Федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский государственный технический университет им. Н.Э. Баумана (национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ – Информатика, искусственный интеллект и системы управления
КАФЕДРА – Информационные системы и телекоммуникации

Отчет к лабораторной работе №3

по курсу "Программное обеспечение встроенных систем"

направления 09.04.02

"Разработка программного обеспечения. Отладка и профилирование кода."

Выполнил:

студент группы ИУ3-11М

Щесняк С.С.

Проверил:

Федоров С.В.

Москва 2022

Оглавление

1. Цель работы

Освоить практические навыки адаптации, отладки, профилирования ПО системы на кристалле в целях удовлетворения требований по быстродействию и объему для отдельных его функций. Изучить влияние ручных оптимизаций циклов на быстродействие программы. Изучить оптимизации компилятора на уровне формируемого ассемблерного кода.

2. Задание

1. Используя счетчики производительности, оцените для проекта из первой лабораторной работы:
    a. Время выполнения первого вызова обработчика прерывания;
    b. Общее время выполнения десяти первых вызовов обработчика прерывания;

    Проведите измерения для различных настроек оптимизации кода по быстродействию и по объему. Запишите полученные результаты. Разместите обработчик прерывания в тесно связанной памяти и повторите измерение времени выполнения первого обработчика прерывания и первых десяти вызовов. Для обработчика прерывания сопоставьте время выполнения с включенной и отключенной оптимизацией, при размещении во внешней и в тесно связанной памяти. Объясните различие по быстродействию на разных уровнях оптимизации по сгенерированному коду на уровне ассемблера.
2. Реализуйте задание для самостоятельного выполнения как две функции. Одна должна быть реализована в виде обычного цикла, другая - с развертыванием цикла.
    a. *Индивидуальное задание:* реализовать функцию расчета контрольной суммы UDP для блока данных.

    Произведите оценку быстродействия исходного цикла и развернутого с различными настройками оптимизации по быстродействию. Определите, за сколько инструкций выполняется одна итерация цикла,

сколько тактов приходится на обработку одного элемента массива и как был оптимизирован цикл. Требуется провести сравнение развернутой и неразвернутой реализации для уровней оптимизации Off и Level3.

3. В настройках пакета поддержки платы включите профайлинг. Модифицируйте исходный код таким образом, чтобы программа завершала работу по определенному условию. Изучите отчет профайлера. На примере функции работы с массивом определите с помощью Performance Counter, какой оверхед в циклах вносит вызов функции mcount в выполнение функции. Изучите ассемблерный код и найдите в нем вызов функции mcount.

3. Ход работы

1. В рамках данной части лабораторной работы требовалось выполнить тестирование производительности функции обработки прерываний от кнопок ПО системы на кристалле с помощью счетчика производительности. Его результаты представлены в таблице ниже.

Таблица 1. Результаты тестирования производительности функции обработки прерываний от кнопок для одного и десяти нажатий при различных настройках оптимизации кода по быстродействию и объему.

Уровень оптимизации	Время выполнения одного вызова обработчика, в тактах	Время выполнения десяти вызовов обработчика, в тактах
Без оптимизации	73	388
Level 1	50	284
Level 2	50	284
Level 3	50	284
Size	50	284

В целях формирования представления о причинах возникновения различий по быстродействию рассматриваемой функции при отключенной и включенной оптимизации рассмотрим соответствующий код на уровне ассемблера.

Листинг ассемблерного кода обработчика прерываний при отключенной оптимизации:

```
10000000 <handle_button_interrupts>:
#ifdef ALT_ENHANCED_INTERRUPT_API_PRESENT
static void handle_button_interrupts(void* context)
#else
static void handle_button_interrupts(void* context, alt_u32 id)
#endif
{
; сохранение контекста в стеке
10000000:	defffd04	addi	sp,sp,-12
10000004:	df000215	stw	fp,8(sp)
10000008:	df000204	addi	fp,sp,8
1000000c:	e13ffe15	stw	r4,-8(fp)
        PERF_BEGIN(PERFORMANCE_COUNTER_0_BASE,1);
10000010:	0007883a	mov	r3,zero ; обнуление r3
10000014:	00820034	movhi	r2,2048 ; формирование адреса
10000018:	10c40535	stwio	r3,4116(r2) ; загрузка 0 по адресу 4116(r2)
        /* Cast context to edge_capture's type. It is important that this
        be declared volatile to avoid unwanted compiler optimization. */
        volatile int* edge_capture_ptr = (volatile int*) context;
1000001c:	e0bffe17	ldw	r2,-8(fp) ; выгрузка контекста из памяти
10000020:	e0bfff15	stw	r2,-4(fp) ; запись по указателю edge_capture_ptr
        /*
        * Read the edge capture register on the button PIO.
        * Store value.
        */
        *edge_capture_ptr =
        IORD_ALTERA_AVALON_PIO_EDGE_CAP(BUTTONS_BASE);
10000024:	00820034	movhi	r2,2048 ; формируем адрес
10000028:	10c42337	ldwio	r3,4236(r2) ; загрузка значения кода нажатой кнопки в r3
        *edge_capture_ptr =
1000002c:	e0bfff17	ldw	r2,-4(fp) ; запись в регистр r2 адреса edge_capture_ptr
10000030:	10c00015	stw	r3,0(r2) ; запись r3 по адресу 0(r2)
```

```
        /* Write to the edge capture register to reset it. */
        IOWR_ALTERA_AVALON_PIO_EDGE_CAP(BUTTONS_BASE, 0);
10000034:   0007883a    mov     r3,zero ; обнуление r3
10000038:   00820034    movhi   r2,2048 ; формирование адреса
1000003c:   10c42335    stwio   r3,4236(r2) ; загрузка 0 по адресу 4236(r2)
        /* Read the PIO to delay ISR exit. This is done to prevent a
        spurious interrupt in systems with high processor -> pio
        latency and fast interrupts. */
        IORD_ALTERA_AVALON_PIO_EDGE_CAP(BUTTONS_BASE);
10000040:   00820034    movhi   r2,2048 ; формирование адреса
10000044:   10842337    ldwio   r2,4236(r2) ; загрузка значения по адресу 4236(r2) в r2
        calls+=1;
10000048:   d0a01017    ldw     r2,-32704(gp) ; выгрузка контекста из памяти
1000004c:   10800044    addi    r2,r2,1 ; прибавление единицы к calls
10000050:   d0a01015    stw     r2,-32704(gp) ; сохранение значения calls в памяти
        PERF_END(PERFORMANCE_COUNTER_0_BASE,1);
10000054:   0007883a    mov     r3,zero ; обнуление r3
10000058:   00820034    movhi   r2,2048 ; формирование адреса
1000005c:   10c40435    stwio   r3,4112(r2) ; загрузка 0 по адресу 4112(r2)
}
; восстановление контекста
10000060:   0001883a    nop
10000064:   e037883a    mov     sp,fp
10000068:   df000017    ldw     fp,0(sp)
1000006c:   dec00104    addi    sp,sp,4
10000070:   f800283a    ret ; завершение работы
```

