Некоммерческое акционерное общество

АЛМАТИНСКИЙ УНИВЕРСИТ ЭНЕРГЕТИКИ И СВЯЗИ

Кафедра «Электроника»

ЦИФРОВЫЕ ПРОЦЕССОРЫ ОБРАБОТКИ СИГНАЛОВ

Методические указания по выполнению лабораторных работ для студентов специальности 5В071600 - Приборостроение

Алматы 2015

СОСТАВИТЕЛИ: С.Н. Петрищенко, Н.С. Бакирова, А.Нурланулы. Цифровые процессоры обработки сигналов. Методические указания по выполнению лабораторных работ для студентов специальности 5В071600-Приборостроение. - Алматы: АУЭС, 2015. – 32 с.

Методические указания содержат рабочие задания к шести лабораторным работам по микропроцессорам специального назначения. Дана краткая теоретическая справка, методика проведения и обработки данных, приведен перечень рекомендуемой литературы и контрольные вопросы.

Занятия проводятся на персональных компьютерах с применением симулятора сигнального процессора TMS320C50 и стенда УМК Texas Instruments-1 для изучения микроконтроллера семейства MSP430.

Табл. 4, рис. 4, библиогр. – 6 назв.

Методические указания предназначены для студентов специальности 5В071600 – Приборостроение.

Рецензент: доцент А.А.Куликов

Печатается по плану издания некоммерческого акционерного общества «Алматинский университет энергетики и связи» на 2015 г.

1 Лабораторная работа №1. Работа с симулятором процессора TMS320C50

Цель работы: получить практические навыки работы с симулятором процессора и освоить основные приемы работы с ним.

1.1 Краткая теоретическая справка

Симулятор предназначен для программной имитации работы процессора TMS320C50 и отладки программ, предназначенных для выполнения на данном процессоре. Имитатор выполняет программы не в реальном масштабе времени. При выполнении программы в нем возможен контроль, а также модификация (изменение) состояния основных регистров процессора и содержимого памяти при тех или иных операциях.

Программа симулятора *SIM5X* обладает дружественным интерфейсом и позволяет вводить основные команды, управляющие работой, как в командной строке, так и посредством манипулятора “мышь”, используя экранное меню. Симулятор позволяет загружать и выполнять файлы, полученные в результате трансляции и компоновки программы, написанной на языке ассемблера.

Процесс подготовки исполняемых программ для процессора можно разделить на четыре этапа:

- создание одного или нескольких модулей-файлов *(имя файла).asm,* с исходным текстом программы, написанным на языке ассемблера;
- трансляция полученного файла *(имя файла).asm* программой *DSPA.EXE* и создание объектного файла *(имя файла).obj* и листинга *(имя файла).lst.* При этом, помимо перевода текста программы, производится проверка исходного текста программы на наличие синтаксических ошибок ассемблера, сообщения о которых приводятся в листинге;
- написание командного файла компоновки *(имя файла).cmd* с указанием компонуемых программных модулей, порядка соединения секций и используемых областей памяти для размещения этих секций;
- компоновка модулей и секций с помощью компоновщика *DSPALNK.EXE* и получение выходного файла *(имя файла).out* и файла *(имя файла).map,* позволяющего контролировать правильность распределения памяти и соединения секций, заданные в командном файле *(имя файла).cmd.*

Графическая панель, формируемая программой симулятора, организована по оконному типу и дает возможность:

- загружать исполняемые программы и просматривать их дизассемблерную версию;
- выделять, перемещать и менять размеры окон;
- выполнять программы в пошаговом режиме и контролировать их выполнение, следя за изменениями, происходящими с содержимым регистров и областей памяти процессора;

- задавать точки останова при выполнении программы;
- вводить различные команды через командное меню, посредством клавиатуры или «мышью»;
- контролировать время выполнения программ.

После запуска программы симулятора на экране монитора появляются четыре окна:

1) Окно *DISASSEBLY* - для отлаживания текста дизассемблированной программы.

2) Окно *CPU* - для отражения состояния аккумулятора процессора (ACC) и его буфера (ACCB), регистра результата умножения (PREG), программного счетчика (PC), регистрового файла, состоящего из восьми вспомогательных регистров (AR1 – AR0) и индексного регистра (INDX), регистров состояния (ST0, ST1 и PMST) и других регистров.

3) Окно *MEMORY* – для отражения памяти процессора.

4) Окно *COMMAND* – для введения разных команд процессора.

Примером программы, на базе которой будут отрабатываться основные шаги по подготовке исполняемых программ для процессора, является программа с именем *SIM*. Она представлена в файле *sim.asm* и состоит из нескольких частей, в каждой из которых реализуется некоторая операция.

В таблице 1.1 представлена карта размещения в памяти данных переменных, используемых в программе.

Т а б л и ц а 1.1

Переменная	Адрес ячейки памяти данных (Hex)	Начальное значение переменной (Hex)
X	0800	1
Y	0801	3
Z	0802	0
X1	0803	4
Y1	0804	5
Z1	0805	0
X2	0806	0a
Y2	0807	0b
Z2	0808	0c

Ниже представлен полный текст программы *SIM*:

```
; Программа выполнения простейших операций на TMS320C50
            .version    50
            .mmregs                 ; Разрешение использования ;символьных
                                    ;имен  регистров процессора.
; Задание значений переменных в памяти данных
            .data
X           .word       1           ; Начальное значение  X
Y           .word       2           ; Начальное значение  Y
Z           .word       0           ; Начальное значение Z
X1          .word       4           ; Начальное значение  X1
Y1          .word       5           ; Начальное значение Y1
```

```
Z1          .word       0           ; Начальное значение Z1
X2          .word       0ah         ; Начальное значение X2
Y2          .word       0bh         ; Начальное значение Y2
Z2          .word       0ch         ; Начальное значение Z2
; Резервирование области памяти размером 6 ячеек
            .bss        M1,6        ; Символический адрес ;первой
                                    ячейки M1
; Таблица векторов прерывания
            .sect       " Vectors "
RESET       B           START       ; Начальная строка таблицы
                                    ;векторов прерывания –
                                    ;переход к основной программе
;       Основная программа
            .text
START       LDP         #0          ; Указатель страницы памяти данных
            OPL         #04h,PMST   ; Задать значение регистра PMST
            ZAP
            CLRC        TNTM        ; Запретить режим прерывания
            SPM         0           ; Установить нулевой сдвиг PREG
L1:
            LDP         #X          ;Указать на страницу памяти, хранящую X
            ZAP                     ;Обнулить ACC и регистр PREG
            LACC        X           ;Загрузить в ACC содержимое ячейки X
            ADD         Y           ;Прибавить к ACC содержимое ячейки Y
            SACL        Z           ;Сохранить содержимое младшего слова ACC
                                    в ячейке Z
L2:
            LDP         #X1         ;Указать на страницу памяти, хранящую X1
            ZAP                     ;Обнулить ACC и регистр PREG
            LT          X1          ;Загрузить в T-регистр содержимое ячейки X1
            MPY         Y1          ;Перемножить X1 и Y1. Результат в PREG
            SPL         Z1          ;Сохранить содержимое младшего слова
                                    ;PREG в ячейке Z1

L3:
            LDP         #X1         ;Указать на страницу памяти, хранящую X2
            ZAP                     ;Обнулить ACC и регистр PREG
            LACC        Y1          ;Загрузить в ACC содержимое ячейки X2
            SUB         X1          ;Вычесть из ACC содержимое ячейки Y2
            SACL        Z2          ;Сохранить содержимое младшего слова ACC
                                    в ячейке Z2
L4:
            LDP         #X          ;Указать на страницу памяти, хранящую X
            ZAP                     ;Обнулить ACC и регистр PREG
            LACC        Z1          ;Загрузить в ACC содержимое ячейки Z1
            SACL        X           ;Сохранить содержимое младшего слова ACC
                                    в ячейке X
L5:
            LDP         #Y2         ;Указать на страницу памяти, хранящую Y
            MAR         *,AR0       ; Сделать активным регистр AR0
            LAR         AR0,Y2      ;Загрузить в AR0 содержимое ячейки Y2
L6:
```

```
        LDP     #Z          ;Указать на страницу памяти, хранящую X
        LAR     AR6,#Z      ;Загрузить в AR6 адрес ячейки Z
L7:
        LDP     #X1         ;Указать на страницу памяти, хранящую X1
        ZAP                 ;Обнулить ACC и регистр PREG
        LACC    #07fffh     ;Загрузить в ACC число 7fff (Hex)
        SACL    Y2          ;Сохранить содержимое младшего слова ACC
                            в ячейке Y2
        SACB                ;Сохранить содержимое ACC в ACCB
        SAMM    AR1         ;Сохранить содержимое ACC в регистре AR1

L8:
        LDP     #X          ;Указать на страницу памяти, хранящую X
        ZAP                 ;Обнулить ACC и регистр PREG
        LT      X           ;Загрузить в регистр T  переменную X
        MPY     X1          ;Умножить X на X1 результат в регистре
                            PREG
        LTA     Y           ; Загрузить в регистр T  переменную Y,
                            ;предыдущее произведение из PREG
                            ;добавляется в ACC
        MPY     Y1          ; Умножить Y на Y1 результат в регистре ;
                            ;PREG
        LTA     Z           ;Загрузить в регистр T  переменную Z,
                            ;предыдущее произведение из PREG
                            ;добавляется в ACC
        MPY     Z1          ; Умножить Z на Z1 результат в регистре
                            ;PREG
        APAC                ;предыдущее произведение из PREG
                            ;добавляется в ACC
        SACL    X2          ;Сохранить содержимое младшего слова
                            ;ACC в ячейке X2
        B       START       ;Переход к началу программы для
                            ;повторения
        .end                ; Конец программы
```

