目录

简单 CPU 硬件结构和微指令设计

尹达恒
（江南大学物联网工程学院，江苏　无锡）

1　设计要求

- 设计一个由微指令控制的简单加法 CPU，使其能从输入寄存器中先后读入两个数进行相加，并将结果存入输出寄存器；
- 设计一个由微指令控制的循环加法 CPU，使其能从输入寄存器中先后读入多个数进行相加，并将结果存入输出寄存器。

2　总体设计

2.1　硬件结构

根据设计要求，简单 CPU 硬件结构将包含以下元件：

- 一个控制器；
- 一个数据总线；
- 一个算术逻辑单元；
- 四个寄存器（两个数据寄存器、一个输入寄存器、一个输出寄存器）。

总体设计图如图 1。图中各部分作用如下：

- 输入寄存器：存储将要输入到数据总线中的值；
- 数据寄存器 1/2：存储将要进行算术逻辑运算的值；
- 输出寄存器：存储将要输出的值；
- 算术逻辑单元：对数据寄存器 1/2 中的值进行算术逻辑运算；
- 数据总线：负责数据在寄存器之间的转移；
- 控制器：对微指令进行译码，控制上述所有部分的动作。

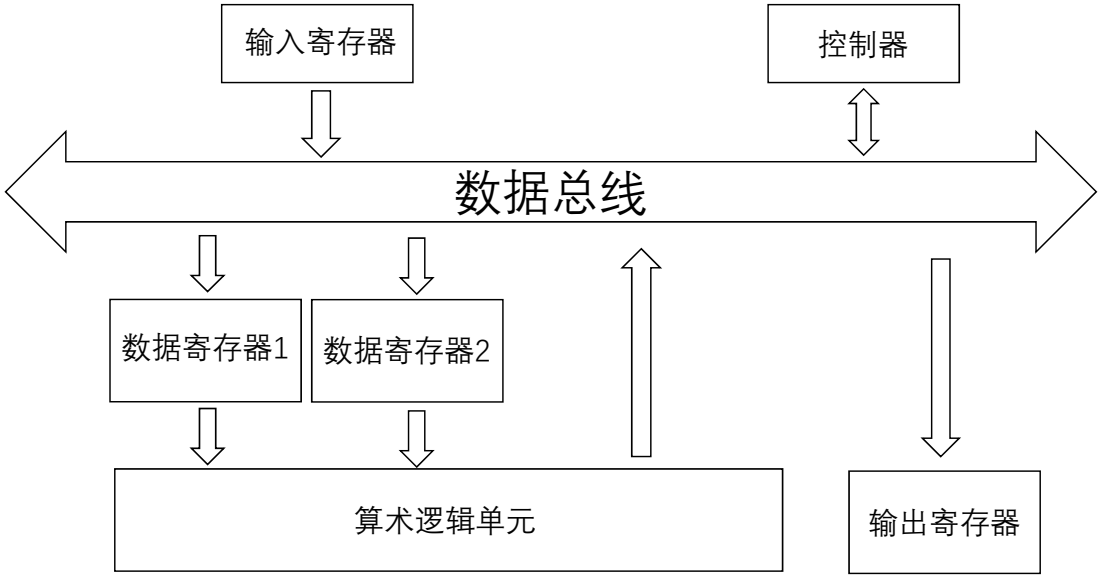

图 1. 硬件总体结构

2.2 指令结构

根据设计要求，简单 CPU 指令集将至少包含以下功能：

- 将输入寄存器中的值存入数据总线中；
- 将数据总线中的值存入数据寄存器 1/2 或输出寄存器中；
- 将算术逻辑单元中的运算结果存入数据总线中。

且为了能够使用实验板上的开关阵列进行实验，CPU 指令集按顺序执行的相邻指令之间必须只能有一位不同。

3 硬件结构设计

3.1 各元件选型及特性

3.1.1 输入寄存器元件 reg_74244

输入寄存器使用 ISE 自带的寄存器元件 reg_74244，其元件示意图如图 2。该元件在输出使能端口 oen 为低电平时使输出端口 Qout(7:0) 的 8 位电平状态与输入端口 Din(7:0) 相同。

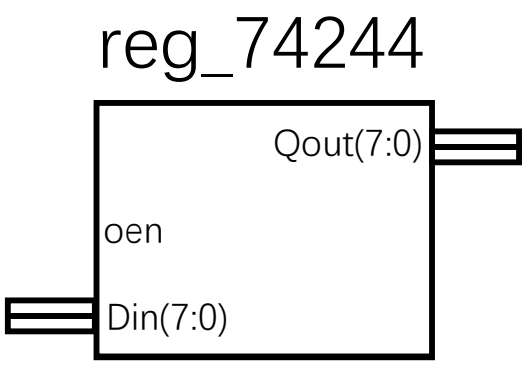

图 2. reg_74244 元件示意图

3.1.2 数据寄存器 1/2 和输出寄存器元件 reg_74373

数据寄存器和输出寄存器使用 ISE 自带的寄存器元件 reg_74373，其元件示意图如图 3。该元件在时钟端口 clk 输入一个时钟脉冲时有以下动作：

- 若输出使能端口 oen_n 为低电平，则使输出端口 Qout(7:0) 的 8 位电平状态与内部寄存值相同；
- 若输入使能端口 gwe 为高电平，则将输入端口 Din(7:0) 的值写入内部寄存器。

