目录

实验平台说明

特征	说明
操作系统版本和系统类型	Windows 7 64 bit enterprise edition
EDA 软件名称和版本	Quartus II 13.1 64 bit
DE2-115 开发板编号	1904615S

自查清单

特征	图序或表序
原理图	图 5.1
Verilog 代码	见四，设计步骤
Flow Summary	图 5.2
RTL 图	图 5.3
状态机图	图 5.4
工艺图	图 5.5
Timing Analyzer	×
仿真图	图 5.6
SignalTapII 图	×
硬件运行图	图 5.789

一、需求分析

调查分析现有的计算器产品，提炼出待设计集成电路的各项指标。

本次大作业前我对计算器成品进行了调研。研究了其中硬件，软件等实现的平台及方法。以一个典型案例举例。卡西欧 M-1 计算器。

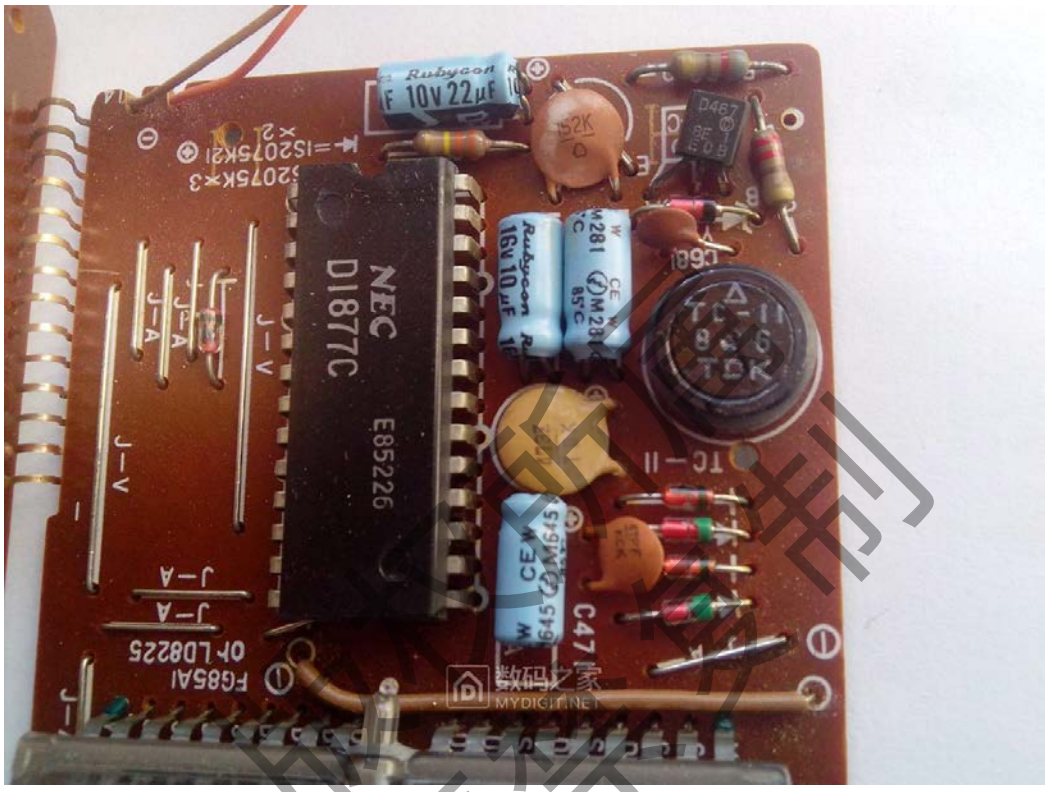


图 1.1

硬件：可以看出显示和计算都由一个芯片提供计算。包括显示驱动，按键输入，结果储存，连乘连加。

软件：以 8051 内核举例，由于连加连减的需求，算法大约两种：

1.直接计算储存型，经典串行，检测按键->输入判断（符号或数据）->相应操作。

2.堆栈型，生成两个栈以储存符号和数据，在按下等号时进行计算，这也是高级计算器目前常用算法。

提炼出共性：

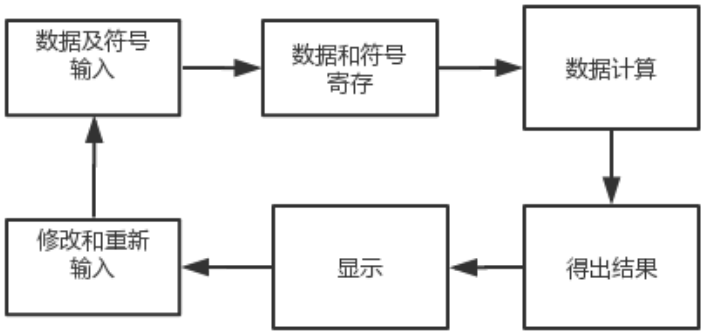


图 1.2

指标：

输入：-20000 到+20000

数据寄存：两个寄存器

输出：相应范围结果

显示：数码管等。

需求模块：加法器，除法器，乘法器，寄存器，状态机，切换器。

二、方案对比

1.根据指标设计方案：

	方案一	方案二	方案三
输入方式	串行输入	并行输入	串行输入
计算方式	模块计算法	模块计算法	模块计算法
寄存方式	寄存器寄存	寄存器寄存	寄存器寄存
显示方式	数码管十进制	数码管十六进制	LCD+十进制显示
交互方式	红外键盘	拨码开关+按键	红外键盘

