lab 3 汇编程序设计

PB21081601 张芷苒

一 实验目的

- 理解RISC-V常用32位整数指令功能
- 熟悉RISC-V汇编仿真软件RARS，掌握程序调试的基本方法
- 掌握RISC-V简单汇编程序设计，以及存储器初始化文件(COE)的生成方法
- 理解CPU调试模块PDU的使用方法

二 实验任务

1. 设计汇编程序：计算斐波那契-卢卡斯数列的前 n 项，并生成 COE 文件。
2. 设计 32bits 移位寄存器，通过 PDU 实现对移位寄存器数据的实时操作。

三 实验设计

1. 设计汇编程序：计算斐波那契-卢卡斯数列，生成 COE 文件

代码如下：

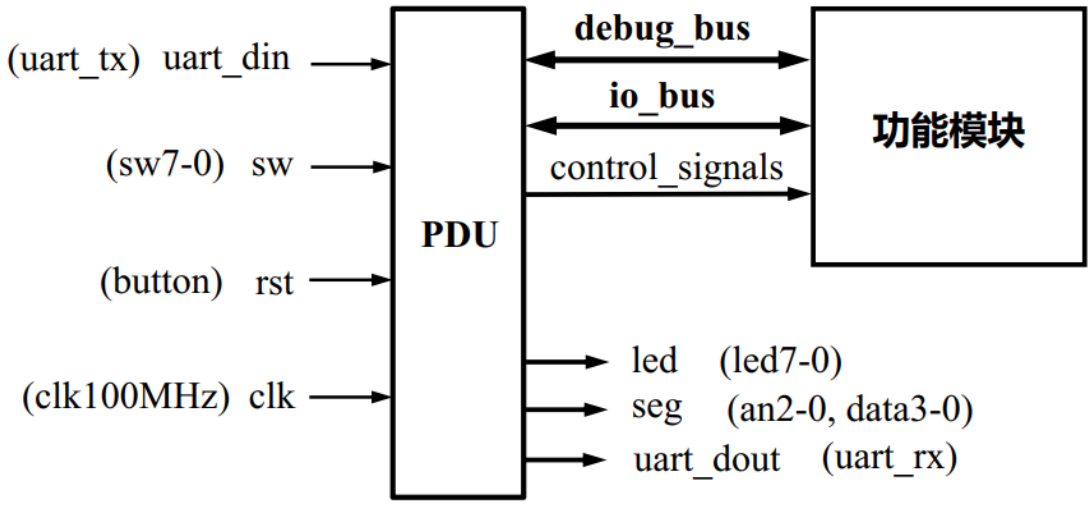
```
.data
    1                   #initialize fn-2 as 1
    1                   #initialize fn-1 as 1
.text
li x21, 0               #x21 for saving the current Fibonacci number fn
lw x19, 0(x0)           #load fn-2 from memory to x19
lw x20, 4(x0)           #load fn-1 from memory to x20
li x18, 3               #set the value of i to 3
addi x5, x0, 40         #set the number of Fibonacci numbers to calculate to 40

calculate:              #loop for calculating the Fibonacci numbers
blt x5, x18, done       #exit the loop if i exceeds the number of Fibonacci numbers to
calculate
add x21, x19, x20       #calculate the current Fibonacci number fn as the sum of fn-1 and
fn-2
addi x7, x18, -1        #calculate the address of fn in memory
add x6, x7, x7
```

```
add x6, x6, x6
sw x21, 0(x6)           #store the current Fibonacci number fn in memory
addi x19, x20, 0        #update fn-2 to the previous value of fn-1
addi x20, x21, 0        #update fn-1 to the current value of fn
addi x18, x18, 1        #increment i by 1 to move to the next Fibonacci number
j calculate             #jump back to the beginning of the loop to calculate the next
Fibonacci number

done:
ret                     #return from the calculate function to exit the program
```

这段代码使用递归方法计算斐波那契数列中的前40个数字，其中，x19和x20分别存储前两个斐波那契数fn-2和fn-1，x21存储当前计算的斐波那契数fn，x18存储循环变量i（数组下标），x5存储需要计算的斐波那契数的个数。在循环中，使用blt指令判断是否已经计算到所需的斐波那契数的个数，如果是则退出循环。每次循环中，根据斐波那契数的定义，计算出当前的斐波那契数fn，并将其存储到内存中。然后更新fn-2和fn-1的值，以便计算下一个斐波那契数。最后，程序使用ecall指令调用系统调用来结束程序的执行。

