Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Компьютерные системы и сети»

ОТЧЕТ

по практикуму № 1

по курсу «Архитектура ЭВМ»

на тему: «Разработка и отладка программ в вычислительном комплексе Тераграф с помощью библиотеки leonhard x64 xrt»

Студент ИУ7-52Б (Группа) _____ (Подпись, дата) А. П. Лемешев (И. О. Фамилия)

Преподаватель _____ (Подпись, дата) Е. Н. Дубровин (И. О. Фамилия)

2022 г.

СОДЕРЖАНИЕ

1 Цель практикума

Практикум посвящен освоению принципов работы вычислительного комплекса **Тераграф** и получению практических навыков решения задач обработки множеств на основе гетерогенной вычислительной структуры. В ходе практикума необходимо ознакомиться с типовой структурой двух взаимодействующих программ: хост-подсистемы и программного ядра **sw_kernel**. Участникам предоставляется доступ к удаленному серверу с ускорительной картой и настроенными средствами сборки проектов, конфигурационный файл для двухъядерной версии микропроцессора **Леонард Эйлер**, а также библиотека **leonhard x64 xrt** с открытым исходным кодом.

2 Индивидуальное задание

Вариант 12. Система сбора сетевой статистики. Сформировать в хост-подсистеме и передать в SPE таблицу из 1024 ip адресов 195.19.32.0/22 (адреса 195.19.32.0 .. 195.19.35.255), где для каждого адреса сформированы четыре 16-ти разрядных счетчика (начальное значение — 0). Далее отправлять из хост-подсистемы номер счетчика и ip адрес. При каждом обращении увеличить соответствующий счетчик на 1. По запросу хост-подсистемы выдать состояние счетчиков для запрошенного ip адреса.

3 Код программы

В листинге 3.1 показан код из файла **host_main.cpp**. В листинге 3.2 показан код из файла **sw_kernel_main.c**.