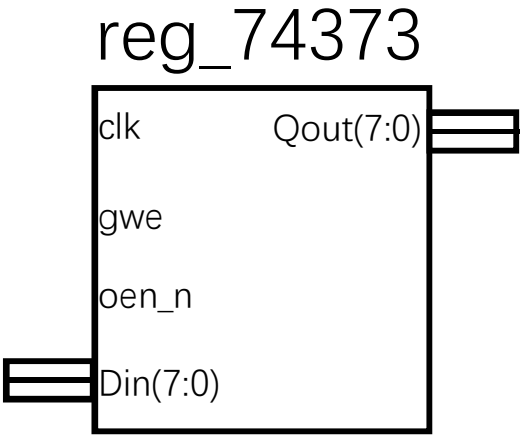

图 3. reg_74373 元件示意图

3.1.3 算术逻辑单元 alu_74181

由于 ISE 自带的算术逻辑单元 alu_74181 为四位运算器，无法满足本设计方案中八位运算器的需求，因此需要进行修改，修改后的 VHDL 代码如附录 A所示，其元件示意图如图 4。该元件的输入端口 C_n、A(7:0)、B(7:0) 和输出端口

C_n_plus4、F(7:0) 的关系受输入端口 S(4:0) 的值控制。其中当 S(4:0) 端口为 0110 时，F=C_n+A+B。

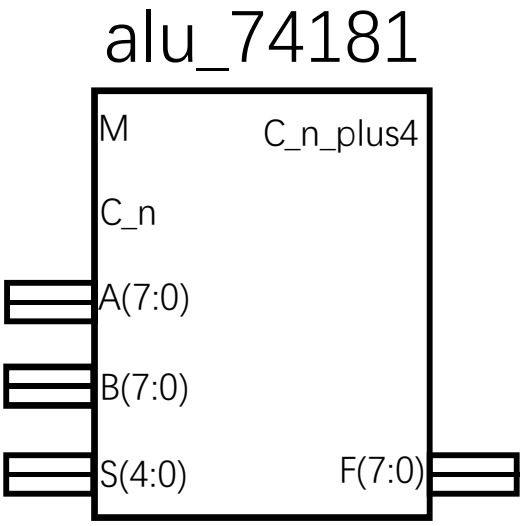

图 4. alu_74181 元件示意图

3.1.4 数据总线 data_bus

数据总线使用 ISE 自带的数据总线元件 data_bus，其元件示意图如图 5所示。该元件在时钟端口 clk 输入一个时钟脉冲时有以下动作：

- 若端口 we1 为高电平，则将端口 data_in1(7:0) 的值写入内部寄存器；
- 若端口 we1 为低电平且端口 we2 为高电平，则将端口 data_in2(7:0) 的值写入内部寄存器；
- 若端口 we1、we2 为低电平且端口 we3 为高电平，则将端口 data_in3(7:0) 的值写入内部寄存器；
- 若端口 we1、we2、we3 为低电平且端口 we4 为高电平，则将端口 data_in4(7:0) 的值写入内部寄存器；
- 若端口 we1、we2、we3、we4 为低电平且端口 we_io1 为高电平，则将端口 data_io1(7:0) 的值写入内部寄存器；
- 若端口 we1、we2、we3、we4、we_io1 为低电平且端口 we_io2 为高电平，则将端口 data_io2(7:0) 的值写入内部寄存器；
- 使四个八位输出端口 data_out1/2/3/4(7:0) 值等于 data_bus 内部寄存器的值，若端口 we1、we2、we3、we4、we_io1、we_io2 均为低电平，则使输出端口 data_io1/2(7:0) 的值等于内部寄存器的值。