2. 设计32bits移位寄存器，通过PDU实现对移位寄存器数据的实时操作

PDU 的逻辑结构：



按照要求修改 `shift_reg.v` 。代码中的 `TODO` 部分是实现移位寄存器逻辑的占位符。

该代码定义了一个移位寄存器模块，其中有一个数据输入端口(`din`)，一个用于开关十六进制代码的4位输入端口(`hex`)，以及用于 `add` 、 `del` 和 `set` 信号的输入端口。该模块还有一个用于数据输出(`dout`)的输出端口。

在 `always` 块内，根据输入信号实现了移位的逻辑。如果 `rst` 信号高，则将寄存器重置为全0。如果 `set` 高，则将输入的 `din` 加载到寄存器中。如果 `add` 高，则寄存器向左移动，并将 `hex` 输入加载到最低的4位。如果 `del` 高，则寄存器向右移动，丢弃最高的4位，仅保留最低的28位。

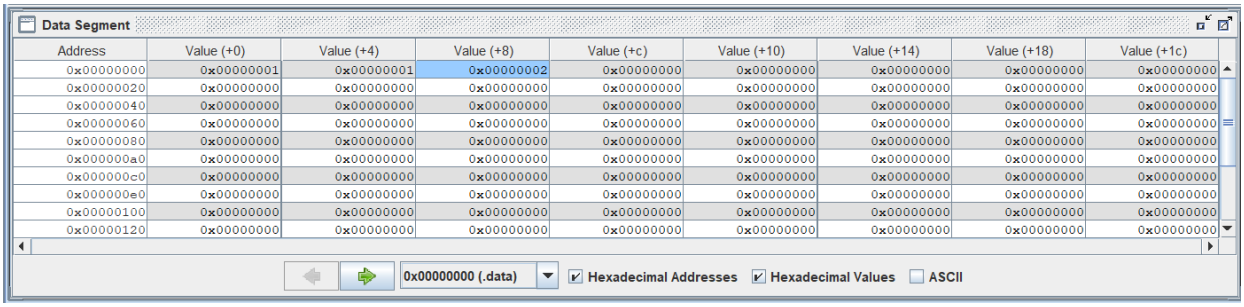
```
`timescale 1ns / 1ps
```

```
module Shift_reg(
    input rst,
    input clk,          // Work at 100MHz clock

    input [31:0] din,   // Data input
    input [3:0] hex,    // Hexadecimal code for the switches
    input add,          // Add signal
    input del,          // Delete signal
    input set,          // Set signal

    output reg [31:0] dout  // Data output
);

    // TODO
    always @ (posedge clk) begin
        if (rst)
        begin
            dout <= 32'b0;
        end
        else if (set)
        begin
            dout <= din;
        end
        else if (add)
        begin
            dout <= {dout[27:0], hex[3:0]};
        end
        else if (del)
        begin
            dout <= {4'b0, dout[31:4]};
        end
    end
endmodule
```

四 实验结果

1. 设计汇编程序：计算斐波那契-卢卡斯数列，生成 COE 文件

在 RARS 中运行，可以看到数列的前两项为 1，1，40项数列存储在地址为0x0000 开始的数据段Data Segment.



