

Федеральное государственное бюджетное образовательное учреждение  
высшего образования



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

---

ФАКУЛЬТЕТ – Информатика, искусственный интеллект и системы  
управления

КАФЕДРА – Информационные системы и телекоммуникации

Отчёт по лабораторной работе № 2  
«Исследование процессов обработки сигналов»  
по дисциплине «Микропроцессорные устройства обработки сигналов»

Задание № 20

Студент группы ИУ3-21М

С.С. Щесняк

Преподаватель кафедры ИУ3

А.И. Германчук

Москва, 2023

## Содержание

1 Цель работы .....	2
2 Описание задания .....	2
3 Результаты выполнения задания .....	3
4 Вывод .....	5
Приложение А .....	6
Приложение Б .....	15

## 1 Цель работы

Целью данной лабораторной 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 ее выполнении используется симулятор микропроцессора TMS320C5515 или оценочная плата TMS320C5515 DSP Evaluation Module (TMDXEVM5515).

## 2 Описание задания

Лабораторная работа заключается в разработке функции адаптивной среднеквадратичной фильтрации с задержкой на языке программирования C, и ее сравнение по эффективности с аналогичной функцией из состава стандартной библиотеки для обработки сигналов.

Выполнение лабораторной работы состоит из отладки и профилирования разработанной программы. В ее рамках требуется реализовать вышеупомянутый алгоритм обработки сигналов двумя средствами: с помощью тестовой функции, написанной на языке C, и функции из состава стандартной библиотеки обработки сигналов.

Профилирование программы заключается в измерении числа циклов микропроцессора, затраченных на выполнение обработки одних и тех же тестовых данных, расположенных в директории соответствующей функции стандартной библиотеки для обработки сигналов.

### 3 Результаты выполнения задания

Разработанная функция `dlms` выполняет адаптивную среднеквадратическую фильтрацию с задержкой. Выходной сигнал цифрового нерекурсивного фильтра определяется выражением

$$r[i] = \sum_{k=0}^{nh-1} h[k] * x[i - k], 0 \leq i \leq nx - 1. \quad (1)$$

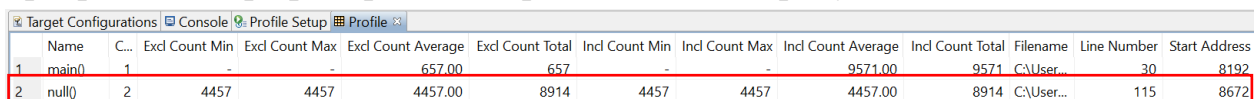
При вычислении нового отсчета данного сигнала  $r[i]$  по завершению каждого шага  $k$  производится коррекция текущего коэффициента  $h_k[i]$  в соответствии с формулой

$$h_k[i + 1] = h_k[i] + step * e[i - 1] * x[i - k - 1], \quad (2)$$

где  $e[i - 1] = des[i - 2] - r[i - 2]$  – сигнал ошибки.

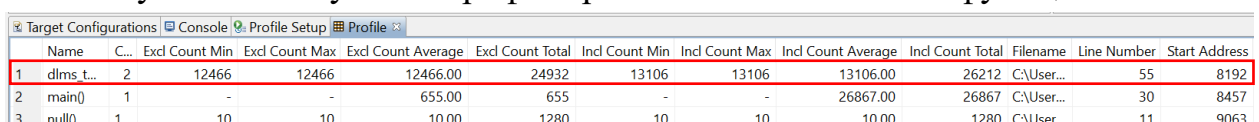
В соответствии с требованиями задания, представленный алгоритм обработки сигналов был реализован двумя средствами: с помощью тестовой функции `dlms_test`, написанной на языке C, и функции `dlms` из состава стандартной библиотеки обработки сигналов (см. приложение А). Полный текст этой программы приведен в приложении Б.

В рамках данной лабораторной работы также требуется провести тестирование и профилирование разработанной программы в интегрированной среде проектирования Code Composer Studio с помощью симулятора микропроцессора TMS320C5515. Результаты работы программы и профилирования представлены на рисунках ниже. Результаты работы программы и профилирования представлены на рисунках ниже.



