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

Факультет ИУ «Информатика и системы управления»

Кафедра ИУ-3 «Информационные системы и телекоммуникации»

Методические указания
к лабораторной работе 2
«Исследование процессов обработки сигналов»
по дисциплине «Микропроцессорные устройства обработки сигналов»

Для студентов, обучающихся по направлениям
2304002468, 2304007468 и 2302010065

Продолжительность работы 4 часа

Составители:
проф. каф. ИУЗ, д.т.н. В.С. Выхованец
преп. каф. ИУЗ А.И. Германчук

Москва, 2020

Содержание

1 Общие указания

Целью работы является изучение стандартных процедур обработки сигналов и данных, а также их реализация в интегрированной среде проектирования Code Composer Studio версии 5 и на микропроцессоре TMS320C5515 компании Texas Instruments Incorporated. При выполнении лабораторной работы используется симулятор микропроцессора TMS320C5515 или оценочноя плата TMS320C5515 DSP Evaluation Module (TMDXEVM5515) [1].

Для выполнения лабораторной работы на симуляторе микропроцессора TMS320C5515 необходимо загрузить и установить среду разработки Code Composer Studio версии 5.5.0 (данная версия является последней, включающей в себя симулятор процессора) с сайта компании Texas Instruments. Для этого необходимо сначала зарегистрироваться на сайте компании (www.ti.com), затем перейти на страницу загрузки среды разработки. Для этого можно воспользоваться поиском по сайту, либо перейти на страницу продукта через выпадающее меню *Design Resources* на главной странице сайта.

Лабораторная работа заключается в разработке программы обработки сигналов на языке программирования C, и ее сравнение по эффективности с аналогичной функцией из состава стандартной библиотеки для обработки сигналов [2].

Выполнение лабораторной работы состоит их отладки и профилирования программы, выполняющей обработку сигналов и данных, заданных в теме индивидуального задания. Реализация алгоритма обработки осуществляется двумя средствами: тестовой функции, написанной на языке C, и функцией из состава стандартной библиотеки обработки сигналов.

Профилирование программы заключается в измерении числа циклов микропроцессора, затраченных на выполнение обработки одних и тех же тестовых данных. Тестовые данные предусмотрены для каждой библиотечной функции и расположены в папке Examples стандартной библиотеки для обработки сигналов.

2 Задание на самостоятельную подготовку

Для подготовки к лабораторной работе необходимо:

– повторить материал лекций по архитектуре и организации микропроцессора TMS320C5515 и его программированию и изучить исходные тексты программ из Приложений А и Б;

– разработать исходный текст функции на языке C, реализующей алгоритм обработки данных, заданный в индивидуальном задании;

– подготовить исходный текст функции из состава стандартной библиотеки обработки сигналов, реализующей тот же алгоритм обработки данных, для ее выполнения в среде CCS.

3 Выполнение лабораторной работы

3.1 Создание проекта

3.1.1 Для создания проекта выберите пункты меню *File* → *New* → *Project*. Откроется форма выбора типа проекта.

3.1.2 В списке типов проекта необходимо выбрать пункт *Code Composer Studio* и подпункт *CCS Project.* Откроется форма нового проекта.

3.1.3 В поле *Project Name* введите имя проекта, например, *MPUOS_LAB2*. При установке флага *Use default location* проект будет создан в папке текущего пользователя. В противном случае место размещения папки проекта в файловой системе следует выбрать после нажатия на кнопку *Browse*.

3.1.4 В ниспадающем списке *Output Type* выберите *Executable* для создания исполняемой программы.

3.1.5 В разделе *Device* в ниспадающем списке выберите семейство микропроцессоров *C5500*, вариант *C551x* и *EVM5515*, соединение *Texas Instruments Simulator.*

3.1.6 В разделе *Project Template and Examples* выберите *Empty Project (with main.c)*. Нажмите кнопку *Finish*. CCS v5 создаст проект, который отобразится в окне *Project Explorer*.

3.1.7 Для включения в проект новых файлов в контекстном меню, открывающемся при нажатии правой кнопки мыши на имени созданного проекта, выберите пункт *New* и создайте файл *lab2.asm*. Для добавления в проект существующих файлов выберите пункт меню *Project* → *Add Files* и в появившемся окне *New Source File* задайте место размещения файла *lab1.asm* при установленном флаге *Copy File* для копирования этих файлов в папку проекта.

