

同济大学计算机系

计算机系统实验报告



龙芯 MIPS 指令集 CPU 核 LS132R 改造与性能验证

学 号 2154312

姓 名 郑博远

专 业 计算机科学与技术

授课老师 秦国锋

一、实验环境与实验内容

1.1 实验环境

操作系统	Windows 11
开发环境	Vivado 2016.2
硬件配置	XILINX NEXYS 4 DDR

1.2 实验内容

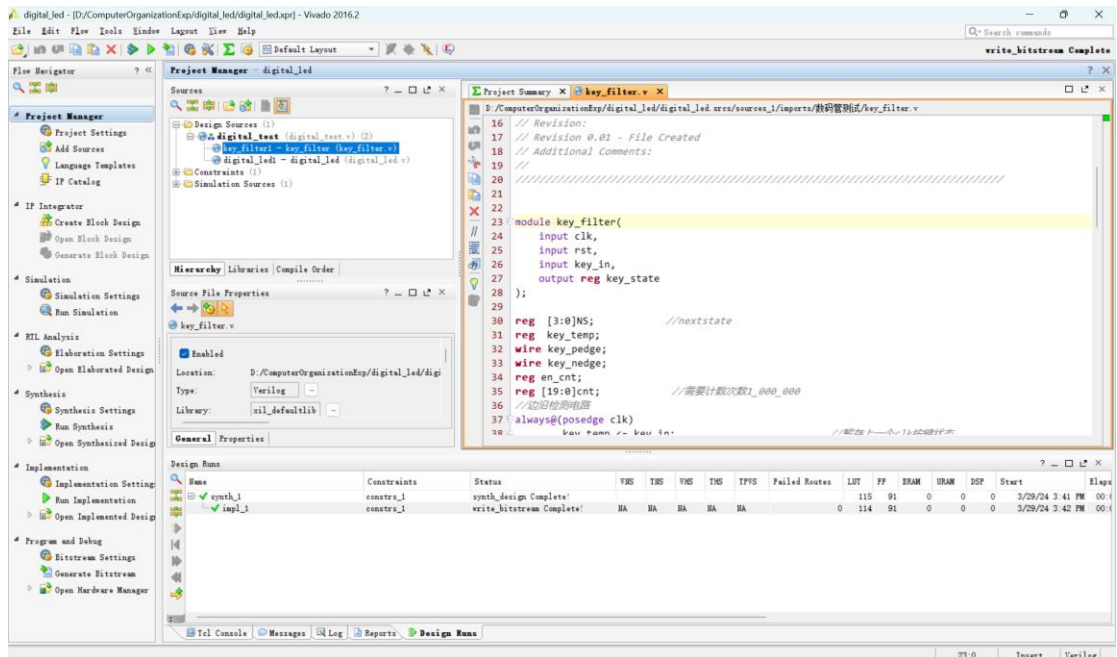
本次实验通过数码管实验、Flash 读取实验以及 AXI 通信实验入手，在龙芯 CPU LS132R 上运行自己编写的测试程序。我使用 C 语言和 MIPS 指令分别编写了性能验证程序。C 语言程序利用 gcc 编译器编译生成目标程序，而 MIPS 指令汇编程序则通过 Mars 编译生成目标程序。此外，本次实验中对两种方式下 CPU 的定点运算性能进行了测试、比较和分析。

在将龙芯 CPU LS132R 移植到 Nexys 4 开发板上的过程中，首先需要将 C 语言代码程序烧录进 flash 中，然后烧录 FPGA 程序。当按下开发板上的复位键时，LS132R 核会首先进入固定地址取指令，通常是从 flash 上取指令。启动过程中，flash 程序会将数据和其他必要内容搬运到适当的 RAM 中，完成整个初始化过程，然后进入 main 函数运行 C 语言程序。