表 1.1

2.方案分析

根据方案设计，大约有每个方式有两种不确定方案

横向对比：TUCHUAN`

1）串行输入：串行输入的数据为后一位数据*16+输入数据。

并行输入：并行输入的数据可以直接储存。

2）红外键盘：红外键盘以按键键值为基础，需要在串行输入的基础上进行使用。

拨码开关：在并行输入上可以直接使用。

3）LCD 显示：指令显示，每次计算结果需要使用状态机驱动，并进行二十进制转换。

数码管显示：可以不使用二十进制转换，需使用状态机转换显示方法。

考虑到难度问题，本次设计采用 planB。

三、原理分析

对自己的设计方法从原理上进行详细分析。

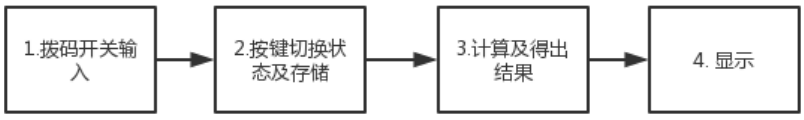


图 3.1

1. 拨码开关输入：由于方便输入以 SW17 和 SW16 为+-*/输入，00 为+，01 为-，10 为乘，11 为除。以 SW15 为正负号输入，SW14~SW0 为 32767 的绝对值输入。

原理：SW 为并行输入。

2. KEY 按键功能，KEY0 为复位，按下即寄存器清零，输出结果清 0。

   KEY1 为状态切换，按下即可以在 操作数 1，操作数 2，结果数 3 之间由数码管显示。

   KEY2 为按键储存，在操作数 12 状态时，按下即可储存操作数，结果会相应变化。

   原理： KEY 为串行输入，并在下降沿执行相应的状态切换。使用寄存器储存操作数，在某状态下就使能某操作数。

3. 计算模块：在此使用三个模块：减法器，乘法器，除法器。减法运算由操作数补码计算所得。

   原理：由操作数正负和符号组成新状态机。则有 4*4=16 种情况，对 16 种情况进行分析，从而对正负号以及结果进行组合。

4. 显示模块: 使用数码管 HEX0~7 进行显示数据的功能,

   原理利用数码管上不同 LED 的相对位置的亮灭进行显示数字 等功能

   其中,各种图像的表示方法罗列在器件手册中.

5. 溢出：LEDG3 为溢出检测，仅在除数为 0 时出现。

状态机表示	LEDG3	LEDG2	LEDG1	LEDG0
操作数 1 显示+储存	0	0	0	1
操作数 2 显示+储存	0	0	1	0
结果数显示	0	1	0	0
溢出	1	*	*	*

表 3.1

四、设计步骤

1. 输入及储存设计:

1) 输入状态机设计方法

Verilog 代码及解释。

```
always@(negedge KEY1,negedge KEY0)
begin
    if(!KEY0) state<=2'b00;
else
    if(!KEY1)
    begin
```

```
        case(state)
        2'b00:state<=2'b01;
        2'b01:state<=2'b10;
        2'b10:state<=2'b00;
        default:state<=2'b00;
        endcase
        end
    end

    always@(state)
    begin
    case(state)
    2'b00:begin    LEDG[2:0]=3'b001;end
    2'b01:begin    LEDG[2:0]=3'b010;end
    2'b10:begin    LEDG[2:0]=3'b100;end
    endcase
    end
    switch_321 m1(state,EN1,EN2,EN3);
    register_HEX m2(KEY2,KEY0,EN1,SW[15:0],OUT1[15:0]);
    register_HEX m3(KEY2,KEY0,EN2,SW[15:0],OUT2[15:0]);
```

