# 实验六 数字系统综合设计

## 实验目的

综合运用本课程所学习的知识，设计并实现复杂的数字系统。

## 实验内容

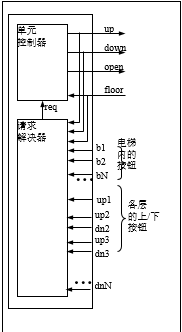
完成基础**实验6.1**；从**实验6,2、实验6.3、实验6.4中选择其中一个完成即可**。本实验评分依据系统复杂度、完成程度、系统展示效果、答辩讲述等进行综合评价。

1. **实验6.1——将实验2.3中的32位逐位进位加法器改写成流水线加法器**：流水级数自定义，完成波形仿真验证即可。
2. **实验6.2——设计电梯控制系统：**自己设计并实现状态机和整个电梯控制系统的数字逻辑电路，通过仿真和Ego1实验板验证设计的正确性。

描述：设计一个4层楼的电梯控制系统，完成电梯向上或向下移动到被请求楼层（假设电梯每移动一层需要1秒）。请求可来自每层楼的呼叫按钮，也可来自电梯内的目的楼层选择。当电梯到达被请求楼层后，打开电梯门10秒（假设该电梯内只有楼层按键，没有开门和关门按键），然后关闭电梯门前往下一个被请求楼层；如果没有请求则停在本层。电梯运行中保持电梯门关闭。当同时有多个请求时，应答的优先原则为尽可能不改变电梯运动方向且距离当前楼层最近。

板级验证参考说明（可自行设计板上资源的使用方案）：

* 用实验板上开关模拟电梯口的向上按钮。开关输入值的改变模拟按钮被按下的信号（不管是0-> 1还是1-> 0都是按下按钮）。当开关输入值改变且电梯没有停在同层，则点亮对应的led灯；当电梯停在同层时，则led不亮。
* 用类似的方法模拟电梯口的向下按钮。
* 用类似的方法模拟电梯内的按钮。
* 用1个数码管显示当前电梯所在的楼层。
* 用1个数码管显示电梯运行状态（向上、向下、停止）。
* 用1个数码管显示电梯控制器状态机的状态值（可用于调试）。
* 用1个数码管显示电梯门开关状态。



**电梯控制器模块示意图**

1. **实验6.3——简单处理器设计**

简单的处理器设计主要包括控制器、运算器和数据通路设计。

设计并实现一个简单处理器。能够实现6种指令Load、Move、Add、Sub、Mul和Show，详细的要求如下：

1. 实现8位处理器，即寄存器宽度为8位（除乘法结果寄存器外）。
2. 处理器中包含4个8位寄存器存放数据，和一个16位的寄存器存放乘法指令的结果。
3. 实现程序计数器PC(Program counter)，用于标志指令的地址（序号）。PC起始地址为0，每执行一条指令PC加1。
4. Mul指令要求实现一个模块来进行乘法的计算，在模块内部可以任意选择采用组合逻辑或是时序逻辑的方式实现乘法器。**不允许调用IP核或是采用类似于a\*b的方式直接实现。**
5. 指令长度为8位。指令利用实验板上的拨码开关进行输入，Load指令可以用拨码开关同时将指令和数据输入。并能够利用数码管显示输入的指令和数据。在执行加、减法或者乘法指令的时候，最终的执行结果要求能够通过数码管显示出来。即可以实现手动输入指令，并显示输入的指令，执行完成之后能够显示结果。
6. 指令执行的结果要求可以保存至存储模块中，并读出来进行显示。显示要求左侧四个数码管显示PC，右侧四个数码管显示指令执行结果。可以任意选择数的进制。

