parameter a0_width=18;
parameter a1_width=18;
parameter b0_width=18;
parameter b1_width=18;
parameter s_width=54;
input signed [a0_width-1:0] a0;
input signed [a1_width-1:0] a1;
input signed [b0_width-1:0] b0;
input signed [b1_width-1:0] b1;
input [53:0] c;
input reset, clock, ce;
output signed [s_width-1:0] s;
wire signed [s_width-1:0] p0, p1, p;
reg signed [a0_width-1:0] a0_reg=18'h00000;
reg signed [a1_width-1:0] a1_reg=18'h00000;
```

```
reg signed [b0_width-1:0] b0_reg=18'h00000;
reg signed [b1_width-1:0] b1_reg=18'h00000;
reg signed [s_width-1:0] p0_reg=54'h00000000000000;
reg signed [s_width-1:0] p1_reg=54'h00000000000000;
reg signed [s_width-1:0] s_reg=54'h00000000000000;

always @(posedge clock)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)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+c;

always @(posedge clock)begin
    if(reset)begin
        s_reg <= 0;
    end
      else begin
        if(ce) begin
            s_reg <= p;
        end
    end
end

assign s = s_reg;
endmodule
```

VHDL 示例：

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity top is
      generic(
        a0_width : integer := 18;
        a1_width : integer := 18;
        b0_width : integer := 18;
        b1_width : integer := 18;
        s_width : integer := 54
      );
      port (
```

```
        s: out std_logic_vector(s_width-1 downto 0);
        a0 : in std_logic_vector(a0_width-1 downto 0);
        a1 : in std_logic_vector(a0_width-1 downto 0);
        b0 : in std_logic_vector(b0_width-1 downto 0);
        b1 : in std_logic_vector(b0_width-1 downto 0);
        c  : in std_logic_vector(s_width -2 downto 0);
        ce: in std_logic;
        clock: in std_logic;
        reset: in std_logic
    );
end top;
architecture Behavioral of top is
    signal a0_reg: std_logic_vector(a0_width-1 downto 0);
    signal a1_reg: std_logic_vector(a1_width-1 downto 0);
    signal b0_reg: std_logic_vector(b0_width-1 downto 0);
    signal b1_reg: std_logic_vector(b1_width-1 downto 0);
    signal p0: std_logic_vector(a0_width+b0_width-1 downto 0);
    signal p1: std_logic_vector(a1_width+b1_width-1 downto 0);
    signal p0_reg: std_logic_vector(s_width-1 downto 0);
    signal p1_reg: std_logic_vector(s_width-1 downto 0);
    signal p: std_logic_vector(s_width-1 downto 0);
    signal s_reg: std_logic_vector(s_width-1 downto 0);
begin

    process(clock)begin
        if(rising_edge(clock))then
            if(reset='1')then
                a0_reg<=(others=>'0');
                a1_reg<=(others=>'0');
                b0_reg<=(others=>'0');
                b1_reg<=(others=>'0');
            else
                if(ce='1')then
                    a0_reg<=a0;
```

```
                    a1_reg<=a1;
                    b0_reg<=b0;
                    b1_reg<=b1;
                end if;
            end if;
        end if;
    end process;

    p0<=a0_reg*b0_reg;
    p1<=a1_reg*b1_reg;

    process(clock)begin
        if(rising_edge(clock))then
            if(reset='1')then
                p0_reg<=(others=>'0');
                p1_reg<=(others=>'0');
            else
                if(ce='1')then
                    p0_reg<=x"0000"&"00"&p0;
                    p1_reg<=x"0000"&"00"&p1;
                end if;
            end if;
        end if;
    end process;

    p<=p0_reg+p1_reg+('0'&c);

    process(clock)begin
        if(rising_edge(clock))then
            if(reset='1')then
                s_reg<=(others=>'0');
            else
                if(ce='1')then
                    s_reg<=p;
```

```
                    end if;
                end if;
            end if;
        end process;
        s<=s_reg;
    end Behavioral;
```

6.5.2 ∑(A0*B0±A1*B1)功能

以经过 PIPE0_REG、PIPE1_REG、OUT_REG、异步复位模式的 MULTADDALU18X18 介绍 DOUT= ∑(A0*B0±A1*B1)功能的实现，其编码形式可如下所示：

Verilog 示例：

```
module acc(a0, a1, b0, b1, s, accload, ce, reset, clk);
parameter a_width=18;
parameter b_width=18;
parameter s_width=54;
input unsigned [a_width-1:0] a0, a1;
input unsigned [b_width-1:0] b0, b1;
input accload, ce, reset, clk;
output unsigned [s_width-1:0] s;
wire unsigned [s_width-1:0] s_sel;
wire unsigned [s_width-1:0] p0, p1;
reg unsigned [s_width-1:0] p0_reg=54'h00000000000000;
reg unsigned [s_width-1:0] p1_reg=54'h00000000000000;
reg unsigned [s_width-1:0] s=54'h00000000000000;
reg acc_reg0=1'b0;
reg acc_reg1=1'b0;

assign p0 = a0*b0;
assign p1 = a1*b1;
always @(posedge clk or posedge reset)begin
    if(reset) begin
        p0_reg <= 0;
        p1_reg <= 0;
    end else if(ce) begin
```

```
            p0_reg <= p0;
            p1_reg <= p1;
        end
    end
    always @(posedge clk)begin
          if(ce) begin
            acc_reg0 <= accload;
            acc_reg1 <= acc_reg0;
        end
    end
    assign s_sel = (acc_reg1 == 1) ? s : 0;
    always @(posedge clk or posedge reset)begin
        if(reset) begin
            s <= 0;
        end else if(ce) begin
            s <= s_sel + p0_reg - p1_reg;
        end
    end
    endmodule
```

