**“电子设计自动化(EDA)”实验报告(六)**

姓名 刘宗鑫 学号 201710413046 日期 2020.06.12

**一、实验名称：**使用Component设计计数器和加法器

**二、实验内容和实验目的**

实验内容：

1．设计一个硬件可操作的模24计数器，并在数码管上显示。

1) 用VHDL 语句设计一个十进制counter24计数器。用Cyclone II EP2C35F672C6芯片实现。

2) 用VHDL语句设计一个segment4to7显示译码器,驱动共阳极数码管。用Cyclone II EP2C35F672C6芯片实现。

3) 用VHDL component语句设计一个顶层文件top01.vhd , 实现硬件可操作的模24计数器，带en控制。用Cyclone II EP2C35F672C6芯片实现，按DE2开发板的实际连线来输入输出信号的引脚。

2．设计一个4bit 加法器adder4bits (纯仿真,不需要硬件实验箱)

方法1，先设计一个1bit全加器(fulladder1bit),然后把fulladder1bit当作component使用，构成4bit全加器。

方法2，直接用无符号数算术相加实现。

3．设计一个由移位寄存器构成的序列信号发生器 (纯仿真,不需要硬件实验箱)，用component语句实现。

实验目的：掌握FPGA硬件开发的基本技能，包括通过USB-Blaster下载线把sof文件下载到FPGA芯片，进行硬件功能验证。更熟练地使用VHDL设计数字电路。

**三、实验步骤、实现方法(或设计思想)及实验结果**

( 实验步骤：与实验指导书中的实验步骤相同，但需要添加细节步骤；

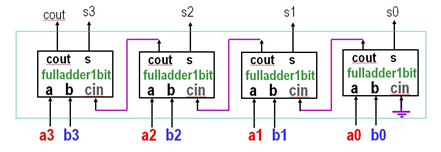
实现方法：附上原理图，或程序代码或程序流程，并简要说明；

实验结果：附上仿真结果，并解释说明)

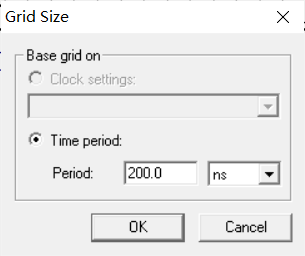
实验步骤：

**实验二**

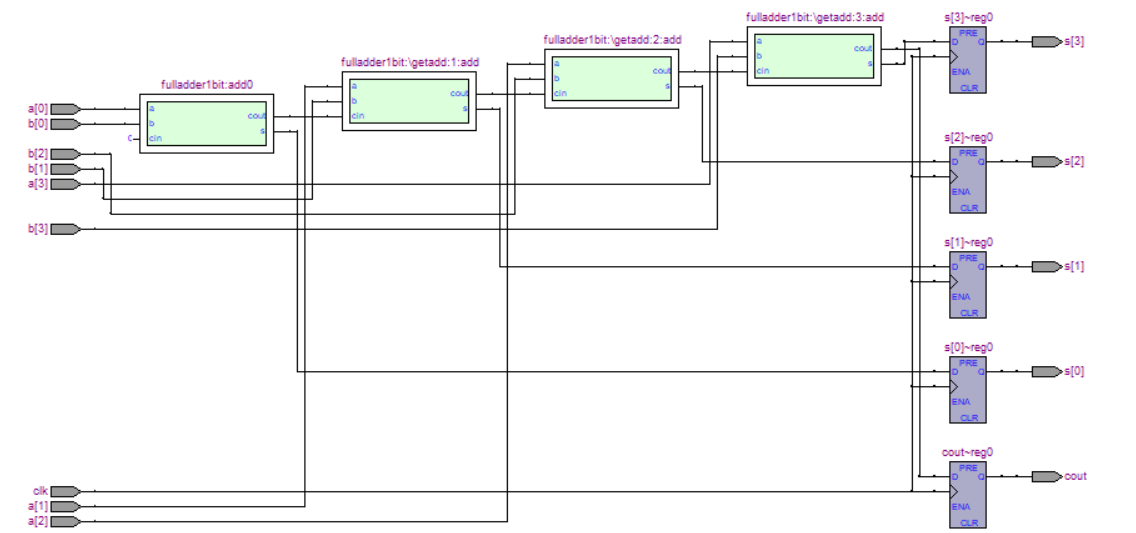
方法1，先设计一个1bit全加器(fulladder1bit),然后把fulladder1bit当作component使用，构成4bit全加器(adder4bits)，原理如下所示。



