第 8 次实验报告

实验 10 单周期处理器综合应用设计实验

22336313 郑鸿鑫

一． 实验准备

本次实验为处理器综合应用设计试验， 在上次搭建并仿真成功的处理基础上， 将处理器运行到 FPGA 板子上进行验证。并在 Mars 上设计汇编应用程序， 利用 Mars 仿真器产生二进制代码放在自己所设计的 CPU 上面。行， 查看功能结果。 本实验将会贯通之前所有的实验环节， 打通之前所有的实验知识点。

二． 实验目的

1. 掌握单周期 CPU 的实现方法，代码实现方法
2. 掌握测试单周期 CPU 的方法
3. 输入输出原理

三． 实验设备

PC 机一台，Basys3 开发板，Xlinix Vivado 开发套件。

四． 实验内容

本实验需要在上次仿真成功的基础上，将处理器运行到 FPGA 板子上进行验证。所以需要:

1. 可以通过板子查看寄存器的状态。
2. 处理器可以通过板子接收外部输入。
3. 在 Mars 上编写汇编代码, ， 产生二进制 coe 文件，使得处理

器实现简单的计算器功能（加，减，乘，除）。

4. 由于 Mars 编写的汇编代码中需要用到 sllv 和 slrv 这两条指令，所以需要扩展实验 9 的 ALU，增加左移，右移的功能。

五． 实验步骤

改造 CPU，使得 CPU 具备输入输出能力和在线调试能力。

1. 从实验 9 中，导入单周期 CPU;

2. 寄存器堆调试接口改写：

```
1  `timescale 1ns / 1ps
2  module regfile(
3      input wire clk,
4      input wire we3,
5      input wire[4:0] ra1,ra2,wa3,
6      input wire[31:0] wd3,
7      output wire[31:0] rd1,rd2,
8      input wire [4:0] ra_debug,
9      output wire [31:0] rd_debug
10     //add one more port here
11     );
12     reg [31:0] rf[31:0];
13     always @(posedge clk) begin
14         if(we3) begin
15             rf[wa3] <= wd3;
16         end
17     end
18     assign rd1 = (ra1 != 0) ? rf[ra1] : 0;
19     assign rd2 = (ra2 != 0) ? rf[ra2] : 0;
20     //add one more port here for debug at run-time
21     assign rd_debug = (ra_debug != 0) ? rf[ra_debug] : 0;
22 endmodule
```

为了能够在上板后查看寄存器状态，需要对原有的寄存器调试接口进行改写，在 regfile.v 文件中增加 debug 接口（代码第 8-9 行），其中 ra_debug[4:0]（代码第 8 行）表示读口地址，并将 ra_debug 绑定到板子上的 4 个按钮上，通过组合不同的按钮实现对 16 个寄存器的索引寄存器的结果通过 rd_debug[31:0]（代码第 9 行）输出，并连接到七段数码管

通过七段数码管显示，用于检查每个寄存器的值是否符合预期。（下图代码第 19 行）

3. 接受外部输入：

```
1  `timescale 1ns / 1ps
2  module top(
3      input wire clk,rst,
4      input wire [3:0] ra_debug_button,
5      input [15:0] switch_value,
6      output [3:0] select,
7      output [6:0] seg
8      );
9      wire [31:0] rd_debug;
10     wire [4:0] ra_debug;
11     assign ra_debug = {1'b0,ra_debug_button};
12     wire[31:0] pc,instr,readdata,readdata_mem,outdata,dataadr,writedata;
13     clock_div CC(clk,CLK);//时钟分频器
14     mips mips(CLK,rst,pc,instr,memwrite,dataadr,writedata,readdata,ra_debug,rd_debug);
15     Inst_Rom imem(CLK,pc[9:2],instr);
16     Data_Ram dmem(~CLK,memwrite,dataadr,writedata,readdata_mem);
17     assign readdata = (dataadr[14] == 0) ? readdata_mem : {16'b0,switch_value};
18     //用于检查当前访问的内存区域是存储器空间还是外设空间并选通对应的数据
19     display U2(clk,rst,rd_debug[15:0],seg,select);
20     //将寄存器中的值的低16位在7段数码管上显示
21 endmodule
```