VHDL 示例：

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity top is
      generic(
        a0_width : integer := 18;
        a1_width : integer := 18;
        b0_width : integer := 18;
        b1_width : integer := 18;
        s_width : integer := 54
      );
      port (
            s: buffer std_logic_vector(s_width-1 downto 0);
```

```
        a0 : in std_logic_vector(a0_width-1 downto 0);
        a1 : in std_logic_vector(a0_width-1 downto 0);
        b0 : in std_logic_vector(b0_width-1 downto 0);
        b1 : in std_logic_vector(b0_width-1 downto 0);
        accload: in std_logic;
        ce: in std_logic;
        clock: in std_logic;
        reset: in std_logic
    );
end top;
architecture Behavioral of top is
    signal p0: std_logic_vector(a0_width+b0_width-1 downto 0);
    signal p1: std_logic_vector(a1_width+b1_width-1 downto 0);
    signal p0_reg: std_logic_vector(s_width-1 downto 0);
    signal p1_reg: std_logic_vector(s_width-1 downto 0);
    signal p: std_logic_vector(s_width-1 downto 0);
    signal s_sel: std_logic_vector(s_width-1 downto 0);
    signal acc_reg0,acc_reg1:std_logic;
begin
    p0<=a0*b0;
    p1<=a1*b1;
    process(clock,reset)begin
        if(reset='1')then
            p0_reg<=(others=>'0');
            p1_reg<=(others=>'0');
        else
            if(rising_edge(clock))then
                if(ce='1')then
                    p0_reg<=x"0000"&"00"&p0;
                    p1_reg<=x"0000"&"00"&p1;
                end if;
            end if;
        end if;
    end process;
```

```
    process(clock)begin
        if(rising_edge(clock))then
            if(ce='1')then
                acc_reg0<=accload;
                acc_reg1<=acc_reg0;
            end if;
        end if;
    end process;

    process(acc_reg1,s)begin
        if(acc_reg1='1')then
            s_sel<=s;
        else
            s_sel<=(others=>'0');
        end if;
    end process;

    process(clock,reset)begin
        if(reset='1')then
            s<=(others=>'0');
        else
            if(rising_edge(clock))then
                if(ce='1')then
                    s<=s_sel+p0_reg-p1_reg;
                end if;
            end if;
        end if;
    end process;
end Behavioral;
```

6.5.3 A0*B0±A1*B1+CASI 功能

以经过 PIPE0_REG、PIPE1_REG、OUT_REG、异步复位模式的 MULTADDALU18X18 介绍 DOUT= A0*B0±A1*B1+CASI 功能的实现，其编码形式可如下所示：

Verilog 示例：

```
module top(a0, a1, a2, b0, b1, b2, a3, b3, s, clock, ce, reset);
parameter a_width=18;
parameter b_width=18;
parameter s_width=54;
input signed [a_width-1:0] a0, a1, a2, b0, b1, b2, a3, b3;
input clock, ce, reset;
output signed [s_width-1:0] s;
wire signed [s_width-1:0] p0, p1, p2, p3, s0, s1;
reg signed [s_width-1:0] p0_reg=54'h00000000000000;
reg signed [s_width-1:0] p1_reg=54'h00000000000000;
reg signed [s_width-1:0] p2_reg=54'h00000000000000;
reg signed [s_width-1:0] p3_reg=54'h00000000000000;
reg signed [s_width-1:0] s0_reg=54'h00000000000000;
reg signed [s_width-1:0] s1_reg=54'h00000000000000;
reg signed [a_width-1:0] a0_reg=18'h00000;
reg signed [a_width-1:0] a1_reg=18'h00000;
reg signed [a_width-1:0] a2_reg=18'h00000;
reg signed [a_width-1:0] a3_reg=18'h00000;
reg signed [b_width-1:0] b0_reg=18'h00000;
reg signed [b_width-1:0] b1_reg=18'h00000;
reg signed [b_width-1:0] b2_reg=18'h00000;
reg signed [b_width-1:0] b3_reg=18'h00000;
always @(posedge clock or posedge reset)begin
    if(reset)begin
        a0_reg <= 0;
        a1_reg <= 0;
        a2_reg <= 0;
        a3_reg <= 0;
        b0_reg <= 0;
        b1_reg <= 0;
        b2_reg <= 0;
        b3_reg <= 0;
    end else begin
        if(ce)begin
```

