# 课 程 设 计 说 明 书

课程设计名称 数字电路与逻辑设计

专　　业 国际教育学院

班　　级 183609

学 生 姓 名 高一钧

指 导 教 师 刘丽伟

2020 年 1 月 6 日

# 课 程 设 计 任 务 书

| 课程设计题目 | 基于 Verilog HDL 的数字电路设计与验证 | | |
|---|---|---|---|
| 起止日期 | 2020 年 1 月 6 日—2020 年 1 月 10 日 | 设计地点 | 北湖主楼 11110 |

设计任务及日程安排：

设计任务：

1. 加法器的模块设计与验证/乘法器的模块设计与验证
2. 8 位数值比较器模块设计与验证
3. 编码器的模块设计与验证/译码器的模块设计与验证
4. 异步 JK 触发器的模块设计与验证/同步 JK 触发器的模块设计与验证
5. 模 60 的 BCD 码加法计数器设计与验证/二进制加减计数器的设计与验证

日程安排：

2020.1.6　　完成设计任务 1-2 模块设计

2020.1.7　　完成设计任务 3 模块设计

2020.1.8　　完成设计任务 4 模块设计

2020.1.9　　完成设计任务 5 模块设计

2020.1.10　　整理、撰写设计报告书，测试答辩

注：此任务书由指导教师在课程设计前填写，发给学生座位本门课程设计的依据。

# 基于 Verilog HDL 的数字电路设计与验证

## 实验一： 加法器的模块设计与验证

### 实验要求

1. 用 Verilog HDL 语言在 Quartus Ⅱ 编译环境下描写出简单的 8 位加法器及其测试程序；
2. 用测试程序在 Modelsim 仿真环境中对 8 位加法器进行波形仿真测试；画出仿真波形；
3. 总结实验步骤和实验结果

### 模块接口格式

输入为两组 8bit 数据 iA，iB，输出为 9bit 相加结果 oAdd

```
module adder
    (    input [7:0] iA, iB,
         output [8:0] oAdd
    );
```

### 模块程序：

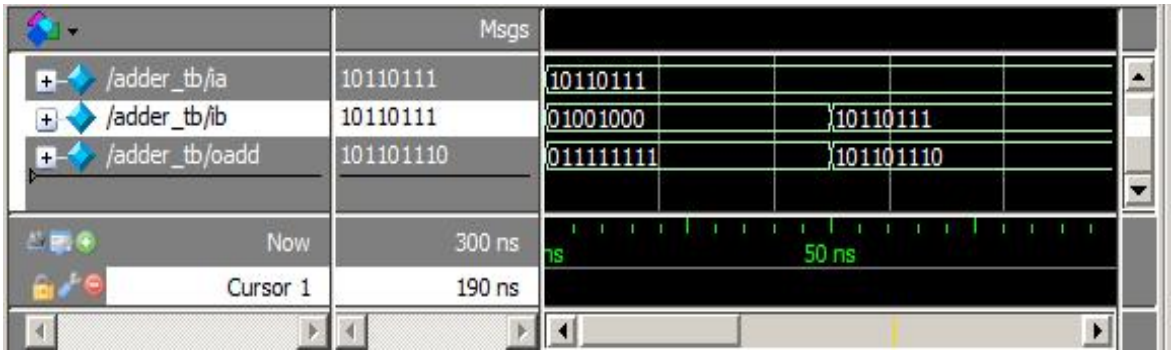
```
module adder(iA,iB,oAdd);
    input [7:0] iA,iB;
    output [8:0] oAdd;
    assign oAdd = iA + iB;
endmodule
```

### 测试程序：

```
 `timescale 1ns/1ns
 module adder_tb;

 reg [ 7: 0] ia=8'b1011_0111;
 reg [ 7: 0] ib=8'b0100_1000;
 wire [ 8: 0] oadd;

 initial begin
 #50 ia=8'b1011_0111;
      ib=8'b1011_0111;
 #50 $stop;
 end

adder adder_inst
(
```

```
 .iA(ia),
 .iB(ib),
 .oAdd(oadd)
 );

endmodule
```