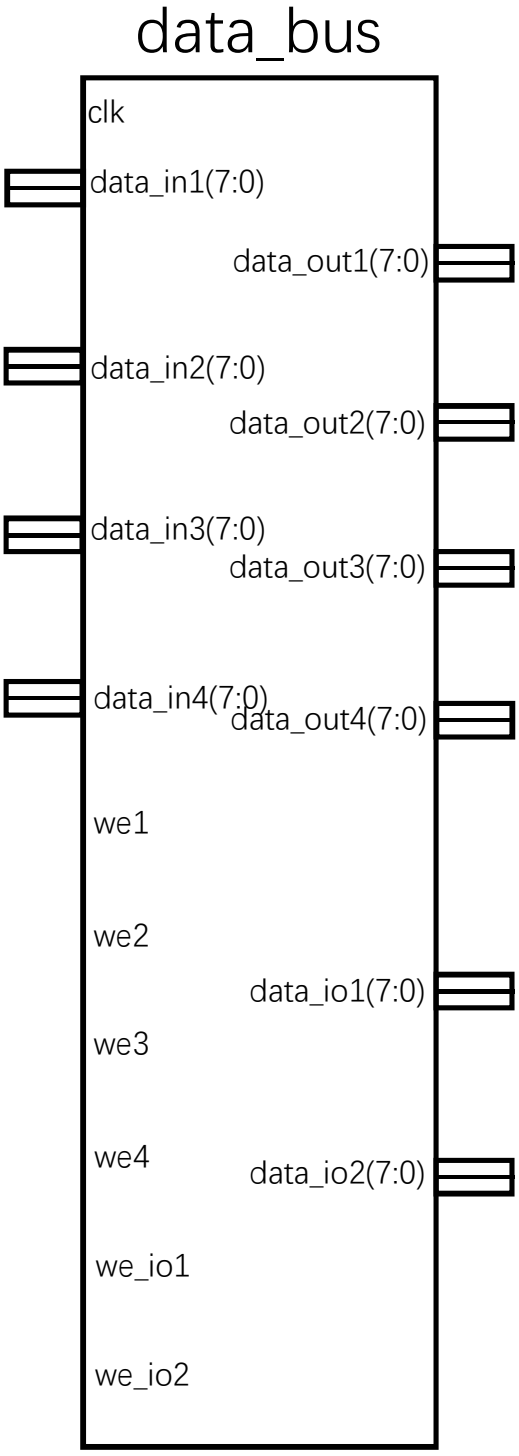

图 5. data_bus 元件示意图

3.1.5 控制器 romc

由于 ISE 自带的算术逻辑单元 romc 为二四译码器，无法满足本设计方案的需求，因此需要进行修改，修改后的 VHDL 代码如附录 B和附录 C，其元件符号如图 6。romc 本质上是一个 4-9 译码器，输入部分 S0/S1/S2/S3 为四位微指令，剩下 9 位输出分别接在各元件上控制元件工作。romc 元件控制位如表 1所示。

表 1. romc 元件控制位表

输出位	8	7	6	5	4	3	2	1	0
控制位	OENN3	GWE3	OENN2	GWE2	OENN1	GWE1	WE2	WE1	OEN

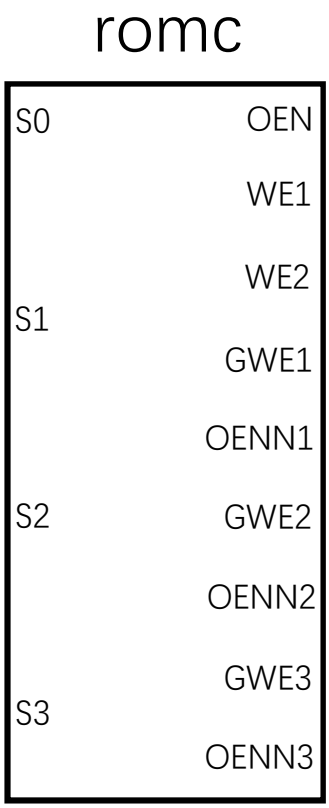

图 6. romc 元件示意图

3.2 元件连接

3.2.1 控制器 romc 的连接方案

- OEN 接输入寄存器 reg_74244 的使能端 oen，控制输入寄存器的使能；
- WE1/2 分别接数据总线 data_bus 的输入使能端 we1/2，控制总线读取输入寄存器的值或算术逻辑单元的运算结果；

- GWE1/2 和 OENN1/2 分别接数据寄存器 1/2 的输入使能端 gwe 和输出使能端 oen_n，控制数据寄存器 1/2 从数据总线中取数和输出到算术逻辑单元；
- GWE3 和 OENN3 分别接输出寄存器的输入使能端 gwe 和输出使能端 oen_n，控制输出寄存器从数据总线中取数和输出。

3.2.2 其他元件的连接方案

- 输入寄存器 reg_74244：数据输入端口 Din 接外部数据输入端口 DATA_IN(7:0)、输出端口 Qout 接数据总线输入端口 data_in1；
- 算术逻辑单元 alu_74181：运算控制端口 S 接外部数据输入端口 CTL(3:0)、输入端口 A、B 分别接数据寄存器 1/2 的输出端口 Qout、输出端口接数据总线输入端口 data_in2；
- 输出寄存器 reg_74244：输入端口 Din 接数据总线输出端口 data_out3、输出端口 Qout 接数据输出端口 DATA_OUT(7:0)；
- 数据总线 data_bus：we3、we4、we_io1、we_io2 接地、data_out1/2 分别接数据寄存器 1/2 的输入端口 Din。

