实验报告

实验题目：数制系统　　日期：2018 年 10 月 19 日

姓名:罗晏宸　学号:PB17000297　成绩:_____

实验目的：

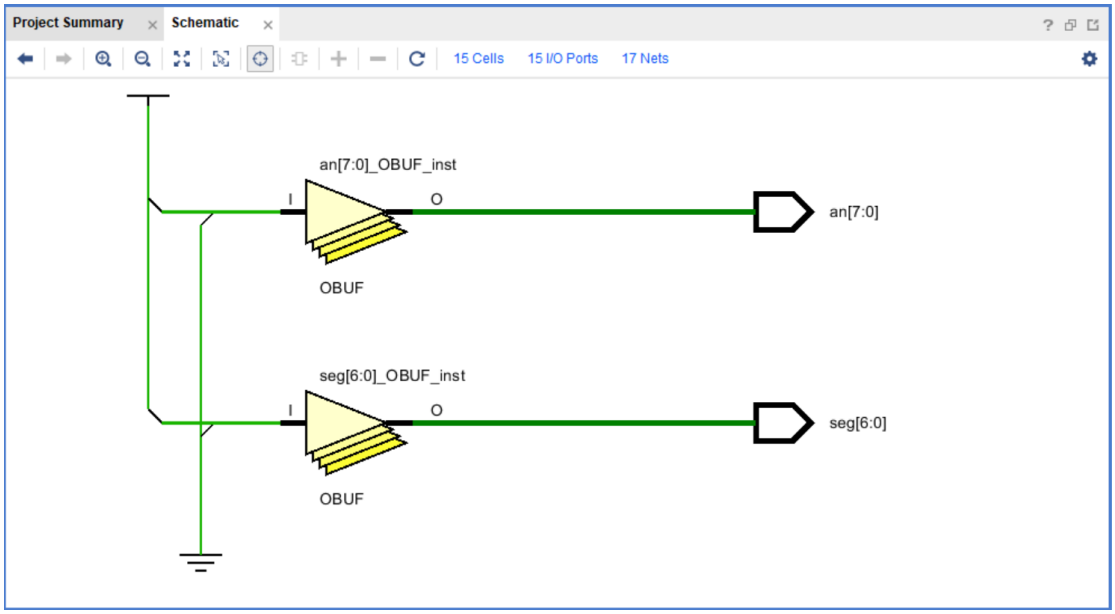
1. 学习使用 Verilog 语言定义不同进制的数
2. 设计能将数据从一个进制转化到另一个的组合电路（以二进制至十进制为例）并加以实现
3. 设计能实现简单的加法操作的组合电路并加以实现
4. 学习超前进位加法电路的原理并设计实现以提高加法操作速度

实验内容（截图、照片与代码）

Lab2_1_1

代码 Lab2_1_1.v

```
`timescale 1ns / 1ps
module lab2_1_1(
    output [7:0] an,
    output [6:0] seg
    );
    reg [3:0] x = 4'd7;
    assign seg[0] = ((~x[3] && ~x[2] && ~x[1] && x[0]) | (~x[3] && x[2] && ~x[1] && ~x[0])) && ~(x[3] && (x[2] | x[1]));
    assign seg[1] = (~x[3] && x[2] && ~x[1] && x[0]) | (~x[3] && x[2] && x[1] && ~x[0]) | (x[3] && (x[2] | x[1]));
    assign seg[2] = (~x[3] && ~x[2] && x[1] && ~x[0]) | (x[3] && (x[2] | x[1]));
    assign seg[3] = ((~x[3] && ~x[2] && ~x[1] && x[0]) | (~x[3] && x[2] && ~x[1] && ~x[0]) | (~x[3] && x[2] && x[1] && x[0])) && ~(x[3] && (x[2] | x[1]));
    assign seg[4] = ((~x[3] && ~x[2] && ~x[1] && x[0]) | (~x[3] && ~x[2] && x[1] && x[0]) | (~x[3] && x[2] && ~x[1] && ~x[0]) | (~x[3] && x[2] && ~x[1] && x[0]) | (~x[3] && x[2] && x[1] && x[0]) | (x[3] && ~x[2] && ~x[1] && x[0])) && ~(x[3] && (x[2] | x[1]));
    assign seg[5] = ((~x[3] && ~x[2] && ~x[1] && x[0]) | (~x[3] && ~x[2] && x[1] && ~x[0]) | (~x[3] && ~x[2] && x[1] && x[0]) | (~x[3] && x[2] && x[1] && x[0])) && ~(x[3] && (x[2] | x[1]));
    assign seg[6] = ((~x[3] && ~x[2] && ~x[1] && ~x[0]) | (~x[3] && ~x[2] && ~x[1] && x[0]) | (~x[3] && x[2] && x[1] && x[0])) && ~(x[3] && (x[2] | x[1]));
    assign an[0] = 0;
    assign an[1] = 1;
    assign an[2] = 1;
    assign an[3] = 1;
    assign an[4] = 1;
    assign an[5] = 1;
    assign an[6] = 1;
    assign an[7] = 1;
endmodule
```