为了能够实现外部与板子上单周期 CPU 运行程序的交互，需要引入外设(例如键盘)，这里我们使用拨码开关作为一个简单的外设，16 个拨码开关映射成一个 16 位的寄存器。通常情况下，操作系统将内存空间映射为几个不同的区域，作为系统的保留空间、程序的执行空间以及外设的保留空间(显卡的显存、键盘的缓冲区等)。这里，data memory 中 0x0-0x3FFF 为存储器空间，和之前实验 9 的使用方式完全一致，0x4000-0xFFFF_FFFF 映射为外设空间，0x4000 号地址为拨码开关外设的状态寄存器基址。通过访存指令 lw 获取拨码开关的状态，在 CPU 的实现中，引入一个 Mux，用于检查当前访问的内存区域是存储器空间还是外设空间并选通对应的数据。(上图代码第 17 行)。

4. 添加左移，右移指令：

```
`timescale 1ns / 1ps
module aludec(
    input wire[5:0] funct,
    input wire[1:0] aluop,
    output reg[2:0] alucontrol
    );
    // add your code here
    always @(*) begin
        case(aluop)
        2'b00: begin
            alucontrol = 3'b010;
        end
        2'b01: begin
            alucontrol = 3'b110;
        end
        2'b10: begin
            case(funct)
            6'b100000: alucontrol = 3'b010;//加
            6'b100010: alucontrol = 3'b110;//减
            6'b100100: alucontrol = 3'b000;//与
            6'b100101: alucontrol = 3'b001;//或
            6'b101010: alucontrol = 3'b111;//SLT
            6'b000100: alucontrol = 3'b100;//左移指令的aluctrl
            6'b000110: alucontrol = 3'b101;//右移指令的aluctrl
            //前两位是10，则说明是移位指令;
            endcase
        end
        default: alucontrol = 3'bXXX;
        endcase
    end
endmodule
```

在原有的指令集基础上，加入两个 R-Type 指令 sllv (变量逻辑左移), srlv (变量逻辑右移)， 为后续的上板程序提供支持。 引入新的指令只需要 aludec.v 产生的控制信号 alucontrol 进行拓展，并在 alu.v 中实现对应的逻辑左移、右移功能。 编写简单的程序， 验证对于这些指令集的支持，以及确保这些修改没有影响其他原有指令的正常功能。上图在 aludec.v 中扩展了左右移指令的译码，并保证使用的编码不与之前的功能冲突。下图则将 op（即上图产生 alucontrol）的对应操作改为 3 位，使 ALU 支持对

左右移指令的计算。

```
`timescale 1ns / 1ps
module alu(
    input wire[31:0] a,b,
    input wire[2:0] op,
    output reg[31:0] y,
    output reg overflow,
    output wire zero
    );
    wire[31:0] s,bout;
    assign bout = op[2] ? ~b : b;
    assign s = a + bout + op[2];
    always @(*) begin
        case (op)
            3'b000: y <= a & bout;
            3'b001: y <= a | bout;
            3'b010: y <= s;
            3'b110: y <= s;
            3'b111: y <= s[31];
            3'b100: y <= b << a;
            3'b101: y <= b >> a;
            default : y <= 32'b0;
        endcase
    end
    assign zero = (y == 32'b0);

    always @(*) begin
        case (op[2:1])
            2'b01:overflow <= a[31] & b[31] & ~s[31] |
                              ~a[31] & ~b[31] & s[31];
            2'b11:overflow <= ~a[31] & b[31] & s[31] |
                              a[31] & ~b[31] & ~s[31];
            default : overflow <= 1'b0;
        endcase
    end
endmodule
```