由上述连接方案得到的元件连接示意图如图 7所示。

3.2.3 外部输入/输出的连接方案

- 指令输入：romc 四个指令输入端口 S0/1/2/3 分别接 Atlys 开发板上的开关 sw0/1/2/3；
- 时钟输入：数据总线 data_bus、数据寄存器 1/2 和输出寄存器 reg_74373 的 clk 端口接外部时钟输入端口；
- 数据输入：外部数据输入端口 DATA_IN(7:0) 接 EES261 的开关 SW9～SW16；
- 算术逻辑单元运算控制输入：外部数据输入端口 CTL(3:0) 在 EES261 的开关 SW5～SW8；
- 数据输出：数据输出端口 DATA_OUT(7:0) 接 EES261 的 LED 灯 LED1～LED8。

由上述连接方案编写的外部端口定义文件如附录 D。

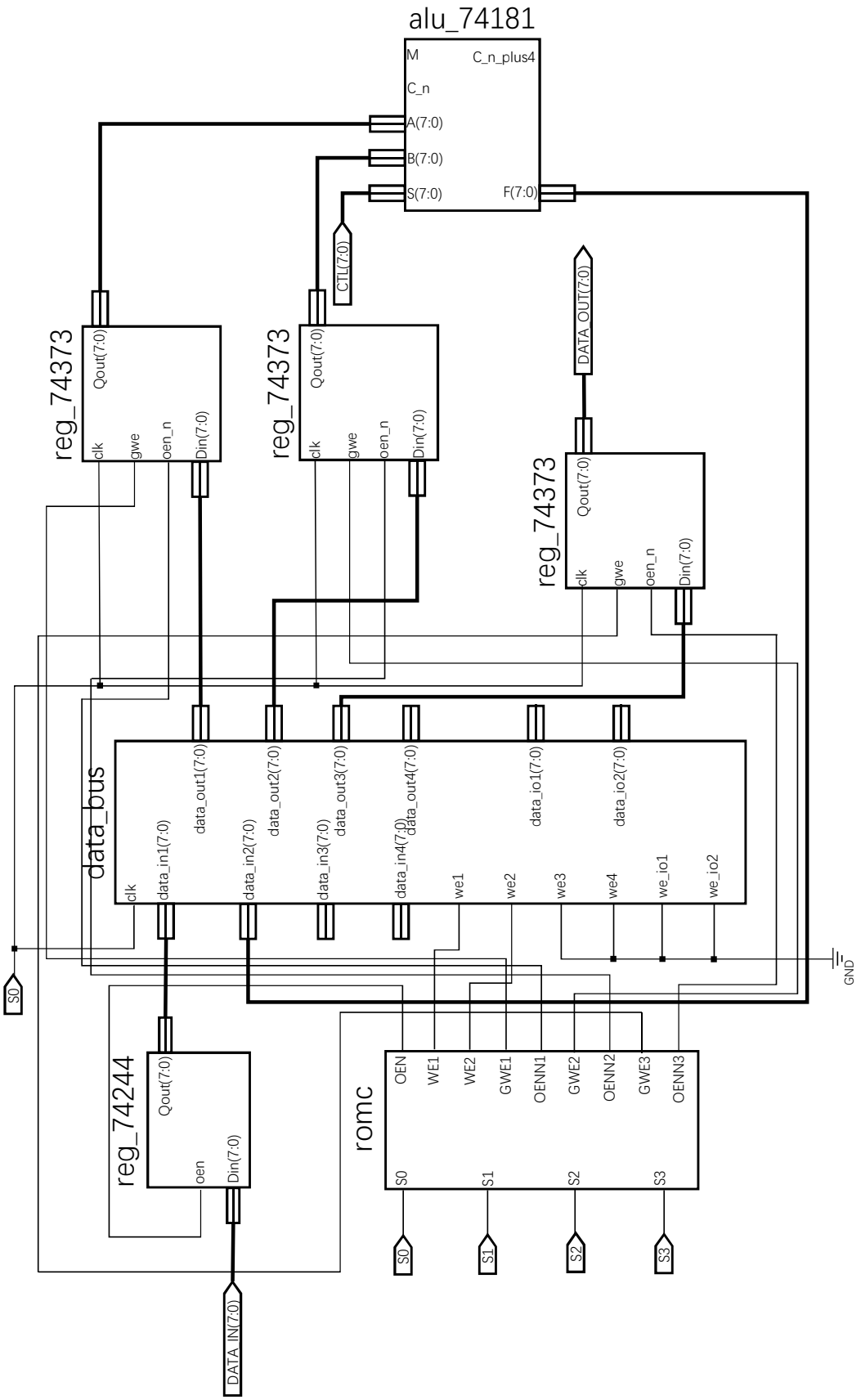

图 7. 元件的连接方案示意图

4 指令结构设计

4.1 简单加法器指令结构设计

由 3.1节所示的元件特性以及 3.2.1节可知，一个加法操作的步骤以及表 1中对应的 romc 在一个时钟周期内输出值如下（0 表示高电平，1 表示低电平，X 表示可以为任意值）：