```
                a0_reg <= a0;
                a1_reg <= a1;
                a2_reg <= a2;
                a3_reg <= a3;
                b0_reg <= b0;
                b1_reg <= b1;
                b2_reg <= b2;
                    b3_reg <= b3;
            end
        end
    end

    assign p0 = a0_reg*b0_reg;
    assign p1 = a1_reg*b1_reg;
    assign p2 = a2_reg*b2_reg;
    assign p3 = a3_reg*b3_reg;

    always @(posedge clock or posedge reset)begin
        if(reset)begin
            p0_reg <= 0;
            p1_reg <= 0;
            p2_reg <= 0;
               p3_reg <= 0;
        end else begin
            if(ce)begin
                p0_reg <= p0;
                p1_reg <= p1;
                p2_reg <= p2;
                p3_reg <= p3;
            end
        end
    end

    assign s0 = p0_reg + p1_reg;
```

```
always @(posedge clock or posedge reset)begin
    if(reset)begin
        s0_reg <= 0;
    end else begin
        if(ce)begin
            s0_reg <= s0;
        end
    end
end
assign s1 = s0_reg + p2_reg - p3_reg;
always @(posedge clock or posedge reset)begin
    if(reset)begin
        s1_reg <= 0;
    end else begin
        if(ce)begin
            s1_reg <= s1;
        end
    end
end
assign s=s1_reg;
endmodule
```

VHDL 示例：

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity top is
      generic(
        a_width : integer := 18;
        b_width : integer := 18;
        s_width : integer := 54
      );
      port (
            s: buffer std_logic_vector(s_width-1 downto 0);
```

```
            a0,a1,a2,a3 : in std_logic_vector(a_width-1 downto 0);
            b0,b1,b2,b3 : in std_logic_vector(b_width-1 downto 0);
            ce: in std_logic;
            clock: in std_logic;
            reset: in std_logic
        );
    end top;
    architecture Behavioral of top is
        signal a0_reg,a1_reg,a2_reg,a3_reg: std_logic_vector(a_width-1
downto 0);
        signal b0_reg,b1_reg,b2_reg,b3_reg: std_logic_vector(b_width-1
downto 0);
        signal p0,p1,p2,p3: std_logic_vector(a_width+b_width-1 downto
0);
        signal s0,s1: std_logic_vector(s_width-1 downto 0);
        signal p0_reg,p1_reg,p2_reg,p3_reg: std_logic_vector(s_width-1
downto 0);
        signal s0_reg,s1_reg: std_logic_vector(s_width-1 downto 0);
    begin
        process(clock,reset)begin
            if(reset='1')then
                a0_reg<=(others=>'0');
                a1_reg<=(others=>'0');
                a2_reg<=(others=>'0');
                a3_reg<=(others=>'0');
                b0_reg<=(others=>'0');
                b1_reg<=(others=>'0');
                b2_reg<=(others=>'0');
                b3_reg<=(others=>'0');
            else
                if(rising_edge(clock))then
                    if(ce='1')then
                        a0_reg<=a0;
                        a1_reg<=a1;
                        a2_reg<=a2;
```

```
                    a3_reg<=a3;
                    b0_reg<=b0;
                    b1_reg<=b1;
                    b2_reg<=b2;
                    b3_reg<=b3;
                end if;
            end if;
        end if;
    end process;

    p0<=a0_reg*b0_reg;
    p1<=a1_reg*b1_reg;
    p2<=a2_reg*b2_reg;
    p3<=a3_reg*b3_reg;

    process(clock,reset)begin
        if(reset='1')then
            p0_reg<=(others=>'0');
            p1_reg<=(others=>'0');
            p2_reg<=(others=>'0');
            p3_reg<=(others=>'0');
        else
            if(rising_edge(clock))then
                if(ce='1')then
                    p0_reg<=x"0000" & "00" & p0;
                    p1_reg<=x"0000" & "00" & p1;
                    p2_reg<=x"0000" & "00" & p2;
                    p3_reg<=x"0000" & "00" & p3;
                end if;
            end if;
        end if;
    end process;

    s0<=p0_reg+p1_reg;
```

```
    process(clock,reset)begin
        if(reset='1')then
            s0_reg<=(others=>'0');
        else
            if(rising_edge(clock))then
                if(ce='1')then
                    s0_reg<=s0;
                end if;
            end if;
        end if;
    end process;

    s1<=s0_reg+p2_reg-p3_reg;

    process(clock,reset)begin
        if(reset='1')then
            s1_reg<=(others=>'0');
        else
            if(rising_edge(clock))then
                if(ce='1')then
                    s1_reg<=s1;
                end if;
            end if;
        end if;
    end process;
    s<=s1_reg;
end Behavioral;
```

7 Arora V DSP 编码规范

Arora V DSP（Digital Signal Processing）数字信号处理，可以实现乘法、预加、累加、移位等功能，每个 DSP 有 2 个独立的 clock 时钟信号、2 个独立的 ce 使能信号、2 个独立的 reset 复位信号。