Листинг ассемблерного кода обработчика прерываний на третьем уровне оптимизации:

```
10000000 <handle_button_interrupts>:
static void handle_button_interrupts(void* context)
#else
static void handle_button_interrupts(void* context, alt_u32 id)
#endif
{
        PERF_BEGIN(PERFORMANCE_COUNTER_0_BASE,1);
10000000:   00820034    movhi   r2,2048 ; формирование адреса
10000004:   10040535    stwio   zero,4116(r2) ; загрузка 0 по адресу 4116(r2)
        /*
        * Read the edge capture register on the button PIO.
        * Store value.
        */
```

```
        *edge_capture_ptr =
        IORD_ALTERA_AVALON_PIO_EDGE_CAP(BUTTONS_BASE);
10000008:   10842337    ldwio   r2,4236(r2) ; загрузка значения кода нажатой кнопки в r2
        *edge_capture_ptr =
1000000c:   20800015    stw     r2,0(r4) ; запись r2 по адресу 0(r4)
        /* Write to the edge capture register to reset it. */
        IOWR_ALTERA_AVALON_PIO_EDGE_CAP(BUTTONS_BASE, 0);
10000010:   00820034    movhi   r2,2048 ; формирование адреса
10000014:   10042335    stwio   zero,4236(r2) ; загрузка 0 по адресу 4236(r2)
        /* Read the PIO to delay ISR exit. This is done to prevent a
        spurious interrupt in systems with high processor -> pio
        latency and fast interrupts. */
        IORD_ALTERA_AVALON_PIO_EDGE_CAP(BUTTONS_BASE);
10000018:   10842337    ldwio   r2,4236(r2) ; загрузка значения по адресу 4236(r2) в r2
        calls+=1;
1000001c:   d0a00f17    ldw     r2,-32708(gp) ; выгрузка контекста из памяти
10000020:   10800044    addi    r2,r2,1 ; прибавление единицы к calls
10000024:   d0a00f15    stw     r2,-32708(gp) ; сохранение значения calls в памяти
        PERF_END(PERFORMANCE_COUNTER_0_BASE,1);
10000028:   00820034    movhi   r2,2048 ; формирование адреса
1000002c:   10040435    stwio   zero,4112(r2) ; загрузка 0 по адресу 4112(r2)
}
10000030:   f800283a    ret ; завершение работы
```

Анализ представленного выше кода позволяет заключить, что повышение быстродействия кода практически в полтора раза при включении оптимизации произошло ввиду уменьшения объема кода благодаря использованию регистровой оптимизации.

Далее требовалось разместить обработчик прерываний от кнопок в тесно связанной памяти и выполнить повторное тестирование его производительности. Результаты тестирования приведены в таблице ниже.

Таблица 2. Результаты тестирования производительности размещенного в тесно связанной памяти обработчика прерываний от кнопок для одного и десяти нажатий при различных настройках оптимизации кода по быстродействию и объему.

Уровень оптимизации	Время выполнения одного вызова обработчика, в тактах	Время выполнения десяти вызовов обработчика, в тактах
Без оптимизации	35	350
Level 1	26	260
Level 2	26	260
Level 3	26	260
Size	26	260

Анализ данных, представленных в таблицах 1 и 2 позволяет заключить, что размещение обработчика прерываний в тесно связанной памяти позволяет получить практически двухкратный выигрыш в быстродействии кода.

2. В рамках данной части лабораторной работы требовалось реализовать задание для самостоятельного выполнения в виде двух функций: без развертывания цикла (simple_loop) и с развертыванием цикла (unrolled_loop); провести тестирование их производительности при различных настройках оптимизации кода по быстродействию и размерах обрабатываемого массива. Его результаты представлены в таблице ниже.

Таблица 3. Результаты тестирования производительности функций расчета контрольной суммы UDP при различных настройках оптимизации кода по быстродействию и размерах блока данных (число вызовов функций $C = 100$).

Уровень оптимизации	Без развертывания цикла		С развертыванием цикла	
	Время выполнения C вызовов функции, в тактах	Время выполнения одной итерации, в тактах	Время выполнения C вызовов функции, в тактах	Время выполнения одной итерации, в тактах
Без оптимизации, $N = 255$	878703	34,459	610347	23,935
Level 3, $N = 255$	155066	6,081	70048	2,747
Level 3, $N = 511$	308926	6,046	141475	2,769
Level 3, $N = 1023$	617932	6,040	282817	2,765
Level 3, $N = 2047$	1252161	6,117	578530	2,826

Анализ данных, представленных в таблице выше позволяет заключить, что увеличение размера блока данных приводит к увеличению времени работы программы.

В целях формирования представления о причинах возникновения различий по быстродействию рассматриваемой функции при отключенной и включенной оптимизации рассмотрим соответствующий код на уровне ассемблера.