1、 总线从输入寄存器中取出第一个数，romc 输出 XXXXXXX10；

2、 总线将取出的数放入数据寄存器 1 中，romc 输出 XXXXX1XXX；

3、 总线从输入寄存器中取出第二个数，romc 输出 XXXXXXX10；

4、 总线将取出的数放入数据寄存器 2 中，romc 输出 XXX1XXXXX；

5、 总线从算术逻辑单元中取出结果，romc 输出 XX0X0X10X；

6、 总线将取出的数放入输出寄存器中，romc 输出 X1XXXXXXX。

在上述步骤中，某些步骤间的任意值位不重合，可以进行合并，从而使指令更加精简，合并结果如下：

1、 总线从输入寄存器中取出第一个数并将取出的数放入数据寄存器 1 中，romc 输出 XXXXX1X10，经过两个时钟周期完成；

2、 总线从输入寄存器中取出第二个数并将取出的数放入数据寄存器 2 中，romc 输出 XXX1X0X10，经过两个时钟周期完成；

3、 总线从算术逻辑单元中取出结果并将取出的数放入输出寄存器中输出，romc 输出 01000010X，经过两个时钟周期完成。

指令和 romc 输出的对应关系不唯一。综合考虑，可以给出 romc 控制位输出的一个设计方案并为其分配微指令，得到简单加法器的一种指令结构及作用如表 2。以此编写的 romc 元件定义如附录 B所示。运算器的指令执行步骤是 $0000 \rightarrow 0001 \rightarrow 0011$。

4.2 循环加法器指令结构设计

循环加法器指令在简单加法器指令的基础上实现。当算术逻辑单元得出运算结果后，总线从算术逻辑单元中取出结果但不直接放入输出寄存器中，而是再

表 2. 简单加法器指令结构

步骤	微指令	romc 输出	作用
1	0000	101011010	总线从输入寄存器中取出一个数放入数据寄存器 1 中
2	0001	101110010	总线从输入寄存器中取出一个数放入数据寄存器 2 中
3	0011	010000100	总线从算术逻辑单元中取出结果放入输出寄存器中输出

放入数据寄存器中进行下一轮取数-运算的循环。和简单加法器类似，循环加法器指令结构及作用如表 3所示。

表 3. 循环加法器指令结构

步骤	微指令	romc 输出	作用
1、3	0000	101010010	总线从输入寄存器中取出一个数暂存于总线寄存器
2	0001	101011000	总线将暂存的数放入数据寄存器 1 中
4	0010	101110000	总线将暂存的数放入数据寄存器 2 中
5	0011	010000100	总线从算术逻辑单元中取出计算结果暂存并输出
6	0111	101110000	总线将暂存的计算结果放入数据寄存器 2 中
循环 1	1111	101011010	总线从输入寄存器中取出一个数放入数据寄存器 1 中
循环 2	1110	010000100	同 0011
循环 3	1100	101110000	同 0111
循环 4	1000	101011010	同 1111
循环 5	1001	010000100	同 0011
循环 6	1011	101110000	同 0111

以此编写的 romc 元件定义如附录 C所示。指令的执行顺序为：0000 → 0001 → 0000 → 0010 → 0011 → 0111，随后开始指令循环：1111 → 1110 → 1100 → 1000 → 1001 → 1011。每一轮指令循环都包含两轮“总线从输入寄存器中取出一个数放入数据寄存器 1 中 → 总线将暂存的计算结果放入数据寄存器 2 中 → 总线从算术逻辑单元中取出计算结果暂存并输出”的步骤循环，在每轮步骤循环的第一步输入不同的数，每一轮的运算结果在步骤循环的第二步中输出，即可在保证每一步微指令都只变动一位的情况下实现多个数的循环相加。

