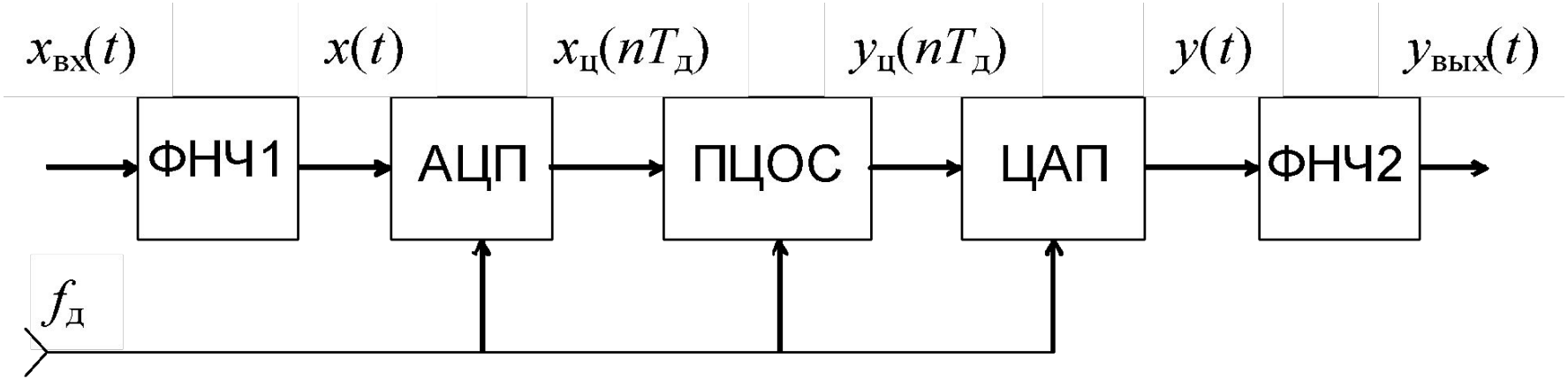
Общая структура системы цифровой обработки аналоговых сигналов



Процессор цифровой обработки сигналов(**ПЦОС**) *или* цифровой сигнальный процессор (англ. digital signal processor, **DSP**) — специализированный микропроцессор, предназначенный для обработки оцифрованных сигналов (обычно, в режиме реального времени).

Типовые задачи цифровой обработки сигналов

1) Цифровая фильтрация:
   - фильтры с конечными импульсными характеристиками (КИХ);
   - фильтры с бесконечными импульсными характеристиками (БИХ).
2) Спектральный анализ (дискретное преобразование Фурье (ДПФ), вейвлет-преобразования).
3) Корреляционный анализ.
4) Цифровой синтез непрерывных сигналов.
5) Пропорциональное автоматическое управление (ПИД-регуляторы).

Скалярное произведение векторов (сумма поэлементных произведений)

$$s = \sum_{i=0}^{N-1} x(i) \cdot y(i),$$

x(i), *y(i)* – элементы целочисленных массивов *x[0 .. N-1]*, *y[0..N-1]*;

N – длина массивов.

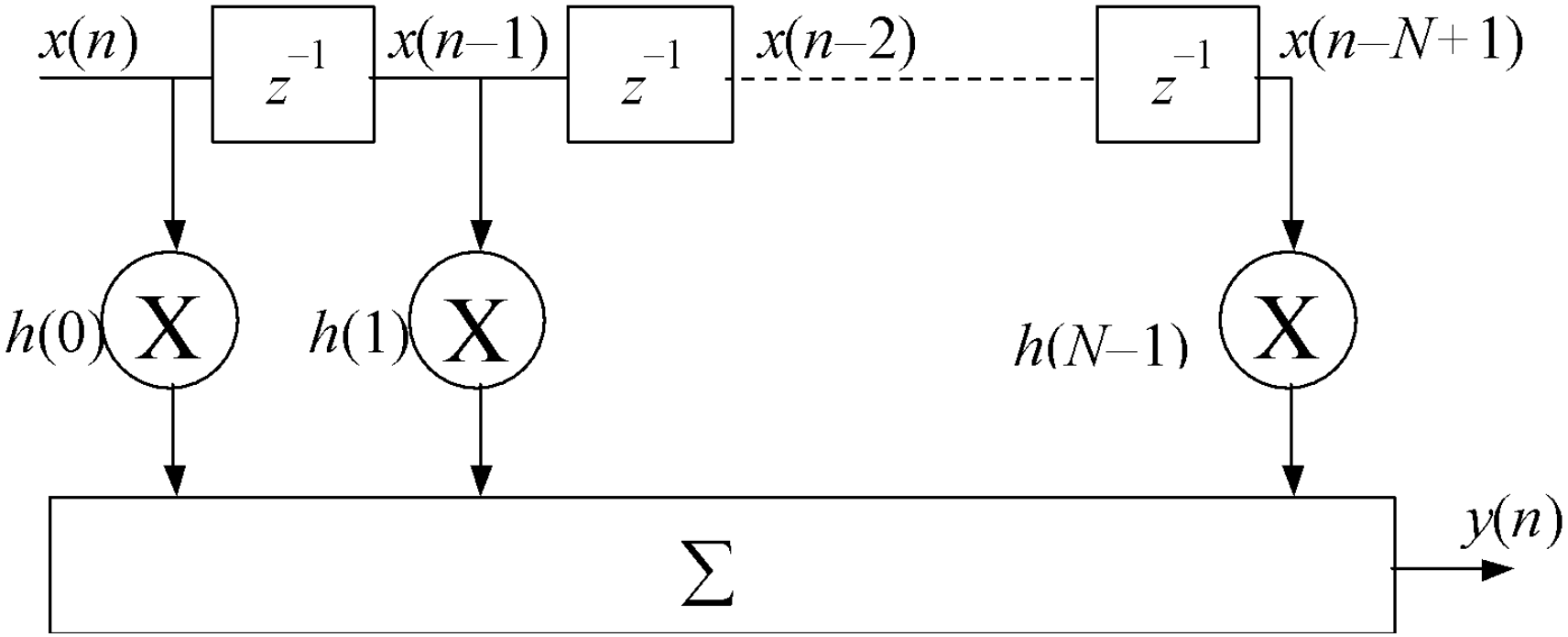
Корреляционный анализ

$$r(j) = \frac{1}{N}\sum_{n=0}^{N-1} x(n)y(n+j) = \frac{1}{N}\sum_{n=0}^{N-1} x(n-j)y(n),$$

$x(n - j)$ –дискретный отсчет задержанного сигнала $x(t - \tau)$;
$y(n)$ - дискретный отсчет сигнала $y(t)$;
N – количество принимаемых во внимание отсчетов последовательностей x, y.
Для задач ЦОС реального времени необходимо вычислять последовательность значений $r(0)$ .. $r(N)$ за время поступления N отсчетов входных сигналов

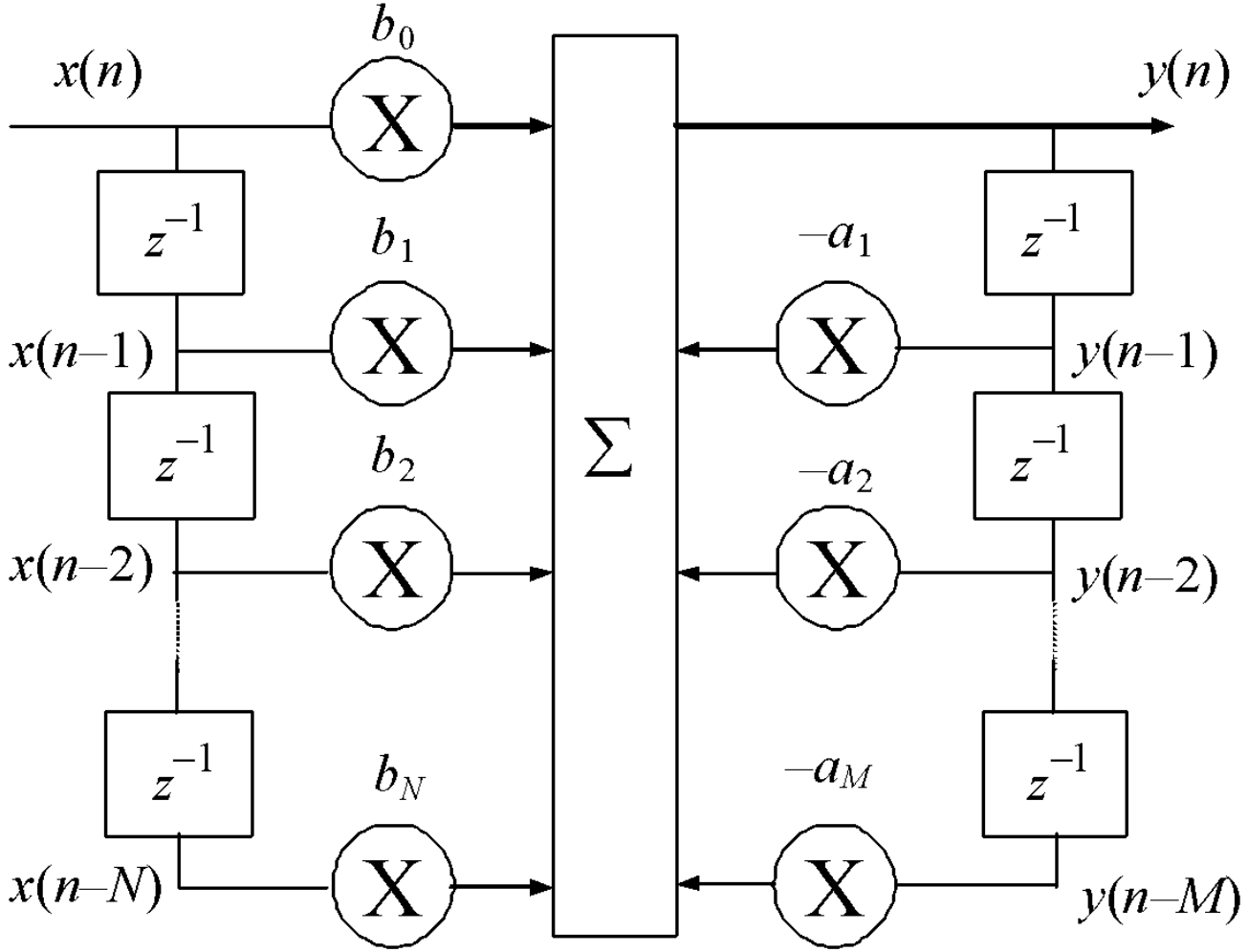
Нерекурсивный фильтр (КИХ)

