

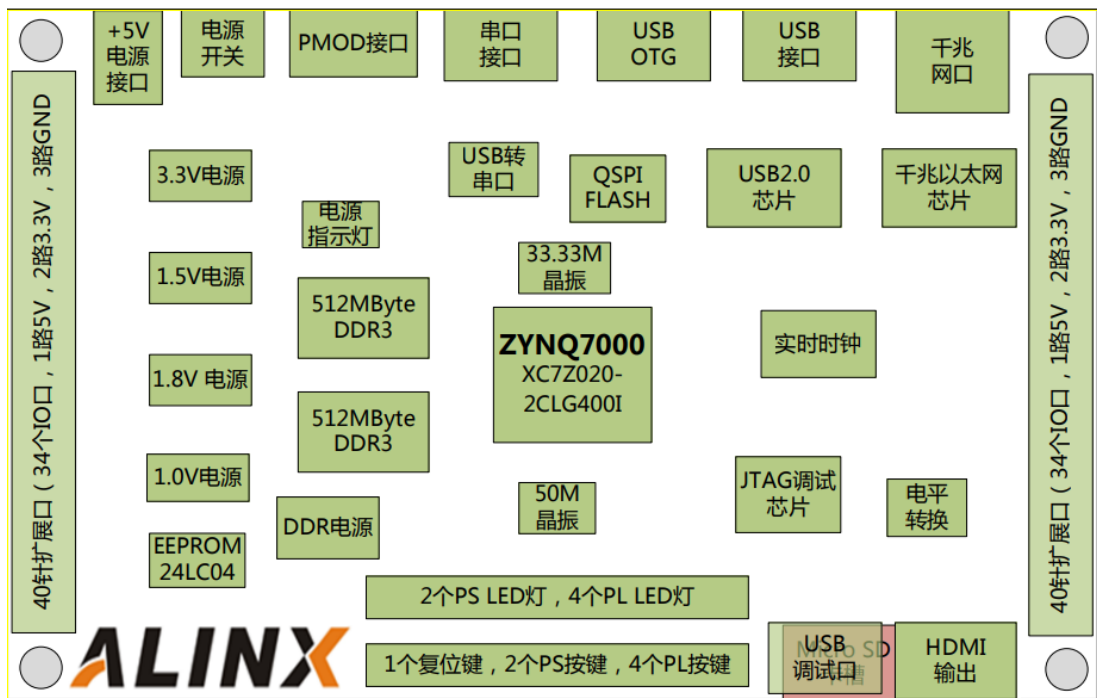
2024-3-31

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。

- 1.+5V 电源输入,最大 2A 电流保护;
- 2.Xilinx ARM+FPGA 芯片 Zynq-7000 XC7Z020-2CLG400I
- 3.两片大容量的 4Gbit (共 8Gbit) 高速 DDR3 SDRAM,可作为 ZYNQ 芯片数据的缓存, 也可以作为操作系统运行的内存;
- 4.一片 256Mbit 的 QSPI FLASH, 可用作 ZYNQ 芯片的系统文件和用户数据的存储;
- 5.一路 10/100M/1000M 以太网 RJ-45 接口, 可用于和电脑或其它网络设备进行以太网

数据交换;

6.一路 HDMI 图像视频输入输出接口, 能实现 1080P 的视频图像传输;

7.一路高速 USB2.0 HOST 接口, 可用于开发板连接鼠标、键盘和 U 盘等 USB 外设;

8. 一路高速 USB2.0 OTG 接口, 用于和 PC 或 USB 设备的 OTG 通信;

9.一路 USB Uart 接口, 用于和 PC 或外部设备的串口通信;

10.一片的 RTC 实时时钟, 配有电池座, 电池的型号为 CR1220。一片 IIC 接口的 EEPROM 24LC04;

11.6 个用户发光二极管 LED, 2 个 PS 控制, 4 个 PL 控制;

12.7 个按键, 1 个 CPU 复位按键, 2 个 PS 控制按键, 4 个 PL 控制按键;