7.1 预加功能

Arora V DSP 可以实现 26 以下位宽的有符号预加操作，支持静态加减操作、动态加减操作以及同步异步复位模式等。根据位宽的不同可以推断为 MULTALU27X18 和 MULT27X36。预加操作需与乘法操作相配合才可推断出来。

7.1.1 静态预加减功能

以 MULTALU27X18 为例介绍静态预加减功能的实现，其编码形式可如下所示：

Verilog 示例:

```
module top(a,b,d,out);
input signed[15:0]a;
input signed[15:0]b;
input signed[15:0]d;
output signed[31:0]out;
assign out=(a+d)*b;
endmodule
```

VHDL 示例：

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;


entity top is
```

```
    generic(
      a_width : integer := 16;
      b_width : integer := 16;
      d_width : integer := 16;
      dout_width : integer := 32
    );
    port (
        dout: out signed (dout_width downto 0);
        a : in signed (a_width-1 downto 0);
        b : in signed (b_width-1 downto 0);
        d : in signed (d_width-1 downto 0)
    );
end top;
architecture Behavioral of top is
    signal ad_add:signed (a_width    downto 0);
begin
    ad_add<=resize(a,17)+resize(d,17);
    dout<=ad_add*b;
end Behavioral;
```

7.1.2 动态预加减功能

以 MULTALU27X18 为例介绍动态预加减功能的实现，其编码形式可如下所示：

Verilog 示例：

```
module top(clk,ce,reset,a,b,d,psel,out);
input signed[15:0]a;
input signed[15:0]b;
input signed[15:0]d;
input psel;
input [1:0]clk,ce,reset;
output reg signed[31:0]out;
reg signed[31:0]preg;
always@(posedge clk[0])begin
    if(reset[0])begin
        preg<=0;
```

```
        end
        else begin
            if(ce[0])begin
                preg<=psel ? (a+d)*b : (a-d)*b;
            end
        end
    end

    always@(posedge clk[1])begin
        if(reset[1])begin
            out<=0;
        end
        else begin
            if(ce[1])begin
                out<=preg;
            end
        end
    end
    endmodule
```

VHDL 示例：

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity top is
    generic(
      a_width : integer := 16;
      b_width : integer := 16;
      d_width : integer := 16;
      dout_width : integer := 33
    );
    port (
        dout: out signed (dout_width-1 downto 0);
        a : in signed (a_width-1 downto 0);
```

```
            b : in signed (b_width-1 downto 0);
            d : in signed (d_width-1 downto 0);
            psel : in std_logic;
            clk,ce,reset:in signed (1 downto 0)
        );
    end top;
    architecture Behavioral of top is
        signal preg:signed (dout_width-1 downto 0);
    begin
        process(clk)begin
            if(rising_edge(clk(0)))then
                if(reset(0)='1')then
                    preg<=(others=>'0');
                else
                    if(ce(0)='1')then
                        if(psel='1')then
                            preg<=( (a(a_width-1)&a) + (d(d_width-1)
&d) )*b;
                        else
                            preg<=( (a(a_width-1)&a) - (d(d_width-1)
&d) )*b;
                        end if;
                    end if;
                end if;
            end process;

            process(clk)begin
                if(rising_edge(clk(1)))then
                    if(reset(1)='1')then
                        dout<=(others=>'0');
                    else
                        if(ce(1)='1')then
                            dout<=preg;
                        end if;
```

```
            end if;
        end if;
    end process;
end Behavioral;
```

7.2 乘法功能

Arora V DSP 可以实现 27X36 以下位宽的有符号乘法操作，根据位宽不同可以推断为MULTALU27X18、MULT12X12、MULT27X36。以经过AREG、BREG、PIPE_REG、OUT_REG、异步复位模式的 MULTALU27X18 为例介绍其实现，其编码形式可如下所示：

Verilog 示例：

```
module top(a,b,clk,ce,rst,out);
input signed[26:0]a;
input signed[17:0]b;
input   [1:0]clk,ce,rst;
output signed[44:0]out;
reg signed[26:0]a_reg;
reg signed[17:0]b_reg;
reg signed[44:0]pp_reg,out_reg;
always@(posedge clk[0] or posedge rst[0])begin
    if (rst[0])begin
        a_reg<=0;
        b_reg<=0;
    end
    else begin
        if(ce[0])begin
            a_reg<=a;
            b_reg<=b;
        end
    end
end
always@(posedge clk[1] or posedge rst[1])begin
    if (rst[1])begin
        pp_reg<=0;
    end
```

```
        else begin
            if(ce[1])begin
                pp_reg<=a_reg*b_reg;
            end
        end
    end
    always@(posedge clk[1] or posedge rst[1])begin
        if (rst[1])begin
            out_reg<=0;
        end
        else begin
            if(ce[1])begin
                out_reg<=pp_reg;
            end
        end
    end

    assign out=out_reg;
    endmodule
```