3.1.8 Введите подготовленные исходные тексты программ в окна текстового редактора *main.c* и *lab2.asm*.

3.2 Сборка проекта

3.2.1 Выполните сборку проекта, выбрав пункт меню *Project* → *Build Project*. Исправьте обнаруженные ошибки компиляции (ассемблирования) и повторите сборку снова.

3.2.2 В случае необходимости измените параметры сборки, выбрав в контекстном меню проекта (окно *Project Explorer*) пункт *Build Options*. Аналогичные настройки через контекстное меню, открывающееся при нажатии правой кнопки мыши на имени файла, можно выполнить и для каждого файла проекта.

3.3 Запуск отладчика

3.3.1 Установите фокус ввода на имени проекта в окне *Project Explorer* в выберите пункт меню *Run → Debug*. По этой команде запуститься отладчик и откомпилированный ранее загрузочный файл с именем проекта и расширением *.out* загрузится для исполнения в память целевой платформы.

3.3.2 Для выполнения загруженной программы по шагам используйте пункты меню *Target → Step Into (F5)* или *Target → Step Over (F6)*, а для запуска программы – *Target → Run (F8)*.

3.3.3 При пошаговой отладке программы содержимое регистров процессора контролируйте в окне *Registers*, содержимое ячеек памяти – в окне *Memory*, текст программы на языке С с подсвеченным текущим оператором – в окне редактора соответствующего исходного файла, а исполняемы код программы на языке ассемблера – в окне *Disassemly*.

3.3.4 Для ускорения прохода отладчика до требуемого оператора (команды) используйте точки останова. Для задания точки останова установите фокус ввода на требуемый оператор (команду) и выберите в контекстном меню пункт *New Breakpoint* или *Toggle Breakpoint*. После установки точек останова используйте команду запуска программы *Target → Run (F8)*, которая приведет к выполнению программы до тех пор, пока не наступит очередь выполнения команды, помеченной точкой останова.

3.4 Профилирование программы

3.4.1 Непосредственно после запуска отладчика (п. 3.3.1), задайте режим профилирования, выбрав пункт меню *Tools → Profile → Setup Profile Data Collection*. В появившемся окне *Profile Setup* нажмите кнопку *Activate*, установите флаг *Profile all Function for CPU Cycles* и сохраните произведенные изменения, нажав кнопку *Save*.

3.4.2 Откройте окно результатов профилирования *Profile*, выбрав пункт меню *Tools → Profile → Sview Function Profile Result*.

3.4.3 Запустите программу, выбрав пункт меню *Run – Resume*, и дождитесь остановки ее выполнения в режиме отладки.

3.4.4 В окне профилирование *Profile* проконтролируйте появление таблицы с результатами профилирования. Для сохранения результатов профилирования в формате электронной таблицы *.csv* выберите пункт контекстному меню *Data → Export All* и укажите имя и размещения файла с данными профилирования.

3.4.5 Завершите отладку, выбрав пункт меню *Run → Terminate*.

3.4.6 При анализе данных профилирования следует учесть назначение колонок:

– *Name* – задается имя профилируемой функции;

– *Calls* – число вызовов функции;

– *Excl Count Min* (*Max, Average, Total*) – максимальное (минимальное, среднее, общее) число циклов выполнения функции, исключая число циклов, необходимое для выполнения вызываемых в ее теле других функций;

– *Incl Count Min* (*Max, Average, Total*) – максимальное (минимальное, среднее, общее) число циклов выполнения функции, включая число циклов, необходимое для выполнения вызываемых в ее теле других функций;

– *Filename* – имя файла функции

– *Line Number* – номер строки в файле;

– *Start Address* – начальный адрес программы в памяти.

4 Результаты выполнения

Для проверки выполнения лабораторной работы преподавателем, студенту необходимо прислать архив с проектом, включающий в себя все ресурсы, необходимые для сборки проекта (файлы с исходным кодом, файлы библиотек, конфигурационные, разметки памяти и т. д.) и отчет по лабораторной работе.