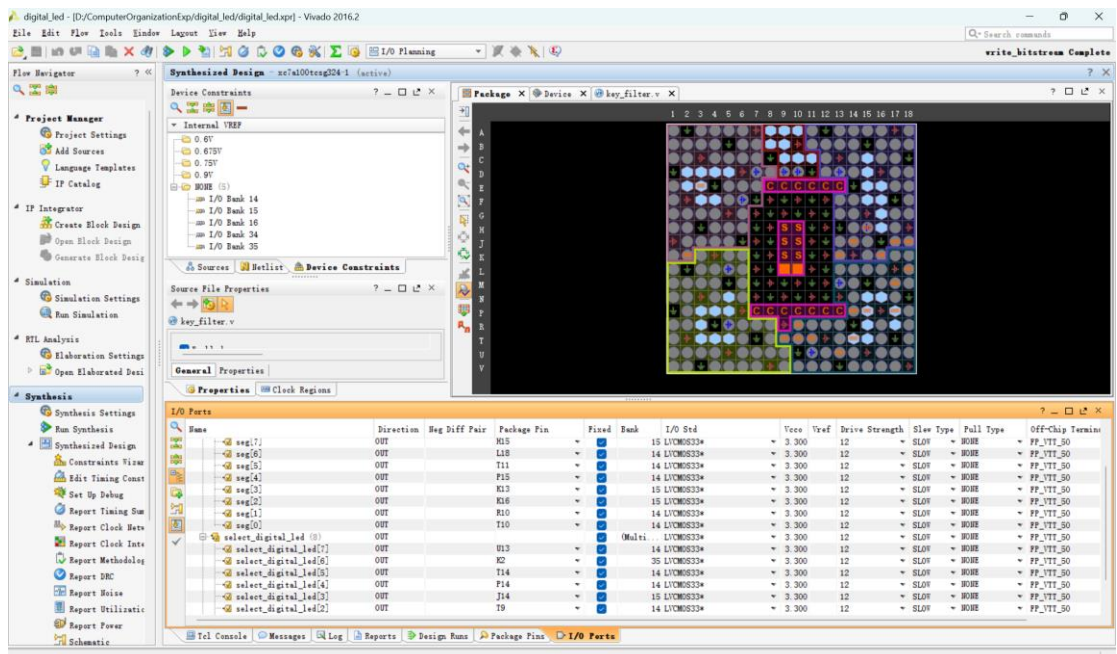
二、实验过程与方法

2.1 数码管实验

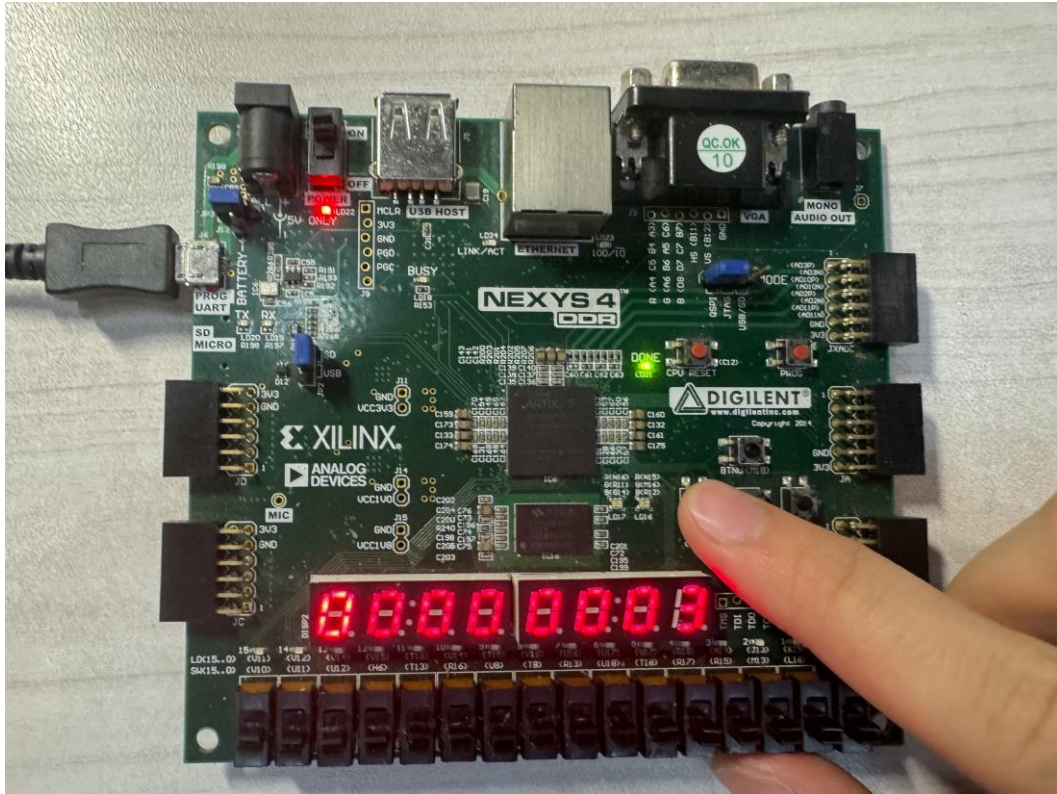
1. 创建工程，导入相关代码：



2. 添加管脚约束：

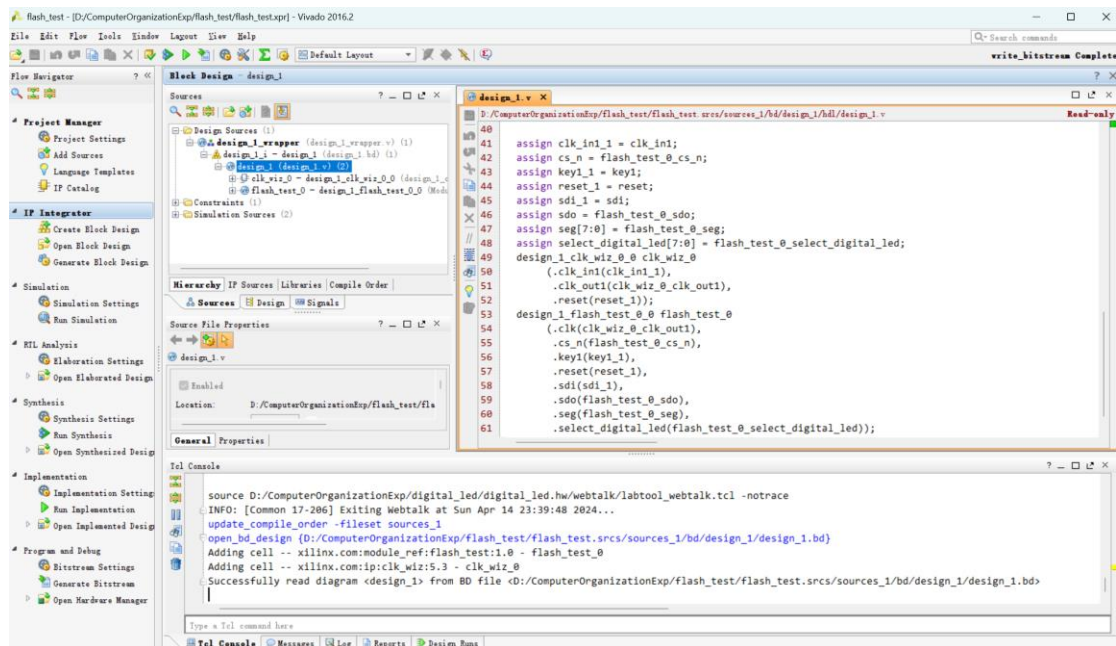


3. 下载线连接开发板，打开开发板电源，进行下载。按动 P17 按钮后，数码管数字值增加 1。如图所示：

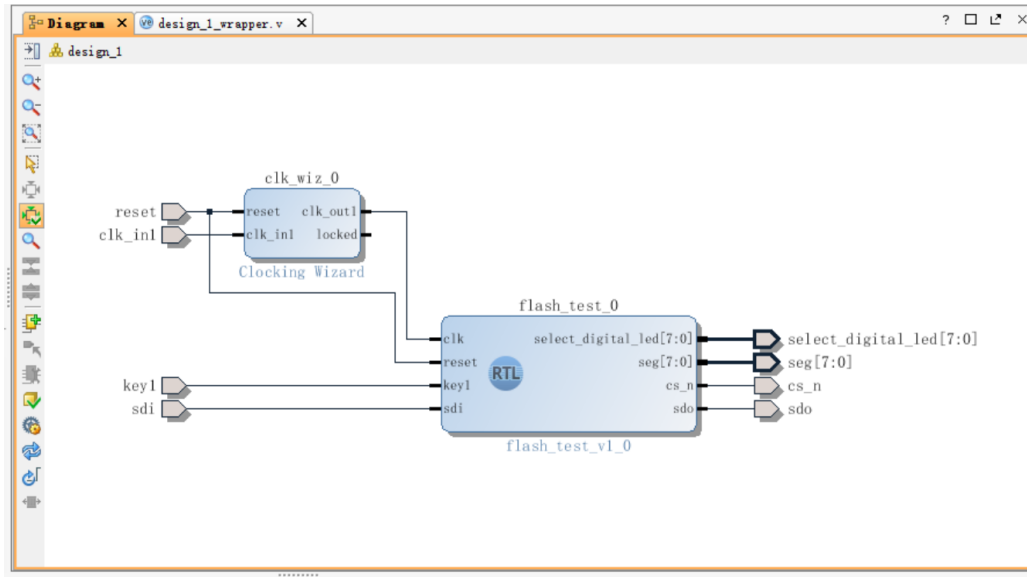


2.2 Flash 读取实验

1. 创建工程，导入相关代码：



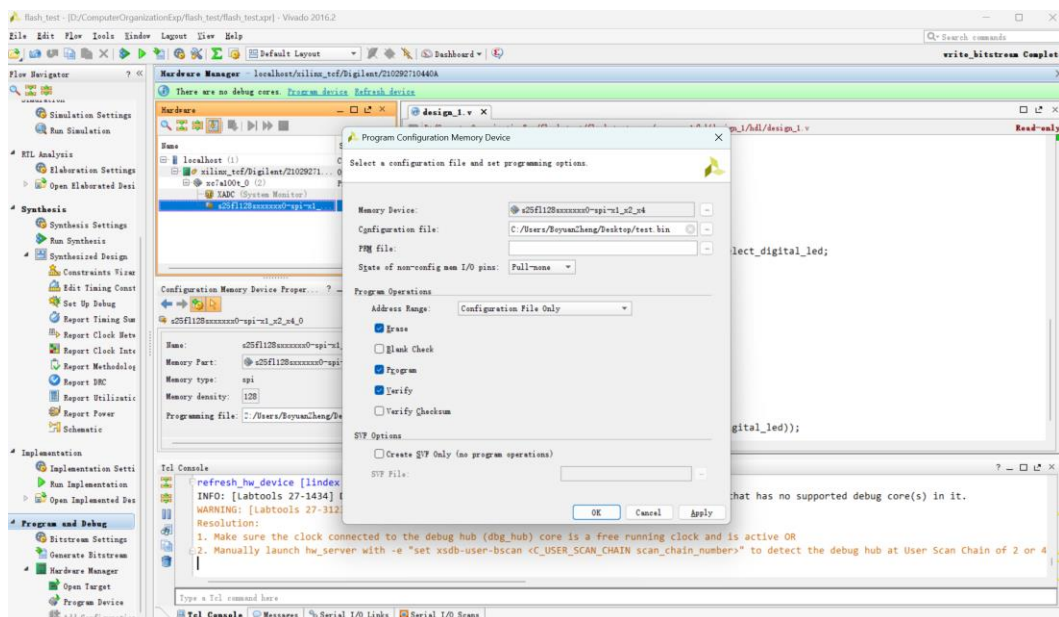
2. 创建 Block Design，并连线如下：



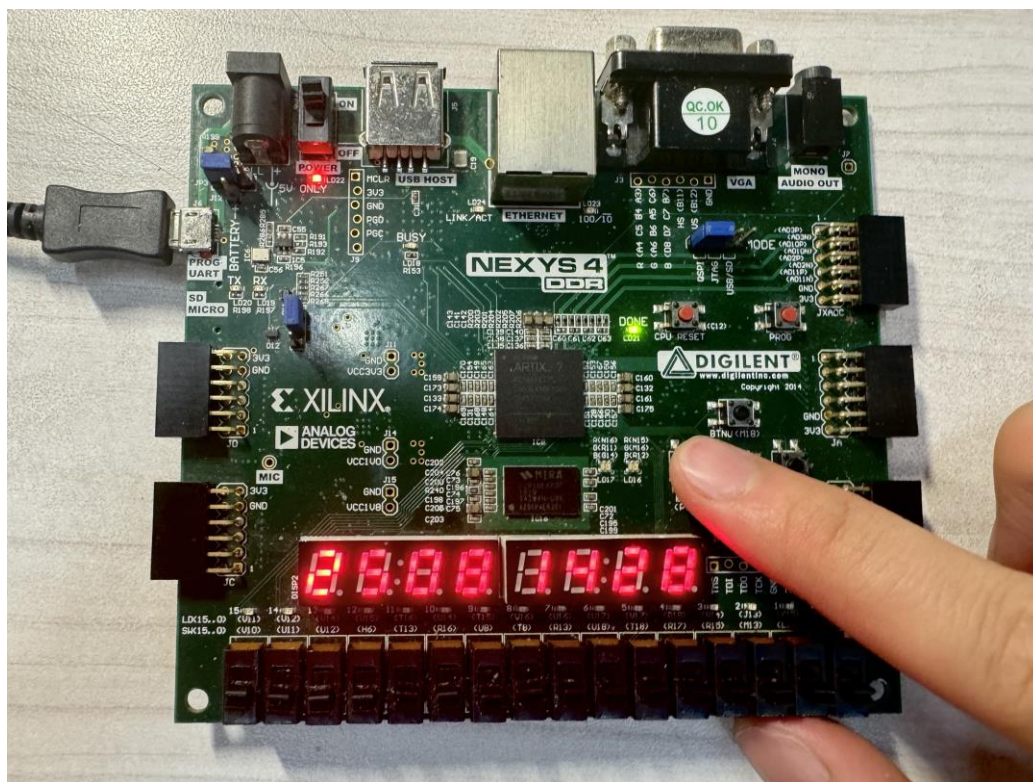
3. 添加管脚约束：

Name	Direction	Hog Diff Pair	Package Pin	Fixed	Bank	I/O Std	Vcco	Vref	Drive Strength	Slew Type	Pull Type	Off-Chip Termination
All ports (22)												
reset_61443 (1)	IN					14 LVCMOS33*	3.300				PULL-UP	PP_VTT_50
seg (5)	OUT					15 LVCMOS33*	3.300	12	SLOW	NONE		PP_VTT_50
seg[7]	OUT		R05			14 LVCMOS33*	3.300	12	SLOW	NONE		PP_VTT_50
seg[6]	OUT		L18			14 LVCMOS33*	3.300	12	SLOW	NONE		PP_VTT_50
seg[5]	OUT		T11			14 LVCMOS33*	3.300	12	SLOW	NONE		PP_VTT_50
seg[4]	OUT		F15			14 LVCMOS33*	3.300	12	SLOW	NONE		PP_VTT_50
seg[3]	OUT		R13			15 LVCMOS33*	3.300	12	SLOW	NONE		PP_VTT_50
seg[2]	OUT		R26			15 LVCMOS33*	3.300	12	SLOW	NONE		PP_VTT_50
seg[1]	OUT		R10			14 LVCMOS33*	3.300	12	SLOW	NONE		PP_VTT_50
seg[0]	OUT		T10			14 LVCMOS33*	3.300	12	SLOW	NONE		PP_VTT_50
select_digital_led (8)	OUT					14 LVCMOS33*	3.300	12	SLOW	NONE		PP_VTT_50
select_digital_led[7]	OUT		U13			14 LVCMOS33*	3.300	12	SLOW	NONE		PP_VTT_50
select_digital_led[6]	OUT		R2			35 LVCMOS33*	3.300	12	SLOW	NONE		PP_VTT_50
select_digital_led[5]	OUT		T14			14 LVCMOS33*	3.300	12	SLOW	NONE		PP_VTT_50
select_digital_led[4]	OUT		F14			14 LVCMOS33*	3.300	12	SLOW	NONE		PP_VTT_50
select_digital_led[3]	OUT		J14			15 LVCMOS33*	3.300	12	SLOW	NONE		PP_VTT_50
select_digital_led[2]	OUT		T9			14 LVCMOS33*	3.300	12	SLOW	NONE		PP_VTT_50
select_digital_led[1]	OUT		J18			14 LVCMOS33*	3.300	12	SLOW	NONE		PP_VTT_50
select_digital_led[0]	OUT		J17			15 LVCMOS33*	3.300	12	SLOW	NONE		PP_VTT_50
Scalar ports (3)												