Листинг 3.1 – Код из файла **host_main.cpp**

```
#include <iostream>
#include <stdio.h>
#include <stdexcept>
#include <iomanip>

#ifdef _WINDOWS
#include <io.h>
#else
#include <unistd.h>
#endif

#include "experimental/xrt_device.h"
#include "experimental/xrt_kernel.h"
#include "experimental/xrt_bo.h"
#include "experimental/xrt_ini.h"
#include "gpc_defs.h"
#include "leonhardx64_xrt.h"
#include "gpc_handlers.h"

#define BURST 1024 // кол-во адресов в SPE таблице
#define BYTE 256 // константа для перевода младших двух байтов
   IP-адреса в число

const uint64_t start_ip = 32 * BYTE; // начальное смещение
   IP-адреса
union uint64
{
    uint64_t    u64;
    uint32_t    u32[2];
    uint16_t    u16[4];
    uint8_t     u8[8];
};
uint64_t rand64()
{
    uint64 tmp;
    tmp.u32[0] =  rand();
```

```
    tmp.u32[1] =  rand();

    return tmp.u64;
}
static void usage()
{
    std::cout << "usage: <xclbin> <sw_kernel>" << std::endl;
}
int main(int argc, char **argv)
{
    unsigned int cores_count = 0;
    float LNH_CLOCKS_PER_SEC;
    __foreach_core(group, core) cores_count++;
    // Assign xclbin
    if (argc < 3)
    {
        usage();
        throw std::runtime_error("Test failed: no xclbin
           specified.\n");
    }
    // Open device #0
    leonhardx64 lnh_inst = leonhardx64(0, argv[1]);
    __foreach_core(group, core)
    {
        lnh_inst.load_sw_kernel(argv[2], group, core);
    }
    // Запись множества из BURST key-value и его
       последовательное чтение
    // через Global Memory Buffer
    // Выделение памяти под буферы gpc2host и host2gpc для
       каждого ядра и группы
    uint64_t
       *host2gpc_buffer[LNH_GROUPS_COUNT][LNH_MAX_CORES_IN_GROUP];
    __foreach_core(group, core)
    {
        host2gpc_buffer[group][core] = (uint64_t *)malloc(2 *
           BURST * sizeof(uint64_t));
    }
    uint64_t
       *gpc2host_buffer[LNH_GROUPS_COUNT][LNH_MAX_CORES_IN_GROUP];
    __foreach_core(group, core)
```

```
{
    gpc2host_buffer[group][core] = (uint64_t *)malloc(2 *
       BURST * sizeof(uint64_t));
}
// Вводим IP и превращаем его в смещение относительно
   начального IP
uint64_t tmp_ip[4];
int cnt_num = 0;
printf("Input your IP-address: ");
scanf("%llu.%llu.%llu.%llu", tmp_ip, tmp_ip + 1, tmp_ip +
   2, tmp_ip + 3);
printf("Following IP-address was received:
   %llu.%llu.%llu.%llu\n",
        tmp_ip[0], tmp_ip[1], tmp_ip[2], tmp_ip[3]);
// Проверка попадания введённого IP-адреса в заданный
   диапазон (адреса 195.19.32.0 .. 195.19.35.255)
if (tmp_ip[0] != 192 || tmp_ip[1] != 19 || tmp_ip[2] < 32
   ||
         tmp_ip[2] > 35 || tmp_ip[3] > 255)
{
    printf("Error: incorrect IP-address.\n");
    return -1;
}
printf("Enter number of a counter (1 .. 4): ");
scanf("%d", &cnt_num);
if (cnt_num < 1 || cnt_num > 4)
{
    printf("Error: incorrect number of a counter.\n");
    return -2;
}
uint64_t user_ip = tmp_ip[2] * BYTE + tmp_ip[3];
uint64_t offset = user_ip - start_ip;
uint64_t user_key = offset;
uint64_t start_key = 0;
uint8_t counters[4] = {0};
counters[cnt_num - 1]++;
// Создание массива ключей и значений для записи в lnh64
__foreach_core(group, core)
{
    for (size_t i = 0; i < BURST; i++)
    {
```

```
            // Первый элемент массива uint64_t - key
            host2gpc_buffer[group][core][2 * i] = start_key +
               i;

            // Второй uint64_t - value
            host2gpc_buffer[group][core][2 * i + 1] =
               counters[cnt_num - 1];
        }
    }
    // Запуск обработчика insert_burst
    __foreach_core(group, core)
    {
        lnh_inst.gpc[group][core]->start_async(__event__(insert_burst
    }
    // DMA запись массива host2gpc_buffer в глобальную память
    __foreach_core(group, core)
    {
        lnh_inst.gpc[group][core]->buf_write(BURST * 2 *
           sizeof(uint64_t),
                (char *)host2gpc_buffer[group][core]);
    }
    // Ожидание завершения DMA
    __foreach_core(group, core)
    {
        lnh_inst.gpc[group][core]->buf_write_join();
    }
    // Передать количество key-value и наш ключ
    __foreach_core(group, core)
    {
        lnh_inst.gpc[group][core]->mq_send(BURST);
        lnh_inst.gpc[group][core]->mq_send(user_key);
    }
    // Запуск обработчика для последовательного обхода
       множества ключей
    __foreach_core(group, core)
    {
        lnh_inst.gpc[group][core]->start_async(__event__(search_burst
    }
    // Получить количество ключей и значение по переданному
       ключу
    unsigned int
```

```
   count[LNH_GROUPS_COUNT][LNH_MAX_CORES_IN_GROUP];
unsigned int
   answer[LNH_GROUPS_COUNT][LNH_MAX_CORES_IN_GROUP];
__foreach_core(group, core)
{
    count[group][core] =
       lnh_inst.gpc[group][core]->mq_receive();
    answer[group][core] =
       lnh_inst.gpc[group][core]->mq_receive();
}
// Прочитать количество ключей
__foreach_core(group, core)
{
    lnh_inst.gpc[group][core]->buf_read(count[group][core]
       *
            2 * sizeof(uint64_t), (char
               *)gpc2host_buffer[group][core]);
}
// Ожидание завершения DMA
__foreach_core(group, core)
{
    lnh_inst.gpc[group][core]->buf_read_join();
}
// Чтение значения, полученного по ключу и проверка
   целостности данных
__foreach_core(group, core)
{
    uint64_t value = answer[group][core];
    uint64_t orig_value = host2gpc_buffer[group][core][2 *
       user_key + 1];
    printf("State of chosen counter (#%d): %llu.\n",
       cnt_num, value);
    printf("State of all counters: %lld %llu %llu %llu.\n",
            counters[0], counters[1], counters[2],
               counters[3]);
    if (value == orig_value)
        printf("Data is correct.\n");
    else
        printf("Data is incorrect.)\n");
}
__foreach_core(group, core)
```

```
    {
        free(host2gpc_buffer[group][core]);
        free(gpc2host_buffer[group][core]);
    }
    return 0;
}
```