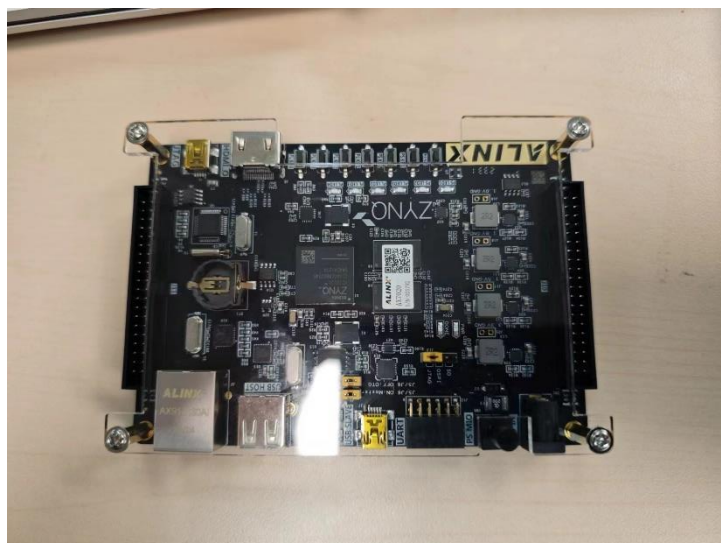
13.板载一个 33.333Mhz 的有源晶振, 给 PS 系统提供稳定的时钟源, 一个 50MHz 的有源晶振, 为 PL 逻辑提供额外的时钟;

14.2 路 40 针的扩展口 (2.54mm 间距), 用于扩展 ZYNQ 的 PL 部分的 IO。可以接 7 寸 TFT 模块、摄像头模块和 AD/DA 模块等扩展模块;

15.一个 12 针的扩展口 (2.54mm 间距), 用于扩展 ZYNQ 的 PS 系统的 MIO;

16.一路 USB JTAG 口, 通过 USB 线及板载的 JTAG 电路对 ZYNQ 系统进行调试和下载。1 路 Micro SD 卡座(开发板背面), 用于存储操作系统镜像和文件系统

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]
```


第二种加法器:

```
COM - PUTTY
6143552436 + 5670367407 = 11833918045 6413109763 + 5412345023 = 11825454786
4493161740 + 5914847405 = 10408015145 6459676060 + 6128375756 = 12584805946
6148214032 + 4587896338 = 10736110370 5419086107 + 5024686114 = 10465492221
1453873780 + 4585222256 = 5246999936 5861763995 + 4760041740 = 1062105643
5221972308 + 4614409077 = 9836381385 4835524470 + 4377978040 = 9313503310
338087797 + 5175919130 = 1056506935 4354290534 + 5634001496 = 9988298030
4790236573 + 5987733308 = 10738058885 4306735921 + 4908515747 = 11215776968
5854945182 + 5988452096 = 11615397278 4089373681 + 5591311536 = 11660705217
4870013516 + 4727980847 = 9597624362 6180388189 + 5955264796 = 12135572885
6221509578 + 5970133931 = 12191634509 6329734393 + 6321084993 = 12650793376
6996246308 + 6073297129 = 11183543433 5522398258 + 4465868635 = 9988298030
5387336040 + 4521584003 = 9908920043 4360315825 + 6273163952 = 10633479777
5511023508 + 5632439975 = 11147463484 5973410702 + 6265858921 = 12239278523
3784501285 + 6278962862 = 12063464120 5232857401 + 5398951672 = 10633480713
6018618461 + 6288104244 = 12257922905 4846029805 + 6005097228 = 10851127033
546889474 + 5727386530 = 1074184244 4106278978 + 6035573062 = 12141443038
4637160715 + 4814809633 = 9451970348 5392216115 + 5901576191 = 11293768806
5557225575 + 4373607681 = 9929333266 5791996815 + 4403649190 = 10195646805
5872220390 + 6133907194 = 11991186084 5771780187 + 6109315345 = 1188101532
5203953563 + 4673901860 = 9877855423 4309582241 + 4906714130 = 9216296371
1168582861 + 468528324 = 16803973496 4470611012 + 5781701894 = 11232312916
4386111010 + 6103554071 = 10489655081 6026674767 + 5135616837 = 11162291604
5689423159 + 4835360414 = 10524793613 4785091805 + 5610035524 = 10395127729
4944312064 + 5513815211 = 1045697337 5654890844 + 5643539910 = 11317829564
4737018386 + 4380166807 = 9127175193 5437548557 + 6134636813 = 11572205370
6080220538 + 4489012067 = 1040922657 4124097812 + 5272486534 = 11398584346
5537764560 + 5612425667 = 11150190227 6096738408 + 4960474159 = 11057212567
2353641731 + 4752740764 = 1010382495 5545628808 + 5935121763 = 11480996571
6339364421 + 6003424745 = 12419793166 6375515686 + 5902262331 = 12277778017
6240230742 + 4863341116 = 11103571858 5127674579 + 5603599112 = 10731273691
60 mod additions, time elapsed is 239 us
average time per mod addition is 3.983333
```

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]);
decreslt[i][0] = MSD_33_mReadReg(deviceAddresses[i], k_reg[6]);
decreslt[i][1] = MSD_33_mReadReg(deviceAddresses[i], k_reg[7]);
decreslt[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(decreslt,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_1.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;
}

```