4. 向 Flash 中烧录 bin 数据：

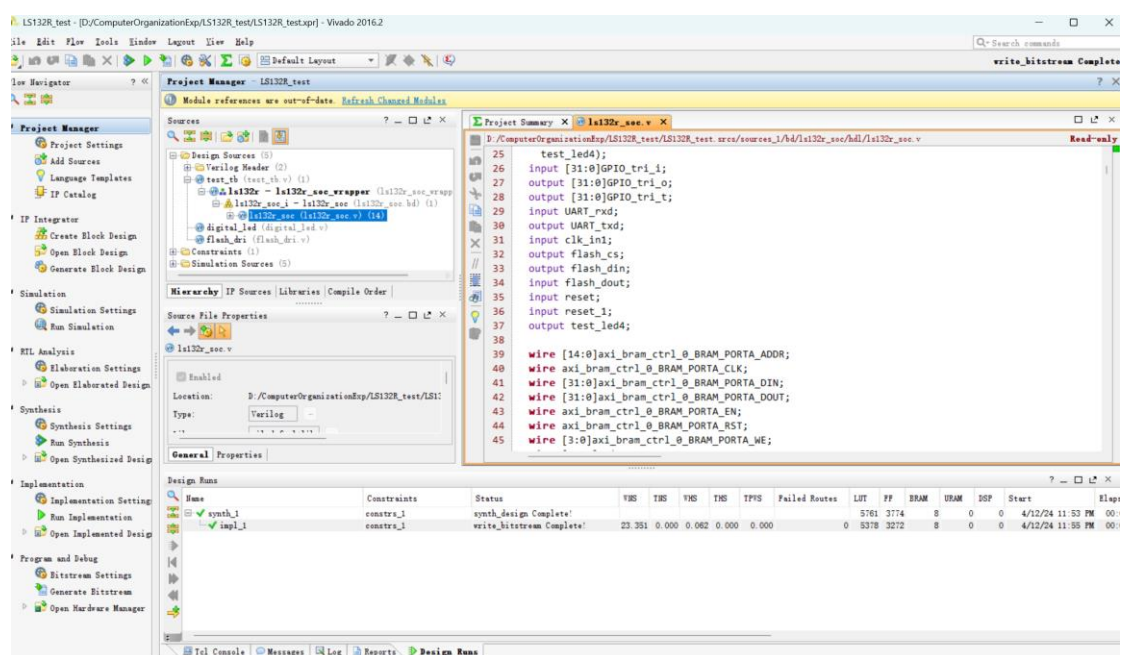


5. 下载线连接开发板，打开开发板电源，进行下载。按动 P17 按钮后，数码管显示下一次从 Flash 中读到的数值。如图所示：

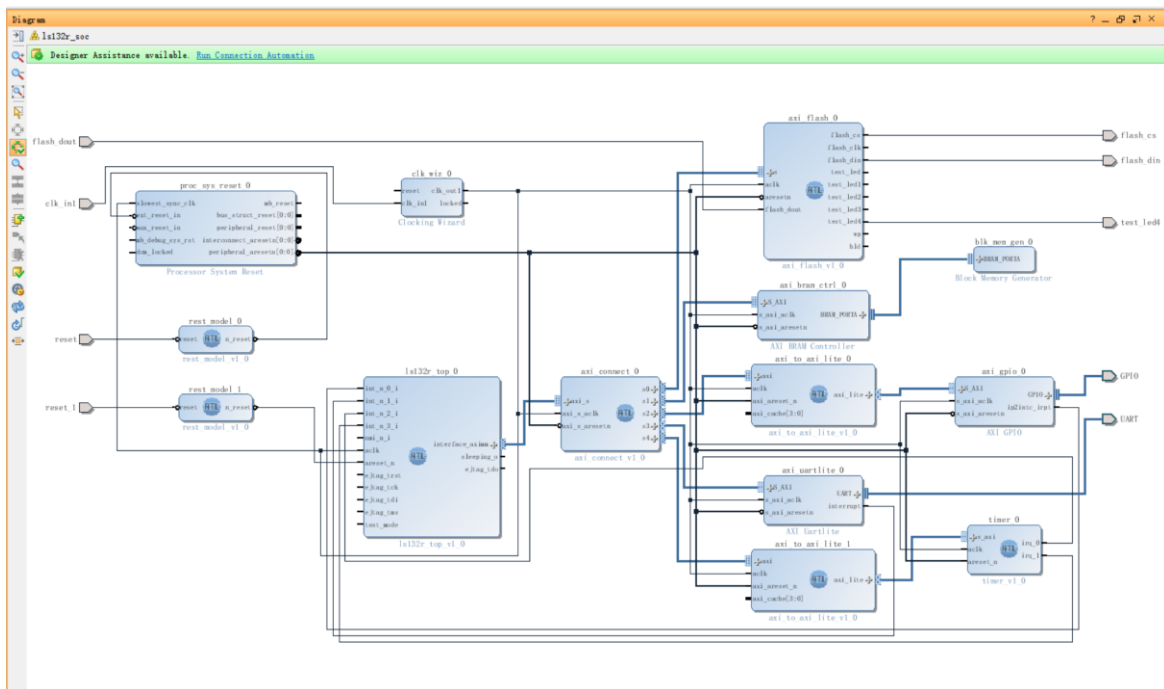


2.3 AXI 通信实验

1. 创建工程，导入相关代码：



2. 创建 Block Design 并进行连线:

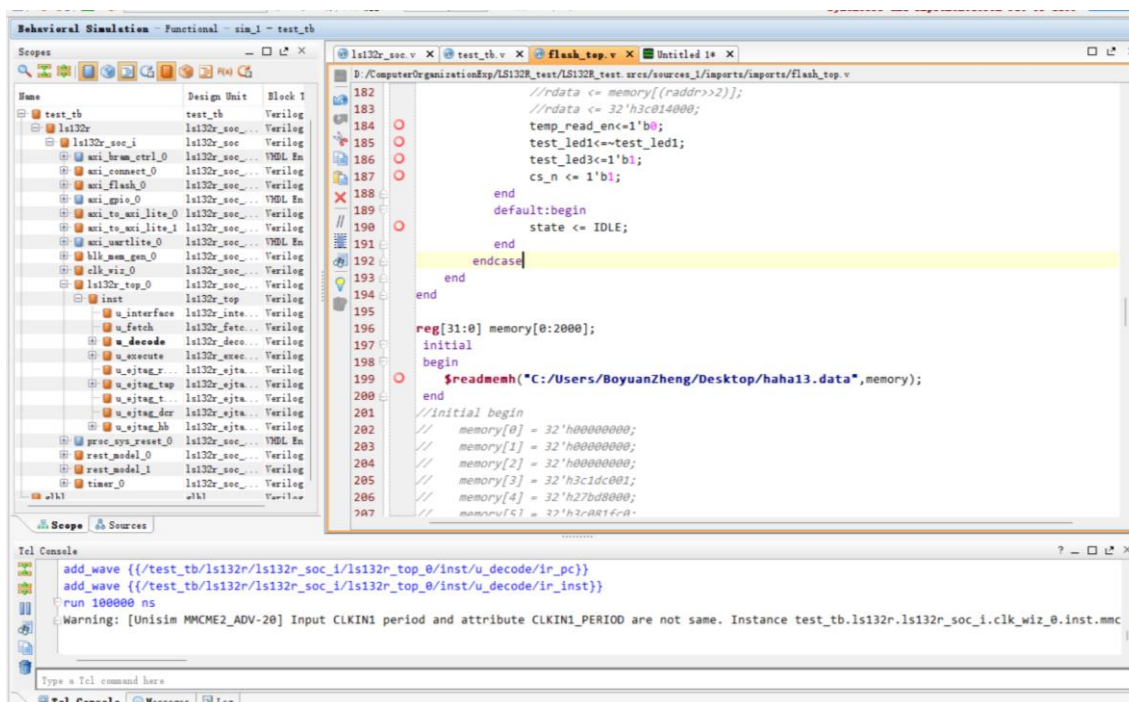


3. 进行地址分配:

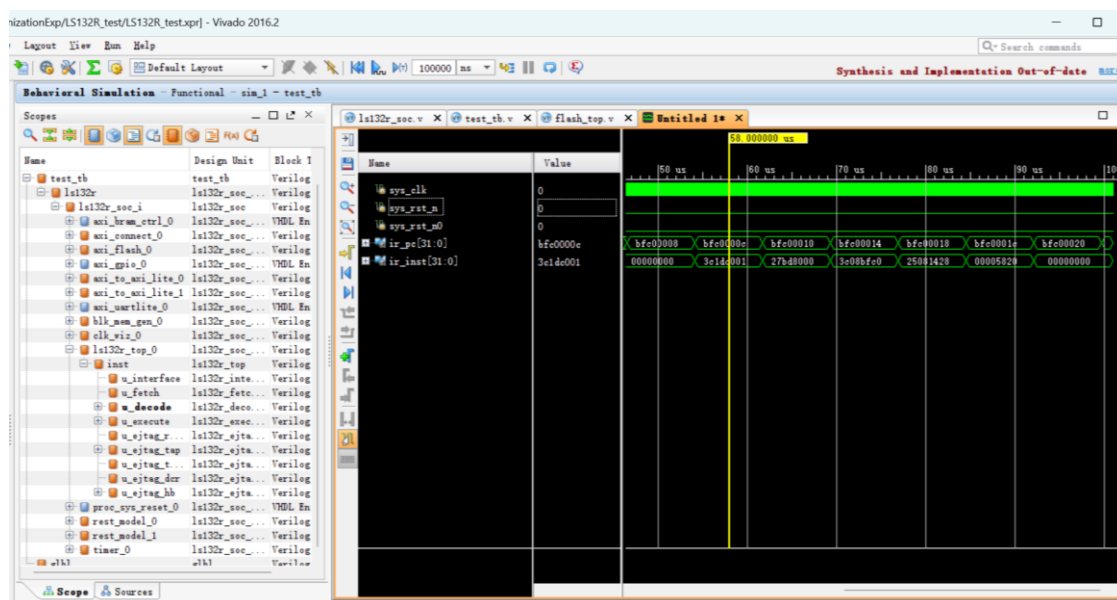
Cell	Slave Interface	Base Name	Offset Address	Range	High Address
ls132r_top_0					
interface_aximm (32 address bits : 4G)					
Unmapped Slaves (1)					
axi_connect_0	axi_s	reg0			
axi_to_axi_lite_0					
axi_lite (32 address bits : 4G)					
axi_gpio_0	S_AXI	Reg	0xD000_0000	64K	0xD000_FFFF
axi_connect_0					
s0 (32 address bits : 4G)					
Unmapped Slaves (1)					
axi_flash_0	s	reg0			
s1 (32 address bits : 4G)					
Unmapped Slaves (1)					
axi_bram_ctrl_0	S_AXI	Mem0			
s2 (32 address bits : 4G)					
Unmapped Slaves (1)					
axi_to_axi_lite_0	axi	reg0			
s3 (32 address bits : 4G)					
Unmapped Slaves (1)					
axi_uartlite_0	S_AXI	Reg			
s4 (32 address bits : 4G)					
Unmapped Slaves (1)					
axi_to_axi_lite_1	axi	reg0			
axi_to_axi_lite_1					