Листинг ассемблерного кода обработчика прерываний при отключенной оптимизации:

```
000002e4 <simple_loop>:

uint32_t simple_loop(uint16_t* array, uint32_t size){
   2e4:  defffb04        addi    sp,sp,-20 ; выделение памяти на 5 4-х байтных слова
   2e8:  df000415        stw     fp,16(sp) ; запись текущего значения frame pointer в стек
; установка frame pointer на начало первого свободного слова
   2ec:  df000404        addi    fp,sp,16
   2f0:  e13ffd15        stw     r4,-12(fp) ; запись первого аргумента функции по адресу fp-12
   2f4:  e17ffc15        stw     r5,-16(fp) ; запись второго аргумента функции по адресу fp-16
        uint32_t i;
        uint32_t check_sum = 0;
   2f8:  e03ffe15        stw     zero,-8(fp) ; запись 0 по адресу fp-8
        for(i=0; i<size; i++){
   2fc:  e03fff15        stw     zero,-4(fp) ; запись i=0 по адресу fp-4
   300:  00000d06        br      338 <simple_loop+0x54> ; переход к управлению циклом for
                check_sum += array[i];
   304:  e0bfff17        ldw     r2,-4(fp) ; запись в регистр r2 текущего значения i
   308:  1085883a        add     r2,r2,r2 ; вычисление смещения относительно начала блока данных
   30c:  1007883a        mov     r3,r2 ; запись результата в r3
   310:  e0bffd17        ldw     r2,-12(fp) ; запись в регистр r2 адреса начала блока данных
   314:  10c5883a        add     r2,r2,r3 ; вычисление адреса i-го и i+1-го элемента
   318:  1080000b        ldhu    r2,0(r2) ; запись в регистр r2 значения i-го элемента
   31c:  10bfffcc        andi    r2,r2,65535 ; применение к полученному значению маски из единиц
   320:  e0fffe17        ldw     r3,-8(fp) ; запись в регистр r3 текущего значения check_sum
   324:  1885883a        add     r2,r3,r2 ; вычисление нового значения check_sum
   328:  e0bffe15        stw     r2,-8(fp) ; запись результата по адресу fp-8
        for(i=0; i<size; i++){
   32c:  e0bfff17        ldw     r2,-4(fp) ; запись в регистр r2 текущего значения i
   330:  10800044        addi    r2,r2,1 ; инкремент переменной цикла
   334:  e0bfff15        stw     r2,-4(fp) ; запись результата по адресу fp-4
   338:  e0ffff17 ldw    r3,-4(fp) ; запись в регистр r3 нового значения i
   33c:  e0bffc17        ldw     r2,-16(fp) ; запись в регистр r2 значения size
; переход к началу цикла, если i<size
   340:  18bff036        bltu    r3,r2,304 <simple_loop+0x20>
        }
        check_sum += check_sum >> 16;
   344:  e0bffe17        ldw     r2,-8(fp) ; запись в регистр r2 значения контрольной суммы
   348:  1004d43a        srli    r2,r2,16 ; логический сдвиг данного значения вправо на 16
   34c:  e0fffe17        ldw     r3,-8(fp) ; запись в регистр r3 значения контрольной суммы
```

```
; вычисление суммы старшего и младшего полуслов контрольной суммы
   350: 1885883a      add     r2,r3,r2
   354: e0bffe15      stw     r2,-8(fp) ; запись результата по адресу fp-8
        check_sum = ~check_sum & 0xffff;
   358: e0bffe17      ldw     r2,-8(fp) ; запись в регистр r2 значения контрольной суммы
   35c: 0084303a      nor     r2,zero,r2 ; вычисление инверсии значения контрольной суммы
   360: 10bfffcc      andi    r2,r2,65535 ; применение к полученному значению маски из единиц
   364: e0bffe15      stw     r2,-8(fp) ; запись результата по адресу fp-8
        return check_sum;
; запись в регистр r2 итогового значения контрольной суммы
   368: e0bffe17      ldw     r2,-8(fp)
}
   36c: e037883a      mov     sp,fp ; установка указателя стека на frame pointer
   370: df000017      ldw     fp,0(sp) ; восстанавливаем старое значение frame pointer
   374: dec00104      addi    sp,sp,4 ; восстановление stack pointer
   378: f800283a      ret ; завершение работы

0000037c <unrolled_loop>:

uint32_t unrolled_loop(uint16_t* array, uint32_t size){
   37c: defff904      addi    sp,sp,-28 ; выделение памяти на 7 4-х байтных слов
   380: df000615      stw     fp,24(sp) ; запись текущего значения frame pointer в стек
; установка frame pointer на начало первого свободного слова
   384: df000604      addi    fp,sp,24
   388: e13ffb15      stw     r4,-20(fp) ; запись первого аргумента функции по адресу fp-20
   38c: e17ffa15      stw     r5,-24(fp) ; запись второго аргумента функции по адресу fp-24
        uint32_t i;
        uint32_t check_sum_2 = 0;
   390: e03ffe15      stw     zero,-8(fp) ; запись 0 по адресу fp-8
        uint32_t lenlong = size/4;
   394: e0bffa17      ldw     r2,-24(fp) ; запись в регистр r2 значения size
   398: 1004d0ba      srli    r2,r2,2 ; логический сдвиг данного значения вправо на 2
   39c: e0bffd15      stw     r2,-12(fp) ;запись результата по адресу fp-12
        uint32_t lenshort = size%4;
   3a0: e0bffa17      ldw     r2,-24(fp) ; запись в регистр r2 значения size
; вычисление остатка от деления путем применения соответствующей маски
   3a4: 108000cc      andi    r2,r2,3
   3a8: e0bffc15      stw     r2,-16(fp) ;запись результата по адресу fp-16
        for(i=0; i<lenlong; i++){
   3ac: e03fff15      stw     zero,-4(fp) ; запись i=0 по адресу fp-4
   3b0: 00002a06      br      45c <unrolled_loop+0xe0> ; переход к управлению циклом for
                check_sum_2 += array[i*4];
```

```
    3b4:  e0bfff17        ldw     r2,-4(fp) ; запись в регистр r2 текущего значения i
; вычисление смещения относительно начала блока данных логическим сдвигом влево на 3
    3b8:  100490fa        slli    r2,r2,3
    3bc:  e0fffb17        ldw     r3,-20(fp) ; запись в регистр r3 адреса начала блока данных
    3c0:  1885883a        add     r2,r3,r2 ; вычисление адреса i-го и i+1-го элемента
    3c4:  1080000b        ldhu    r2,0(r2) ; запись в регистр r2 значения i-го элемента
    3c8:  10bfffcc        andi    r2,r2,65535 ; применение к полученному значению маски из единиц
    3cc:  e0fffe17        ldw     r3,-8(fp) ; запись в регистр r3 текущего значения check_sum_2
    3d0:  1885883a        add     r2,r3,r2 ; вычисление нового значения check_sum_2
    3d4:  e0bffe15        stw     r2,-8(fp) ; запись результата по адресу fp-8
                    check_sum_2 += array[i*4+1];
; выполняются аналогичные вычисления для i*4+1-го элемента
    3d8:  e0bfff17        ldw     r2,-4(fp)
    3dc:  100490fa        slli    r2,r2,3
    3e0:  10800084        addi    r2,r2,2
    3e4:  e0fffb17        ldw     r3,-20(fp)
    3e8:  1885883a        add     r2,r3,r2
    3ec:  1080000b        ldhu    r2,0(r2)
    3f0:  10bfffcc        andi    r2,r2,65535
    3f4:  e0fffe17        ldw     r3,-8(fp)
    3f8:  1885883a        add     r2,r3,r2
    3fc:  e0bffe15        stw     r2,-8(fp)
; выполняются аналогичные вычисления для i*4+2-го элемента
                    check_sum_2 += array[i*4+2];
    400:  e0bfff17        ldw     r2,-4(fp)
    404:  100490fa        slli    r2,r2,3
    408:  10800104        addi    r2,r2,4
    40c:  e0fffb17        ldw     r3,-20(fp)
    410:  1885883a        add     r2,r3,r2
    414:  1080000b        ldhu    r2,0(r2)
    418:  10bfffcc        andi    r2,r2,65535
    41c:  e0fffe17        ldw     r3,-8(fp)
    420:  1885883a        add     r2,r3,r2
    424:  e0bffe15        stw     r2,-8(fp)
                    check_sum_2 += array[i*4+3];
; выполняются аналогичные вычисления для i*4+3-го элемента
    428:  e0bfff17        ldw     r2,-4(fp)
    42c:  100490fa        slli    r2,r2,3
    430:  10800184        addi    r2,r2,6
    434:  e0fffb17        ldw     r3,-20(fp)
    438:  1885883a        add     r2,r3,r2
    43c:  1080000b        ldhu    r2,0(r2)
```

```
   440: 10bfffcc        andi    r2,r2,65535
   444: e0fffe17        ldw     r3,-8(fp)
   448: 1885883a        add     r2,r3,r2
   44c: e0bffe15        stw     r2,-8(fp)
        for(i=0; i<lenlong; i++){
   450: e0bfff17        ldw     r2,-4(fp) ; запись в регистр r2 текущего значения i
   454: 10800044        addi    r2,r2,1 ; инкремент переменной цикла
   458: e0bfff15        stw     r2,-4(fp) ; запись результата по адресу fp-4
   45c: e0ffff17 ldw    r3,-4(fp) ; запись в регистр r3 нового значения i
; запись в регистр r2 результата целочисленного деления size на 4
   460: e0bffd17        ldw     r2,-12(fp)
 ; переход к началу цикла, если i<size/4
   464: 18bfd336        bltu    r3,r2,3b4 <unrolled_loop+0x38>
        }
        for(i=0; i<lenshort; i++){
   468: e03fff15        stw     zero,-4(fp) ; запись i=0 по адресу fp-4
   46c: 00001006        br      4b0 <unrolled_loop+0x134> ; переход к управлению циклом for
                check_sum_2 += array[lenlong*4+i];
; запись в регистр r2 результата целочисленного деления size на 4
   470: e0bffd17        ldw     r2,-12(fp)
; вычисление смещения относительно начала блока данных
; умножение значения lenlong на 4 логическим сдвигом влево на 2
   474: 100690ba        slli    r3,r2,2
   478: e0bfff17        ldw     r2,-4(fp) ; запись в регистр r2 текущего значения i
   47c: 1885883a        add     r2,r3,r2 ; вычисление индекса
   480: 1085883a        add     r2,r2,r2 ;вычисление смещения относительного начала блока данных
   484: 1007883a        mov     r3,r2 ; запись в регистр r3 вычисленного смещения
   488: e0bffb17        ldw     r2,-20(fp) ; запись в регистр r3 адреса начала блока данных
   48c: 10c5883a        add     r2,r2,r3 ; вычисление адреса i-го и i+1-го элемента
   490: 1080000b        ldhu    r2,0(r2) ; запись в регистр r2 значения i-го элемента
   494: 10bfffcc        andi    r2,r2,65535 ; применение к полученному значению маски из единиц
   498: e0fffe17        ldw     r3,-8(fp) ; запись в регистр r3 текущего значения check_sum_2
   49c: 1885883a        add     r2,r3,r2 ; вычисление нового значения check_sum_2
   4a0: e0bffe15        stw     r2,-8(fp) ; запись результата по адресу fp-8
        for(i=0; i<lenshort; i++){
   4a4: e0bfff17        ldw     r2,-4(fp) ; запись в регистр r2 текущего значения i
   4a8: 10800044        addi    r2,r2,1 ; инкремент переменной цикла
   4ac: e0bfff15        stw     r2,-4(fp) ; запись результата по адресу fp-4
   4b0: e0ffff17 ldw    r3,-4(fp) ; запись в регистр r3 нового значения i
; запись в регистр r2 остатка от деления size на 4
   4b4: e0bffc17        ldw     r2,-16(fp)
; переход к началу цикла, если i<size%4
```

```
    4b8:  18bfed36        bltu    r3,r2,470 <unrolled_loop+0xf4>
          }
          check_sum_2 += check_sum_2 >> 16;
    4bc:  e0bffe17        ldw     r2,-8(fp) ; запись в регистр r2 значения контрольной суммы
    4c0:  1004d43a        srli    r2,r2,16 ; логический сдвиг данного значения вправо на 16
    4c4:  e0fffe17        ldw     r3,-8(fp) ; запись в регистр r3 значения контрольной суммы
; вычисление суммы старшего и младшего полуслов контрольной суммы
    4c8:  1885883a        add     r2,r3,r2
    4cc:  e0bffe15        stw     r2,-8(fp) ; запись результата по адресу fp-8
          check_sum_2 = ~check_sum_2 & 0xffff;
    4d0:  e0bffe17        ldw     r2,-8(fp) ; запись в регистр r2 значения контрольной суммы
    4d4:  0084303a        nor     r2,zero,r2 ; вычисление инверсии значения контрольной суммы
    4d8:  10bfffcc        andi    r2,r2,65535 ; применение к полученному значению маски из единиц
    4dc:  e0bffe15        stw     r2,-8(fp) ; запись результата по адресу fp-8
          return check_sum_2;
; запись в регистр r2 итогового значения контрольной суммы
    4e0:  e0bffe17        ldw     r2,-8(fp)
}
    4e4:  e037883a        mov     sp,fp ; установка указателя стека на frame pointer
    4e8:  df000017        ldw     fp,0(sp) ; восстанавливаем старое значение frame pointer
    4ec:  dec00104        addi    sp,sp,4 ; восстановление stack pointer
    4f0:  f800283a        ret ; завершение работы
```