截图 1-lab2_1_1 原理图

照片 1-lab2_1_1 下载

“照片 1-lab2_1_1 下载”的说明：本工程使用数据流风格实现一位已定义的十进制数在 7 段数码管上的显示；代码中已定义 `x = 4'd7`，右侧七段数码管 0 位 CA，CB，CC 管亮，其余管灭，其余位七段数码管全灭，灯亮的数码管笔画构成数字‘7’字样形成输出。

Lab2_2_1

代码 lab2_2_1.v

```
`timescale 1ns / 1ps
module lab2_2_1(
    input [3:0] v,
    output z,
    output [7:0] an,
    output [6:0] seg
    );
    wire [3:0] m;
    lab2_2_1_partA (v,z,m);
    bcdto7segment_dataflow (m,an,seg);
endmodule
```

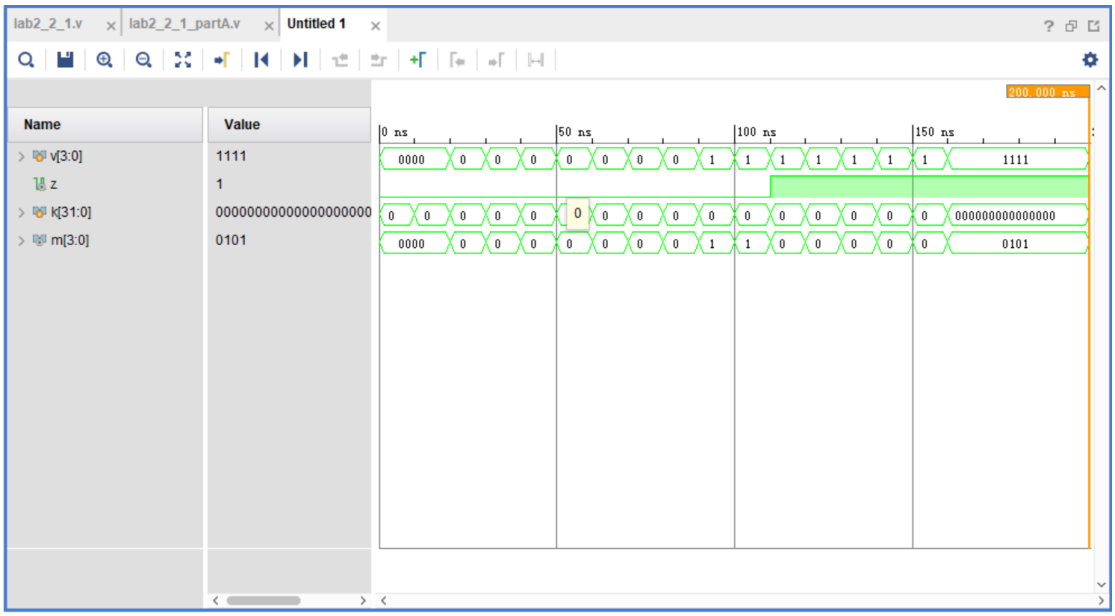
代码 lab2_2_1_partA.v

```
`timescale 1ns / 1ps
module lab2_2_1_partA(
    input [3:0] v,
    output z,
    output [3:0] m
    );
    wire [3:0] CircuitAOutput;
    assign z = (v[3] & v[2]) | (v[3] & v[1]);
    assign CircuitAOutput[3] = 0;
    assign CircuitAOutput[2] = v[1] & v[2];
    assign CircuitAOutput[1] = ~v[1];
    assign CircuitAOutput[0] = v[0];
    assign m[3] = (~z & v[3]) | (z & CircuitAOutput[3]);
    assign m[2] = (~z & v[2]) | (z & CircuitAOutput[2]);
    assign m[1] = (~z & v[1]) | (z & CircuitAOutput[1]);
    assign m[0] = (~z & v[0]) | (z & CircuitAOutput[0]);
endmodule
```

