**Министерство науки и высшего образования Российской Федерации**

**Мытищинский филиал**

**Федерального государственного бюджетного образовательного учреждение**

**высшего образования**

**«Московский государственный технический университет**

**имени Н.Э. Баумана**

**(национальный исследовательский университет)»**

**(МФ МГТУ им. Н.Э. Баумана)**

Факультет «Космический»

Кафедра «Прикладная математика, информатика и вычислительная техника»

**Лабораторная работа**

**Изучение ввода, обработки и вывода аудиоинформации в процессорной системе DE 2-115 Media Computer**

**Цель работы:** изучение использования аудио кодека для записи и воспроизведения звука, а также генерации звука определенной частоты.

**Предполагаемые результаты обучения**.

Выполнив лабораторную работу студенты:

Станут понимать, что из себя представляет звук, его представление в цифровом виде, принцип работы аудио кодека, а также аудиобуфера в процессорной системе DE2-70/115.

Смогут выполнять работу с звуком в процессорной системе DE2-70/115, создавать разнообразные звуковые эффекты.

**Подготовка к лабораторной работе**

Для выполнения лабораторной работы необходимо изучить по материалам лекций и предложенной литературы:

1. Способы представления звуковых сигналов в цифровой форме.
2. Принцип работы и назначение регистров и отдельных полей аудио порта.
3. Уясните принцип работы программы.
4. Уясните пункты заданий и отредактируйте программу в соответствии с вариантом.

**Исходные файлы, используемые в работе**

- Файл audio.c (Приложение 1) содержит программу на языке программирования Си, которая записывает звуки с помощью микрофона и воспроизводит их, используя динамики, подключенные к стенду. Аудио кодек, установленный на стенде, производит оцифровку звукового сигнала каждую 1/48000 секунды. При нажатии кнопки KEY1 выполняется очистка входных аудио буферов. Затем программа проверяет, имеются ли данные во входных аудио буферах. Во время записи звука горит светодиод LED0. При нажатии кнопки KEY2 программа выполняет очистку выходных аудио буферов и осуществляет запись в них данных, сохраненных в памяти при записи звука. Во время воспроизведения записи горит светодиод LED1. При нажатии кнопки KEY3 программа выполняет визуализацию записанного сигнала на экран монитора.

- Файл effects.c (Приложение 2) содержит модифицированную программу из файла «audio.c» на языке программирования Си. Программа модифицирована таким образом, что при установленном в верхнем положении переключателя SW0 воспроизведение аудиозаписи замедляется, при установленном в верхнее положение переключателе SW1 – ускоряется. Для замедления воспроизведения необходимо повторять запись каждого оцифрованного значения в выходной буфер, для ускорения – пропускать каждое второе значение. Также при установке переключателя SW2 в верхнее положение на воспроизводимую аудиозапись накладывается эффект эхо. Для этого, при записи в выходной буфер оцифрованного значения на него накладывается ранее оцифрованное значение, но имеющее меньшее значение громкости.

- Файл synth.c 3 содержит программу, реализующую музыкальный синтезатор. Пока нажата клавиша на клавиатуре, воспроизводится звук определенной частоты. В таблице 2 представлено значение частоты звука и соответствующая ей клавиша на клавиатуре.

1. **Порядок выполнения работы**

**Часть 1. Представление звука в цифровом виде.**

1. Открыть программу **Audacity**. Сохранить пустой проект
2. Добавить новую дорожку (Треки -> Добавить новый -> Монотрек).
3. Создать тон (Создать -> Тон...). В качестве формы волны выбрать **синусоиду**. В качестве амплитуды выбрать 0,5. Частоту можно выбрать любую между 100 Гц и 1000 Гц. Длительность можно выбрать любую.
4. Нажать на кнопку воспроизведения, затем на кнопку стоп.
5. Выделить дорожку, нажав на кнопку «Выделить» слева снизу на дорожке, после чего выбрать меню Анализ -> График спектра. Выбрать размер 2048, 4096, 8192. Сравнить графики. Убедиться, что пик совпадает с выбранной ранее частотой синусоиды. Закрыть окно с графиком.
6. Приблизить масштаб (Ctrl+колесико мыши вверх, либо Ctrl+1) до тех пор, пока не будут видны отдельные точки. Это – сэмплы. Отдалить масштаб на одно деление.
7. Добавить еще одну моно-дорожку. Создать в ней синусоиду с другой частотой, но теми же длительностью и амплитудой. Воспроизвести. Остановить. Сравнить внешний вид двух дорожек.
8. Выделить всё (Ctrl+A). Микшировать дорожки

(Треки -> Микшировать -> Свести и обработать в новый трек). Сравнить исходные и получившийся.