提示：存储模块可使用IP核或二维数组实现。

指令描述：指令长度为8位或16位，采用4位操作码，操作数地址都是2位

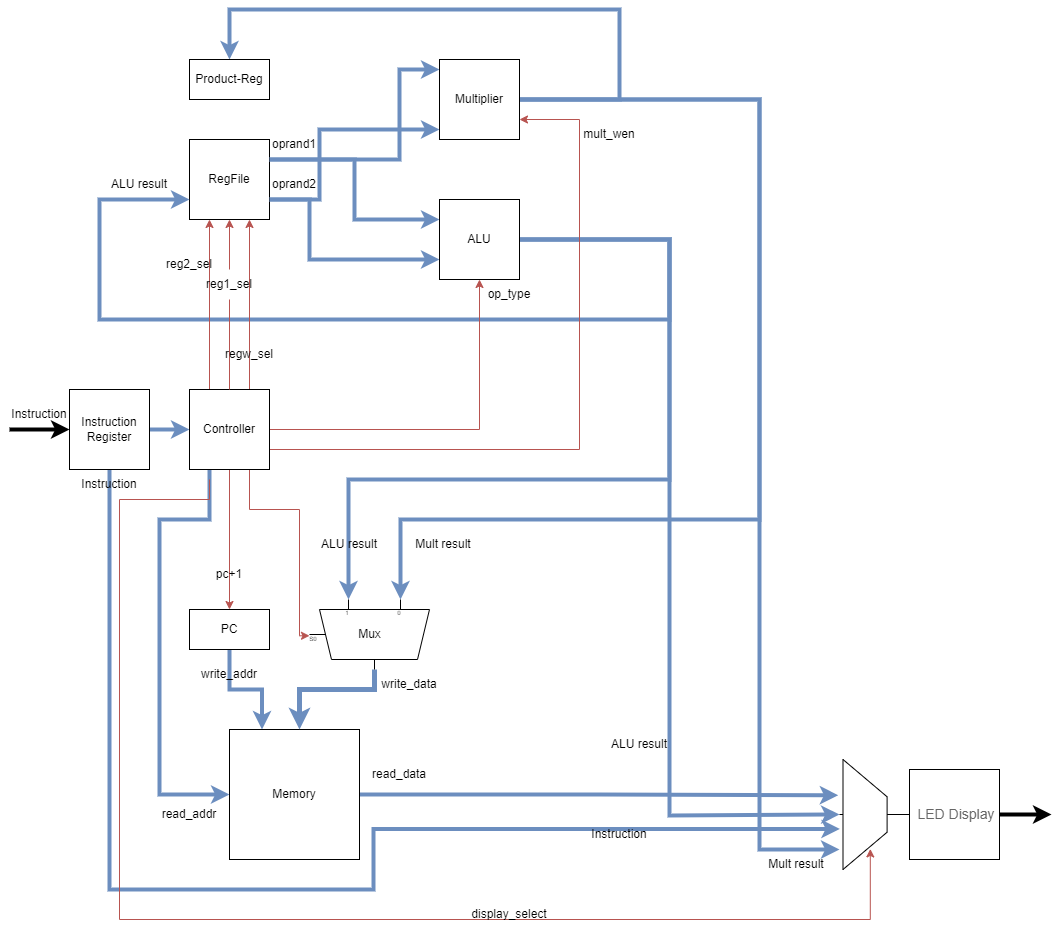
|  |  |
| --- | --- |
| 操作 | 执行的功能 |
| Load Rx, Data | RxData(数据) |
| Move Rx,Ry | Rx |
| Add Rx,Ry | Rx[Rx]+[Ry] |
| Sub Rx,Ry | Rx [Rx] – [Ry] |
| Mul Rx,Ry | P[Rx]\*[Ry] |
| Show Addr | LED {Addr,Result[Addr]} |

指令编码格式如下

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 指令名称 | 指令格式 | | | |
| 指令操作码 | 操作数1 | 操作数2 | 附加数据段 |
| Load | 0000 | Rx | 00 | Data(8位) |
| Move | 0001 | Rx | Ry | 无 |
| Add | 0010 | Rx | Ry | 无 |
| Sub | 0011 | Rx | Ry | 无 |
| Mul | 0100 | Rx | Ry | 无 |
| Show | 1111 | Addr | | 无 |