代码 bcdto7segment_dataflow.v

```
`timescale 1ns / 1ps
module bcdto7segment_dataflow(
    input [3:0] x,
    output [7:0] an,
    output [6:0] seg
    );
    assign seg[0] = ((~x[3] && ~x[2] && ~x[1] && x[0]) | (~x[3] && x[2]
&& ~x[1] && ~x[0])) && ~(x[3] && (x[2] | x[1]));
    assign seg[1] = (~x[3] && x[2] && ~x[1] && x[0]) | (~x[3] && x[2] &&
x[1] && ~x[0]) | (x[3] && (x[2] | x[1]));
    assign seg[2] = (~x[3] && ~x[2] && x[1] && ~x[0]) | (x[3] && (x[2] |
x[1]));
    assign seg[3] = ((~x[3] && ~x[2] && ~x[1] && x[0]) | (~x[3] && x[2]
&& ~x[1] && ~x[0]) | (~x[3] && x[2] && x[1] && x[0])) && ~(x[3] &&
(x[2] | x[1]));
    assign seg[4] = ((~x[3] && ~x[2] && ~x[1] && x[0]) | (~x[3] && ~x[2]
&& x[1] && x[0]) | (~x[3] && x[2] && ~x[1] && ~x[0]) | (~x[3] && x[2]
```

```
&& ~x[1] && x[0]) | (~x[3] && x[2] && x[1] && x[0]) | (x[3] && ~x[2] &&
~x[1] && x[0])) && ~(x[3] && (x[2] | x[1]));
    assign seg[5] = ((~x[3] && ~x[2] && ~x[1] && x[0]) | (~x[3] && ~x[2]
&& x[1] && ~x[0]) | (~x[3] && ~x[2] && x[1] && x[0]) | (~x[3] && x[2]
&& x[1] && x[0])) && ~(x[3] && (x[2] | x[1]));
    assign seg[6] = ((~x[3] && ~x[2] && ~x[1] && ~x[0]) | (~x[3] &&
~x[2] && ~x[1] && x[0]) | (~x[3] && x[2] && x[1] && x[0])) && ~(x[3] &&
(x[2] | x[1]));
    assign an[0] = 0;
    assign an[1] = 1;
    assign an[2] = 1;
    assign an[3] = 1;
    assign an[4] = 1;
    assign an[5] = 1;
    assign an[6] = 1;
    assign an[7] = 1;
endmodule
```



截图 2-lab2_2_1 仿真

照片 2-lab2_2_1 下载 A

“照片 2-lab2_2_1 下载 A”的说明：本工程使用数据流风格实现将 4 比特的二进制输入转化为相应的 2 位十进制数（BCD 码）后输出到 LED 灯（最高位）和最右边的七段数码管（低位）上；图中开关 switch[1]拨至开，其余开关拨至关，表示输入为 v=$(0010)_B$；led 灯均灭；右侧七段数码管 0 位 CA，CB，CD，CE，CG 管亮，其余管灭，其余位七段数码管全灭，灯亮的数码管笔画构成数字‘2’字样形成输出，表示输出为$(2)_D$.

