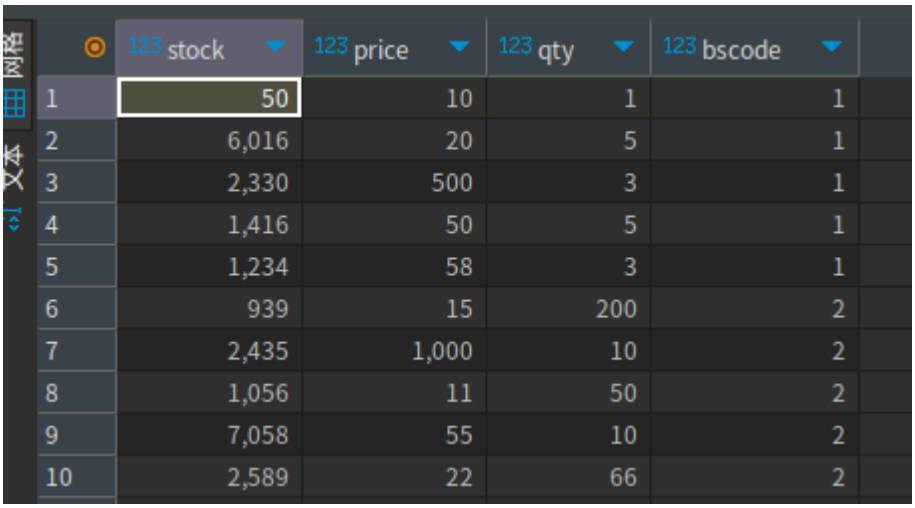
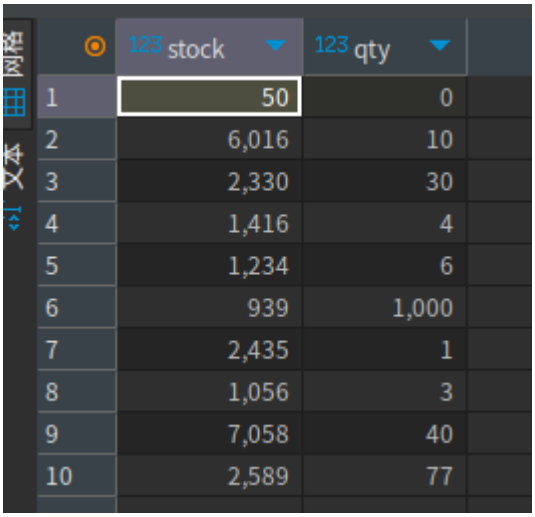
報告

委託



	stock	price	qty	bscode
1	50	10	1	1
2	6,016	20	5	1
3	2,330	500	3	1
4	1,416	50	5	1
5	1,234	58	3	1
6	939	15	200	2
7	2,435	1,000	10	2
8	1,056	11	50	2
9	7,058	55	10	2
10	2,589	22	66	2

股票庫存



	stock	qty
1	50	0
2	6,016	10
3	2,330	30
4	1,416	4
5	1,234	6
6	939	1,000
7	2,435	1
8	1,056	3
9	7,058	40
10	2,589	77

