实验4 数据通路和有限状态机设计

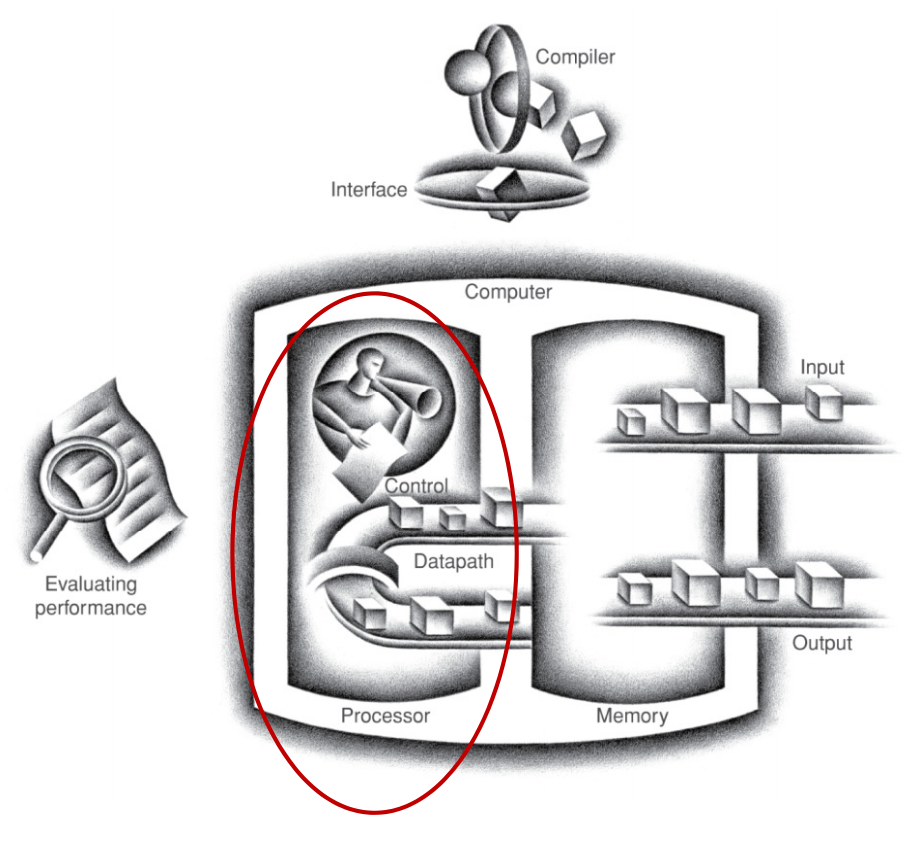
* 任务描述
* 相关知识
* 实验内容
* 遇到问题和解决方法
* 实验心得、意见和建议

## 任务描述

综合应用掌握的简单组合电路（实验2）和简单时序电路（实验3）的设计方法，完成一个数据通路的设计，并为该数据通路配上一个控制器（有限状态机），最后将所有的实验综合起来，实现一个简单的处理器（自动运算电路）。

## 相关知识

在学习完实验3（简单时序电路设计）后，实验4将尝试较为复杂的时序电路设计（比如处理器）。此类电路设计主要包含“数据通路”（Datapath）和“控制器”（Controller）两大部分，在经典计算机模型中，处理器部分如图1红框所示。其中，数据通路负责数据的操作，包括算术运算和传输数据；控制器负责数据的控制，通常以有限状态机（FSM：Finite State Machine）方式实现，包括控制流的输入、输出，以及控制数据通路中数据的传输顺序。另外，处理器旁通常会有一个“存储器”（Memory），可根据地址存取程序指令和数据。注意，数据通路自身并不能工作，只能通过控制器输出控制信号，输入到数据通路的各个单元，才能完成处理器的工作。因此，一个经典处理器通常是由数据通路和控制器组合完成的；与之对应的，本实验共包含三个步骤：数据通路（步骤1），有限状态机（步骤2），和自动运算处理器（步骤3）。



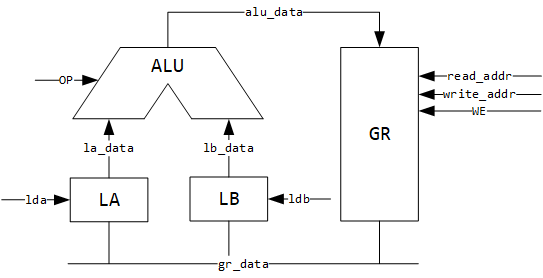
**图1处理器由数据通路和控制器构成**

## 实验内容

**1. 数据通路设计（步骤1）**

【实验样例】

图2所示处理器中，有4个逻辑单元，包括一个计算单元ALU，两个寄存器单元LA和LB，以及一个双端口存储器GR。各个单元的外部控制信号包括OP, lda, ldb, read\_addr, write\_addr, WE等，数据通路内部传输数据包括la\_data, lb\_data, gr\_data和alu\_data。



