**《数字逻辑》实验报告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **姓名** | **魏永森** | | **年级** | | **2019级** |
| **学号** | **20192242** | | **专业、班级** | | **计算机科学与技术 卓越01** |
| **实验名称** | **实验六 加减法器** | | | | |
| **实验时间** | **11.20** | **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、掌握一位全加器的逻辑功能，学会多位全加器的设计原理。  2、学会利用加法器完成减法器的设计与实现。  3、掌握多位数码管分时复用显示原理及应用。  4、掌握位拼接运算符的使用。 | | | | | |
| 二、实验项目内容  1、设计一个 1 位全加器，然后由 32 个 1 位全加器组成 32 位加法器，并通过  写仿真文件、看 RTL 电路图验证其正确性。  2、在加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 4 位加法器，并编写顶层模块将加法器模块和 7 段数码管模块连  接起来，将 SW3~SW0 作为第一操作数（可显示在左起第一个 7 段数码管  上），SW7~SW4 作为第二操作数（可显示在左起第二个 7 段数码管上），将  结果显示在最右侧的 7 段数码管上，进位和溢出标志通过 LED 灯显示。  3、用全加器来构建全减器，画出电路图，设计并实现带进位位和溢出位的 32  位加/减法器，将 SW3~SW0 作为第一操作数（可显示在左起第一个 7 段数  码管上），SW7~SW4 作为第二操作数（可显示在左起第二个 7 段数码管上），  SW15=0（1）时做加（减）法，将结果显示在最右侧的 7 段数码管上，进  位和溢出标志可通过 LED 灯显示。  4、将所设计的加减法器与系统自带的“+\-”进行比较。可以从 RTL 电路分  析、仿真波形、开发板资源使用情况等方面进行比较。 | | | | | |
| 三、实验设计  **数码管的使用:**  动态数码管显示的原理是： 每次选通其中一位， 送出这位要显示的内 容， 然后一段时间选通下位送出对应数据， 4 个数码管这样依次选通并送 出相应的数据，结束后再重复进行。这样只要选通时间取合适由于人眼的视觉暂留，数码管看起来就是连续显示。  32位数字的表示：为了缩短位数，我们用8位16进制数来表示32位2进制数。  然后数码管上的四个数字，每个数字可以表示一个16进制数字（0~f）。所以一个32位的2进制数，只要用8个数码管数字来表示就可以。  **补码与原码以及计算：**有符号数字的第一位为符号位。0代表为正数，1代表为负数。  我们用4位的二进制数来举例：  -1：1111； 第一位为符号位1，后三位为1（001）的补码(取反加一就得到111）  2：0010； 第一位为符号位，后三位为2（010）的补码。因为符号位是0，正数的补码是自身。  加法操作：-1+2=1111+0010=10001 最高位第五位舍去，留下四位0001.  结果0001的符号位是0，后三位补码为001（正数补码是自身），转换成十进制数就是+1，也就是正确的结果。  如果是-2+1  -2：1110 1：0001  -2 + 1 =1110+0001=1111；  符号位是1，对后三位求补码（取反加一）为001；  故而最后的结果为负数，大小是001；  也就是-1。  减法就是家去一个负数，原理同上。补码原码的转换。  这个就是有符号数的加减法的原理。  然后**实验困难**之处是拨码开关和数码管有限。我们需要用更少的开关，实现更多的输入。我们设计的是用两个拨码开关（4种组合 00，01，10，11）来实现一个32位数字的4部分的输入（[31:24],[23:16],[15:8],[7:0].利用case语句实现就可以。  为了便于观察，我们还决定使用8个LED灯来表示 两个操作数的八部分输入。 | | | | | |