以 KEY1 为按键输入切换状态，LEDG 为状态显示。

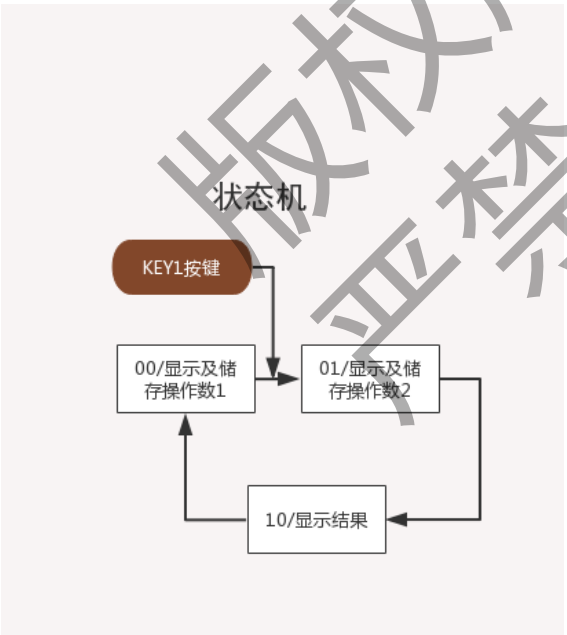


图 4.1

2)D-触发器及寄存器设计方法

```
/**
  * @brief    D flip-flop.
  * @param    NULL.
  * @note   NULL.
```

```
   * @retval The output is the D of D flip-flop.
   */


module D_trigger
                    (input Clk,R,D,EN,
                     output reg Q);
 always @(posedge Clk or negedge R)
 begin
        if(!R) Q<=1'b0;
        else if(EN)Q<=D;
              //    else Q<=1'b0;
 end
endmodule
/**
   * @brief    Enter 16 signals and latch.
   * @param    NULL.
   * @note     NULL.
   * @retval output is the latch signals.
   */

module register_HEX
                    (input Clk,R,EN,
                     input [15:0] D,
                     output [15:0]Q);
 D_trigger m1 (Clk,R,D[0],EN,Q[0]);
 D_trigger m2 (Clk,R,D[1],EN,Q[1]);
 D_trigger m3 (Clk,R,D[2],EN,Q[2]);
 D_trigger m4 (Clk,R,D[3],EN,Q[3]);
 D_trigger m5 (Clk,R,D[4],EN,Q[4]);
 D_trigger m6 (Clk,R,D[5],EN,Q[5]);
 D_trigger m7 (Clk,R,D[6],EN,Q[6]);
 D_trigger m8 (Clk,R,D[7],EN,Q[7]);
 D_trigger m9 (Clk,R,D[8],EN,Q[8]);
 D_trigger m10 (Clk,R,D[9],EN,Q[9]);
 D_trigger m11 (Clk,R,D[10],EN,Q[10]);
 D_trigger m12 (Clk,R,D[11],EN,Q[11]);
 D_trigger m13 (Clk,R,D[12],EN,Q[12]);
 D_trigger m14 (Clk,R,D[13],EN,Q[13]);
 D_trigger m15 (Clk,R,D[14],EN,Q[14]);
 D_trigger m16 (Clk,R,D[15],EN,Q[15]);
Endmodule
```

使用 16 个 D 触发器组成触发器。可以暂存 2 个字节的数据。

2. 计算模块:

1）计算模块采用方法非常直接，将五种情况列出：（分别是）

操作数+操作数

操作数+操作数补码

操作数补码+操作数

操作数*操作数

操作数/操作数

```
//*************将所有状况的运算运算出。
        assign OUT1[30:16]=15'b000000000000000;
        assign OUT2[30:15]=16'b0000000000000000;
        assign OUT3[30:15]=16'b0000000000000000;
        assign OUT4[30]=1'b0;
        assign OUT5[30:15]=4'h0000;
        myadd m1(IN1[14:0],IN2[14:0],OUT1[15],OUT1[14:0]);
        myadd    m2(IN1[14:0],sub_IN2,cout[0],OUT2[14:0]);
        myadd    m3(sub_IN1,IN2[14:0],cout[1],OUT3[14:0]);
        mymult m4(IN1[14:0],IN2[14:0],OUT4[29:0]);
        mydivi m5(IN2[14:0],IN1[14:0],OUT5[14:0],remain);
```

2）分情况讨论

状态机

State:

0000 正加正	1000 正*正
0001 正加负	1001 正*负
0011 负加负	1011 负*负
0010 负加正	1010 负*正
0100 正减正	1100 正除正
0101 正减负	1101 正除负
0111 负减负	1111 负除负
0110 负减正	1110 负除正

加法：

```
        4'b0000:begin OUT[31]<=1'b0;OUT[30:0]<=OUT1[30:0];end
        4'b0001:begin
                        if(IN1[14:0]>=IN2[14:0])
                        begin
                            OUT[31]<=1'b0;
                            OUT[30:0]<=OUT2[30:0];
                        end
                        else if(IN1[14:0]<IN2[14:0])
                        begin
```

```
                    OUT[31]<=1'b1;
                    OUT[30:0]<=OUT3[30:0];
                end
            end
        4'b0011:begin OUT[31]<=1'b1;OUT[30:0]<=OUT1[30:0];end
        4'b0010:begin
                if(IN1[14:0]>=IN2[14:0])
                begin
                    OUT[31]<=1'b1;
                    OUT[30:0]<=OUT2[30:0];
                end
                else if(IN1[14:0]<IN2[14:0])
                begin
                    OUT[31]<=1'b0;
                    OUT[30:0]<=OUT3[30:0];
                end
            end
```

减法：

```
        4'b0100:    begin //+ - +
                if(IN1[14:0]>=IN2[14:0])
                begin
                    OUT[31]<=1'b0;
                    OUT[30:0]<=OUT2[30:0];
                end
                else if(IN1[14:0]<IN2[14:0])
                begin
                    OUT[31]<=1'b1;
                    OUT[30:0]<=OUT3[30:0];
                end
            end
        4'b0101:    begin OUT[31]<=1'b0;OUT[30:0]<=OUT1[30:0];end//+ - -
        4'b0111:begin //- - -
                if(IN1[14:0]>=IN2[14:0])
                begin
                    OUT[31]<=1'b1;
                    OUT[30:0]<=OUT2[30:0];
                end
                else if(IN1[14:0]<IN2[14:0])
                begin
                    OUT[31]<=1'b0;
                    OUT[30:0]<=OUT3[30:0];
                end
            end
        4'b0110:begin OUT[31]<=1'b1;OUT[30:0]<=OUT1[30:0];end
```

乘法：

```
4'b1000:begin OUT[31]<=1'b0;OUT[30:0]<=OUT4[30:0];end
4'b1001:begin OUT[31]<=1'b1;OUT[30:0]<=OUT4[30:0];end
4'b1011:begin OUT[31]<=1'b0;OUT[30:0]<=OUT4[30:0];end
4'b1010:begin OUT[31]<=1'b1;OUT[30:0]<=OUT4[30:0];end
```