Листинг ассемблерного кода обработчика прерываний на третьем уровне оптимизации:

```
000001f0 <simple_loop>:
}

uint32_t simple_loop(uint16_t* array, uint32_t size){
          uint32_t i;
          uint32_t check_sum = 0;
          for(i=0; i<size; i++){
; r4 – адрес текущего элемента блока данных, изначально совпадает с адресом его начала
; r5 – значение второго аргумента функции
; переход к концу функции, если size=0
    1f0:  28000c26        beq     r5,zero,224 <simple_loop+0x34>
    1f4:  294b883a        add     r5,r5,r5 ; вычисление смещения до конца массива
    1f8:  2147883a        add     r3,r4,r5 ; вычисление адреса конца массива
          uint32_t check_sum = 0;
    1fc:  0005883a        mov     r2,zero ; запись в регистр r2 текущего (нулевого) значения check_sum
                check_sum += array[i];
```

```
; начало цикла
   200: 2140000b        ldhu    r5,0(r4) ; запись в регистр r2 значения i-го элемента
        for(i=0; i<size; i++){
; смещение указателя на текущий элемент массива на 1 позицию
   204: 21000084        addi    r4,r4,2
                check_sum += array[i];
; вычисление нового значения check_sum
   208: 1145883a        add     r2,r2,r5
        for(i=0; i<size; i++){
; переход к началу цикла, если адрес текущего элемента блока данных не совпадает с его концом
   20c: 193ffc1e        bne     r3,r4,200 <simple_loop+0x10>
        }
        check_sum += check_sum >> 16;
; логический сдвиг значения контрольной суммы вправо на 16
   210: 1006d43a        srli    r3,r2,16
; вычисление суммы старшего и младшего полуслов контрольной суммы
   214: 1885883a        add     r2,r3,r2
        check_sum = ~check_sum & 0xffff;
   218: 0084303a        nor     r2,zero,r2 ; вычисление инверсии значения контрольной суммы
   21c: 10bfffcc        andi    r2,r2,65535 ; применение к полученному значению маски из единиц
   220: f800283a        ret ; завершение работы
        for(i=0; i<size; i++){
   224: 00bfffd4        movui   r2,65535 ; запись единиц в младшие 16 бит результата
        return check_sum;
}
   228: f800283a        ret ; завершение работы

0000022c <unrolled_loop>:

uint32_t unrolled_loop(uint16_t* array, uint32_t size){
        uint32_t i;
        uint32_t check_sum_2 = 0;
        uint32_t lenlong = size/4;
; вычисление результата целочисленного деления путем логического сдвига вправо на 4
   22c: 280cd0ba        srli    r6,r5,2
        uint32_t lenshort = size%4;
; вычисление остатка от деления путем применения соответствующей маски
   230: 2ac000cc        andi    r11,r5,3
        for(i=0; i<lenlong; i++){
; переход к циклу добора, если size/4=0
   234: 30000e26        beq     r6,zero,270 <unrolled_loop+0x44>
```

```
; вычисление смещения, соответствующего концу цикла, логическим сдвигом влево на 3
   238:  301490fa        slli    r10,r6,3
   23c:  2007883a        mov     r3,r4 ; запись в регистр r3 указателя на начало блока данных
        uint32_t check_sum_2 = 0;
; запись в регистр r6 текущего (нулевого) значения check_sum_2
   240:  000d883a        mov     r6,zero
; вычисление значения указателя, соответствующего концу цикла
   244:  5115883a        add     r10,r10,r4
                check_sum_2 += array[i*4];
   248:  1880000b        ldhu    r2,0(r3) ; запись в регистр r2 значения i*4-го элемента
                check_sum_2 += array[i*4+1];
   24c:  1a40008b        ldhu    r9,2(r3) ; запись в регистр r9 значения i*4+1-го элемента
                check_sum_2 += array[i*4+2];
   250:  1a00010b        ldhu    r8,4(r3) ; запись в регистр r8 значения i*4+2-го элемента
                check_sum_2 += array[i*4+3];
   254:  19c0018b        ldhu    r7,6(r3) ; запись в регистр r7 значения i*4+3-го элемента
; вычисление суммы данных четырех элементов массива
   258:  1245883a        add     r2,r2,r9
   25c:  1205883a        add     r2,r2,r8
   260:  11c5883a        add     r2,r2,r7
        for(i=0; i<lenlong; i++){
; смещение указателя на текущий элемент массива на 4 позиции
   264:  18c00204        addi    r3,r3,8
                check_sum_2 += array[i*4+3];
; вычисление нового значения check_sum_2
   268:  308d883a        add     r6,r6,r2
        for(i=0; i<lenlong; i++){
; переход к началу цикла, если указатель на текущий элемент массива не совпадает с указателем, соответствующим концу цикла
   26c:  50fff61e        bne     r10,r3,248 <unrolled_loop+0x1c>
        }
        for(i=0; i<lenshort; i++){
; переход к концу цикла, если size%4=0
   270:  58001426        beq     r11,zero,2c4 <unrolled_loop+0x98>
                check_sum_2 += array[lenlong*4+i];
; вычисление разности размера блока данных и остатка от его деления на 4
   274:  00bfff04        movi    r2,-4
   278:  288a703a        and     r5,r5,r2
; вычисление значения указателя на следующий после обработанной части элемент блока данных
   27c:  2945883a        add     r2,r5,r5
   280:  2085883a        add     r2,r4,r2
   284:  10c0000b        ldhu    r3,0(r2) ; запись данного элемента в регистр r3
```

```
        for(i=0; i<lenshort; i++){
   288: 58800060        cmpeqi  r2,r11,1 ; является ли данный элемент последним в блоке?
                check_sum_2 += array[lenlong*4+i];
   28c: 30cd883a        add     r6,r6,r3 ; вычисление нового значения check_sum_2
        for(i=0; i<lenshort; i++){
; если да, то выполняется переход в конец цикла
   290: 10000c1e        bne     r2,zero,2c4 <unrolled_loop+0x98>
                check_sum_2 += array[lenlong*4+i];
; вычисление значения указателя на следующий после обработанной части элемент блока данных
   294: 28800044        addi    r2,r5,1
   298: 1085883a        add     r2,r2,r2
   29c: 2085883a        add     r2,r4,r2
   2a0: 1080000b        ldhu    r2,0(r2) ; запись данного элемента в регистр r2
        for(i=0; i<lenshort; i++){
   2a4: 5ac000d8        cmpnei  r11,r11,3 ; является ли данный элемент последним в блоке?
                check_sum_2 += array[lenlong*4+i];
   2a8: 308d883a        add     r6,r6,r2 ; вычисление нового значения check_sum_2
        for(i=0; i<lenshort; i++){
; если да, то выполняется переход в конец цикла
   2ac: 5800051e        bne     r11,zero,2c4 <unrolled_loop+0x98>
                check_sum_2 += array[lenlong*4+i];
; вычисление значения указателя на следующий после обработанной части элемент блока данных
   2b0: 29400084        addi    r5,r5,2
   2b4: 294b883a        add     r5,r5,r5
   2b8: 214b883a        add     r5,r4,r5
   2bc: 2880000b        ldhu    r2,0(r5) ; запись данного элемента в регистр r2
   2c0: 308d883a        add     r6,r6,r2 ; вычисление нового значения check_sum_2
        }
        check_sum_2 += check_sum_2 >> 16;
; логический сдвиг значения контрольной суммы вправо на 16
   2c4: 3004d43a        srli    r2,r6,16
; вычисление суммы старшего и младшего полуслов контрольной суммы
   2c8: 1185883a        add     r2,r2,r6
        check_sum_2 = ~check_sum_2 & 0xffff;
   2cc: 0084303a        nor     r2,zero,r2 ; вычисление инверсии значения контрольной суммы
        return check_sum_2;
}
   2d0: 10bfffcc        andi    r2,r2,65535 ; применение к полученному значению маски из единиц
   2d4: f800283a        ret ; завершение работы
```