VHDL 示例：

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity top is
    generic(
      a_width : integer := 27;
      b_width : integer := 18;
      dout_width : integer := 45
    );
    port (
        dout: out signed (dout_width-1 downto 0);
        a : in signed (a_width-1 downto 0);
        b : in signed (b_width-1 downto 0);
```

```
        clk,ce,rst:in signed (1 downto 0)
    );
end top;
architecture Behavioral of top is
    signal a_reg :   signed (a_width-1 downto 0);
    signal b_reg :   signed (b_width-1 downto 0);
    signal pp_reg,out_reg:signed (dout_width-1 downto 0);
begin
    process(clk(0),rst(0))begin
        if(rst(0)='1')then
            a_reg<=(others=>'0');
            b_reg<=(others=>'0');
        else
            if(rising_edge(clk(0)))then
                if(ce(0)='1')then
                    a_reg<=a;
                    b_reg<=b;
                end if;
            end if;
        end if;
    end process;

    process(clk(1),rst(1))begin
        if(rst(1)='1')then
            pp_reg<=(others=>'0');
        else
            if(rising_edge(clk(1)))then
                if(ce(1)='1')then
                    pp_reg<=a_reg*b_reg;
                end if;
            end if;
        end if;
    end process;
```

```
    process(clk(1),rst(1))begin
        if(rst(1)='1')then
            out_reg<=(others=>'0');
        else
            if(rising_edge(clk(1)))then
                if(ce(1)='1')then
                    out_reg<=pp_reg;
                end if;
            end if;
        end if;
    end process;
    dout<=out_reg;
end Behavioral;
```

7.3 乘加功能

Arora V DSP 可以实现 27X18 以下位宽的有符号乘加操作，支持静态的加减操作、动态的加减操作以及同步异步复位模式等。乘加功能根据位宽的不同可以推断为 MULTALU27X18 和 MULTADDALU12X12。

7.3.1 静态加减功能

以静态乘加功能的 MULTADDALU12X12 为例介绍静态加减功能的实现，其编码形式可如下所示：

Verilog 示例：

```
module top(a,b,d0,d1,out);
input signed[11:0]a;
input signed[11:0]b;
input signed[11:0]d0;
input signed[11:0]d1;
output signed[24:0]out;
assign out=a*b + d0*d1;
endmodule
```

VHDL 示例：

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
```

```
entity top is
    generic(
      a_width : integer := 12;
      b_width : integer := 12;
      dout_width : integer := 25
    );
    port (
        dout: out signed (dout_width-1 downto 0);
        a,d0 : in signed (a_width-1 downto 0);
        b,d1 : in signed (b_width-1 downto 0)
    );
end top;
architecture Behavioral of top is
    signal mult1,mult2 : signed (dout_width-2 downto 0);
begin
    mult1<=a*b;
    mult2<=d0*d1;
    dout<=(mult1(dout_width-2) & mult1) +(mult2(dout_width-2) &
mult2);
end Behavioral;
```

7.3.2 动态加减功能

以经过 AREG、BREG、CREG、PREG、OREG、同步复位模式的动态乘加功能的 MULTALU27X18 为例介绍动态加减功能的实现，其编码形式可如下所示：

Verilog 示例：

```
module top(a,b,c,clk,ce,rst,sel,addsub,out);
input signed[11:0]a;
input signed[11:0]b;
input signed[47:0]c;
input   [1:0]clk,ce,rst;
input sel;
input[1:0] addsub;
output signed[47:0]out;
reg signed[11:0]a_reg;
```

```
reg signed[11:0]b_reg;
reg signed[47:0]c_reg;

reg signed[47:0]pp_reg,out_reg;
wire signed[47:0]c_sig;
assign c_sig=sel ? c_reg : 0;
always@(posedge clk[0])begin
    if (rst[0])begin
        a_reg<=0;
        b_reg<=0;
        c_reg<=0;
    end
    else begin
        if(ce[0])begin
            a_reg<=a;
            b_reg<=b;
            c_reg<=c;
        end
    end
end
always@(posedge clk[1])begin
    if (rst[1])begin
        pp_reg<=0;
    end
    else begin
        if(ce[1])begin
            pp_reg<=a_reg*b_reg;
        end
    end
end
always@(posedge clk[1])begin
    if (rst[1])begin
        out_reg<=0;
    end
```

```
            else begin
                if(ce[1])begin
                    out_reg<= addsub==2'b00 ? pp_reg + c_sig :
                              addsub==2'b01 ? -pp_reg + c_sig:
                              addsub==2'b10 ? pp_reg - c_sig : -pp_reg-
c_sig;
                end
            end
        end

        assign out=out_reg;
        endmodule
```