1. Load 加载指令，指令的操作码为4’b0000，操作数地址分别为Rx，Ry（不使能），要求加载数据通过外部Data信号给出。
2. Move 移动指令，指令的操作码是4’b0001，操作数地址分别为Rx，Ry，都为2位,将Ry指定的寄存器中的值移动到Rx指定的寄存器中。
3. Add 加法指令，指令的操作码是4’b0010，操作数地址分别为Rx,Ry，将Rx、Ry指定的寄存器中的数据做加法运算之后，结果保存到Rx指定的寄存器中。
4. Sub 减法指令，指令的操作码是4’b0011，操作数地址分别为Rx,Ry，将Rx、Ry指定的寄存器中的数据做减法运算之后，结果保存到Rx指定的寄存器中。
5. Mul 乘法指令，指令的操作码是4’b0100，操作数地址分别为Rx,Ry，将Rx、Ry指定的寄存器数据做乘法运算之后，结果保存至2倍数据位宽的寄存器P中。
6. Show 显示命令，指令操作码4’b1111，原有两寄存器共四位操作数的地址整体作为Addr。该指令要求能读取在PC=Addr时，程序执行的指令结果。高四位数码管显示PC，低四位数码管显示数据。show指令不需要将结果写入存储器。

实验可以自行选择实现处理器的方式，以下提供了一种可能的设计，你可以根据自己的需要来修改。图中的红色线表示控制信号，蓝色线表示数据。



1

扩展功能（以下为可选项，可以选择其中的一个或多个）

扩展指令中的指令长度为8位或16位。

* 1. 将指令存放在存储器当中（允许使用IP核），从存储器中读取指令并执行。要求用按键开关为处理器提供时钟，不显示指令内容只选择。
  2. 实现MovePH,MovePL指令，指定两个寄存器Rx,Ry作为目的寄存器，将乘积P放到Rx和Ry当中。Rx存放高位，Ry存放低位。在此基础上计算的值

|  |  |
| --- | --- |
| 操作 | 执行的功能 |
| MovePH Rx | RxP[15:8] |
| MovePL Rx | RxP[7:0] |

* 1. 实现扩展要求A，要求基础要求F中的存储模块不允许使用IP核实现，而是采用开发板提供的SRAM作为处理器的存储器。
  2. 实现扩展要求A，在此基础上实现条件跳转指令BranchEQ,指令长度为16位,其中四位是指令操作码，其值为4’b1000，操作数地址为Rx,Ry，附加数据段是一个四位的立即数Data.

|  |  |
| --- | --- |
| 操作 | 执行的功能 |
| BranchEQ Rx,Ry,Data | If([Rx]==[Ry])PCData |

附加指令的编码如下

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 指令名称 | 指令格式 | | | |
| 指令操作码 | 操作数1 | 操作数2 | 附加数据段 |
| Load | 0000 | Rx | 00 | Data |
| Move | 0001 | Rx | Ry | 无 |
| Add | 0010 | Rx | Ry | 无 |
| Sub | 0011 | Rx | Ry | 无 |
| Mul | 0100 | Rx | Ry | 无 |
| MovePH | 0101 | Rx | 00 | 无 |
| MovePL | 0110 | Rx | 00 | 无 |
| BranchEQ | 1000 | Rx | Ry | Data |
| Show | 1111 | Addr | |  |

1. **实验6.4——开放设计**

我们已经学习了许多模块电路，例如数据选择器、解码器、触发器、寄存器等等，也使用了Ego1平台上的一部分硬件资源，比如LED灯、拨码开关、数码管、SRAM等等。请应用前面学习过的各种数字逻辑设计知识，并结合Ego1的平台资源（音频接口、VGA接口、UART接口、蓝牙接口、通用I/O接口等等，可参考用户手册），自行搭建一个较为完整的应用示范，更好的掌握复杂数字系统逻辑设计要点。