5. 验证寄存器 debug 功能和外设读取功能

   编写下列简单测试代码， 并在 Mars 上编写汇编代码， 产生二进制 coe 文件。拨动拨码开关， 然后并在板子通过按键选择例如$1 寄存器， 通过七段数码管查看拨码开关是否符合预期。

```
.text
loop: lw $1, 0x4000($0)
        j loop
```

6. 在 Mars 上编写汇编代码，实现简单的计算器功能：

   CPU 可以读取 16 位拨码开关的状态， 其中 15-14 位拨码开关为计算器的功能位，其中功能位的定义如下：

   00 01 10 11

   加法 减法 乘法 除法

   其中， 乘法和除法运算需要使用加减法实现(复用 mars 实验代码)剩下的 14 位（平均分为 7 位） 分别为两个输入数； 要求根据 15-14 位拨码状态调用不同的子函数， 然后将最后的计算结果放在$4 和$8 里面(方便按钮选择);并通过七段数码管显示(低 16 位)。

   代码如下：

   loop 段是对两个 7 位数据的操作和对两位操作码的译码，将其符号扩展并分别存储到 7 号和 9 号寄存器，详细说明见注释。Menu 段是对 6 号寄存器中的两位操作码进行识别，从而跳转到加，减，乘，除 4 种对应的功能。

```
.text
loop:
    lw $1,0x4000($0)# store switch_value to $1
    addi $3,$0,0x00003F80
    addi $5,$0,0x0000007F
    and $7,$3,$1# store [13:7] to $7
    and $9,$5,$1# store [6:0] to $9
    addi $2,$0,14
    addi $3,$0,7
    srlv $6,$1,$2# store [15:14] to $6  $6 from 0 to 3
    srlv $7,$7,$3# store [13:7] to $7
    addi $2,$0,6
    srlv $20,$9,$2
    srlv $21,$7,$2
    addi $22,$0,0xFFFFFF80
    beq $20,$0,Signext_num1
Signext_num2:#num2 in $9
    or $9,$9,$22
Signext_num1:#num1 in $7
    beq $21,$0,Menu
    or $7,$7,$22
Menu:
    beq $6,0,ADD
    beq $6,1,SUB
    beq $6,2,MUL
    beq $6,3,DIV
ADD:
    add $13,$7,$9
    j Store
SUB:
    sub $13,$7,$9
    j Store
```

乘法功能代码实现如上，详细说明已写在注释中，其中加减乘 3 种运算都将结果放在 13 号寄存器中，然后再通过 Store 段将 13 号寄存器中的值存入 4 号寄存器（为了方便在板子上查看，只需按住一个对应按钮就可以查看 4 号寄存器中的值）。

```
MUL:
      addi $10, $zero,1
      slt $11, $7, $zero  # Check if x is negative
      beq $11, $zero, x_positive # If x is positive, branch to x_positive
      sub $7, $zero, $7  # Otherwise, negate x
x_positive:
      slt $12, $9, $zero  # Check if y is negative
      beq $12, $zero, y_positive # If y is positive, branch to y_positive
      sub $9, $zero, $9  # Otherwise, negate y
y_positive:
      add $13, $zero, $zero  # Initialize result to 0

mul_loop:

   and $14, $9, $10  # Get the least significant bit of y
   beq $14, $zero, skip_add  # If the bit is 0, skip the add
   add $13, $13, $7  # Add x to result
skip_add:
   srlv $9, $9, $10  # Shift y right by 1
   sllv $7, $7, $10  # Shift x left by 1

   beq $9, $zero, mul_done  # If y is 0, go to mul_done
   j mul_loop  # Otherwise, repeat the loop

mul_done:
   beq $11, $zero, check_neg  # If x is positive, skip the negation
   sub $13, $zero, $13  # Otherwise, negate the result
check_neg:
   beq $12, $zero, Store  # If y is positive, skip the negation
   sub $13, $zero, $13  # Otherwise, negate the store_result
      j Store
Store:
      add $4,$13,$0
      j loop
```