1. 首先编写代码实现1bit全加器。
2. 再次编写代码通过1bit全加器实现4bit全加器。
3. 仿真验证，看看是不是能正确做加法运算。
   1. 添加并设置波形
   2. 设置时间

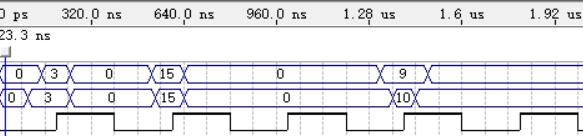


1. 查看 quartusII 编译器根据VHDL代码，生成的电路原理逻辑图。



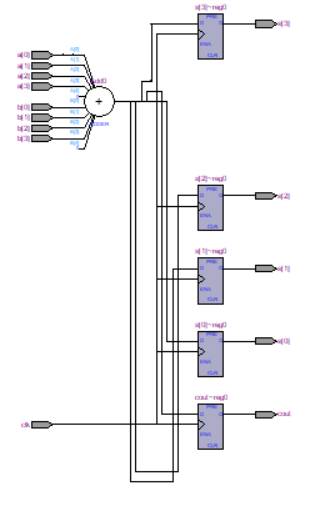
方法2，直接用无符号数算术相加实现。建议不要建立两个project,使用configuration语句选择两个结构体（architecture）中的一个就可以了。

