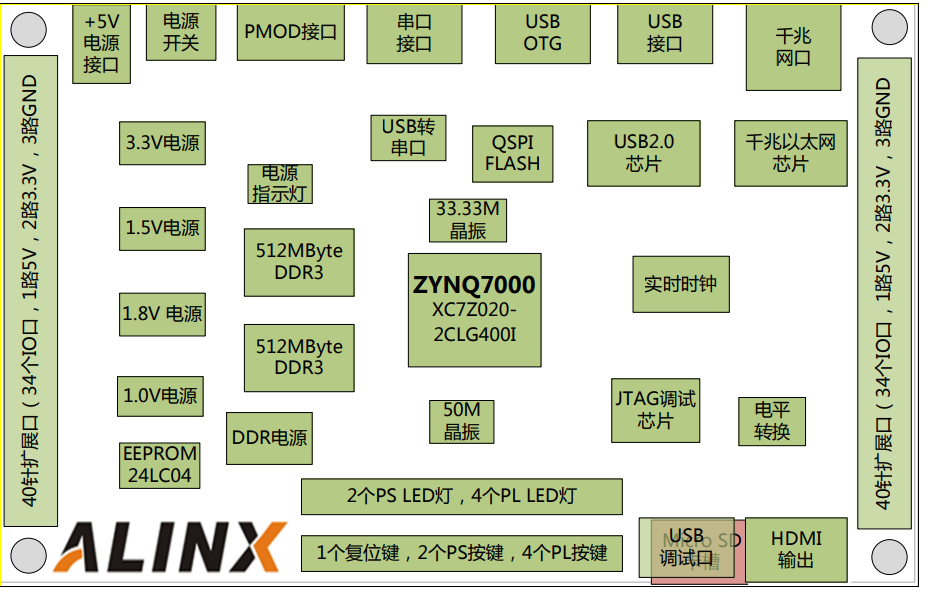
2024-3-31

MSD加法与电子加法计算时间对比实验

## 1实验用具

针对此次试验我们采用Alinx7020，此款开发板使用的是 Xilinx 公司的 Zynq7000 系列的芯片，型号为 XC7Z020-2CLG400I， 400 个引脚的 FBGA 封装。ZYNQ7000 芯片可分成处理器系统部分 Processor System（PS） 和可编程逻辑部分 Programmable Logic（PL）。在 AX7020 开发板上，ZYNQ7000 的 PS 部分和 PL 部分都搭载了丰富的外部接口和设备，方便用户的使用和功能验证。另外开发板上 集成了 Xilinx USB Cable 下载器电路，用户只要用一个 USB 线就可以对开发板进行下载和调 试。图1为整个 AX7020 整个系统的结构示意图:



通过这个示意图，我们可以看到，我们这个开发平台所能含有的接口和功能。

下面是实物展示：

## **实验目的**

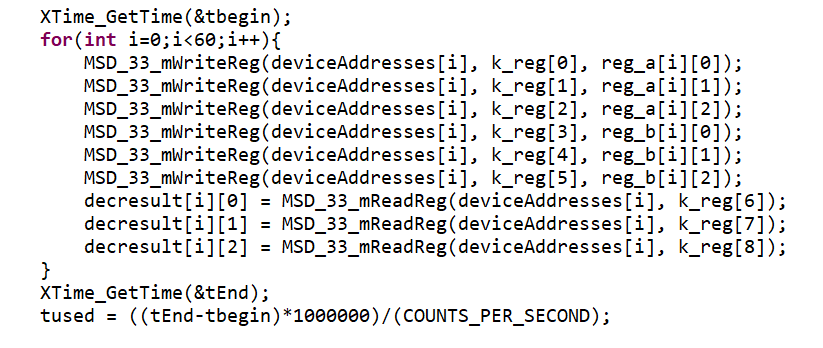
针对两种电子加法器的性能测试以及对比，我们将使用XC7Z020-2CLG400I型FPGA开发板上的资源。首先，我们会利用该板上的双核ARM Cortex-A9中的传统加法器，该加法器为32位，代表第一种加法器。接着，我们将利用可编程逻辑单元阵列（PL）在同一开发板上构建多个33位的最高有效位（MSD）加法器，以代表第二种加法器。

在测试过程中，我们将记录并比较两种加法器的计算用时。对于第一种加法器，我们将测试其处理一个32位数据的加法所需的时间。而对于第二种加法器，我们会测试其处理多个33位数据的加法所需的时间，并且注意到由于其并行处理的特性，其计算时间可能会显著减少。