$$y(n) = \sum_{m=0}^{N-1} h(m)x(n-m)$$



Рекурсивный фильтр (БИХ)

$$y(n) = \sum_{l=0}^{N} b_l x(n-l) - \sum_{k=1}^{M} a_k y(n-k).$$



Дискретное преобразование Фурье (ДПФ)

$$\dot{X}(k) = \frac{1}{N}\sum_{i=0}^{N} \dot{w}(kn)\dot{x}(n),$$

где $\dot{X}(k)$ – комплексная k-я гармоника частотного спектра;

$\dot{x}(n)$ –дискретный отсчет комплексного сигнала $\dot{x}(t)$;

$\dot{W}(kn) = exp[-(j2\pi k \cdot n)/N]$ – комплексные коэффициенты;

$n = 0 \ldots N\text{-}1$ – номер отсчета сигнала;

$k = 0 \ldots N\text{-}1$ – номер гармоники частотного спектра;

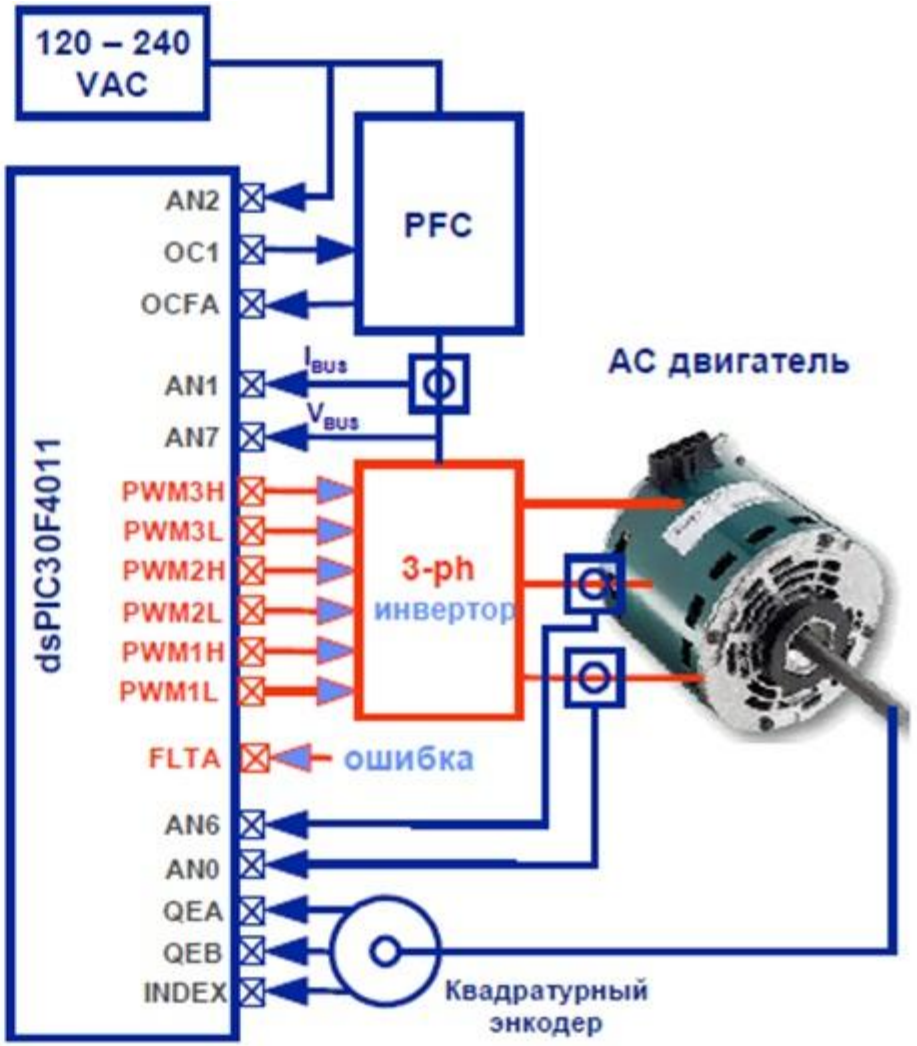
$N = 2^m$ – число отсчетов.

Области применения ЦСП

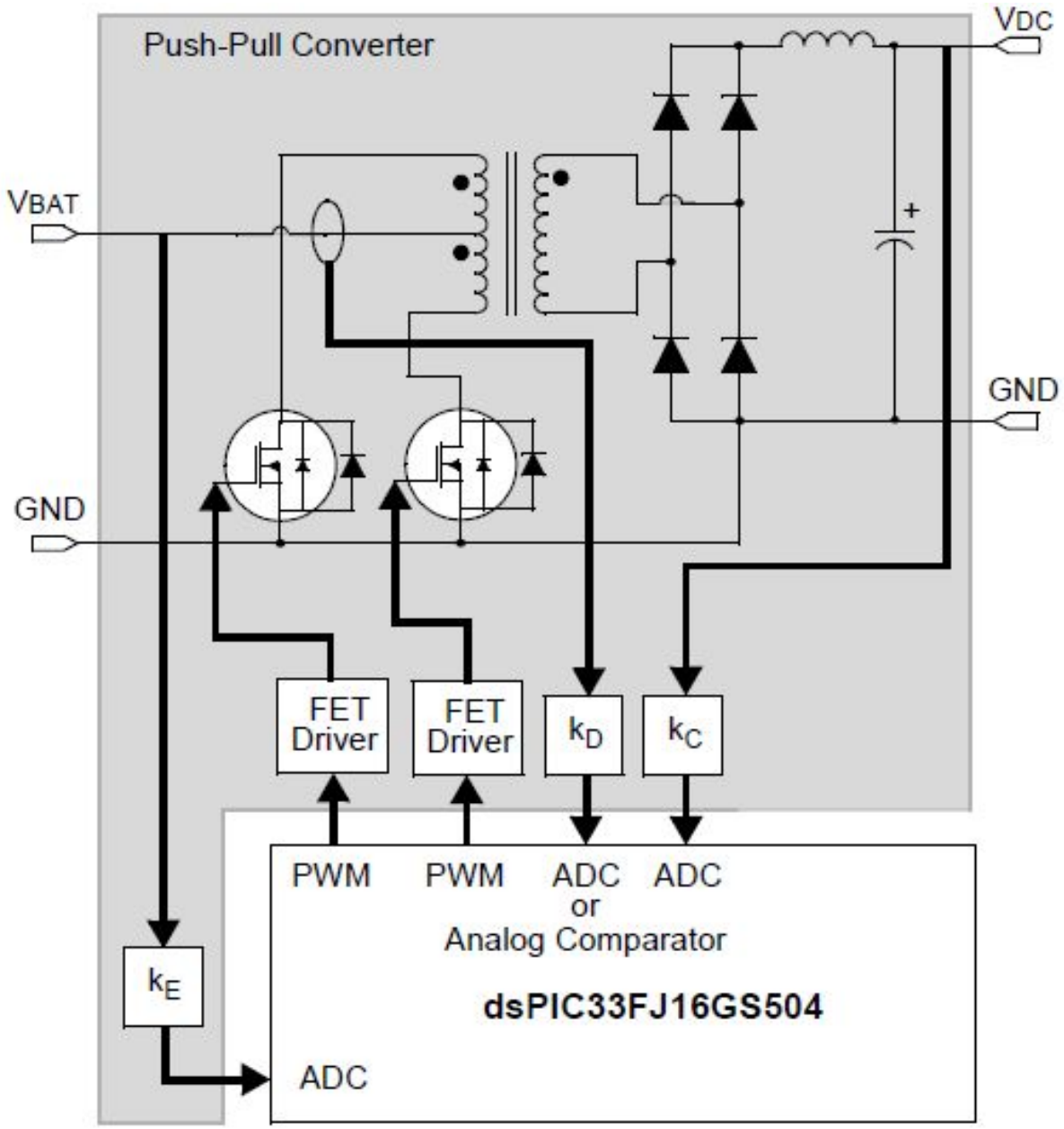
Обработка звуковых сигналов,
распознавание речи,
обработка сигналов в других частотных диапазонах,
обработка изображений,
распознавание образов,
пропорциональное автоматическое управление (регулирование) в электроприводах, преобразователях электроэнергии и других задачах).