1. Нажать на кнопку «соло» на получившемся треке, чтобы не слышать остальных, после чего воспроизвести. Звук не должен отличаться от того, который был до микширования. Выделить этот трек кнопкой «Выделить», и посмотреть его спектр. Убедиться в наличии двух пиков, связанных с указанными частотами.
2. Создать новый проект. Создать новую монодорожку. Создать тон типа **меандр**. В качестве амплитуды выбрать 0,3. Частоту можно выбрать любую между 100 Гц и 1000 Гц. Длительность можно выбрать любую.
3. Воспроизвести, а также посмотреть на внешний вид. Выделить дорожку и открыть график спектра. Навести на самый большой пик курсор и посмотреть на панель «Пик». **Записать** частоту и громкость в децибелах. Повторить для трёх по громкости пиков.
4. Для каждого из пиков добавить новый трек, в котором создать **синусоиду** с частотой, совпадающей с частотой этого пика. После чего эту дорожку выделить и нормализовать (Эффекты -> Нормализация...) до громкости, соответствующей этому пику.
5. По очереди воспроизвести каждую из дорожек (с помощью кнопки «соло»). После чего воспроизвести все дорожки **кроме меандра** (с помощью кнопки «тихо»). После чего воспроизвести **только меандр** (с помощью кнопки «соло»).
6. Выделить все дорожки кроме **меандра**. Для этого нужно выделить всё (Ctrl + A), а затем нажать на кнопку «выделить» под **меандром**, удерживая при этом клавишу Ctrl. Микшировать выделенные треки (Треки -> Микшировать -> Свести и обработать в новый трек). Обратить внимание на внешний вид полученного трека.

**Часть 2. Понятие шума и фильтров.**

1. Создать новый проект в Audacity.
2. Генерировать синусоиду с длительностью в секунду и любой частотой в пределах от 100Гц до 440 Гц и амплитудой 0,7.
3. Создать новую монодорожку. Сгенерировать в ней белый шум (Создать -> Шум) с амплитудой 0,2 и длительностью в 2 секунды.
4. Выделить всё. Микшировать в новый трек. Провести анализ спектра получившейся дорожки, а также посмотреть на её внешний вид.
5. Выделить дорожку. Выбрать меню Эффекты->ФНЧ. Это фильтр низкой частоты. Выбрать 500 Гц и 6Дб, нажать ОК.
6. Посмотреть на внешний вид дорожки. Нажать на «Соло» на этой дорожке. Воспроизвести. Посмотреть на график спектра.
7. Отменить ФНЧ (Ctrl + Z). Снова выделить дорожку и применить ФНЧ, но теперь выбрать 12 Дб. Посмотреть на внешний вид и воспроизвести.
8. **Подключение стенда**
9. Подключите к стенду микрофон и динамики. Микрофон подключается к разъему MIC розового цвета, наушники – к разъему LINE-OUT зеленого цвета.
   1. Подсоедините стенд к источнику питания и инструментальному компьютеру, используя соответствующие кабели, входящие в комплект поставки.
   2. Запустите приложение AMP (IFPGAMP) и создайте в нем новый проект. Разместите проект в папке с вашей фамилией. Следует напомнить, что в пути проекта не должно быть русскоязычных символов.
   3. Выберите в качестве процессорной системы DE2-70 Media Computer / DE2-115 Media Computer.
   4. Выберите в следующем диалоговом окне тип программы –C Program.
   5. В качестве исходного файла выберите программу audio.c.
   6. В параметрах системы выберите: Host connection: USB-Blaster (тип соединения с хостом); Processor: Nios2 (процессорная система); Terminal device: JTAG\_UART (для обмена символами между стендом и инструментальным компьютером).
   7. В настройках памяти выберите Linker Section Presets: Basic (определение адресов для размещения кода). Программа должна начинаться с нулевого адреса.
10. **Запись и воспроизведение аудиосигнала на стенде.**
11. Загрузите программу из файла «audio.c», предварительно ознакомившись с её описанием. В Audacity создайте ноту в соответствии с вариантом (Таблица 1). Зафиксируйте микрофон в непосредственной близости от источника сигнала. Запустите программу и выполните запись аудиосигнала, созданного в приложении Audacity. Воспроизведите полученный сигнал и убедитесь в корректности его записи. Визуализируйте сигнал с помощью программы и зафиксируйте полученное изображение.
12. Не останавливая программу, создайте в Audacity ту же ноту второй октавы и повторите запись, воспроизведение и визуализацию сигнала. Сделайте вывод об изменении внешнего вида волны. Зафиксируйте вывод в отчет.
13. Перезапустите программу.
    1. В приложении Audacity установите громкость воспроизведения на 80%. Выполните запись и визуализацию ноты первой октавы.
    2. Затем повысив громкость воспроизведения до 100% повторите запись и визуализацию ноты. Положение микрофона при этом должно быть неизменным.
    3. Сделайте вывод об изменении внешнего вида сигнала в зависимости от его громкости.

