درس: آزمایشگاه معماری کامپیوتر(جلسه هشتم)

- مروری بر VHDL(ادامه)
- آزمایش هفتم

نیمسال اول ۱۴۰۱

نکات دستور case-when

- می‌توان در choiceها بازه قرار دهیم (type غیر Vector).
- می‌توان در choiceها از اپراتورها نیز استفاده نمود.
- Choiceها نباید با هم overlap داشته باشند.
- از دستور case-when معمولا در طراحی ماشین حالت‌ها و مدارهای دارای انتخاب بدون اولویت استفاده می‌شود.
- زیرا ترتیب در نوشتن انتخاب‌ها مهم است، با تایید شدن یک case، از آن case statement خارج می‌شود.

```
CASE temp IS
    WHEN 0 =>
        …
    WHEN 1 to 3  =>
        …
    WHEN “01” to “11” =>--ILLEGAL
    WHEN 4|5|6  =>
        …
    WHEN 6|7|8 =>  --ILLEGAL
    WHEN OTHERS  =>
        NULL;
END CASE;
```

Arithmetic Logic Unit

واحد ALUخود از دو واحد داخلی تشکیل شده است:

واحد محاسبه که بخش های محاسباتی ریاضی را انجام می‌دهد و با نام AUشناخته می شود که مخفف کلمه Arithmetic Unitاست.

واحد منطق که عملیات های منطقی را روی داده ها انجام می دهد و با نام LUکه مخفف کلمه Logic Unitاست شناخته می‌شود.

Arithmetic Logic Unit(cont.)

ALUپردازنده بیشتر به عنوان بخشی از پردازنده طراحی و ایجاد شده است و تنها واحدی است که در پردازنده وظیفه محاسبه جمع و تفریق ها را بر عهده دارد.
همچنین در خصوص عملیات ضرب ALUپردازنده می تواند به راحتی عملیات ضرب را بین دو عدد صحیح از نوع Integerانجام دهد.

عملیات تقسیم نیز در واحد ALUپردازنده تحت شرایط خاصی انجام می شود. عملیات تقسیم معمولاً در پردازنده ها توسط بخش های محاسباتی شناور یا همان Floating Point Unitانجام می شود
این واحد کار انجام عملیات های محاسباتی را در پردازنده ها انجام می دهند. منظور از عملیات های محاسباتی منطقی عملیات های جمع و تفریق و ضرب است.
همچنین عملیات های منطقی مانند عملیات های AND – OR – XORو Notنیز در این واحد انجام می شود.

در واحد ALUبر اساس پهنای اختصاص یافته در ساختار پردازنده امکان انجام عملیات ها با توان متفاوت وجود دارد. به عنوان مثال اگر واحد محاسبه و منطق به صورت گذرگاه ۴ بیتی باشد می تواند حداکثر از ۱۶ عمل مختلف پشتیبانی کند.

Arithmetic Logic Unit(cont.)

خروجی های واحد محاسبه و منطق را شکل های مختلف تشکیل می دهند:

مهمترین خروجی های واحد ALUعبارت‌اند از:

رقم نقلی

این نوع رقم ها معمولاً در عملیات های جمع رخ می دهد.

رقم قرضی:

این نوع رقم ها معمولاً در عمل تفریق به وجود می آیند.

بیت سرریز:

بیت سرریز یا همان Overflowکه معمولاً در اثر عمل شیفت باینری رخ می دهد.

بر اساس استاندارد ها عملیات های زیادی در واحد محاسبه و منطق انجام می شود. این عملیات ها شامل اعمال حسابی مانند عمل جمع، جمع با رقم نقلی، تفریق، تفریق با رقم قرضی، محاسبه مکمل دوم یک عملوند، افزایش و کاهش یک واحدی یکی از عملوندها و همچنین عمل منطقی بین بیت ها مانند اعمال محاسبه ای ANDو ORو XORمی باشد.