Анализ представленного выше кода позволяет заключить, что повышение быстродействия кода при включении оптимизации произошло благодаря выносу инвариантного кода из тела цикла, использованию последовательности одинаковых команд вместо цикла небольшого размера, регистровой оптимизации.

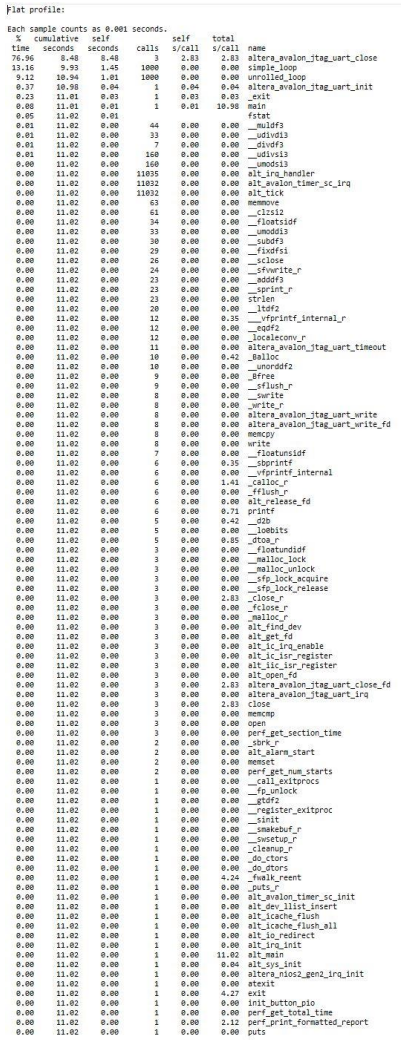
3. В рамках данной части лабораторной работы требовалось провести тестирование производительности двух разработанных функций при включенном профайлинге. Его результаты представлены в таблице ниже.

Таблица 4. Результаты тестирования производительности функций расчета контрольной суммы UDP при включенном профайлинге (число вызовов функций $C = 1000$, размер блока $N = 2047$).

Уровень оптимизации	Время выполнения C вызовов функции, в тактах	Время выполнения C вызовов функции, в тактах
Без оптимизации	70537573	48994945
Level 3	12447807	5623109

Анализ результатов тестирования производительности функций расчета контрольной суммы UDP при включенном и отключенном профайлинге позволяет заключить, что отслеживание быстродействия кода с помощью средств компилятора влечет за собой дополнительные временные затраты (для simple_loop – примерно в 3000 тактов, для функции unrolled_loop – в 2500 тактов).

Вид отчета профайлера представлен на рисунках ниже:

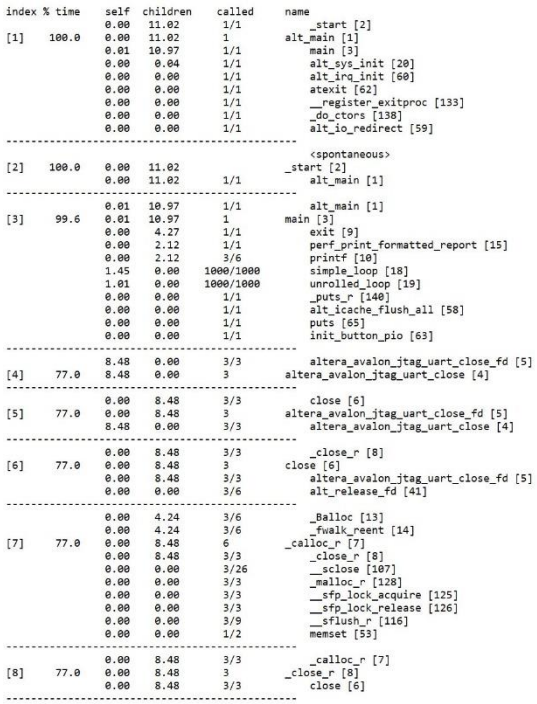


Flat profile:

Each sample counts as 0.001 seconds.