**图2某处理器的数据通路**

首先实现各个单元的模块：

**module register(clk, rst\_n, en, d, q);**

**parameter WIDTH = 8;**

**input clk, rst\_n, en;**

**input [WIDTH-1:0] d;**

**output reg [WIDTH-1:0] q;**

**always @(posedge clk) begin**

**if (!rst\_n) q <=0;**

**else if (en) q <= d;**

**end**

**endmodule**

**module alu(a, b, op, q);**

**parameter WIDTH = 8;**

**input [WIDTH-1:0] a, b;**

**input [1:0] op;**

**output reg [WIDTH-1:0] q;**

**always @(\*) begin**

**case(op)**

**2'b00: q = a + b;**

**2'b01: q = a & b;**

**2'b10: q = a ^ b;**

**2'b11: q = a | b;**

**default: q = 0;**

**endcase**

**end**

**endmodule**

**module ram(data, read\_addr, write\_addr, clk, we, q);**

**parameter DATA\_WIDTH = 8;**

**parameter ADDR\_WIDTH = 3;**

**input clk, we;**

**input [DATA\_WIDTH-1:0] data;**

**input [ADDR\_WIDTH-1:0] read\_addr, write\_addr;**

**output reg [DATA\_WIDTH-1:0] q;**

**// 申明存储器数组**

**reg [DATA\_WIDTH-1:0] ram[2\*\*ADDR\_WIDTH-1:0];**

**initial begin //对存储器初始化**

**$readmemh("ram\_init.txt", ram); end**

**always @(posedge clk) begin**

**if (we)**

**ram[write\_addr] <= data;**

**q <= ram[read\_addr];**

**end**

**endmodule**

其次利用结构描述设计方法，利用以上模块实现图2的数据通路：

**module datapath\_top(clk, rst, lda, ldb, read\_addr, write\_addr, we, op);**

**input clk, rst, lda, ldb, we;**

**input [4:0] read\_addr, write\_addr;**

**input [1:0] op;**

**wire [31:0] gr\_data, alu\_data;**

**wire [31:0] la\_data, lb\_data;**

**register #(32) LA (clk, rst, lda, gr\_data, la\_data);**

**register #(32) LB (clk, rst, ldb, gr\_data, lb\_data);**

**ram #(32, 5) GR (alu\_data, read\_addr, write\_addr, clk, we, gr\_data);**

**alu #(32) ALU (la\_data, lb\_data, op, alu\_data);**

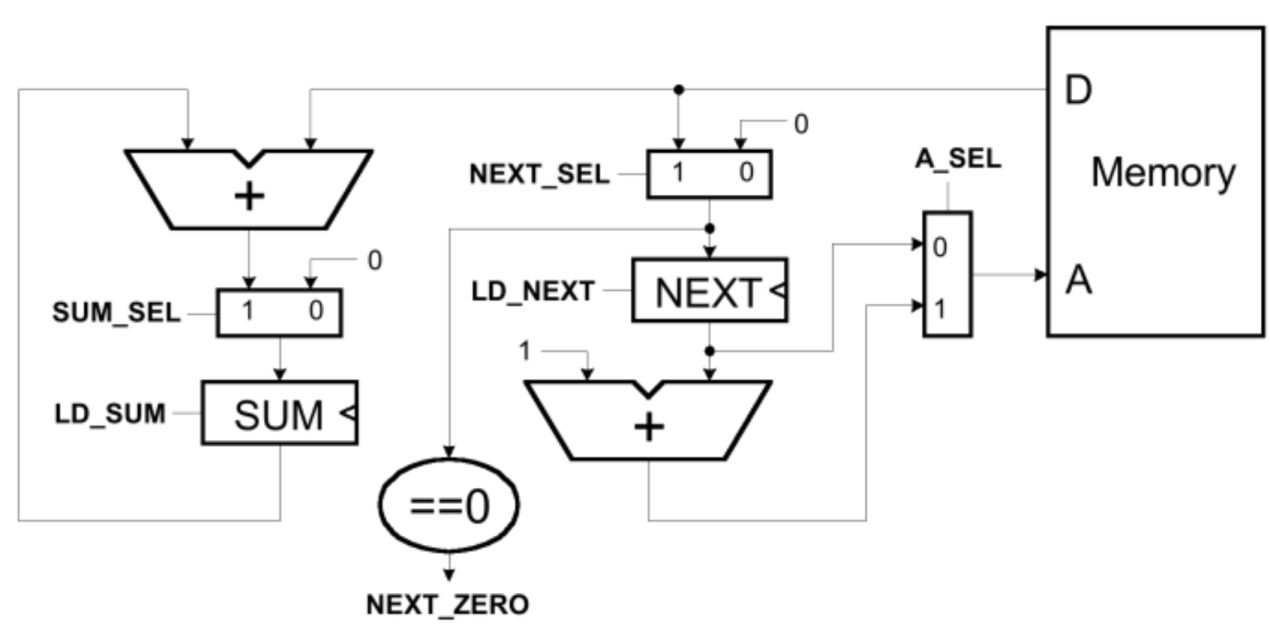
**endmodule**

注：register #(32) LA (clk, rst, lda, gr\_data, la\_data)传递参数32到LA模块，使得其WIDTH = 32。

【实验要求】

请参照实验样例，实现图3所示的数据通路。图3给出的数据通路里，SUM和NEXT是寄存器，Memory是存储器，+是加法器，==0是比较器，其它则是多路选择器。具体要求如下：

* 图中数据线的宽度和各个器件的数据线宽度初始设计时均为8位，要求构成数据通路时可以扩充至16位或者是32位；
* 设计的数据通路能够正确综合，Vivado所示的电路原理图与图3给出的一致。



**sum\_out**

**图3 数据通路图**

【实验提示】

1. 分别设计n位加法器模块，n位2选1多路选择器模块，n位比较器模块。（用parameter传参来扩展）
2. 设计一个含同步复位rst和加载load端的n位寄存器模块

当load=1时，对输入的n位数据进行同步寄存，即让输入D的值赋给输出Q。

1. 设计一个n位存储器模块，存储器中存放如下的链表（具体见图7），链表第1个节点在0号地址，各节点的第一个地址存放下一个节点的地址，各节点的第二个地址中存放着要进行求和运算的数据，当下一个节点的地址为0时，表示到达链表的结尾，求和运算结束。

00000003

00000002

00000000

00000007

00000004

00000000

00000000

0000000b

00000006

00000000

00000000

00000000

00000008

00000000

00000000

00000000

注：存储器存放该链表的过程可以如下实现：1) 将该链表存入一个文本文件；

2) 用系统函数$readmemh读该文本文件对存储器进行初始化。具体可见教材readmemh的语法。

1. 利用以上模块完成图3的数据通路模块的设计

输入端口有：时钟clk，复位rst，加载信号SUM\_SEL, NEXT\_SEL, A\_SEL, LD\_SUM, LD\_NEXT。

输出端口有: 链尾标志NEXT\_ZERO, 求和结果sum\_out。

**【实验填写】**

参照实验样例，根据实验提示完成实验要求，包括：

1. 图3各个单元模块的代码（参考课堂PPT）

选择器：

module mux(out, a, b, sel);

parameter WIDTH = 8;

output reg [WIDTH-1:0] out;

input [WIDTH-1:0] a, b;

input sel;

always @(sel or a or b)

begin

case(sel)