Автоматическое управление в электроприводах

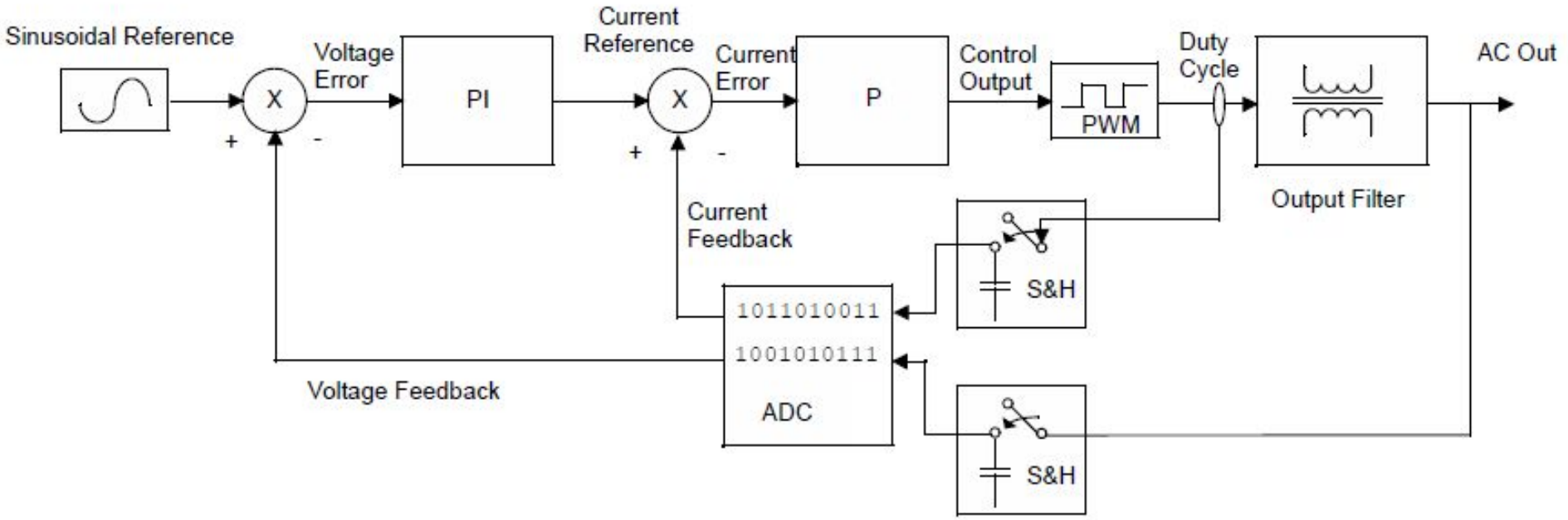
- Используется для управления двигателями
- До четырех генераторов ШИМ
- Различные настройки позволяют управлять
  - AC двигателями
  - DC двигателями
  - Источниками питания
- Высокая частота и высокое разрешение ШИМ позволяют реализовывать более сложные алгоритмы управления
- Аппаратный детектор ошибки



Управление инвертором



Контур автоматического управления инвертором



Основные особенности ПЦОС

Сигнальные процессоры оптимизированы по быстродействию для выполнения:

1) операций «умножение с накоплением» (англ. multiply-accumulate, MAC)

$Y = Y + A \times B$, где Y, A, B — элементы массивов;

2) одновременной выборки команды и двух операндов для быстрого выполнения команды MAC. Для этого в ЦСП используют модифицированную Гарвардскую архитектуру и две независимые области памяти, со своим комплектом шин адреса и данных.

3) циклов с заданной длиной (использование циклических буферов без программных счетчиков цикла);

4) ввода и вывода кодов отсчетов сигналов с равномерной дискретизацией.

Популярные модели ЦСП

TI Inc.

TMS32010 1983 г. 16-разрядный ЦСП с фиксированной точкой

Семейство TMS32C1x.

TMS32C2x 1867ВМ2Т (аналог TMS32C25)

Семейства TMS320C3x, TMS320C4x с плавающей точкой.

1867ВЦ6Ф (аналог TMS32C30)

C28x Delfino с плавающей точкой,

C28x Piccolo, C28x с фиксированной точкой,

C240x

C5000 (**C55x**) 1867ВЦ2Т (аналог TMS32C50)

1967ВЦ1Т (аналог TMS320C546A)

C6000 DSP (C66x, C674x)

C6000 DSP + ARM (66AK2x, OMAP-1Lx)

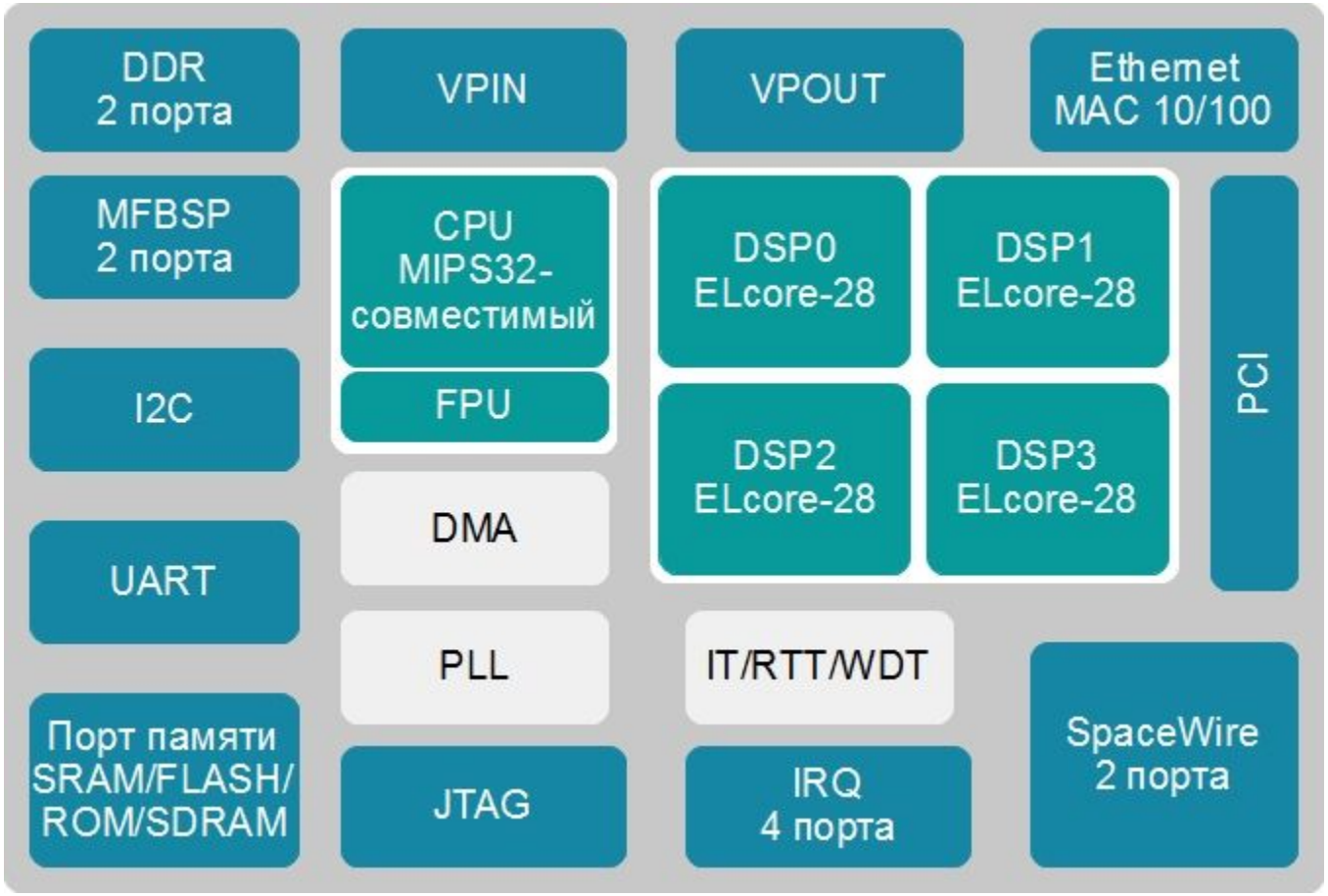
Analog Devices

ADSP-21xx

Blackfin

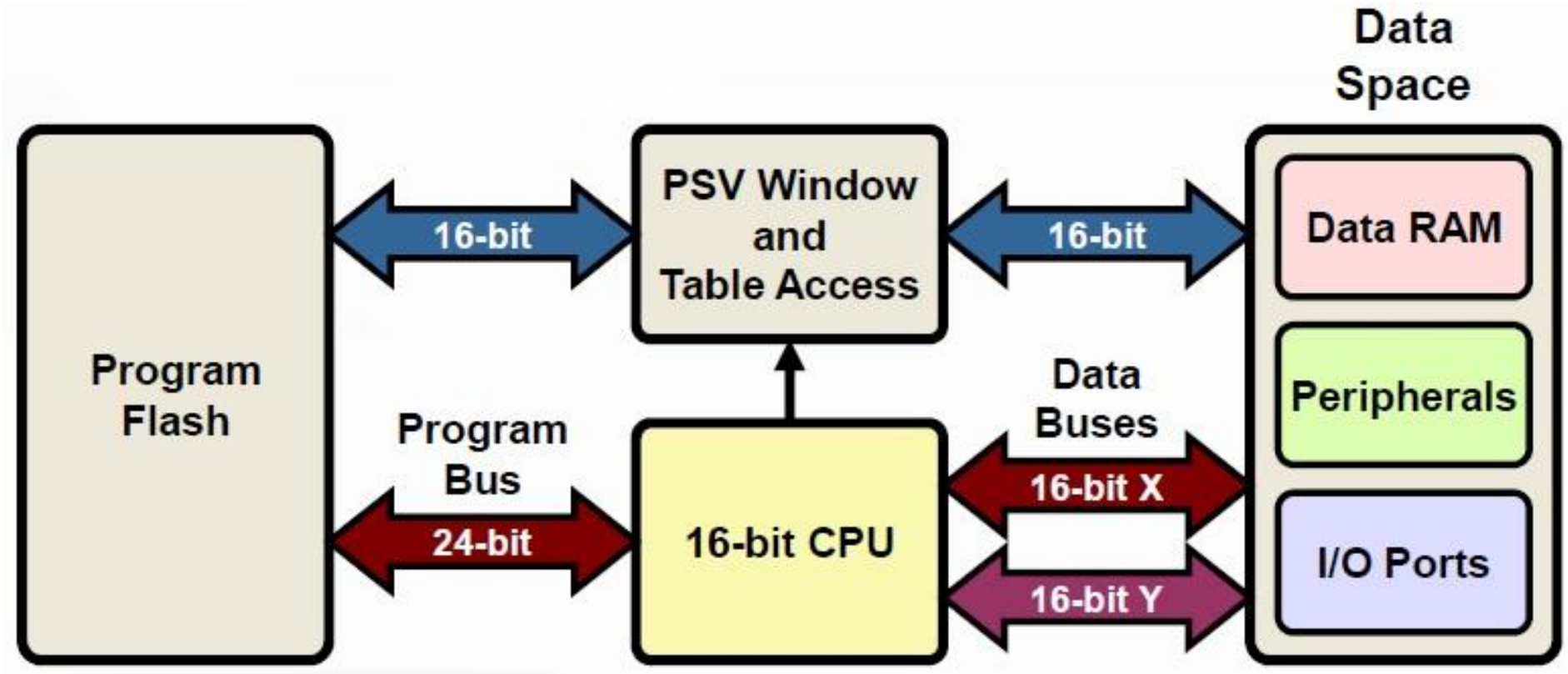
TigerSHARC 1967ВН028 (аналог ADSP-TS201)