1.2 Порядок выполнения работы

1.2.1 Создать свою рабочую папку и в ней свой файл *(имя файла).asm* с текстом программы *SIM*.

1.2.2 Скопировать в созданную папку файлы транслятора и компоновщика *dspa.exe* и *dsplnk.exe.*

1.2.3 Оттранслировать полученный файл *(имя файла).asm,* получить объектный файл *(имя файла).obj* и листинг - *(имя файла).lst* с помощью программы *dspa.exe*. Для этого в режиме *MS DOS* войти в свою папку и набрать в командной строке следующую команду:

dspa.exe -lcs (имя файла).asm.

При отсутствии ошибок произвести операцию компоновки и получить выходной файл *(имя файла).out* и файл *(имя файла).map*, набрав команду вида:

dsplnk.exe (имя файла).cmd.

Полученный файл *(имя файла).out* используется при работе с программой симулятора.

1.2.4 Скопируйте в свою рабочую папку файлы:

1) *sim5x.exe* - программа симулятора.

2) *alias.bat* - файл с набором макрокоманд имитатора.

3) *siminit.cmd* – управляющий файл для имитатора.

4) *sim.out* – учебная программа.

1.2.5 Запустите программу симулятора. На экране монитора появятся окна графического интерфейса, отражающих состояние основных элементов архитектуры процессора. Изучите состав и расположение этих окон.

1.2.6 Для загрузки файла *sim.out*, с исполняемой программой, используйте меню *LOAD*, выбрав в ней команду *LOAD*. В открывшимся окне наберите имя файла исполняемой программы с расширением и нажмите клавиши *Enter* и *F8*.

1.2.7 На экране в окне *MEMORY*, начиная с адреса *0800* (карта памяти), найдите содержимое ячеек памяти, где хранятся переменные *X,Y,Z,X1,Y1,Z1,X2,Y2,Z2*.

1.2.8 Выполните программу в пошаговом режиме, используя пункт меню *F8*. Проследите за изменением содержимого регистров процессора и ячеек памяти данных, хранящих переменные, при выполнении каждой команды. Проследите за соответствием результатов ожидаемым.

1.2.9 Вернитесь в начало программы командой *RESTART*.

1.2.10 Установите в программе несколько точек останова в местах меток программы. Установить точку останова в нужном месте программы можно, поместив на нее курсор и нажав левую кнопку мыши. При этом выбранная строка программы в окне *DISSASEMBLE* будет подсвечена. Выполнение программы между точками останова происходит при выполнении команды *RUN* (или нажатии клавиши *F5*). Для снятия точки останова нужно снова установить курсор в нужном месте программы и нажать левую кнопку мыши.

1.2.11 Выведете в окне *WATCH* значения переменных *X2,Y2,Z2*, а также значения регистра *AR0* и аккумулятора *ACC* в форматах *x* (шестнадцатеричный) и *d* (десятичный). Например, для вывода значений *X2* и регистра *AR0* нужно войти в пункт меню *WATCH*, выбрать команду *ADD* и набрать в открывшемся окне:

1) Для переменных:

а) адрес (в строке Expression), например, ***0x0806**;

б) имя переменной (в строке Label), например, *x2*;

в) формат вывода (в строке Format), например, *x* или *d*.

2) Для регистров:

а) имя регистра, например, *AR0*;

б) формат вывода, например, *x* или *d*.

Удаление переменной или регистра из окна *Watch* производится выбором стоки *Delete* в пункте меню *Watch* и указанием номера строки в окне удаляемой переменной.

1.2.12 Для измерения времени выполнения программы в тактах основной частоты работы процессора необходимо вывести в окне *WATCH* значение регистра *clk*. Для измерения времени выполнения нужного фрагмента программы необходимо в начале и в конце этого фрагмента установить точки останова, выполнить программу до начала фрагмента (команда *RUN*) и затем выполнить команды программы до конца фрагмента (снова команда *RUN*).

Измерьте время выполнения основной части программы, учитывая, что цикл команды исследуемого процессора равен 50 нс.

1.2.13 Вернитесь в начало программы с помощью команды *RESTART*. Выполните первые 10 команд, для этого наберите команду *STEP 10*.

1.2.14 Вернитесь в начало программы и выполните часть программы с начала до метки *L3*, набрав команду *GO* и адрес команды, соответствующей метке L3.

Выйдите из программы симулятора, набрав в командной строке *QUIT*.

По результатам лабораторной работы отчет не составляется. В конце работы проводится защита. При ответах на вопросы необходимо подтвердить навыки, полученные при работе с симулятором.

1.3 Содержание отчета

1.3.1 Текст программы с подробными комментариями.

1.3.2 Ответы на контрольные вопросы.

1.4 Контрольные вопросы

1.4.1 Поясните команды запуска программы на выполнение. Для чего может использоваться каждый из режимов запуска?

1.4.2 Поясните назначение основных регистров процессора и их место в памяти процессора.

1.4.3 Для чего используются точки останова при выполнении программы? Чем отличается пошаговый режим выполнения программ от других режимов? В каких форматах могут быть представлены числа в окнах имитатора?

1.4.4 Как сохранить содержимое выбранной области памяти?

1.4.5 Как в окне *MEMORY* вывести информацию о содержимом ячеек памяти данных, начиная с адреса *0200*?

1.4.6 Создайте окно просмотра *WATCH* и выведите в нем информацию о содержимом ячейки *Z2*.

1.4.7 Проделайте операцию сложения чисел *0FF* и *9*. Выполните эту операцию, используя программу *SIM*.

2 Лабраторная работа №2. Представление данных в процессорах с фиксированной запятой

Цель работы: изучение форматов представления целых, дробных, положительных и отрицательных чисел в цифровом процессоре обработки сигналов TMS320C50.

2.1 Краткая теоретическая справка

Система команд процессора с фиксированной запятой ориентирована на обработку двоичных чисел, представленных в дополнительном коде, который позволяет представлять как положительные, так и отрицательные числа при знаковом старшем бите. В зависимости от положения запятой используют целые или дробные числа.

В 16-разрядных процессорах TMS320C50 шина данных 16-ти разрядная, а аккумулятор (ACC) - 32-х разрядный. Целочисленная арифметика позволяет представлять большие числа, но создает проблемы с сохранением результата в 16-ти разрядной ячейке памяти. При использовании дробной арифметики 32-х разрядный аккумулятор позволяет увеличить точность вычислений. Для сохранения результата в 16-ти разрядной ячейке памяти используются старшие разряды аккумулятора. При этом потеря точности несущественна, так как младшие разряды аккумулятора содержат шум округлений.

На рисунке 2.1 показано соответствие исходных дробных чисел и чисел в процессоре в десятичной системе и дополнительном коде.