همچنین در واحد ALUاعمال شیفت بیتی مانند اعمالی که در آن یک عملوند به سمت چپ یا راست شیفت داده می‌شود و شامل شیفت منطقی، شیفت حسابی و چرخش نیز انجام می شود.

کد یک مثال برای آشنایی با عملکرد در سطح مدلسازی

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use ieee.NUMERIC_STD.all;

entity ALU is
  generic(constant shift_no : natural := 1; --number of shited or rotated bits
          constant width : natural := 16);
  Port(A, B : in  STD_LOGIC_VECTOR(width-1 downto 0);
       ALU_SEL : in  STD_LOGIC_VECTOR(3 downto 0);  -- 16 tasks
       ALU_OUT : out  STD_LOGIC_VECTOR(width-1 downto 0);
       Carryout : out std_logic);  -- Carryout flag
end ALU;

architecture Behavioral of ALU is
   signal Result : std_logic_vector (width-1 downto 0);
begin

process(A, B, ALU_SEL) --PAY ATTENTION TO THE INPUTS
   variable temp: std_logic_vector (width downto 0);
begin
   temp := (others => '0');
   CASE(ALU_SEL) IS
      WHEN "0000" => -- Addition
         temp := ('0' & A) + ('0' & B);
         Result <= temp(width-1 downto 0);
      WHEN "0001" => -- Subtraction
         temp := ('0' & A) - ('0' & B);
         Result <= temp(width-1 downto 0);
```

طول داده های ما

دقت شود که این مدار کاملا combinational است

در این مثال که Alu ۱۶ کار را انجام می دهد

یک نتیجه temporary

یک بیت بیشتر

جلوگیری از ایجاد latch

```
CASE(ALU_SEL) IS
   WHEN "0000" => -- Addition
      temp := ('0' & A) + ('0' & B);
      Result <= temp(width-1 downto 0);
   WHEN "0001" => -- Subtraction
      temp := ('0' & A) - ('0' & B);
      Result <= temp(width-1 downto 0);
   WHEN "0010" => -- Multiplication
      Result <= std_logic_vector(to_unsigned((to_integer(unsigned(A)) * to_integer(unsigned(E
   WHEN "0011" => -- Division
      Result <= std_logic_vector(to_unsigned(to_integer(unsigned(A)) / to_integer(unsigned(B)
   WHEN "0100" => -- Logical shift left
      Result <= std_logic_vector(unsigned(A) sll shift_no);
   WHEN "0101" => -- Logical shift right
      Result <= std_logic_vector(unsigned(A) srl shift_no);
   WHEN "0110" => -- Rotate left
      Result <= std_logic_vector(unsigned(A) rol shift_no);
   WHEN "0111" => -- Rotate right
      Result <= std_logic_vector(unsigned(A) ror shift_no);
   WHEN "1000" => -- Logical and
      Result <= A and B;
   WHEN "1001" => -- Logical or
      Result <= A or B;
   WHEN "1010" => -- Logical xor
      Result <= A xor B;
   WHEN "1011" => -- Logical nor
      Result <= A nor B;
   WHEN "1100" => -- Logical nand
      Result <= A nand B;
```

عملیات حسابی
منطقی
شیفت
مقایسه
کارهای دلخواه به انتخاب

دقت شود که به علت استفاده از /, * لزوما این ماژول قابل سنتز نیست و بیشتر جنبه ی مدلسازی دارد

```
            Result <= A nand B;
        WHEN "1101" => -- Logical xnor
            Result <= A xnor B;
        WHEN "1110" => -- Greater comparison
            if(A>B) then
                Result <= (others => '1') ;
            else
                Result <= (others => '0') ;
            end if;
        WHEN "1111" => -- Equal comparison
            if(A=B) then
                Result <= (others => '1') ;
            else
                Result <= (others => '0') ;
            end if;
        WHEN OTHERS => Result <= (others => 'X');
    END CASE;
    Carryout <= temp(width); -- Carryout flag
end process;