*Таблица 1. Варианты.*

|  |  |
| --- | --- |
| **Вариант** | **Нота** |
| 1 | Ля 1 октавы (440 Гц) |
| 2 | Си 1 октавы (493 Гц) |
| 3 | До 2 октавы (523 Гц) |
| 4 | Ре 2 октавы (587 Гц) |
| 5 | Ми 2 октавы (659 Гц) |
| 6 | Фа 2 октавы (698 Гц) |
| 7 | Соль 2 октавы (784 Гц) |
| 8 | Ля 2 октавы (880 Гц) |
| 9 | Си 2 октавы (987 Гц) |

**5.Добавление аудиоэффектов в запись**

1. Загрузите программу из файла «effects.c», предварительно ознакомившись с её описанием. Запустите программу и по аналогии с предыдущем пунктом экспериментально определите, что происходит с сигналом при наложении на него определенных эффектов.

2. Сделайте так, чтобы воспроизведение замедлялось не в 2 раза, а в любое другое число раз. Зафиксируйте внесенные изменения в отчете.

3. Проделайте то же самое с ускорением и также зафиксируйте в отчете.

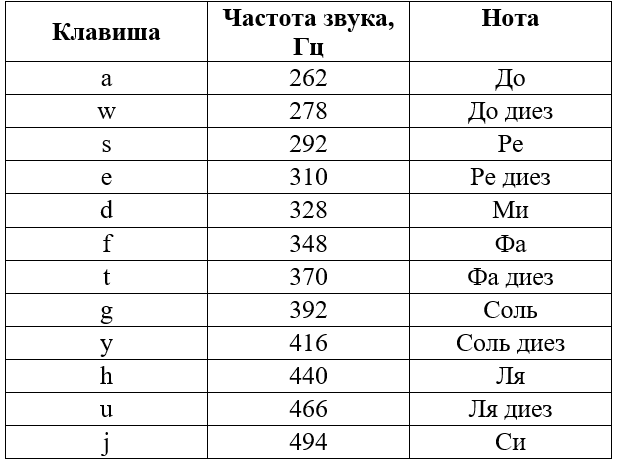
4. Задайте другую задержку для эффекта эхо. Попробуйте сделать эхо не двойным, а более многократным. Зафиксируйте все внесенные изменения в отчете.

1. **Генерация звука определенной частоты.**

Программа из файла «synth.c» (Приложение 3) использует файл «tones.h», содержащий сэмплы 12 нот (не рекомендуется открывать этот файл, он содержит очень много значений).

Необходимо в меню Program Settings в поле Additional compiler flags для установки максимального уровня оптимизации указать значение -О3.

*Таблица 2. Синтезатор (ноты и клавиши)*



1. Загрузите программу из файла «synth.c», предварительно ознакомившись с её описанием. Запустите программу, ознакомьтесь с её работой.

2\*. Используя клавиатуру, сыграйте мелодию, по нотам, которые находятся в папке с материалами лабораторной работы и называются «мелодия.png».

3\*. Напишите программу на языке Си, которая сыграет мелодию из предыдущего пункта.

**Отчетные материалы**

Отчет должен включать в себя:

1. Цель работы.
2. Краткие сведения из теоретической части.
3. Информацию по выполнению пунктов задания.
4. Краткое заключение.

**Контрольные вопросы**

1. Что такое звук?
2. Как называется звук, выходящий за пределы частот звука обычного?
3. Каковы характеристики звука?
4. От чего зависит громкость звука?
5. Как получить меандр?
6. Что такое белый шум?
7. В чём состоит эффект эха?
8. Как устроена аудиосистема в процессорной системе NIOS II?
9. В чём состоит замедление и ускорение оцифрованного звука?
10. Чем отличаются ноты соседних октав?

**Список источников**

1. Ефремов Н. В., Бородин А. А. Инструментальные средства проектирования и отладки цифровых систем на программируемом кристалле фирмы «Altera». Учебное пособие. — М.: ФГБОУ ВПО МГУЛ, 2012. — 151 с.
2. Altera Monitor Program. [Электронный ресурс] // URL: <http://www-ug.eecg.toronto.edu/desl/docs/Monitor_tutorial.pdf>.

**Приложение 1**

В приложение 1 следует поместить материал по теории в виде сжатого конспекта, со ссылками на источники. При желании, студенты могут поглубже познакомиться с этим материалом.