除法：

```
4'b1100:begin if(IN2==15'b000000000000000 ) LEGG_OVERFLOW=1'b1;
                                    else
OUT[31]<=1'b0;OUT[30:0]<=OUT5[30:0];LEGG_OVERFLOW=1'b0;end
4'b1101:begin if(IN2==15'b000000000000000 ) LEGG_OVERFLOW=1'b1;
                                    else
OUT[31]<=1'b1;OUT[30:0]<=OUT5[30:0];LEGG_OVERFLOW=1'b0;end
4'b1111:begin if(IN2==15'b000000000000000 ) LEGG_OVERFLOW=1'b1;
                                    else
OUT[31]<=1'b0;OUT[30:0]<=OUT5[30:0];LEGG_OVERFLOW=1'b0;end
4'b1110:begin if(IN2==15'b000000000000000 ) LEGG_OVERFLOW=1'b1;
                                    else
OUT[31]<=1'b1;OUT[30:0]<=OUT5[30:0];LEGG_OVERFLOW=1'b0;end
```

溢出检测：LEDG3 为溢出检测，仅在除数为 0 时出现。

3. 显示模块:

使用显示模块 HEX 和 switch，使用此模块进行对 2 中的数据进行显示。

对数据进行显示选择

```
module switch_323
                    (input[1:0] IN,
                     input[15:0]DATA1,DATA2,
                     input[31:0]DATA3,
                     output reg[31:0] OUT);
    always @(IN,DATA1,DATA2)
    begin
    if(IN==2'b00)
        begin OUT[15:0]<=DATA1;OUT[31:16]<=4'h0000;end
    else if(IN==2'b01)
        begin OUT[15:0]<=DATA2;OUT[31:16]<=4'h0000;end
    else if(IN==2'b10)
        begin OUT<=DATA3; end
    else
        begin OUT<=8'h00000000;end
    end
endmodule
```

HEX 显示

```
module SEG_HEX
	(
		iDIG,
		oHEX_D
	);
input	[3:0]	iDIG;
output	[6:0]	oHEX_D;
reg	[6:0]	oHEX_D;
always @(iDIG)
		begin
			case(iDIG)
			4'h0: oHEX_D <= 7'b1000000; //0
			4'h1: oHEX_D <= 7'b1111001; //1
			4'h2: oHEX_D <= 7'b0100100; //2
			4'h3: oHEX_D <= 7'b0110000; //3
			4'h4: oHEX_D <= 7'b0011001; //4
			4'h5: oHEX_D <= 7'b0010010; //5
			4'h6: oHEX_D <= 7'b0000010; //6
			4'h7: oHEX_D <= 7'b1111000; //7
			4'h8: oHEX_D <= 7'b0000000; //8
			4'h9: oHEX_D <= 7'b0011000; //9
			4'ha: oHEX_D <= 7'b0001000; //a
			4'hb: oHEX_D <= 7'b0000011; //b
			4'hc: oHEX_D <= 7'b1000110; //c
			4'hd: oHEX_D <= 7'b0100001; //d
			4'he: oHEX_D <= 7'b0000110; //e
			4'hf: oHEX_D <= 7'b0001110; //f
		 default: oHEX_D <= 7'b1000000; //0
			endcase
		end
endmodule
```

4. 顶层及辅助模块:

1）顶层模块为各个模块例化

```
calculate u1(
			.SW(SW),
			.KEY0(KEY[0]),
			.KEY1(KEY[1]),
			.KEY2(KEY[2]),
			.LEDG(LEDG),
```

```
                .DATA(hex_data)
);

//the key code is display on HEX0 ~ HEX3
//the custom code is display on HEX4 ~ HEX7
SEG_HEX u2( //display the HEX on HEX0
                .iDIG(hex_data[3:0]),
                .oHEX_D(HEX0)
              );
SEG_HEX u3( //display the HEX on HEX1
                .iDIG(hex_data[7:4]),
                .oHEX_D(HEX1)
                );
SEG_HEX u4(//display the HEX on HEX2
                .iDIG(hex_data[11:8]),
                .oHEX_D(HEX2)
                );
SEG_HEX u5(//display the HEX on HEX3
                .iDIG(hex_data[15:12]),
                .oHEX_D(HEX3)
                );
SEG_HEX u6(//display the HEX on HEX4
                .iDIG(hex_data[19:16]),
                .oHEX_D(HEX4)
                );
SEG_HEX u7(//display the HEX on HEX5
                .iDIG(hex_data[23:20]) ,
                .oHEX_D(HEX5)
                );
SEG_HEX u8(//display the HEX on HEX6
```

```
        .iDIG(hex_data[27:24]) ,
        .oHEX_D(HEX6)
        );
SEG_HEX u9(//display the HEX on HEX7
        .iDIG(hex_data[31:28]) ,
        .oHEX_D(HEX7)
        );
```