除法功能代码实现如上，详细说明已写在注释中，需要对除数为 0 的情况做特殊处理，其中除法的结果通过专门的 end 代码段存储，将商从 12 号寄存器存入 4 号寄存器，将余数从 13 号寄存器存入 8 号寄存器（8 号和 4 号便于上板子查看值同理）。

```
DIV:
   beq $9,$0,loop #fenmu = 0
      slt $10, $7, $zero  # Check if dividend is negative
   slt $11, $9, $zero  # Check if divisor is negative

   # Take the absolute values of dividend and divisor
   beq $10, $zero, check_divisor_negative  # If dividend is positive, go to check_divisor_negative
   sub $7, $zero, $7  # Negate the dividend
check_divisor_negative:
   beq $11, $zero, division  # If divisor is positive, go to division
   sub $9, $zero, $9  # Negate the divisor

division:
   add $12, $zero, $zero  # Initialize quotient to 0
   add $13, $zero, $7  # Initialize a copy of the absolute dividend

division_loop:
   sub $13, $13, $9  # Subtract the absolute divisor from the copy of the absolute dividend
   slt $14, $13, $zero  # Check if the result of subtraction is negative
   beq $14, $zero, division_increment  # If the result is not negative, go to division_increment
   add $13, $13, $9  # Restore the absolute dividend
   j store_result  # Jump to store_result if the result is negative

division_increment:
   addi $12, $12, 1  # Increment the quotient
   beq $13, $zero, store_result  # Continue looping if the absolute dividend is not 0
   j division_loop

store_result:
   beq $10, $zero, check_divisor_negative_result  # If the original dividend is positive, go to check_divisor_negative_result
   sub $12, $zero, $12  # Negate the quotient
   sub $13, $zero, $13  # Negate the remainder
check_divisor_negative_result:
   beq $11, $zero, end  # If the original divisor is positive, go to end
   sub $12, $zero, $12  # Negate the quotient
end:
      add $4,$12,$0
      add $8,$13,$0
      j loop
```

特别说明：由于单周期 CPU 只支持 add，sub，addi，lw，sw，beq，j，slt，sllv，slrv10 条核心指令集，所以在编写汇编代码时，不能用到超出这些范围的指令。