Перед отправкой архива необходимо убедиться, что проект импортируется в среду разработки, успешно собирается и корректно работает, будучи разархивированным в произвольную папку на локальном диске. Наиболее частая причина возникновения ошибок сборки при таком методе добавления проекта заключается в том, что ресурсы, необходимые для сборки, подключаются к проекту по абсолютному пути.

4.1 Требования к отчету

Отчет по лабораторной работе должен содержать:

– титульный лист;

– содержание;

– описание индивидуального задания;

– результаты экспериментального исследования;

– выводы и рекомендации;

– список использованной литературы;

– приложение (текст программ на языке С).

– приложение (результат дизассемблирования функции на языке С);

– приложение (текст библиотечной функции с комментариями).

Оформление отчета осуществляется в соответствии с ГОСТ 2.105-95 [3], список литературы по ГОСТ Р 7.0.5–2008 [4].

5 Контрольные вопросы

Вопрос 1. Какие средства оптимизации программ на языке С использовались в лабораторной работе и почему?

Вопрос 2. Какие средства ускорения выполнения программы использованы в библиотечной функции и какой эффект от их применения?

Вопрос 3. Обоснуйте методику разработки программы для цифровой обработки сигналов с использованием высокоуровневых и низкоуровневых средств программирования.

Список литературы

[1] TMS320C5515 Evaluation Module (EVM). Technical Reference. – Spectrum Digital, 2010. – 76 p.

[2] TMS320C55x. DSP Library Programmer's Reference. – Texas Instruments, 2009. – 144 p.

[3] ГОСТ 2.105-95. Единая система конструкторской документации. Общие требования к текстовым документам. – М.: Изд-во стандартов, 2012. – 26 с.

[4] ГОСТ Р 7.0.5–2008. Библиографическая ссылка. Общие требования и правила составления. – М.: Изд-во стандартов, 2009. – 23 с.

Приложение А

Исходный текст модуля на языке С

```
#include "stdio.h"
#define len 3
/* Вектора A, B и C комплесных чисел длины 3*/
int VecA[2*len]={0x8475, 0x7B54, 0x957C, 0x10C3, 0x58B3, 0xEF42};
int VecB[2*len]={0x73F8, 0xA18E, 0x3894, 0x8457, 0xB6A5, 0x0A22};
int VecC[2*len];
/* Объявление внешних и внутренних функция */
extern void cmul1(int*, int*, int*, int);
void cmul2(int*, int*, int*, int);
/* Программа вызова функций умножения векторов комплексных чисел */
void main(void)
{
    int i, j;
    cmul1(VecA, VecB, VecC, len);
    for(i=0, j=0; i < len; i++, j+=2)
    {
        printf("%d, %d\n", VecC[j], VecC[j+1]);
    }
    cmul2(VecA, VecB, VecC, len);
    for(i=0, j=0; i < len; i++, j+=2)
    {
        printf("%d, %d\n", VecC[j], VecC[j+1]);
    }
}
/* Функция умножение векторов комплексных чисел 2*/
void cmul2(int* VecA, int* VecB, int* VecC, int lenght )
{
    register int i, j;
#pragma MUST_ITERATE (1)
    for(i=0, j=0; i < lenght; i++, j+=2)
    {
        VecC[j] = (VecA[j]*VecB[j] << 1) - (VecA[j+1]*VecB[j+1] << 1);
        VecC[j+1] = (VecA[j]*VecB[j+1] << 1)+ (VecA[j+1]*VecB[j] << 1);
    }
}
```