1'b0: out=a;

1'b1: out=b;

endcase

end

endmodule

加法器：

module add(a, b ,q);

parameter WIDTH = 8;

input [WIDTH-1:0] a, b;

output reg [WIDTH-1:0] q;

always @(\*) begin

q = a + b;

end

endmodule

比较器：

module compare\_zero(a, out);

parameter WIDTH=8;

input [WIDTH-1:0] a;

output reg out;

always @(a) begin

if(a==0)

out=1;

else out=0;

end

endmodule

寄存器：

module register(clk, rst, load, d, q);

parameter WIDTH = 8;

input clk, rst, load;

input [WIDTH-1:0] d;

output reg [WIDTH-1:0] q;

always @(posedge clk) begin

if (rst) q <=0;

else if (load) q <= d;

end

endmodule

存储器：

module memory( data, addr);

parameter data\_WIDTH=8;

parameter addr\_WIDTH=8;

input [data\_WIDTH-1:0] addr;

output reg [data\_WIDTH-1:0] data;

reg [data\_WIDTH-1:0] rom [addr\_WIDTH-1:0];

initial begin //对存储器初始化，在D盘根目录

$readmemh("D:/ram\_init.txt", rom); end

always @(addr)

data = rom[addr];

endmodule

1. 数据通路的代码：

module datapath(clk, rst, SUM\_SEL, NEXT\_SEL, A\_SEL, LD\_SUM, LD\_NEXT, NEXT\_ZERO, sum\_out);

input clk,rst,SUM\_SEL,NEXT\_SEL,A\_SEL,LD\_SUM,LD\_NEXT;

output NEXT\_ZERO;

output [31:0] sum\_out;

parameter data\_WIDTH=32;

parameter addr\_WIDTH=16;

wire [data\_WIDTH-1:0] A\_SEL\_OUT, MEMORY\_OUT, ADD1\_OUT, SUM\_SEL\_OUT, NEXT\_SEL\_OUT, NEXT\_OUT, ADD2\_OUT;

wire [data\_WIDTH-1:0] ZERO, ONE;

assign ZERO=0;

assign ONE=1;

mux #(data\_WIDTH) MYSUM\_SEL(SUM\_SEL\_OUT, ZERO, ADD1\_OUT,SUM\_SEL);

mux #(data\_WIDTH) MYNEXT\_SEL(NEXT\_SEL\_OUT, ZERO, MEMORY\_OUT, NEXT\_SEL);

mux #(data\_WIDTH) MYA\_SEL(A\_SEL\_OUT, NEXT\_OUT, ADD2\_OUT, A\_SEL);

add #(data\_WIDTH) MYADD1(sum\_out, MEMORY\_OUT, ADD1\_OUT);

add #(data\_WIDTH) MYADD2(ONE, NEXT\_OUT, ADD2\_OUT);

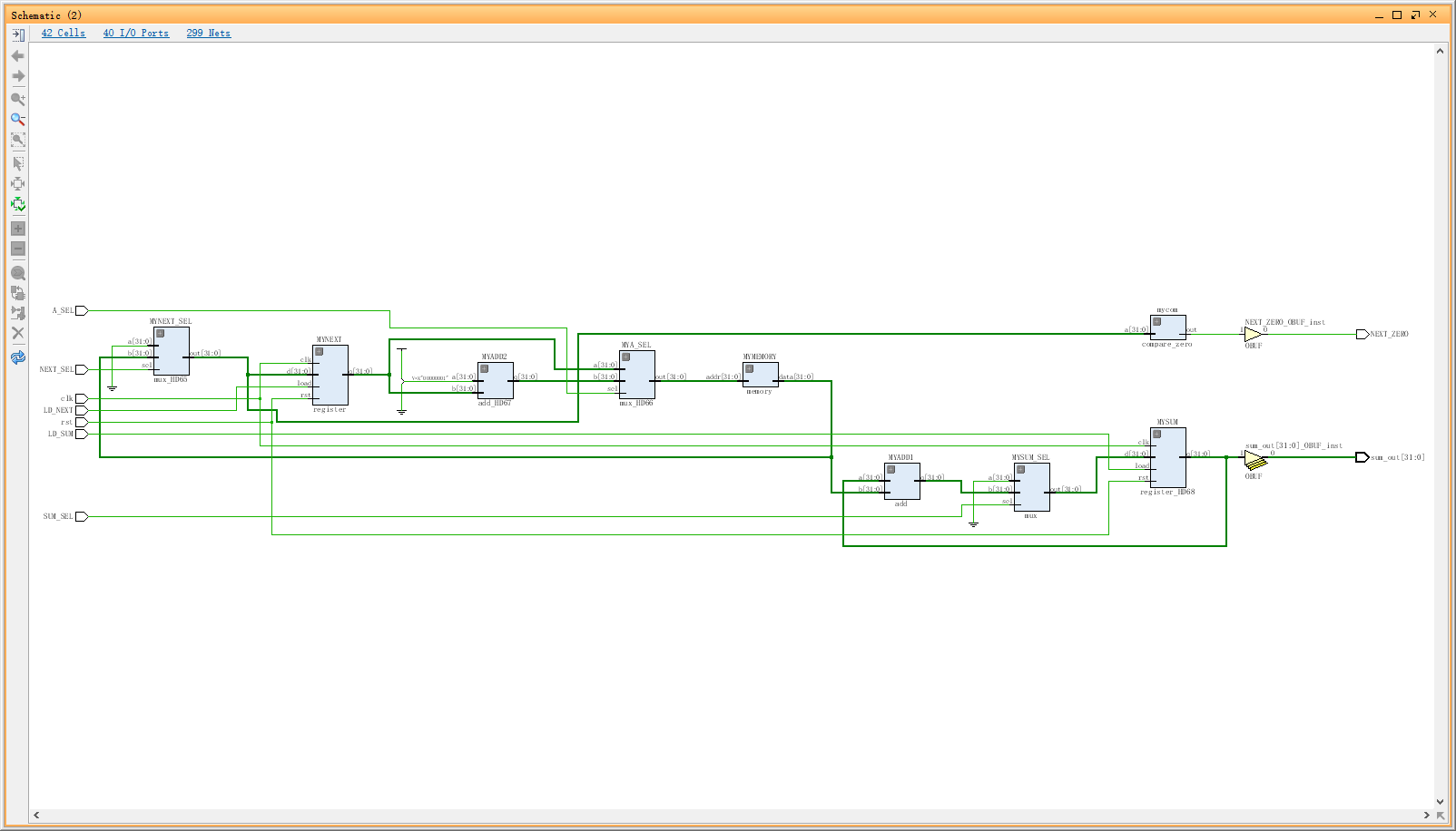
memory #(data\_WIDTH, addr\_WIDTH) MYMEMORY(MEMORY\_OUT, A\_SEL\_OUT);

