**《数字逻辑》实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **姓名** | |  | | **年级** | |  |
| **学号** | |  | | **专业、班级** | |  |
| **实验名称** | **加减法器** | | | | | |
| **实验时间** | **2019/12/23** | | **实验地点** | | **DS1410** | |
| **实验成绩** |  | | **实验性质** | | **□验证性 □设计性 □综合性** | |
| 教师评价：  □算法/实验过程正确； □源程序/实验内容提交 □程序结构/实验步骤合理；  □实验结果正确； □语法、语义正确； □报告规范；  评语：  评价教师签名（电子签名）： | | | | | | |
| 一、实验目的   1. 加深对全加器的理解。 2. 学会器件的串联操作，实现更高级的功能。 3. 加深对器件复用的理解和实际操作。 4. 深入理解数码管动态扫描的原理，并掌握数码管的实际应用。 5. 理解数制运算的原理，加深对补码的理解，体会有符号数和无符号数的区别，从而对计算机实现的底层有更深入的认识。 | | | | | | |
| 1. 实验项目内容 2. 设计一个1位全加器，然后由32个1位全加器串联组成32位加法器（画出电路原理图），并通过写仿真文件、看RTL电路图、下载至开发板验证其正确性； 3. 用全加器来构建全减器（画出电路原理图），实现一个32位加减法器，并通过写仿真文件、看RTL电路图、下载至开发板验证其正确性； 4. 编写顶层模块将加减法器模块和7段数码管显示模块连接起来，将SW15~SW12作为第一操作数低4位，SW11~SW8作为第二操作数低4位，SW0=0（1）时做加（减）法，添加选择控制SW1~SW2，能选择将操作数1或操作数2或结果显示在7段数码管上。 5. 在3）中的加减法器中增加进位标志和溢出标志，设计实现一个带进位标志和溢出标志的32位加减法器，将SW15~SW12作为第一操作数（可显示在左起第一个7段数码管上），SW11~SW8作为第二操作数（可显示在左起第二个7段数码管上），将结果显示在7段数码管上，进位和溢出标志通过LED灯显示； 6. 将所设计的加减法器与系统自带的“+\-”进行比较。 | | | | | | |
| 1. 实验设计 2. **1位全加器的设计**    1. 真值表  |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | X\_i | Y\_i | C\_i+1 | S\_i | | 0 | **0** | **0** | **0** | **0** | | 0 | **0** | **1** | **0** | **1** | | 0 | **1** | **0** | **0** | **1** | | 0 | **1** | **1** | **1** | **0** | | 1 | **0** | **1** | **1** | **0** | | 1 | **1** | **0** | **1** | **0** | | 1 | **1** | **1** | **1** | **1** |  * 1. **电路**      1. **5位全加减器的设计**    * 1. **说明**  * 由于32位的全加器无法在FPGA上实现，因此做了5位有符号数的加减法器，其中低四位是数据位，最高位是符号位。 * 支持正负数的输入及运算（例如 -13-(-9) 将输出 -4）。   + 1. **层次结构**      * + 1. **五位全加器**        1. **RTL电路图**      * + - 1. **设计思路** * 采用generate语句，实例化5个一位全加器。 * 进位的输出信号，由一位全加器串行连接后自然得到。 * 溢出信号：如果两个加数的最高位（符号位）相同，而结果的最高位（符号位）与之不同，则有溢出。   + 1. **5位全加减器**        1. **RTL电路图**   D:\qq\1614395133\Image\Group\Image1\XD}}HG4BCPO1US4L7HN)SVB.png   * + - 1. **概念说明电路图**      * + - 1. **处理说明** * 考虑输入的数为有符号数，即可能为正也可能为负数。 * 考虑输入的数为原码形式。 * 全部转化为补码进行运算，并将结果转化为原码形式返回。   + - 1. **求补码** * 对于第一操作数，判断其符号位，如果最高位为1（负数），则保持最高位不变，剩下的取反，然后再加1。 * 对于第二操作数，由于涉及到减法，单独处理。设置减法标志符和补码标志符。   + 最高位（符号位）的处理：被减数的最高位与减法标志符做异或。   + 如果被减数为负数，减去一个负数相当于加一个正数，故将被减数的最高位与减法标志符做异或，此时二者均为1，异或后得到0，作为处理后的第二操作数，即转化为了一个正数。