## 实验要求

1. 在实验报告中尽量包含：系统设计说明及图、RTL模块图及必要说明、状态机设计（如果有的话）、测试方案及仿真验证、板级测试结果等。
2. 提交实验报告和实验的完整工程。
3. **本实验需要**找助教或老师**验收**。
4. 所完成的实验可参考其它已有设计，但不可照搬，需有改进，**鼓励原创**。

## VGA显示示例（仅供参考）

VGA（Video Graphics Array）视频图形阵列是IBM于1987年提出的一个使用模拟信号的电脑显示标准。VGA接口即电脑采用VGA标准输出数据的专用接口。VGA接口共有15针，分成3排，每排5个孔，显卡上应用最为广泛的接口类型，绝大多数显卡都带有此种接口。它传输红、绿、蓝模拟信号以及同步信号(水平和垂直信号)。

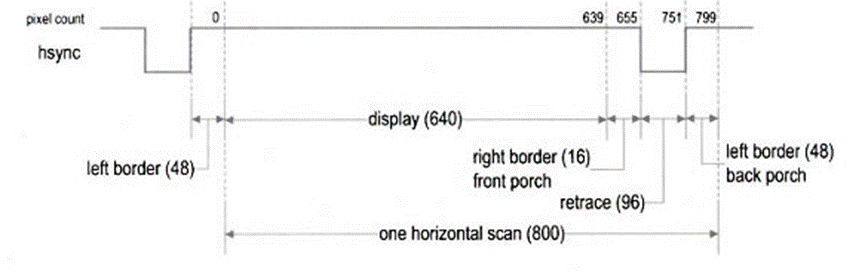
使用Verilog HDL语言对VGA进行控制一般只需控制行扫描信号、列扫描信号和红绿蓝三色信号输出即可。

VGA输出可分为四个模块：时钟分频模块、数据组织模块、接口控制模块和顶层模块。以下进行分块描述。

时钟模块分频模块对FPGA系统时钟进行分频。由于使用的显示屏参数为640\*480\*60Hz，其真实屏幕大小为800\*525，因此所需时钟频率为800\*525\*60Hz=25.175MHz，可近似处理为25MHz。FPGA系统时钟为100M，因此将其四分频即可基本满足显示要求。

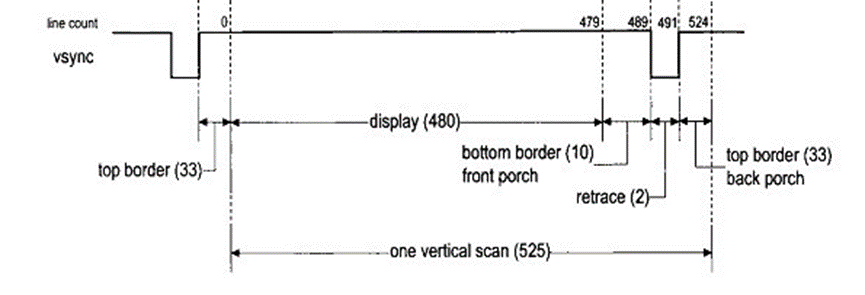
数据组织模块是将预备输出的数据组织为可以通过VGA接口控制的数据形式，本次设计中因接口已经协调，数据可不经过此模块进行组织，故可忽略该模块。

接口控制模块通过VGA接口对显示屏进行控制。VGA的扫描顺序是从左到右，从上到下。例如在640X480的显示模式下，从显示器的左上角开始往右扫描，直到640个像素扫完，再回到最左边，开始第二行的扫描，如此往复，到第480行扫完时即完成一帧图像的显示。这时又回到左上角，开始下一帧图像的扫描。如果每秒能完成60帧，则称屏幕刷新频率为60Hz。宏观上，一帧屏幕由480个行和640个列填充而成，而实际上，一帧屏幕除了显示区，还包含其他未显示部分，作为边框或者用来同步。具体而言，一个完整的行同步信号包含了左边框、显示区、右边框还有返回区四个部分，总共800个像素，其分配如下：