4. 将 block design 设置为顶层文件,通过 block design 帮助生成 verilog 模块。

5. 修改并设置 flash_top.v, 读取仿真时对应的文件:

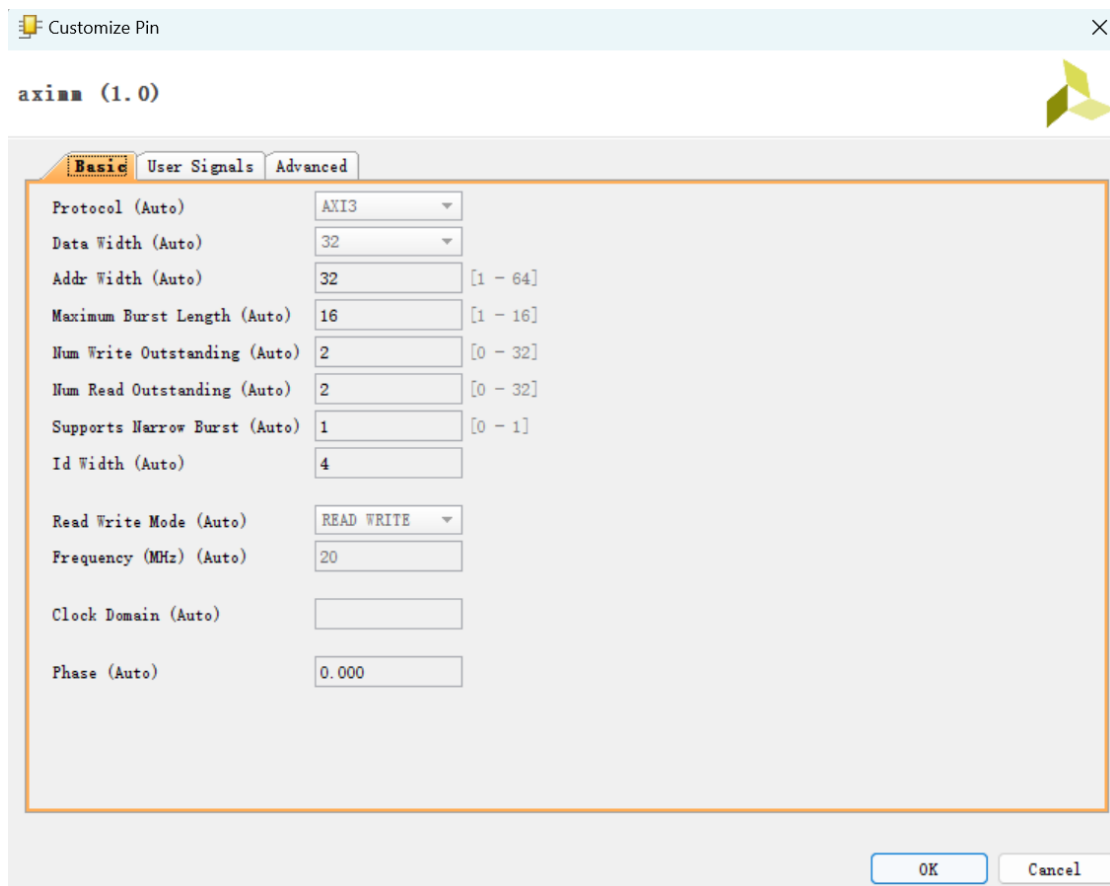


6. 运行前仿真，观察到 PC 正常取值：



2.4 汇编版点亮 LED 实验

1. 修改时钟分频输出为 20MHz，同时修改 Block Design 中其他对应位置处的频率，如下图所示：



2. 将对应的 asm 汇编代码汇编成 bin 文件：

```
命令提示符
Microsoft Windows [版本 10.0.22631.3447]
(c) Microsoft Corporation. 保留所有权利。

C:\Users\BoyuanZheng\Desktop\Logon_cpu_trans_to_nexy_4发放\Logon_cpu_trans_to_nexy_4\toolchain\build>..\bin\mips-mti-elf-as.exe -32 -mips32 led_asm.s -o led_asm.o
led_asm.s: Assembler messages:
led_asm.s:8: Warning: used $at without ".set noat"
led_asm.s:9: Warning: used $at without ".set noat"
led_asm.s:11: Warning: used $at without ".set noat"
led_asm.s:14: Warning: used $at without ".set noat"
led_asm.s:15: Warning: used $at without ".set noat"
led_asm.s:18: Warning: used $at without ".set noat"
led_asm.s:28: Warning: used $at without ".set noat"
led_asm.s:29: Warning: used $at without ".set noat"
led_asm.s:32: Warning: used $at without ".set noat"

C:\Users\BoyuanZheng\Desktop\Logon_cpu_trans_to_nexy_4发放\Logon_cpu_trans_to_nexy_4\toolchain\build>..\bin\mips-mti-elf-ld.exe -T mytest.ld led_asm.o -o led_asm.om
..\bin\mips-mti-elf-ld.exe: warning: cannot find entry symbol Reset_Handler; defaulting to 00000000bfc00000

C:\Users\BoyuanZheng\Desktop\Logon_cpu_trans_to_nexy_4发放\Logon_cpu_trans_to_nexy_4\toolchain\build>..\bin\mips-mti-elf-objcopy.exe -O binary led_asm.om led_asm.bin

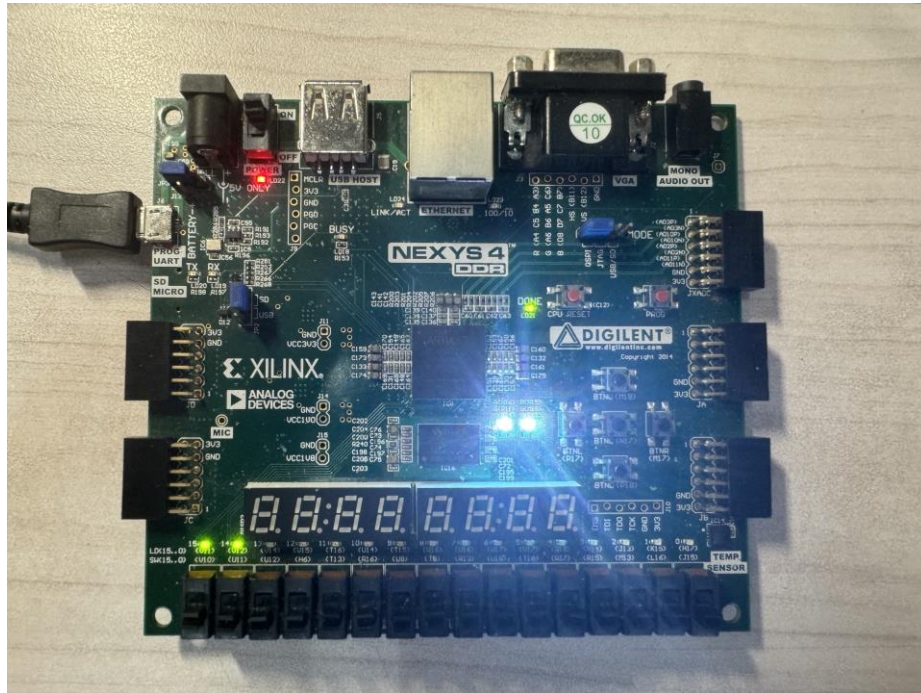
C:\Users\BoyuanZheng\Desktop\Logon_cpu_trans_to_nexy_4发放\Logon_cpu_trans_to_nexy_4\toolchain\build>..\bin\mips-mti-elf-objdump.exe -D led_asm.om > led_asm.asm

C:\Users\BoyuanZheng\Desktop\Logon_cpu_trans_to_nexy_4发放\Logon_cpu_trans_to_nexy_4\toolchain\build>
```

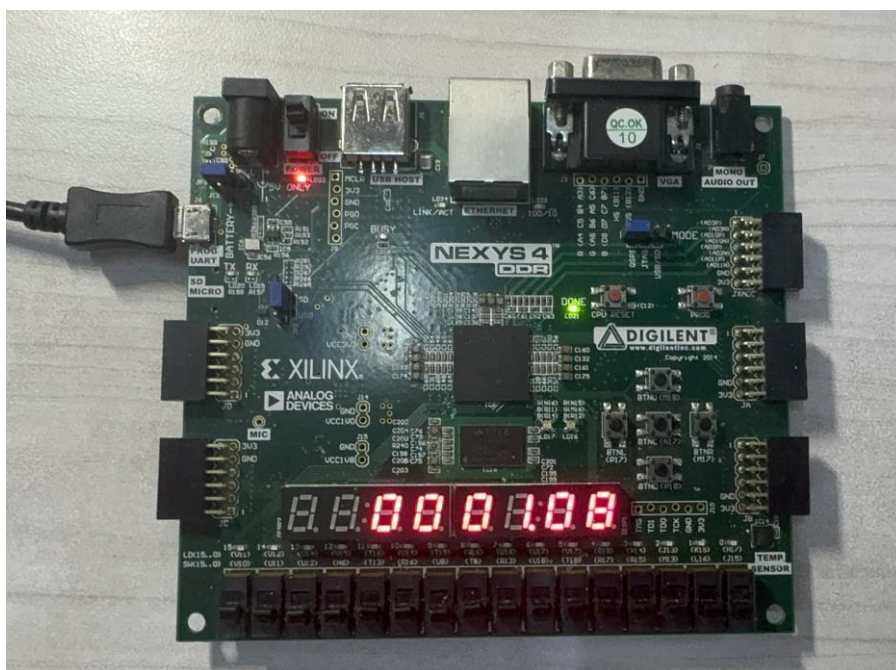
3. 向 Flash 中烧录刚刚生成的 bin 数据。

3. 向 Flash 中烧录刚刚生成的 bin 数据。

4. 下载线连接开发板，打开开发板电源，进行下载。在本次工程中， P17 是总线复位按钮， M18 是 CPU 复位按钮。先按动总线复位，再按动 CPU 复位，此后两个 RGB 灯白色常亮， V10、U11 闪烁。如图所示：