通过这些测试，我们将能够直观地评估两种加法器的性能优劣，进而了解到是否值得投资于构建更复杂但性能更优异的MSD加法器。

**期望的实验结果：**第2种加法器的工作用时短。即：第2种加法器快。

## **实验准备**

1. **时间计算**：在 Xilinx 提供的库中，**xtime\_l.h** 是与时间管理相关的一个头文件，用于处理时间和延时功能。XTime\_GetTime(XTime Xtime) 获取当前时间，结果存储在 Xtime 中。通过计算时间差再乘以时钟周期获取到准确时间。Alinx的频率为50Mhz对应时钟周期是20ns。
2. **MSD加法器并行原理**：在可编程逻辑 (PL) 端，我们采用信号机制来协调多个运算单元的操作。具体来说，当 60 个多位数数字 (MSD) 加法器完成填充操作后，系统会通过一个同步信号触发所有加法器并行开始计算。这种方式确保了所有加法器同时进行运算，提高了整体计算效率。最终，所有加法器的计算结果会统一返回，确保数据处理的一致性和同步性。

## **实验策略**

1. 利用第二种加法器的“可重构出任意大小加法器”的特性，我们构建了一个33位的最高有效位（MSD）加法器。这样，我们可以直接对33位数据进行加法运算，而第一种加法器则需要两次计算才能完成一个加法运算：首先计算低32位，然后计算高32位（尽管实验数据中仅有一位）。相比之下，第二种加法器只需一次计算就能完成33位数据的加法。
2. 利用第二种加法器的“可重构出多个加法器”的优势，我们构建了60个33位的MSD加法器，并使它们能够并行工作。这意味着我们可以一次性计算60个33位数的加法，而第一种加法器则只能逐个计算33位数的加法（每次加法还需要两次计算）

## 实验过程

第一步：数据准备。我们将60对原始数据送入内存。数据范围是2^32-2^33，范围从 4294967295 到 8589934591

data\_a= [

6163550658, 6413109763, 4493162740, 4459676060, 6148214032,

5439006107, 4433876780, 5861763895, 5221972308, 4935524470,

5389687797, 4354296534, 4790236573, 6306732821, 5654945182,

6069373681, 4870013515, 6180308189, 6221500578, 6329734393,

5090246305, 5522595258, 5387336040, 4360315825, 5515023509,

5973410702, 5784501258, 5232857401, 6019818661, 4846029805,

5046807674, 6106273973, 4637160715, 5392211615, 5555725575,

5791996815, 5837229290, 5778789187, 5203953563, 4309582241,

6168520261, 5470611017, 4386111010, 6026674767, 5689423199,

4785091805, 4964832056, 5654890046, 4737018386, 5437568557,

6000280530, 6126097812, 5537764560, 6096738408, 5350641731,

5545828808, 6339368421, 6375512686, 6240230742, 5127674579]

data\_b= [

5670367407, 5412345023, 5914847405, 6128327586, 4587896338,

5026686114, 4935222256, 4760041748, 4614409077, 4377978840,

5175819138, 5634001496, 5947783309, 4908543247, 5960452096,

5591331536, 4727949047, 5955264796, 5970133931, 6321056983,

6073297128, 4455663635, 4521584003, 6273163952, 5632439975,

6265859821, 6278962862, 5399591672, 6238104244, 6005097228,

5727386570, 6035573062, 4814809633, 5901575191, 4373607691,

4403649190, 6153957194, 6109318345, 4673901860, 4906714130,

4635253234, 5761701894, 6103554071, 5135616837, 4835360414,

5610035924, 5553855281, 5663039910, 4390156807, 6134636813,

4409012067, 5272496534, 5612425667, 4960474159, 4752740764,

5935161763, 6080424745, 5902262331, 4863341116, 5603599112]

第二步：加法运算。在第一种加法器上，需要重复执行60次加法指令。而在第二种加法器上，由于其并行处理能力，我们只需重复执行1次加法指令（运算），每次同时计算60对位数据。

1. **第一种加法器：**

**for** (**int** i = **0**; i < NUM\_BINARIES; i++)

{

binary\_add(bin\_nums\_a, bin\_nums\_b, bin\_result, i);

}

1. **第二种加法器：**

MSD\_33\_mWriteReg(deviceAddresses[i], k\_reg[**0**], reg\_a[i][**0**]);