Рисунок 2.1

Формат представления дробных чисел определяется как формат Q15 (15-число значащих разрядов справа от разделительной точки). Промежуточные значения вычисляемых дробных величин на выходе

умножителя и АЛУ могут быть представлены в других форматах, в частности, Q31 или Q30.

Чтобы записать исходное дробное число в формате Q15 процессора, его необходимо умножить на 32768, отбросить дробную часть и перевести в 16-ричную систему (в дополнительном коде, если исходная дробь отрицательна). Ассемблер позволяет производить эти вычисления при выполнении трансляции программы.

Для перевода результата вычислений, произведенных в процессоре из формата Q15 в обычный вид, необходимо:

1) Перевести число из 16-ричной системы в десятичную.
2) Разделить полученное число на 32768.

Операции алгебраического сложения/вычитания чисел в дополнительном коде как беззнаковых чисел выполняются в АЛУ по правилам обычной двоичной арифметики.

Операция умножения выполняется в умножителе, который реализует алгоритм умножения двоичных чисел в дополнительном коде.

При умножении целых чисел результат содержится во всех 32 разрядах. При умножении чисел, величина которых не превышает 2^8, результат будет находиться в младшем слове регистра P.

При умножении дробных чисел в формате Q15 старшее слово 32-разрядного регистра P, в котором находится результат, является его округленным значением в формате Q14 (с двумя знаковыми разрядами) с отброшенными младшими значащими разрядами.

Возможны два варианта решения ликвидации «лишних» знаковых разрядов:

1) Сдвигом на один бит влево содержимого аккумулятора, если необходимо сохранить в памяти его старшее слово.

2) Путем сдвига содержимого регистра P влево.

Второй вариант сдвига можно осуществить сдвигателем P-регистра, который управляется 2-х разрядным регистром PM, расположенным в регистре управления ST1. Содержимое PM можно менять командой SPM с указанием величины сдвига влево на 1 или 4 разряда или вправо на 6 разрядов. PM устанавливается в 0 после общего сброса или командой SPM 0.

Если установить режим сдвига на 1 бит влево, то при сохранении в аккумуляторе содержимого P-регистра формат представления чисел в аккумуляторе будет Q31 (при умножении чисел в формате Q15).

Режим расширения знака процессора определяется состоянием бита SXM (Sign Extension Mode), который расположен в регистре состояния процессора. Он показывает, использовать расширение знака или нет.

Установка бита выполняется командами:

SETC SXM; SXM =1 - установка бита (АЛУ оперирует с числами в дополнительном коде).

CLRC SXM; SXM = 0 - сброс бита (АЛУ оперирует с беззнаковыми числами).

После общего сброса процессора бит SXM установлен в 1.

Двоичное представление дробных чисел в дополнительном коде имеет следующие особенности: если к (+1) в двоичном коде прибавить 1/32767, получится –1, т.е. число кардинально изменяется: вместо большого положительного становится большим отрицательным.

В процессоре TMS320C50 бит контроля переполнения OV (Overflow) располагается в регистре состояния и устанавливается в случае, когда величина в аккумуляторе переходит границу между большими положительными и отрицательными числами, а также в случае, если осуществлен перенос в 33-ий, несуществующий бит.

В процессорах с фиксированной запятой предусмотрен режим работы с переполнением, когда при выходе положительного числа за пределы разрядной сетки устанавливается наибольшее возможное положительное число, а при выходе отрицательного числа за пределы разрядной сетки устанавливается наибольшее по модулю отрицательное число.

Режим работы с переполнением или без определяется значением бита OVM регистра состояния ST1. При значении 1 происходит работа в режиме с переполнением, при значении 0 - без переполнения.

Значение бита OVM, равное 0, может быть установлено командой ROVM или CLRC OVM, значение 1 - командой SOVM или SETC OVM.

Ниже приведен пример стандартной конфигурации регистров состояния процессора для организации процедуры его инициализации.

```
SETC    SXM   ; разрешить использовать дополнительный код;
CLRC    OVM   ; разрешить переполнение;
SPM     1     ; осуществить сдвиг на 1 бит влево при копировании
              ; P – регистра в аккумулятор.
```

2.2 Порядок выполнения работы

2.2.1 Скопируйте в свою рабочую папку файлы:

1) *sim5x.exe* - программа имитатора.

2) *alias.bat* - файл с набором макрокоманд имитатора.

3) *siminit.cmd* - управляющий файл для имитатора.

4) *lab3.out* - исходный текст программы.

2.2.2 Запустите имитатор процессора, набрав команду *sim5x.exe*.

2.2.3 Вычислить выражения для целых и дробных чисел (приведены ниже). Используемые значения коэффициентов приведены в таблице TABLE. При использовании дробных чисел их значения нормируются к единице, которой соответствует величина 32768.

2.2.4 Размещение переменных и констант в памяти.

```
        .bss    A,13
B               .set    A+1
C               .set    B+1
D               .set    C+1
```

```
E           .set    D+1
F           .set    E+1
X           .set    F+1
I           .set    X+2
IL          .set    I+1
ILL         .set    IL+1
Y           .set    ILL+1
```

2.2.5 Таблица значений констант, размещенных в памяти программ

```
TABLE       .word   32768*9/10, 32768*8/10, 32768*7/10; A=0.9, B=0.8, C=0.7
            .word   32768*6/10, 32768*-1, 32768*4/10    ; D=0.6, E=-1.0, F=0.4
            .word   32768*0,0,5,100                     ; X=0, XA=0, I=5, IL=100
            .word   1000,0,0                            ; ILL=1000, Y=0, Y+1=0
```

2.2.6 Перепись констант из памяти программ в память данных

```
.sect   "Vectors"
RESET       B       M1
.text
M1:
*  Перепись констант из памяти программ в память данных в зарезервированное место

    LAR    AR1,#A
    MAR    *,AR1
    LACC   #TABLE
    RPT    #12
    TBLR   *+
```

Наберите команду *mem 0·200*. Выполните команду *Go INT*. При этом должна быть выполнена часть программы до метки *INT* и в памяти данных появятся переписанные константы из памяти программ.

2.2.7 Работа с целыми числами в младших разрядах аккумулятора. Вычисление выражения Y = (I·I) + (IL·IL) + (ILL·ILL).

```
INT:
    LDP    #A    ;указатель страницы памяти на размещение A
    ZAC          ; (ACC)=0
    SACL   Y+1    ; сохранение младших разрядов ACC - ACCL
    LT     I
    MPY    I    ; (P)=I*I
    LTA    IL     ; (ACC)=I*I
    SACL   Y+1    ; сохранение ACCL
    MPY    IL           ; (P)=IL*IL
    LTA    ILL    ; (ACC)=I*I+IL*IL
    SACL   Y+1    ; сохранение  ACCL
    MPY    ILL          ; ILL*ILL
    APAC         ; (ACC)=(I*I)+(IL*IL)+(ILL*ILL)
    SACL   Y+1    ; сохранение младшего слова ACCL
    SACH   Y      ; сохранение старшего слова ACCH
```

Выполните фрагмент программы до метки FLOAT1 по шагам.

Сравните результаты вычисления всех частичных сумм и последнего результата с точными значениями, полученными при ручном расчете.

Обратите внимание, что ошибка появляется при переполнении младшего слова ACC после прибавления третьего произведения.