**Приложение 2**

**Листинг программы audio.c**

#define BUF\_SIZE 144000 ***//*** ***примерно 3 секунды записи (@ 48K samples/sec)***

#define BUF\_75 96 ***//*** ***75% заполненности аудио буфера (96 слов)***

void check\_KEYs(int \* KEY1, int \* KEY2, int \* counter)

{

volatile int \* KEY\_ptr = (int \*) 0x10000050; ***// адрес кнопок***

volatile int \* audio\_ptr = (int \*) 0x10003040; ***// адрес аудио порта***

int KEY\_value;

KEY\_value = \*(KEY\_ptr); ***// считывание значения нажатой кнопки***

while (\*KEY\_ptr); ***// ожидание, когда кнопка будет отжата***

if (KEY\_value == 0x2) ***// проверка нажата ли KEY1***

{

***// установка счетчика в 0***

\*counter = 0;

***// очистка аудио порта***

\*(audio\_ptr) = 0x4;

\*(audio\_ptr) = 0x0;

\*KEY1 = 1;

}

else if (KEY\_value == 0x4) ***// проверка нажата ли KEY2***

{

***// установка счетчика в 0***

\*counter = 0;

***// очистка аудио порта***

\*(audio\_ptr) = 0x8;

\*(audio\_ptr) = 0x0;

\*KEY2 = 1;

}

}

main()

{

int \* audio\_ptr = (int \*) 0x10003040;

int \* green\_LED\_ptr = (int \*) 0x10000010;

int fifospace;

int record=0, play = 0, buffer\_index = 0;

int left\_buffer[BUF\_SIZE];

int right\_buffer[BUF\_SIZE];

while(1)

{

check\_KEYs ( &record, &play, &buffer\_index );

if (record)

{

fifospace = \*(audio\_ptr + 1);

\*(green\_LED\_ptr) = 0x1;

if ((fifospace & 0x000000ff) <BUF\_75)

{

while ( (fifospace & 0x000000FF) && (buffer\_index < BUF\_SIZE) )

{

left\_buffer[buffer\_index] = \*(audio\_ptr + 2);

right\_buffer[buffer\_index] = \*(audio\_ptr + 3);

++buffer\_index;

if (buffer\_index == BUF\_SIZE)

{

***// воспроизведение завершено***

record = 0;

\*(green\_LED\_ptr) = 0x0; ***// выключение LEDG***

}

fifospace = \*(audio\_ptr + 1); ***// чтение из аудиопорта***

}

}

}

else if (play)

{

\*(green\_LED\_ptr) = 0x2; ***// включение LEDG\_1***

fifospace = \*(audio\_ptr + 1); ***//чтение из аудиопорта***

if ( (fifospace & 0x00FF0000) > BUF\_75 ) ***// проверяем поле WSRC***

{

***// вывод пока не опустеет буфер или не переполнится аудиопорт***

while ( (fifospace & 0x00FF0000) && (buffer\_index < BUF\_SIZE) )

{

\*(audio\_ptr + 2) = left\_buffer[buffer\_index];

\*(audio\_ptr + 3) = right\_buffer[buffer\_index];

++buffer\_index;

if (buffer\_index == BUF\_SIZE)

{

***// воспроизведение завершено***

play = 0;

\*(green\_LED\_ptr) = 0x0; ***// выключение LEDG***

}

fifospace = \*(audio\_ptr + 1); ***// чтение из аудиопорта*** }

}

}

}

}

**Содержимое файла «effects.c»**

else if(play) ***//ускоренное воспроизведение***

{

\*(green\_LED\_ptr) = 0x2; ***// turn on LEDG\_1***

fifospace = \*(audio\_ptr + 1); ***// read the audio port fifospace register***

if ( (fifospace & 0x00FF0000) > BUF\_75 ) ***// check WSRC***

{

***// output data until the buffer is empty or the audio-out FIFO is full***

while ( (fifospace & 0x00FF0000) && (buffer\_index < BUF\_SIZE) )

{

\*(audio\_ptr + 2) = left\_buffer[buffer\_index];

\*(audio\_ptr + 3) = right\_buffer[buffer\_index];

buffer\_index=buffer\_index + 2; ***//пропуск каждого второго значения при воспроизведении***

if (buffer\_index == BUF\_SIZE)

{

***// done playback***

play = 0;

\*(green\_LED\_ptr) = 0x0; ***// turn off LEDG***

}

fifospace = \*(audio\_ptr + 1); ***// read the audio port fifospace register***

}

}

}

//-----------------------------------------------------------------------------------

else if(play) ***//замедленное воспроизведение***

{

\*(green\_LED\_ptr) = 0x2; ***// turn on LEDG\_1***

fifospace = \*(audio\_ptr + 1); ***// read the audio port fifospace register***

if ( (fifospace & 0x00FF0000) > BUF\_75 ) ***// check WSRC***

{

***// output data until the buffer is empty or the audio-out FIFO is full***

while ( (fifospace & 0x00FF0000) && (buffer\_index < BUF\_SIZE) )

{

\*(audio\_ptr + 2) = left\_buffer[buffer\_index];

\*(audio\_ptr + 3) = right\_buffer[buffer\_index];

c++; ***//дублирование каждого второго значения при воспроизведении***

if (c==2)