MSD\_33\_mWriteReg(deviceAddresses[i], k\_reg[**1**], reg\_a[i][**1**]);

MSD\_33\_mWriteReg(deviceAddresses[i], k\_reg[**2**], reg\_a[i][**2**]);

MSD\_33\_mWriteReg(deviceAddresses[i], k\_reg[**3**], reg\_b[i][**0**]);

MSD\_33\_mWriteReg(deviceAddresses[i], k\_reg[**4**], reg\_b[i][**1**]);

MSD\_33\_mWriteReg(deviceAddresses[i], k\_reg[**5**], reg\_b[i][**2**]);

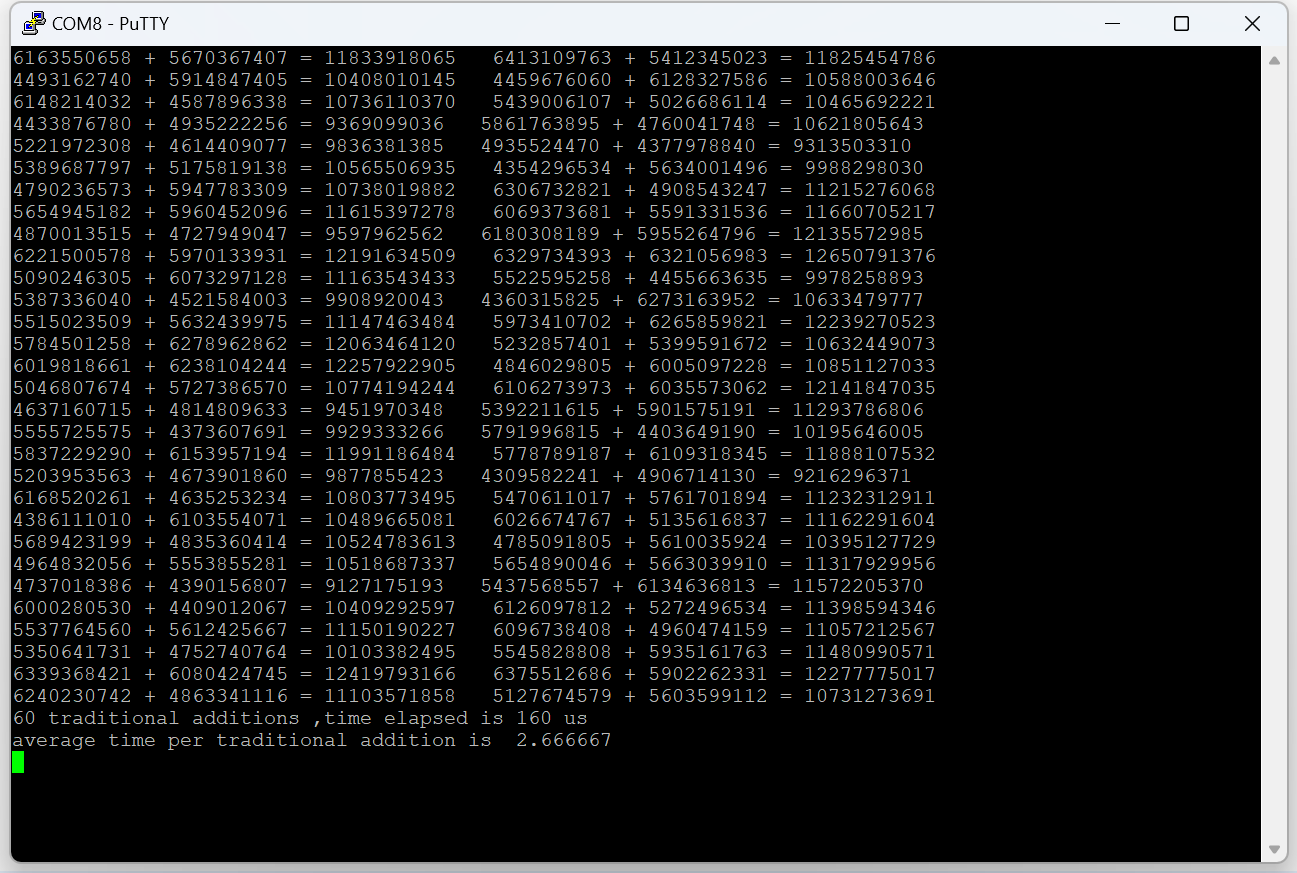
decresult[i][**0**] = MSD\_33\_mReadReg(deviceAddresses[i], k\_reg[**6**]);

decresult[i][**1**] = MSD\_33\_mReadReg(deviceAddresses[i], k\_reg[**7**]);