register #(data\_WIDTH) MYNEXT(clk, rst, LD\_NEXT, NEXT\_SEL\_OUT, NEXT\_OUT);

register #(data\_WIDTH) MYSUM(clk, rst, LD\_SUM, SUM\_SEL\_OUT, sum\_out);

compare\_zero #(data\_WIDTH) mycom(NEXT\_SEL\_OUT, NEXT\_ZERO);

endmodule

1. 数据通路的电路原理图（生成Schematic）：

**2. 有限状态机设计（步骤2）**

【实验样例】

给定某一类激光计时器（图4），不按按钮(即B=0)，激光器关闭(即X=0)；按了按钮(即B=1)，激光器会发射3个周期(即X=1)；3个周期后激光器关闭(即X=0)。

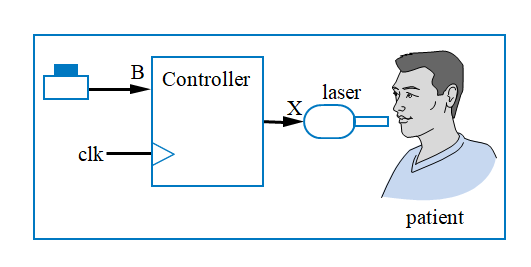


图4 激光计时器

该类激光计时器的有限状态机如图5所示，拥有Off（关闭），On1~On3（第1~3个周期激光发射）一共四个状态。每一个时钟周期都触发一次状态迁移。

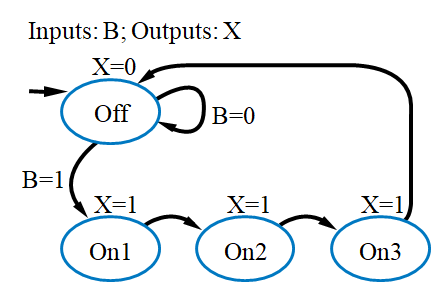


图5 该激光计时器的有限状态机

图5所示状态机的代码如下：

**module LaserTimer(B, X, Clk, Rst);**

**input B;**

**output reg X;**

**input Clk, Rst;**

**parameter S\_Off = 0, S\_On1 = 1, S\_On2 = 2, S\_On3 = 3;**//对应四个状态

**reg [1:0] State, StateNext;** //当前状态和下一个时钟周期的状态

(接上页)