HLS

```
#include <ap_int.h>
#include <hls_stream.h>
#include <iostream>
#include <stdio.h>
using namespace hls;

#define stock_len 14
#define qty_len 14
#define price_len 20
#define bs_len 2
#define order_len stock_len + qty_len + price_len + bs_len

typedef ap_uint<order_len> order_t;
```

```
typedef ap_uint<stock_len> stock_t;
typedef ap_uint<qty_len> qty_t;
typedef ap_uint<price_len> price_t;
typedef ap_uint<bs_len> bs_t;

// 預設最大處理數量
const int customers_max = 50;

// 讀取客戶資料
static void read_input(order_t *customer_data, hls::stream<order_t>
&inStream, int number_of_customer);

// 判斷資料
static void compare(int *initdata, hls::stream<order_t> &inStream,
hls::stream<int> &outStream, int number_of_customer);

// 回傳對應的結果
static void write_result(int *customer_ans, hls::stream<int> &outStream,
int number_of_customer);

extern "C"
{
    void riskcontrol(order_t *customer_data, int *customer_ans, int
customers_number);
}
```

```
#include "hls.hpp"
#include <ap_int.h>
#include <hls_stream.h>
using namespace hls;

// 讀取客戶資料
static void read_input(order_t *customer_data, hls::stream<order_t>
&inStream, int number_of_customer)
{
mem_rd:
    for (int i = 0; i < number_of_customer; i++)
    {
#pragma HLS LOOP_TRIPCOUNT min = customers_max max = customers_max
        inStream << customer_data[i];
    }
}

// 判斷
static void compare(int *initdata, hls::stream<order_t> &inStream,
hls::stream<int> &outStream, int number_of_customer)
{
execute:
    int money = 0;
    for (int i = 0; i < number_of_customer; i++)
    {
```

```
#pragma HLS LOOP_TRIPCOUNT min = customers_max max = customers_max
        order_t order = inStream.read();
        stock_t tempstock = order(stock_len + price_len + qty_len + bs_len
- 1, price_len + qty_len + bs_len);
        price_t tempprice = order(price_len + qty_len + bs_len - 1,
price_len);
        qty_t tempqty = order(qty_len + bs_len - 1, qty_len);
        bs_t tempbs = order(bs_len - 1, 0);
        money = tempqty * tempprice;
        if (i == 0)
            printf("dataassign i=%d 資料: [%d] stock: [%d] price: [%d] qty:
[%d] bs: [%d]\n", i, order, tempstock, tempprice, tempqty, tempbs);

        if (tempbs == 1)
        { // 買單檢查額度
            if (5000000 < money)
                outStream.write(1);
            else
                outStream.write(0);
        }
        else
        { // 賣單檢查庫存
            if (initdata[i] < tempqty)
                outStream.write(1);
            else
                outStream.write(0);
        }
    }
}

// 回傳對應的結果
static void write_result(int *customer_ans, hls::stream<int> &outStream,
int number_of_customer)
{
mem_wr:
    for (int i = 0; i < number_of_customer; i++)
    {
#pragma HLS LOOP_TRIPCOUNT min = customers_max max = customers_max
        customer_ans[i] = outStream.read();
    }
}

extern "C"
{
    void riskcontrol(order_t *customer_data, int *customer_ans, int
customers_number)
    {
#pragma HLS INTERFACE m_axi port = customer_data bundle = gmem0 depth = 32
#pragma HLS INTERFACE m_axi port = customer_ans bundle = gmem0 depth = 32
        static hls::stream<order_t> customers_in("input_stream");
        static hls::stream<int> customers_out("output_stream");
        int initdata[] = {0, 10, 30, 4, 6, 1000, 1, 3, 40, 77};
#pragma HLS STREAM variable = customers_in depth = 32
#pragma HLS STREAM variable = customers_out depth = 32
```

```
#pragma HLS dataflow
        read_input(customer_data, customers_in, customers_number);
        compare(initdata, customers_in, customers_out, customers_number);
        write_result(customer_ans, customers_out, customers_number);
    }
}
```