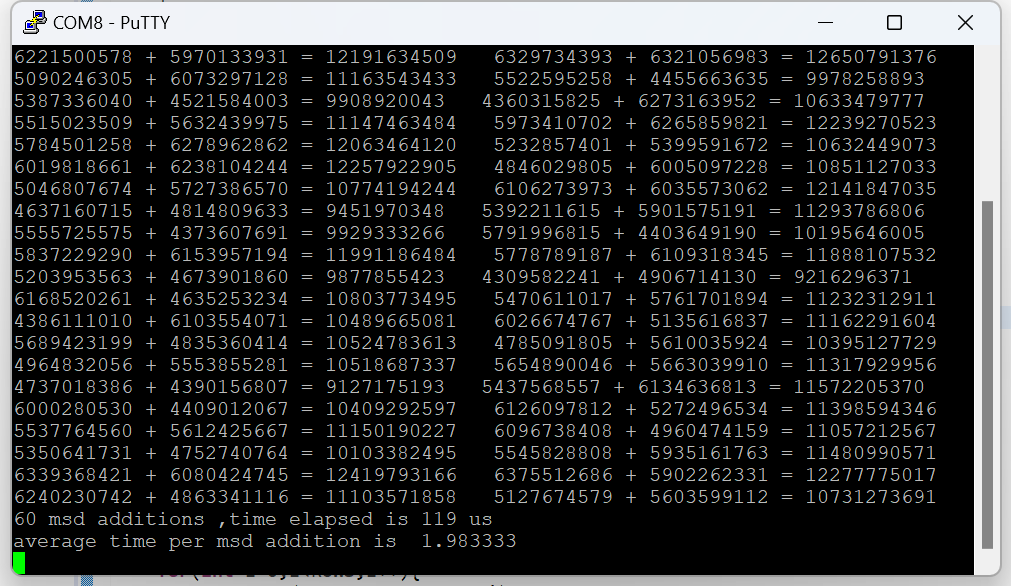
decresult[i][**2**] = MSD\_33\_mReadReg(deviceAddresses[i], k\_reg[**8**]);

第三步：显示运算结果。第二步的运算结果会先存储在内存中。当60个数据计算完毕后，再执行第三步。屏幕将显示如下信息：

第一种加法器实验结果：



第二种加法器：



数据对比见MSD数据对比.xlsx

## 附录

### MSD加法代码

**#include** "stdio.h"

**#include** "stdlib.h"

**#include** "string.h"

**#include** "xparameters.h"

**#include** "msd\_33.h"

**#include** "xil\_io.h"

**#include** "sleep.h"

**#include** "stdlib.h"

**#include** "stdint.h"

**#include** "math.h"

**#include** "xtime\_l.h"

**#define** max(a, b) ((a) > (b) ? (a) : (b))

**#define** START\_ADDRESS 0x43c00000

**#define** END\_ADDRESS 0x43FB0000

**#define** STEP\_SIZE 0x10000 // 假设步长为 0x10000

**#define** ROWS 60

**#define** COLS 3

**#define** ARRAY\_SIZE 60

**long** **long**\* **generate\_random\_array**(**long** **long**\* array, **int** size, **int** random\_seed) {

**long** **long** lower\_bound = (1LL << 32) - 1; // 2^32 - 1

**long** **long** upper\_bound = (1LL << 33) - 1; // 2^33 - 1

**srand**(random\_seed);

// 生成随机数并存储在数组中

**for** (**int** i = 0; i < size; ++i) {

array[i] = ((**long** **long**)**rand**() % (upper\_bound - lower\_bound + 1)) + lower\_bound;

}

**return** array;

}

**void** **decToBin**(**long** **long** dec, **int** bin[])

{

**if** (dec < 0)

{

dec \*= -1;

bin[0] = 1; // 0代表正，1代表负

}

**else**

{

bin[0] = 0;

}

**long** **long** num = dec;

**int** temp[99];

**int** i;

**for** (i = 0; num != 0; i++)

{

temp[i] = num % 2; // 使用 % 运算符计算余数

num /= 2;