VGA行扫描时序

同样的，一个完整的垂直同步信号也分为四个区域，总共525个像素，分配如下：



VGA场扫描时序

模块通过组织输出行扫描信号、列扫描信号和三原色信号对显示屏实现控制。

|  |
| --- |
| 示例代码：  module VGA(  clk\_n,  rst,  hsync\_r,  vsync\_r,  OutRed,  OutGreen,  OutBlue,  num  );  input clk\_n;  input rst;  input [199:0] num;  output reg hsync\_r;  output reg vsync\_r;  output[3:0] OutRed;  output[3:0] OutGreen;  output[3:0] OutBlue;  wire [9:0] R [19:0];  assign R[0] = num[9:0];  assign R[1] = num[19:10];  assign R[2] = num[29:20];  assign R[3] = num[39:30];  assign R[4] = num[49:40];  assign R[5] = num[59:50];  ……  End module  VGA时序控制:  reg[9:0]xsync,ysync;  always @(posedge clk\_n or posedge rst) begin  if (rst) begin  xsync <= 10'd0;  end  else if (xsync == 10'd799) begin  xsync <= 10'd0;  end  else begin  xsync <= xsync + 1;  end  end  always @(posedge clk\_n or posedge rst) begin  if (rst) begin  ysync <= 10'd0;  end  else if (ysync == 10'd524) begin  ysync <= 10'd0;  end  else if (xsync == 10'd799) begin  ysync <= ysync + 1;  end  end    ……    wire valid;  assign valid = (xsync > 143) && (xsync < 784) && (ysync > 34) && (ysync < 515);  将VGA显示屏按X，Y轴方向划分成20\*10个方块  wire [9:0]x\_pos, y\_pos;  assign x\_pos = xsync - 143;  assign y\_pos = ysync -34;    wire [9:0] x;  wire [19:0] y;  assign x[0] = (x\_pos >= 201) && (x\_pos <= 224);  assign x[1] = (x\_pos >= 225) && (x\_pos <= 248);  assign x[2] = (x\_pos >= 249) && (x\_pos <= 272);  assign x[3] = (x\_pos >= 273) && (x\_pos <= 296);  ……  assign y[0] = (y\_pos >= 1) && (y\_pos <= 24);  assign y[1] = (y\_pos >= 25) && (y\_pos <= 48);  assign y[2] = (y\_pos >= 49) && (y\_pos <= 72);  assign y[3] = (y\_pos >= 73) && (y\_pos <= 96);  assign y[4] = (y\_pos >= 97) && (y\_pos <=120);  assign y[5] = (y\_pos >= 121) && (y\_pos <=144);  ……  parameter high = 12'b1111\_1111\_1111;  reg [11:0] vga\_rgb;  always @(posedge clk\_n or posedge rst) begin  if (rst) begin  vga\_rgb <= 0;  end  else if (valid)  begin  if (x\_pos>=201 && x\_pos<=440)  if (x[0]&y[0]&R[0][0])  vga\_rgb <= high;  else if (x[1] & y[0] &R[0][1])  vga\_rgb <= high;  else if (x[2] & y[0] &R[0][2])  vga\_rgb <= high;  else if (x[3] & y[0] &R[0][3])  vga\_rgb <= high;  ……  else  vga\_rgb <= 12'b0000\_0000\_1111;  else  vga\_rgb <= 12'b0000\_0000\_0000;  end  else  begin  vga\_rgb <= 0;  end  end  然后输出RGB值。  assign OutRed = vga\_rgb[11:8];  assign OutGreen = vga\_rgb[7:4];  assign OutBlue = vga\_rgb[3:0]; |