VHDL 示例：

```
        library IEEE;
        use IEEE.STD_LOGIC_1164.ALL;
        use IEEE.NUMERIC_STD.ALL;

        entity top is
            generic(
              a_width : integer := 12;
              b_width : integer := 12;
              c_width : integer := 48;
              dout_width : integer := 48
            );
            port (
                dout: out signed (dout_width-1 downto 0);
                a : in signed (a_width-1 downto 0);
                b : in signed (b_width-1 downto 0);
                c : in signed (c_width-1 downto 0);
                sel : in std_logic;
                addsub : in std_logic_vector(1 downto 0);
                clk,ce,rst:in signed (1 downto 0)
            );
        end top;
```

```
architecture Behavioral of top is
    signal a_reg :   signed (a_width-1 downto 0);
    signal b_reg :   signed (b_width-1 downto 0);
    signal c_reg :   signed (c_width-1 downto 0);
    signal pp_reg:signed (a_width+b_width-1 downto 0);
    signal c_sig:signed (c_width-1 downto 0);
    signal out_reg :signed (dout_width-1 downto 0);
begin
    process(sel,c_reg)begin
        if(sel='1')then
            c_sig<=c_reg;
        else
            c_sig<=(others=>'0');
        end if;
    end process;

    process(clk(0))begin
        if(rising_edge(clk(0)))then
            if(rst(0)='1')then
                a_reg<=(others=>'0');
                b_reg<=(others=>'0');
                c_reg<=(others=>'0');
            else
                if(ce(0)='1')then
                    a_reg<=a;
                    b_reg<=b;
                    c_reg<=c;
                end if;
            end if;
        end if;
    end process;
    process(clk(1))begin
        if(rising_edge(clk(1)))then
            if(rst(1)='1')then
```

```
                pp_reg<=(others=>'0');
            else
                if(ce(1)='1')then
                    pp_reg<=a_reg*b_reg;
                end if;
            end if;
        end if;
    end process;

    process(clk(1))begin
        if(rising_edge(clk(1)))then
            if(rst(1)='1')then
                out_reg<=(others=>'0');
            else
                if(ce(1)='1')then
                    if(addsub="00")then
                        out_reg<=pp_reg + c_sig;
                    elsif(addsub="01")then
                        out_reg<= -pp_reg+c_sig ;
                    elsif(addsub="10")then
                        out_reg<= pp_reg - c_sig;
                    else
                        out_reg<= -pp_reg - c_sig;
                    end if;
                end if;
            end if;
        end if;
    end process;

    dout<=out_reg;
end Behavioral;
```

7.4 累加功能

Arora V DSP 可以实现 27X18 以下位宽的有符号累加操作，支持静态累加操作、动态累加操作以及同步异步复位模式等。累加功能根据位宽的不同可以推断为 MULTALU27X18 和 MULTADDALU12X12。

7.4.1 静态累加功能

以经过 OREG、同步复位模式的 MULTALU27X18 为例介绍静态累加功能的实现，其编码形式可如下所示：

Verilog 示例：

```
module top(a,b,clk,ce,rst,out);
parameter widtha=27;
parameter widthb=18;

input signed[26:0]a;
input signed[17:0]b;
input clk,ce,rst;
output signed[44:0]out;
reg signed[44:0]out_reg;
wire signed[44:0]s_sel;
wire signed[44:0]mout;

assign s_sel= out_reg;
assign mout=a*b;
always@(posedge clk)begin
    if (rst)begin
        out_reg<=0;
    end
    else begin
        if(ce)begin
            out_reg<=s_sel+mout;
        end
    end
end
assign out=out_reg;
endmodule
```

VHDL 示例：

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity top is
    generic(
      a_width : integer := 27;
      b_width : integer := 18;
      dout_width : integer := 45
    );
    port (
        dout: out signed (dout_width-1 downto 0);
        a : in signed (a_width-1 downto 0);
        b : in signed (b_width-1 downto 0);
        clk,ce,rst:in std_logic
    );
end top;
architecture Behavioral of top is
    signal mout :signed (dout_width -1 downto 0);
    signal s_sel :signed (dout_width -1 downto 0);
    signal out_reg :signed (dout_width-1 downto 0);
begin
    mout<=a*b;
    s_sel<=out_reg;
    process(clk)begin
        if(rising_edge(clk))then
            if(rst='1')then
                out_reg<=(others=>'0');
            else
                if(ce='1')then
                    out_reg<=s_sel+mout;
                end if;
            end if;
```

```
        end if;
    end process;

    dout<=out_reg;
end Behavioral;
```

7.4.2 动态累加功能

以经过 AREG、BREG、OREG、同步复位模式的 MULTALU27X18 为例介绍动态累加功能的实现，其编码形式可如下所示：

Verilog 示例：

```
module top(a,b,clk,ce,rst,accsel,out);
parameter PRE_LOAD = 48'h000000000012;

input signed[26:0]a;
input signed[17:0]b;
input   [1:0]clk,ce,rst;
input accsel;
output signed[44:0]out;
reg signed[44:0]out_reg;
wire signed[44:0]s_sel;
wire signed[44:0]mout;
reg signed[26:0]a_reg;
reg signed[17:0]b_reg;
always@(posedge clk[0])begin
    if (rst[0])begin
        a_reg<=0;
        b_reg<=0;
    end
    else begin
        if(ce[0])begin
            a_reg<=a;
            b_reg<=b;
        end
    end
end
```