2）数个 switch 函数用于为状态机做出选择。

```
/*****************************/
module switch_hex
                (
                  input EN,
                  input [15:0] IN1,IN2,
                  output reg [15:0]Q
                  );
    always @(EN)
    begin
    if(EN)
        begin Q=IN1;end
    else
        begin Q=IN2;end
    end
endmodule
/*****************************/
module switch
                (input EN,
                  input IN,
                  output reg Q1,Q2);
```

```
    always @(EN)
    begin
    if(EN)
        begin Q1=IN;Q2=!IN;end
    else
        begin Q1=1'b0;Q2=1'b0; end
    end
endmodule
/*****************************/
module switch_321
                    (input[1:0] IN,
                      output reg Q1,Q2,Q3);
    always @(IN)
    begin
    if(IN==2'b00)
        begin Q1=1'b1;Q2=1'b0;Q3=1'b0;end
    else if(IN==2'b01)
        begin Q1=1'b0;Q2=1'b1;Q3=1'b0;end
    else if(IN==2'b10)
        begin Q1=1'b0;Q2=1'b0;Q3=1'b1;end
    else
        begin Q1=1'b0;Q2=1'b0;Q3=1'b0;end
    end
endmodule
/*****************************/
module switch_323
                    (input[1:0] IN,
                     input[15:0]DATA1,DATA2,
                     input[31:0]DATA3,
                     output reg[31:0] OUT);
```

```
always @(IN,DATA1,DATA2)
begin
if(IN==2'b00)
    begin OUT[15:0]<=DATA1;OUT[31:16]<=4'h0000;end
else if(IN==2'b01)
    begin OUT[15:0]<=DATA2;OUT[31:16]<=4'h0000;end
else if(IN==2'b10)
    begin OUT<=DATA3; end
else
    begin OUT<=8'h00000000;end
end
endmodule
```

五、测试方法

为验证集成电路的功能和性能，详细描述自己的测试方法。

1. 原理图：

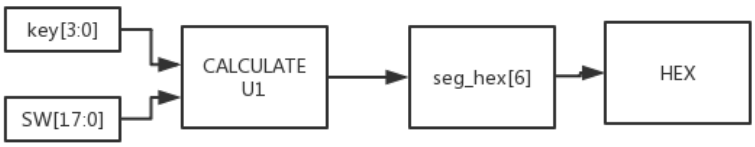


图 5.1

2. Verilog 代码如第五部分

3. Flow Summary

Flow Summary	
Flow Status	In progress - Thu Dec 19 19:04:12 2019
Quartus II 64-Bit Version	13.1.0 Build 162 10/23/2013 SJ Full Version
Revision Name	JSQ_FZN
Top-level Entity Name	JSQ_FZN_top
Family	Cyclone IV E
Device	EP4CE115F29C7
Timing Models	Final
Total logic elements	531
Total combinational functions	499
Dedicated logic registers	34
Total registers	34
Total pins	83
Total virtual pins	0
Total memory bits	0
Embedded Multiplier 9-bit elements	2
Total PLLs	0