5 运行测试

5.1 线路连接

按图 7在 ISE 软件中进行元件线路连接，结果如图 8。

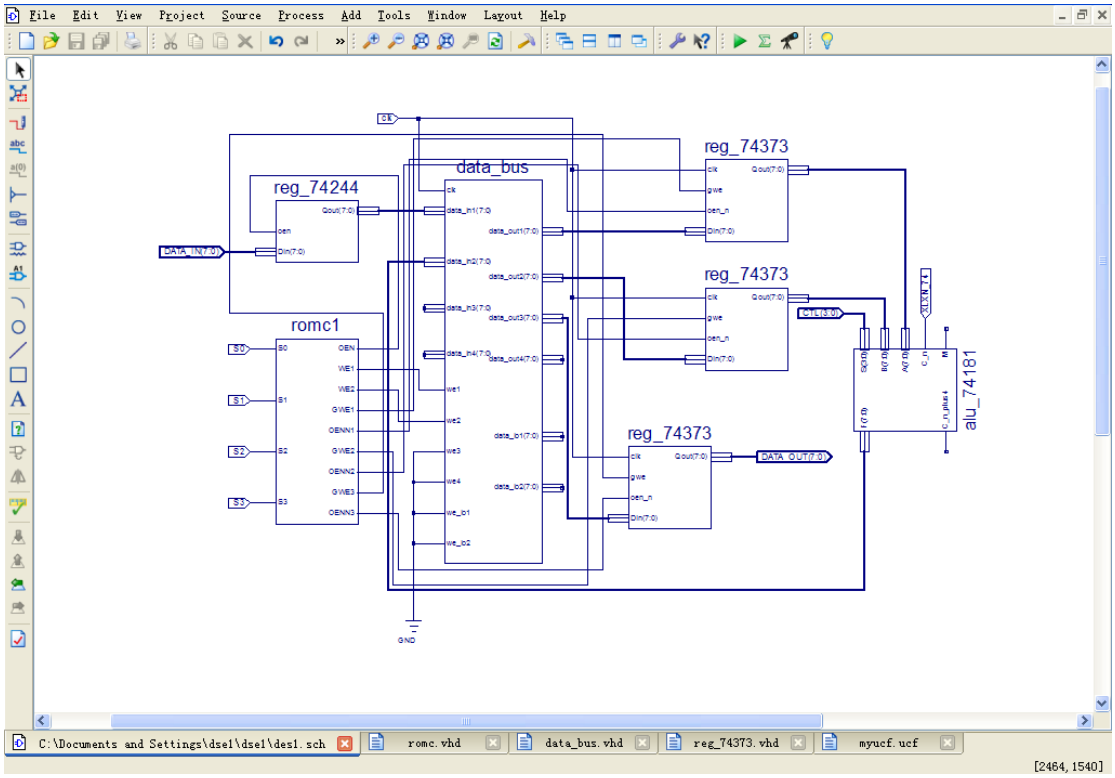

图 8. ISE 中的元件线路图

5.2 简单加法器测试

将元件线路图中的 romc 元件定义修改为附录 B中的元件定义，对工程进行编译并下载到 Xilink 测试平台，保持 CTL(3:0)（EES261 的开关 SW5～SW8）为 0110，按照 4.1节中给出的指令执行顺序改变微指令输入 S0/1/2/3（拨动 Atlys 开发板上的开关 sw0/1/2/3），在输入端 DATA_IN(7:0)（EES261 的开关 SW9～SW16）输入两个数相加，观察输出结果 DATA_OUT(7:0)（EES261 的 LED 灯 LED1～LED8）。

5.3 循环加法器测试

将元件线路图中的 romc 元件定义修改为附录 C中的元件定义，对工程进行编译并下载到 Xilink 测试平台，保持 CTL(3:0) 为 0110，按照 4.2节中给出的指令执行顺序改变微指令输入 S0/1/2/3，在输入端 DATA_IN(7:0) 输入多个数相加，观察每一步的输出结果 DATA_OUT(7:0)。

5.4 测试结果

5.4.1 简单加法器运行测试结果

当 DATA_IN 输入的两个数为 00000001 时，DATA_OUT 输出为 00000010，表明简单加法器 CPU 能正常工作。

5.4.2 循环加法器运行测试结果

当 DATA_IN 输入保持为 00000001 时，DATA_OUT 在每次循环中都会加一，表明循环加法器 CPU 能正常工作。

6 总结

6.1 遇到的问题

本次设计中遇到的主要问题是 romc 的 VHDL 程序输出位的顺序问题，开始测试时在 romc 的 VHDL 程序中将微指令译码得到的 romc 控制位输出写反导致测试结果不正确，错误排查花费了较多时间。

此外，本次设计中还遇到了循环运算的问题。由于输入指令的方式为手拨开关，因此每个命令之间都会经过多个时钟周期，若此时 romc 的 WE2 和 GWE1（或 GWE2）同时输出高电平，则总线会循环进行“从输入寄存器中取出一个数放入数据寄存器 1（或 2）中 → 从算术逻辑单元中取出运算结果放入数据寄存器 1（或 2）中”，导致数据寄存器内的值不断自增，使得输出结果不正确，错误排查花费了一定时间。

6.2 心得体会

- 更加深入地理解了计算机 CPU 的工作原理；
- 对计算机的体系结构有了更加深入的了解；
- 巩固了计算机组成原理的知识；
- 初步掌握了 VHDL 语言和 ISE 的使用方法。

附录

A alu_74181 修改后的 VHDL 程序

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

use IEEE.NUMERIC_STD.ALL;

entity alu_74181 is
  Port ( A : in  STD_LOGIC_VECTOR (7 downto 0);
       B : in  STD_LOGIC_VECTOR (7 downto 0);
       S : in  STD_LOGIC_VECTOR (3 downto 0);
       M : in  STD_LOGIC;
       C_n : in  STD_LOGIC;
       F : out  STD_LOGIC_VECTOR (7 downto 0);
       C_n_plus4 : out  STD_LOGIC);
end alu_74181;

architecture Behavioral of alu_74181 is

signal data_o_logic : STD_LOGIC_VECTOR (7 downto 0);
signal data_o_arith : STD_LOGIC_VECTOR (8 downto 0);

signal data_sub_tmp : STD_LOGIC_VECTOR (8 downto 0);

signal C_n_arith : STD_LOGIC_VECTOR (8 downto 0);

begin

  F <= data_o_logic when M = '1' else
       data_o_arith(7 downto 0);
  -- carry out
  C_n_plus4 <= not data_o_arith(4) when M = '0' else '1';