```
assign s_sel= accsel == 1'b1 ? out_reg : PRE_LOAD;
assign mout=a_reg*b_reg;

always@(posedge clk[1])begin
    if (rst[1])begin
        out_reg<=0;
    end
    else begin
        if(ce[1])begin
            out_reg<=s_sel+mout;
        end
    end
end

assign out=out_reg;
endmodule
```

VHDL 示例：

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity top is
    generic(
      a_width : integer := 27;
      b_width : integer := 18;
      dout_width : integer := 45
    );
    port (
        dout: out signed (dout_width-1 downto 0);
        a : in signed (a_width-1 downto 0);
        b : in signed (b_width-1 downto 0);
        accsel : in std_logic;
        clk,ce,rst:in signed (1 downto 0)
```

```
    );
end top;
architecture Behavioral of top is
    signal a_reg :   signed (a_width-1 downto 0);
    signal b_reg :   signed (b_width-1 downto 0);
    signal mout,s_sel :signed (dout_width-1 downto 0);
    signal out_reg :signed (dout_width-1 downto 0);
begin
    process(clk(0))begin
        if(rising_edge(clk(0)))then
            if(rst(0)='1')then
                a_reg<=(others=>'0');
                b_reg<=(others=>'0');
            else
                if(ce(0)='1')then
                    a_reg<=a;
                    b_reg<=b;
                end if;
            end if;
        end if;
    end process;

    process(accsel,out_reg)begin
        if(accsel='1')then
            s_sel<=out_reg;
        else
            s_sel<='0'&x"00000000012";
        end if;
    end process;
    mout<=a_reg*b_reg;
    process(clk(1))begin
        if(rising_edge(clk(1)))then
            if(rst(1)='1')then
                out_reg<=(others=>'0');
```

```
            else
                if(ce(1)='1')then
                    out_reg<=s_sel + mout;
                end if;
            end if;
        end if;
    end process;

    dout<=out_reg;
end Behavioral;
```

7.5 级联功能

Arora V DSP 可以实现 27X18 以下位宽的有符号级联操作，支持级联操作的原语有 MULTALU27X18 和 MULTADDALU12X12。

7.5.1 静态级联功能

以 2 个 MULTADDALU12X12 级联为例介绍静态级联功能的实现，其编码形式可如下所示：

Verilog 示例：

```
module top(a,b,d0,d1,a1,b1,d2,d3,out);
input signed[11:0]a,a1;
input signed[11:0]b,b1;
input signed[11:0]d0,d2;
input signed[11:0]d1,d3;
output signed[24:0]out;
assign out=a*b + d0*d1+a1*b1 + d2*d3;
endmodule
```

VHDL 示例:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity top is
    generic(
      a_width : integer := 12;
```

```
    b_width : integer := 12;
    dout_width : integer := 25
  );
  port (
      dout: out signed (dout_width-1 downto 0);
      a,a1,d0,d1 : in signed (a_width-1 downto 0);
      b,b1,d2,d3 : in signed (b_width-1 downto 0)
  );
end top;
architecture Behavioral of top is
    signal mult1,mult2,mult3,mult4:signed(dout_width-2 downto 0);
begin
    mult1<=a*b;
    mult2<=d0*d1;
    mult3<=a1*b1;
    mult4<=d2*d3;
    dout<=(mult1(dout_width-2) & mult1) + (mult2(dout_width-2) &
mult2) + (mult3(dout_width-2) & mult3) + (mult4(dout_width-2) & mult4);
end Behavioral;
```

7.5.2 动态级联功能

以 2 个 MULTALU27X18 级联为例介绍动态级联功能的实现，其编码形式可如下所示：

Verilog 示例：

```
module top(clk,ce,rst,sel,a,b,a1,b1,out);
parameter widtha=27;
parameter widthb=18;
parameter widthout=widtha+widthb;

input signed[widtha-1:0]a,a1;
input signed[widthb-1:0]b,b1;
input [1:0]clk,ce,rst;
input sel;
output reg signed[widthout:0]out;
reg signed[widtha-1:0]a_reg,a1_reg;
```

```
reg signed[widthb-1:0]b_reg,b1_reg;
reg signed[widthout-1:0]p0_reg,p1_reg;

always@(posedge clk[0])begin
    if(rst[0])begin
        a_reg <= 0;
        a1_reg <= 0;
        b_reg <= 0;
        b1_reg <= 0;
    end
    else begin
        if(ce[0])begin
            a_reg <= a;
            a1_reg <= a1;
            b_reg <= b;
            b1_reg <= b1;
        end
    end
end

always@(posedge clk[1])begin
    if(rst[1])begin
        p0_reg <= 0;
        p1_reg <= 0;
    end
    else begin
        if(ce[1])begin
            p0_reg <= a_reg*b_reg;
            p1_reg <= a1_reg*b1_reg;
        end
    end
end