ЦСП 1892ВМ7Я (“Элвис”) платформа «МУЛЬТИКОР»

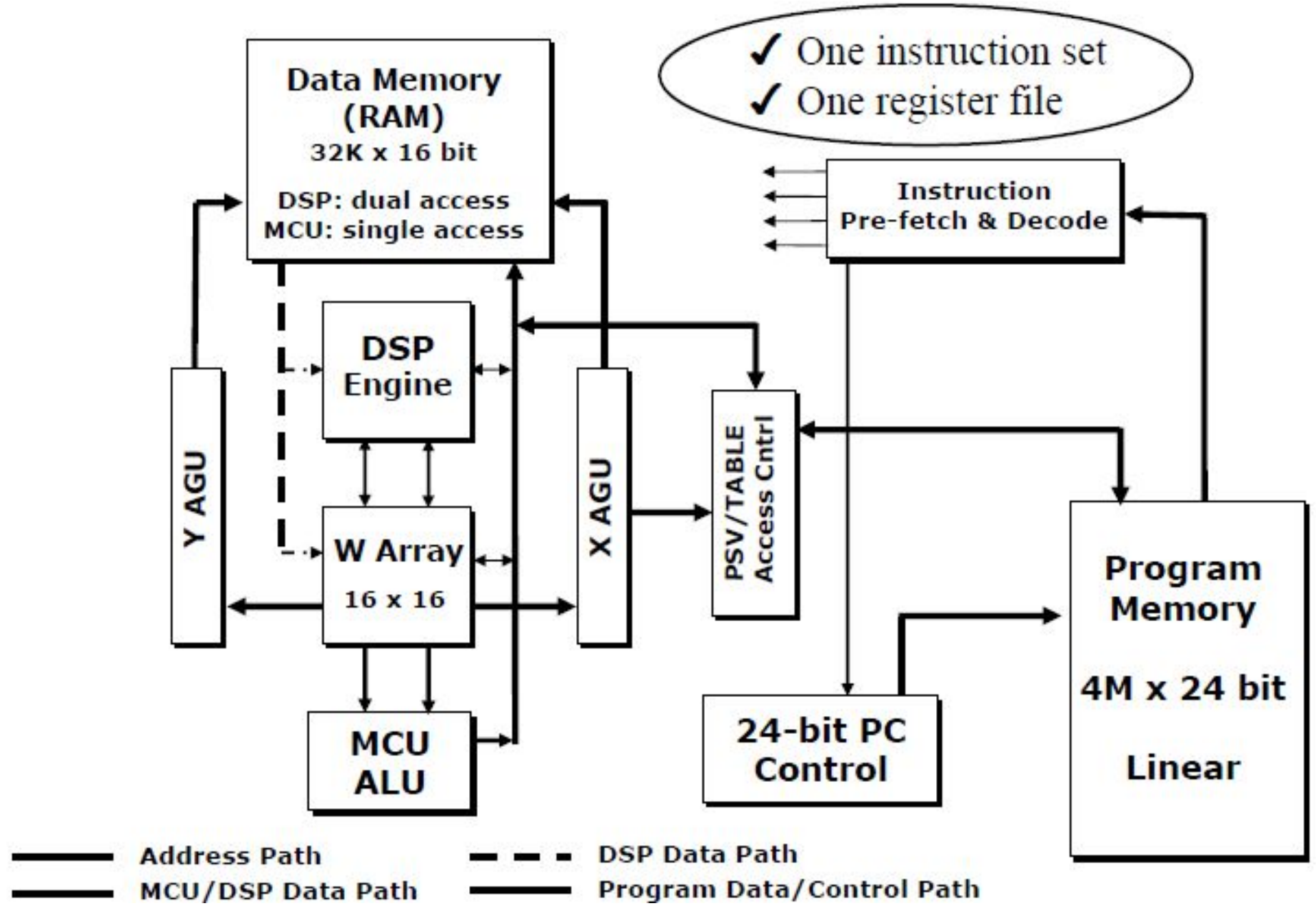


int32: 6400 Моп/с, 32 операции за 1 такт;
int16: 25600 Моп/с, 128 операций за 1 такт;
int8: 38400 Моп/с, 192 операции за 1 такт.

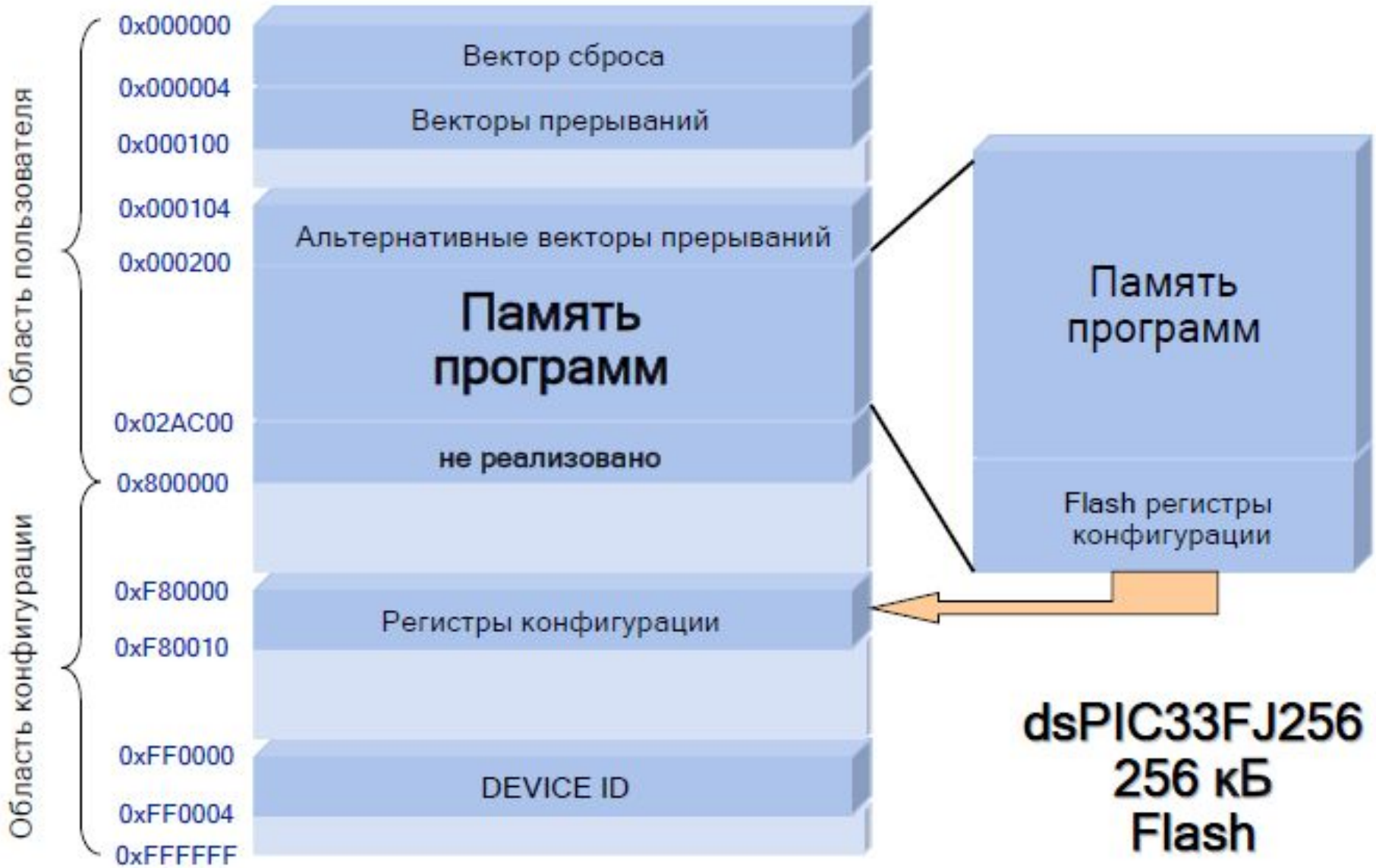
Модифицированная Гарвардская архитектура