2.5 C 语言版时钟实验



1. 前置步骤与 2.4 中类似。

2. 下载线连接开发板，打开开发板电源，进行下载。在本次工程中， P17 是总线复位按钮， M18 是 CPU 复位按钮。先按动总线复位，再按动 CPU 复位，此后数码管显示当前运行秒数。如上图所示。

三、程序修改说明

本次实验中，我未对龙芯 CPU 内核代码进行修改。在仿真时， flash_top.v 文件中有如下部分需要修改：

在仿真时，应当从文件中读取数据：

```
rdata <= memory[(raddr>>2)];  
...  
reg[31:0] memory[0:2000];  
initial  
begin  
    $readmemh("C:/Users/BoyuanZheng/Desktop/haha13.data",m  
    emory);  
end
```

在下板时，读取数据部分代码应该修改为：

```
rdata[31:24]<=temp_rdata[7:0];  
rdata[23:16]<=temp_rdata[15:8];  
rdata[15:8]<=temp_rdata[23:16];  
rdata[7:0]<=temp_rdata[31:24];
```


四、约束文件修改说明

本实验中未对约束文件进行修改，具体约束文件如下：

```
set_property PACKAGE_PIN E3 [get_ports clk_in1_0]
set_property IOSTANDARD LVCMOS33 [get_ports clk_in1_0]
set_property PACKAGE_PIN E18 [get_ports
{GPIO_0_tri_io[31]}]
set_property PACKAGE_PIN G17 [get_ports
{GPIO_0_tri_io[30]}]
set_property PACKAGE_PIN D17 [get_ports
{GPIO_0_tri_io[29]}]
set_property PACKAGE_PIN E17 [get_ports
{GPIO_0_tri_io[28]}]
set_property PACKAGE_PIN N17 [get_ports
{GPIO_0_tri_io[27]}]
set_property PACKAGE_PIN P18 [get_ports
{GPIO_0_tri_io[26]}]
set_property PACKAGE_PIN L16 [get_ports
{GPIO_0_tri_io[25]}]
set_property PACKAGE_PIN J15 [get_ports
{GPIO_0_tri_io[24]}]
set_property PACKAGE_PIN V12 [get_ports
{GPIO_0_tri_io[23]}]
set_property PACKAGE_PIN V11 [get_ports
{GPIO_0_tri_io[22]}]
set_property PACKAGE_PIN N16 [get_ports
{GPIO_0_tri_io[21]}]
set_property PACKAGE_PIN R11 [get_ports
{GPIO_0_tri_io[20]}]
set_property PACKAGE_PIN G14 [get_ports
{GPIO_0_tri_io[19]}]
set_property PACKAGE_PIN N15 [get_ports
{GPIO_0_tri_io[18]}]
set_property PACKAGE_PIN M16 [get_ports
{GPIO_0_tri_io[17]}]
set_property PACKAGE_PIN R12 [get_ports
{GPIO_0_tri_io[16]}]
set_property PACKAGE_PIN U13 [get_ports
{GPIO_0_tri_io[15]}]
set_property PACKAGE_PIN K2 [get_ports {GPIO_0_tri_io[14]}]
set_property PACKAGE_PIN T14 [get_ports
{GPIO_0_tri_io[13]}]
```

```

set_property PACKAGE_PIN P14 [get_ports
{GPIO_0_tri_io[12]}}]
set_property PACKAGE_PIN J14 [get_ports
{GPIO_0_tri_io[11]}}]
set_property PACKAGE_PIN T9 [get_ports {GPIO_0_tri_io[10]}}]
set_property PACKAGE_PIN J18 [get_ports {GPIO_0_tri_io[9]}}]
set_property PACKAGE_PIN J17 [get_ports {GPIO_0_tri_io[8]}}]
set_property PACKAGE_PIN T10 [get_ports {GPIO_0_tri_io[7]}}]
set_property PACKAGE_PIN R10 [get_ports {GPIO_0_tri_io[6]}}]
set_property PACKAGE_PIN K16 [get_ports {GPIO_0_tri_io[5]}}]
set_property PACKAGE_PIN K13 [get_ports {GPIO_0_tri_io[4]}}]
set_property PACKAGE_PIN P15 [get_ports {GPIO_0_tri_io[3]}}]
set_property PACKAGE_PIN T11 [get_ports {GPIO_0_tri_io[2]}}]
set_property PACKAGE_PIN L18 [get_ports {GPIO_0_tri_io[1]}}]
set_property PACKAGE_PIN H15 [get_ports {GPIO_0_tri_io[0]}}]
set_property PACKAGE_PIN P17 [get_ports reset_0]
set_property PACKAGE_PIN M18 [get_ports reset_1]
set_property PACKAGE_PIN C17 [get_ports UART_0_rxd]
set_property PACKAGE_PIN D18 [get_ports UART_0_txd]
set_property PACKAGE_PIN L13 [get_ports flash_cs_0]
set_property PACKAGE_PIN K17 [get_ports flash_din_0]
set_property PACKAGE_PIN K18 [get_ports flash_dout_0]
set_property IOSTANDARD LVCMOS33 [get_ports
{GPIO_0_tri_io[31]}}]
set_property IOSTANDARD LVCMOS33 [get_ports
{GPIO_0_tri_io[30]}}]
set_property IOSTANDARD LVCMOS33 [get_ports
{GPIO_0_tri_io[29]}}]
set_property IOSTANDARD LVCMOS33 [get_ports
{GPIO_0_tri_io[28]}}]
set_property IOSTANDARD LVCMOS33 [get_ports
{GPIO_0_tri_io[27]}}]
set_property IOSTANDARD LVCMOS33 [get_ports
{GPIO_0_tri_io[26]}}]
set_property IOSTANDARD LVCMOS33 [get_ports
{GPIO_0_tri_io[25]}}]
set_property IOSTANDARD LVCMOS33 [get_ports
{GPIO_0_tri_io[24]}}]
set_property IOSTANDARD LVCMOS33 [get_ports
{GPIO_0_tri_io[23]}}]
set_property IOSTANDARD LVCMOS33 [get_ports
{GPIO_0_tri_io[22]}}]
set_property IOSTANDARD LVCMOS33 [get_ports

```

```

{GPIO_0_tri_io[21]}}
set_property      IOSTANDARD      LVCMOS33      [get_ports
{GPIO_0_tri_io[20]}}
set_property      IOSTANDARD      LVCMOS33      [get_ports
{GPIO_0_tri_io[19]}}
set_property      IOSTANDARD      LVCMOS33      [get_ports
{GPIO_0_tri_io[18]}}
set_property      IOSTANDARD      LVCMOS33      [get_ports
{GPIO_0_tri_io[17]}}
set_property      IOSTANDARD      LVCMOS33      [get_ports
{GPIO_0_tri_io[16]}}
set_property      IOSTANDARD      LVCMOS33      [get_ports
{GPIO_0_tri_io[15]}}
set_property      IOSTANDARD      LVCMOS33      [get_ports
{GPIO_0_tri_io[14]}}
set_property      IOSTANDARD      LVCMOS33      [get_ports
{GPIO_0_tri_io[13]}}
set_property      IOSTANDARD      LVCMOS33      [get_ports
{GPIO_0_tri_io[12]}}
set_property      IOSTANDARD      LVCMOS33      [get_ports
{GPIO_0_tri_io[11]}}
set_property      IOSTANDARD      LVCMOS33      [get_ports
{GPIO_0_tri_io[10]}}
set_property      IOSTANDARD      LVCMOS33      [get_ports
{GPIO_0_tri_io[9]}}
set_property      IOSTANDARD      LVCMOS33      [get_ports
{GPIO_0_tri_io[8]}}
set_property      IOSTANDARD      LVCMOS33      [get_ports
{GPIO_0_tri_io[7]}}
set_property      IOSTANDARD      LVCMOS33      [get_ports
{GPIO_0_tri_io[6]}}
set_property      IOSTANDARD      LVCMOS33      [get_ports
{GPIO_0_tri_io[5]}}
set_property      IOSTANDARD      LVCMOS33      [get_ports
{GPIO_0_tri_io[4]}}
set_property      IOSTANDARD      LVCMOS33      [get_ports
{GPIO_0_tri_io[3]}}
set_property      IOSTANDARD      LVCMOS33      [get_ports
{GPIO_0_tri_io[2]}}
set_property      IOSTANDARD      LVCMOS33      [get_ports
{GPIO_0_tri_io[1]}}
set_property      IOSTANDARD      LVCMOS33      [get_ports
{GPIO_0_tri_io[0]}}

```

```
set_property IOSTANDARD LVCMOS33 [get_ports reset_0]
set_property IOSTANDARD LVCMOS33 [get_ports reset_1]
set_property IOSTANDARD LVCMOS33 [get_ports UART_0_rxd]
set_property IOSTANDARD LVCMOS33 [get_ports UART_0_txd]
set_property IOSTANDARD LVCMOS33 [get_ports flash_cs_0]
set_property IOSTANDARD LVCMOS33 [get_ports flash_din_0]
set_property IOSTANDARD LVCMOS33 [get_ports flash_dout_0]

set_property PACKAGE_PIN V14 [get_ports test_led4_0]
set_property IOSTANDARD LVCMOS33 [get_ports test_led4_0]
```