| 四、实验过程或算法  首先设置一个时钟clk信号进行时序逻辑控制。  **启动系统:**  然后有rst\_n复位（使能）信号来启动和复位整个系统。  Rst\_n=1;打开系统。 rst\_n=0;关闭系统并重置两个操作数。  **数码管的显示:**  32位数在数码管上需要8个数字来显示（每个数字可显示二进制数字的四位，最大位1’hf,即十进制的15），分成两个完整的四位数码管来显示。  两个操作数的[15:0]占据一个四位数码管，[31:16]再占据一个，两次数码管显示可以完整的显示出32位数字。  具体的数码管显示过程写在了display模块中。该模块会读进去一个16位的数据，将其分割成四部分（因为数码管的引脚是一样的，所以理论上显示的应该是同一个数字，但是我们可以让数码管的四个数字快速切换值，利用视觉暂留效果实现四位数码管显示不同的数字）显在再数码管上。Sm\_wei控制这这四部分。  然后根据sm\_wei选出的四位二进制数字（转化成16进制数正好在0~f之间），通过sm\_duan进行输出。Sm\_duan是一个8位矢量，代表了数码管的8个引脚，根据这8个引脚的信号来控制数码管的输出显示数字。  比如当前sm\_wei=4’b1111; // 即10进制的15；16进制的f。  那么通过case语句使得sm\_duan=8’b 8'b1000\_1110;显示出“f”。  其他数字与段的控制同理。  **对两个操作数的操作:**  In\_ctrl用来控制当前输入和数码管显示的操作数是第几个。置为0时为操作数一的读入与显示，置为1时为操作数二的读入与显示。  Operator\_ctrl矢量时两位（四种不同情况）。用来控制每个32位数的段。将其分为四部分。每部分表示8位数，在数码管上占用两位数。然后operator\_ctrl[1:0]的四种组合情况，可以表示完整32位数.并利用输出LED灯显示的矢量信号led[7:0]配合显示目前对操作数的输入位置。  Operator\_tmp[7:0]矢量信号用来输入我们想要的八位二进制数字。对两个操作数的输入时会用到。  具体操作如下：  In\_ctrl=0;operator\_ctrl=00;对第一个数的[7:0]位进行输入，led[0]亮。  In\_ctrl=0;operator\_ctrl=01;对第一个数的[15:8]位作输入，led[1]亮。  In\_ctrl=0;operator\_ctrl=11;对第一个数的[23:16]位作输入，led[2]亮。  In\_ctrl=0;operator\_ctrl=10;对第一个数的[31:24]位作输入，led[3]亮。  In\_ctrl=1;operator\_ctrl=00;对第二个数的[7:0]位进行输入，led[4]亮。  In\_ctrl=1;operator\_ctrl=01;对第二个数的[15:8]位作输入，led[5]亮。  In\_ctrl=1;operator\_ctrl=11;对第二个数的[23:16]位作输入，led[6]亮。  In\_ctrl=1;operator\_ctrl=10;对第二个数的[31:24]位作输入，led[7]亮。  上述情况的输入操作均使用operator\_tmp[7:0]矢量从拨码开关输入。  **结果的计算与显示:**  Result\_ctrl信号表示了是否要在数码管上面显示结果。而operation\_select 信号则是和其一同产生作用的运算选择（加减法）。Duan\_ctrl是对结果的分段显示（数码管八位的前四位和后四位，数码管每位可存储4位二进制数，最大为1’hf。）  Result\_ctrl=1,operation\_select=0;duan\_ctrl=0显示减法结果的后四位。  Result\_ctrl=1,operation\_select=0;duan\_ctrl=0显示减法结果的前四位。  Result\_ctrl=1,operation\_select=1;duan\_ctrl=0显示加法结果的后四位。  Result\_ctrl=1,operation\_select=1;duan\_ctrl=0显示加法法结果的前四位。  当result\_ctrl=0时，显示当前对两个操作数的输入情况，不显示加减法结果。  **负数与溢出:**  使用is\_fu 与 is\_over信号两个输出信号来表示目前两个操作数的逻辑计算结果是否是负数与是否超过32位（最高位是符号位）的存储范围。如果有相应的信号产生。与之连接的LED会亮起显示。  **具体计算过程实现:**  显而易见，减法就是加法的变化，改变符号位即可，所以我们主要来看32位加法器的实现。  首先读进去两个32位的数X[31:0],Y[31:0].  我们必须定义，为了方便在开发板上输入，（方便描述，我们写成4位二进制数表示），-1我们在开发板上的输入为 1001；  1（符号位）001（十进制1）；  因为这样会便于认知，如果我们用标准的计算数，-1应该是1111；但是从开发板上输入的时候我们还得换算，比较麻烦，所以这个流程我们放到程序里面。-1就是符号位1加上二进制数001组成4’b1001就可以。  