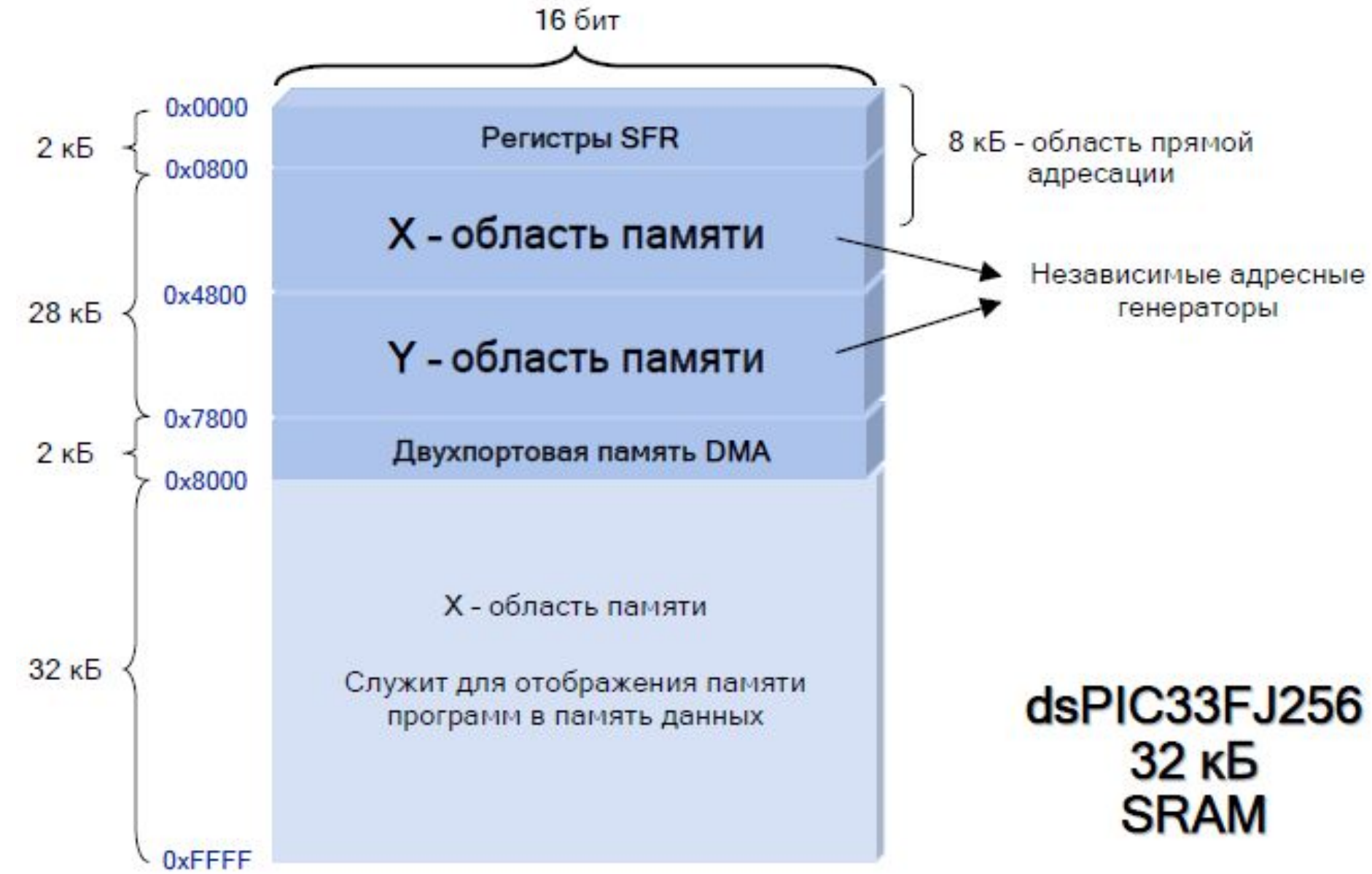
Архитектура dsPIC



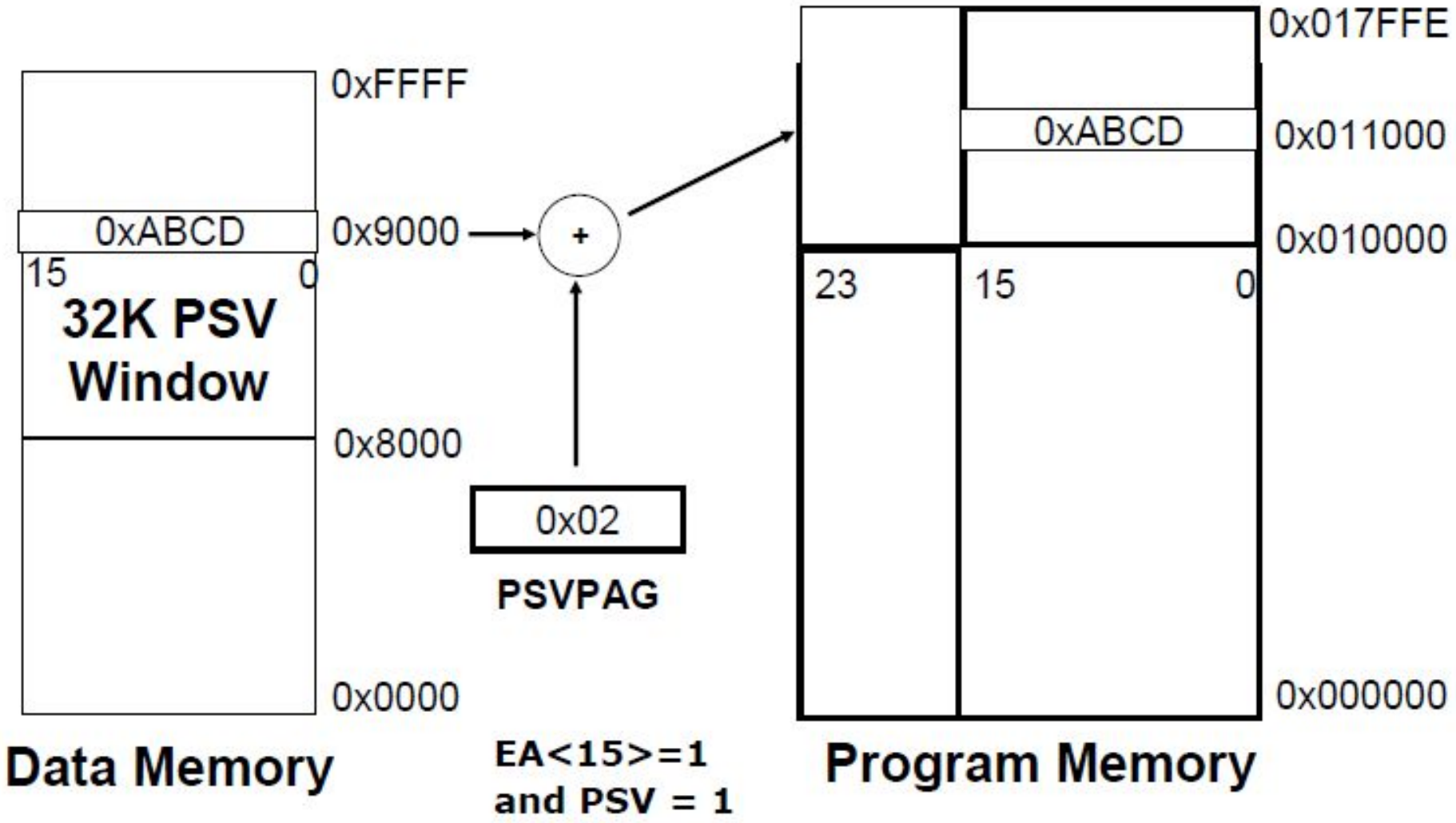
Память программ



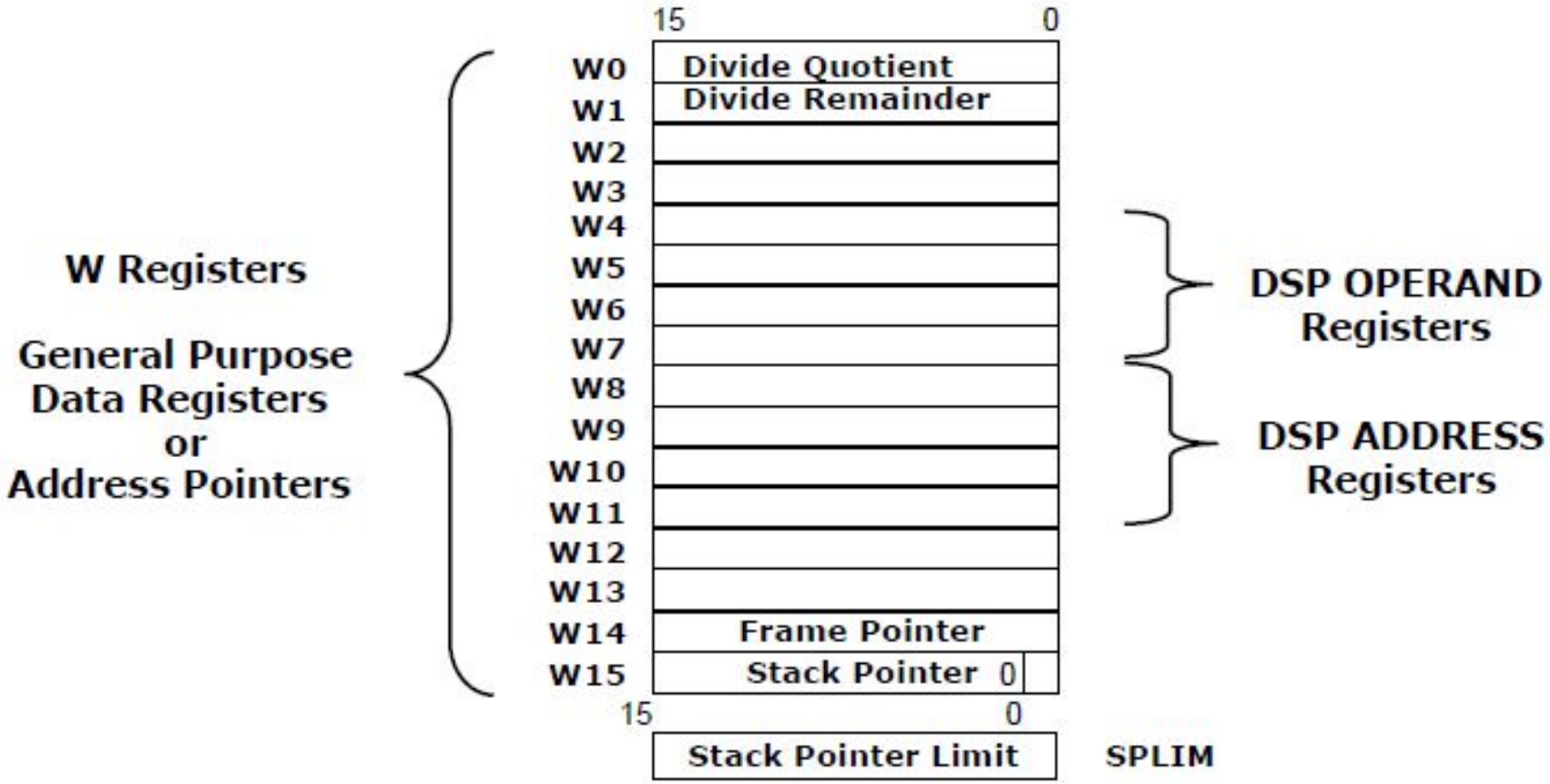
Память данных



Program Space Visibility



Programmer's Model



Addressing Modes

- Generic Addressing Modes:
  - Inherent ⇒ **NOP, RESET, PUSH, POP, etc.**
  - Literal (immediate) ⇒ **MOV #0x1800, W0**
  - Register ⇒ **ADD W4, W5, W6**
  - Memory Direct ⇒ **ADD 0x500, WREG**
  - Register Indirect with:
    - Pre-inc or Pre-dec ⇒ **ADD W4, [++W5], [--W6]**
    - Post-inc or Post-dec ⇒ **MOV [W4++], [W8--]**
    - Signed Literal Offset
  - Register Indexed ⇒ **MOV [W4+W5], [W6]**
- Special Addressing Modes:
  - Modulo (for circular buffers)
  - Bit Reverse (for FFT's)

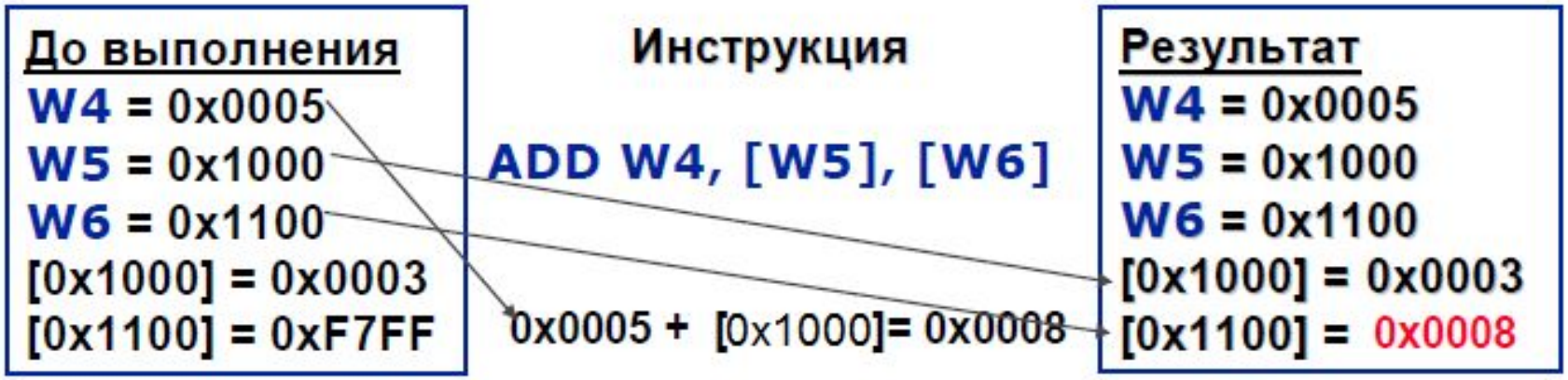
Прямая адресация