六． 实验结果

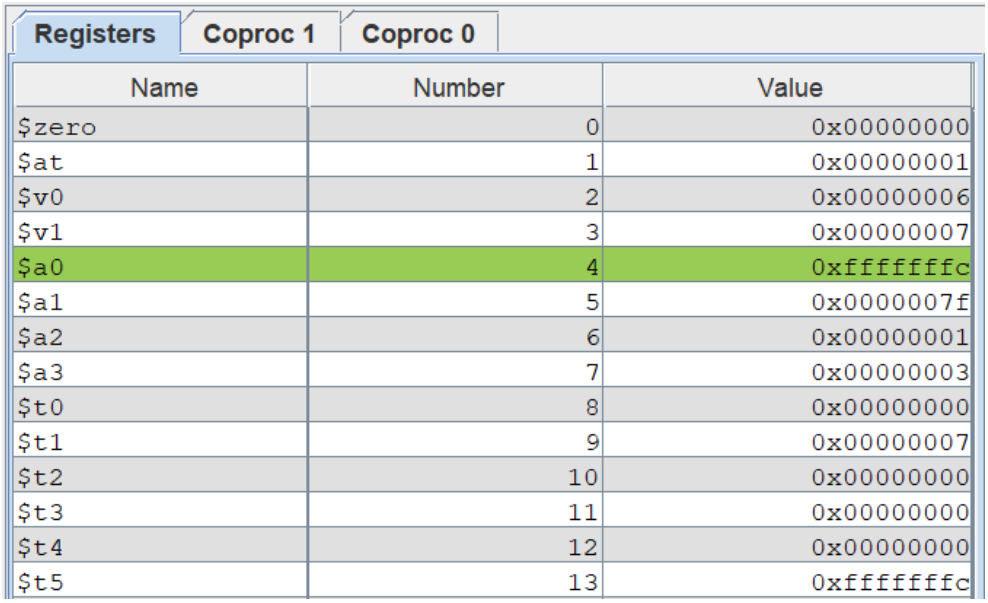
Mars 编译运行测试代码：

加法测试 以 7+3 = a 为例

$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000006
$v1	3	0x00000007
$a0	4	0x0000000a
$a1	5	0x0000007f
$a2	6	0x00000000
$a3	7	0x00000003
$t0	8	0x00000000
$t1	9	0x00000007
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x0000000a

结果为 a 且存储在 4 号寄存器，符合预期。

减法测试 以 3 - 7 = -4（16 进制的 FFFC）为例



Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000001
$v0	2	0x00000006
$v1	3	0x00000007
$a0	4	0xfffffffc
$a1	5	0x0000007f
$a2	6	0x00000001
$a3	7	0x00000003
$t0	8	0x00000000
$t1	9	0x00000007
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0xfffffffc

结果为 FFFC 且存储在 4 号寄存器，符合预期。

乘法测试 以 3 * 7 = 21（16 进制的 15）为例

Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000002
$v0	2	0x00000006
$v1	3	0x00000007
$a0	4	0x00000015
$a1	5	0x0000007f
$a2	6	0x00000002
$a3	7	0x00000018
$t0	8	0x00000000
$t1	9	0x00000000
$t2	10	0x00000001
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000015

结果为 15 且存储在 4 号寄存器，符合预期。

除法测试 以 7 / 3 = 2 余 1 为例

Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000003
$v0	2	0x00000006
$v1	3	0x00000007
$a0	4	0x00000002
$a1	5	0x0000007f
$a2	6	0x00000003
$a3	7	0x00000007
$t0	8	0x00000001
$t1	9	0x00000003
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000002
$t5	13	0x00000001