{

++buffer\_index;

c=0;

}

if (buffer\_index == BUF\_SIZE)

{

***// done playback***

play = 0;

\*(green\_LED\_ptr) = 0x0; ***// turn off LEDG***

}

fifospace = \*(audio\_ptr + 1); ***// read the audio port fifospace register***

}

}

}

//-----------------------------------------------------------------------------------

else if(play) ***//эхо***

{

\*(green\_LED\_ptr) = 0x2; ***// turn on LEDG\_1***

fifospace = \*(audio\_ptr + 1); ***// read the audio port fifospace register***

if ( (fifospace & 0x00FF0000) > BUF\_75 ) ***// check WSRC***

{

***// output data until the buffer is empty or the audio-out FIFO is full***

while ( (fifospace & 0x00FF0000) && (buffer\_index < BUF\_SIZE) )

{

if (buffer\_index<=8000)***//если значение счетчика <=8000, то обычное воспроизведение***

{

\*(audio\_ptr + 2) = left\_buffer[buffer\_index];

\*(audio\_ptr + 3) = right\_buffer[buffer\_index];

}

else ***//если нет, то воспроизведение с наложение части сигнала проигранной ранее***

{

\*(audio\_ptr + 2) = left\_buffer[buffer\_index]+left\_buffer[buffer\_index-8000];

\*(audio\_ptr + 3) = right\_buffer[buffer\_index]+right\_buffer[buffer\_index-8000];

}

++buffer\_index;

if (buffer\_index == BUF\_SIZE)

{

***// done playback***

play = 0;

\*(green\_LED\_ptr) = 0x0; ***// turn off LEDG***

}

fifospace = \*(audio\_ptr + 1); ***// read the audio port fifospace register***

}

}

}

**Содержимое файла «synth.c»**

#include <stdio.h>

#define BUF\_SIZE 240000

#define BUF\_THRESHOLD 96 ***// 75% от 128 словного буфера***

#include ***"E:\efremov\audio\tones.h" // путь к файлу может быть другим!***

void check\_KEYs(int \* KEY1, int \* KEY2,int \* KEY3,int \* KEY4,int \* KEY5,int \* KEY6,

int \* KEY7,int \* KEY8,int \* KEY9,int \* KEY10,int \* KEY11,int \* KEY12,

long int \* counter,int \*stop)

{

int \* PS2\_ptr = (int \*) 0x10000100;

int work=0,work2=0,work3=0,work4=0,work5=0,work6=0,work7=0,work8=0,work9=0,work10=0,

work11=0,work12=0;

int PS2\_data,RVALID=0,Data;

PS2\_data =\*PS2\_ptr;

Data=PS2\_data&0x000000ff;

RVALID=PS2\_data&0x00008000;

if ((RVALID!=0) && (Data==0x1C))

if (\*stop==0) {work=1;}

else \*stop=0;

if ((RVALID!=0) && (Data==0x1D))

if (\*stop==0) {work2=1;}

else \*stop=0;

if ((RVALID!=0) && (Data==0x1B))

if (\*stop==0) {work3=1;}

else \*stop=0;

if ((RVALID!=0) && (Data==0x24))

if (\*stop==0) {work4=1;}

else \*stop=0;

if ((RVALID!=0) && (Data==0x23))

if (\*stop==0) {work5=1;}

else \*stop=0;

if ((RVALID!=0) && (Data==0x2B))

if (\*stop==0) {work6=1;}

else \*stop=0;

if ((RVALID!=0) && (Data==0x2C))

if (\*stop==0) {work7=1;}

else \*stop=0;

if ((RVALID!=0) && (Data==0x34))

if (\*stop==0) {work8=1;}

else \*stop=0;

if ((RVALID!=0) && (Data==0x35))

if (\*stop==0) {work9=1;}

else \*stop=0;

if ((RVALID!=0) && (Data==0x33))

if (\*stop==0) {work10=1;}

else \*stop=0;

if ((RVALID!=0) && (Data==0x3C))

if (\*stop==0) {work11=1;}

else \*stop=0;

if ((RVALID!=0) && (Data==0x3B))

if (\*stop==0) {work12=1;}

else \*stop=0;

if (\*KEY1==0) if (work==1)

{

\*counter = 0;

\*KEY1 = 1;

}

if (\*KEY2==0) if (work2==1)

{

\*counter = 0;

\*KEY2 = 1;

}

if (\*KEY3==0) if (work3==1)

{

\*counter = 0;

\*KEY3 = 1;

}

if (\*KEY4==0) if (work4==1)

{

\*counter = 0;

\*KEY4 = 1;

}

if (\*KEY5==0) if (work5==1)

{

\*counter = 0;

\*KEY5 = 1;

}

if (\*KEY6==0) if (work6==1)