    ALU_OUT <= Result; -- ALU out

end Behavioral;
```

تعریف حالت default

نوشتن testbench

```
process
   begin
      -- hold reset state for 20 ns
      A <= x"000A";
      B <= x"0002";
      ALU_Sel <= x"0";
      wait for 20 ns;

      for i in 1 to 15 loop
         ALU_Sel <= ALU_Sel + x"1";
      wait for 20 ns;
      end loop;

      ALU_Sel <= x"0";  --Addition
      A <= x"FFFF";
      B <= x"0100";
      wait for 20 ns;
      ALU_Sel <= x"1";  --Subtraction
      A <= x"000A";
      B <= x"00F6";
      wait for 20 ns;
   wait;
end process;
END;
```

بررسی عملکرد

مشکل ایجاد شدن Latch در طراحی و روش از بین بردن آن

- اساسا Latch زمانی ایجاد می‌شود که در یک Combinational Process و در یک Conditional assignment، یک خروجی تحت تمامی حالت‌ها یا شرط‌های امکان‌پذیر، مقدار نگیرد. یعنی برای مثال در یک شرط یا حالت، مقداردهی به آن خروجی فراموش شود و از قلم بیفتد. معمولا ایجاد Latch به صورت سهوی توسط طراح اتفاق می‌افتد. در این حالت ابزار سنتز کننده، عبارت شرطی ناتمامی ایجاد می‌کند.
- ایجاد Latch هنگام سنتز مدار مطلوب نیست، زیرا باعث افزایش مساحت، delay و تغییر timing مدار می‌شود.
- این Latch در مدارهای ترکیبی (بدون Clock) ایجاد می‌شود. در مدارهای ترتیبی این مشکل وجود ندارد.

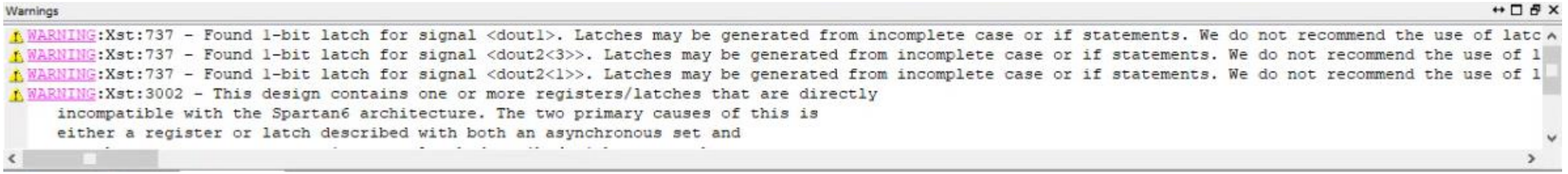
چگونگی رفع مشکل ایجاد latch

```
process (din, en)
   begin
      if en = '1' then
         dout1 <= din;
      end if;

-- Solution 1 :
      if en = '1' then
         dout1 <= din;
      else
         dout1 <= ...;
      end if;
end process;
```

```
process (din, en)
   begin
      dout1 <= '0';
      if en = '1' then
         dout1 <= din;
      end if;
```

درون process مشکلی از جهت چندبار مقداردهی به یک سیگنال وجود ندارد
پس با مقداردهی اولیه می توانیم مشکل را رفع کنیم.



چگونگی رفع مشکل ایجاد latch (ادامه)