## 仿真波形：




# 实验二： 8 位数值比较器模块设计与验证

## 实验要求

1. 用 Verilog HDL 语言在 Quartus II 编译环境下描写出简单的 8 位数据比较器及其测试程序；
2. 用测试程序在 Modelsim 仿真环境中对 8 位比较器进行波形仿真测试；画出仿真波形；
3. 总结实验步骤和实验结果

## 模块接口格式

模块名 compare；
输入为两组待比较的 8bit 数据 iA,iB；
输出为 3 个 1bit 的数据：oL, oG, oM
oL=1 表示 iA>iB，oG=1 表示 iA=iB，oM=1 表示 iA<iB

## 模块程序：

```
`timescale 1ns / 1ns
module comparison(iA,iB,oL,oG,oM);
    input [7:0]iA;
    input [7:0]iB;
```

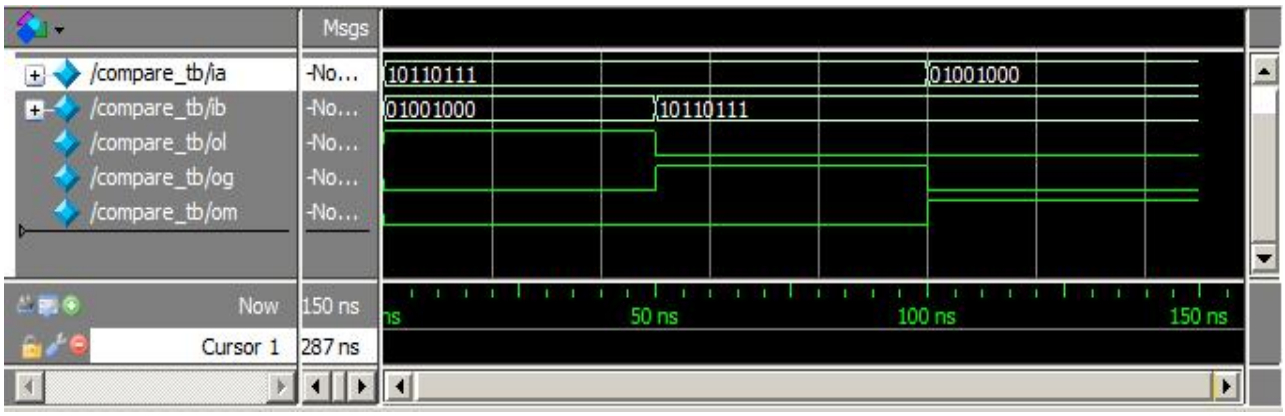
```
    output reg oL;
    output reg oG;
    output reg oM;

always@(iA or iB)
if(iA>iB)
    begin
        oL = 1;
        oG = 0;
        oM = 0;
    end
else if(iA<iB)
    begin
        oL = 0;
        oG = 0;
        oM = 1;
    end
else
    begin
        oL = 0;
        oG = 1;
        oM = 0;
    end

endmodule
```

## 测试程序：

```
`timescale 1ns / 1ns
module comparison_test;
    reg[7:0] iA;
    reg[7:0] iB;
    wire oL;
    wire oG;
    wire oM;

comparison uut(
    .iA(iA),
    .iB(iB),
```

```
    .oL(oL),
    .oG(oG),
    .oM(oM)
);

initial
    begin
        iA = 8'b1010_1010;
        iB = 8'b1011_0011;
        #50 iA = 8'b1110_0001;
            iB = 8'b0001_1110;
    end
endmodule
```