Прямая адресация регистров

- Доступ к набору W регистров
  - **W0 - W15** (0x0000 - 0x001F в памяти данных)
  - Поддержка доступа к слову и байту слова
  - Используется, когда данные находятся в W регистрах
- Например,
  - **IOR W2, W4, W6**
    - Побитовое ИЛИ значений **W2** и **W4**, сохранение результата в **W6**

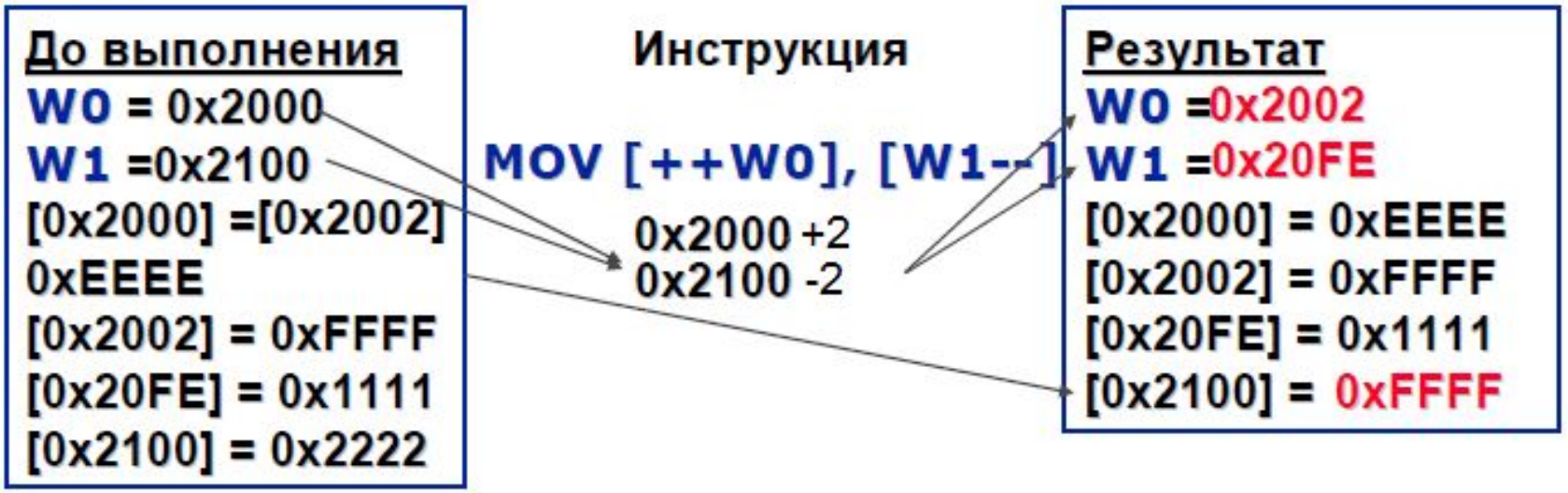
Косвенная адресация

- Значения W регистров является указателями на память
  - Обеспечивает доступ ко всему объему адресуемой памяти (64 КБ)
  - Режим косвенной адресации возможен в большинстве инструкций



Косвенная адресация с пре- и пост- модификацией

- Позволяет изменять значения указателей (декремент, инкремент)
  - Си-подобная пре- и пост- модификация указателей
  - Изменение значения указателей на 1 при доступе к байту и на 2 при доступе к слову