除了符号位之外，不需要额外处理。   + 如果被减数为正数，即求其相反数的补码，符号位已经处理，剩余位数与减法标志符异或。（此时得到的结果并不是补码，因为还没有加1）     1. **数码管应用**        1. **说明：** * 最左边一个是第一操作数(不显示符号位)。 * 第二个是第二操作数（不显示符号位）。 * 第四个是运算结果（不显示符号位）。 * 显示以16进制形式显示。 * 模块有分频模块，位选模块，段选模块，解码模块，加减法器模块。   + - 1. **动态扫描** * **必要性**：显示多位数字时，一个数码管只能显示一位数字，所以显示几位数字就需要几个数码管。一个数码管占用7个管脚，如果4位就需要28个管脚，造成资源的浪费。 * **动态扫描：**让每个数码管轮流显示，而不是全部显示。利用人的视觉暂留现象，从而使得看起来像是同时亮了一样。   + - 1. **时钟分频** * 由于FPGA内置的时钟频率较高，不利于操作，因此可以进行简单分频。 * 经过测试，将100MHz分频为2000Hz（即使用50000个内置时钟周期作为一个新的时钟周期）   + - 1. **位选模块** * 设置初始的电平为4b'1110表示第四个数码管亮（显示运算结果）。 * 进行轮流点亮，每个时钟周期进行移位（循环）变为1101, 1011, 0111。   + - 1. **段选模块** * 根据位选信号，向不同的数码管传入对应的数据。 * 当1110时，第四个数码管亮，传入运算的结果，当1011时，第二个数码管亮，传入第二操作数，当0111时，第一个数码管亮，传入第一操作数。   + - 1. **解码模块** * 8个led灯的输入的解码，让led灯对应的位闪烁（低电平亮灯）。 * 参照FPGA的7段数码管引脚进行编写。 | | | | | | |
| 1. 实验过程或算法(关键步骤、核心代码注解等） 2. **一位全加器** 3. **描述**    1. 输入有x表示第一操作数, y表示第二操作数, cin表示进位值    2. 输出有sum表示结果，cout表示进位。    3. 进位的条件是x和y均为1或者xy中有一个为1且进位cin为1。 4. **verilog代码**   module **fullAdder**(      input x, y, cin,      output sum, cout  );  assign sum = x ^ y ^ cin;  assign cout = x & y | (x ^ y) & cin;  endmodule *// fullAdder*   1. **32位全加器**    1. **Verilog语言描述**   module **Add**(      cin, x, y, sum, cout, overflow  );      parameter n = 32;      input cin;  *//进位输入信号*      input [n-1:0] x, y; *//加数与被加数*      output [n-1:0] sum; *//加法结果*      output cout, overflow;  *//进位输出信号和溢出信号*      wire [n:0] tout;    *//进位信号共有n+1位*      assign tout[0] = cin;   *//进位输入信号赋值给进位信号的第一位*      assign cout = tout[n];  *//进位信号的末位为进位输出信号*    *//使用生成能力，实例化5个全加器，得到4位的加法结果sum*      genvar i;      generate          for(i = 0; i <= n-1; i = i + 1)          begin              fullAdder stage(x[i], y[i], tout[i], sum[i], tout[i+1]);          end      endgenerate    *//溢出位：如果加数和被加数最高位（符号位）相同可而结果的最高位（符号位）与之不同，则有溢出*      assign overflow = (x[n-1] & y[n-1] & ~sum[n-1] | (~x[n-1] & ~y[n-1] & sum[n-1]));  endmodule *// Add*   * 1. **仿真**      1. **仿真数据：**      * + 1. **仿真结果**        1. **5位全加减器**    1. **描述**   调用5位全加器模块，传入处理后的第一操作数和第二操作数。若求补码，将cin设置为`1`表示求补码时的加1运算。   * 1. **Verilog语言实现**   module **FullAddSub**(      x, y, subFlag, sum, cout, overflow  );  parameter n = 5;  input [n-1:0] x, y; *//输入两个做加减法的数*  input subFlag;  *//加减法标志*  output [n-1:0] sum; *//运算结果*  output cout;    *//进位位*  output overflow;  *//溢出*  *// 对输入的x，y进行处理求补码*  wire[n-1:0] proy;  wire [n-1:0] prox;  assign prox = (x[n-1] == 1) ? (~{2'b0, x[n-2:0]} + 2'b1): x; *// x 补码*  wire [n-1:0] compFlag; *// 是否对y进行求补码的标志*  wire negativeY; *// 表示y为负数*  wire cin; *// 表示被减数转为补码时的加1*  assign negativeY = (y[n-1] == 1) ? 1 : 0; *// y是否为负数*  *//是否求补码，进行按位异或：当要做减法或是Y为负数的时候求补码*  assign compFlag = (subFlag ^ negativeY) ? 5'b11111 : 5'b0;  assign cin = (subFlag ^ negativeY) ? 1 : 0;  assign proy[n-2:0] = y[n-2:0] ^ compFlag; *// 求补码：除最高位 异或*  assign proy[n-1] = subFlag ^ y[n-1]; *// 最高位单独处理*  wire [n-1:0] compSum;  Add myadd(.cin(cin), .x(prox), .y(proy), .sum(compSum), .cout(cout), .overflow(overflow));  *// 符号位不变，其余位取反、+1*  assign sum[n-1] = compSum[n-1];  assign sum[n-2:0] = (compSum[n-1] == 1) ? ~compSum[n-2:0]+1 : compSum[n-2:0];  endmodule *// FullAddSub*   1. **仿真** 2. **仿真数据**      1. **仿真结果**      1. **数码管及FPGA实现**   module **smg**(clk,data,sm\_wei,sm\_duan,rst);  input clk,rst;  input [15:0] data;  output [3:0] sm\_wei;  output [7:0] sm\_duan;  *//----------------------------------------------------------*  *//分频：更新频率为100Hz*  integer clk\_cnt;    *//分频周期计数*  reg clk\_2000Hz;  always @(posedge clk)  if(rst == 1) begin  *//重置数码管分频显示*      clk\_cnt <= 1'b0;      clk\_2000Hz <= 1'b0;  end  else if(clk\_cnt == 32'd25000)  begin      clk\_cnt <= 1'b0;  *//半个周期结束，重新计数*      clk\_2000Hz <= ~clk\_2000Hz;    *//时钟翻转*  end  else clk\_cnt <= clk\_cnt + 1'b1;  *//分频周期计数+1，又过去一个时钟周期*  *//让数码管于每个分频了的时钟周期在4个位交替闪烁*  *//利用人眼视觉暂留，让人看到四个数码管显示的不同内容*  reg [3:0] wei\_ctrl=4'b1110;  always @(posedge clk\_2000Hz)      wei\_ctrl <= {wei\_ctrl[2:0],wei\_ctrl[3]};  *//段控制：每一位的数码管的8个led灯的输入*  reg [3:0]duan\_ctrl;  always @(wei\_ctrl)      case(wei\_ctrl)          4'b1110:duan\_ctrl=data[3:0];          4'b1101:duan\_ctrl=data[7:4];          4'b1011:duan\_ctrl=data[11:8];          4'b0111:duan\_ctrl=data[15:12];          default:duan\_ctrl=4'hf;        endcase  *//----------------------------------------------------------*  *//解码模块：8个led灯的输入的解码，让led灯对应的位闪烁（低电平亮灯）*  