**// CombLogic**

**always @(State, B) begin**

**case (State)**

**S\_Off: begin**

**X <= 0;** //初始状态Off：关闭

**if (B == 0)**

**StateNext <= S\_Off;** //不按按钮，保持关闭

**else**

**StateNext <= S\_On1;** //按了按钮，下一个状态为On1

**end**

**S\_On1: begin**

**X <= 1;** //激光发射第1个周期

**StateNext <= S\_On2;** //下一个状态自动迁移为On2

**end**

**S\_On2: begin**

**X <= 1;**  //激光发射第2个周期

**StateNext <= S\_On3;** //下一个状态自动迁移为On3

**end**

**S\_On3: begin**

**X <= 1;** //激光发射第3个周期

**StateNext <= S\_Off;** //下一个状态自动迁移为Off

**end**

**endcase**

**end**

**// StateReg**

**always @(posedge Clk) begin**

**if (Rst == 1 )**

**State <= S\_Off;** //激光器复位

**else**

**State <= StateNext;** //迁移到下一个状态

**end**

**endmodule**

该状态机的testbench代码如下：

**`timescale 1ns / 1ps**

**module laser\_timer\_tb( );**

**reg Clk\_s, Rst\_s, B\_s;**

**wire X\_s;**

**LaserTimer dut(Clk\_s, Rst\_s, B\_s, X\_s);**

**always begin // 周期为20ns的时钟**

**Clk\_s <= 0;**

**#10;**

**Clk\_s <= 1;**

**#10;**

**end**

**initial begin**

**Rst\_s <= 1;**  //复位启动

**B\_s <= 0;** //按钮未按下

**@(posedge Clk\_s);** //到达下一个时钟上升沿

**#5 if (X\_s != 0)** //延迟5ns后验证复位是否成功

**$display("%t: Reset failed", $time);**

**Rst\_s <= 0;** //复位关闭

**@(posedge Clk\_s);**

**#5 B\_s <= 1;** //按下按钮

**@(posedge Clk\_s);**

**#5 B\_s <= 0;**  //松开按钮

**if (X\_s != 1)** //验证状态On\_1

**$display("%t: First X=1 failed", $time);**

**@(posedge Clk\_s);**

**#5 if (X\_s != 1)** //验证状态On\_2

**$display("%t: Second X=1 failed", $time);**

**@(posedge Clk\_s);**

**#5 if (X\_s != 1)** //验证状态On\_3

**$display("%t: Third X=1 failed", $time);**

**@(posedge Clk\_s);**

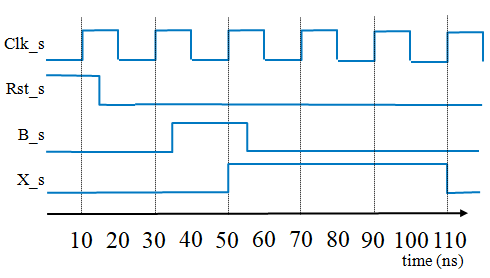
**#5 if (X\_s != 0)** //验证状态Off

**$display("%t: Final X=0 failed", $time);**

**end**

**endmodule**

仿真结果如下：



【实验要求】

假设有限状态机的状态转移图如图6所示。根据状态转移图，按照有限状态机（FSM）标准的实现模式来编写Verilog程序代码。具体要求如下：

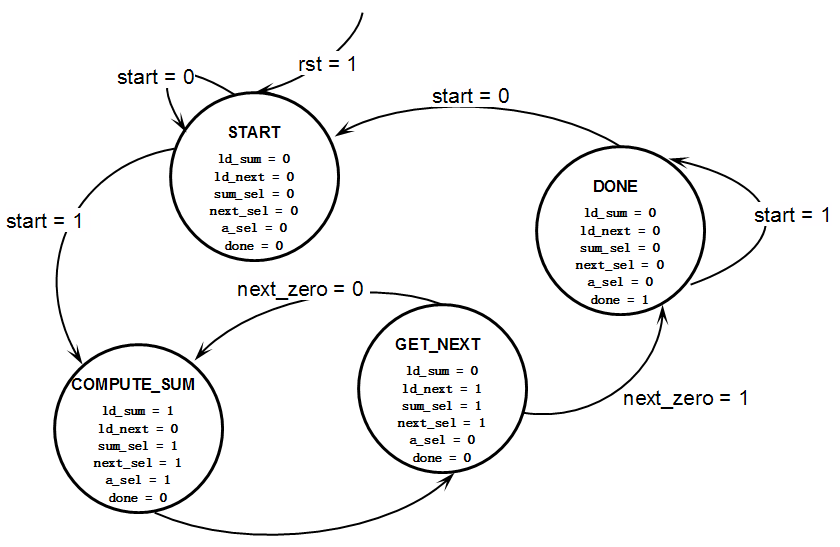
* 设计的有限状态机（FSM）能够正确综合；
* 编写有限状态机的仿真程序，完成有限状态机（FSM）的功能仿真，有限状态机功能仿真正确。

【实验提示】

该控制器模块的端口有：

输入端口：时钟clk，复位rst，启动求和start，链尾标志next\_zero

输出端口: 控制信号LD\_SUM,LD\_NEXT,SUM\_SEL,NEXT\_SEL,A\_SEL，求和结束DONE。



**图6状态转移图**

**【实验填写】**

参照实验样例，根据实验提示完成实验要求，包括：

1. 图6的Verilog程序代码

module FSM(clk,rst,start,next\_zero, LD\_SUM, LD\_NEXT, SUM\_SEL, NEXT\_SEL, A\_SEL, DONE);

input clk,rst,start,next\_zero;

output reg LD\_SUM,LD\_NEXT,SUM\_SEL,NEXT\_SEL,A\_SEL,DONE;

parameter START=0, COMPUTE\_SUM=1, GET\_NEXT=2, DONES=3;

reg [1:0] state, statenext;

always @(state, start, next\_zero)begin

case(state)

START:begin

LD\_SUM<=0;

LD\_NEXT<=0;

SUM\_SEL<=0;

NEXT\_SEL<=0;

A\_SEL<=0;

DONE<=0;

if(start==0)

statenext<=START;

else

statenext<=COMPUTE\_SUM;

end

COMPUTE\_SUM:begin

LD\_SUM<=1;

LD\_NEXT<=0;

SUM\_SEL<=1;

NEXT\_SEL<=1;

A\_SEL<=1;

DONE<=0;

statenext<=GET\_NEXT;

end

GET\_NEXT:begin

LD\_SUM<=0;

LD\_NEXT<=1;

SUM\_SEL<=1;

NEXT\_SEL<=1;

A\_SEL<=0;

DONE<=0;

if(next\_zero==0)

statenext<=COMPUTE\_SUM;

else

statenext<=DONES;

end

DONES:begin

LD\_SUM<=0;

LD\_NEXT<=0;

SUM\_SEL<=0;

NEXT\_SEL<=0;

A\_SEL<=0;

DONE<=1;

if(start==0)

statenext<=START;

else

statenext<=DONES;

end

endcase

end

always @(posedge clk) begin

if(rst==1)

state<=START;

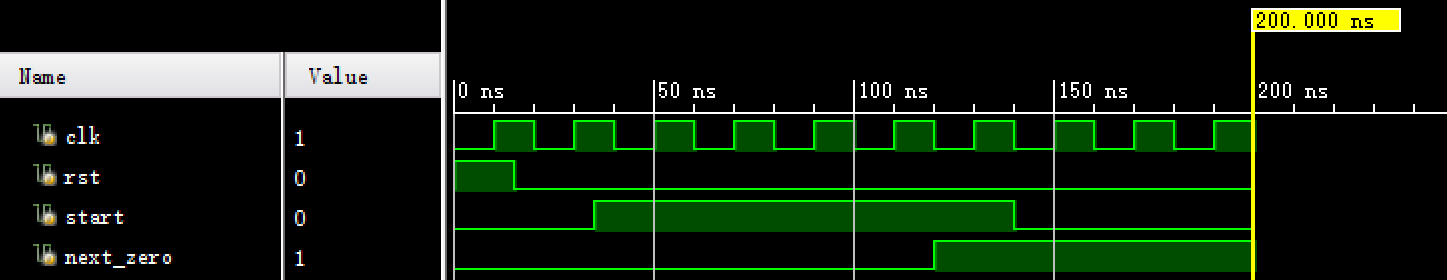
else

state<=statenext;

end

endmodule

1. 设计testbench进行仿真测试，输入信号波形如下图：



`timescale 1ns / 1ps

module FSM\_tb();

reg clk, rst, start, next\_zero;

wire LD\_SUM, LD\_NEXT, SUM\_SEL, NEXT\_SEL, A\_SEL, DONE;