always@(posedge clk[1])begin
```

```
    if(rst[1])begin
        out <= 0;
    end
    else begin
        if(ce[1])begin
            out <= sel ? p0_reg + p1_reg : p1_reg;
        end
    end
end
endmodule
```

VHDL 示例：

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity top is
    generic(
      a_width : integer := 27;
      b_width : integer := 18;
      dout_width : integer := 46
    );
    port (
        dout: out signed (dout_width-1 downto 0);
        a,a1 : in signed (a_width-1 downto 0);
        b,b1 : in signed (b_width-1 downto 0);
        sel : in std_logic;
        clk,ce,rst:in std_logic_vector(1 downto 0)
    );
end top;
architecture Behavioral of top is
    signal a_reg,a1_reg : signed (a_width-1 downto 0);
    signal b_reg,b1_reg : signed (b_width-1 downto 0);
    signal p0_reg,p1_reg :signed (dout_width -2 downto 0);
begin
```

```
process(clk(0))begin
    if(rising_edge(clk(0)))then
        if(rst(0)='1')then
            a_reg<=(others=>'0');
            a1_reg<=(others=>'0');
            b_reg<=(others=>'0');
            b1_reg<=(others=>'0');
        else
            if(ce(0)='1')then
                a_reg<=a;
                a1_reg<=a1;
                b_reg<=b;
                b1_reg<=b1;
            end if;
        end if;
    end if;
end process;

process(clk(1))begin
    if(rising_edge(clk(1)))then
        if(rst(1)='1')then
            p0_reg<=(others=>'0');
            p1_reg<=(others=>'0');
        else
            if(ce(1)='1')then
                p0_reg<=a_reg*b_reg;
                p1_reg<=a1_reg*b1_reg;
            end if;
        end if;
    end if;
end process;

process(clk(1))begin
    if(rising_edge(clk(1)))then
```

```
                if(rst(1)='1')then
                    dout<=(others=>'0');
                else
                    if(ce(1)='1')then
                        if(sel='1')then
                            dout<=(p0_reg(dout_width -2) & p0_reg) +
(p1_reg(dout_width -2) & p1_reg);
                        else
                            dout<=p1_reg(dout_width -2) & p1_reg;
                        end if;
                    end if;
                end if;
            end if;
        end process;

    end Behavioral;
```

7.6 移位功能

Arora V DSP 可以实现 27X18 以下位宽的有符号移位操作，支持移位操作的原语有 MULTALU27X18。

以输入 a 经过两级 register 为例介绍移位功能的实现，其编码形式可如下所示：

Verilog 示例：

```
module top(a, d0, d1, b0, b1, p0, p1, clk, ce, rst);
input signed[25:0] a,d0,d1;
input signed[17:0] b0,b1;
input clk, ce, rst;
output signed [44:0] p0, p1;
wire signed [26:0] paddsub0, paddsub1;
reg signed [25:0] a_reg0, a_reg1,a_reg2,d0_reg,d1_reg;

always @(posedge clk)
begin
    if(rst)begin
        a_reg0 <= 0;
```

```
            a_reg1 <= 0;
               a_reg2 <= 0;
            d0_reg <= 0;
            d1_reg <= 0;
        end else begin
            if(ce)begin
                    a_reg0 <= a;
                    a_reg1 <= a_reg0;
                    a_reg2 <= a_reg1;
                    d0_reg <= d0;
                    d1_reg <= d1;
            end
        end
end
assign paddsub0 = a_reg0 + d0_reg;
assign paddsub1 = a_reg2 + d1_reg;

assign p0 = paddsub0 * b0;
assign p1 = paddsub1 * b1;
endmodule
```

VHDL 示例：

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity top is
      generic(
        a_width : integer := 26;
        b_width : integer := 18;
        dout_width : integer := 45
      );
      port (
           p0,p1: out signed (dout_width-1 downto 0);
```

```
        a,d0,d1 : in signed (a_width-1 downto 0);
        b0,b1 : in signed (b_width-1 downto 0);
        clk,ce,rst:in std_logic
    );
end top;
architecture Behavioral of top is
    signal a_reg0,a_reg1 ,a_reg2,d0_reg,d1_reg: signed (a_width-1
downto 0);
    signal paddsub0,paddsub1 :signed (a_width downto 0);
begin
    process(clk)begin
        if(rising_edge(clk))then
            if(rst='1')then
                a_reg0<=(others=>'0');
                a_reg1<=(others=>'0');
                a_reg2<=(others=>'0');
                d0_reg<=(others=>'0');
                d1_reg<=(others=>'0');
            else
                if(ce='1')then
                    a_reg0<=a;
                    a_reg1<=a_reg0;
                    a_reg2<=a_reg1;
                    d0_reg<=d0;
                    d1_reg<=d1;
                end if;
            end if;
        end if;
    end process;

    paddsub0<=(a_reg0(a_width-1) & a_reg0) + (d0_reg (a_width-1)
& d0_reg);
    paddsub1<=(a_reg2(a_width-1) & a_reg2) + (d1_reg (a_width-1)
& d1_reg);
```

```
        p0<=paddsub0 * b0;
        p1<=paddsub1 * b1;

    end Behavioral;
```


GOWIN高云
智慧逻辑 定制未来