40项都计算结束后：

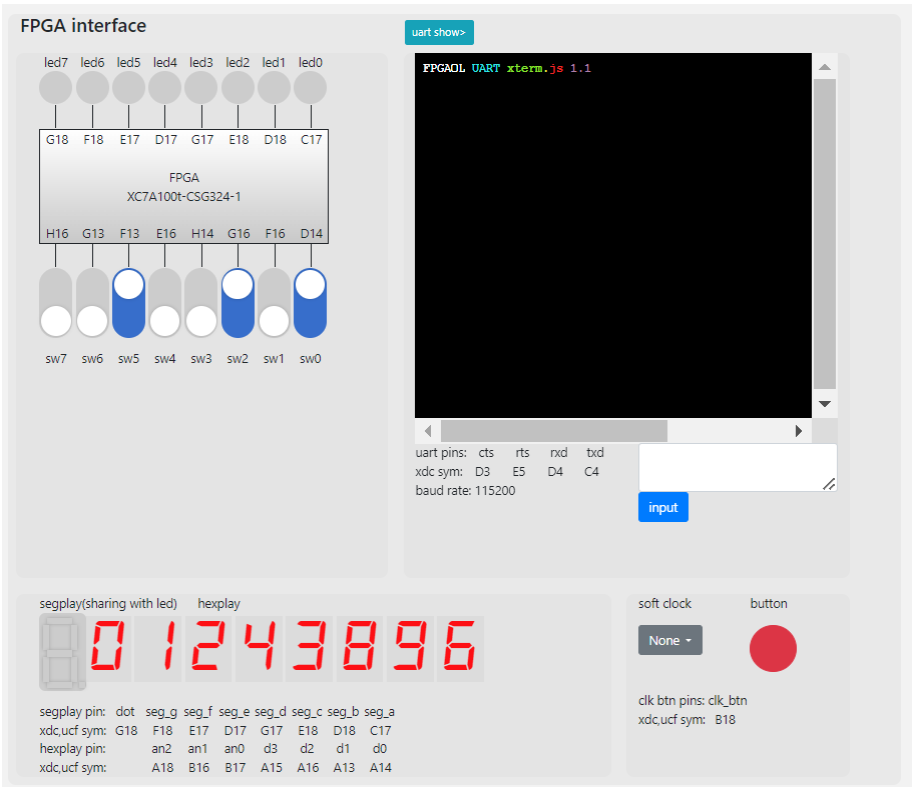
Data Segment

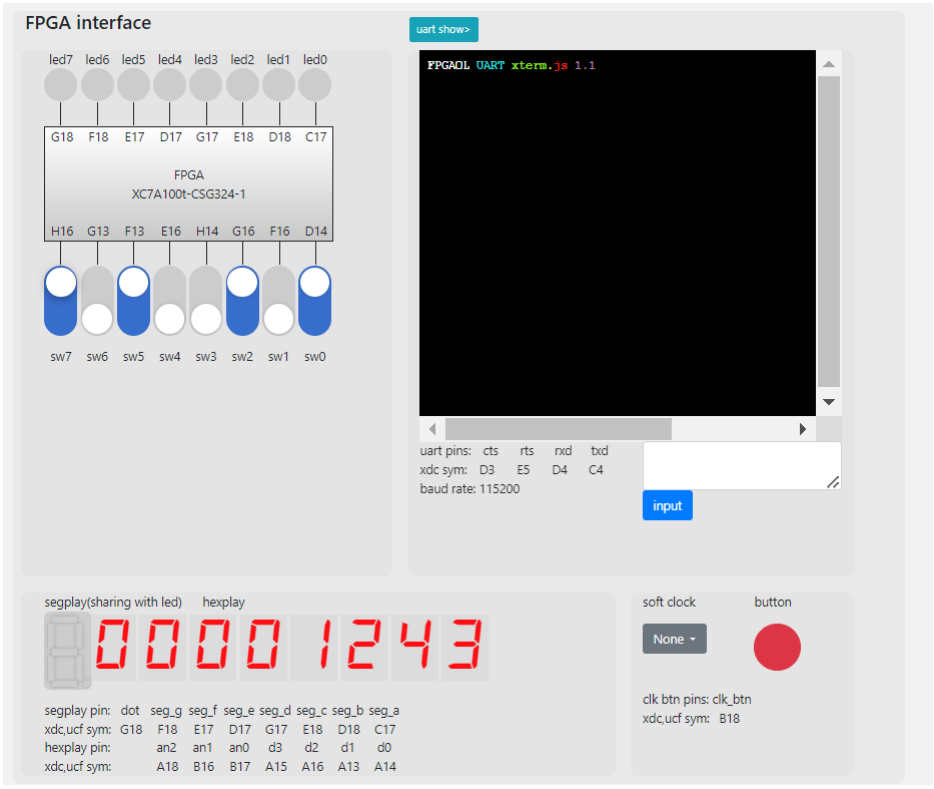
Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x00000000	0x00000001	0x00000001	0x00000002	0x00000003	0x00000005	0x00000008	0x0000000d	0x00000015
0x00000020	0x00000022	0x00000037	0x00000059	0x00000090	0x000000e9	0x00000179	0x00000262	0x000003db
0x00000040	0x0000063d	0x00000a18	0x00001055	0x00001a6d	0x00002ac2	0x0000452f	0x00006ff1	0x0000b520
0x00000060	0x00012511	0x0001da31	0x0002ff42	0x0004d973	0x0007d8b5	0x000cb228	0x00148add	0x00213d05
0x00000080	0x0035c7e2	0x005704e7	0x008cccc9	0x00e3d1b0	0x01709e79	0x02547029	0x03c50ea2	0x06197ecb
0x000000a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x000000c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x000000e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