## 仿真波形：




# 实验三 ：编码器的模块设计与验证

## 实验要求

1. 设计一个类似 74x148 的 8 线-3 线优先编码器，编码器输入输出均为低电平有效，其真值表如下：

表 1 类似 74x148 的 8 线-3 线优先编码器真值表

| 输入 | | | | | | | | | 输出 | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\overline{ST}$ | $\bar{I}_7$ | $\bar{I}_6$ | $\bar{I}_5$ | $\bar{I}_4$ | $\bar{I}_3$ | $\bar{I}_2$ | $\bar{I}_1$ | $\bar{I}_0$ | $\bar{Y}_2$ | $\bar{Y}_1$ | $\bar{Y}_0$ | $\bar{Y}_{EX}$ | $Y_S$ |
| 1 | × | × | × | × | × | × | × | × | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 0 | 0 | × | × | × | × | × | × | × | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | × | × | × | × | × | × | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 | × | × | × | × | × | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 | × | × | × | × | 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 | 0 | × | × | × | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 0 | × | × | 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | × | 1 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 |

2. 用 Verilog HDL 语言在 Quartus Ⅱ 编译环境下编写此 8 线-3 线优先编码器的模块及其测试程序；
3. 用测试程序在 Modelsim 仿真环境中对上述编码器进行波形仿真测试；画出仿真波形；
4. 总结实验步骤和实验结果

## 模块接口格式

模块名 encoder；
iI_L：低电平有效的 8 位编码器数据输入
iST_L：选通输入端
oY_L：低电平有效的 3 位编码输出
oYex_L：优先扩展输出端
oYs：选通输出端

## 模块程序：

```
module decoder3_8 (iI_L, iST_L, oYex_L, oYs, oY_L);
    input [7:0] iI_L;
    input iST_L;
    output [2:0] oY_L;
    output oYex_L;
    output oYs;
    reg [2:0] oY_L;
    reg oYex_L, oYs;
    always @(iI_L or iST_L)
        if(iST_L)
            begin
                oY_L <= 3'b111;
```

```
                oYex_L <= 1;
                oYs <= 1;
            end
        else if (iI_L[7] == 0)
            begin
                oY_L <= 3'b000;
                oYex_L <= 0;
                oYs <= 1;
            end
        else if (iI_L[6] == 0)
            begin
                oY_L <= 3'b001;
                oYex_L <= 0;
                oYs <= 1;
            end
        else if (iI_L[5] == 0)
            begin
                oY_L <= 3'b010;
                oYex_L <= 0;
                oYs <= 1;
            end
        else if (iI_L[4] == 0)
            begin
                oY_L <= 3'b011;
                oYex_L <= 0;
                oYs <= 1;
            end
        else if (iI_L[3] == 0)
            begin
                oY_L <= 3'b100;
                oYex_L <= 0;
                oYs <= 1;
            end
        else if (iI_L[2] == 0)
            begin
                oY_L <= 3'b101;
                oYex_L <= 0;
                oYs <= 1;
            end
        else if (iI_L[1] == 0)
            begin
                oY_L <= 3'b110;
                oYex_L <= 0;
                oYs <= 1;
```

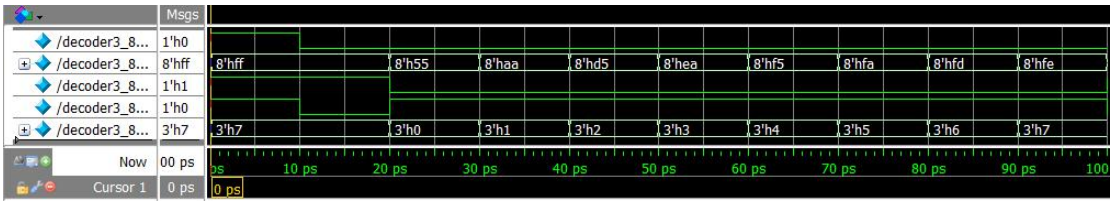
```
                end
        else if (iI_L[0] == 0)
                begin
                    oY_L <= 3'b111;
                    oYex_L <= 0;
                    oYs <= 1;
                end
        else if (iI_L == 8'b11111111)
                begin
                    oY_L <= 3'b111;
                    oYex_L <= 1;
                    oYs <= 0;
                end
        else
                begin
                    oY_L <= 3'b111;
                    oYex_L <= 1;
                    oYs <= 1;
                end
endmodule
```