{

\*counter = 0;

\*KEY6 = 1;

}

if (\*KEY7==0) if (work7==1)

{

\*counter = 0;

\*KEY7 = 1;

}

if (\*KEY8==0) if (work8==1)

{

\*counter = 0;

\*KEY8 = 1;

}

if (\*KEY9==0) if (work9==1)

{

\*counter = 0;

\*KEY9 = 1;

}

if (\*KEY10==0) if (work10==1)

{

\*counter = 0;

\*KEY10 = 1;

}

if (\*KEY11==0) if (work11==1)

{

\*counter = 0;

\*KEY11 = 1;

}

if (\*KEY12==0) if (work12==1)

{

\*counter = 0;

\*KEY12 = 1;

}

if ((RVALID!=0) && (Data==0xf0))

{

// \*counter = 48000;

\*KEY1=0;

\*KEY2=0;

\*KEY3=0;

\*KEY4=0;

\*KEY5=0;

\*KEY6=0;

\*KEY7=0;

\*KEY8=0;

\*KEY9=0;

\*KEY10=0;

\*KEY11=0;

\*KEY12=0;

\*stop=1;

work=0;

work2=0;

work3=0;

work4=0;

work5=0;

work6=0;

work7=0;

work8=0;

work9=0;

work10=0;

work11=0;

work12=0;

}

}

void main()

{

volatile int \* audio\_ptr = (int \*) 0x10003040; ***// адрес аудио порта***

int stop=0;

long int fifospace;

int key1 = 0, key2 = 0,key3=0,key4=0,key5=0,key6=0,key7=0,key8=0,key9=0,key10=0,key11=0,key12=0;

long int buffer\_index = 0;

while(1)

{

check\_KEYs(&key1,&key2,&key3,&key4,&key5,&key6,&key7,&key8,&key9,&key10,&key11,&key12, &buffer\_index,&stop);

if (key1==1)

{

fifospace = \*(audio\_ptr + 1); ***// чтение из аудиопорта***

if ( (fifospace & 0x00FF0000) > BUF\_THRESHOLD ) // check WSRC

{

***// вывод пока не будет пуст буфер или не переполнится аудиопорт*** while ( (fifospace & 0x00FF0000) && (buffer\_index < BUF\_SIZE) )

{

\*(audio\_ptr + 2) = tone1[buffer\_index];

\*(audio\_ptr + 3) = tone1[buffer\_index];

++buffer\_index;

check\_KEYs(&key1,&key2,&key3,&key4,&key5,&key6,&key7,&key8,&key9,&key10,&key11,&key12, &buffer\_index,&stop);

if (buffer\_index == BUF\_SIZE)

{

***// воспроизведение закончено***

key1 = 0; ***// выключение LEDG***

}

fifospace = \*(audio\_ptr + 1); // read the audio port fifospace register

}

}

}

if (key2==1)

{

fifospace = \*(audio\_ptr + 1); ***// чтение из аудиопорта***

if ( (fifospace & 0x00FF0000) > BUF\_THRESHOLD ) ***// проверка состояния WSRC***

{

***// вывод пока не будет пуст буфер или не переполнится аудиопорт*** while ( (fifospace & 0x00FF0000) && (buffer\_index < BUF\_SIZE) )

{

\*(audio\_ptr + 2) = tone2[buffer\_index];

\*(audio\_ptr + 3) = tone2[buffer\_index];

++buffer\_index;

check\_KEYs(&key1,&key2,&key3,&key4,&key5,&key6,&key7,&key8,&key9,&key10,&key11,&key12, &buffer\_index,&stop);

if (buffer\_index == BUF\_SIZE)

{

***// воспроизведение завершено***

key2 = 0;

}

fifospace = \*(audio\_ptr + 1); ***// чтение из аудиопорта*** }

}

}

if (key3==1)

{

fifospace = \*(audio\_ptr + 1); ***// чтение из аудиопорта*** if ( (fifospace & 0x00FF0000) > BUF\_THRESHOLD ) ***// проверка состояния WSRC***

{

***// output data until the buffer is empty or the audio-out FIFO is full***

while ( (fifospace & 0x00FF0000) && (buffer\_index < BUF\_SIZE) )

{

\*(audio\_ptr + 2) = tone3[buffer\_index];

\*(audio\_ptr + 3) = tone3[buffer\_index];

++buffer\_index;

check\_KEYs(&key1, &key2,&key3,&key4,&key5,&key6,&key7,&key8,&key9,&key10,&key11,&key12, &buffer\_index,&stop);

if (buffer\_index == BUF\_SIZE)

{

***// done playback***

key3 = 0;

}

fifospace = \*(audio\_ptr + 1); ***// read the audio port fifospace register***

}

}

}

if (key4==1)