2.2.8 Работа с дробными числами. Вычисление выражения

$$X = (A \cdot B) + (C \cdot D) + (E \cdot F).$$

1. Влияние опции режима переполнения OVM:

а) отсутствие сдвига результата при передаче его из ACC в ЗУ;

```
FLOAT1:
LDP    #A
     SPM     0       ; PM = 0
     CLRC      OVM     ; OVM = 0
     CALL    Q31       ; вызов п/программы вычисления X
FLOAT2:
     SETC      OVM     ; PM = 0, OVM = 1
     CALL    Q31       ; вызов п/программы вычисления X
* Подпрограмма вычисления X без свига на 1 разряд при передаче из ACC в ЗУ

Q31:
      ZAC
      SACH    X
      LT      A
      MPY     B
      LTA     C
      SACH    X
      MPY     D
      LTA     E
      SACH    X
      MPY     F
      APAC
      SACH    X
      RET
      .end
```

б) наличие сдвига результата при передаче его из ACC в ЗУ;

```
FLOAT3:
      LDP     #A
      SPM     0       ; PM = 0
      CLRC      OVM     ; OVM  0
            CALL    Q30       ; вызов п/программы вычисления X
FLOAT4:
      SETC      OVM     ; PM = 0, OVM = 1
      CALL    Q30       ; вызов п/программы вычисления X

* Подпрограмма вычисления X со свигом на 1 разряд при передаче из ACC в ЗУ
Q30:
      ZAC                 ; (ACC)=0
      SACH    X+1,1   ; сохранение старшего слова ACC со сдвигом на 1 разряд
      LT      A  ; (T)=A
      MPY     B           ; (P)=A*B
      LTA     C ; (T)=C  (ACC)=A*B
      SACH    X,1       ; сохранение старшего слова ACC со сдвигом на 1 разряд
```

```
      MPY    D          ; (P)=C*D
      LTA    E ; (T)=E   (ACC)=A*B+C*D
      SACH   X,1        ; сохранение старшего слова ACC со сдвигом на 1 разряд
      MPY    F; (P)=E*F
      APAC              ; (ACC)=A*B+C*D+E*F
      SACH   X,1        ; сохранение старшего слова ACC со сдвигом на 1 разряд
      RET
```

2. Влияние опции PM - сдвига результата умножения при передаче из регистра P в АЛУ при отсутствии сдвига общего результата при передаче из ACC в ЗУ;

```
FLOAT5:
      SPM    1      ; PM = 1
      CLRC OVM      ; OVM = 0
      CALL   Q31    ; вызов п/программы вычисления X
FLOAT6:
      SETC    OVM   ; OVM = 1
      CALL   Q31    ; вызов п/программы вычисления X
```

Выполнить по шагам фрагменты программ до метки SXM. Результаты выполнения команд процессора (результаты вычислений) занесите в таблицу. В неё заносятся значения всех частичных сумм и последнего результата в зависимости от опций и сдвига; сравните результаты с точными расчетами. Значения частичных сумм и последнего результата в десятичном виде можно просмотреть в окне *Watch*.

3. Влияние опции режима расширения знака SXM на выполнение операций в младшем слове ACC. При SXM=0 числа считаются беззнаковыми, при SXM=1 числа считаются знаковыми в дополнительном коде.

```
* SXM=0
      CLRC    SXM
      ZAC
      SUB    #5     ; (ACC)=0-5
      ZAC
      SUB    #(-5)  ; (ACC)=0-(-5)
      LACC   #(-5)    ; загрузка в ACC числа
      ADD    #6     ; (ACC)=-5+6
      LACC   #(-5)    ; загрузка в ACC числа
      SUB    #(-6)  ; (ACC)=-5-(-6)
*  SXM=1
      SETC    SXM
      ZAC
      SUB    #5     ; (ACC)=0-5
      ZAC
      SUB    #(-5)  ; (ACC)=0-(-5)
*      LACK   #(-5)   ; загрузка в ACC числа
       LACC    #(-5)
      ADD    #6     ; (ACC)=-5+(6)
*      LACK   #(-5)   ; загрузка в ACC числа
```

```
      LACC    #(-5)
      SUB     #(-6)   ; (ACC)=-5-(-6)
HALT:
      NOP
```

Т а б л и ц а 2.1

PM	OVM	Сдвиг при записи в память	Промежуточные и последние результаты			Примечания
			Точные	ACC	Память	
0	0	0	0.72 1.14 0.74			
0	1	0				
0	0	1				
0	1	1				
1	0	0				
1	1	0				

Т а б л и ц а 2.2

SXM	Операция	Результат
0		
1		

2.3 Содержание отчета

2.3.1 Текст программы с подробными комментариями.

2.3.2 Таблицы 3.1 и 3.2, содержащие состояния регистров PM, SXM, OVM и результаты работы программы.

2.3.3 Выводы по результатам сравнения точных значений, полученных при ручном расчете и значений, полученных в результате выполнения программы.

2.4 Контрольные вопросы

2.4.1 Поясните последовательность перевода двоичного положительного числа в двоичное отрицательное число.

2.4.2 Поясните запись «короткого» числа в «длинный» регистр.

2.4.3 Переведите в двоичный код следующие числа: 3; -3; 30; -30; 100.

2.4.4 Почему после выполнения умножения мы сохраняем старшее слово аккумулятора?

2.4.5 Зачем при сохранении старшего слова аккумулятора мы применяем сдвиг влево на 1 разряд?

2.4.6 Что означает «формат Q15»?

2.4.7 Чем определяется точность представления целых двоичных чисел?

2.4.8 Чем определяется точность представления дробных двоичных чисел?

3 Лабораторная работа №3. Ознакомление с учебным стендом и его программным обеспечением

Цель работы: изучить аппаратную часть и программное обеспечение учебного стенда для изучения микроконтроллерных устройств семейства MSP430.

3.1 Краткая теоретическая справка

3.1.1 Описание лабораторного стенда УМК Texas Instruments-1

Стенд УМК Texas Instruments-1 разработан на основе экспериментального набора MSP-EXPCC430RF4, отражающего работу микроконтроллеров семейства MSP·430xx, которые имеют фон-неймановскую архитектуру и содержат 16 - битное центральное процессорное устройство (ЦПУ) типа RISC и периферийные модули.

Сочетание современного ЦПУ и отображаемых в памяти аналоговых и цифровых периферийных модулей делает данное семейство пригодным для работы в приложениях, связанных с обработкой смешанных сигналов.

Общий вид учебного стенда представлен на рисунке 3.1.

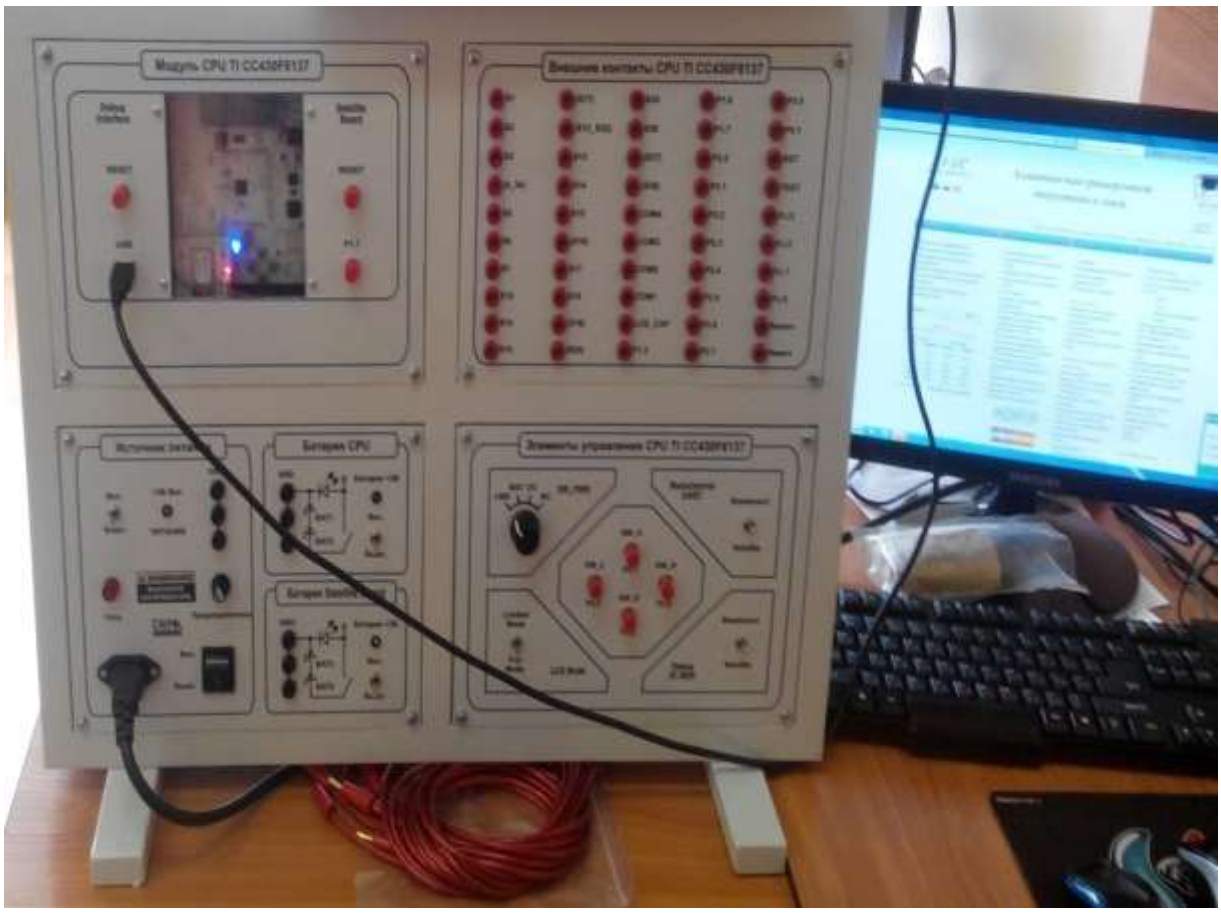

Рисунок 3.1

Как видно из рисунка 3.1 внешнее поле учебного стенда разделено на четыре блока:

- блок модуля CPU TI CC430F137;
- блок питания;
- блок элементов управления CPU TI CC430F137;
- блок внешних контактов CPU TI CC430F137.