所以我们将X，Y两个32位数据，输入开发板的时候，第一步就是要将其转化为正确的标准的计算补码。用X0[31:0],Y0[31:0]来存储。  若X[31]==1,也就是符号位为1的话，我们就对后面30~0总共31位取反之后加上原来的X[31]就组成标准的计算器负数补码了,就是X0[31:0].  举个栗子，还是用四位数的举例，为了方便输入，我们定义1001为-1，但是实际输入以后，在程序里面对后三位“001”，取反加一，变为”111”,加上符号位，也就是”1111”.这就是标准的补码表示”-1”.  当然如果X[31]符号位本来就是0的话就不用进行求补码的操作了。  Y的操作也是这样。  这样之后得到标准的计算器表示补码X0[31:0],Y0[31:0].  想要计算结果就是利用全加器代码for循环，计算结果就行，得到的结果就是正确的结果。然后如果结果的符号位是0的话，不进行任何操作。如果符号位是1，那么使得is\_fu=1,剩下的31位数据，取反加一之后通过数码管显示就是正确结果。  然后我们来讨论溢出。当X，Y一正一负的时候，两者进行加法操作不可能有溢出，永远在31位数字和1位符号位可表示的范围之内。  但是当X，Y的符号位相同的时候就有可能超出31位数字和1位符号就可能储存不下。   1. 对于X，Y都是正数的话：(四位数举例)   我们知道X,Y是我们输入的数字。(7:0111 7:0111)  X0，Y0是我们求补码之后得到的数字。(7:0111 7:0111)  很显然，X，Y相加会溢出，因为1位符号位加3位数字位，存储 （-7~7）。  我们只需要根据全加器模块，直接对X，Y进行操作。  [wide-1:0] S; [wide:0] C;  **S[i] = X[i]^Y[i]^C[i];**  **C[i+1] = (X[i]&Y[i])|(Y[i]&C[i])|(C[i]&X[i]);**  如果C[wide-1]=1,即说明，X加Y产生了溢出。如果没有溢出的话，结果S[31:0]就是正确的结果。  (2)  我们知道X,Y是我们输入的数字。(-7:1111 -7:1111)  X0，Y0是我们求补码之后得到的数字。(-7:1001 -7:1001)  要想判断，是否溢出，我们对X0，Y0进行全加器操作时行不通的。  我们可以简单的发现只要第一次输入的X,Y.对他们直接进行相加就可以。因为他们溢出的话，后三位相加会产生第四位的溢出1，然而X，Y本来的符号位就是1，所以三个1相加就还是1.然后对C[wide-1]进行检验。如果其值为1，说明产生了进位。  如果没有产生进位的话，想要的到正确的相加值，只要再对X0,Y0进行全加器操作就行，得到的S就是结果。  这两种情况的最后，都得对S[31:0]进行判断。  如果符号位是0，那么直接用数码管对剩下的31位输出显示即可。  如果符号位是1，那么需要对后面31位求补码才是我们理解的正确的值，用数码管对后31位进行输出，再用is\_fu=1输出值为负即可。  减法运算：  因为我们为了方便输入进行的操作，（还是4位数举例）  -1我们输入1001就行。那么 -1 - -1实际上就是-1 + 1；  然后根据我们的定义-1=1001；变换符号位1=0001；不需进行后面几位的求补码操作，然后用add32模块计算就可以。 | | | | | |
| 五、实验过程中遇到的问题及解决情况  开始的时候遇到了底层模块不能在顶层模块调用的问题。后来发现是因为模块不能再always语句中调用。于是就在顶层模块中定义了好几个控制数码管的sm\_wei和显示数码管的段sm\_duan.命名错开就可以。然后再顶层代码的最后，分别对这些变量调用display模块就可以。这样就能输出显示了。  本次实验刚开始的时候没有做出带符号位的加减法，然后我们就主要做出了无符号数的加减法。然后根据已经完成的模块，修改add32加法模块就可以实现有符号数的加减法。 | | | | | |
| 六、实验结果及分析和（或）源程序调试过程  整体RTL仿真：    Add32模块RTL仿真:    数码管：    顶层模块激励仿真：      数码管输出sm\_duan 控制数码管的亮与暗，所以仿真结果是不确定值。  下面是使用 -1 + -4 作为示例。  左面的led亮起代表是负数，5是大小，结果是-5；    改变运算符号 -1 - -4  结果为3；    七、小组分工情况说明  魏永森：设计出无符号位的算法，根据位数的需求设计并编程完成了七段数码管显示的操作相关的模块，编写激励程序验证是否符合设计要求。  张鑫：设计出无符号位的算法，再根据有符号位加减法的规则设计并编程完成有符号位的加减法相关的模块，编写激励程序验证是否符合设计要求。 | | | | | |