{

fifospace = \*(audio\_ptr + 1); ***// read the audio port fifospace register***

if ( (fifospace & 0x00FF0000) > BUF\_THRESHOLD ) ***// check WSRC***

{

***// output data until the buffer is empty or the audio-out FIFO is full***

while ( (fifospace & 0x00FF0000) && (buffer\_index < BUF\_SIZE) )

{

\*(audio\_ptr + 2) = tone4[buffer\_index];

\*(audio\_ptr + 3) = tone4[buffer\_index];

++buffer\_index;

check\_KEYs(&key1, &key2,&key3,&key4,&key5,&key6,&key7,&key8,&key9,&key10,&key11,&key12, &buffer\_index,&stop);

if (buffer\_index == BUF\_SIZE)

{

***// done playback***

key4 = 0;

}

fifospace = \*(audio\_ptr + 1); ***// read the audio port fifospace register***

}

}

}

if (key5==1) {

fifospace = \*(audio\_ptr + 1); ***// read the audio port fifospace register***

if ( (fifospace & 0x00FF0000) > BUF\_THRESHOLD ) ***// check WSRC***

{

***// output data until the buffer is empty or the audio-out FIFO is full***

while ( (fifospace & 0x00FF0000) && (buffer\_index < BUF\_SIZE) )

{

\*(audio\_ptr + 2) = tone5[buffer\_index];

\*(audio\_ptr + 3) = tone5[buffer\_index];

++buffer\_index;

check\_KEYs(&key1, &key2,&key3,&key4,&key5,&key6,&key7,&key8,&key9,&key10,&key11,&key12, &buffer\_index,&stop);

if (buffer\_index == BUF\_SIZE)

{

***// done playback***

key5 = 0;

}

fifospace = \*(audio\_ptr + 1); ***// read the audio port fifospace register***

}

}

}

if (key6==1)

{

fifospace = \*(audio\_ptr + 1); ***// read the audio port fifospace register***

if ( (fifospace & 0x00FF0000) > BUF\_THRESHOLD ) ***// check WSRC***

{

***// output data until the buffer is empty or the audio-out FIFO is full***

while ( (fifospace & 0x00FF0000) && (buffer\_index < BUF\_SIZE) )

{

\*(audio\_ptr + 2) = tone6[buffer\_index];

\*(audio\_ptr + 3) = tone6[buffer\_index];

++buffer\_index;

check\_KEYs(&key1, &key2,&key3,&key4,&key5,&key6,&key7,&key8,&key9,&key10,&key11,&key12, &buffer\_index,&stop);

if (buffer\_index == BUF\_SIZE)

{

***// done playback***

key6 = 0;

}

fifospace = \*(audio\_ptr + 1); ***// read the audio port fifospace register***

}

}

}

if (key7==1)

{

fifospace = \*(audio\_ptr + 1); ***// read the audio port fifospace register***

if ( (fifospace & 0x00FF0000) > BUF\_THRESHOLD ) ***// check WSRC***

{

***// output data until the buffer is empty or the audio-out FIFO is full***

while ( (fifospace & 0x00FF0000) && (buffer\_index < BUF\_SIZE) )

{

\*(audio\_ptr + 2) = tone7[buffer\_index];

\*(audio\_ptr + 3) = tone7[buffer\_index];

++buffer\_index;

check\_KEYs(&key1, &key2,&key3,&key4,&key5,&key6,&key7,&key8,&key9,&key10,&key11,&key12, &buffer\_index,&stop);

if (buffer\_index == BUF\_SIZE)

{

***// done playback***

key7 = 0;

}

fifospace = \*(audio\_ptr + 1); ***// read the audio port fifospace register***

}

}

}

if (key8==1)

{

fifospace = \*(audio\_ptr + 1); ***// read the audio port fifospace register***

if ( (fifospace & 0x00FF0000) > BUF\_THRESHOLD ) ***// check WSRC***

{

***// output data until the buffer is empty or the audio-out FIFO is full***

while ( (fifospace & 0x00FF0000) && (buffer\_index < BUF\_SIZE) )

{

\*(audio\_ptr + 2) = tone8[buffer\_index];

\*(audio\_ptr + 3) = tone8[buffer\_index];

++buffer\_index;

check\_KEYs(&key1, &key2,&key3,&key4,&key5,&key6,&key7,&key8,&key9,&key10,&key11,&key12, &buffer\_index,&stop);

if (buffer\_index == BUF\_SIZE)

{

***// done playback***

key8 = 0;

}

fifospace = \*(audio\_ptr + 1); ***// read the audio port fifospace register***

}

}

}

if (key9==1)

{

fifospace = \*(audio\_ptr + 1);  ***// read the audio port fifospace register***

if ( (fifospace & 0x00FF0000) > BUF\_THRESHOLD ) ***// check WSRC***

{

***// output data until the buffer is empty or the audio-out FIFO is full***

while ( (fifospace & 0x00FF0000) && (buffer\_index < BUF\_SIZE) )

