



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 «Программное обеспечение ЭВМ и информационные технологии»

---

## Отчет по практикуму №1 по курсу "Архитектура ЭВМ"

Тема Разработка и отладка программ в вычислительном комплексе Тераграф с помощью  
библиотеки leonhard x64 xrt

Студент Сапожков А. М.

Группа ИУ7-53Б

Оценка (баллы) \_\_\_\_\_

Преподаватель Попов А. Ю.

Москва — 2022 г.

# Содержание

<b>1</b>	<b>Выполнение лабораторной работы</b>	<b>4</b>
1.1	Индивидуальное задание . . . . .	4
1.2	Код программы . . . . .	4
1.3	Тестирование программного обеспечения . . . . .	10
	<b>Заключение</b>	<b>11</b>

# Введение

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.

# 1 Выполнение лабораторной работы

## 1.1 Индивидуальное задание

### Вариант 1 (19)

Сетевой коммутатор на 128 портов. Сформировать в хост-подсистеме и передать в SPE таблицу коммутации из 254 ip адресов 195.19.32.1/24 (адреса 195.19.32.1 .. 195.19.32.254). Каждому адресу поставить в соответствие один из 128 интерфейсов (целые числа 0..127). Выполнить тестирование работы коммутатора, посылая из хост-подсистемы ip адреса и сравнивая полученный от GPC номер интерфейса с ожидаемым.

## 1.2 Код программы

Листинг 1.1 – sw\_kernel\_main.c

```
1 #include <stdlib.h>
2 #include <unistd.h>
3 #include "lnh64.h"
4 #include "gpc_io_swk.h"
5 #include "gpc_handlers.h"
6
7 #define SW_KERNEL_VERSION 26
8 #define DEFINE_LNH_DRIVER
9 #define DEFINE_MQ_R2L
10 #define DEFINE_MQ_L2R
11 #define __fast_recall__
12
13 #define TEST_STRUCTURE 1
14
15 extern lnh lnh_core;
16 extern global_memory_io gmio;
17 volatile unsigned int event_source;
18
19 int main(void) {
20     //Leonhard driver structure should be initialised
21     lnh_init();
```

```

22 //Initialise host2gpc and gpc2host queues
23 gmio_init(lnh_core.partition.data_partition);
24 for (;;) {
25     //Wait for event
26     while (!gpc_start());
27     //Enable RW operations
28     set_gpc_state(BUSY);
29     //Wait for event
30     event_source = gpc_config();
31     switch(event_source) {
32         case __event__(insert_burst) : insert_burst(); break;
33         case __event__(search_burst) : search_burst(); break;
34     }
35     //Disable RW operations
36     set_gpc_state(IDLE);
37     while (gpc_start());
38
39 }
40 }
41
42 void insert_burst() {
43     lnh_del_str_sync(TEST_STRUCTURE);
44     unsigned int count = mq_receive();
45     unsigned int size_in_bytes = 2*count*sizeof(uint64_t);
46     uint64_t *buffer = (uint64_t*)malloc(size_in_bytes);
47     buf_read(size_in_bytes, (char*)buffer);
48     for (int i=0; i<count; i++) {
49         lnh_ins_sync(TEST_STRUCTURE, buffer[2*i], buffer[2*i+1]);
50     }
51     lnh_sync();
52     free(buffer);
53 }
54
55 void search_burst() {
56     lnh_sync();
57     unsigned int count = lnh_get_num(TEST_STRUCTURE);
58     mq_send(count);
59     auto key = mq_receive();
60     lnh_search(1, key);
61     mq_send(lnh_core.result.value);
62 }

```

Листинг 1.2 – host\_main.cpp

```
1 #include <iostream>
2 #include <stdio.h>
3 #include <stdexcept>
4 #include <iomanip>
5 #ifdef _WINDOWS
6 #include <io.h>
7 #else
8 #include <unistd.h>
9 #endif
10
11
12 #include "experimental/xrt_device.h"
13 #include "experimental/xrt_kernel.h"
14 #include "experimental/xrt_bo.h"
15 #include "experimental/xrt_ini.h"
16
17 #include "gpc_defs.h"
18 #include "leonhardx64_xrt.h"
19 #include "gpc_handlers.h"
20
21 #define BURST 254
22
23 union uint64 {
24     uint64_t    u64;
25     uint32_t    u32[2];
26     uint16_t    u16[4];
27     uint8_t     u8[8];
28 };
29
30 uint64_t rand64() {
31     uint64 tmp;
32     tmp.u32[0] = rand();
33     tmp.u32[1] = rand();
34     return tmp.u64;
35 }
36
37 static void usage()
38 {
39     std::cout << "usage: _xclbin>_sw_kernel>\n\n";
40 }
```

```

41
42 const uint64_t start_ip = 195019032001;
43
44 int main(int argc, char** argv)
45 {
46     unsigned int cores_count = 0;
47     float LNH_CLOCKS_PER_SEC;
48
49     __foreach_core(group, core) cores_count++;
50
51     //Assign xclbin
52     if (argc < 3) {
53         usage();
54         throw std::runtime_error("FAILED_TEST\nNo xclbin
55         specified");
56     }
57
58     //Open device #0
59     leonhardx64 ln_h_inst = leonhardx64(0, argv[1]);
60     __foreach_core(group, core) {
61         ln_h_inst.load_sw_kernel(argv[2], group, core);
62     }
63
64     uint64_t
65         *host2gpc_buffer[LNH_GROUPS_COUNT][LNH_MAX_CORES_IN_GROUP];
66     __foreach_core(group, core) {
67         host2gpc_buffer[group][core] = (uint64_t*)
68             malloc(2*BURST*sizeof(uint64_t));
69     }
70
71     uint64_t
72         *gpc2host_buffer[LNH_GROUPS_COUNT][LNH_MAX_CORES_IN_GROUP];
73     __foreach_core(group, core) {
74         gpc2host_buffer[group][core] = (uint64_t*)
75             malloc(2*BURST*sizeof(uint64_t));
76     }
77
78     uint64_t tmp_ip[4];
79     printf("Input your IP:");
80     scanf("%llu.%llu.%llu.%llu", tmp_ip, tmp_ip + 1, tmp_ip + 2,
81         tmp_ip + 3);
82     printf("Got IP: %llu.%llu.%llu.%llu\n", tmp_ip[0], tmp_ip[1],