  C_n_arith <= "00000000" & (not C_n);

  -- 74181 logic operation
  process(A,B,S,M)
  begin
    case (S) is
      when "0000" =>
        data_o_logic <= not A;
      when "0001" =>
        data_o_logic <= not (A or B);
```

```
44        when "0010" =>
45          data_o_logic <= (not A) and B;
46        when "0011" =>
47          data_o_logic <= (others => '0');
48        when "0100" =>
49          data_o_logic <= not (A and B);
50        when "0101" =>
51          data_o_logic <= not B;
52        when "0110" =>
53          data_o_logic <= (A xor B);
54        when "0111" =>
55          data_o_logic <= A and (not B);
56        when "1000" =>
57          data_o_logic <= (not A) or B;
58        when "1001" =>
59          data_o_logic <= (A xnor B);
60        when "1010" =>
61          data_o_logic <= B;
62        when "1011" =>
63          data_o_logic <= A and B;
64        when "1100" =>
65          data_o_logic <= "00000001";
66        when "1101" =>
67          data_o_logic <= A or (not B);
68        when "1110" =>
69          data_o_logic <= A or B;
70        when "1111" =>
71          data_o_logic <= A;
72        when others =>
73          data_o_logic <= (others => '0');
74      end case;
75    end process;
76
77    -- 74181 arithmetic operation
78    process(A,B,S,M,C_n_arith)
79    begin
80      case (S) is
81        when "0000" =>
82          data_o_arith <= ('0'&A) + C_n_arith;
83        when "0001" =>
84          data_o_arith <= '0'&(A or B) + C_n_arith;
85        when "0010" =>
86          data_o_arith <= '0'&(A or (not B)) + C_n_arith;
87        when "0011" =>
88          -- if C_n = 0, minus 1,carry bit is 0; if C_n = 1,carry bit is 1;
89          data_o_arith <= "01111" + C_n_arith;
90        when "0100" =>
91          data_o_arith <= ('0'&A) + ('0'&(A and (not B))) + C_n_arith;
```

```
92        when "0101" =>
93          data_o_arith <= ('0'&(A or B))+('0'&(A and (not B)))+ C_n_arith;
94        when "0110" =>
95          -- if sub function, carry bit is different from add function
96          data_sub_tmp <= ('0'&A) - ('0'&B) - 1 + C_n_arith;
97          data_o_arith <= not data_sub_tmp(4) & data_sub_tmp(7 downto 0);
98        when "0111" =>
99          -- if sub function, carry bit is different from add function
100         data_sub_tmp <= ('0'&(A and (not B)))- 1 + C_n_arith;
101         data_o_arith <= not data_sub_tmp(4) & data_sub_tmp(7 downto 0);
102       when "1000" =>
103         data_o_arith <= ('0'&A) +('0'&(A and B))+ C_n_arith;
104       when "1001" =>
105         data_o_arith <= ('0'&A) + ('0'&B) + C_n_arith;
106       when "1010" =>
107         data_o_arith <= ('0'&(A or (not B))) + ('0'&(A and B))+ C_n_arith;
108       when "1011" =>
109         -- if sub function, carry bit is different from add function
110         data_sub_tmp <= ('0'&(A and B)) - 1 + C_n_arith;
111         data_o_arith <= not data_sub_tmp(4) & data_sub_tmp(7 downto 0);
112       when "1100" =>
113         data_o_arith <= ('0'&A) + ('0'&A) + C_n_arith;
114       when "1101" =>
115         data_o_arith <= ('0'&(A or B)) + ('0'&A) + C_n_arith;
116       when "1110" =>
117         data_o_arith <= ('0'&(A or (not B))) + ('0'&A) + C_n_arith;
118       when "1111" =>
119         -- if sub function, carry bit is different from add function
120         data_sub_tmp <= ('0'&A) - 1 + C_n_arith;
121         data_o_arith <= not data_sub_tmp(4) & data_sub_tmp(7 downto 0);
122
123       when others =>
124         data_o_arith <= (others => '0');
125     end case;
126   end process;
127 end Behavioral;
```

B 简单加法器 romc 修改后的 VHDL 程序

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

entity romc1 is
  Port ( S0 : in  STD_LOGIC;
       S1 : in  STD_LOGIC;
       S2 : in  STD_LOGIC;
       S3 : in  STD_LOGIC;
       OEN : out  STD_LOGIC;
       WE1 : out  STD_LOGIC;
       WE2 : out  STD_LOGIC;
       GWE1 : out  STD_LOGIC;
       OENN1 : out  STD_LOGIC;
       GWE2 : out  STD_LOGIC;
       OENN2 : out  STD_LOGIC;
       GWE3 : out  STD_LOGIC;
       OENN3 : out  STD_LOGIC);
end romc1;

architecture Behavioral of romc1 is
  signal addr  : std_logic_vector(3 downto 0);  --input
  signal rdata : std_logic_vector(8 downto 0);  --output
begin

addr <= s3 & s2 & s1 & s0 ;
  process(addr)
    begin
    case (addr) is
      when "0000" => rdata <= "101011010";
      when "0001" => rdata <= "101110010";
      when "0011" => rdata <= "010000100";
      when others => rdata <= "000000000";
    end case;
  end process;

  OEN<=rdata(0);
  WE1<=rdata(1);
  WE2<=rdata(2);
  GWE1<=rdata(3);
  OENN1<=rdata(4);
  GWE2<=rdata(5);
  OENN2<=rdata(6);
  GWE3<=rdata(7);
  OENN3<=rdata(8);
end Behavioral;
```