图 5.2

4.RTL 图

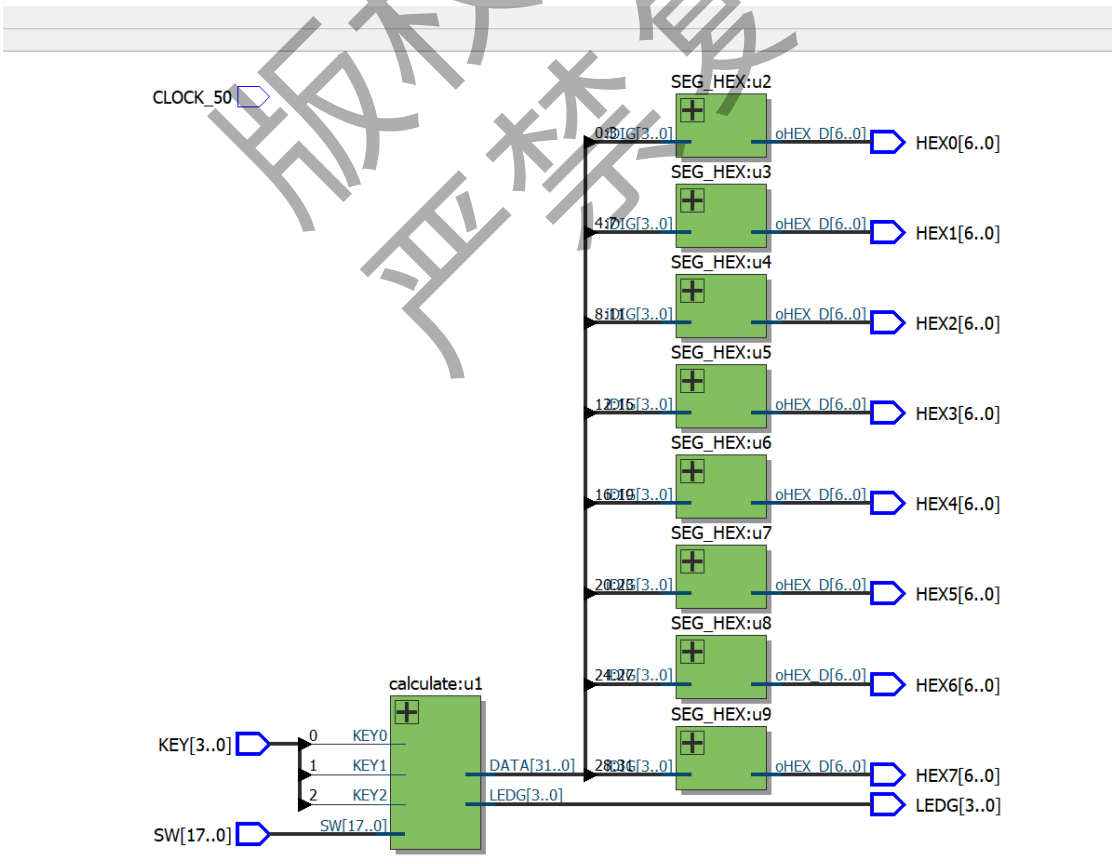


图 5.3

5. 状态机图

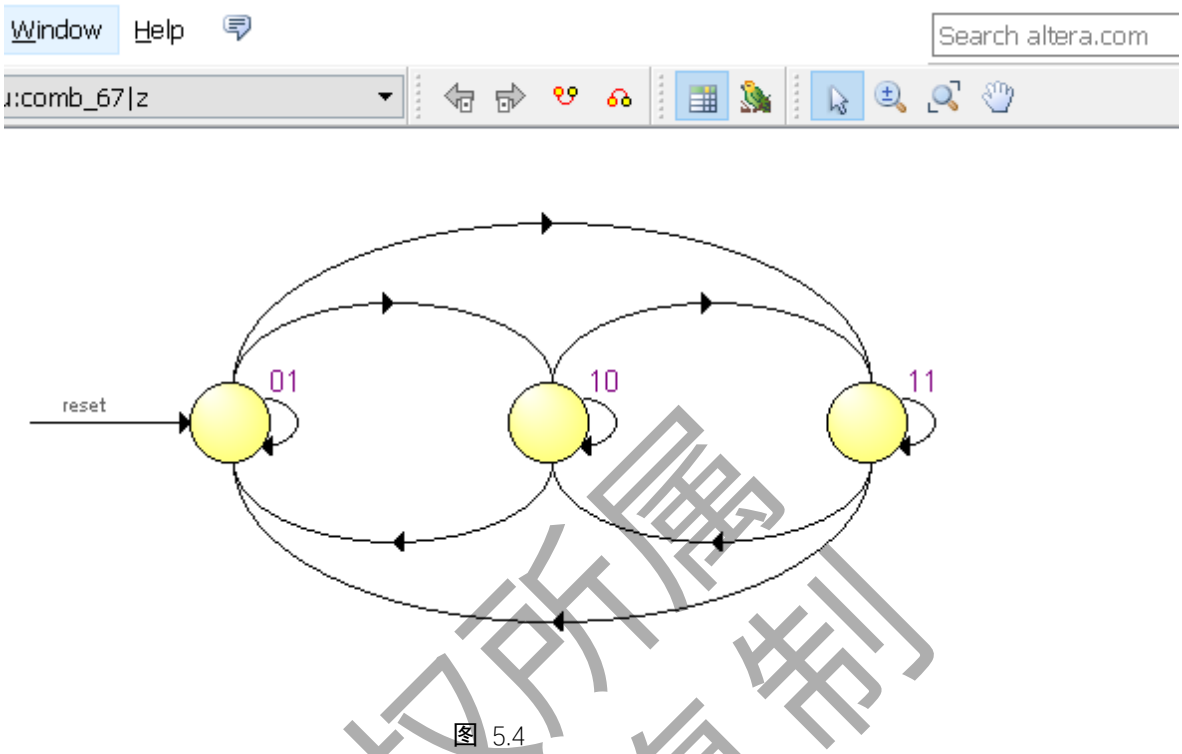


图 5.4

6. 工艺图

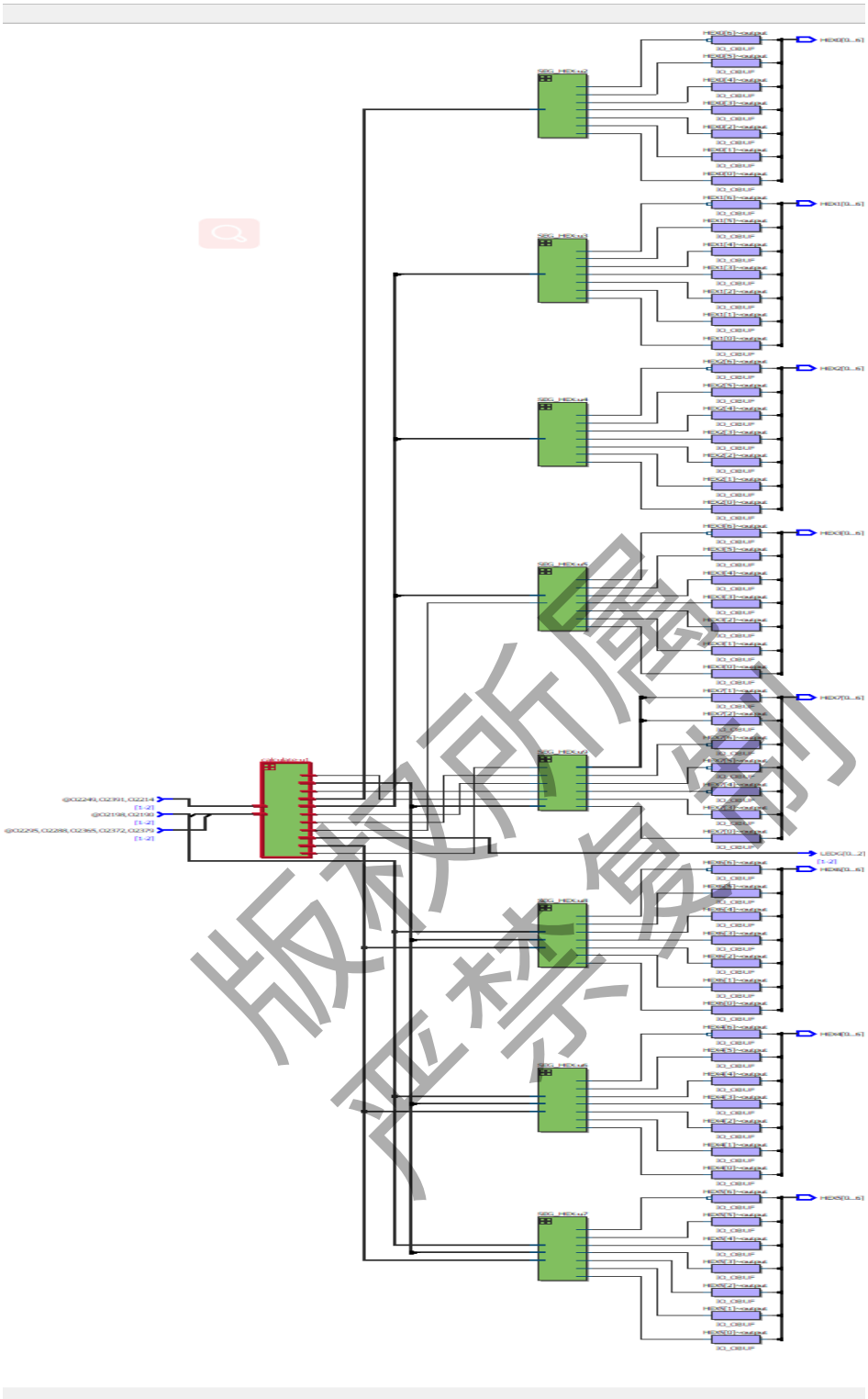


图 5.5

7. 仿真图

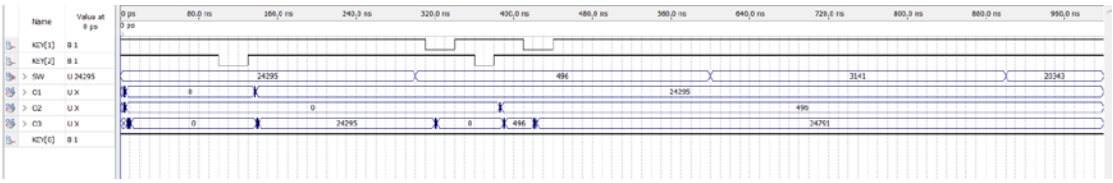


图 5.6

8. 硬件运行图