## 测试程序：

```
`timescale 1 ps/ 1 ps
module decoder3_8_tb;
    reg iST_L;
    reg [7:0] iI_L;
    wire oYexT_L;
    wire oYs;
    wire [2:0] oYL;

    decoder3_8 i(
        .iST_L(iST_L),
        .oYexT_L(oYexT_L),
        .oYs(oYs),
        .iI_L(iI_L),
        .oYL(oYL)
        );

    initial
        begin
            iST_L = 1;
            iI_L = 8'b11111111;
            #10 iST_L = 0;
            #10 iI_L = 8'b01010101;
```

```
        #10 il_L = 8'b10101010;
        #10 il_L = 8'b11010101;
        #10 il_L = 8'b11101010;
        #10 il_L = 8'b11110101;
        #10 il_L = 8'b11111010;
        #10 il_L = 8'b11111101;
        #10 il_L = 8'b11111110;
        #10 il_L = 8'b11111111;
    end

endmodule
```

## 仿真波形：




# 实验四： 异步 JK 触发器的模块设计与验证

## 实验要求

1. 用 Verilog HDL 语言在 Quartus Ⅱ编译环境下分别描写出带异步清 0、置 1 端（低电平有效）的 JK 触发器及各自测试程序；

2. 用测试程序在 Modelsim 仿真环境中对 JK 触发器进行波形仿真测试；画出仿真波形；

3. 总结实验步骤和实验结果

## 模块接口格式

带异步清 0、置 1 端的 JK 触发器：

模块名： JK_FF1；

输入： clk：时钟信号；

set：异步置 1；

reset：异步清 0；

j：触发器输入；

k：触发器输入；

q：触发器输出；

## 模块程序：

```
module JK_FF2(clk, j, k, q);
```

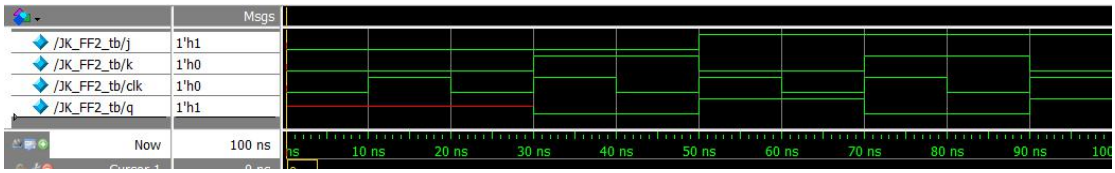
```
input clk, j, k;
output q;
reg q;
wire qb;
always@(posedge clk)
begin
    case({j,k})
    2'b00: q <= q;
    2'b01: q <= 1'b0;
    2'b10: q <= 1'b1;
    2'b11: q <= ~q;
    default: q <= q;
    endcase
end

assign qb = ~q;

endmodule
```

## 测试程序：

```
`timescale 1ns/1ns
module JK_FF2_tb;

reg j,k,clk;
wire q;

always
begin
    #10 clk = ~clk;
end
```

```
initial
begin
    clk = 0;
    j =   1'b0;
    k = 1'b0;
    #30 j = 1'b0; k = 1'b1;
    #20 j = 1'b1; k = 1'b0;
    #20 j = 1'b1; k = 1'b1;
    #20 j = 1'b1; k = 1'b0;
end

JK_FF2 i(
.j(j),
.k(k),
.clk(clk),
.q(q)
);
endmodule
```

## 仿真波形：




# 实验五：模 60 的 BCD 码加法计数器设计与验证

## 实验要求

1. 用 Verilog HDL 语言在 Quartus Ⅱ 编译环境下描写出一个模 60 的 BCD 码加法计数器及其测试程序；
2. 要求该计数器具有同步复位 reset（高电平有效）；同步置数 load（高电平有效）；
3. 用测试程序在 Modelsim 仿真环境中对计数器器进行波形仿真测试；画出仿真波形；
4. 总结实验步骤和实验结果