HOST

```
#include <iostream>
#include <string>
#include <cstring>
#include <sstream>
#include <map>
#include <vector>
#include <algorithm>
#include <numeric> // For std::transform_reduce
#include <iomanip> // For std::setw
#include <mysql.h>

#include "cmdlineparser.h"

// FPGA 相關
#include "experimental/xrt_bo.h"
#include "experimental/xrt_device.h"
#include "experimental/xrt_kernel.h"

using namespace std;

#define DATA_SIZE 256

MYSQL *conndb;

struct Order
{
    int stockno;
    int price;
    int qty;
    int bs;
};

struct Limit
{
    int stockno;
    int qty;
};

Order sw_orderlist[10];
Limit sw_limitlist[10];
```

```
int ConnectDB()
{
    conndb = mysql_init(NULL);
    mysql_options(conndb, MYSQL_OPT_NONBLOCK, 0);
    if (!mysql_real_connect(conndb, "192.168.199.235", "crcft", "Aa1234",
"fpgatest", 3306, NULL, 0))
    {
        cout << "連接資料庫錯誤" << endl;
        mysql_close(conndb);
        return -1;
    }
    return 0;
}

int QueryOrder(int64_t *list)
{
    int i = 0;
    MYSQL_RES *res;
    MYSQL_ROW row;
    stringstream sql_query;
    sql_query.str(""); // 字串流清零，將流中的資料全部清除
    sql_query.clear();
    sql_query << "SELECT * FROM `Order`";
    if (mysql_real_query(conndb, sql_query.str().c_str(),
sql_query.str().length()))
    {
        cout << "QueryOrder ERROR: " << string(mysql_error(conndb)) <<
endl;
        cout << "ERROR query:" << sql_query.str() << endl;
        return -1;
    }

    res = mysql_use_result(conndb);
    Order order_obj;
    string key;
    while ((row = mysql_fetch_row(res)) != NULL)
    {

        order_obj.stockno = atoi(row[0]);
        order_obj.price = atoi(row[1]);
        order_obj.qty = atoi(row[2]);
        order_obj.bs = atoi(row[3]);
        std::stringstream ss;
        ss.str("");
        ss.clear();
        ss << order_obj.stockno << order_obj.price << order_obj.qty <<
order_obj.bs;
        cout << "Order data:[" << ss.str() << "]" << endl;
        list[i] = atoll(ss.str().c_str());
        sw_orderlist[i] = order_obj;
        i++;
    }
    mysql_free_result(res);
    return 0;
```