Name	C...	Excl Count Min	Excl Count Max	Excl Count Average	Excl Count Total	Incl Count Min	Incl Count Max	Incl Count Average	Incl Count Total	Filename	Line Number	Start Address
1 main()	1	-	-	657.00	657	-	-	9571.00	9571	C:\User...	30	8192
2 null()	2	4457	4457	4457.00	8914	4457	4457	4457.00	8914	C:\User...	115	8672

Рисунок 1 – Результат профилирования библиотечной функции `dlms`



Name	C...	Excl Count Min	Excl Count Max	Excl Count Average	Excl Count Total	Incl Count Min	Incl Count Max	Incl Count Average	Incl Count Total	Filename	Line Number	Start Address
1 dlms t...	2	12466	12466	12466.00	24932	13106	13106	13106.00	26212	C:\User...	55	8192
2 main()	1	-	-	655.00	655	-	-	26867.00	26867	C:\User...	30	8457
3 null()	1...	10	10	10.00	1280	10	10	10.00	1280	C:\User...	11	9063

Рисунок 2 – Результат профилирования разработанной функции `dlms_test`

Address	Value	Value	Value	Value	Value	Value	Value	Value	Value	Value	Value
0x005124	h										
0x005124	-6	-72	324	-125	261	-196	126	-56	194	228	168
0x00512F	138	-212	203	-314	215	-387	189	-424	223	-30	77
0x00513A	115	3	66	-219	400	-8	283	75	-48	-121	
0x005144	hn										
0x005144	-7	-68	319	-127	258	-194	127	-52	193	230	165
0x00514F	140	-214	203	-315	213	-386	189	-424	223	-25	73
0x00515A	115	3	68	-218	405	-8	287	77	-53	-118	
0x005164	r										
0x005164	0	0	0	-2	13	-11	1	-1	-13	37	0
0x00516F	6	-31	-4	-1	-26	112	-164	120	-224	186	-236
0x00517A	205	-223	-16	-17	-65	416	-152	508	-509	571	-634
0x005185	439	-442	165	-312	-198	29	-249	341	-383	372	-282
0x005190	889	-240	1050	-287	528	-529	499	-1	-100	181	-524
0x00519B	169	-941	731	-787	373	-478	183	75	242		
0x0051A4	rn										
0x0051A4	0	0	-1	-3	13	-14	2	-3	-13	37	0
0x0051AF	7	-34	-2	-4	-24	110	-161	115	-225	185	-235
0x0051BA	207	-225	-14	-23	-58	416	-147	508	-512	565	-636
0x0051C5	433	-444	169	-321	-202	26	-248	331	-387	372	-274
0x0051D0	886	-230	1044	-277	536	-530	507	-3	-97	161	-515
0x0051DB	169	-947	730	-801	385	-481	169	79	248		

Рисунок 3 – Результат работы программы

Тестирование программы показало, что созданная функция `dlms` работает корректно. Небольшие отличия между ее вектором выходных данных `rn` и вектором выходных данных `r` библиотечной функции, а также между векторами коэффициентов фильтров `hn` и `h` соответственно объясняются различием подходов к округлению промежуточных результатов вычислений.

Профилирование программы показало, что созданная функция `dlms` в среднем работает в 2,94 раза медленнее, чем ее аналог. Так, среднее число циклов выполнения функции `dlms_test`, включая число циклов, необходимых для выполнения вызываемых в ее теле других функций, составляет 13106, тогда как для библиотечной функции оно составляет 4457.

## 4 Вывод

В рамках данной лабораторной работы был изучен и реализован на языке программирования C цифровой нерекурсивный фильтр с использованием алгоритма адаптивной среднеквадратической фильтрации с задержкой. Для тестирования и профилирования разработанной программы были использованы возможности интегрированной среды проектирования Code Composer Studio компании Texas Instruments. Анализ результатов выполнения данной программы показал, что созданная функция работает корректно и в 2,94 раза медленнее, чем библиотечная.

