Computer Architecture Lab Session 5

# RA

# CLA

# CSA

# 2-Digits BCD Seven Segment

# Bardia Ardakanian 9831072

# Ali Asad 9831004

## (Ripple-Adder) جمع کننده آبشاری

در این نوع جمع کننده برای بدست آوردن جمع دوعدد n -بیتی، در‌هر مرحله یک بیت از هر کدام را به Full Adder می‌دهیم و دو خروجی می‌گیریم.

خروجی اولی حاصل جمع (sum) و دومی Carry out است که در مرحله بعد به عنوان Carry in به Full Adder بعدی داده می‌شود.

در هر مرحله هر Full Adder نیاز به Carry out مرحله قبلی دارد.

در ادامه شماتیک RA را مشاهده می‌کنید:



## جمع کننده با پیش‌بینی بیت نقلی (Carry Look-ahead Adder)

اگر دو متغیر P , G را همانند زیر تعریف کنیم:

Generate Gi=AiBi

Propagate Pi=Ai+Bi

آنگاه میتوان Carry هر مرحله را به صورت زیر نوشت:

C0=G0+Cin P0

C1=G1+ G0 P1+P0 P1 Cin

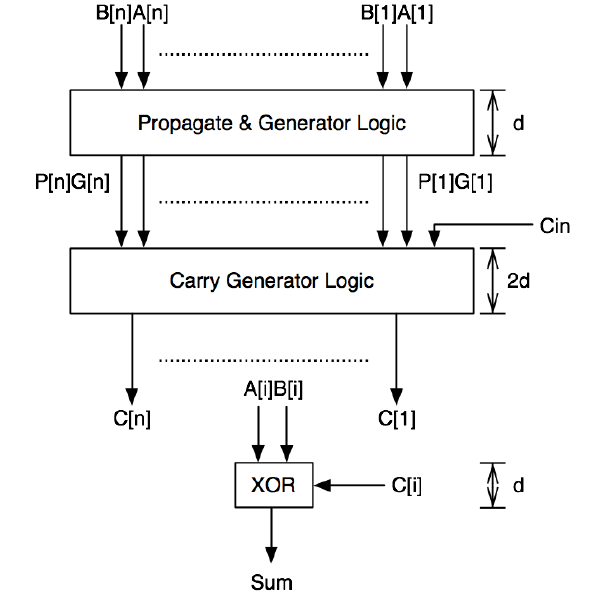
C2=G2+ G1 P2+G0P1P2 +P0 P1P2 Cin

Ci= Gi+Ci-1 Pi

Cn-1= Gn-1+Gn-2 Pn-1+Gn-3Pn-1Pn-2 +…+P0 P1…Pn-2Pn-1 Cin

حال بدون نیاز به اینکه هر FA منتظر خروجی FA قبلی باشد، می‌تواند به طور موازی حاصل جمع را بدست آورد و دیگر منتظر Carry مرحله قبلی نباشد زیرا رابطه بیان شده برای هر Carry، به طور مستقل نوشته شده است.

در ادامه شماتیک CLA را مشاهده می‌کنید:

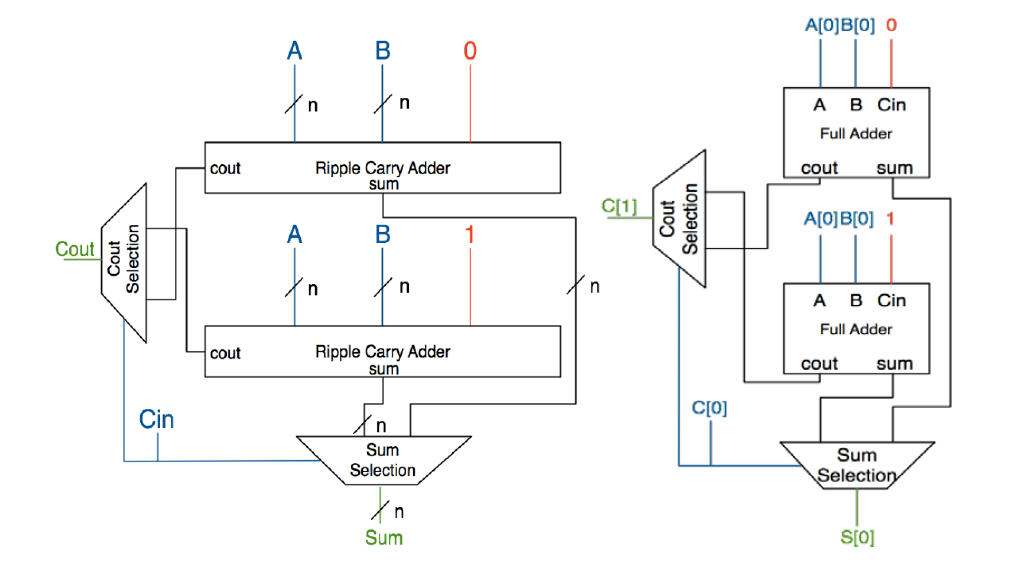


در ابتدا با صرف زمان d ،P,G ها را بدست می‌آوریم در مرحله بعد با صرف زمان 2d چون مدار SOP است،Carry ها آماده است و نهایتا حاصل جمع نهایی را بدست می آوریم.