```

```

    tmp_ip[2], tmp_ip[3]);
76 int64_t user_ip = ((tmp_ip[0] * 1000 + tmp_ip[1]) * 1000 +
    tmp_ip[2]) * 1000 + tmp_ip[3];
77 int64_t offset = user_ip - start_ip;
78 printf("Offset from the start IP: %d\n", offset);
79
80 if (offset < 0 || offset >= BURST) {
81     printf("Error: incorrect IP\n");
82
83     return -1;
84 }
85
86 uint64_t user_key = offset;
87 uint64_t start_key = 0;
88
89 __foreach_core(group, core) {
90     for (int i=0; i<BURST; i++) {
91         host2gpc_buffer[group][core][2*i] = start_key + i;
92         host2gpc_buffer[group][core][2*i+1] = rand64() % 128;
93     }
94 }
95
96
97 __foreach_core(group, core) {
98     lnh_inst.gpc[group][core]—>\
99     start_async(__event__(insert_burst));
100 }
101
102 __foreach_core(group, core) {
103     lnh_inst.gpc[group][core]—>buf_write(BURST*2*\
104     sizeof(uint64_t), (char*)host2gpc_buffer[group][core]);
105 }
106
107 __foreach_core(group, core) {
108     lnh_inst.gpc[group][core]—>buf_write_join();
109 }
110
111 __foreach_core(group, core) {
112     lnh_inst.gpc[group][core]—>mq_send(BURST);
113     lnh_inst.gpc[group][core]—>mq_send(user_key);
114 }

```



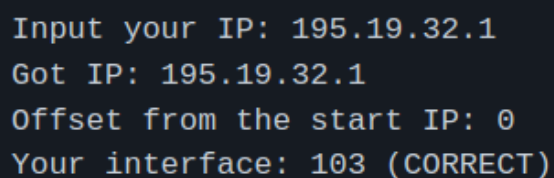
```

115
116 __foreach_core(group, core) {
117     lnh_inst.gpc[group][core]—>\
118     start_async(__event__(search_burst));
119 }
120
121 unsigned int count[LNH_GROUPS_COUNT][LNH_MAX_CORES_IN_GROUP];
122 unsigned int answer[LNH_GROUPS_COUNT][LNH_MAX_CORES_IN_GROUP];
123
124 __foreach_core(group, core) {
125     count[group][core] =
126         lnh_inst.gpc[group][core]—>mq_receive();
127     answer[group][core] =
128         lnh_inst.gpc[group][core]—>mq_receive();
129 }
130
131 __foreach_core(group, core) {
132     lnh_inst.gpc[group][core]—>buf_read(count[group][core]*2*\
133     sizeof(uint64_t), (char*)gpc2host_buffer[group][core]);
134 }
135
136 __foreach_core(group, core) {
137     lnh_inst.gpc[group][core]—>buf_read_join();
138 }
139
140 __foreach_core(group, core) {
141     uint64_t value = answer[group][core];
142     uint64_t orig_value =
143         host2gpc_buffer[group][core][2*user_key+1];
144     printf("Your interface: %llu", value);
145
146     if (value == orig_value) {
147         printf("(CORRECT)\n");
148     }
149     else {
150         printf("(INCORRECT)\n");
151     }
152 }

```

```
153     __foreach_core(group, core) {  
154         free(host2gpc_buffer[group][core]);  
155         free(gpc2host_buffer[group][core]);  
156     }  
157  
158     return 0;  
159 }
```

## 1.3 Тестирование программного обеспечения



```
Input your IP: 195.19.32.1  
Got IP: 195.19.32.1  
Offset from the start IP: 0  
Your interface: 103 (CORRECT)
```

Рисунок 1.1 – Тест программы

# 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, выполняющая действия, описанные в индивидуальном задании.