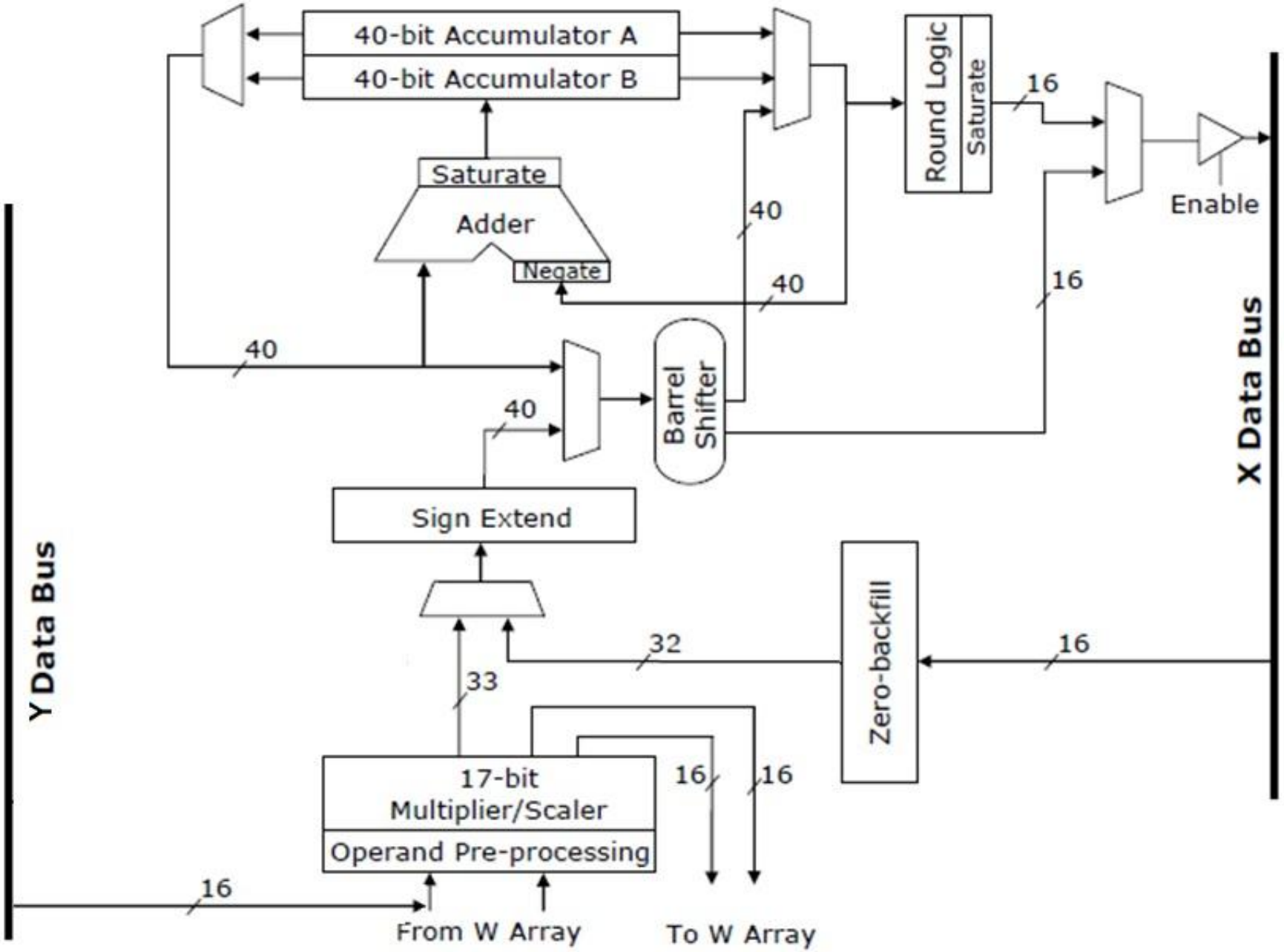
Косвенная адресация со смещением указателя

- Указатель формируется путем сложения значений двух W регистров
  - При этом значения W регистров не меняются

Наборы инструкций

- Систему команд можно условно разделить на следующие наборы инструкций:
  - (MOVE) Инструкции перемещения
  - (MATH) Математические инструкции
  - (LOGIC) Логические инструкции
  - (SHIFT / ROTATE) Инструкции сдвига
  - (BIT) Битовые инструкции
  - (STACK) Инструкции работы со стеком
  - (PROGRAM FLOW) Инструкции управления ходом программы
  - (CONTROL) Инструкции управления
  - (DSP) DSP инструкции (только для *dsPIC*)

Ядро DSP


40-bit Accumulator A
40-bit Accumulator B
Round Logic
Saturate
16
Enable
Saturate
Adder
Negate
40
40
16
40
40
Barrel Shifter
X Data Bus
Y Data Bus
Sign Extend
Zero-backfill
32
16
33
17-bit Multiplier/Scaler
Operand Pre-processing
16
16
16
From W Array
To W Array

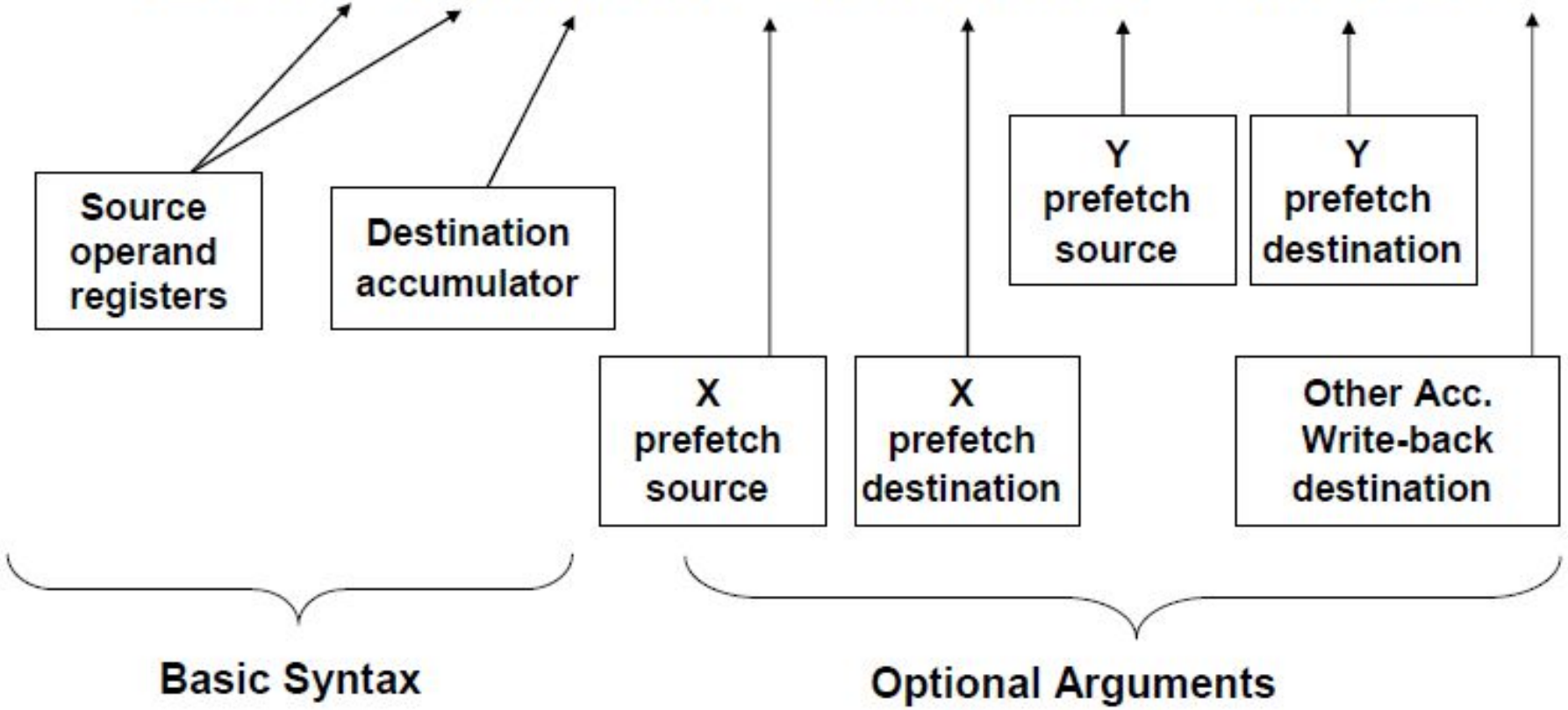
Инструкции ядра MAC

DSP инструкция	Алгебраический эквивалент	Выполняемая операция
MAC	$a = a + b \times c$	Умножение двух операндов, добавление результата к аккумулятору
MSC	$a = a - b \times c$	Умножение двух операндов, вычитание результата из аккумулятора
MPY	$a = b \times c$	Умножение двух операндов, сохранение результата в аккумулятор
MPY.N	$a = -b \times c$	Умножение двух операндов, инверсия, сохранение результата в аккумулятор
ED	$a = (b - c)^2$	Вычисление евклидовой метрики, сохранение результата в аккумулятор
EDAC	$a = a + (b - c)^2$	Вычисление евклидовой метрики, добавление результата к аккумулятору

Single Cycle MAC Instruction

FIR Filter Tap = 1 instruction cycle (33 ns)

- Sample Instruction Syntax
  - **MAC W4*W5, A, [W8]+=2, W4, [W10]-=6, W5, W13**



- значение **W4** умножается на значение **W5**, результат умножения добавляется к аккумулятору **ACCA**;
- регистр **W4** загружается значением по указателю **W8** (указатель должен указывать на адрес в сегменте X ОЗУ);
- указатель **W8** увеличивается на 2;
- регистр **W5** загружается значением по указателю **W10** (указатель должен указывать на адрес в сегменте Y ОЗУ);
- указатель **W10** уменьшается на 2;
- значение аккумулятора **ACCB** сохраняется по указателю **W13**;

Циклический буфер (модульная адресация)


```
XMODSRT = (unsigned int) array1;
XMODEND = (unsigned int) array1 + 2*LEN - 1;
YMODSRT = (unsigned int) array2;
YMODEND = (unsigned int) array2 + 2*LEN - 1;
```

```
MOV  #0xC0A8, w10        ; set XMD = W8 and YMD = W10
MOV  w10, MODCON         ; enable X & Y Modulus
```

Вычисление ВКФ

Назначение регистров:

w3 – (LEN-1)
w2 – указатель для массива значений функции R(n)
w1 – указатель для массива Y
w0 – указатель для массива X

В цикле вычисления R[n]
W8 – указатель для массива X
W10 – указатель для массива Y
W4, W5 – хранение извлеченных сомножителей

```
sub     W3,#2, W6                    ; W6 = LEN-3 (для do loop )
```

$$R(0) = y(0)\cdot x(0) + y(1)\cdot x(1) + y(2)\cdot x(2) + y(3)\cdot x(3)$$
$$R(1) = y(3)\cdot x(0) + y(0)\cdot x(1) + y(1)\cdot x(2) + y(2)\cdot x(3)$$
$$R(2) = y(2)\cdot x(0) + y(3)\cdot x(1) + y(0)\cdot x(2) + y(1)\cdot x(3)$$
$$R(3) = y(1)\cdot x(0) + y(2)\cdot x(1) + y(3)\cdot x(2) + y(0)\cdot x(3)$$

[W8]	W[10]
x(0)	**y(0)**
x(1)	y(1)
x(2)	y(2)
x(3)	y(3)