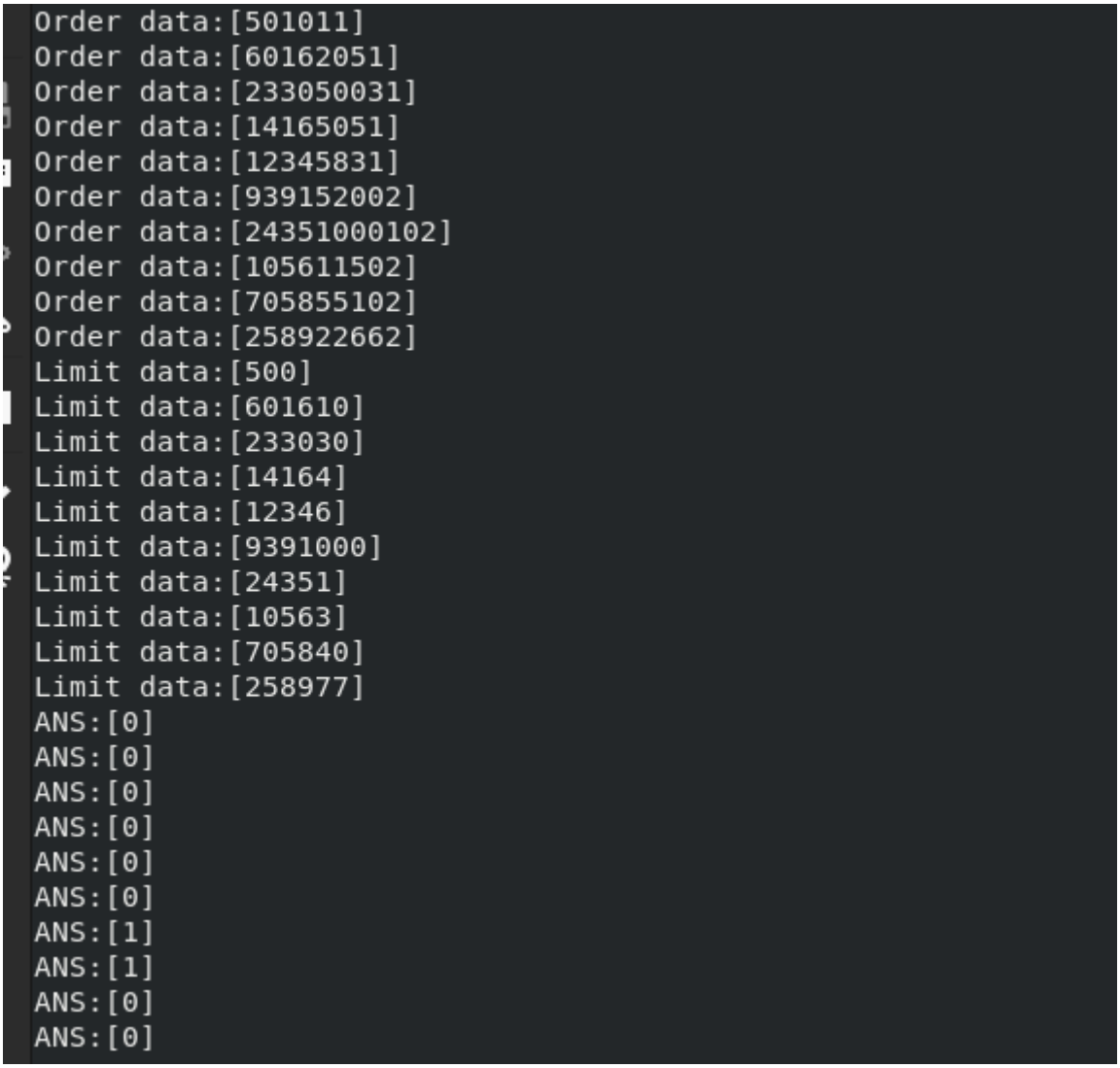
```
}

int QueryLimit(int64_t *list)
{
    int i = 0;
    MYSQL_RES *res;
    MYSQL_ROW row;
    stringstream sql_query;
    sql_query.str(""); // 字串流清零，將流中的資料全部清除
    sql_query.clear();
    sql_query << "SELECT * FROM `Limit`";
    if (mysql_real_query(conndb, sql_query.str().c_str(),
sql_query.str().length()))
    {
        cout << "QueryLimit ERROR: " << string(mysql_error(conndb)) <<
endl;
        cout << "ERROR query:" << sql_query.str() << endl;
        return -1;
    }

    res = mysql_use_result(conndb);
    Limit limit_obj;
    while ((row = mysql_fetch_row(res)) != NULL)
    {

        limit_obj.stockno = atoi(row[0]);
        limit_obj.qty = atoi(row[1]);
        std::stringstream ss;
        ss.str("");
        ss.clear();
        ss << limit_obj.stockno << limit_obj.qty;
        cout << "Limit data:[" << ss.str() << "]" << endl;
        list[i] = atoll(ss.str().c_str());
        sw_limitlist[i] = limit_obj;
        i++;
    }
    mysql_free_result(res);
    return 0;
}

int SearchAmt(std::vector<int> account_vector, int account)
{
    int account_to_find = account;
    auto it = std::find(account_vector.begin(), account_vector.end(),
account_to_find);
    if (it != account_vector.end())
    {
        // 帳號的index 一定是偶數
        int index = std::distance(account_vector.begin(), it);
        if (index % 2 == 0)
        {
            if (index + 1 < int(account_vector.size()))
            {
                int amt = account_vector[index + 1]; // 找帳號對應的度
```

```
                std::cout << "帳號 " << account_to_find << " 的額度是：" <<
amt << std::endl;
                return amt;
            }
        }
        else
        {
            std::cout << "未找到帳號 " << account_to_find << " 的額度" <<
std::endl;
        }
    }
    else
    {
        std::cout << "無此帳號 " << account_to_find << std::endl;
    }
    return -1;
}

int main(int argc, char **argv)
{
    int64_t orderlist[100];
    int64_t limitlist[100];
    int status;
    status = ConnectDB();
    if (status != 0)
    {
        cout << "status=" << status << endl;
        exit(1);
    }

    status = QueryOrder(orderlist);
    if (status != 0)
    {
        cout << "status=" << status << endl;
        exit(1);
    }

    status = QueryLimit(limitlist);
    if (status != 0)
    {
        cout << "status=" << status << endl;
        exit(1);
    }

    //------以上從資料庫拿完資料------//

    int sw_ans[50];
    int sw_use_amt = 0;
    int money = 0;
    for (int i = 0; i <= 50; i++)
    {
        if (sw_orderlist[i].bs == 1)
        {
            money = sw_orderlist[i].qty * sw_orderlist[i].price * 1000;
```

```
            sw_use_amt += money;
            if (5000000 < money)
            {
                sw_ans[i] = 1;
            }
            else
            {
                sw_ans[i] = 0;
            }
        }
        else if (sw_orderlist[i].bs == 2)
        {
            if (sw_limitlist[i].stockno == sw_orderlist[i].stockno)
            {