% time	cumulative seconds	self seconds	calls	self s/call	total s/call	name
76.96	8.48	8.48	3	2.83	2.83	altera_avalon_jtag_uart_close
13.16	9.93	1.45	1000	0.00	0.00	simple_loop
9.12	10.94	1.01	1000	0.00	0.00	unrolled_loop
0.37	10.98	0.04	1	0.04	0.04	altera_avalon_jtag_uart_init
0.23	11.01	0.03	1	0.03	0.03	_exit
0.08	11.01	0.01	1	0.01	10.98	main
0.05	11.02	0.01				fstat
0.01	11.02	0.00	44	0.00	0.00	__muldf3
0.01	11.02	0.00	33	0.00	0.00	__udivdi3
0.01	11.02	0.00	7	0.00	0.00	__divdf3
0.01	11.02	0.00	160	0.00	0.00	__udivsi3
0.00	11.02	0.00	160	0.00	0.00	__umodsi3
0.00	11.02	0.00	11035	0.00	0.00	alt_irq_handler
0.00	11.02	0.00	11032	0.00	0.00	alt_avalon_timer_sc_irq
0.00	11.02	0.00	11032	0.00	0.00	alt_tick
0.00	11.02	0.00	63	0.00	0.00	memmove
0.00	11.02	0.00	61	0.00	0.00	__clzsi2
0.00	11.02	0.00	34	0.00	0.00	__floatsidf
0.00	11.02	0.00	33	0.00	0.00	__umoddi3
0.00	11.02	0.00	30	0.00	0.00	__subdf3
0.00	11.02	0.00	29	0.00	0.00	__fixdfsi
0.00	11.02	0.00	26	0.00	0.00	__sclose
0.00	11.02	0.00	24	0.00	0.00	__sfvwrite_r
0.00	11.02	0.00	23	0.00	0.00	__adddf3
0.00	11.02	0.00	23	0.00	0.00	__sprint_r
0.00	11.02	0.00	23	0.00	0.00	strlen
0.00	11.02	0.00	20	0.00	0.00	__ltdf2
0.00	11.02	0.00	12	0.00	0.35	___vfprintf_internal_r
0.00	11.02	0.00	12	0.00	0.00	__eqdf2
0.00	11.02	0.00	12	0.00	0.00	_localeconv_r
0.00	11.02	0.00	11	0.00	0.00	altera_avalon_jtag_uart_timeout
0.00	11.02	0.00	10	0.00	0.42	_Balloc
0.00	11.02	0.00	10	0.00	0.00	__unorddf2
0.00	11.02	0.00	9	0.00	0.00	_Bfree
0.00	11.02	0.00	9	0.00	0.00	__sflush_r
0.00	11.02	0.00	8	0.00	0.00	__swrite
0.00	11.02	0.00	8	0.00	0.00	_write_r
0.00	11.02	0.00	8	0.00	0.00	altera_avalon_jtag_uart_write
0.00	11.02	0.00	8	0.00	0.00	altera_avalon_jtag_uart_write_fd
0.00	11.02	0.00	8	0.00	0.00	memcpy
0.00	11.02	0.00	8	0.00	0.00	write
0.00	11.02	0.00	7	0.00	0.00	__floatunsidf
0.00	11.02	0.00	6	0.00	0.35	__sfprintf
0.00	11.02	0.00	6	0.00	0.00	__vfprintf_internal
0.00	11.02	0.00	6	0.00	1.41	_calloc_r
0.00	11.02	0.00	6	0.00	0.00	_fflush_r
0.00	11.02	0.00	6	0.00	0.00	alt_release_fd
0.00	11.02	0.00	6	0.00	0.71	printf
0.00	11.02	0.00	5	0.00	0.42	__d2b
0.00	11.02	0.00	5	0.00	0.00	__lo0bits
0.00	11.02	0.00	5	0.00	0.85	_dtoa_r
0.00	11.02	0.00	3	0.00	0.00	__floatundidf
0.00	11.02	0.00	3	0.00	0.00	__malloc_lock
0.00	11.02	0.00	3	0.00	0.00	__malloc_unlock
0.00	11.02	0.00	3	0.00	0.00	__sfp_lock_acquire
0.00	11.02	0.00	3	0.00	0.00	__sfp_lock_release
0.00	11.02	0.00	3	0.00	2.83	_close_r
0.00	11.02	0.00	3	0.00	0.00	_fclose_r
0.00	11.02	0.00	3	0.00	0.00	_malloc_r
0.00	11.02	0.00	3	0.00	0.00	alt_find_dev
0.00	11.02	0.00	3	0.00	0.00	alt_get_fd
0.00	11.02	0.00	3	0.00	0.00	alt_ic_irq_enable
0.00	11.02	0.00	3	0.00	0.00	alt_ic_isr_register
0.00	11.02	0.00	3	0.00	0.00	alt_iic_isr_register
0.00	11.02	0.00	3	0.00	0.00	alt_open_fd
0.00	11.02	0.00	3	0.00	2.83	altera_avalon_jtag_uart_close_fd
0.00	11.02	0.00	3	0.00	0.00	altera_avalon_jtag_uart_irq
0.00	11.02	0.00	3	0.00	2.83	close
0.00	11.02	0.00	3	0.00	0.00	memcmp
0.00	11.02	0.00	3	0.00	0.00	open
0.00	11.02	0.00	3	0.00	0.00	perf_get_section_time
0.00	11.02	0.00	2	0.00	0.00	_sbrk_r
0.00	11.02	0.00	2	0.00	0.00	alt_alarm_start
0.00	11.02	0.00	2	0.00	0.00	memset
0.00	11.02	0.00	2	0.00	0.00	perf_get_num_starts
0.00	11.02	0.00	1	0.00	0.00	__call_exitprocs
0.00	11.02	0.00	1	0.00	0.00	__fp_unlock
0.00	11.02	0.00	1	0.00	0.00	__gtdf2
0.00	11.02	0.00	1	0.00	0.00	__register_exitproc
0.00	11.02	0.00	1	0.00	0.00	__sinit
0.00	11.02	0.00	1	0.00	0.00	__smakebuf_r
0.00	11.02	0.00	1	0.00	0.00	__swsetup_r
0.00	11.02	0.00	1	0.00	0.00	_cleanup_r
0.00	11.02	0.00	1	0.00	0.00	_do_ctors
0.00	11.02	0.00	1	0.00	0.00	_do_dtors
0.00	11.02	0.00	1	0.00	4.24	_fwalk_reent
0.00	11.02	0.00	1	0.00	0.00	_puts_r
0.00	11.02	0.00	1	0.00	0.00	alt_avalon_timer_sc_init
0.00	11.02	0.00	1	0.00	0.00	alt_dev_llist_insert
0.00	11.02	0.00	1	0.00	0.00	alt_icache_flush
0.00	11.02	0.00	1	0.00	0.00	alt_icache_flush_all
0.00	11.02	0.00	1	0.00	0.00	alt_io_redirect
0.00	11.02	0.00	1	0.00	0.00	alt_irq_init
0.00	11.02	0.00	1	0.00	11.02	alt_main
0.00	11.02	0.00	1	0.00	0.04	alt_sys_init
0.00	11.02	0.00	1	0.00	0.00	altera_nios2_gen2_irq_init
0.00	11.02	0.00	1	0.00	0.00	atexit
0.00	11.02	0.00	1	0.00	4.27	exit
0.00	11.02	0.00	1	0.00	0.00	init_button_pio
0.00	11.02	0.00	1	0.00	0.00	perf_get_total_time
0.00	11.02	0.00	1	0.00	2.12	perf_print_formatted_report
0.00	11.02	0.00	1	0.00	0.00	puts