Слева на блоке модуля расположены USB – порт и кнопка перезапуска (RESET) основной платы (CC430F6137). Посередине находится окно для возможности наблюдения за текущим состоянием экспериментального набора, справа от него - кнопка перезапуска (RESET) беспроводной платы (CC430F5137), а также продублированная кнопка без фиксации P1.7.

Подача питания на стенд может осуществляться тремя способами:

- переключатель питания (Switch Power - SW_PWR) повернуть в положение USB. Подключить стенд USB шнуром к персональному компьютеру или ноутбуку;
- переключатель питания (SW_PWR) повернуть в положение BAT. Переключить тумблеры в блоках Батарея CPU, Батарея Satellite Board в положение Вкл.;
- переключатель питания (SW_PWR) повернуть в положение DC. В блоке стенда «Источник питания» подключите шнур 220В, воткните его в розетку, щелкните нижний переключатель в положение Вкл., затем переключите тумблер в верхнем левом углу в состояние Вкл., индикатор «+3В Вкл Питание» проинформирует вас о подаче питания на стенд загоревшимся зеленым светодиодом.

Блок «Элементы управления CPU TI CC430F6137» состоит из пяти контуров:

1) Управление питанием, рассмотрен ранее.

2) Backchannel UART , программирование через UART (возможность подключения к протоколу передачи данных на экспериментальный набор возможна только в присутствии преподавателя).

Два положения: программирование базовой и беспроводной платы.

3) Кнопки 2.0 - 2.3 , в стандартной программе кнопки навигации по меню вверх (SWITCH_UP (SW_U), вниз (SWITCH_DOWN (SW_D), влево (SWITCH_LEFT (SW_L) , (SWITCH_RIGHT (SW_R) вправо.

4) Управление отображением на LCD дисплее:

- все сегменты (Full Mode), 96 сегментов;
- ограниченное число сегментов (Limited Mode), 64 сегмента.

5) Управление отладочным интерфейсом программирования через шнур USB (Debug I/F SBW). Два положения:

- программирование базовой платы;
- программирование беспроводной платы.

Экспериментальный набор предусматривает использование двух модулей:

- базовой платы MSP430f6137;

- ретрансляционной (спутниковой) платы MSP430f5137.

Для разработки алгоритмов и написания программ используется программное обеспечение IAR Embedded Workbench 5.6.

В таблице 3.1 представлены основные характеристики экспериментального набора.

Таблица 3.1

Архитектура	MSP430 / 16-бит
Тактовая частота, МГц	20
Количество линий ввода/вывода	44
Flash-память, Кб	32
ОЗУ, Кб	4
Каналов 12-разрядного АЦП	12
Частота дискретизации АЦП, тыс.отсчетов в сек.	200
Частота встроенного RC-генератора	9,4 KHz + 32,768 KHz + (0,1-100)MHz
16-битных таймеров	2

Отличительные характеристики микроконтроллеров семейства MSP430·2xx:

- сверхнизкое энергопотребление;
- оптимизировано для современных языков программирования высокого уровня;
- набор команд состоит всего из 27 инструкций; поддерживается 7 режимов адресации;
- векторная система прерываний с расширенными возможностями.
- флэш-память с возможностью внутрисхемного программирования позволяет гибко изменять программный код.

3.2 Порядок выполнения работы

3.2.1 Переключатель питания (Switch Power - SW_PWR) повернуть в положение USB. Подключить стенд USB шнуром к персональному компьютеру (ПК) или к ноутбуку.

3.2.2 Для создания нового проекта заходим в меню Project программы IAR Embedded Workbench 5.6 и жмем Create New Project. В появившемся окне создаем C project, то есть для программирования на языке СИ.

3.2.3 После нажатия на ОК компилятор сам вставит для нас файл с пустой функцией main() в проект и пропишет туда остановку WatchDog’a, как показано на рисунке 3.2.

3.2.4 В пункте меню Options for mode «test» (правая кнопка мыши на проект) в категории General Options задаем тип используемого устройства CC430F6137 для базовой платы**.**

3.2.5 Теперь идем во вкладку Debugger и выбираем там наш программатор/отладчик и настраиваем его как FET Debugger.

```
1
2   #include "io430.h"
3
4   int main( void )
5   {
6     // Stop watchdog timer to prevent time out reset
7     WDTCTL = WDTPW + WDTHOLD;
8
9     return 0;
10  }
11
```

Рисунок 3.2

3.2.6 Скопируем нижеприведенную программу, которая по нажатию кнопки P2.1(SW_U) базовой платы должна зажигать светодиоды PJ.0-PJ.3.

```
#include "cc430f6137.h"
int main( void )
{  // Stop watchdog timer to prevent time out reset
  WDTCTL = WDTPW + WDTHOLD;
    PJDIR  |= BIT3; //Установка порта PJ.3 в качестве выхода
   PJOUT &= ~BIT3; //Установка выхода порта PJ.3 в ноль
   PJDIR  |= BIT2; //Установка порта PJ.2 в качестве выхода
   PJOUT &= ~BIT2; //Установка выхода порта PJ.2 в ноль
   PJDIR  |= BIT1; //Установка порта PJ.1 в качестве выхода
   PJOUT &= ~BIT1; //Установка выхода порта PJ.1 в ноль
   PJDIR  |= BIT0; //Установка порта PJ.0 в качестве выхода
   PJOUT &= ~BIT0; //Установка выхода порта PJ.0 в ноль
   P2REN |= BIT1; // Резистивная подтяжка кнопки P2.1 на единицу или (P2IN & BIT1) == 1
   P2OUT |= BIT1; //Установка выхода порта P2.1  в единицу
   P2DIR &= ~BIT1; //Установка порта P2.1 в качестве входа
   while (1) // основной цикл программы
   {    if ((P2IN & BIT1) == 0) // Если потенциал кнопки P2.1 равен нулю (так как включили подтяжку он будет равен нулю по нажатию кнопки)
     {PJOUT |= BIT3 ;  PJOUT |= BIT2 ; PJOUT |= BIT1 ; PJOUT |= BIT0 ; } else {PJOUT &= ~BIT3; PJOUT &= ~BIT2; PJOUT &= ~BIT1; PJOUT &= ~BIT0;}  // то зажечь ДИОДЫ PJ.(0-3) // иначе потушить ДИОД PJ.(0-3)
   }}
```

3.2.7 Нажимаем кнопку START ( ), потом нажимаем кнопку GO ( ).

3.2.8 Скопируем нижеприведенную программу, которая по нажатию кнопки P1.7 должна зажигать светодиод P1.0.

```
#include "cc430f6137.h"
int main( void )
{  // Stop watchdog timer to prevent time out reset
```

```
  WDTCTL = WDTPW + WDTHOLD;
   P1DIR  |= BIT0; //Установка порта P1.0 в качестве выхода
   P1OUT &= ~BIT0; //Установка выхода порта P1.0 в ноль
   P1REN |= BIT7; // Резистивная подтяжка кнопки P1.7 на единицу или (P2IN
& BIT1) == 1
   P1DIR &= ~BIT7; //Установка порта P1.7 в качестве входа
   P1OUT |= BIT7; //Установка выхода порта P1.7  в единицу
    while (1) // основной цикл программы
     if ((P1IN & BIT7) == 0) // Если потенциал кнопки P1.7 равен нулю, так как
включили подтяжку он будет равен нулю по нажатию
     P1OUT |= BIT0 ; else P1OUT &= ~BIT0;    // то зажечь ДИОД P1.0 // иначе
потушить ДИОД P1.0
}
```

3.2.9 Повторяем пункт 3.2.7.