                if (sw_limitlist[i].qty < sw_orderlist[i].qty)
                {
                    sw_ans[i] = 1;
                }
                else
                {
                    sw_ans[i] = 0;
                }
            }
        }
        else
        {
            sw_ans[i] = 0;
        }
        cout << "ANS:[" << sw_ans[i] << "]" << endl;
    }

    // Command Line Parser
    sda::utils::CmdLineParser parser;

    // Switches
    //**************//"<Full Arg>",  "<Short Arg>", "<Description>", "
<Default>"
    parser.addSwitch("--xclbin_file", "-x", "input binary file string",
"");
    parser.addSwitch("--device_id", "-d", "device index", "0");
    parser.parse(argc, argv);

    // Read settings
    std::string binaryFile = parser.value("xclbin_file");
    int device_index = stoi(parser.value("device_id"));

    if (argc < 3)
    {
        parser.printHelp();
        return EXIT_FAILURE;
    }

    std::cout << "Open the device" << device_index << std::endl;
    auto device = xrt::device(device_index);
```

```
    std::cout << "Load the xclbin " << binaryFile << std::endl;
    auto uuid = device.load_xclbin(binaryFile);

    size_t vector_size_bytes = sizeof(int) * DATA_SIZE;

    auto krnl = xrt::kernel(device, uuid, "riskcontrol");

    std::cout << "Allocate Buffer in Global Memory\n";
    auto device_order_data = xrt::bo(device, vector_size_bytes,
krnl.group_id(0));
    auto device_result = xrt::bo(device, vector_size_bytes,
krnl.group_id(1));

    device_order_data.write(orderlist);

    // 把硬體結果接回來本地
    auto result = device_result.map<int *>();

    // Synchronize buffer content with device side
    std::cout << "synchronize input buffer data to device global memory\n";

    device_order_data.sync(XCL_BO_SYNC_BO_TO_DEVICE);

    std::cout << "Execution of the kernel\n";
    auto run = krnl(device_order_data, device_result);
    run.wait();

    // Get the output;
    std::cout << "Get the output data from the device" << std::endl;
    device_result.sync(XCL_BO_SYNC_BO_FROM_DEVICE);

    // memcmp 是用來判斷兩段記憶體區塊內容是否相同的函式
    int ret = std::memcmp(result, sw_ans, 10);
    if (ret > 0)
    {
        std::cout << "result > sw_ans" << std::endl;
    }
    else if (ret < 0)
    {
        std::cout << "result < sw_ans" << std::endl;
    }
    else
    {
        std::cout << "結果一樣?" << std::endl;
    }
    std::cout << "bufReference=" << &sw_ans << std::endl;
    std::cout << "device_result=" << &device_result << std::endl;
    std::cout << "TEST PASSED\n";
    return 0;
}
```

圖片


Order data:[501011]
Order data:[60162051]
Order data:[233050031]
Order data:[14165051]
Order data:[12345831]
Order data:[939152002]
Order data:[24351000102]
Order data:[105611502]
Order data:[705855102]
Order data:[258922662]
Limit data:[500]
Limit data:[601610]
Limit data:[233030]
Limit data:[14164]
Limit data:[12346]
Limit data:[9391000]
Limit data:[24351]
Limit data:[10563]
Limit data:[705840]
Limit data:[258977]
ANS:[0]
ANS:[0]
ANS:[0]
ANS:[0]
ANS:[0]
ANS:[0]
ANS:[1]
ANS:[1]
ANS:[0]
ANS:[0]