C 循环加法器 romc 修改后的 VHDL 程序

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

entity romc1 is
    Port ( S0 : in  STD_LOGIC;
           S1 : in  STD_LOGIC;
           S2 : in  STD_LOGIC;
           S3 : in  STD_LOGIC;
           OEN : out  STD_LOGIC;
           WE1 : out  STD_LOGIC;
           WE2 : out  STD_LOGIC;
           GWE1 : out  STD_LOGIC;
           OENN1 : out  STD_LOGIC;
           GWE2 : out  STD_LOGIC;
           OENN2 : out  STD_LOGIC;
           GWE3 : out  STD_LOGIC;
           OENN3 : out  STD_LOGIC);
end romc1;

architecture Behavioral of romc1 is
    signal addr  : std_logic_vector(3 downto 0);  --input
    signal rdata : std_logic_vector(8 downto 0);  --output
begin

addr <= s3 & s2 & s1 & s0 ;

    process(addr)
   begin
    case (addr) is
      when "0000" => rdata <= "101010010";
      when "0001" => rdata <= "101011000";
      when "0010" => rdata <= "101110000";
      when "0011" => rdata <= "010000100";
      when "0111" => rdata <= "101110000";
      when "1111" => rdata <= "101011010";
      when "1110" => rdata <= "010000100";
      when "1100" => rdata <= "101110000";
      when "1000" => rdata <= "101011010";
      when "1001" => rdata <= "010000100";
      when "1011" => rdata <= "101110000";
      when others => rdata <= "000000000";
    end case;
  end process;

  OEN<=rdata(0);
```

```
46      WE1<=rdata(1);
47      WE2<=rdata(2);
48      GWE1<=rdata(3);
49      OENN1<=rdata(4);
50      GWE2<=rdata(5);
51      OENN2<=rdata(6);
52      GWE3<=rdata(7);
53      OENN3<=rdata(8);
54  end Behavioral;
```

D 外部端口定义

```
###------------CLOCK-----------
NET "clk"   LOC = "L15";
#
###-----------Atlys Switch input-------------------
NET "S0"    LOC = A10;  # Atlys sw0
NET "S1"    LOC = D14;  # Atlys sw1
NET "S2"    LOC = C14;  # Atlys sw2
NET "S3"    LOC = P15;  # Atlys sw3
#NET "atlys_sw[4]"    LOC = P12;  # Atlys sw4
#NET "atlys_sw[5]"    LOC = R5;   # Atlys sw5
#NET "atlys_sw[6]"    LOC = T5;   # Atlys sw6
NET "XLXN_74"    LOC = E4;   # Atlys sw7
#
###------------EES261 switch input----------
#NET "swt[19]" LOC = "U11";    #SW20
#NET "swt[18]" LOC = "R10";    #SW19
#NET "swt[17]" LOC = "U10";   #SW18
#NET "swt[16]" LOC = "R8";   #SW17
#
NET "DATA_IN[7]" LOC = "M8";  #SW16
NET "DATA_IN[6]" LOC = "U8";   #SW15
NET "DATA_IN[5]" LOC = "U7";   #SW14
NET "DATA_IN[4]" LOC = "N7";   #SW13
#
NET "DATA_IN[3]" LOC = "T6";   #SW12
NET "DATA_IN[2]" LOC = "R7";   #SW11
NET "DATA_IN[1]" LOC = "N6";    #SW10
NET "DATA_IN[0]" LOC = "U5";    #SW9
#
NET "CTL[3]" LOC = "V5";    #SW8
NET "CTL[2]" LOC = "P7";    #SW7
NET "CTL[1]" LOC = "T7";    #SW6
NET "CTL[0]" LOC = "V6";    #SW5
#
##----------EES261  leds output------------
NET "DATA_OUT<0>" LOC = "U16";    #LED1
NET "DATA_OUT<1>" LOC = "U15";    #LED2
NET "DATA_OUT<2>" LOC = "U13";    #LED3
NET "DATA_OUT<3>" LOC = "M11";    #LED4
NET "DATA_OUT<4>" LOC = "R11";    #LED5
NET "DATA_OUT<5>" LOC = "T12";    #LED6
NET "DATA_OUT<6>" LOC = "N10";    #LED7
NET "DATA_OUT<7>" LOC = "M10";    #LED8
#
```