商存储在 4 号寄存器，余数存储在 8 号寄存器，符合预期

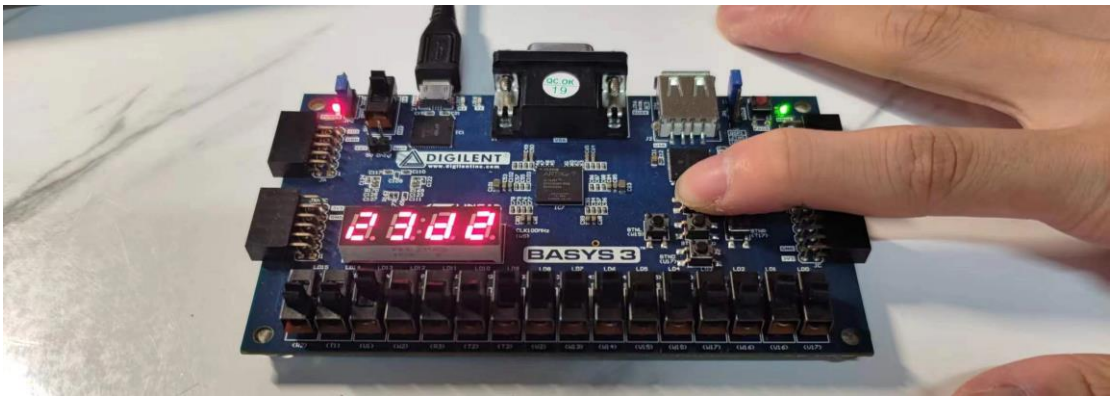
上板验证接收外设输入功能

利用简单的汇编代码如下：

```
.text
loop: lw $1, 0x4000($0)
        j loop
```

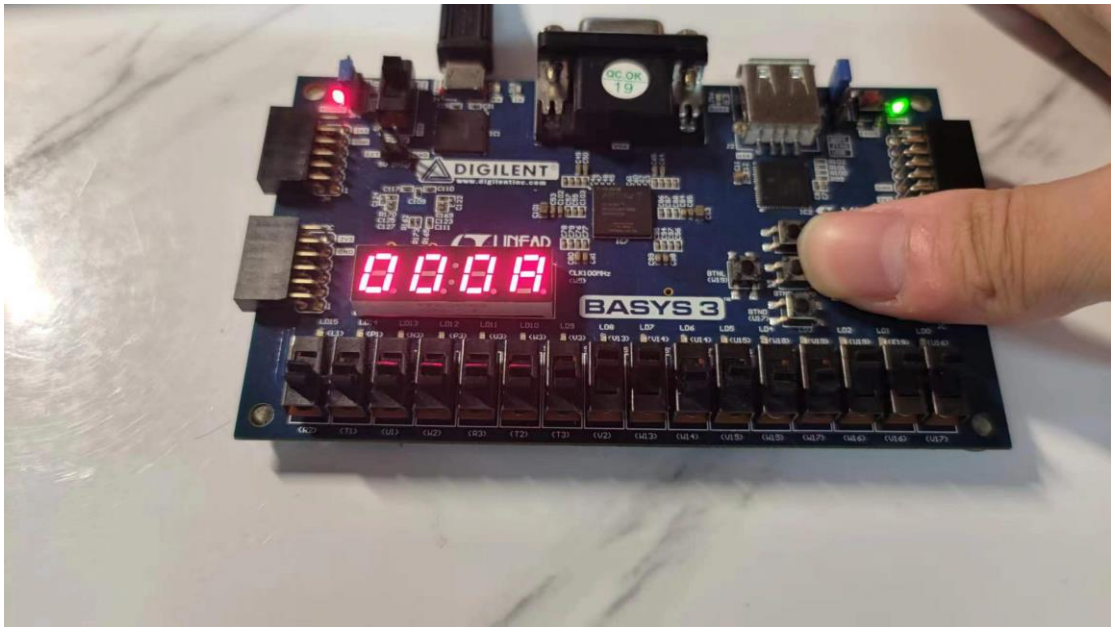
实现功能为将 16 位拨码开关的输入读取并存储到 1 号寄存器中。

在拨码开关输入 0010001111010010（即 16 进制的 23d2），按下按钮查看 1 号寄存器的值，得到 23d2，符合预期

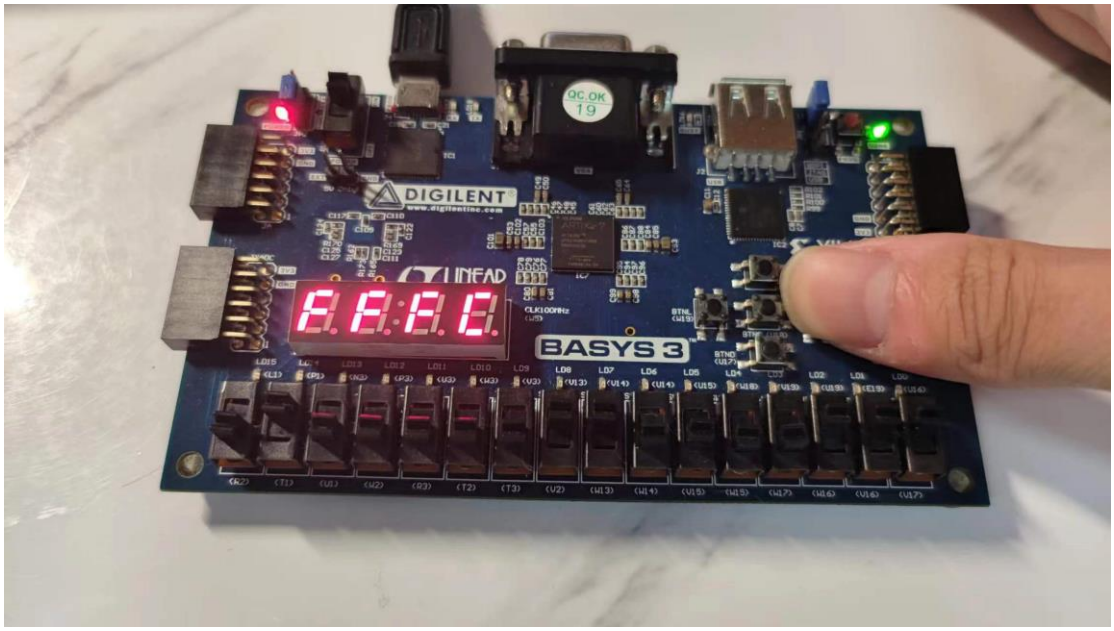


若功能无误，则将代码实际上板验证，结果如下所示：