}

**for** (**int** j = 1, index = i - 1; j <= i; j++, index--)

{

bin[j] = temp[index];

}

bin[i + 1] = -2; // 结束标志

}

**void** **binToMsd**(**int** bin[], **int** msd[])

{

**int** index\_msd = 0;

**if** (bin[0] == 1)

{

**for** (**int** index\_bin = 1; bin[index\_bin] != -2; index\_bin++, index\_msd++)

{

msd[index\_msd] = 0;

**if** (bin[index\_bin] == 1)

msd[index\_msd] = -1;

}

msd[index\_msd] = -2; // 结束标志

}

**else**

{

**for** (**int** index\_bin = 1; bin[index\_bin] != -2; index\_bin++, index\_msd++)

{

msd[index\_msd] = bin[index\_bin];

}

msd[index\_msd] = -2; // 结束标志

}

}

**int** **bin\_to\_dec**(**char** bin[], **int** len)

{

**int** result = 0, base = 1;

**for** (**int** i = len - 1; i >= 0; i--, base = base \* 2)

result = result + (bin[i] - '0') \* base;

**return** result;

}

**long** **long** **msdToDec**(**int** msd[])

{

**long** **long** result = 0;

**int** len = 0;

**for** (; msd[len] != -2; len++)

;

**for** (**int** i = 0, j = len - 1; i < len; i++, j--)

{

**long** **long** t = 1;

**for** (**int** k = 0; k < j; k++)

{

t \*= 2;

}

result += msd[i] \* t;

}

**return** result;

}

**void** **convertToBinaryString**(**int** \*msd, **char** \*result)

{

**int** i = 0;

**while** (msd[i] != -2)

{

**if** (msd[i] == 0)

{

**strcat**(result, "00");

}

**else** **if** (msd[i] == 1)

{

**strcat**(result, "01");

}

**else** **if** (msd[i] == -1)

{

**strcat**(result, "10");

}

i++;

}

}

**void** **binaryStringToMSD**(**char** \*binaryString, **int** \*intArray, **int** length)

{

**for** (**int** i = 0; i < length; i++)

{

**char** tmp[3];

**strncpy**(tmp, binaryString + i \* 2, 2);

tmp[2] = '\0';

**if** (**strcmp**(tmp, "00") == 0)

{

intArray[i] = 0;

}

**else** **if** (**strcmp**(tmp, "01") == 0)

{

intArray[i] = 1;

}

**else** **if** (**strcmp**(tmp, "10") == 0)

{

intArray[i] = -1;

}

// Add more conditions as needed

}

}

**void** **cdecToBin**(**unsigned** value, **int** bits, **char** \*binaryArray)

{

**for** (**int** i = bits - 1; i >= 0; i--)

{

binaryArray[bits - 1 - i] = ((value >> i) & 1) + '0';

}

binaryArray[bits] = '\0'; // 添加 null 终止符

}

**void** **write\_reg**(**unsigned** **int** array[][3], **long** **long** value,**int** row) {

**int** bin[36];

**int** msd[36];

decToBin(value, bin);

binToMsd(bin, msd);

**char** z[67] = "";

convertToBinaryString(msd, z);

**int** len\_z = **strlen**(z);

**if** (len\_z <= 32)

{

array[row][0] = bin\_to\_dec(z, len\_z);

}

**else** **if** (len\_z > 32 && len\_z <= 64)

{

array[row][0] = bin\_to\_dec(z + len\_z - 32, 32);

array[row][1] = bin\_to\_dec(z, len\_z - 32);

}

**else** **if** (len\_z > 64 && len\_z <= 96)

{

array[row][0] = bin\_to\_dec(z + len\_z - 32, 32);

array[row][1] = bin\_to\_dec(z + len\_z - 64, 32);

array[row][2] = bin\_to\_dec(z, len\_z - 64);

}

}

**void** **printUnsignedArray**(**unsigned** **int** array[][3], **int** rows, **int** cols) {

**for** (**int** i = 0; i < rows; ++i) {

**for** (**int** j = 0; j < cols; ++j) {

**printf**("%u ", array[i][j]);

}

**printf**("\n");

}

}

**void** **regToDec**(**unsigned** **int** decresult[][3],**long** **long** \* ansDec,**int** index){

**char** ans\_c[70] = "";

cdecToBin(decresult[index][2], 6, ans\_c);

cdecToBin(decresult[index][1], 32, ans\_c + 6);

cdecToBin(decresult[index][0], 32, ans\_c + 38);