1. 仿真验证，看看是不是能正确做加法运算。
   1. 添加并设置波形



* 1. 进行仿真

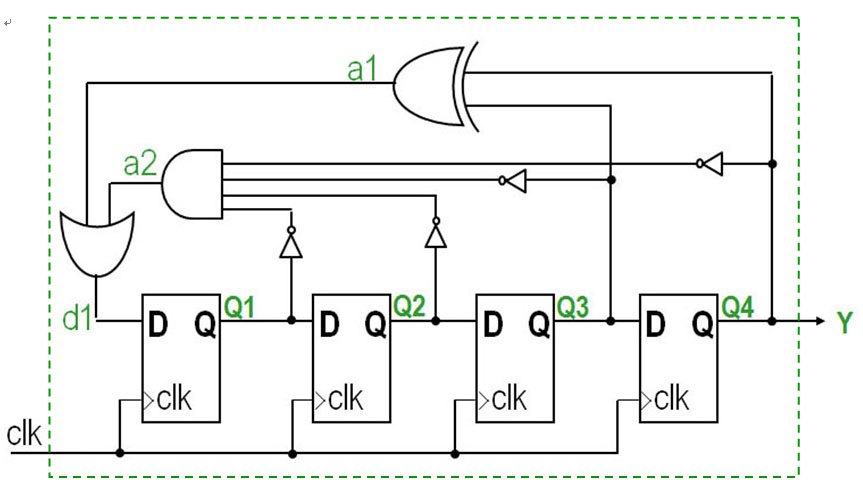
1. 查看 quartusII 编译器根据VHDL代码，生成的电路原理逻辑图。



**实验二**

方法一

1. 设计一个D触发器（例如mydff），然后把这个D触发器当作元件。



1. 根据电路图添加代码实现功能。
2. 设置波形，进行仿真。



方法二

1. 设计D触发器
2. 使用generate语句建立其余触发器。
3. 设置波形，并进行仿真。



实现方法：

**实验二**

|  |  |
| --- | --- |
| **fulladder1bit** | library ieee;  use ieee.std\_logic\_1164.all;  use ieee.std\_logic\_unsigned.all;  entity fulladder1bit is  port  (      cout, s: out std\_logic;      a, b, cin : in std\_logic  );  end fulladder1bit;  architecture arc of fulladder1bit is  begin      process( a, b, cin )          variable tmp : std\_logic;      begin          tmp := a or b;          if a = '1' and b = '1' then              cout <= '1'; s <= cin;          elsif tmp = '1' and cin = '1' then              cout <= '1'; s <= '0';          else              cout <= '0'; s <= tmp or cin;          end if;      end process;  end arc; |
| **fulladder4bit** | library ieee;  use ieee.std\_logic\_1164.all;  use ieee.std\_logic\_unsigned.all;  entity adder4bits is  port  (      clk: in std\_logic;      cout: out std\_logic;      a, b: in std\_logic\_vector(3 downto 0);      s: out std\_logic\_vector(3 downto 0)  );  end adder4bits;  architecture arc1 of adder4bits is      component fulladder1bit      port(          cout, s: buffer std\_logic;          a, b, cin: in std\_logic      );      end component;      signal cout\_tmp: std\_logic;      signal tmp: std\_logic\_vector( 3 downto 0 );      signal s\_tmp: std\_logic\_vector( 3 downto 0 );  begin        add0: fulladder1bit port map( tmp(0), s\_tmp(0), a(0), b(0), '0' );      getadd: for i in 1 to 3 generate          add: fulladder1bit port map(  tmp(i), s\_tmp(i), a(i), b(i), tmp(i - 1) );      end generate getadd;      process( clk )      begin          if clk'event and clk = '1' then              cout <= tmp(3);              s <= s\_tmp;          end if;      end process;  end arc1;  architecture arc2 of adder4bits is  begin      process( clk )          variable tmp: std\_logic\_vector( 3 downto 0 );      begin          if clk'event and clk = '1' then              tmp := a + b;              if ( tmp < a ) and ( tmp < b ) then                  cout <= '1';              else                  cout <= '0';              end if;              s <= tmp;          end if;      end process;  end arc2;  architecture arc3 of adder4bits is  begin      process( clk )          variable a5bit, b5bit, ans: std\_logic\_vector( 4 downto 0 );      begin          if clk'event and clk = '1' then              a5bit := '0' & a;              b5bit := '0' & b;              ans := a5bit + b5bit;              s(3 downto 0) <= ans( 3 downto 0 );              cout <= ans(4);          end if;      end process;  end arc3;  configuration cfg of adder4bits is      for arc3      end for;  end cfg; |

**说明**

* 对于1bit全加器，直接使用条件语句输出真实值。
* 对于4bit全加器，使用了三种方式进行实现
  + 使用generate语句对1bit全加器进行级联。
  + 使用判断语句判断结果是否溢出，如果结果溢出，则说明有进位。
  + 使用5bit的变量来暂存结果。通过最高位来看是否会发生进位。

**实验三**

|  |  |
| --- | --- |
| **mydiff** | library ieee;  use ieee.std\_logic\_1164.all;  use ieee.std\_logic\_unsigned.all;  entity mydff is  port  (      q : out std\_logic;      d, clk : in std\_logic  );  end mydff;  architecture arc of mydff is  begin      process( clk )      begin          if clk'event and clk = '1' then              q <= d;          end if;      end process;  end arc; |
| **generator** | library ieee;  use ieee.std\_logic\_1164.all;  use ieee.std\_logic\_unsigned.all;  entity generator is  port  (      y : out std\_logic;      clk: in std\_logic  );  end generator;  architecture arc1 of generator is      component mydff      port(          q : out std\_logic;          d, clk : in std\_logic      );      end component;      signal d1, q1, q2, q3, q4, a1, a2: std\_logic;  begin      dff0: mydff port map( q1, d1, clk );      dff1: mydff port map( q2, q1, clk );      dff2: mydff port map( q3, q2, clk );      dff3: mydff port map( q4, q3, clk );        a1 <= q3 xor q4;      a2 <= not ( q4 or q3 or q2 or q1 );      d1 <= a1 or a2;      y <= q4;  end arc1;  architecture arc2 of generator is      component mydff      port(          q : out std\_logic;          d, clk : in std\_logic      );      end component;      signal d1, a1, a2: std\_logic;      signal q: std\_logic\_vector( 4 downto 1 );  begin      dff0: mydff port map( q(1), d1, clk );      getD: for i in 1 to 3 generate          D: mydff port map( q(i + 1), q(i), clk );      end generate getD;        a1 <= q(3) xor q(4);      a2 <= not ( q(4) or q(3) or q(2) or q(1));      d1 <= a1 or a2;      y <= q(4);  end arc2;  configuration cfg of generator is      for arc2      end for;  end cfg; |