Листинг 3.2 – Код из файла **sw_kernel_main.c**

```
/*
 * gpc_test.c
 *
 * sw_kernel library
 *
 *  Created on: April 23, 2021
 *      Author: A.Popov
 */

#include <stdlib.h>
#include <unistd.h>
#include "lnh64.h"
#include "gpc_io_swk.h"
#include "gpc_handlers.h"

#define SW_KERNEL_VERSION 26
#define DEFINE_LNH_DRIVER
#define DEFINE_MQ_R2L
#define DEFINE_MQ_L2R
#define __fast_recall__

#define TEST_STRUCTURE 1

extern lnh lnh_core;
extern global_memory_io gmio;
volatile unsigned int event_source;

int main(void)
{
    //Leonhard driver structure should be initialised
    lnh_init();
    //Initialise host2gpc and gpc2host queues
    gmio_init(lnh_core.partition.data_partition);
    for (;;)
```

```
    {
        //Wait for event
        while (!gpc_start());
        //Enable RW operations
        set_gpc_state(BUSY);
        //Wait for event
        event_source = gpc_config();
        switch(event_source)
        {
            case __event__(insert_burst) : insert_burst();
               break;
            case __event__(search_burst) : search_burst();
               break;
        }
        //Disable RW operations
        set_gpc_state(IDLE);
        while (gpc_start());
    }
}

void insert_burst()
{
    //Удаление данных из структур
    lnh_del_str_sync(TEST_STRUCTURE);
    //Объявление переменных
    unsigned int count = mq_receive();
    unsigned int size_in_bytes = 2*count*sizeof(uint64_t);
    //Создание буфера для приема пакета
    uint64_t *buffer = (uint64_t*)malloc(size_in_bytes);
    //Чтение пакета в RAM
    buf_read(size_in_bytes, (char*)buffer);
    //Обработка пакета - запись
    for (int i=0; i<count; i++)
    {
        lnh_ins_sync(TEST_STRUCTURE,buffer[2*i],buffer[2*i+1]);
    }
    lnh_sync();
    free(buffer);
}

void search_burst() {
```

```
    //Ожидание завершения предыдущих команд
    lnh_sync();
    //Объявление переменных
    unsigned int count = lnh_get_num(TEST_STRUCTURE);
    //Передать количество key-value
    mq_send(count);
    //Получить ключ
    auto key = mq_receive();
    //Поиск по ключу
    lnh_search(1, key);
    //Отправка ответа
    mq_send(lnh_core.result.value);
    // mq_send(buffer[2*1+1]);
}
```

4 Результат работы программы

На рисунках 4.1–4.4 показаны результаты работы программы.