## 模块接口格式

模块名： count60；

输入： [7:0] qout：计数结果输出

cout：进位输出

[7:0] data：置数输入

load：同步置数，高电平有效；

reset：同步清 0，高电平有效；

clk：时钟信号；

## 模块程序：

```
module count60(clk, rst_n, ld, data, qout, c);
input clk, rst_n, ld;
input [7:0] data;
output [7:0] qout;
output c;
    reg c = 0;
    reg[3:0] ten, unit;
    assign qout = {ten, unit};
    always@(posedge clk or negedge rst_n or posedge ld)
    begin
        if(!rst_n)
            {ten, unit} <= 0;
        else if(ld)
            {ten, unit} <= data;
        else
        begin
            if(ten == 5)
            begin
                if(unit == 8)
                begin
                    c = 1;
                    unit <= unit+1;
                end
                else if(unit == 9)
                begin
                    c = 0;
```

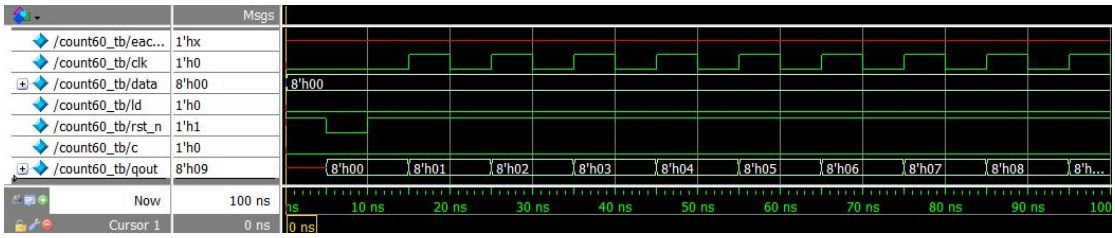
```
                {ten, unit} <= 0;
            end
            else
                unit <= unit+1;
        end
        else if(unit == 9)
        begin
            ten <= ten+1;
            unit <= 0;
        end
        else
            unit <= unit+1;
    end
end
endmodul
```

## 测试程序：

```
`timescale 1 ns/ 1 ns
module count60_tb();
reg eachvec;
reg clk;
reg [7:0] data;
reg ld;
reg rst_n;
wire c;
wire [7:0]    qout;

count60 i(
    .c(c),
    .clk(clk),
    .data(data),
    .ld(ld),
    .qout(qout),
    .rst_n(rst_n)
);

initial
```

```
begin
    rst_n = 1;
    ld = 0;
    clk = 0;
    data = 0;
    #5 rst_n = 0;
    #5 rst_n = 1;
    repeat(2000)
        #5 clk = ~clk;
    data = {4'd4, 4'd7};
    ld = 1;
    #5 ld = 0;
    repeat(20)
        #5 clk = ~clk;
    $stop;
end
endmodule
```

## 仿真波形：




## 课程设计总结：

因为我是的专业是计算机科学与技术，从没接触过 HDL 语言，我接触过最底层的语言就是汇编语言，这次学习 VHDL 可以说刷新了我对编程的认识。

因为计算机认识的只有二进制，我认为衡量一种编程语言效率的唯一标准就是对二进制数据流的处理能力，但很多语言这种能力并不强，原因是很多语言都是加入了编译或者虚拟机来强化代码而不是直接对底层进行操作，例如，结构化的 C 语言，先通过预处理生成.i 文件，再通过编译生成.s 文件， 之后汇编成.o 文件，最后连接到一起生成 a.out、面向对象的 java 语言，先进行词法分析生成 token，然后进行语法分析构造语法抽象树，然后填充一个由符号信息和符号地址的表格，再对语法树进行语义分析，例如变量是否被声明，赋值与变量类型是否匹配，最后生成字节码，把前面每个阶段都转化成字节码存储起来。看着这些复杂的编译过程，在回头看看 VHDl 的过程，甚至不需要操作系统，这意味着 VHDL 可以在一个时钟周期内执行多个指令，而那些主流的编程