Рисунок 1. Плоский профиль.



```
index % time    self  children    called     name
                0.00   11.02       1/1           _start [2]
[1]    100.0    0.00   11.02       1         alt_main [1]
                0.01   10.97       1/1           main [3]
                0.00    0.04       1/1           alt_sys_init [20]
                0.00    0.00       1/1           alt_irq_init [60]
                0.00    0.00       1/1           atexit [62]
                0.00    0.00       1/1           __register_exitproc [133]
                0.00    0.00       1/1           _do_ctors [138]
                0.00    0.00       1/1           alt_io_redirect [59]
-----------------------------------------------
                                                 <spontaneous>
[2]    100.0    0.00   11.02                 _start [2]
                0.00   11.02       1/1           alt_main [1]
-----------------------------------------------
                0.01   10.97       1/1           alt_main [1]
[3]     99.6    0.01   10.97       1         main [3]
                0.00    4.27       1/1           exit [9]
                0.00    2.12       1/1           perf_print_formatted_report [15]
                0.00    2.12       3/6           printf [10]
                1.45    0.00    1000/1000        simple_loop [18]
                1.01    0.00    1000/1000        unrolled_loop [19]
                0.00    0.00       1/1           _puts_r [140]
                0.00    0.00       1/1           alt_icache_flush_all [58]
                0.00    0.00       1/1           puts [65]
                0.00    0.00       1/1           init_button_pio [63]
-----------------------------------------------
                8.48    0.00       3/3           altera_avalon_jtag_uart_close_fd [5]
[4]     77.0    8.48    0.00       3         altera_avalon_jtag_uart_close [4]
-----------------------------------------------
                0.00    8.48       3/3           close [6]
[5]     77.0    0.00    8.48       3         altera_avalon_jtag_uart_close_fd [5]
                8.48    0.00       3/3           altera_avalon_jtag_uart_close [4]
-----------------------------------------------
                0.00    8.48       3/3           _close_r [8]
[6]     77.0    0.00    8.48       3         close [6]
                0.00    8.48       3/3           altera_avalon_jtag_uart_close_fd [5]
                0.00    0.00       3/6           alt_release_fd [41]
-----------------------------------------------
                0.00    4.24       3/6           _Balloc [13]
                0.00    4.24       3/6           _fwalk_reent [14]
[7]     77.0    0.00    8.48       6         _calloc_r [7]
                0.00    8.48       3/3           _close_r [8]
                0.00    0.00       3/26          __sclose [107]
                0.00    0.00       3/3           _malloc_r [128]
                0.00    0.00       3/3           __sfp_lock_acquire [125]
                0.00    0.00       3/3           __sfp_lock_release [126]
                0.00    0.00       3/9           __sflush_r [116]
                0.00    0.00       1/2           memset [53]
-----------------------------------------------
                0.00    8.48       3/3           _calloc_r [7]
[8]     77.0    0.00    8.48       3         _close_r [8]
                0.00    8.48       3/3           close [6]
-----------------------------------------------
```

Рисунок 2. Иерархия вызовов отчета профайлера.

Фрагмент листинга ассемблерного кода с вызовом функции mcount:

```
uint32_t simple_loop(uint16_t* array, uint32_t size){
    208:  f811883a   mov    r8,ra
    20c:  000e2000   call   e200 <_mcount>
    210:  403f883a   mov    ra,r8
  uint32_t i;
  uint32_t check_sum = 0;
…
```

4. Вывод

В результате выполнения лабораторной работы были изучены влияние ручных оптимизаций циклов на быстродействие программы, различные варианты оптимизаций компилятора на уровне формируемого ассемблерного кода. Кроме того, были приобретены практические навыки адаптации, отладки, профилирования ПО системы на кристалле в целях удовлетворения требований по быстродействию и объему для отдельных его функций.

5. Приложения

Приложение 1. Листинг кода, используемого в рамках выполнения общей части лабораторной работы.

```
#include <stdio.h>
#include <unistd.h>
#include "altera_avalon_pio_regs.h"
#include "altera_avalon_performance_counter.h"
#include "sys/alt_cache.h"
#include "system.h"

#define NONE_PRESSED 0xF  // Value read from button PIO when no buttons
pressed
#define DEBOUNCE 30000  // Time in microseconds to wait for switch debounce

#define KEY0_PRESSED 0xE
#define KEY1_PRESSED 0xD
#define KEY2_PRESSED 0xB
#define KEY3_PRESSED 0x7

volatile int edge_capture;
volatile int calls=0;
int call_opt=1;

#ifdef ALT_ENHANCED_INTERRUPT_API_PRESENT
static void handle_button_interrupts(void* context)
#else
static void handle_button_interrupts(void* context, alt_u32 id)
#endif
{
   PERF_BEGIN(PERFORMANCE_COUNTER_0_BASE,1);
   /* Cast context to edge_capture's type. It is important that this
   be declared volatile to avoid unwanted compiler optimization. */
   volatile int* edge_capture_ptr = (volatile int*) context;
   /*
   * Read the edge capture register on the button PIO.
   * Store value.
   */
   *edge_capture_ptr =
   IORD_ALTERA_AVALON_PIO_EDGE_CAP(BUTTONS_BASE);
   /* Write to the edge capture register to reset it. */
   IOWR_ALTERA_AVALON_PIO_EDGE_CAP(BUTTONS_BASE, 0);
   /* Read the PIO to delay ISR exit. This is done to prevent a
   spurious interrupt in systems with high processor -> pio
   latency and fast interrupts. */
   IORD_ALTERA_AVALON_PIO_EDGE_CAP(BUTTONS_BASE);
   calls+=1;
   PERF_END(PERFORMANCE_COUNTER_0_BASE,1);
}

/* Initialize the button_pio. */
static void init_button_pio()
{
   /* Recast the edge_capture pointer to match the
   alt_irq_register() function prototype. */
   void* edge_capture_ptr = (void*) &edge_capture;
   /* Enable all 4 button interrupts. */
   IOWR_ALTERA_AVALON_PIO_IRQ_MASK(BUTTONS_BASE, 0xf);
   /* Reset the edge capture register. */
   IOWR_ALTERA_AVALON_PIO_EDGE_CAP(BUTTONS_BASE, 0x0);
```

```
   /* Register the ISR. */
   void handle_button_interrupts(void* context)
__attribute__((section(".tcm")));
   #ifdef ALT_ENHANCED_INTERRUPT_API_PRESENT
   alt_ic_isr_register(BUTTONS_IRQ_INTERRUPT_CONTROLLER_ID,
   BUTTONS_IRQ,
   handle_button_interrupts,
   edge_capture_ptr, 0x0);
   #else
   alt_irq_register( BUTTONS_IRQ,
   edge_capture_ptr,
   handle_button_interrupts );
   #endif
}

int main(void)
{
   alt_icache_flush_all();
   int buttons;  // Use to hold button pressed value
   int led = 0x01;  // Use to write to led
   int left=1;
   int prev_led;
   int mask;

   printf("Simple\n");   // print a message to show that program is running

   init_button_pio();
   IOWR_ALTERA_AVALON_PIO_DATA(GREEN_LED_BASE,led); // write initial value to
pio

   PERF_RESET(PERFORMANCE_COUNTER_0_BASE);
   PERF_START_MEASURING(PERFORMANCE_COUNTER_0_BASE);
   while (1)
   {
      //buttons = IORD_ALTERA_AVALON_PIO_DATA(BUTTONS_BASE); // read buttons
via pio

      if (edge_capture)  // if button pressed
      {
         /*
         if (led >= 0x80)  // if pattern is 00000001 on board (leds in
reverse order)
            led = 0x01;  // reset pattern
         else
            led = led << 1;  // shift right on board (led0 is far left)
         */
         switch(edge_capture){
            case(1): {
               led = 0x00; //shut off every led
               break;
            }
            case(2):
            case(4): {
               prev_led=led;
               if(left){ mask = 0x80; }
               else{ mask = 0x01; }
               while (led == prev_led){
                  led = led & ~mask;
                  if (left) { mask = mask>>1; }
                  else { mask = mask<<1; }
               }
               if (left) left=0;
               else left=1;
```

```
                break;
            }
            case(8):{
                led = 0xFF; //turn on every led
                break;
            }
        }
        edge_capture = 0;

        if(calls==1 && call_opt==0){

perf_print_formatted_report(PERFORMANCE_COUNTER_0_BASE,ALT_CPU_FREQ,3,"DAC
Wait","MLA loop","synth_frame");
            PERF_STOP_MEASURING(PERFORMANCE_COUNTER_0_BASE);
        }
        if(calls==10 && call_opt==1){

perf_print_formatted_report(PERFORMANCE_COUNTER_0_BASE,ALT_CPU_FREQ,3,"DAC
Wait","MLA loop","synth_frame");
            PERF_STOP_MEASURING(PERFORMANCE_COUNTER_0_BASE);
        }

        IOWR_ALTERA_AVALON_PIO_DATA(GREEN_LED_BASE,led);  // write new value
to pio

        /* Switch debounce routine
           Wait for small delay after intial press for debouncing
           Wait for release of key
           Wait for small delay after release for debouncing */

        usleep (DEBOUNCE);
        while (buttons != NONE_PRESSED)  // wait for button release
           buttons = IORD_ALTERA_AVALON_PIO_DATA(BUTTONS_BASE);   // update
        usleep (DEBOUNCE);
      }

   }

}  // end
```