## Carry Select Adder) جمع کننده انتخابی (

در این مدار به جای آنکه هر FA منتظر محاسبه Carry های مرحله قبل شود، تصمیم می‌گیرد که به ازای هر دو حالت Carry (صفر یا یک) حاصل جمع را حساب کند و وقتی Carry مرحله های قبل مشخص شد،‌ یکی از دو حاصل جمع و همچنین یکی از Carry های خروجی را به وسیله MUX انتخاب کرده و به خروجی بدهد.

در ادامه شماتیک این مدار را مشاهده می‌کنید.



## مقایسه جمع کننده ها

Cost(RA) = n \* Cost(FA) = n \* (1CXor + 3CAnd + 1COr) = 5nC

Delay(RA)Sum = (n - 1)2d + d = (2n-1)d

Delay(RA)Carry out = 2nd

در این جمع کننده چون تاخیر و همچنین هزینه به تعداد بیت ورودی وابسته است، اگر تعداد بیت بالا برود هم از نظر هزینه و هم از نظر تاخیر ممکن است به صرفه نباشد.  
اما در عوض پیاده سازی ساده تری دارد و اگر تعداد بیت هایی که در سخت افزار استفاده می‌شود کم باشد می‌تواند گزینه مناسبی باشد.

در CLA همانطور که بیان شد، تاخیر 4d بدست آمد که این تاخیر چون با تعداد بیت ورودی وابسته نیست، با افزایش تعداد بیت همچنان تاخیر ثابتی را دارد که مزیت این جمع کننده به حساب می‌آید اما مشکل اساسی این جمع کننده در پیاده سازی آن است.

چون ما اولا فرض کردیم گیت ها با هرچند ورودی تاخیر یکسانی دارند و دوما اینکه این گیت ها قابل پیاده سازی است اما در عمل تاخیر یک گیت با تعداد ورودی های متفاوت، زمانی متفاوت دارد و همچنین پیاده سازی گیت هایی با بیش از ۴ ورودی امکان پذیر نمی‌باشد یا به صرفه نیست. در عمل از ترکیبی از RA و CLA استفاده می‌شود.

در CSA هم از نظر هزینه و هم از نظر تاخیر به صرفه نمی‌باشد و صرفا برای کاربرد های خاصی ممکن است بهینه باشد.

به این دلیل که اولا از نظر هزینه باید دو جمع کننده استفاده شود و علاوه بر آن باید دو MUX برای تعیین خروجی ها استفاده شود و دوما اینکه تاخیر MUX ها چون زیاد است عملا هدف استفاده از این روش را زیر سوال میبرد بدین صورت که نه تنها باعث سریع تر شدن نمی‌شود، بلکه باعث کندی مدار می‌شود.

## 2-Digits BCD Seven Segment

پیاده سازی آن همانند پیاده سازی مدار یک بیتی متناظرش است و بدین منظور از دو Seven Segment که به طور مستقل کار می کنند استفاده می‌کنیم و در هر مرحله هر رقم را به صورت جدا بررسی می‌کنیم.

پس باقی مانده BCD ورودی را به ۱۰ گرفته و به عنوان رقم یکان به مدار معرفی می‌کنیم.

و همچنین برای بدست آوردن دهگان عدد ورودی را بر ۱۰ تقسیم می‌کنیم.

همچنین در هر مرحله چک می‌کنیم که اگر عدد ورودی داده شده از ۹۹ بزرگتر است، ورودی ما معتبر نیست و هر کدام از رقم ها باید Null باشند

در ادامه کد مدار و تست بنچ آن را مشاهده می‌کنید.

library IEEE;

use IEEE.STD\_LOGIC\_1164.all;

use IEEE.NUMERIC\_STD.all;

entity two\_digit\_bcd\_7segment is

    port (

        BCDin       : in std\_logic\_vector (6 downto 0);

        firstDigit  : out std\_logic\_vector (6 downto 0);

        secondDigit : out std\_logic\_vector (6 downto 0)