照片 3-lab2_2_1 下载 B

“照片 3-lab2_2_1 下载 B”的说明：本工程使用数据流风格实现将 4 比特的二进制输入转化为相应的 2 位十进制数（BCD 码）后输出到 LED 灯（最高位）和最右边的七段数码管（低位）上；图中开关 switch[0]，switch[1]，switch[3]拨至开，其余开关拨至关，表示输入为 v=$(1011)_B$；灯 led[0]亮，其余灯灭；右侧七段数码管 0 位 CB，CC 管亮，其余管灭，其余位七段数码管全灭，灯亮的数码管笔画构成数字‘1’字样形成输出，表示输出为$(11)_D$.

Lab2_3_1

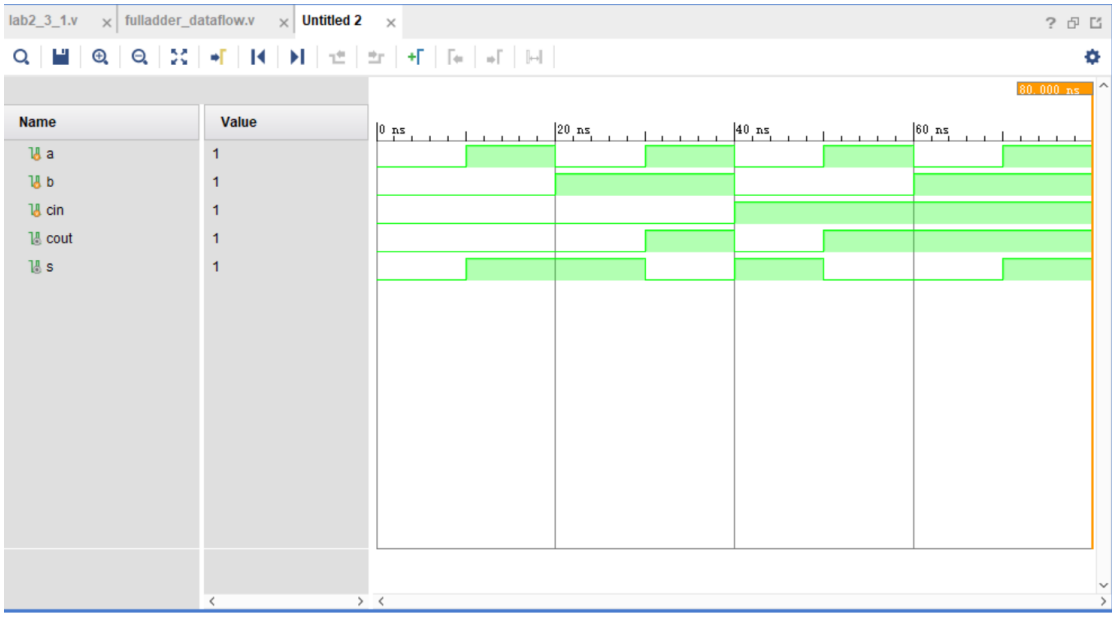
代码 lab2_3_1.v

```
`timescale 1ns / 1ps
module lab2_3_1(
    input [3:0] a,
    input [3:0] b,
    input cin,
    output cout,
    output [3:0] s
    );
    wire out[2:0];
    fulladder_dataflow (a[0],b[0],cin,out[0],s[0]);
    fulladder_dataflow (a[1],b[1],out[0],out[1],s[1]);
    fulladder_dataflow (a[2],b[2],out[1],out[2],s[2]);
```