{

\*(audio\_ptr + 2) = tone9[buffer\_index];

\*(audio\_ptr + 3) = tone9[buffer\_index];

++buffer\_index;

check\_KEYs(&key1, &key2,&key3,&key4,&key5,&key6,&key7,&key8,&key9,&key10,&key11,&key12, &buffer\_index,&stop);

if (buffer\_index == BUF\_SIZE)

{

// done playback

key9 = 0;

}

fifospace = \*(audio\_ptr + 1); ***// read the audio port fifospace register***

}

}

}

if (key10==1)

{

fifospace = \*(audio\_ptr + 1); ***// read the audio port fifospace register***

if ( (fifospace & 0x00FF0000) > BUF\_THRESHOLD ) ***// check WSRC***

{

***// output data until the buffer is empty or the audio-out FIFO is full***

while ( (fifospace & 0x00FF0000) && (buffer\_index < BUF\_SIZE) )

{

\*(audio\_ptr + 2) = tone10[buffer\_index];

\*(audio\_ptr + 3) = tone10[buffer\_index];

++buffer\_index;

check\_KEYs(&key1, &key2,&key3,&key4,&key5,&key6,&key7,&key8,&key9,&key10,&key11,&key12, &buffer\_index,&stop);

if (buffer\_index == BUF\_SIZE)

{

***// done playback***

key10 = 0;

}

fifospace = \*(audio\_ptr + 1); ***// read the audio port fifospace*** ***register***

}

}

}

if (key11==1)

{

fifospace = \*(audio\_ptr + 1); ***// read the audio port fifospace register***

if ( (fifospace & 0x00FF0000) > BUF\_THRESHOLD ) ***// check WSRC***

{

***// output data until the buffer is empty or the audio-out FIFO is full***

while ( (fifospace & 0x00FF0000) && (buffer\_index < BUF\_SIZE) )

{

\*(audio\_ptr + 2) = tone11[buffer\_index];

\*(audio\_ptr + 3) = tone11[buffer\_index];

++buffer\_index;

check\_KEYs(&key1, &key2,&key3,&key4,&key5,&key6,&key7,&key8,&key9,&key10,&key11,&key12, &buffer\_index,&stop);

if (buffer\_index == BUF\_SIZE)

{

***// done playback***

key11 = 0;

}

fifospace = \*(audio\_ptr + 1); ***// read the audio port fifospace register***

}

}

}

if (key12==1)

{

fifospace = \*(audio\_ptr + 1); ***// read the audio port fifospace register***

if ( (fifospace & 0x00FF0000) > BUF\_THRESHOLD ) ***// check WSRC***

{

***// output data until the buffer is empty or the audio-out FIFO is full***

while ( (fifospace & 0x00FF0000) && (buffer\_index < BUF\_SIZE) )

{

\*(audio\_ptr + 2) = tone12[buffer\_index];

\*(audio\_ptr + 3) = tone12[buffer\_index];

++buffer\_index;

check\_KEYs(&key1, &key2,&key3,&key4,&key5,&key6,&key7,&key8,&key9,&key10,&key11,&key12, &buffer\_index,&stop);

if (buffer\_index == BUF\_SIZE)

{

**// done playback**

key12 = 0;

}

fifospace = \*(audio\_ptr + 1); ***// read the audio port fifospace register***

}

}

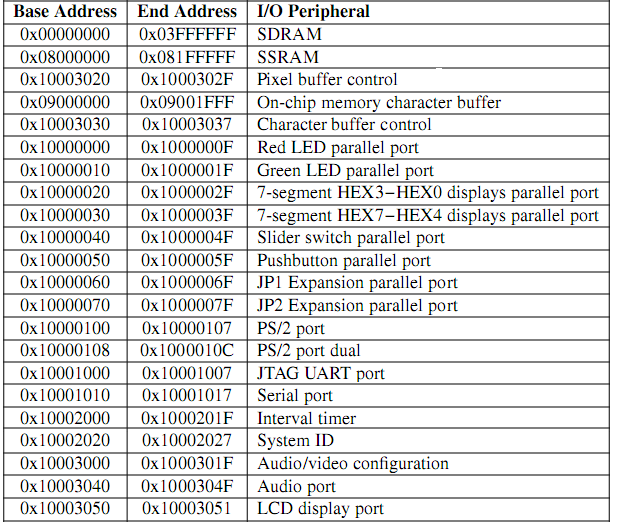
}

}

}

**Приложение 3**

В приложении 3, возможно, следует поместить таблицы с вариантами заданий, может быть некоторые шаблоны, которые студенты должны заполнить в процессе выполнения работы, а потом поместить в свой отчет.

**Карта памяти процессорной системы**