    );

end two\_digit\_bcd\_7segment;

architecture Behavioral of two\_digit\_bcd\_7segment is

begin

    process (BCDin)

    begin

        if to\_integer(unsigned(BCDin)) < 100 then

            case to\_integer(unsigned(BCDin)) mod 10 is

                when 0 =>

                    firstDigit <= "0000001"; *--0*

                when 1 =>

                    firstDigit <= "1001111"; *--1*

                when 2 =>

                    firstDigit <= "0010010"; *--2*

                when 3 =>

                    firstDigit <= "0000110"; *--3*

                when 4 =>

                    firstDigit <= "1001100"; *--4*

                when 5 =>

                    firstDigit <= "0100100"; *--5*

                when 6 =>

                    firstDigit <= "0100000"; *--6*

                when 7 =>

                    firstDigit <= "0001111"; *--7*

                when 8 =>

                    firstDigit <= "0000000"; *--8*

                when 9 =>

                    firstDigit <= "0000100"; *--9*

                when others =>

                    firstDigit <= "1111111"; *--null*

            end case;

            case to\_integer(unsigned(BCDin)) / 10 is

                when 0 =>

                    secondDigit <= "0000001"; *--0*

                when 1 =>

                    secondDigit <= "1001111"; *--1*

                when 2 =>

                    secondDigit <= "0010010"; *--2*

                when 3 =>

                    secondDigit <= "0000110"; *--3*

                when 4 =>

                    secondDigit <= "1001100"; *--4*

                when 5 =>

                    secondDigit <= "0100100"; *--5*

                when 6 =>

                    secondDigit <= "0100000"; *--6*

                when 7 =>

                    secondDigit <= "0001111"; *--7*

                when 8 =>

                    secondDigit <= "0000000"; *--8*

                when 9 =>

                    secondDigit <= "0000100"; *--9*

                when others =>

                    secondDigit <= "1111111"; *--null*

            end case;

        else

            firstDigit  <= "1111111"; *--null*

            secondDigit <= "1111111"; *--null*

        end if;

    end process;

end Behavioral;

library ieee;

use ieee.std\_logic\_1164.all;

entity two\_digit\_tb\_bcd\_7seg is

end two\_digit\_tb\_bcd\_7seg;

architecture behavior of two\_digit\_tb\_bcd\_7seg is

*-- Component Declaration for the Unit Under Test (UUT)*

    component two\_digit\_bcd\_7segment

        port (

            BCDin       : in std\_logic\_vector (6 downto 0);

            firstDigit  : out std\_logic\_vector (6 downto 0);

            secondDigit : out std\_logic\_vector (6 downto 0)

        );

    end component;

*--Inputs*

    signal BCDin       : std\_logic\_vector(6 downto 0) := (others => '0');

*--Outputs*

    signal firstDigit  : std\_logic\_vector(6 downto 0);

    signal secondDigit : std\_logic\_vector(6 downto 0);

begin

*-- Instantiate the Unit Under Test (UUT)*

    uut : two\_digit\_bcd\_7segment port map(

        BCDin       => BCDin,

        firstDigit  => firstDigit,

        secondDigit => secondDigit

    );

*-- Stimulus process*

    stim\_proc : process

    begin

        BCDin <= "0000000"; *-- 0*

        wait for 100 ns;

        BCDin <= "0000011"; *-- 3*

        wait for 100 ns;

        BCDin <= "0000110"; *-- 6*

        wait for 100 ns;

        BCDin <= "0000111"; *-- 7*

        wait for 100 ns;

        BCDin <= "0001001"; *-- 9*

        wait for 100 ns;

        BCDin <= "0001010"; *-- 10*

        wait for 100 ns;

        BCDin <= "0100101"; *-- 37*

        wait for 100 ns;

        BCDin <= "0111111"; *-- 63*

        wait for 100 ns;

        BCDin <= "1100011"; *-- 99*

        wait for 100 ns;

        BCDin <= "1111111"; *-- 127*

        wait for 100 ns;

    end process;

end;

## Wave خروجی برنامه

Wave خروجی Test Becnh برنامه ما به صورت زیر است:



همانطور که مشاهده شد، خروجی به درستی مشخص شده است.

برای مثال در ۴ تست کیس اول که اعداد یک رقمی می‌باشند،‌ رقم دهگان صفر را نشان می‌دهد و رقم یکان مقدار درستی دارد.

همچنین به ازای عدد ۱۲۷ در آخرین تست کیس، چون عدد بزرگتر از ۱۰۰ است،‌ ورودی نادرستی است و مقدار ۱۱۱۱۱۱۱ برای هر کدام منظور شده است که بیانگر نال است.