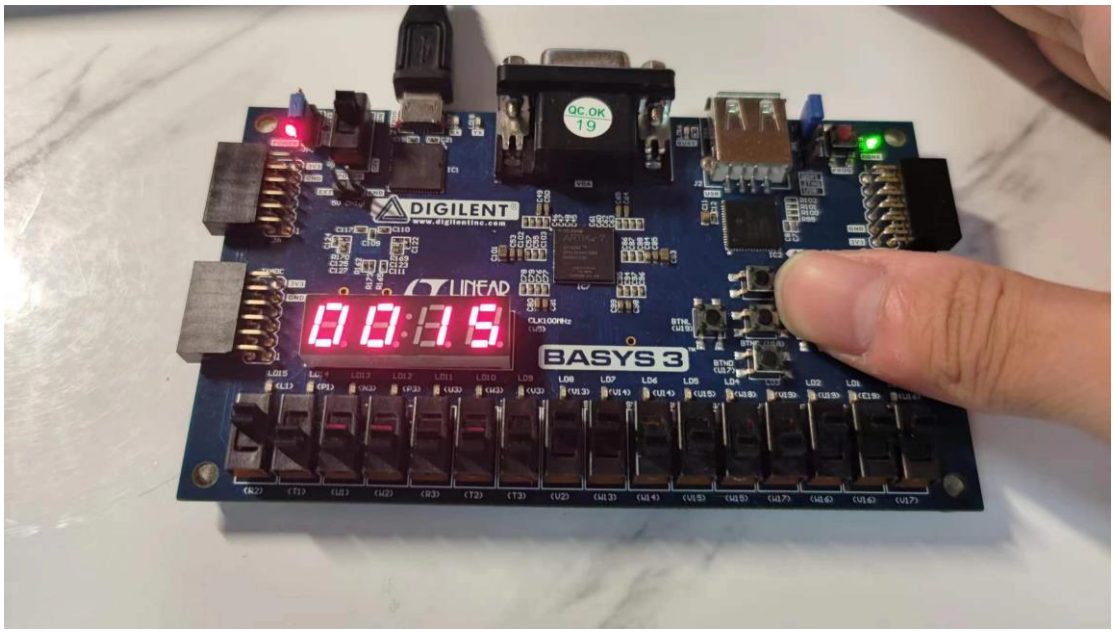
加法演示如下：拨码开关高两位为 00，后 14 位分别输入两个 7 位的二进制数，下图以 7+3 = a 为例



减法演示如下：拨码开关高两位为 01，后 14 位分别输入两个 7 位的二进制数，下图以 3 - 7 = -4（16 进制为 FFFC）为例（由于对其进行符号扩展，所以可以出现负数）



乘法演示如下：拨码开关高两位为 10，后 14 位分别输入两个 7 位的二进制数，下图以 7 * 3 = 21（16 进制为 0015）为例



除法演示如下：拨码开关高两位为 11，后 14 位分别输入两个 7 位

的二进制数，下图以 7 / 3 = 2 余 1 为例（第一张图查看的是 4 号寄存器即商为 2，第二张图查看的是 8 号寄存器即余数为 1）