**int** msd\_c[40];

**int** length = **strlen**(ans\_c) / 2; // Assuming the length is even

binaryStringToMSD(ans\_c, msd\_c, length);

msd\_c[length] = -2;

ansDec[index] = msdToDec(msd\_c);

}

**void** **printLongLongArray**(**long** **long** array[], **int** length) {

**for** (**int** i = 0; i < length; ++i) {

**printf**("%lld ", array[i]);

}

**printf**("\n");

}

**int** **main**() {

**long** **long** array\_a[ARRAY\_SIZE];

**long** **long** array\_b[ARRAY\_SIZE];

**int** arrayLength = (END\_ADDRESS - START\_ADDRESS) / STEP\_SIZE + 1;

**int** deviceAddresses[arrayLength];

**int** currentAddress = START\_ADDRESS;

**for** (**int** i = 0; i < arrayLength; ++i) {

deviceAddresses[i] = currentAddress;

currentAddress += STEP\_SIZE;

}

XTime tEnd, tbegin;

u32 tused;

**long** **long**\* data\_a = generate\_random\_array(array\_a, ARRAY\_SIZE,123);

**long** **long**\* data\_b = generate\_random\_array(array\_b, ARRAY\_SIZE,456);

**unsigned** **int** reg\_a[ROWS][COLS] = { {0} };

**unsigned** **int** reg\_b[ROWS][COLS] = { {0} };

**for**(**int** i=0;i<ROWS;i++){

write\_reg(reg\_a,data\_a[i],i);

write\_reg(reg\_b,data\_b[i],i);

}

**int** k\_reg[9]={0};

**unsigned** **int** decresult[ROWS][COLS] = { {0} };

**for** (**int** k = 0; k < 9; k++)

{

k\_reg[k] = k \* 4;

}

XTime\_GetTime(&tbegin);

**for**(**int** i=0;i<60;i++){

MSD\_33\_mWriteReg(deviceAddresses[i], k\_reg[0], reg\_a[i][0]);

MSD\_33\_mWriteReg(deviceAddresses[i], k\_reg[1], reg\_a[i][1]);

MSD\_33\_mWriteReg(deviceAddresses[i], k\_reg[2], reg\_a[i][2]);

MSD\_33\_mWriteReg(deviceAddresses[i], k\_reg[3], reg\_b[i][0]);

MSD\_33\_mWriteReg(deviceAddresses[i], k\_reg[4], reg\_b[i][1]);

MSD\_33\_mWriteReg(deviceAddresses[i], k\_reg[5], reg\_b[i][2]);

decresult[i][0] = MSD\_33\_mReadReg(deviceAddresses[i], k\_reg[6]);

decresult[i][1] = MSD\_33\_mReadReg(deviceAddresses[i], k\_reg[7]);

decresult[i][2] = MSD\_33\_mReadReg(deviceAddresses[i], k\_reg[8]);

}

XTime\_GetTime(&tEnd);

tused = ((tEnd-tbegin)\*1000000)/(COUNTS\_PER\_SECOND);

**long** **long** ansDec[ROWS] = {0};

**for**(**int** i=0;i<ROWS;i++){

regToDec(decresult,ansDec,i);

}

**for**(**int** i=0;i<60;i++){

**printf**("%lld + %lld = %lld ",data\_a[i],data\_b[i],ansDec[i]);

**if**((i+1)%2==0){

**printf**("\n");

}

}

xil\_printf("%d msd additions ,time elapsed is %d us\r\n",ROWS,tused);

**float** tused\_float = (**float**)tused / 60.0;

**printf**("average time per msd addition is %f\n", tused\_float);

**return** 0;

}

### 传统计算器加法

**#include** <stdio.h>

**#include** <stdlib.h>

**#include** <time.h>

**#include** <math.h>

**#include** "xtime\_l.h"

**#define** NUM\_BINARIES 60

**#define** MAX\_BIT\_LENGTH 34

// 十进制转二进制 数组低位存二进制低位

**void** **decimal\_to\_binary**(**long** **long** decimal\_num, **int** binary\_num[][MAX\_BIT\_LENGTH], **int** index)

{

**for** (**int** i = 0; i < MAX\_BIT\_LENGTH; i++)

{

binary\_num[index][i] = decimal\_num % 2;

decimal\_num /= 2;