reg [7:0]duan;  always @(duan\_ctrl)      case(duan\_ctrl)          4'h0:duan=8'b1100\_0000;*//0*          4'h1:duan=8'b1111\_1001;*//1*          4'h2:duan=8'b1010\_0100;*//2*          4'h3:duan=8'b1011\_0000;*//3*          4'h4:duan=8'b1001\_1001;*//4*          4'h5:duan=8'b1001\_0010;*//5*          4'h6:duan=8'b1000\_0010;*//6*          4'h7:duan=8'b1111\_1000;*//7*          4'h8:duan=8'b1000\_0000;*//8*          4'h9:duan=8'b1001\_0000;*//9*          4'ha:duan=8'b1000\_1000;*//a*          4'hb:duan=8'b1000\_0011;*//b*          4'hc:duan=8'b1100\_0110;*//c*          4'hd:duan=8'b1010\_0001;*//d*          4'he:duan=8'b1000\_0110;*//e*          4'hf:duan=8'b1000\_1110;*//f*          default : duan = 8'b1100\_0000;*//0*      endcase  *//----------------------------------------------------------*  assign sm\_wei = wei\_ctrl;  assign sm\_duan = duan;  endmodule  module **smg\_ip**(x, y, rst, subFlag, clk, sm\_wei, sm\_duan,  symbolX, symbolY, symbolResult, cout, overflow);  input clk, rst;  *//output [3:0]sm\_wei;*  output [3:0] sm\_wei;    *//位控制信号*  output [7:0] sm\_duan;   *//段控制信号*  *//----------------------------------------------------------*  wire [15:0]data;    *//显示在每一位的数据：每四位data对应每一位数码管*  wire [3:0]sm\_wei;  wire [7:0]sm\_duan;  *//----------------------------------------------------------*  input [4:0] x, y;   *//输入的待计算数据，需要完成4位加减法，故x、y设定为5位，最后一位为符号位*  input subFlag;  *//减法标志*  wire [4:0] sum; *//计算结果*  output cout;    *//进位信号*  output overflow;    *//溢出信号*  output symbolX, symbolY, symbolResult;  *//拼接输入信号为16位的data，然后将其输入smg实例化的U1中，操控七段数码管发光*  assign data[15:12] = x[3:0];  assign data[11:8] = y[3:0];  assign data[7:4] = 4'b0000; *//不需要的计算位，将其置为0*  assign data[3:0] = sum[3:0];  *//输出结果*  assign symbolX = x[4];  assign symbolY = y[4];  assign symbolResult = (overflow == 1) ? ~sum[4] : sum[4];  FullAddSub fuladdsub(x, y, subFlag, sum, cout, overflow);  smg U1 (.clk(clk),.data(data),.sm\_wei(sm\_wei),.sm\_duan(sm\_duan),.rst(rst));  endmodule   1. **开发板**   **功能说明：从右开始数，第1-4位拨码开关代表第二操作数的数值，第5位拨码开关代表第二操作数的符号，第6位拨码开关代表操作符（置0代表加，置1代表减），第7-10位拨码开关代表第一操作数的数值，第11位拨码开关代表第一操作数的符号。