## Приложение А

### Библиотечная функция dlms

```
*****  
;  
; Версия 3.00.00  
  
*****  
;  
; Функция: dlms  
  
; Процессор: C55xx  
  
; Описание: цифровой нерекурсивный фильтр с использованием алгоритма  
адаптивной  
  
; среднеквадратической фильтрации с задержкой  
  
;  
  
; Вызов: short oflag = dlms(DATA *h, DATA *x, DATA *r, DATA *des,  
;  
DATA *dbuffer, DATA step, ushort nh, ushort nx)  
;  
  
; ...где  
  
; h[nh] Указатель на вектор коэффициентов фильтра размера nh  
;  
- Элементы вектора хранятся в памяти в обратном порядке:  
h(n-1), ... h(0)  
  
; x[nx] Указатель на вектор входных данных размером nx  
  
; r[nx] Указатель на вектор выходных данных.  
  
; - Разрешается использование одного и того же указателя для  
задания векторов x и r  
  
; des[nx] Указатель на вектор ожидаемых выходных данных
```

```

;    dbuffer[nh+2]    Указатель на структуру буфера задержки

;                      - первым элементом структуры является 16-разрядное
целочисленное значение индекса

;                      самого "старого" элемента кольцевого буфера

;                      - вторым элементом структуры является вышеупомянутый
кольцевой буфер длины nh+1,

;                      предназначенный для хранения отсчетов входного сигнала
при адаптивной фильтрации

;    step    Размер шага =  $2 \cdot \mu$ , который определяет скорость
сходимости алгоритма

;    nh      Число коэффициентов цифрового нерекурсивного фильтра.
Порядок фильтра = nh-1.

;    nx      Число отсчетов входного сигнала.

;    oflag   Флаг переполнения

;           - Если oflag = 1, то было обнаружено переполнение 32 бит

;           - Если oflag = 0, то не было обнаружено переполнение 32 бит

;

; Производительность:

; Циклы:

;    nh = 21: nx = 128: 6056

;    nh = 12: nx = 100: 2940

;    nh = 32: nx = 64: 4456

;    nh = 21: nx = 40: 1920

;    nh = 3: nx = 32: 392

```



```

;   nh = 12: nx = 20:   620

;   nh =  3: nx =  3:   74

;   Размер кода (байт): 353

;

;*****

    .cpl_on ; Режим компилятора

    .arms_off ; Сигнальный режим адресации

    .noremark 5684 ; BRC1 косвенно не изменяется

; Формат стека

;*****

    .asg    0, save_T3    ; Выделение памяти под регистр T3

    .asg    1, save_T2    ; Выделение памяти под регистр T2

    .asg    2, save_AR5    ; Выделение памяти под регистр AR5

    .asg    3, ret_addr    ; Выделение памяти под адрес возврата

    .asg    4, arg_nx      ; Выделение памяти под число отсчетов входного
сигнала

; Макроопределения используемых регистров

;   Вызываемая функция может использовать регистры T0, T1, (X)AR0,
(X)AR1, (X)AR2, (X)AR3, (X)AR4,

;   AC0, AC1, AC2, AC3 без сохранения

```

; Аргументы функции \*x, \*h, \*r, \*des и \*dbuffer размещены в регистрах XAR0-XAR4 соответственно

; Аргументы функции STEP и NH размещены в регистрах T0 и T1 соответственно

;\*\*\*\*\*

.asg AR0, ar\_input ; Аргумент функции \*x

.asg AR1, ar\_coef ; Аргумент функции \*h

.asg AR2, ar\_output ; Аргумент функции \*r

.asg AR3, ar\_des ; Аргумент функции \*des

.asg AR4, ar\_dbuffer ; Аргумент функции \*dbuffer

.asg AR5, ar\_data ; Локальная переменная \*ar\_data - указатель на второй элемент структуры

; буфера задержки

.asg T0, T\_step ; Аргумент функции STEP

.asg T1, T\_nh ; Аргумент функции NH

;\*\*\*\*\*

\*\*\*\*\*

; Тело функции

;\*\*\*\*\*

.def \_dlms

.text

\_dlms:

; Сохранение модифицируемых регистров на стеке и конфигурация регистров  
статуса

;-----

PSH T3, T2 ; Сохранение регистров T3 и T2 на стеке

|| BCLR ARMS ; Интенсивная обработка сигнала

PSHBOTH XAR5 ; Сохранение регистра XAR5 на стеке  
(используется для хранения первого

; элемента структуры буфера задержки)

|| BSET FRCT ; Дробный режим

;

; Конфигурация режимов адресации и переполнения

;-----

; Status registers

MOV XAR4, XAR5 ; Сохранение адреса первого элемента  
структуры буфера задержки в регистре XAR5