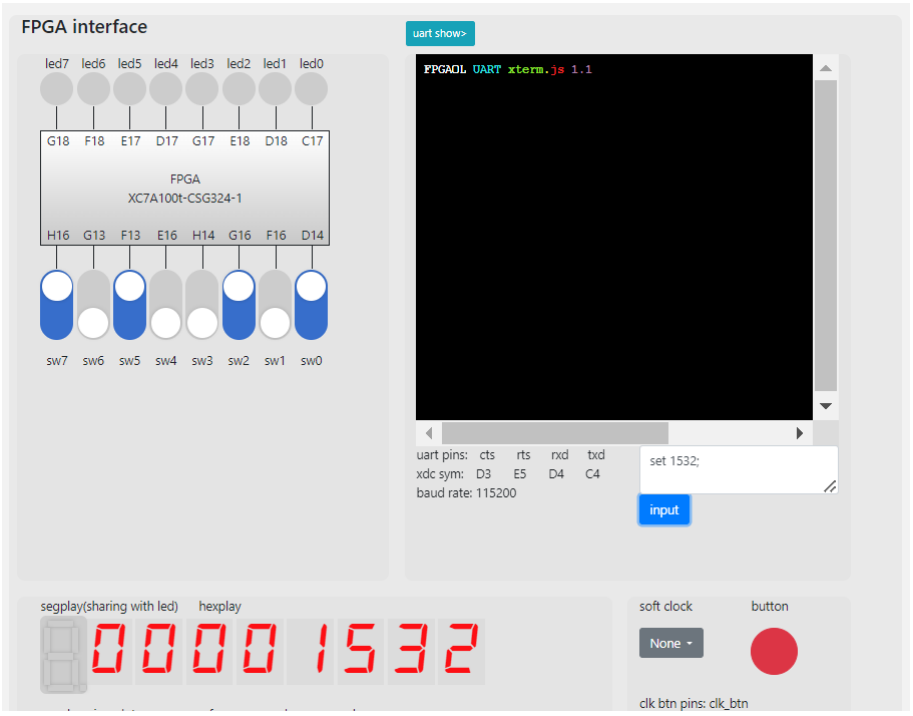
0x00000000 (.data) Hexadecimal Addresses Hexadecimal Values ASCII