FSM dut(clk,rst,start,next\_zero, LD\_SUM, LD\_NEXT, SUM\_SEL, NEXT\_SEL, A\_SEL, DONE);

always begin // 周期为20ns的时钟

clk <= 0;

#10;

clk <= 1;

#10;

end

initial begin

rst <= 1;

start <= 0;

next\_zero <= 0;

#15 rst <= 0;

#20 start <= 1;

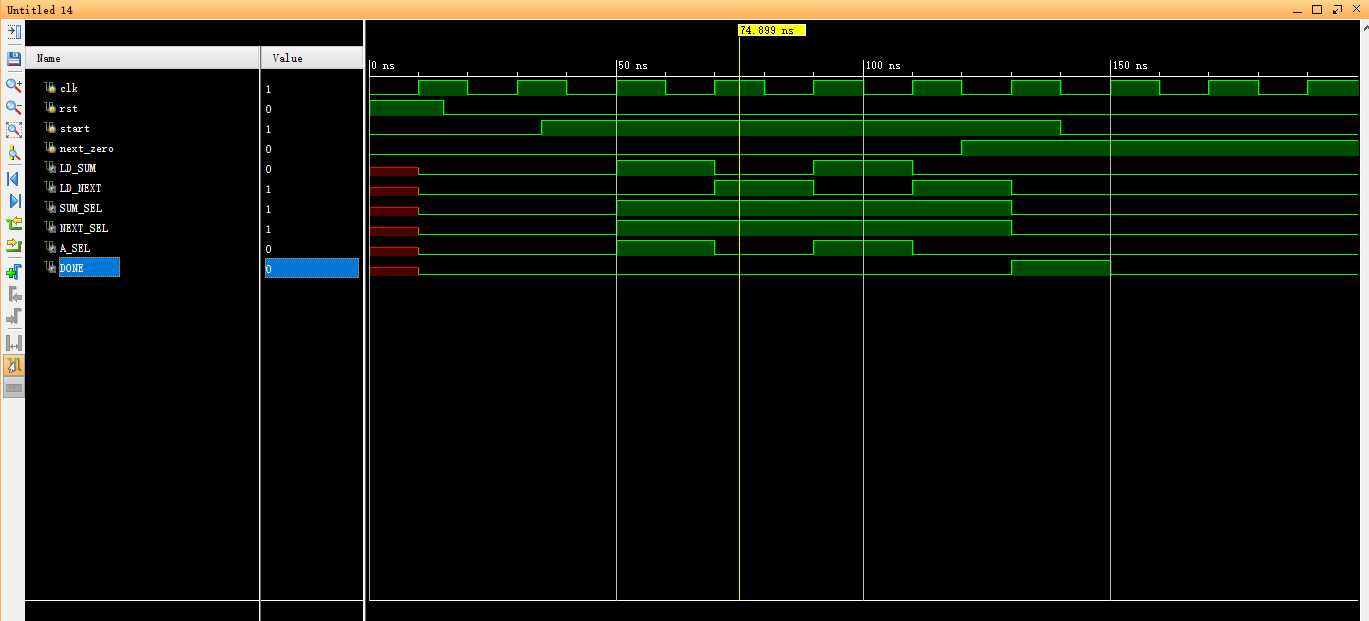
#85 next\_zero <=1;

#20 start <= 0;

end

endmodule

1. 仿真结果图：

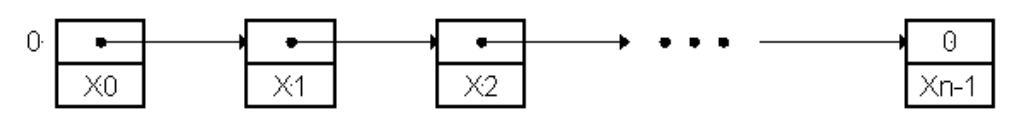
如图，50ns处由于rst为1进入compute\_sum，70ns进入get\_next，90ns由于next\_zero为0又进入compute\_sum，110ns进入get\_next，130ns由于next\_zero为1进入DONE，150ns由于start为0进入START。一切功能正常。

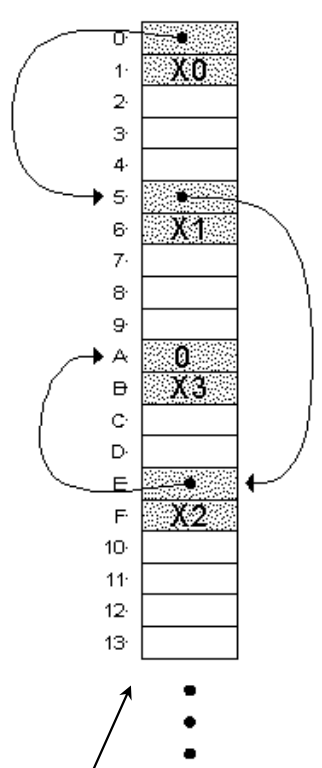
**3. 自动运算电路的设计（步骤3）**

【实验要求】

将实验步骤1实现的数据通路与实验步骤2实现的有限状态机（FSM）结合起来，可以进行以链表方式存储的数据的求和运算。

在存储器中存放的数据链表（第5页所示链表）其结构如下图7所示，链表的各个节点在存储器中不是连续存放，各节点的第一个地址存放下一个节点的地址，各节点的第二个地址中存放着要进行求和运算的数据，当下一个节点的地址为0时，表示到达链表的结尾，求和运算结束。