五、仿真分析

对于撰写好的 asm 或 C 语言程序，通过汇编将其转换为 bin 文件后，还需将其转换为.data 文件才能在前仿真时进行读取。尝试使用实验教程中提供的 Bin2Mem.exe 程序似乎并不能奏效，因此自己撰写 python 代码如下：

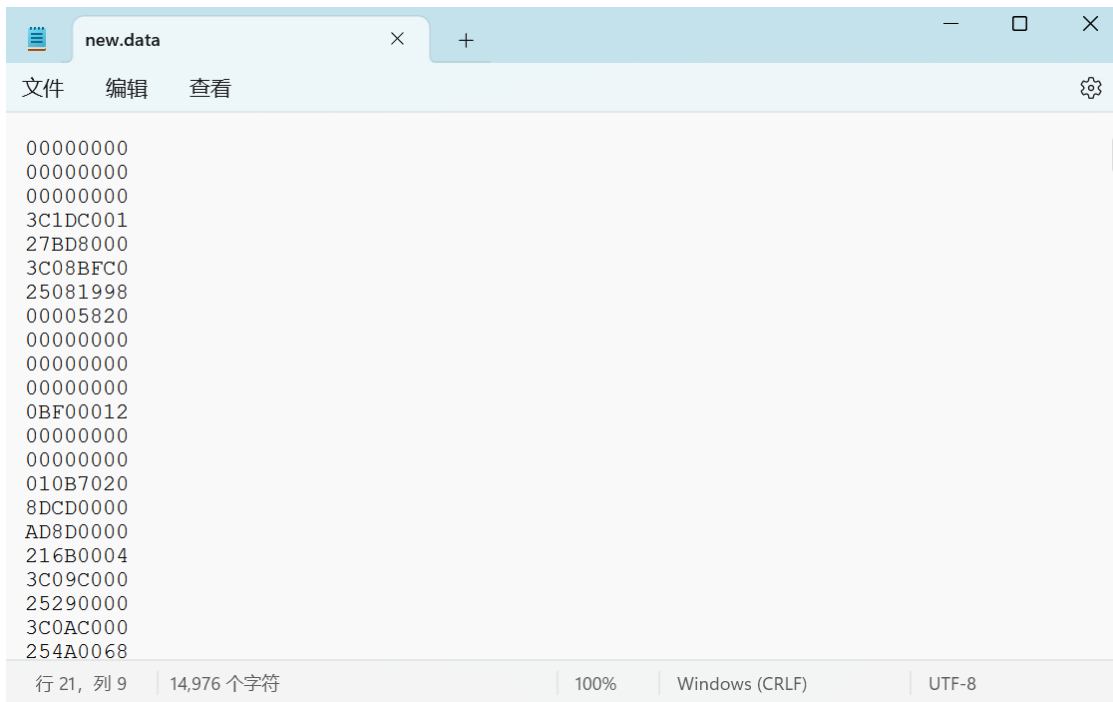
```
name = "test"
filepath = name + ".bin"
targetpath = name + ".data"
target = open(targetpath, "w")

binfile = open(filepath, 'rb')
i = 0
ch = binfile.read(1)

while ch:
    data = ord(ch)
    target.write("%02X" % (data))
    if i % 4 == 3:
        target.write("\n")
    i = i + 1
    ch = binfile.read(1)

binfile.close()
```

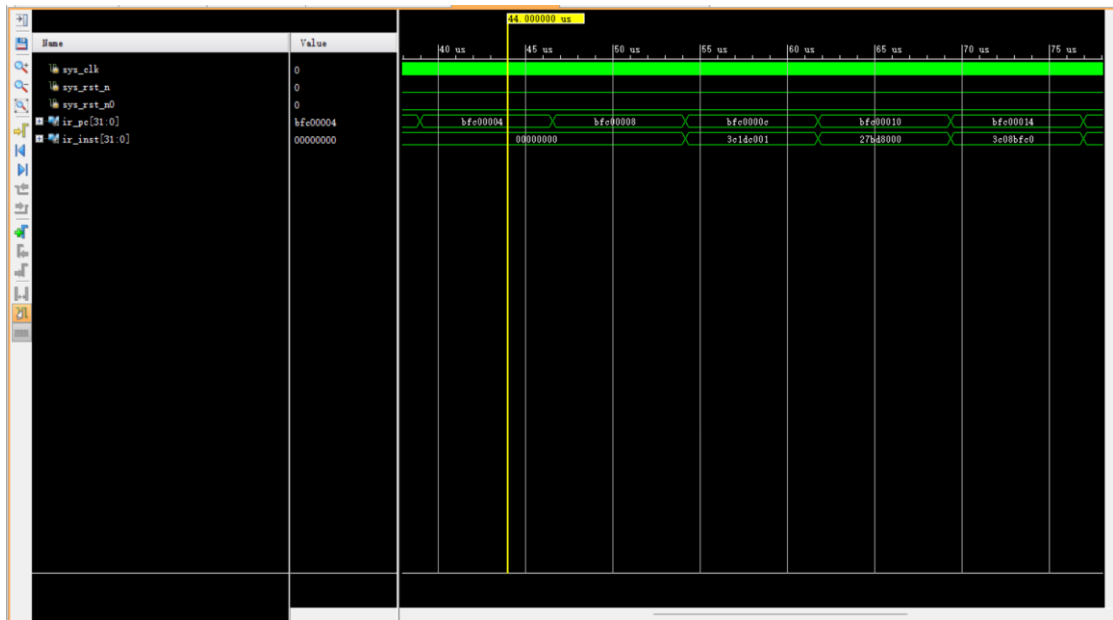

数据文件内容如下：



```
00000000
00000000
00000000
3C1DC001
27BD8000
3C08BFC0
25081998
00005820
00000000
00000000
00000000
0BF00012
00000000
00000000
010B7020
8DCD0000
AD8D0000
216B0004
3C09C000
25290000
3C0AC000
254A0068
```

行 21, 列 9 | 14,976 个字符 | 100% | Windows (CRLF) | UTF-8

观察到前仿真结果如下。可以观察到，龙芯 CPU 在取指令时从地址 0xbfc00000 开始。取到的指令与文件中一致：



六、性能验证数学模型及算法程序

```
a[m],b[m],c[m],d[m];  
a[0]=0.0;↵  
b[0]=1.0;↵  
a[i]=a[i-1]+i;↵  
b[i]=b[i-1]+3i;↵  
c[i]= $\begin{cases} a[i], & 0 \leq i \leq 9 \\ a[i]+b[i], & 10 \leq i \leq 29 \\ (a[i]*b[i]) \ll 1, & 30 \leq i \leq 49 \end{cases}$ ↵  
d[i]= $\begin{cases} b[i]+c[i], & 0 \leq i \leq 9 \\ a[i]*c[i], & 10 \leq i \leq 29 \\ c[i]*b[i]/(d[i-1] \gg 1), & 30 \leq i \leq 49 \end{cases}$ ↵
```

我尝试使用C语言中的浮点数类型 float 与定点数类型 int16_t 来撰写 C 程序，但最终都由于龙芯 CPU 缺乏对应的指令而无法成功运行（龙芯 CPU 不支持浮点运算）。因此，最终我选择了无符号整型数做为 a、b、c、d 数组的类型，并撰写 C 程序，代码如下：

```
#include "../include/minicrt.h"  
#include "../include/system.h"  
#include "../include/gpio.h"  
int kk = 5;  
static unsigned int gpio_data = 0;  
static unsigned int gpio_tri = 0;  
  
//L16 25  
//J15 24  
void write_gpio(void) {  
    unsigned int* gpio_data_addr = GPIO_DATA_ADDR;  
    *gpio_data_addr = gpio_data;  
}
```

```

void udelay1(int us){
    int i = 0;
    for(i=0;i<us;i++){

    }
}

//p17 总线
//m18 cpu
// 是否使能 gpio 全局中断
void enable_gpio_irq(bool enable){
    unsigned int* gpio_gier_addr = GIER_ADDR;
    if(enable == true){
        *gpio_gier_addr = 0xffffffff;
    }else{
        *gpio_gier_addr = 0x0;
    }
}

// 是否使能 gpio 通道 1 的中断
void enable_gpio_channell_irq(bool enable){
    unsigned int* gpio_ip_ier_addr = IP_IER_ADDR;
    if(enable == true){
        *gpio_ip_ier_addr = 0x1;
    }else{
        *gpio_ip_ier_addr = 0x0;
    }
}

void clear_gpio_channell_irq_flag(){
    unsigned int* gpio_ip_isr_addr = IP_ISR_ADDR;
    *gpio_ip_isr_addr = 0x0;
}

//设置 gpio 是输出还是输入
void set_gpio_tri(unsigned int value,bool is_input){
    unsigned int* gpio_tri_addr = GPIO_TRI_ADDR;
    if(is_input == true){
        //这个设计的目的是保证其他位的值不变
        //例如设置第 1 位为输入,is_input = true,value = 0x0000 0001;
        *gpio_tri_addr = gpio_tri | value;
    } else{
        //这个设计的目的是保证其他位的值不变

```

```

        //例如设置第1位为输出,is_input = false,value = 0xffff fffe;
        *gpio_tri_addr = gpio_tri & value;
    }
}

// 闪烁小灯 gpio_22 ,gpio_23
void set_led(bool flag){
    if(flag == true){
        gpio_data = gpio_data | 0x00c00000;
    }else{
        gpio_data = gpio_data & 0xff3ffffff;
    }
    write_gpio();
}

unsigned int read_gpio(unsigned int sel_gpio){
    unsigned int* gpio_data_addr = GPIO_DATA_ADDR;
    return (*gpio_data_addr)&sel_gpio;
}

//      A
//  F      B
//      G
//  E      C
//      D      H
// 0000 0000
// ABCD EFGH
void digital_led(int id,int digital_num){
    unsigned int seg = 0;
    switch(digital_num){
        case 0: seg = 0x03;break; // 0000 0011
        case 1: seg = 0x9f;break; // 1001 1111
        case 2: seg = 0x25;break; // 0010 0101
        case 3: seg = 0x0d;break; // 0000 1101
        case 4: seg = 0x99;break; // 1001 1001
        case 5: seg = 0x49;break; // 0100 1001
        case 6: seg = 0x41;break; // 0100 0001
        case 7: seg = 0x1f;break; // 0001 1111
        case 8: seg = 0x01;break; // 0000 0001
        case 9: seg = 0x09;break; // 0000 1001
        case 10: seg = 0x11;break; // 0001 0001
        default: seg = 0x00;
    }
}