3.2.10 По заданию преподавателя внести коррекцию в программу для зажигания и тушения заданных светодиодов и испытать на стенде.

3.2.11 Для возврата к изначальной конфигурации базовой и беспроводной платы необходимо:

1) Открыть папку firmware, используя путь d:\ti\bin\ firmware.

2) Запустить файл MspExpCc430_FirmwareDownloader.

3) Элементы управления стенда поставить в следующие положения:

- Backchannel UART : BaseBoard;
- Debug I/F SBW : BaseBoard (Satellite).

3.2.12 В открывшемся окне командной строки набрать число 3, соответствующее MSP-EXP430F6137x - LF 433 MHz, что означает выбор нашего устройства с частотой передачи 433 МГц. После ввода числа 3 и нажатия клавиши ENTER появляется окно загрузки стандартной прошивки микроконтроллера, где можно наблюдать диалог загрузки стандартной программы в контроллер.

3.3 Содержание отчета

3.3.1 Текст программ с комментариями.

3.3.2 Ответы на контрольные вопросы.

3.4 Контрольные вопросы

3.4.1 Дайте краткую характеристику базовой платы.

3.4.2 Архитектура микроконтроллера MSP430.

3.4.3 Из каких основных частей состоит экспериментальный набор?

3.4.4 Как работать с программным обеспечением микроконтроллера?

3.4.5 Порядок загрузки начальной конфигурации.

3.4.6 В чем отличие Full Mode от Limited Mode?

3.4.7 Дать обозначение и описание команд PJDIR, PJOUT и PJREN.

4 Лабораторная работа №4. Работа с прерыванием

Цель работы: приобретение навыков организации программного режима прерываний

4.1 Краткая теоретическая справка

Механизм прерываний создан для обеспечения максимально оперативной реакции программы на определенные события.

Прерывание (interrupt) - сигнал, сообщающий процессору о наступлении какого-либо события. При этом выполнение текущей последовательности команд приостанавливается и управление передаётся обработчику прерывания, который реагирует на событие и обслуживает его, после чего возвращает управление в прерванный код.

Ниже приведены команды, которые имеют имеют следующее назначение:

1) PxIE разрешает прерывания для пинов порта Px.

2) PxIES определяет, по какому уровню сигнала будет происходить прерывание.

3) PxIFG определяет флаг прерывания.

4) bis_SR_register(GIE) устанавливает флаг глобального разрешения прерываний (Global Interrupt Enable) в status register.

5) #pragma vector — директива, которая определяет, что нижеследующая функция является обработчиком указанного прерывания.

4.2 Порядок выполнения работы

4.2.1 Переключатель питания (Switch Power - SW_PWR) повернуть в положение USB. Подключить стенд USB шнуром к персональному компьютеру (ПК) или к ноутбуку.

4.2.2 Для создания нового проекта заходим в меню Project программы IAR Embedded Workbench 5.6 и жмем Create New Project. В появившемся окне создаем C project, то есть для программирования на языке СИ.

4.2.3 После нажатия на ОК компилятор сам вставит для нас файл с пустой функцией main() в проект и пропишет туда остановку WatchDog’a, как показано на рисунке 3.2.

4.2.4 В пункте меню Options for mode «test» в категории General Options задаем тип используемого устройства CC430F6137 для базовой платы.

4.2.5 Теперь идем во вкладку Debugger и выбираем там наш программатор/отладчик и настраиваем его как FET Debugger.

4.2.6 Скопируем программу, написанную на языке СИ и представленную ниже, где по нажатию кнопки P2.2 происходит прерывание, которое меняет состояние двух светодиодов PJ.0 и PJ.1.

```
#include "cc430f6137.h"
void main()
{  WDTCTL = WDTPW + WDTHOLD;
P2DIR &= ~BIT2;
P2REN |= BIT2;
P2IE |= BIT2; // Разрешение прерываний на P2.2
P2IES |= BIT2; // Прерывание происходит по 1/0 (отпусканию/нажатию)
P2IFG &= ~BIT2; // Очистка флага прерываний для P2.2
PJDIR |= BIT0 | BIT1;
PJOUT &= ~BIT0;
PJOUT &= ~BIT1;
__bis_SR_register(GIE); // Установка флага глобального разрешения прерываний
while(1);
}
#pragma vector=PORT2_VECTOR
interrupt void P2INT() // Обработчик прерывания
{  P2IE &= ~BIT2; // Запрет прерываний на P2.2
PJOUT ^= BIT0; // PJ.0 меняет своё состояние
PJOUT ^= BIT1; // PJ.1 меняет своё состояние
for(volatile unsigned int i = 100; i != 0; i--); // Задержка
{P2IE |= BIT2; // Разрешение прерываний на P2.2
P2IFG &= ~BIT2; // Очистка флага прерываний для P2.2
}
```

4.2.7 По заданию преподавателя внести коррекцию в программу для организации прерывания, которое по нажатию заданной кнопки меняет состояние заданных светодиодов. Программу испытать на стенде.

4.3 Содержание отчета

4.1.1 Текст программ с комментариями.
4.1.2 Ответы на контрольные вопросы.

4.4 Контрольные вопросы

4.4.1 Объясните механизм операции «прерывание».
4.4.2 Определите назначение команд прерывания.
4.4.3 Приведите алгоритм создания C project.
4.4.4 Укажите в программе, где устанавливается задержка, чему она равна?

5 Лабораторная работа №5. Работа с LCD дисплеем

Цель работы: получение навыков вывода целочисленных и буквенных значений на LCD дисплей базовой платы.

5.1 Краткая теоретическая справка

LCD дисплей представляет собой совокупность сегментов, каждый из которых отображает определенный элемент. Каждый сегмент представляет собой бит информации, расположенный по определенному адресу. Всего на экране размещено 112 сегментов, которые сгруппированы в 14 байтах. Адресация байтов произведена в шестнадцатеричном формате в диапазоне 0A20 - 0A2D. Как видно из рисунка 5.1, для отображения элемента на экране необходимо обратиться к определенному биту соответствующего байта.

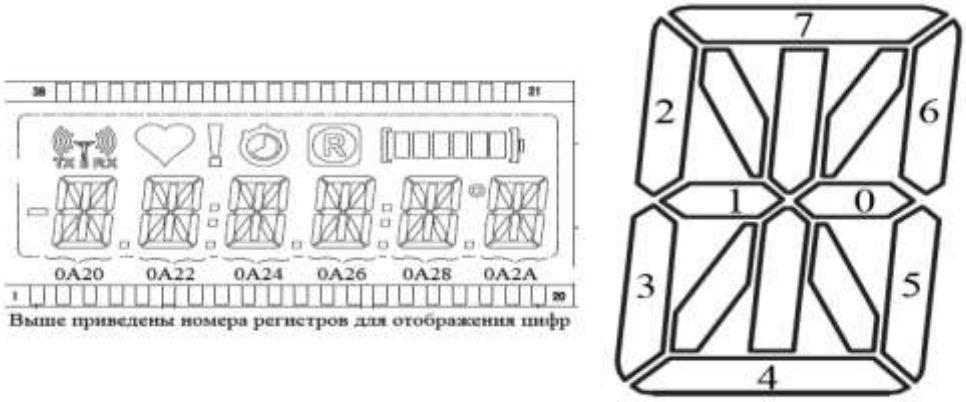

Рисунок 5.1

Для отображения чисел и букв используются 6 элементов, каждый из которых представляет собой совокупность 14 сегментов. Адреса байтов для обращения к сегментам каждого элемента приведены на рисунке 5.1. Например, согласно данному рисунку 5.1, для отображения в первом элементе необходимо обратиться к адресу 0A20 и для отображения цифры 0 необходимо установить все биты, кроме нулевого и первого. Для работы LCD дисплея базовой платы необходимо предварительно ее запрограммировать посредством программного обеспечения IAR Embedded Workbench 5.6. Язык программирования «С».