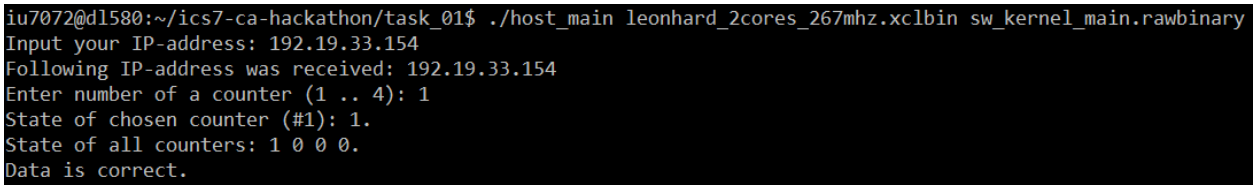

Рисунок 4.1 – Результат работы программы — ч. 1

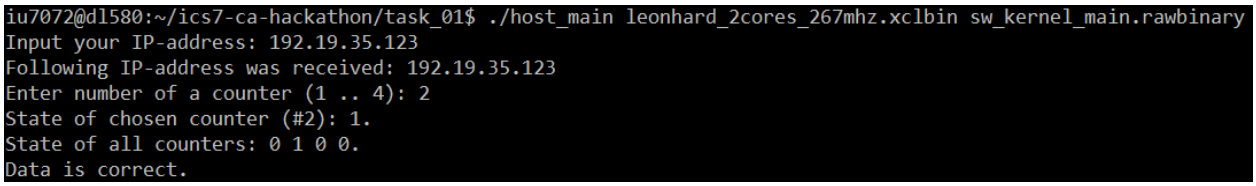

Рисунок 4.2 – Результат работы программы — ч. 2

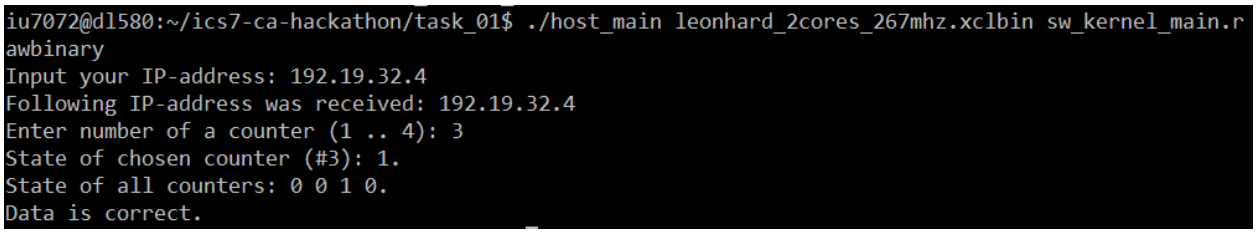

Рисунок 4.3 – Результат работы программы — ч. 3

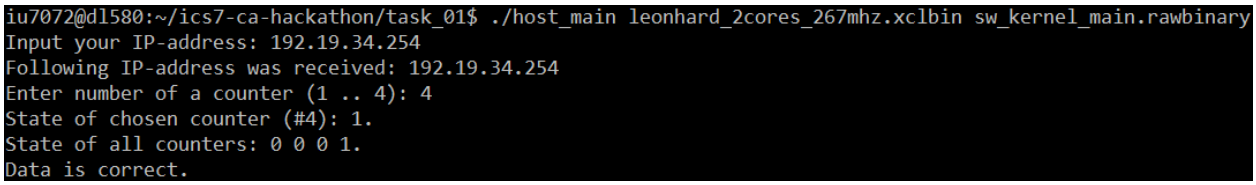

Рисунок 4.4 – Результат работы программы — ч. 4

5 Заключение

В ходе практикума было проведено ознакомление с типовой структурой двух взаимодействующих программ: хост-подсистемы и программного ядра **sw_kernel**. Была разработана программа для хост-подсистемы и обработчика программного ядра, выполняющия действия, описанные в индивидуальном задании (вариант 12).