```



```

switch(id){
    case 0:seg = 0xfe00 | seg;break;           // 1111 1110 fe
    case 1:seg = 0xfd00 | seg;break;           // 1111 1101 fd
    case 2:seg = 0xfb00 | (seg&0xfe);break;     // 1111 1011 fb
    case 3:seg = 0xf700 | seg;break;           // 1111 0111 f7
    case 4:seg = 0xef00 | seg;break;           // 1110 1111 ef
    case 5:seg = 0xdf00 | seg;break;           // 1101 1111 df
    case 6:seg = 0xbf00 | seg;break;           // 1011 1111 bf
    case 7:seg = 0x7f00 | seg;break;           // 0111 1111 7f
    default: seg = 0x00;
}
gpio_data = gpio_data & 0xffff0000;
gpio_data = gpio_data | seg;
write_gpio();
}

```

//说明，虽然 cpu 的输入时钟采用了 20Mhz，但 flash 取数据极慢，估计一条取值需要 20 个周期，顾实际运行

//频率大概为 1~2Mhz,当然您可以选择从 DDR 中进行取指令，

//改变 flash 的时钟也可以实现一定的提速，时间仓促，水平有限暂未实现相关设计。

//说明，本设计是仓促时间中，在龙芯 ls132r cpu 核的情况下，添加外设，

// 添加链接脚本编写的，存在许多未知 bug,请务必抱着怀疑的态度参考使用。

```
int main(void){
```

```

    set_gpio_tri(0xffff0000,false);//设置低 16 位为输出。
    set_gpio_tri(0x02000000,true); //设置第 25 位为输入，L16

```

```

unsigned int flag;
unsigned int a[50], b[50], c[50], d[50];
unsigned int i = 0, display;
a[0] = 0;
b[0] = 1;

```

```

while(1){
    unsigned int cnt = 0;
    while(1){
        if (i < 50 && cnt % 50 == 0){
            if (i > 0){
                a[i] = a[i - 1] + i;
                b[i] = b[i - 1] + 3 * i;
            }
            if (i <= 9){

```

```

        c[i] = a[i];
        d[i] = b[i] + c[i];
    }
    else if (i >= 10 && i <= 29){
        c[i] = a[i] + b[i];
        d[i] = a[i] * c[i];
    }
    else{
        c[i] = (a[i] * b[i]) << 1;
        d[i] = c[i] * b[i] / (d[i - 1] >> 1);
    }
    i++;
}
cnt++;

display = d[i - 1];

digital_led(0, display % 10);
udelay(3000);
digital_led(1, (display / 10 ) % 10);
udelay(3000);
digital_led(2, (display / 100 ) % 10);
udelay(3000);
digital_led(3, (display / 1000) % 10);
udelay(3000);
digital_led(4, (display / 10000) % 10);
udelay(3000);
digital_led(5, (display / 100000) % 10);
udelay(3000);
digital_led(6, (display / 1000000) % 10);
udelay(3000);
digital_led(7, (display / 10000000) % 10);
udelay(3000);

flag = read_gpio(0x03000000);
if(flag !=0)
    set_led(true);
else
    set_led(false);
}

}

return 0;

```

```
}
```

此外，编写汇编 asm 代码如下：

```
.data
A: .space 200
B: .space 200
C: .space 200
D: .space 200

.text
j main

main:
    # 初始化寄存器
    addi $2, $0, 0 # a[i]
    addi $3, $0, 1 # b[i]
    addi $15, $0, 0 # c[i]
    addi $16, $0, 0 # d[i]
    addi $5, $0, 4 # 计数器
    addi $6, $0, 0 # a[i-1]
    addi $7, $0, 1 # b[i-1]
    addi $10, $0, 0 # 用于标记 i<20 或 i<40
    addi $11, $0, 200 # 总计数
    addi $14, $0, 3

    # 保存寄存器内容到数组 A、B、C、D
    lui $27, 0x0000
    addu $27, $27, $0
    sw $2, A($27)
    lui $27, 0x0000
    addu $27, $27, $0
    sw $3, B($27)
    lui $27, 0x0000
    addu $27, $27, $0
    sw $2, C($27)
    lui $27, 0x0000
    addu $27, $27, $0
    sw $3, D($27)

loop:
    # 将 $5(4) / 4 (得到 i) 存入 $12
```

```

srl $12, $5, 2
# $6 加上 i。
add $6, $6, $12
# 将 a[i] 存入 A[i]
lui $27, 0x0000
addu $27, $27, $5
sw $6, A($27)
# $14 (3) * $5/4 (得到 3i)
mul $15, $14, $12
# $7 (b[i-1]) 加上 3i, 并存入 B[i]。
add $7, $7, $15
lui $27, 0x0000
addu $27, $27, $5
sw $7, B($27)
# 若 $5 < 40 (即 i < 10) 则存入 $10
slti $10, $5, 40
bne $10, 1, c1

# (0<=i<=9)
# 将 $6 存入 C[i] (c[i] = a[i])
lui $27, 0x0000
addu $27, $27, $5
sw $6, C($27)
# 将 $7 存入 D[i] (d[i] = b[i])
lui $27, 0x0000
addu $27, $27, $5
sw $7, D($27)
addi $15, $6, 0 # $15 $16 = c[i] d[i]
addi $16, $7, 0
j endc

c1: # (10<=i<=29)
# 若 i < 30 则跳转至 c2
slti $10, $5, 120
addi $27, $0, 1
bne $10, $27, c2
# C[i] = a[i] + b[i]
add $15, $6, $7
lui $27, 0x0000
addu $27, $27, $5
sw $15, C($27)
# D[i] = a[i] * b[i]
mul $16, $15, $6

```



```

    lui $27, 0x0000
    addu $27, $27, $5
    sw $16, D($27)
    j endc

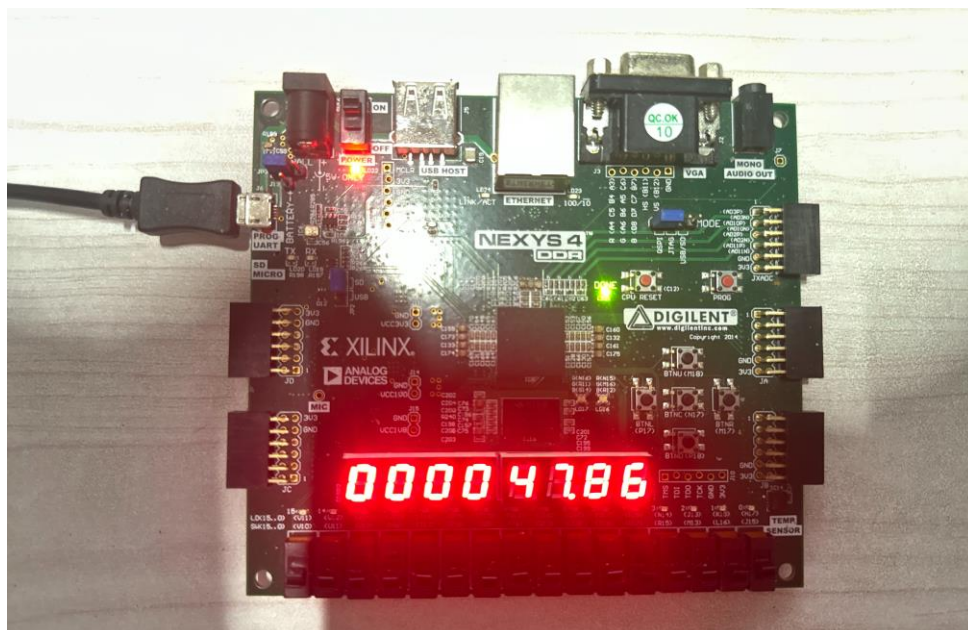
c2: # (i>=30)
    # C[i] = a[i] * b[i]
    mul $15, $6, $7
    lui $27, 0x0000
    addu $27, $27, $5
    sw $15, C($27)
    # D[i] = c[i] * b[i]
    mul $16, $15, $7
    lui $27, 0x0000
    addu $27, $27, $5
    sw $16, D($27)

endc:
    addi $5, $5, 4 # i = i + 1
    bne $5, $11, loop # 若 i != 50 则跳回 loop

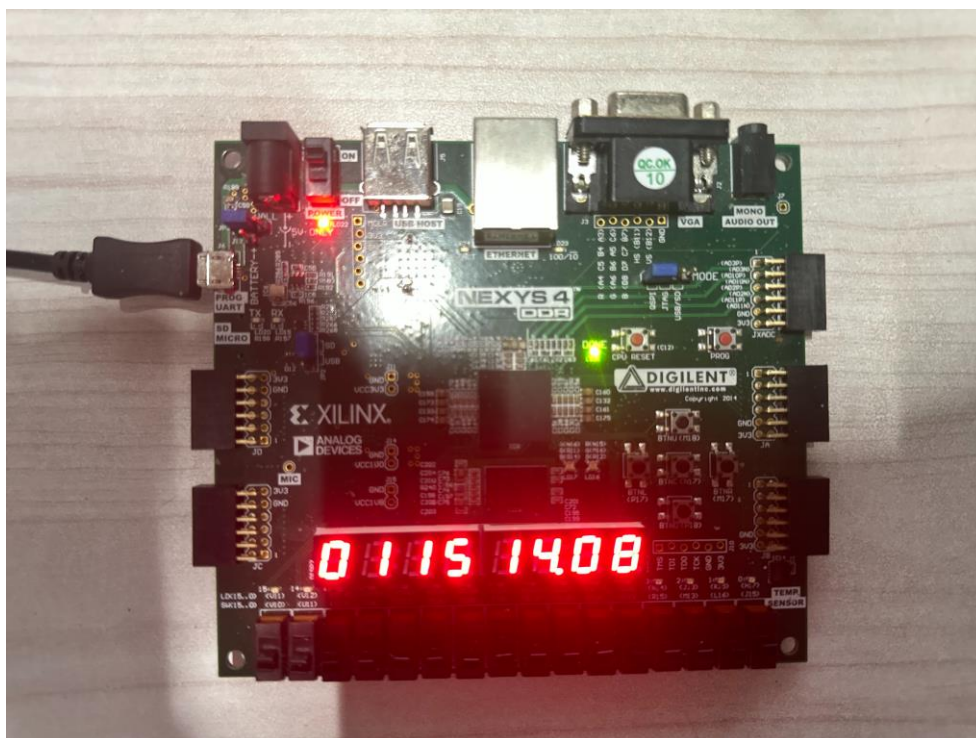
```