}

}

**void** **binary\_to\_decimal**(**int** binary[][MAX\_BIT\_LENGTH], **long** **long** \*result, **int** index)

{

**long** **long** decimal = 0;

**for** (**int** i = 0; i < MAX\_BIT\_LENGTH; ++i)

{

**long** **long** binary\_bit = binary[index][i]; // 获取当前位的二进制值

**if** (binary\_bit)

{

decimal += (1LL << i); // 将当前位的二进制值乘以对应的 2 的次方，并累加到十进制数中

}

}

result[index] = decimal;

}

**void** **binary\_add**(**int** binary1[][MAX\_BIT\_LENGTH], **int** binary2[][MAX\_BIT\_LENGTH], **int** result\_binary[][MAX\_BIT\_LENGTH], **int** index)

{

**int** carry = 0;

**for** (**int** i = 0; i < MAX\_BIT\_LENGTH; i++)

{

**int** sum = binary1[index][i] + binary2[index][i] + carry;

result\_binary[index][i] = sum % 2;

carry = sum / 2;

}

}

**long** **long** \***generate\_random\_array**(**long** **long** \*array, **int** size, **int** random\_seed)

{

**long** **long** lower\_bound = (1LL << 32) - 1; // 2^32 - 1

**long** **long** upper\_bound = (1LL << 33) - 1; // 2^33 - 1

**srand**(random\_seed);

// 生成随机数并存储在数组中

**for** (**int** i = 0; i < size; ++i)

{

array[i] = ((**long** **long**)**rand**() % (upper\_bound - lower\_bound + 1)) + lower\_bound;

}

**return** array;

}

**void** **printArray**(**int** array[][34], **int** rows, **int** cols)

{

**for** (**int** i = 0; i < rows; ++i)

{

**for** (**int** j = 0; j < cols; ++j)

{

**printf**("%d", array[i][j]);

}

**printf**("\n");

}

}

**int** **main**()

{

**long** **long** array\_a[NUM\_BINARIES];

**long** **long** array\_b[NUM\_BINARIES];

**int**(\*bin\_nums\_a)[MAX\_BIT\_LENGTH] = **malloc**(NUM\_BINARIES \* **sizeof**(\*bin\_nums\_a));

**int** bin\_nums\_b[NUM\_BINARIES][MAX\_BIT\_LENGTH]={{0}};

**int**(\*bin\_result)[MAX\_BIT\_LENGTH] = **malloc**(NUM\_BINARIES \* **sizeof**(\*bin\_result));

XTime tEnd, tbegin;

u32 tused;

**for** (**int** i = 0; i < NUM\_BINARIES; i++) {

**for** (**int** j = 0; j < MAX\_BIT\_LENGTH; j++) {

bin\_nums\_a[i][j] = 0;

bin\_nums\_b[i][j] = 0;

bin\_result[i][j] = 0;

}

}

**long** **long** dec\_result[NUM\_BINARIES]={0};

// 生成60个随机的二进制数

**long** **long** \*data\_a = generate\_random\_array(array\_a, NUM\_BINARIES, 123);

**long** **long** \*data\_b = generate\_random\_array(array\_b, NUM\_BINARIES, 456);

**for** (**int** i = 0; i < NUM\_BINARIES; i++)

{

decimal\_to\_binary(data\_a[i], bin\_nums\_a, i);

decimal\_to\_binary(data\_b[i], bin\_nums\_b, i);

}

XTime\_GetTime(&tbegin);

**for** (**int** i = 0; i < NUM\_BINARIES; i++)

{

binary\_add(bin\_nums\_a, bin\_nums\_b, bin\_result, i);

}

XTime\_GetTime(&tEnd);

tused = ((tEnd-tbegin)\*1000000)/(COUNTS\_PER\_SECOND);

**for** (**int** i = 0; i < NUM\_BINARIES; i++)

{

binary\_to\_decimal(bin\_result, dec\_result, i);

}

**for** (**int** i = 0; i < 60; i++)

{

**printf**("%lld + %lld = %lld ", data\_a[i], data\_b[i], dec\_result[i]);

**if** ((i + 1) % 2 == 0)

{

**printf**("\n");

}

}

xil\_printf("%d traditional additions ,time elapsed is %d us\r\n",NUM\_BINARIES,tused);

**float** tused\_float = (**float**)tused / 60.0;

**printf**("average time per traditional addition is %f\n", tused\_float);

**free**(bin\_nums\_a);

**free**(bin\_result);

**return** 0;

}