Приложение Б

Модуль на языке ассемблера

```
    .text
    .def    _cmul1
; void cmul1(int* AR0, int* AR1, int* AR2, int T0)
; AR0 – указатель на вектор A
; AR1 – указатель на вектор B
; AR2 – указатель на вектор C
; T0 – длина комплексных векторов
_cmul1:
    PSH     mmap(ST1_55)              ; Сохранить режим микропроцессора
    PSHBOTH XCDP                      ; Сохранить регистр XCDP
    BSET    FRCT                      ; Режим дробных чисел
    SUB     #1, T0                    ; Значение счетчика чикла
    || BCC  L1, T0 < #0               ; Переход по нулевой длине векторов
    MOV     T0, BRC0                  ; Загрузить счетчик циклов в BRC0
    || MOV  #1, T0                    ; В T0 смещение мнимой части
    MOV     #2, T1                    ; В T1 размер комплексного числа
    AMOV    XAR1, XCDP                ; Указатель на вектор B в XCDP
    || AMOV XAR2, XAR1                ; Указатель на вектор C в XAR1
    RPTBLOCAL L1 -1                   ; Начало цикла
    MPY     *AR0, *CDP+, AC0          ; AC0 = Re(Ai)*Re(Bi)
    :: MPY  *AR0(T0), *CDP+, AC1      ; :: AC1 = Im(Ai)*Re(Bi)
    MAS     *AR0(T0), *CDP+, AC0      ; AC0 -= Im(Ai)*Im(Bi)
    :: MAC  *(AR0+T1), *CDP+, AC1     ; AC0 += Re(Ai)*Im(Bi)
    MOV     pair(LO(AC0)), dbl(*AR1+) ; Запись Ai*Bi в Ci
L1:                                   ; Конец цикла
    POPBOTH XCDP                      ; Восстановить регистр XCDP
    POP     mmap(ST1_55)              ; Восстановить режим микропроцессора
    RET                               ; Возврат из функции
```

Приложение В

Результат дизассемблирования модуля на языке С

```
; void cmul2(int* VecA, int* VecB, int* VecC, int lenght )
; AR0 – указатель на VecA
; AR1 – указатель на VecB
; AR2 – указатель на VecC
; T0 – длина комплексных векторов lenght
cmul2:
     PSH      T2
     PSH      T3
     PSHBOTH  XAR5
     AADD     #-8,SP
     MOV      T0,*SP(#06h)
     MOV      XAR2,dbl(*SP(#04h))
     MOV      XAR1,dbl(*SP(#02h))
     MOV      XAR0,dbl(*SP(#00h))
; for(i=0, j=0; i < lenght; i++, j+=2)
     MOV      #0,T3
     || MOV   #0,T1
     MOV      *SP(#06h),AR1
     CMP      T3 >= AR1, TC1
     BCC      L6, TC1
;{   VecC[j] = VecA[j]*VecB[j]-VecA[j+1]*VecB[j+1];
L5:
     MOV      T1, AR1
     ADD      #1, AR1, T2
     MOV      T1, T0
     MOV      dbl(*SP(#00h)), XAR4
     ADD      #1, AR1, AR5
     MOV      dbl(*SP(#02h)), XAR1
     MOV      dbl(*SP(#02h)), XAR3
     MOV      dbl(*SP(#00h)), XAR2
     MPYM     *AR1(T0), *AR4(T0), AC0
     || MOV   T2, T0
     AADD     AR5, AR3
     MASM     *AR3, *AR2(T0), AC0, AC0
     || MOV   T1,T0
     MOV      dbl(*SP(#04h)), XAR3
     MOV      AC0, *AR3(T0)
;    VecC[j+1] = VecA[j]*VecB[j+1]+VecA[j+1]*VecB[j];
     MOV      T1, AR2
     MOV      T1, AR1
     ADD      #1,AR2, T0
     MOV      dbl(*SP(#02h)), XAR2
     ADD      #1,AR1, T2
     MOV      dbl(*SP(#00h)), XAR1
     MOV      dbl(*SP(#02h)), XAR3
     MOV      dbl(*SP(#00h)), XAR4
     AADD     T1, AR2
     MPYM     *AR2, *AR1(T0), AC0
     || MOV   T1, T0
     AADD     T2, AR3
     MACM     *AR3, *AR4(T0), AC0, AC0
     || MOV   T1, AR1
     ADD      #1, AR1, T0
     MOV      dbl(*SP(#04h)), XAR3
     MOV      AC0, *AR3(T0)
; i < lenght; i++, j+=2
```

```
     ADD      #1, T3
     ADD      #2, T1
     MOV      *SP(#06h), AR1
     CMP      T3 < AR1, TC1
     BCC      L5, TC1
; }
L6:
     AADD     #8, SP
     POPBOTH  XAR5
     POP      T3
     POP      T2
     RET
```