BCLR ACOV1 ; Сброс флага переполнения oflag

BSET AR1LC ; Флаг циклической адресации для AR1 (вектора  
коэффициентов фильтра)

BSET AR5LC ; Флаг циклической адресации для AR5 (кольцевого  
буфера задержки)

;

; Конфигурация передаваемых параметров

;-----

MOV \*ar\_dbuffer+, ar\_data ; Сохранение первого элемента структуры  
буфера задержки в регистре XAR5

MOV AR4, mmap(BSA45) ; Сохранение базового адреса кольцевого  
буфера задержки в регистре BSA45

MOV AR1, mmap(BSA01) ; Сохранение базового адреса вектора  
коэффициентов фильтра в регистре BSA01

MOV #0, ar\_coef ; Обнуление первого элемента данного вектора

ADD #-1, \*SP(arg\_nx) ; Определение сторожевого условия внешнего  
цикла ( $i \leq nx-1$ )

MOV \*SP(arg\_nx), BRC0 ; Конфигурация счетчика повторения  
внешнего цикла

MOV T\_nh, mmap(BK03) ; Настройка вектора коэффициентов  
фильтра

AADD #1, T\_nh ; Определение длины кольцевого буфера задержки

MOV T\_nh, mmap(BK47) ; Настройка кольцевого буфера задержки

ASUB #3, T\_nh ; Определение сторожевого условия внутреннего  
цикла ( $j \leq nh-2$ )

MOV T\_nh, BRC1 ; Конфигурация счетчика повторения  
внутреннего цикла

MOV #0, AC3 ; Обнуление значения сигнала ошибки

|| RPTBLOCAL OuterLoopEnd-1 ; Начало внешнего цикла

MOV \*ar\_input+, \*ar\_data+ ; Запись отсчета входного сигнала в  
буфер задержки

; (перезапись самого "старого" его  
элемента)

MPYM \*ar\_data+, AC3, AC0 ; Вычисление инкремента элемента  
вектора коэффициентов фильтра

|| MOV #0, AC1 ; Обнуление промежуточного значения  
отсчета выходного сигнала

LMS \*ar\_coef, \*ar\_data, AC0, AC1 ; Вычисление нового значения  
коэффициента фильтра и сохранение его в регистре AC0

; Вычисление промежуточного значения отсчета  
выходного сигнала и сохранение его в регистре AC1

|| RPTBLOCAL InnerLoopEnd-1 ; Начало внутреннего цикла

MOV HI(AC0), \*ar\_coef+ ; Сохранение нового значения  
коэффициента фильтра в памяти

|| MPYM \*ar\_data+, AC3, AC0 ; Вычисление инкремента  
следующего элемента вектора коэффициентов фильтра

LMS \*ar\_coef, \*ar\_data, AC0, AC1 ; Вычисление нового значения  
коэффициента фильтра и сохранение его в регистре AC0

InnerLoopEnd:

; Вычисление значения отсчета выходного  
сигнала и сохранение его в регистре AC1

MOV HI(AC0), \*ar\_coef+ ; Сохранение нового значения  
коэффициента фильтра в памяти

|| MOV rnd(HI(AC1)), \*ar\_output+ ; Сохранение значения отсчета  
выходного сигнала в памяти

SUB AC1, \*ar\_des+ << #16, AC2 ; Вычисление значения сигнала  
ошибки

|| AMAR \*ar\_data+ ; Вычисление адреса самого "старого"  
элемента буфера задержки и сохранение его в регистре AR5

MPYR T\_step, AC2, AC3 ; Сохранение значения сигнала ошибки  
в регистре AC3

OuterLoopEnd:

MOV ar\_data, \*-ar\_dbuffer ; Сохранение адреса самого "старого"  
элемента буфера задержки в памяти

; Определение значения флага переполнения

; -----

|| MOV #0, T0 ; Сброс флага переполнения

XCCPART overflow(AC1)

|| MOV #1, T0 ; Установка флага переполнения в единицу, если  
было обнаружено переполнение 32 бит

;