5.2 Порядок выполнения работы

5.2.1 Скопируем программу, представленную ниже и предназначенную для отображения цифр в диапазоне от 0 до 9 с задержкой.

```
#include  "cc430f6137.h"
// Указатель для обращения к памяти контроллера
unsigned char* segmentPointer;
```

```
// Простая функция задержки
void simpleDelay(double delayValue)
{ for (double i = 0; i < delayValue; i++);}
// А это функция для того, чтобы в первой позиции дисплея
// высветилась цифра, переданная в функцию в качестве
// параметра
void showNum(unsigned char num)
{    segmentPointer = ((unsigned char*)0x0A20);
 if (num == 0)
 {*segmentPointer = (unsigned char)(BIT2 + BIT3 + BIT4 + BIT5 + BIT6 +
BIT7);   } // установка сегментов цифры 0
 if (num == 1)
 {*segmentPointer = (unsigned char)(BIT5 + BIT6) } // установка
сегментов цифры 1
 if (num == 2)
 {*segmentPointer = (unsigned char)(BIT0 + BIT1 + BIT3 + BIT4 + BIT6+
BIT7);} // установка сегментов цифры 2
 if (num == 3)
 {*segmentPointer = (unsigned char)(BIT0 + BIT1 + BIT4 + BIT5 + BIT6 +
BIT7);} // установка сегментов цифры 3
 if (num == 4)
 {          *segmentPointer = (unsigned char)(BIT0 + BIT1 + BIT2 + BIT5 +
BIT6);} // установка сегментов цифры 4
 if (num == 5)
 {*segmentPointer = (unsigned char)(BIT0 + BIT1 + BIT2 + BIT4 + BIT5 +
BIT7); } // установка сегментов цифры 5
 if (num == 6)
 {*segmentPointer = (unsigned char)(BIT0 + BIT1 + BIT2  + BIT3 + BIT4 +
BIT5 + BIT7);} // установка сегментов цифры 6
 if (num == 7)
 {*segmentPointer = (unsigned char)(BIT5 + BIT6 + BIT7);} // установка
сегментов цифры 7
 if (num == 8)
 {*segmentPointer = (unsigned char)(BIT0 + BIT1 + BIT2 + BIT3 + BIT4 +
BIT5 + BIT6 + BIT7); } // установка сегментов цифры 8
 if (num == 9)
 {*segmentPointer = (unsigned char)(BIT0 + BIT1 + BIT2 + BIT4 + BIT5 +
BIT6 + BIT7);   }} // установка сегментов цифры 9
 int main( void )
 {    // Сброс watchdog'a
 WDTCTL = WDTPW + WDTHOLD;
  // Очистка памяти дисплея
 LCDBMEMCTL |= LCDCLRBM + LCDCLRM;
 // Настройка дисплея
```

```
// LCD_FREQ = ACLK/16/8 = 256Hz
// Frame frequency = 256Hz/4 = 64Hz, LCD mux 4, LCD on
LCDBCTL0 = (LCDDIV0 + LCDDIV1 + LCDDIV2 + LCDDIV3) | (LCDPRE0 + LCDPRE1) | LCD4MUX | LCDON;
// Настраиваем нужные пины для работы с дисплеем
P5SEL |= (BIT5 | BIT6 | BIT7);
P5DIR |= (BIT5 | BIT6 | BIT7);
// Нужные сегменты дисплея
LCDBPCTL0 = 0xFFFF;        // Выбор сегментов LCD S0-S15
LCDBPCTL1 = 0x00FF;        // Выбор сегментов LCD S16-S22*/
while(1)
{// Поочередно зажигаем цифры на дисплее, затем задержка, и потом
// следующая цифра – и так по кругу
showNum(0);
simpleDelay(64000);
showNum(1);
simpleDelay(64000);
showNum(2);
simpleDelay(64000);
showNum(3);
simpleDelay(64000);
showNum(4);
simpleDelay(64000);
showNum(5);
simpleDelay(64000);
showNum(6);
simpleDelay(64000);
showNum(7);
simpleDelay(64000);
showNum(8);
simpleDelay(64000);
showNum(9);
simpleDelay(64000);  }}
```

5.2.2 Скопируем программу вывода с задержкой на LCD дисплей букв латинского алфавита I, K, N, R, T, W, X, Y, Z с задержкой и представленную ниже.

```
#include  "cc430f6137.h"
// Указатель для обращения к памяти контроллера
unsigned char* segmentPointer;
unsigned char* segmentPoint;
// Простая функция задержки
void simpleDelay(double delayValue)
{for (double i = 0; i < delayValue; i++);}
```

```
// А это функция для того, чтобы в первой позиции дисплея
// высветилась цифра, переданная в функцию в качестве
// параметра
void showabc(unsigned char num)
{    segmentPointer = ((unsigned char*)0x0A29);
 segmentPoint = ((unsigned char*)0x0A28);
 if (num == 0)
 {  *segmentPointer = (unsigned char)(BIT4 + BIT6);
*segmentPoint = (unsigned char)(BIT4 + BIT7);}
 if (num == 1)
 {  *segmentPointer = (unsigned char)(BIT5 + BIT1);
*segmentPoint = (unsigned char)(BIT1 + BIT2 + BIT3); }
 if (num == 2)
 { *segmentPointer = (unsigned char)(BIT7 + BIT5);
*segmentPoint = (unsigned char)(BIT2 + BIT3 + BIT5 + BIT6);}
 if (num == 3)
 {  *segmentPointer = (unsigned char)(BIT1 + BIT7);
  *segmentPoint = (unsigned char)(BIT2 + BIT3 + BIT5 + BIT6);   }
 if (num == 4)
 {   *segmentPointer = (unsigned char)(BIT1);
*segmentPoint = (unsigned char)(BIT0+BIT1+BIT2+BIT3+ BIT7 + BIT6);}
 if (num == 5)
 {  *segmentPointer = (unsigned char)(BIT6 + BIT4);
*segmentPoint = (unsigned char)(BIT7); }
 if (num == 6)
 {  *segmentPointer = (unsigned char)(BIT1 + BIT3 + BIT6);
*segmentPoint = (unsigned char)(BIT2 + BIT3 + BIT5 + BIT6);  }
 if (num == 7)
 {  *segmentPointer = (unsigned char)(BIT1 + BIT7 + BIT3 + BIT5);
*segmentPoint = (unsigned char)(BITA);  }
 if (num == 8)
 { *segmentPointer = (unsigned char)(BIT7);
*segmentPoint = (unsigned char)(BIT0 + BIT4 + BIT5 + BIT6);}
 if (num == 9)
 {  *segmentPointer = (unsigned char)(BIT3 + BIT5);
*segmentPoint = (unsigned char)(BIT0 + BIT1 + BIT4 + BIT7); }}
 int main( void )
 {    // Сброс watchdog'a
  WDTCTL = WDTPW + WDTHOLD;
   // Очистка памяти дисплея
   LCDBMEMCTL |= LCDCLRBM + LCDCLRM;
   // Настройка дисплея
   // LCD_FREQ = ACLK/16/8 = 256Hz
   // Frame frequency = 256Hz/4 = 64Hz, LCD mux 4, LCD on
```

```
LCDBCTL0 = (LCDDIV0 + LCDDIV1 + LCDDIV2 + LCDDIV3) | (LCDPRE0 + LCDPRE1) | LCD4MUX | LCDON;
// Настраиваем нужные пины для работы с дисплеем
P5SEL |= (BIT5 | BIT6 | BIT7);
P5DIR |= (BIT5 | BIT6 | BIT7);
// Нужные сегменты дисплея
LCDBPCTL0 = 0xFFFF;        // Выбор сегментов LCD S0-S15
LCDBPCTL1 = 0x00FF;        // Выбор сегментов LCD S16-S22*/
while(1)
{          // Поочередно зажигаем буквы на дисплее, затем задержка, и потом следующая буква – и так по кругу
showabc(0);
simpleDelay(10000);
showabc(1);
simpleDelay(10000);
showabc(2);
simpleDelay(10000);
showabc(3);
simpleDelay(10000);
showabc(4);
simpleDelay(10000);
showabc(5);
simpleDelay(10000);
showabc(6);
simpleDelay(10000);
showabc(7);
simpleDelay(10000);
showabc(8);
simpleDelay(10000);
showabc(9);
simpleDelay(10000);
}
```

5.3 Содержание отчета

5.3.1 Текст программы с комментариями по заданию преподавателя.
5.3.2 Ответы на контрольные вопросы.