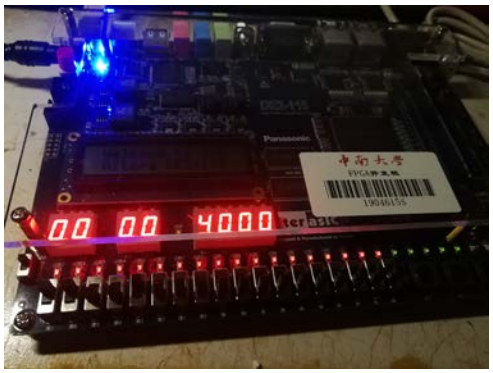


图 5.7

图 5.8

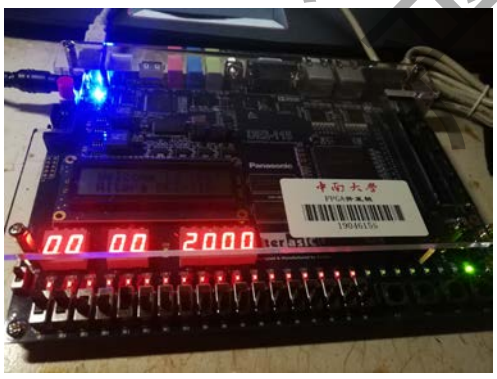


图 5.9

六、测试平台

用图形和文字详细描述测试中使用的资源和它们之间的连接关系。

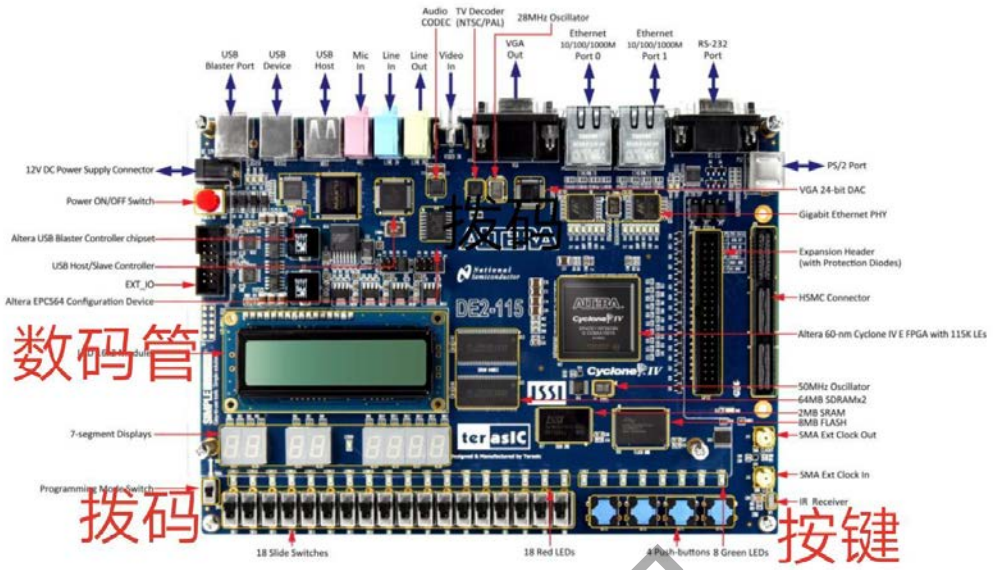


图 5.9

功能：

数码管：显示操作数及结果

拨码：输入运算符及操作数

按键：储存，显示，复位。

操作流程:

1.使用拨码 SW16~17 输入操作符，SW15 输入正负号，SW14~0 输入二进制操作数 1，则此数范围为-32767~+32767。

2.按下 KEY2 以储存数据，此时输入数据会以十六进制显示在数码管上。

3. 按下 SW14~0 输入二进制操作数，则此数范围为-32767~+32767。

4.使用拨码 SW14~0 输入二进制操作数 2，范围亦是+32767~+32767。

5.再次按下 KEY1 将状态切换到显示结果。

6.再次将状态调回任意一或二可以对操作数进行修改。

七、测试结果及讨论

用图表整理自己的测试结果及讨论(4E21=20001)范围足够。

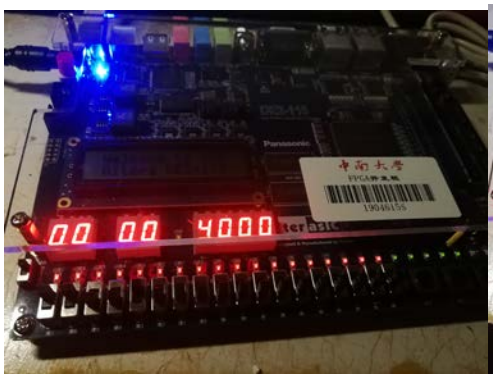
特别提醒：这里的操作数 12 为排除出正负号后的，由于正负号参与显示，所以若数 1 为负 4E21,则显示为 CE21 绝非显示错误，请使用者注意。

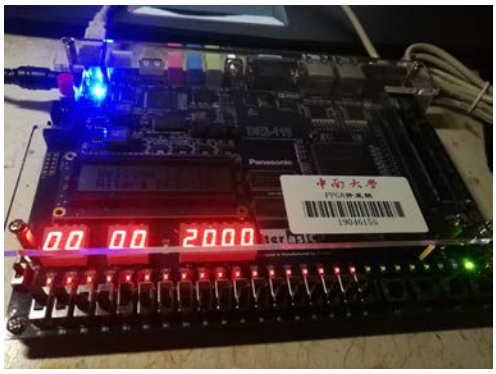
符号	操作数 1	操作数 2	结果	溢出位
加	正 0000 4E21	正'0000 0002	正 0000 0000 0000 4E23	0
加	正 0000 4E21	负'0000 0002	正 0000 0000 0000 4E1F	0
加	负 0000 4E21	正'0000 0002	负 0000 0000 0000 4E1F	0
加	负 0000 4E21	负'0000 0002	负 0000 0000 0000 4E23	0
加	正 0000 4E21	零	正 0000 0000 0000 4E21	0
加	零	正 0000 0002	正 0000 0000 0000 0002	0
减	正 0000 4E21	正'0000 0002	正 0000 0000 0000 4E1F	0
减	正 0000 4E21	负'0000 0002	正 0000 0000 0000 4E23	0
减	负 0000 4E21	正'0000 0002	负 0000 0000 0000 4E23	0
减	负 0000 4E21	负'0000 0002	负 0000 0000 0000 4E1F	0
减	正 0000 4E21	零	正 0000 0000 0000 4E21	0
减	零	正 0000 0200	负 0000 0000 0000 0002	0
乘	正 0000 4E21	正'0000 0200	正 0000 0000 009C 4200	0
乘	正 0000 4E21	负'0000 0200	负 0000 0000 009C 4200	0
乘	负 0000 4E21	正'0000 0200	负 0000 0000 009C 4200	0
乘	负 0000 4E21	负'0000 0200	正 0000 0000 009C 4200	0
乘	正 0000 4E21	零	零	0
乘	零	正 0000 0200	零	0
除	正 0000 4E21	正'0000 0200	正 0000 0000 0000 0027	0
除	正 0000 4E21	负'0000 0200	负 0000 0000 0000 0027	0
除	负 0000 4E21	正'0000 0200	负 0000 0000 0000 0027	0
除	负 0000 4E21	负'0000 0200	正 0000 0000 0000 0027	0
除	正 0000 4E21	零	溢出	1
除	零	正 0000 0200	0000 0000 0000 0000	0

表 7.1

输入输出范围为较大-32767~+32767,所以无溢出标志.

下图为两正数除法





八、特色分析

■八位数码管显示

□清晰可靠。

■范围-32767~+32767

□16 进制表示，程序员利器。

■支持有符号的加减乘除

□功能强大。

■随时查看操作数

□可以随时查看。

■随时修改操作数

□不需改的数据不必重新输入。

■并行计算，并行输出

□处理速度快。

■操作简单