Приложение 2. Листинг кода, используемого в рамках выполнения индивидуального задания по лабораторной работе.

```
#include <stdio.h>
#include <unistd.h>
#include "altera_avalon_pio_regs.h"
#include "altera_avalon_performance_counter.h"
#include "sys/alt_cache.h"
#include "system.h"

#define NONE_PRESSED 0xF  // Value read from button PIO when no buttons
pressed
#define DEBOUNCE 30000  // Time in microseconds to wait for switch debounce

#define KEY0_PRESSED 0xE
#define KEY1_PRESSED 0xD
#define KEY2_PRESSED 0xB
#define KEY3_PRESSED 0x7

volatile int edge_capture;
volatile int calls=0;
int call_opt=1;
```

```
uint32_t simple_loop(uint16_t* array, uint32_t size);
uint32_t unrolled_loop(uint16_t* array, uint32_t size);

#ifdef ALT_ENHANCED_INTERRUPT_API_PRESENT
static void handle_button_interrupts(void* context)
#else
static void handle_button_interrupts(void* context, alt_u32 id)
#endif
{
   PERF_BEGIN(PERFORMANCE_COUNTER_0_BASE,1);
   /* Cast context to edge_capture's type. It is important that this
   be declared volatile to avoid unwanted compiler optimization. */
   volatile int* edge_capture_ptr = (volatile int*) context;
   /*
   * Read the edge capture register on the button PIO.
   * Store value.
   */
   *edge_capture_ptr =
   IORD_ALTERA_AVALON_PIO_EDGE_CAP(BUTTONS_BASE);
   /* Write to the edge capture register to reset it. */
   IOWR_ALTERA_AVALON_PIO_EDGE_CAP(BUTTONS_BASE, 0);
   /* Read the PIO to delay ISR exit. This is done to prevent a
   spurious interrupt in systems with high processor -> pio
   latency and fast interrupts. */
   IORD_ALTERA_AVALON_PIO_EDGE_CAP(BUTTONS_BASE);
   calls+=1;
   PERF_END(PERFORMANCE_COUNTER_0_BASE,1);
}

/* Initialize the button_pio. */
static void init_button_pio()
{
   /* Recast the edge_capture pointer to match the
   alt_irq_register() function prototype. */
   void* edge_capture_ptr = (void*) &edge_capture;
   /* Enable all 4 button interrupts. */
   IOWR_ALTERA_AVALON_PIO_IRQ_MASK(BUTTONS_BASE, 0xf);
   /* Reset the edge capture register. */
   IOWR_ALTERA_AVALON_PIO_EDGE_CAP(BUTTONS_BASE, 0x0);
   /* Register the ISR. */
   void handle_button_interrupts(void* context)
__attribute__((section(".tcm")));
   #ifdef ALT_ENHANCED_INTERRUPT_API_PRESENT
   alt_ic_isr_register(BUTTONS_IRQ_INTERRUPT_CONTROLLER_ID,
   BUTTONS_IRQ,
   handle_button_interrupts,
   edge_capture_ptr, 0x0);
   #else
   alt_irq_register( BUTTONS_IRQ,
   edge_capture_ptr,
   handle_button_interrupts );
   #endif
}

uint32_t simple_loop(uint16_t* array, uint32_t size)
{
    uint32_t i;
    uint32_t check_sum = 0;
    for (i = 0; i < size; i++){
        check_sum += array[i];
    }
    check_sum += check_sum >> 16;
```

```
    check_sum = ~check_sum & 0xffff;
    return check_sum;
}

uint32_t unrolled_loop(uint16_t* array, uint32_t size)
    {
    uint32_t i;
    uint32_t check_sum_2 = 0;
    uint32_t lenlong = size / 4;
    uint32_t lenshort = size % 4;
    for (i = 0; i < lenlong; i++){
        check_sum_2 += array[i * 4];
        check_sum_2 += array[i * 4 + 1];
        check_sum_2 += array[i * 4 + 2];
        check_sum_2 += array[i * 4 + 3];
    }
    for (i = 0; i < lenshort; i++){
        check_sum_2 += array[lenlong * 4 + i];
    }
    check_sum_2 += check_sum_2 >> 16;
    check_sum_2 = ~check_sum_2 & 0xffff;
    return check_sum_2;
}

int main(void)
{
   alt_icache_flush_all();
   int buttons;  // Use to hold button pressed value
   int led = 0x01;  // Use to write to led
   int left=1;
   int prev_led;
   int mask;

   printf("Simple\n");

   init_button_pio();
   IOWR_ALTERA_AVALON_PIO_DATA(GREEN_LED_BASE,led);

   PERF_RESET (PERFORMANCE_COUNTER_0_BASE);
    PERF_START_MEASURING (PERFORMANCE_COUNTER_0_BASE);

    int N = 255;
    uint32_t check_sum = 0;
    uint32_t check_sum_2 = 0;
    uint16_t array[N];
    uint32_t i;
    for (i = 0; i < N; i++){
        array[i] = i;
    }
    for (i = 0; i < 1000; i++) {
        PERF_BEGIN (PERFORMANCE_COUNTER_0_BASE, 1);
        check_sum = simple_loop(array, N);
        PERF_END (PERFORMANCE_COUNTER_0_BASE, 1);

        PERF_BEGIN (PERFORMANCE_COUNTER_0_BASE, 2);
        check_sum_2 = unrolled_loop(array, N);
        PERF_END (PERFORMANCE_COUNTER_0_BASE, 2);
    }
    printf("Simple loop: %#010x\n", check_sum);
    printf("Unrolled loop: %#010x\n", check_sum_2);
    PERF_STOP_MEASURING (PERFORMANCE_COUNTER_0_BASE);
    perf_print_formatted_report (PERFORMANCE_COUNTER_0_BASE, ALT_CPU_FREQ, 2,
"Simple_loop", "Unrolled loop");
```

```
    uint32_t r_check_sum = 0;
    for (i = 0; i < N; i++){
        r_check_sum += array[i];
    }
    r_check_sum += check_sum;
    r_check_sum += r_check_sum >> 16;
    r_check_sum = r_check_sum & 0xFFFF;
    printf("Checking: %#010x\n", r_check_sum);
    exit(0);
}  // end
```