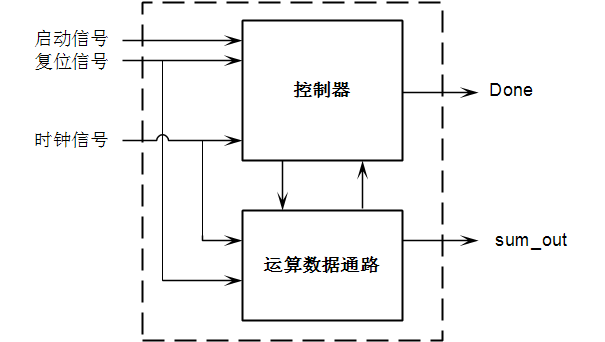




**图7 数据链表及其在存储器中的存放格式**

利用上面设计的数据通路、有限状态机，将它们集成起来，设计并实现一个能够进行上述图7所示链表数据的自动求和运算，该电路的总体框架如图8所示。具体要求如下：

* 完成自动运算求和电路的设计，能够正确综合；
* 编写仿真程序，进行功能仿真，仿真结果正确；



**图8 自动运算电路模块构成图**

存储器初始化文件（存储器每个存储单元32位，共有16个存储单元，最后的求和运算结果 = 2+4+6+8 = 20）：

00000003

00000002

00000000

00000007

00000004

00000000

00000000

0000000b

00000006

00000000

00000000

00000000

00000008

00000000

00000000

00000000

【实验提示】

可参照实验2（简单组合电路设计）中的第四步“用2选1多路选择器构造3选1多路选择器。”利用结构描述，结合步骤1和步骤2的数据通路模块和有限状态机模块，构造自动运算电路，完成图7所示的数据链表的求和运算。

该控制器模块的端口有：

输入端口：时钟clk，复位rst，启动求和start

输出端口: 求和结束DONE，求和结果sum\_out

**【实验填写】**

根据实验提示完成实验要求，包括：

1. 图8的Verilog程序代码

module auto\_add(clk,rst,start,DONE,sum\_out);

input clk,rst,start;

output DONE;

output [31:0] sum\_out;

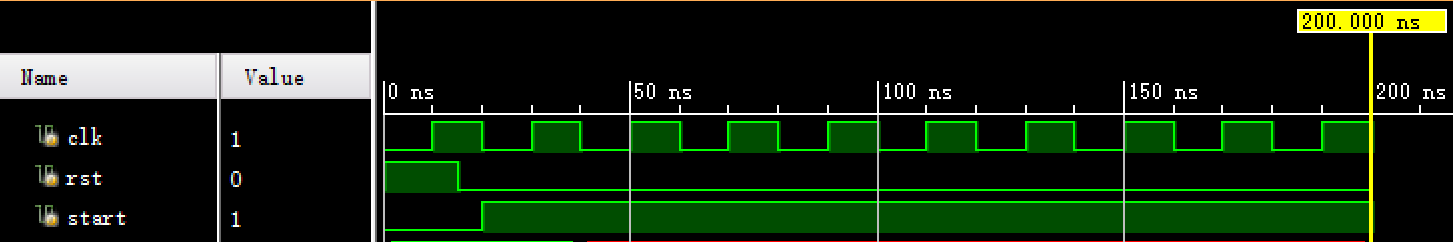
wire SUM\_SEL, NEXT\_SEL, A\_SEL, LD\_SUM, LD\_NEXT, NEXT\_ZERO;

FSM MYFSM(clk,rst,start,NEXT\_ZERO, LD\_SUM, LD\_NEXT, SUM\_SEL, NEXT\_SEL, A\_SEL, DONE);

datapath MYDATAPATH(clk, rst, SUM\_SEL, NEXT\_SEL, A\_SEL, LD\_SUM, LD\_NEXT, NEXT\_ZERO, sum\_out);

endmodule

1. 设计testbench进行仿真测试，输入信号波形如下图：



`timescale 1ns / 1ps

module auto\_add \_tb();

`timescale 1ns / 1ps

module auto\_add\_tb();

reg clk, rst, start;

wire DONE;

wire [31:0] sum\_out;

auto\_add dut(clk,rst,start,DONE,sum\_out);

always begin // 周期为20ns的时钟

clk <= 0;

#10;

clk <= 1;

#10;

end

initial begin

rst <= 1;