语言，这种基于冯诺依曼结构的编程语言即使编译成了机器语言之后，跑起来需要流式取指译码才能执行，指令之间也是流式执行，一条指令就是可能是几十个 cpu 时钟周期，即使是最接近底层的汇编语言，也没法避免这些 cpu 的时钟周期的叠加。举个简单的例子，if-else 结构在 cpu 上是通过比较判断跳转来执行一般 5 个时钟周期，在 verilog 上，直接编译成 mux，一个时钟周期就可以解决，分支越多，verilog 的优势越大。

让计算机越来越好的也不是那些主流编程语言，而是 verilog。举一个例子，GPU 在很多特定的情况是比 CPU 强很多，例如中本聪提出的去中心化的电子记账系统，在白皮书中对工作量证明这一概念的描述中，就意味着可以更改 CPU，只进行一种运算来极大的提高工作效率，具体一点就是用专用电路加速矩阵运算，而 GPU 的这些专用电路就是通过 verilog 实现的。在计算机发展上有一个魔咒是摩尔定律，意思是每 1 美元买到的电脑性能每隔 18-24 个月就会翻一倍，但在随后的发展中工业对计算机性能的要求一直在极大地提高，导致摩尔定律根本跟不上需要。因为传统 CPU 是同构计算，在最初设计时注重的是通用，GPU 作为处理器只是帮助 CPU 处理不好算的并行运算，之后 GPU 每年的性能都会提升 30%。而这种 CPU+GPU 的硬件组合叫做异构这种方式打破了摩尔定律。但在硬件上最流行的异构并不是 CPU+GPU 而是 CPU+FPGA，CPU 兼容性好，如果做大量重复的动作，他每天算的数就会减少很多，例如人工智能这种单一算法一直算就不适合 CPU，面对这些复杂的处理数据的任务，就需要协处理器，要求算法性能高，能低延时的处理并行数据，配置灵活，接口性能高，功率低，就好像刷碗很麻烦于是发明了刷碗机，这个 FPGA 就是刷碗机，中文名是现场可编程门级列阵，可编程就意味着我不需要去用电烙铁焊电路板就能让自定义配置芯片硬件功能，换句话说我可以通过点简单按钮就能让告诉刷碗机我刷的是什么，需要刷多长时间。在这个种异构中，一些框架和编程语言都是翻译成 verilog 再执行的，而且因为 verilog 更加接近于 IO 所以处理海量数据的能力更强应用也最广泛。

从近年看，FPGA 越来越受到 AI 大厂欢迎，被应用的越来越多，云端服务器也在用 CPU+GPU+FPGA 在增强功能，总的来说 CPU 在越做越小，FPGA 在越发展越大。在计算机硬件这方面 verilog 一定会继续发展，甚至我认为，假如编程语言排行榜里不规定必须在 CPU 上运行的话，就目前的发展情况来看 verilog 一定能进前十。

# 课程设计成绩考核表

| 学　院 | 国际教育学院 | | 专　业 | 计算机科学与技术 | |
|---|---|---|---|---|---|
| 班　级 | 183609 | 姓　名 | 高一钧 | 学　号 | 20181173 |
| 课程名称 | 数字电路与逻辑设计 | | | | |
| 课程设计题目 | 基于 Verilog HDL 的数字电路设计与验证 | | | | |

| 考 核 项 目 | 满分值 | 得分 |
|---|---|---|
| 1.独立完成设计任务 | 20 | |
| 2.设计方案、说明书、图纸、程序、计算等完成量 | 30 | |
| 3.创新与发挥 | 10 | |
| 4.答辩（口试） | 40 | |

| 合计得分 | |
|---|---|
| 成　　绩 | |
| 指导教师签字 | |

## 几点说明

一、此表由指导教师进行课程设计成绩评定时填写。

二、课程设计成绩根据学生各项考核最后得分，按“优”、“良”、“中”、“及格”、“不及格”五级评定。期中，90 分以上为“优”；80-89 分为“良”；70-79 分为“中”；60-69 分为“及格”；60 分以下为“不及格”。

三、课程设计结束后，此表由指导教师放入课程设计资料袋，送承担课程的教研室存档。

2020 年　1 月 10 日