; Восстановления сохраненных на стеке регистров

; Возврат к первоначальной конфигурации регистров статуса

; Возврат в вызывающую функцию

;-----

BCLR AR1LC ; Флаг прямой адресации для AR1

POPBOTH XAR5 ; Восстановление регистра XAR5

|| BCLR AR5LC ; Флаг прямой адресации для AR5

POP T3, T2 ; Восстановление регистров T3, T2

|| BSET ARMS ; Установка бита ARMS

RET ; Возврат в вызывающую функцию

|| BCLR FRCT ; сброс бита FRCT

.end

## Приложение Б

### Разработанная программа

```
#include "tms320.h"
```

```
#include "test.h"
```

```
extern int mpy_ll_int(long long, int);
```

```
extern ushort dlms(DATA *x, DATA *h, DATA *r, DATA *des, DATA *dbuffer,  
DATA step, ushort nh, ushort nx);
```

```
void dlms_test(DATA *x, DATA *h, DATA *r, DATA *des, DATA *dbuffer,  
DATA step, ushort nh, ushort nx);
```

```
int ac3v[NX];
```

```
int main(void) {
```

```
    DATA i, j, err, max_err_r, max_err_h;
```

```
    for(j=0; j<2; j++){
```

```
        for (i = 0; i< NH; i++)
```

```
        {
```

```
            h[i] = 0;    // clear coeff buffer (optional)
```

```
            hn[i] = 0;
```

```
        }
```



```

for (i = 0; i < NX; i++){

    r[i] = 0;    // clear output buffer (optional)

    rn[i] = 0;

}


dbuffer[0] = 0;    // clear index

ndbuffer[0] = 0;

for (i = 0; i < NH+2; i++){

    dbuffer[i] = 0; // clear delay buffer (a must)

    ndbuffer[i] = 0;

}


// compute

dlms(x, h, r, des, dbuffer, STEP, NH, NX);

dlms_test(x, hn, rn, des, ndbuffer, STEP, NH, NX);

}


max_err_r = 0;

for(i = 0; i < NX; i++)

{

    err = _abss(r[i] - rn[i]);

    if(err > max_err_r)

```

```

        max_err_r = err;

    }

    max_err_h = 0;

    for(i = 0; i < NH; i++)

    {

        err = _abss(h[i] - hn[i]);

        if(err > max_err_h)

            max_err_h = err;

    }

    return 0;

}

```

```

void dlms_test(DATA *x, DATA *h, DATA *r, DATA *des, DATA *dbuffer,
DATA step, const ushort nh, const ushort nx)

```

```

{

    int *ar_data;

    int *coef_data;

    long long ac0, ac1;

    int i, j, data_index, ac3, n_iter;

    ar_data = (int*) (dbuffer+1);

    data_index = dbuffer[0];

    ac3 = 0;

```

```

for(i = 0; i < nx; i++)
{
    coef_data = (int*) h;

    *(ar_data + data_index) = x[i];

    data_index = _circ_incr(data_index, 1, nh+1);

    ac1 = 0;

    j = 0;

    //Iterations before data_index reaches maximum
    n_iter = (data_index < 2) ? nh - 1 : nh - data_index;

    while(j < n_iter)
    {
        ac0 = _llsmpy(*(ar_data + data_index), ac3);

        data_index++;

        _llslms(coef_data, (ar_data + data_index), ac0, ac1);

        *coef_data++ = (int) (ac0 >> 16);

        j++;
    }

    //Last iteration uses circular increment of data_index - we could skip
max

    ac0 = _llsmpy(*(ar_data + data_index), ac3);

    data_index = _circ_incr(data_index, 1, nh+1);

```

```

    _llslms(coef_data, (ar_data + data_index), ac0, ac1);

    *coef_data++ = (int) (ac0 >> 16);

    j++;

//Iterations after data_index reaches maximum (if we have some)
if(j < nh)
{
    n_iter = nh - n_iter - 1;

    j = 0;

    while(j < n_iter)
    {
        ac0 = _llsmpy(*(ar_data + data_index), ac3);

        data_index++;

        _llslms(coef_data, (ar_data + data_index), ac0, ac1);

        *coef_data++ = (int) (ac0 >> 16);

        j++;
    }
}

r[i] = (DATA) (ac1 >> 16);

data_index = _circ_incr(data_index, 1, nh+1);

```

```
        ac0 = (long long) des[i] - (ac1 >> 16);  
        ac3 = mpy_ll_int(ac0, step);  
  
    }  
    dbuffer[0] = data_index;  
}
```