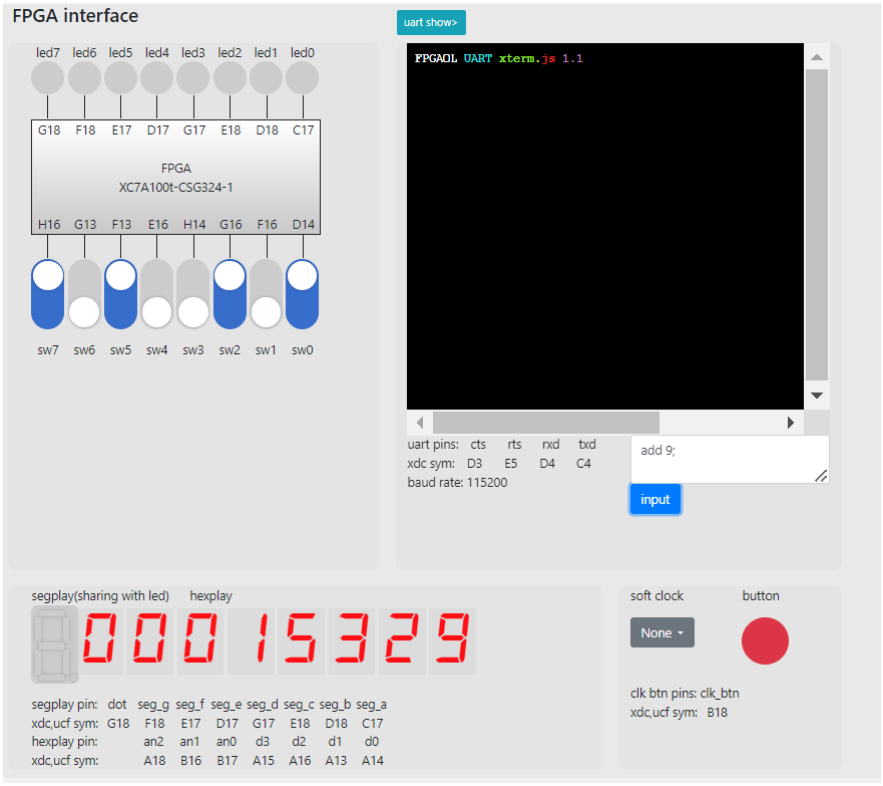
2. 设计32bits移位寄存器，通过PDU实现对移位寄存器数据的实时操作

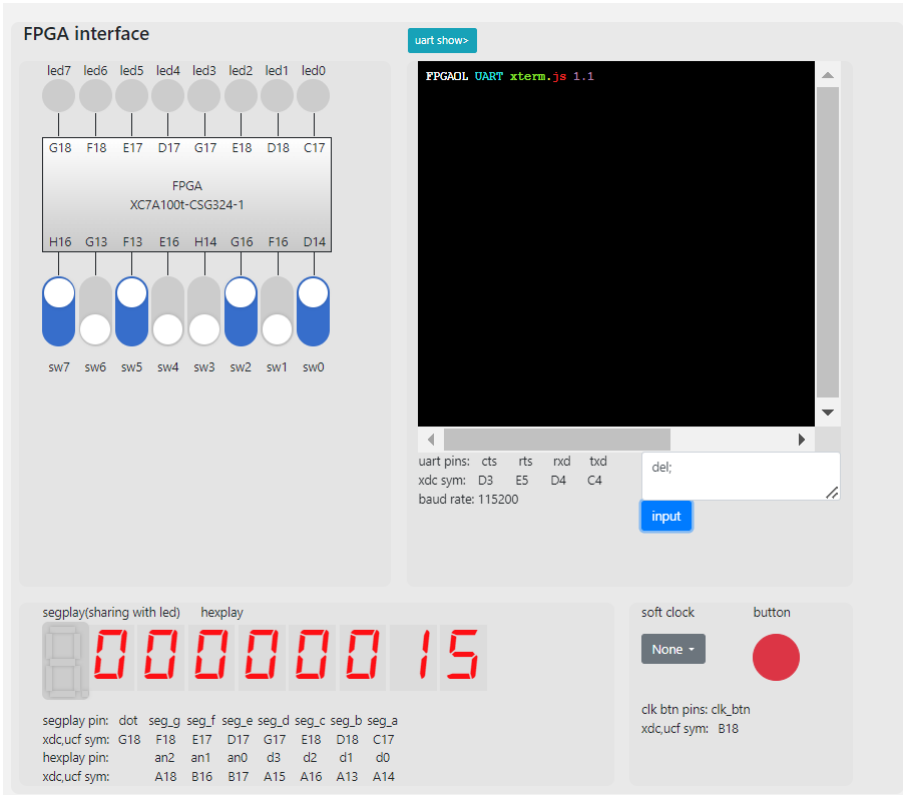
把设计文件加入 vivado，生成 bit 文件并烧写到 fpga 平台。测试结果如下：




FPGA interface
led7 led6 led5 led4 led3 led2 led1 led0
G18 F18 E17 D17 G17 E18 D18 C17
FPGA
XC7A100t-CSG324-1
H16 G13 F13 E16 H14 G16 F16 D14
sw7 sw6 sw5 sw4 sw3 sw2 sw1 sw0
uart show>
FPGAOL UART xterm.js 1.1
uart pins: cts rts rxd txd
xdc sym: D3 E5 D4 C4
baud rate: 115200
input
segplay(sharing with led)
hexplay
0000 1243
segplay pin: dot seg_g seg_f seg_e seg_d seg_c seg_b seg_a
xdc,ucf sym: G18 F18 E17 D17 G17 E18 D18 C17
hexplay pin: an2 an1 an0 d3 d2 d1 d0
xdc,ucf sym: A18 B16 B17 A15 A16 A13 A14
soft clock
button
None
clk btn pins: clk_btn
xdc,ucf sym: B18


FPGA interface
led7 led6 led5 led4 led3 led2 led1 led0
G18 F18 E17 D17 G17 E18 D18 C17
FPGA
XC7A100t-CSG324-1
H16 G13 F13 E16 H14 G16 F16 D14
sw7 sw6 sw5 sw4 sw3 sw2 sw1 sw0
uart show>
FPGAOL UART xterm.js 1.1
uart pins: cts rts rxd txd
xdc sym: D3 E5 D4 C4
baud rate: 115200
set 1532;
input
segplay(sharing with led)
hexplay
0000 1532
soft clock
button
None
clk btn pins: clk_btn





五 总结

本次实验总体难度合适，并且与课内课内理论知识结合较为紧密，做起来体验比之前两个都要好很多。有了实验文档的帮助，对一些要求理解得更加透彻了，也收获了很多课外知识。希望以后的实验能延续这种风格。