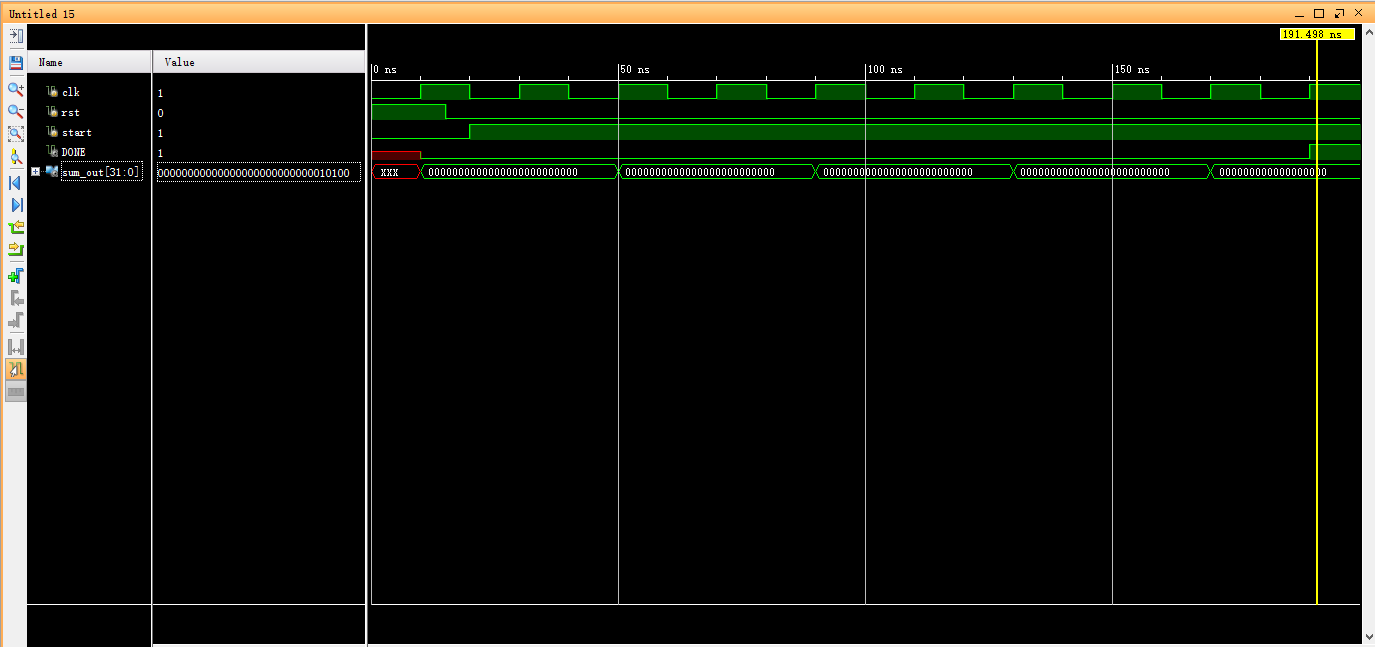
start <= 0;

#15 rst <= 0;

#5 start <= 1;

end

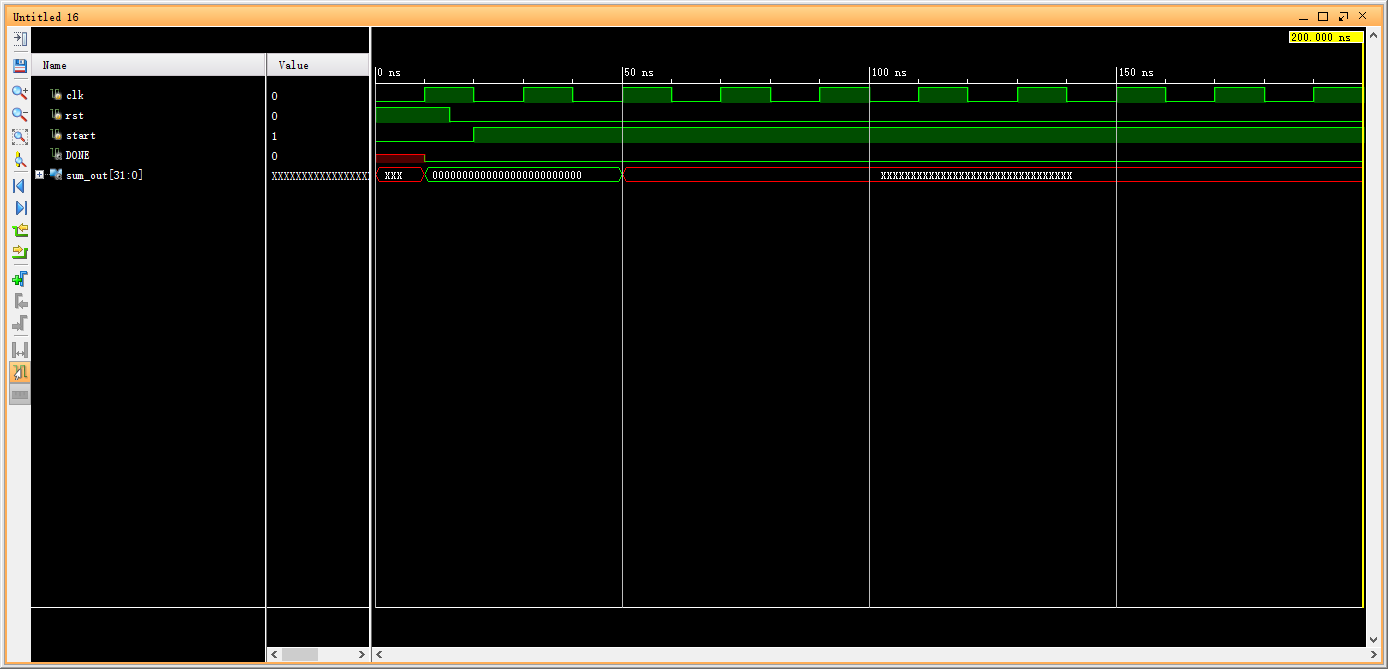
endmodule

1. 仿真结果图：

如图，最后能正确求出结果20，每次计算求和需要两个周期，这是因为需要一个周期求和，一个周期计算下一个地址。

## 遇到问题和解决方法

本次实验遇到的问题其实不多，其中最影响的结果的是我对$readmemh的使用，我由于对路径的记忆特别深刻，所以直接从D盘上面的路径复制下来，结果在最后一个实验中还是会得到如下图的失败。



这种置零后一大批的红色应该是读取的问题，最后百度了以下才发现原来路径要用反斜杠“/”，这才解决了问题。

本实验的操作难度不大，但是难在实验一数据通路中对图的原理的理解。我一开始实在没有看懂，后来看了实验二才理解，是两个周期完成一次运算，然后做实验也就会顺利一些。

## 实验心得、意见和建议

本次实验给我最大的启示就是要总体的看完实验再动手，不然会比较难懂。然后就是在学新的指令的时候不能一知半解就去实践，一定要保证正确后再用，不然会浪费更多的时间。

另外建议老师以后能把实验一和实验二的位置换一下，在没有FSM的情况下实在是很难理解数据通路图。