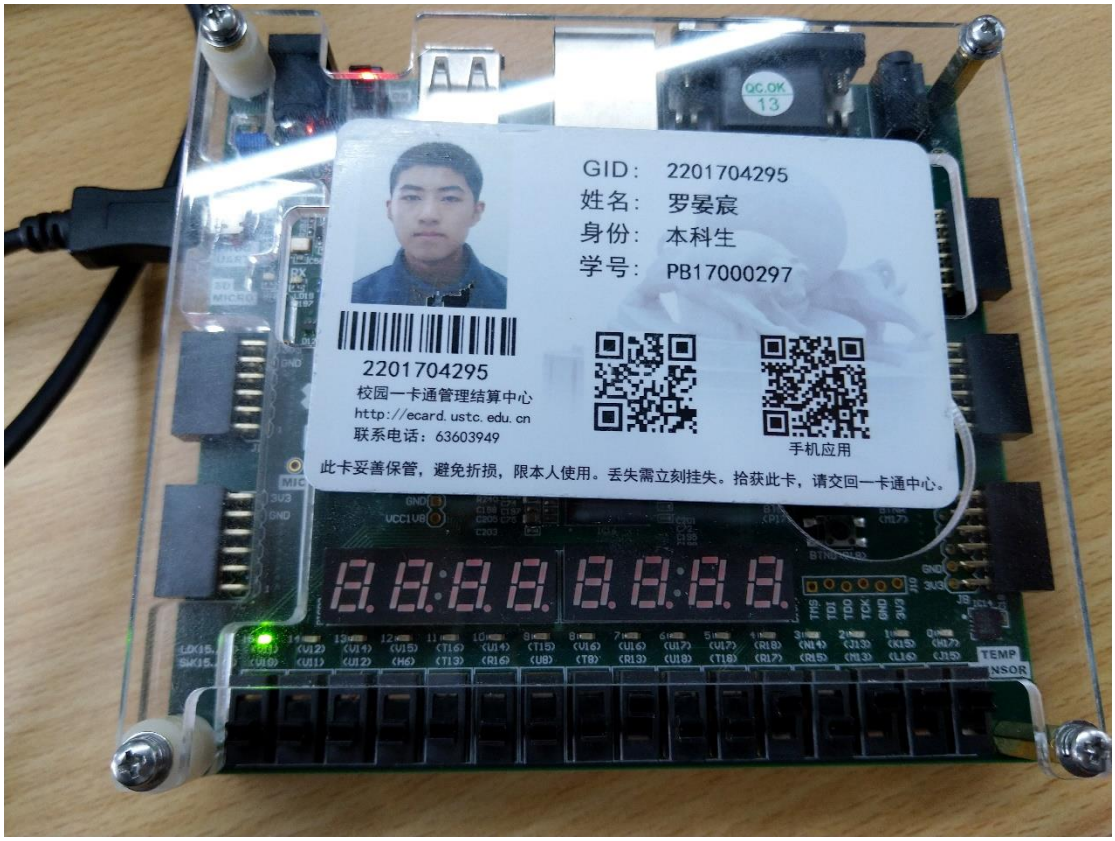
```
    fulladder_dataflow (a[3],b[3],out[2],cout,s[3]);
endmodule
```

代码 fulladder_dataflow.v

```
`timescale 1ns / 1ps
module fulladder_dataflow(
    input a,
    input b,
    input cin,
    output cout,
    output s
    );
    assign s = a ^ b ^ cin;
    assign cout = ( a & cin) | (b & cin) | (a & b);
endmodule
```



截图 3-lab2_3_1 仿真



照片 4-lab2_3_1 下载

“照片 4-lab2_3_1 下载”的说明：本工程使用数据流风格实现 4 比特波纹进位加法器；图中开关 switch[0]，switch[1]，switch[2]，switch[4]，switch[7]，拨至开，其余开关拨至关，表示输入为 a=$(0111)_B$，b=$(1001)_B$，cin=0；led[15]亮，其余灯灭，表示输出为 cout=1，s=$(0000)_B$.

实验总结：

在本实验中，我学习了如何定义不同进制的数，并使用数据流设计不同的转换进制或数码的电路。并利用这些知识实现了 4 位加法器，练习了实验操作与 Verilog 语法知识。