七、性能验证程序下板测试过程与实现

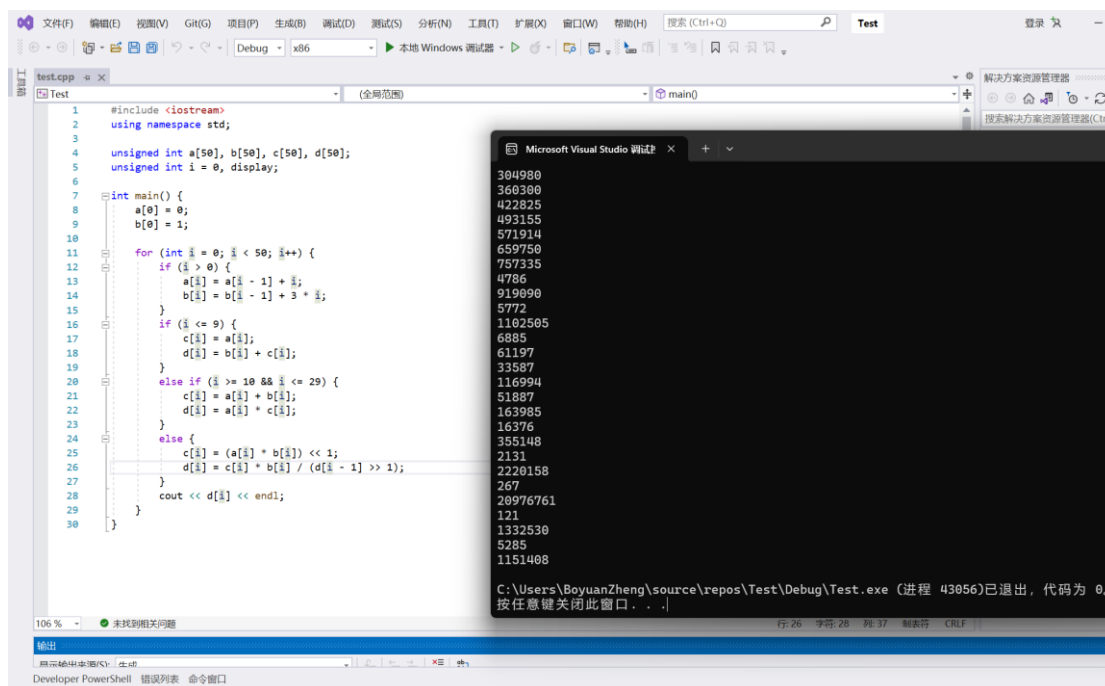
下板后，先后按动 P17、M18 按钮进行复位。此后，7 段数码管间隔跳动 i 从 0~49 逐个计算出的 $d[i]$ 值，如下图所示：



最终，数码管上定格显示 d[49]的值 1151408，如下图：



另外撰写 C++程序，展示计算的每个 d[i]值可以发现，数码管上闪动的 d[i]与最后定格的数值与真实值一致，CPU 运行正确无误：



```
1 #include <iostream>
2 using namespace std;
3
4 unsigned int a[50], b[50], c[50], d[50];
5 unsigned int i = 0, display;
6
7 int main() {
8     a[0] = 0;
9     b[0] = 1;
10
11     for (int i = 0; i < 50; i++) {
12         if (i > 0) {
13             a[i] = a[i - 1] + i;
14             b[i] = b[i - 1] + 3 * i;
15         }
16         if (i <= 9) {
17             c[i] = a[i];
18             d[i] = b[i] + c[i];
19         }
20         else if (i >= 10 && i <= 29) {
21             c[i] = a[i] + b[i];
22             d[i] = a[i] * c[i];
23         }
24         else {
25             c[i] = (a[i] + b[i]) << 1;
26             d[i] = c[i] + b[i] / (d[i - 1] >> 1);
27         }
28         cout << d[i] << endl;
29     }
30 }
```

304980
360300
422825
493155
571914
659750
757335
4786
919090
5772
1192505
6885
61197
33587
116994
51887
163985
16376
355148
2131
2220158
267
20976761
121
1332530
5285
1151408

C:\Users\BoyuanZheng\source\repos\Test\Debug\Test.exe (进程 43056)已退出，代码为 0。
按任意键关闭此窗口...]

八、CPU 的性能指标定性分析（分别用 C 语言和 MIPS 指令编写性能验证程序，C 语言利用 gcc 编译器编译生成目标程序，MIPS 指令汇编程序 Mars 编译生成目标程序，测试并比较分析两种方式 CPU 的定点运算性能及差异，单位：MIPS）

在进行 CPU 的性能指标定性分析时，为方便指令统计与指标分析，修改 C 程序与汇编程序并作如下约定：

1. 通过 Vivado 仿真分析一次性能评估 C 程序所执行的指令数量。为便于分析，确保程序只运行性能验证程序的代码部分，去除 C 语言代码中有关七段数码管与 LED 灯等相关代码；

2. 修改原先用于展示的 C 语言版本程序，不进行七段数码管显示与延迟，主体程序只进行性能测试计算。此外，下板运行时由于计算一次时间过短不便统计，将性能验证测试程序运行 2000 次进行总体时间统计。计算之后，通过 gpio 点亮 LED 灯，能够计算得知 CPU 运行 2000 次测试程序的总时间（忽略最终点亮 LED 灯以及用于控制 2000 次循环的跳转指令等）。

1. 分析 C 语言性能验证程序性能：

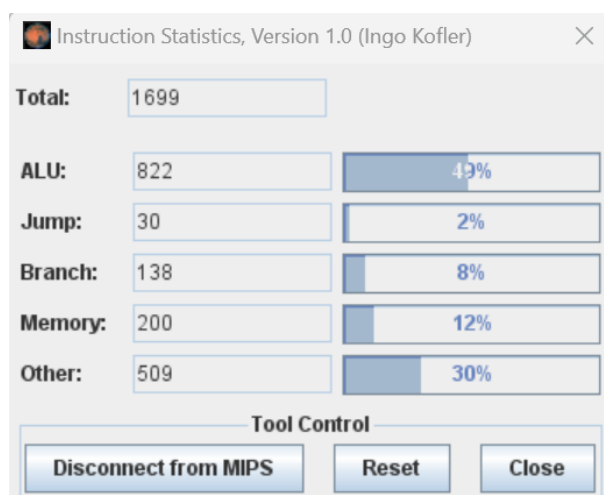
首先，撰写一次仅含纯测试程序，不含数码管、LED 灯等部分的 C 语言代码，使用 gcc 将其编译成二进制指令文件用于 Vivado 仿真。通过分析仿真波形图可以估计得到运行一次 C 语言测试程序共执行 4890 条指令。

其次，将循环 2000 次的 C 语言程序编译、下板运行，并通过 LED 灯的点亮时间估计程序运行总时间。通过实验结果计算得到，程序共运行 31 秒左右。又因为一次循环执行 4890 条指令，2000 次循环共执行 9780000 条指令。因此，可以计算得到吞吐率为：

$$TF = \frac{n}{T} = \frac{4890 \times 2000}{31} = 0.315\text{MIPS}$$

2. 分析 MIPS 指令汇编程序性能：

根据 MARS 分析，一次性能验证程序的汇编代码共 1699 条指令，具体包含 822 条算术逻辑指令、30 条跳转指令、138 条分支指令、200 条访存指令及 509 条其他指令，具体如下图：



根据下板实验结果，MIPS 汇编程序共运行 11 秒左右，因此可以计算得到 MIPS 指令汇编程序的吞吐率为：

$$TF = \frac{n}{T} = \frac{1699 \times 2000}{11} = 0.309\text{MIPS}$$

据在 Nexys 4 DDR 开发板上的实际下板测试结果显示，龙芯 CPU 在 C 语言与 MIPS 汇编所编写的测试程序上表现基本相当，性能差异不大。使用汇编语言和 C 语言编写的性能验证程序的执行吞吐率均能达到 0.3 MIPS 左右。

九、实验体会

在本次实验过程中，我完成了 Nexys 4 DDR 开发板上移植龙芯 CPU LS132R 的实验任务。由于是第一次进行此类的移植实验，我在过程中遇到了不少困难与挑战；但我最终通过不断调试，成功地完成了对龙芯 CPU LS132R 的移植。在整个实验过程中，我学到了很多关于硬件移植和 FPGA 编程的知识，并对嵌入式系统的工作原理有了更深入的理解。

首先，我简单了解了龙芯 CPU LS132R 和 LS232R 两个不同复杂度的 CPU，并按照实验要求我选择了较为简单的 LS132R 进行移植。在移植过程中，我大致研究了 LS132R 的 Verilog 源码和仿真文件，并简单理解了其工作原理和结构。

其次，我学会了使用 Vivado 的 Block Design 功能进行可视化的模块连接，将教程中提供好的各个模块文件组合在一起。我还学会了通过 Clock Wizarding 等不同的 IP 核实现时钟分频等必要的功能。

此外，在 CPU 移植过程中，我还学会了如何将 ASM 汇编程序与 C 语言代码转换为 bin 文件并烧录到 flash 中；并在 FPGA 程序烧录完成后，通过按下开发板上的复位键来启动 LS132R 核。在启动过程中，flash 程序的任务是将必要的内容搬运到 RAM 中，完成初始化过程，然后才能进入 main 函数运行 C 语言程序。这个过程不仅是技术上的挑战，对我来说更是对系统运作机制的了解。

本次龙芯 CPU LS132R 的移植实验让我收获颇丰。我不仅掌握了硬件移植和 FPGA 编程的基本知识，还增强了解决问题的能力和实践经验。我深刻领悟到了移植工作的复杂性和挑战性，但同时也体味到了完成它所带来的成就感。未来我会不断努力学习、提升技术水平，为接下来的实验和项目做好充分准备。