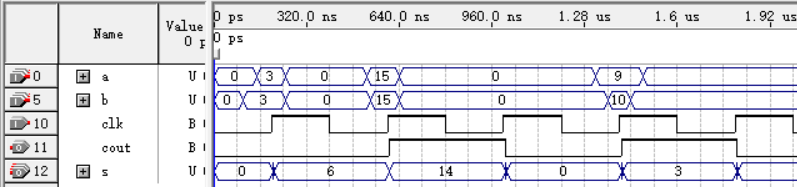
**说明：**

* **第一种方法手动连接所有的端口**
* **第二种方法使用generator语句连接触发器。**

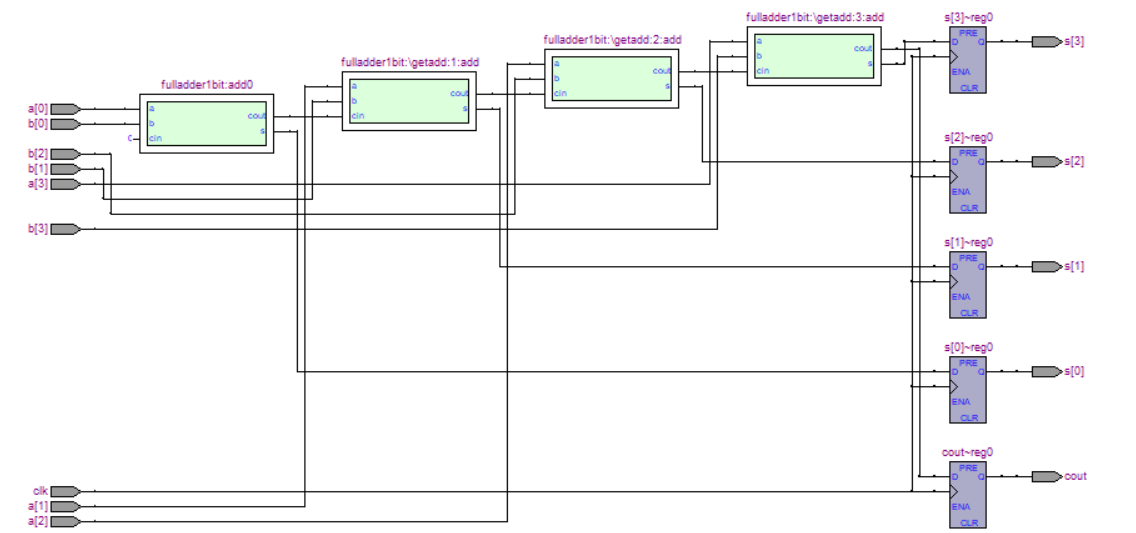
实验结果：

**实验二**

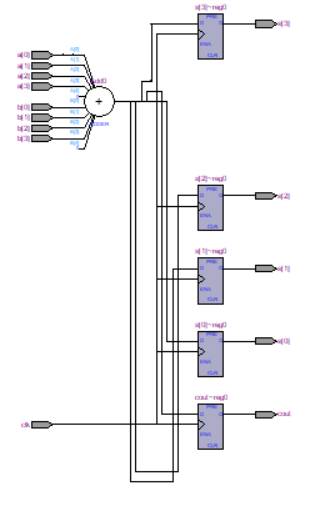
**所有方法仿真波形一致。**



**方法一**

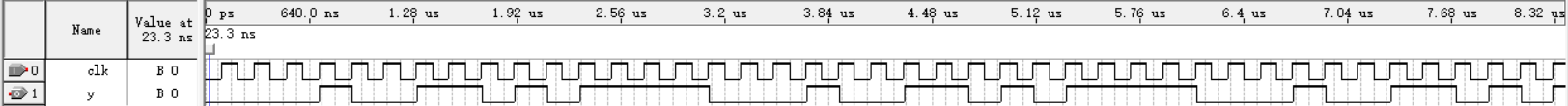


**方法二**

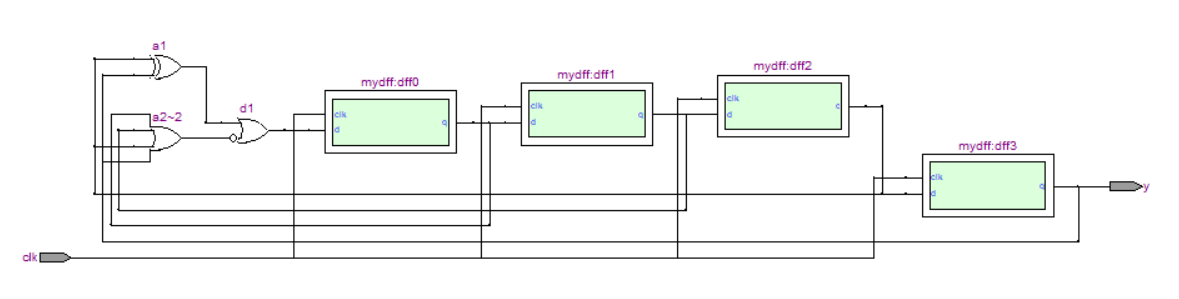


**实验三**

**所有方法仿真波形一致。**



**电路图**



**四、总结**

* **使用generate语句可以大大减轻工作量。**
* **编写程序时需要考虑硬件本身，而不能仅思考软件实现。**

**五、思考题**

思考题 1：本实验所用的芯片EP2C35F672C6里面包含多少个逻辑单元LE ( Logic Cell)? 芯片内部，每16个LE组成一个块(Block), 也称作逻辑阵列块 LAB (Logic Array Block)。问，芯片内部，共有多少个LAB?这些 LAB排列成多少行？多少列？编译project正确之后，可以通过 Tools -> chip planner 查看芯片内部的LAB排列情况和使用情况。用键盘的ctrl键+鼠标滚轮，可以放大或缩小视图，以便清晰地查看LAB的排列布局。

共33216个逻辑单元

共2076个逻辑阵列块

36行，65列

思考题 2：问,本实验的project一共用掉了芯片中的多少个LE? 占用了总资源的百分之几？这些用掉的LE所在的LAB是第几行？第几列？可以通过 Tools -> chip planner查看，鼠标稍微停留在LAB上时，就会自动显示LAB 的行序号、列序号。当然也可以从编译报告中去查看。

一共用掉了芯片中的32个LE，不到1%，

可以查看到如

列，行

2，14

3，14

4，14等lab被使用

思考题 3： (最后做，做完其他题之后再做思考题3), 实验板上提供了一个50MHz的晶体振荡器，已经连接到了PIN\_N2引脚，可以替代手动按键提供时钟。由于50MHz频率太快了，需要增加一个分频器模块(参见图1中counter24的clk端口下面的绿色虚框)，把50MHz分频成1Hz, 再送给counter24做时钟。请编写代码设计一个fdivider模块，fdivider的输入端(clk50MHz)连接到PIN\_N2的50MHz晶振，fdivider的输出端(clk1Hz) 1Hz信号送给counter24,这样就不用手动按键提供时钟了。

|  |
| --- |
| library ieee;  use ieee.std\_logic\_1164.all;  use ieee.std\_logic\_unsigned.all;  entity f\_divider is  port(      clk:in std\_logic;      cout:out std\_logic);  end f\_divider;  architecture arc of f\_divider is  signal cnt:integer range 0 to 50000000;  begin      process(clk)      begin          if(clk'event and clk='1')then              if(q=49999999)then                  cout <= '1';                  cnt <= 0;              else                  cnt <= cnt + 1;                  cout <= '0';              end if;          end if;      end process;      end arc; |