```
 -- Incomplete Assignment:
  dout2 <=    "0101" when sel = "00" else
              "0111" when sel = "01" else
              "1111" when sel = "10"
   -- Solution :
--    dout2 <= "0101" when sel = "00" else
--              "0111" when sel = "01" else
--              "1111" when sel = "10"
--             else (others => '0');
end Behavioral;
```

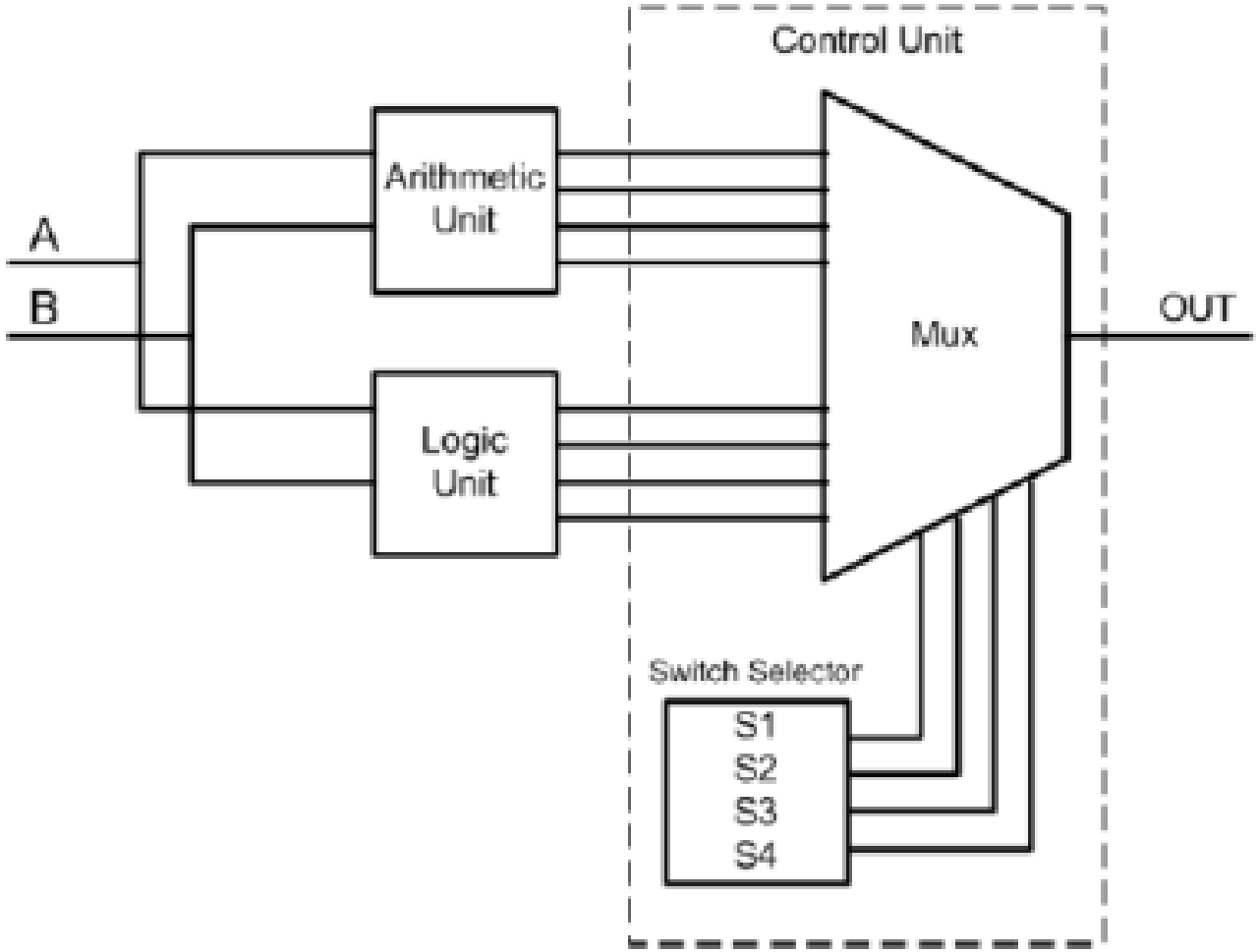
مشکل ایجاد شدن Latch در طراحی و روش از بین بردن آن - ادامه

- راه حل ۱: در دستورهای شرطی از جمله if-then-else، case-when و ... دقت شود که تمامی حالات