**  **数码管第一段代表第一操作数的数值，正下方的LED灯代表其符号（亮为负，不亮为正）；数码管第二段代表第二操作数的数值，正下方的LED灯代表其符号（亮为负，不亮为正）；数码管第四段代表运算结果的数值，正下方的LED灯代表其符号（亮为负，不亮为正）。从右数第一个LED灯代表进位，第二个LED灯代表溢出。**  **（1）上电、初始化**    **（2）测试-8+9，结果为1，且产生溢出信号。（程序设计时参照了教材上有符号数的溢出判定规则，此处可以证明溢出检测的设计正确）**    **（3）测试8-9，结果为-1。**    **（4）测试-8-9，结果为-F，产生进位信号和溢出信号。（程序设计时参照了教材上有符号数的溢出判定规则和进位规则，此处可以证明溢出检测和进位检测的设计正确）**    **（5）测试4-2，结果为2，产生溢出信号。（程序设计时参照了教材上有符号数的溢出判定规则，此处可以证明溢出检测的设计正确）**    **（6）测试0-2，结果为-2。**    通过充分的测试可以看出本次实验的设计正确，功能正常。 | | | | | | |
| 五、实验过程中遇到的问题及解决情况   1. 在求补码的时候，需要保持符号位不变，其余各位置按位取反，然后加一。但是这里的加一是对于整体而言的，即可能由于加一导致符号位发生进位。因此求补码的时候采用了一种较为巧妙的写法，即：   assign prox = (x[n-1] == 1) ? (~{2'b0, x[n-2:0]} + 2'b1): x; *// x 补码*   1. 由于需要按位取反，然而在位数不同的时候并不能直接进行异或操作。考虑使用移位器，对数位多的数据进行移位，也可以考虑将位数少的扩充。本次实验采用了后者，即将1扩充为了11111从而与另一个5位宽度的数据进行异或操作。 2. 由于涉及到求补码和减法运算，因此需要特别关注这一部分。考虑通用化的处理，首先对所有的输入数据（原码）进行求补码，然后将补码再输入到加减法器模块中，做相应的处理。然而这里牺牲了逻辑上的一致性，对一种特殊情况额外考虑，即减一个负数的情况，可以转化成加一个正数。必须承认，这样的设计使得逻辑复杂度变得更高。 | | | | | | |
| 六、实验结果及分析和（或）源程序调试过程  （一） 实验结果展示（源程序调试过程）  **（1） 输入说明**   * **第二操作数**：右边起第1-4个表示第二操作数的数值（二进制），第5个表示其符号，向上拨表示为负数。 * **第一操作数**：右边起第7-10个表示第一操作数的数值（二进制），第11个表示其符号，向上表示为负数。 * **运算符号**：右边起第5个（在两个操作数之间），向上拨表示减法。 * **重置**： 左边起第一个拨码开关。   **（2） 输出说明**   * **操作数以及结果**：显示在7段数码管中 * **操作数以及结果的正负**：以LED灯的形式显示在数码的下方（亮表示负数） * **进位标志符**：显示在右边起第二个LED灯 * **溢出标志符**：显示在右边起第一个LED灯   **（3） 演示**（由于FPGA操作演示已经过检查，故不做多余赘述）  **（4） 补充说明**  对于有符号数来说，溢出信号具有意义，而一般不考虑进位信号。进位信号发生在符号位均为1，或者二者之一为1且运算后结果符号位为0，即当以下情况，会产生进位信号：   1. 两个负数相加。 2. 两个异号数相加，且结果为正数。   （二） 系统自带的加减法操作和本实验的加减法器比较   1. **RTL电路图情况**  |  |  | | --- | --- | | （a）本实验 | s | | （b）系统 |  |   自己编写的32位加减法器的电路复杂程度远远超出系统自带的加减法操作符实现的32位加减法器。   1. **运行功耗对比**  |  |  | | --- | --- | | （a）本实验的加减法器 | IMG_257 | | IMG_258 | | （b）系统的加减法器 |  | | IMG_261 |  1. 系统自带的加减法操作符实现的32位加减法器I/O端口使用远远大于自己编写的32位加减法器 2. 自己编写的32位加减法器消耗功率远远大于系统自带的加减法操作符实现的32位加减法器 3. 自己编写的32位加减法器占用的逻辑资源远远大于系统自带的加减法操作符实现的32位加减法器，是其的几乎10倍   六、小组分工情况说明  本实验原理较为基础，但是针对有符号数各种情况进行处理，仍有一定的复杂性。每个人进行了一定的分工，但也只是有所侧重，实际上每个人都参与了实验的每个过程和每个步骤，大家提出自己的建设性意见，同时也不断的给出质疑，推动进一步完善，每个人都从中得到了十足收获。   1. 田润泽，负责全加器和全减器原理与设计，负责加减法器和数码管的顶层整合，负责实验报告的梳理整合。 2. 姚语涵，负责数码管原理梳理与设计，代码的注解，负责与系统加减法器的对比。 3. 蔡嘉轩，负责加减法器和数码管的优化，进行充分的运行测试和改进。 4. 尹宇慧，负责加减法器和数码管的优化，负责实验报告的完善。 | | | | | | |