5.4 Контрольные вопросы

5.4.1 Краткая характеристика LCD – дисплея.
5.4.2 Какие биты нужно установить для высвечивания цифры 7?
5.4.3 Какие биты нужно установить для высвечивания буквы R?
5.4.4 Составьте листинг программы для высвечивания на дисплее с задержкой 3 секунды следующих символов: + , - , = .

6 Лабораторная работа №6. Передача данных беспроводным путем

Цель работы: получение навыков организации беспроводной передачи данных.

6.1 Краткая теоретическая справка

В данной лабораторной работе осуществляется организация беспроводной передачи данных между двумя базовыми платами MSP-EXP430F6137R4. Для настройки параметров удобно использовать вспомогательное программное обеспечение SmartRFStudio от TI.

SmartRFStudio — полезный инструмент, который помогает проектировщикам беспроводных систем оценить различные радиочастотные модули (РЧ) на ранней стадии разработки.

Инструмент представляет собой приложение для персонального компьютера (ПК), работающее с беспроводными оценочными наборами Texas Instruments на базе РЧ-микросхем серий CCxxxx, в том числе CC430.

Программа запускается под ОС Windows и через USB или параллельный порт взаимодействует с отладочной платой, которая, в свою очередь, подключается по РЧ-каналу к оценочным платам с установленными РЧ-модулями.

Удобный пользовательский интерфейс дает доступ к регистрам настройки РЧ-модуля для быстрого тестирования и настройки параметров РЧ-канала.

Самый главный настраиваемый параметр – это скорость, на разной скорости у приёмника разная чувствительность, а значит, и разная дальность приёма при равных прочих условиях. От скорости будет зависеть и потребление энергии источника питания, при передаче ток от скорости зависит мало, но время передачи зависит сильно.

Если нужно добиться максимального энергосбережения, то лучше настраивать на 250кбит или 500кбит. На используемой плате установлена беспроводная антенна CC1101. Данная антенна имеет возможность осуществлять две модуляции:

- для высокоскоростных режимов лучше(500, 250 кБит) подходит MSK;
- для низкоскоростных 2-GFSK.

6.2 Порядок выполнения работы

6.2.1 Включаем лабораторный стенд и подключаем его к ПК посредством USB.

6.2.2 Запускаем SmartRFStudio при помощи ярлыка на рабочем столе. Открывается окно выбора оборудования.

6.2.3 Используем по умолчанию вкладку Sub 1GHzISMband. Нажатием кнопки «Find device» осуществляем поиск доступного оборудования и выбираем MSP-EXP430F6137R4.

6.2.4 Двойным щелчком по данному устройству в списке открываем главное окно и заходим в меню View, устанавливаем RegisterView и RF Parameters.

6.2.4 В данном окне доступны 2 вкладки. По умолчанию, используется вкладка Sub 1GHzISMband, продолжаем работу с данной вкладкой. Нажатием кнопки «Find device» осуществляем поиск доступного оборудования.

После завершения поиска Smart RF Studio предложит список обнаруженных подключенных устройств, в котором будет только одно доступное - MSP-EXP430F6137R4.

6.2.5 Двойным щелчком по данному устройству в списке открываем главное окно SmartRFStudio. Заходим в меню View, убираем отметку (если есть) с EasyMode, устанавливаем RegisterView и RF Parameters.

В главном окне SmartRFStudio можно выделить четыре области:

1) Набор предустановленных настроек.

2) Тонкая настройка параметров.

3) Работа с демонстрационным оборудованием от TI.

4) Список регистров.

6.2.6 За основу удобно брать одну из заготовок из списка области 1. В нашем случае выбираем первую строку. Далее изменяем значения параметров согласно функциональным возможностям платы, а именно: нас интересует параметр «Base frequency» - базовая частота передачи данных. Устанавливаем ее равной 432,999939 MHz.

6.2.7 После выполнения всех настроек необходимо сохранить их. Для этого осуществляем следующие действия: меню File => SaveCfg и сохраняем конфигурацию. Потом всегда можно открыть её через OpenCfg.

6.2.8 Перейдем на область 4. Выбираем зеленые регистры, изменяющие свое значение в ходе выполнения передачи данных.

6.2.9 Переходим в область 3. В данной области доступны 5 вкладок, нас интересуют две вкладки:

1) Packet TX – вкладка настройки передатчика информации.

2) Packet RX – вкладка настройки приемника информации.

Вкладка «Packet TX» имеется возможность осуществлять передачу произвольно сгенерированной строки кнопка «Random», либо текстовой информации кнопка «Text», либо в шестнадцатеричном формате кнопка «Hex». В данной работе мы будем использовать только первые две, а именно в данном примере только передачу текстовой информации.

6.2.10 Другим важным настроечным параметром вкладки «PacketTX» является количество передаваемых пакетов информации. Данный параметр задается произвольно, однако при установке значения необходимо учитывать

тот факт, что чем больше число передаваемых пакетов, тем дольше будет осуществляться передача.

6.2.11 Для приема текстовой информации в выпадающем списке «Viewing format» необходимо выбрать «Text».

6.2.12 Другой важный параметр настройки вкладки «Packet RX» является параметр «Expected packet count». Данный параметр позволяет подсчитать количество потерянных пакетов данных.

На этом настройка завершена.

6.2.13 Для начала передачи данных необходимо сначала на ПК приемника информации во вкладке «Packet RX» нажать кнопку «Start» и только после этого на ПК передатчика информации во вкладке «Packet TX» нажать аналогичную кнопку «Start».

Передача завершится после завершения передачи последнего пакета данных либо по запросу пользователя нажатием кнопки «Stop».

6.3 Содержание отчета

6.3.1 Текст программы с комментариями по заданию преподавателя.

6.3.2 Ответы на контрольные вопросы.

6.4 Контрольные вопросы

6.4.1 Что будет, если во вкладке Packet RX нажать кнопку START раньше, чем во вкладке TX?

6.4.2 Какая частота передачи между платами?

6.4.3 Что означает «Hex», «Random»?

6.4.4 На какое расстояние может передаваться информация между платами?

6.4.5 Как вернуть базовые настройки платы MSP-EXP430F6137R4?

Список литературы

1 Сперанский В.С. Сигнальные микропроцессоры и их применение в системах телекоммуникаций и электроники. Учебное пособие для вузов. – М.: Горячая линия – Телеком, 2008. – 168 с.

2 Семейство микроконтроллеров MSP430x2xx. Архитектура, программирование, разработка приложений. - М.: изд. дом «Додэка-XXI», 2010. – 452 с.

3. Лэй Э. Цифровая обработка сигналов для инженеров и технических специалистов: Практическое руководство. – М.: ООО «Группа ИДТ», 2007. – 336 с.

4 Айфичер Эммануил, Джервис Барри. Цифровая обработка сигналов: практический подход. - М.: Издательский дом «Вильямс», 2004.- 992 с.

5 Бойко В.И., Гуржий А.Н. и др. Схемотехника электронных систем. Микропроцессоры и микроконтроллеры. – СПб.: БХВ – Петербург, 2004. – 464 с.

6 Петрищенко С.Н., Сигнальные процессоры. Конспект лекций для бакалавров специальности 5В071900 – Радиотехника, электроника и телекоммуникации. – Алматы, 2012. – 33 с.

Содержание

1 Лабораторная работа №1 3
2 Лабораторная работа №2 7
3 Лабораторная работа №3 16
4 Лабораторная работа №4 21
5 Лабораторная работа №5 23
6 Лабораторная работа №6 28
Список литературы 31

Сводный план 2015 г., поз.166

Петрищенко Святослав Николаевич
Нурланулы Алмас
Бакирова Нагима Сапаралиевна

ЦИФРОВЫЕ ПРОЦЕССОРЫ ОБРАБОТКИ СИГНАЛОВ

Методические указания по выполнению лабораторных работ для студентов специальности 5В071600 - Приборостроение

Редактор Л.Сластихина
Специалист по стандартизации Н.К.Молдабекова

Подписано в печать _____ Формат 60х84 1/16
Тираж 100 экз. Бумага типографская №1
Объем 2,0 уч.-из.л. Заказ ___ Цена 1000 тг.

Копировально-множительное бюро
некоммерческого акционерного общества
«Алматинский университет энергетики и связи»
050013, Алматы, Байтурсынова, 126