ممکن در نظر گرفته شود و به ازای آن‌ها تمام خروجی‌هایی که حداقل در یک شرط یا حالت مقدار گرفته اند، مجددا مقداردهی شوند.
- راه حل ۲: به تمامی سیگنال‌هایی که در حالت‌ها و شرط‌های مختلف مقداردهی می‌شوند (سیگنال‌های دست چپی)، در ابتدای Process مقداردهی اولیه انجام داده شود؛ به این دلیل که ممکن است در شرط یا حالتی مقداردهی مجدد فراموش شود. مثال

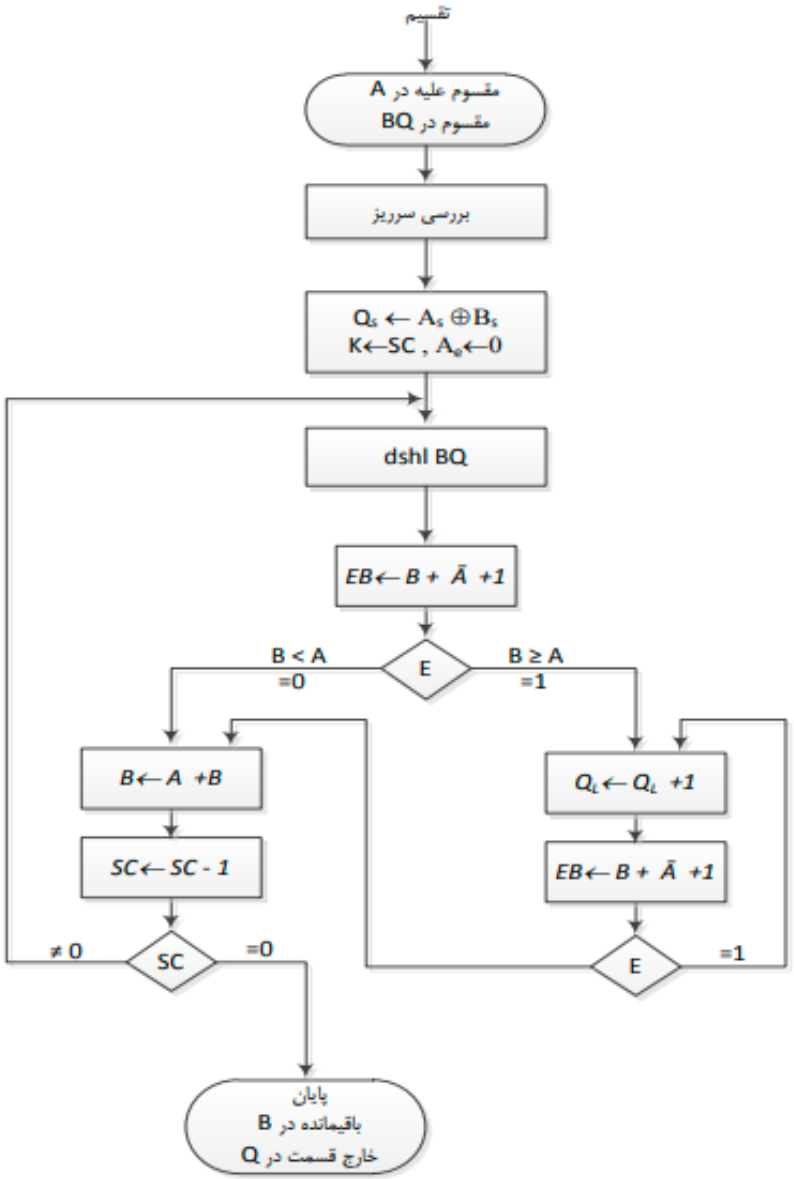
گزارش کار

پیاده سازی واحد محاسبه و منطق به زبان vhdl مشابه شکل زیر

*پیاده سازی یک تقسیم کننده ۸ بیت بر ۴ بیت



فلوچارت تقسیم کننده



فلوچارت الگوریتم تقسیم