[W8]	W[10]
x(0)	**y(3)**
x(1)	y(0)
x(2)	y(1)
x(3)	y(2)

[W8]	W[10]
x(0)	**y(2)**
x(1)	y(3)
x(2)	y(0)
x(3)	y(1)

[W8]	W[10]
x(0)	**y(1)**
x(1)	y(2)
x(2)	y(3)
x(3)	y(0)

Цикл вычисления R[n]

```
        DO   w3, array_loop              ; цикл обработки всех входных выборок

; очистка A, выборка  отсчетов , модификация указателей
        CLR   a, [w8]+=2, w4, [w10]+=2, w5

        REPEAT w6                        ; Выполнение MAC (кроме двух последних)
        MAC  w4*w5, a, [w8]+=2, w5, [w10]+=2, w5
        MAC  w4*w5, a, [w8]+=2, w4, [w10], w5    ; предпоследняя MAC
        MAC  w4*w5, a                    ; последняя MAC
                    ; округление  и сохранение A в выходном буфере
array_loop :
        SAC.R  a,[w2++]                  ;сохранение результата
```

Реализация КИХ-фильтра

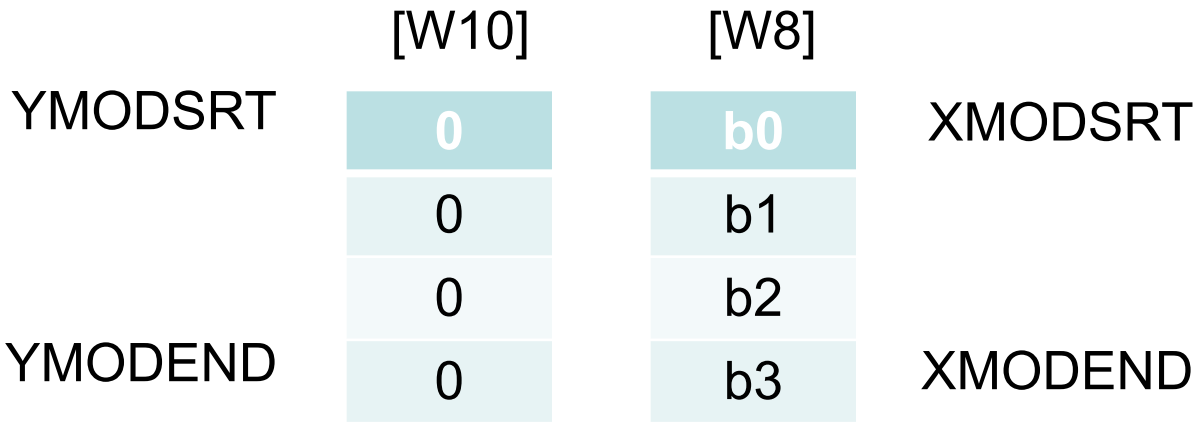
Массив коэффициентов Taps **.section .xdata, data, xmemory**
.section .psvconst, code

Буфер задержки Delay **.section .ydata, data, ymemory**

Структура FIRStruct

```
typedef struct
{
  int  numTaps;          // число коэффициентов
  int *pTapsBase;        // базовый адрес массива коэффициентов
                  // или базовый адрес смещения (psvoffset)
  int *pTapsEnd;         // адрес последнего элемента массива
  int  tapsPage;         // 0xFF00 или номер страницы psvpage
  int *pDelayBase;       // базовый адрес буфера задержки
  int *pDelayEnd;        // адрес последнего элемента буфера
  int *pDelayPtr;        // стартовое значение указателя
} FIRStruct;
```

Циклический буфер (модульная адресация)




```
SetupPointers:
    MOV  [w3+oTapsEnd],w8
    MOV  w8, XMODEND          ; XMODEND = конечный адрес коэффициентов
    MOV  [w3+oTapsBase],w8
    MOV  w8, XMODSRT          ; XMODSRT = базовый адрес коэффициентов
    MOV  [w3+oDelayEnd],w10
    MOV  w10, YMODEND         ; YMODEND = конечный адрес линии задержки
    MOV  [w3+oDelayBase],w10
    MOV  w10, YMODSRT         ; YMODSRT = базовый адрес линии задержки
    MOV  #0xC0A8, w10         ; XMD = W8 and YMD = W10
    MOV  w10, MODCON          ; разрешить X & Y Modulus
    DEC  w0,w0                 ; w0 счетчик циклов
    MOV  [w3+oDelayPtr],w10    ; указатель на текущий элемент линии задержки
```

$$y(0) = b0 \cdot x(0) + b1 \cdot x(-1) + b2 \cdot x(-2) + b3 \cdot x(-3)$$
$$y(1) = b0 \cdot x(1) + b1 \cdot x(0) + b2 \cdot x(-1) + b3 \cdot x(-2)$$
$$y(2) = b0 \cdot x(2) + b1 \cdot x(1) + b2 \cdot x(0) + b3 \cdot x(-1)$$
$$y(3) = b0 \cdot x(3) + b1 \cdot x(2) + b2 \cdot x(1) + b3 \cdot x(0)$$

[W10]	W[8]
X(0)	b0
x(-1)	b1
x(-2)	b2
x(-3)	b3

[W10]	W[8]
X(0)	b0
x(-1)	b1
x(-2)	b2
x(1)	b3

[W10]	W[8]
X(0)	b0
x(-1)	b1
x(2)	b2
x(1)	b3

[W10]	W[8]
X(0)	b0
x(3)	b1
x(2)	b2
x(1)	b3

Назначение регистров:

w3 – указатель на структуру FIR фильтра
w2 – указатель на буфер входных отсчетов
w1 – указатель на буфер выходных отсчетов
w0 – количество выходных отсчетов

В цикле вычисления y[n]
W8 – указатель для буфера коэффициентов
W10 – указатель для буфера линии задержки отсчетов x(n)
W5, W6 – хранение извлеченных сомножителей

Цикл вычисления y[n]

```
        DO   w0, blockLoop              ; цикл обработки всех входных выборок
        MOV  [w2++],[w10]               ; сохранение нового отсчета в буфере

; очистка A, выборка  коэффиц. и входного отсчета, модификация указателей
        CLR   a, [w8]+=2, w5, [w10]+=2, w6

        REPEAT w4                       ; Выполнение MAC (кроме двух последних)
        MAC  w5*w6, a, [w8]+=2, w5, [w10]+=2, w6
        MAC  w5*w6, a, [w8]+=2, w5, [w10], w6    ; предпоследняя MAC
        MAC  w5*w6, a                   ; последняя MAC
                    ; округление  и сохранение A в выходном буфере
blockLoop:
        SAC.R  a,[w1++]                 ;управляется W3
        MOV  w10,[w3+oDelayPtr]         ;обновление указателя
```

Реализация БИХ-фильтра

Массив коэффициентов Coefs **.section .xdata, data, xmemory**
.section .psvconst, code

Буферы состояний
States1, States2 **.section .ydata, data, ymemory**

Структура IIRTransposedStruct

```
typedef struct
{
  int  numSectionsLess1;// число секций второго порядка
  int *pCoefs;          // указатель на массив коэффициентов
  int  psvpage;         // 0xFF00 или номер страницы памяти программ
  int *pStates1;        // указатель на буфер состояний 1
  int *pStates2;        // указатель на буфер состояний 2
  int  finalShift;      // число разрядов сдвига для нормализации
} IIRTransposedStruct;
```

Назначение регистров:

w3 – указатель на структуру БИХ фильтра
w2 – указатель на буфер входных отсчетов
w1 – указатель на буфер выходных отсчетов
w0 – количество выходных отсчетов

В цикле вычисления y[n]
W4 – число секций -1
W9 – число сдвигов для нормализации выходных отсчетов
W3 - число выходных отсчетов -1

W5 –коэффициент
W6 – следующий входной отсчет
W8 – указатель для буфера коэффициентов
W10 – указатель для буфера состояния states1
W11 – указатель для буфера состояния states2

W5, W6, W7 – хранение сомножителей

```
_IIRTransposed:
; сохранение контекста
; инициализация рабочих регистров
; инициализация указателей
        DO w0, transposeBlockLoop ; внешний цикл (по числу выборок)

        MOV [w3+oCoefs], w8          ; w8 = базовый адрес коэффиц.
        MOV [w2++], w6                ; w6 = следующая выборка

        MOV [w3+oStates1], w10      ; w10 = базовый адрес буфера states1
        MOV [w3+oStates2], w11      ; w11 = базовый адрес буфера states2
        MOV [w8++], w5                ; выборка первого коэффициента
        LAC [w10], #1, a              ; выборка состояния фильтра
```

```
        DO  w4, transposeSectionLoop ; внутренний цикл (по числу секций)

        MAC w5*w6, a, [w8]+=2, w5
        LAC [w11], #1, b
        SAC.R a, # -1, w7
        MAC w5*w6, b, [w8]+=2, w5
        MAC w5*w7, b, [w8]+=2, w5
        SAC.R b, # -1, [w10++]
        MPY w5*w6, b, [w8]+=2, w5
        SAC.R a, # -1, w6
        LAC [w10], #1, a
        MAC w5*w7, b, [w8]+=2, w5
transposeSectionLoop:
        SAC.R b, #-1, [w11++]

        LAC w6, a
        SFTAC a, w9                     ; арифметический сдвиг
transposeBlockLoop:
        SAC.R a, [w1++]

; восстановление контекста
```

Дискретное преобразование Фурье

$$\dot{X}(k) = \frac{1}{N}\sum_{n=0}^{N-1} x(n)e^{-j2\pi nk/N} = \frac{1}{N}\sum_{n=0}^{N-1} x(n)[\cos(2\pi nk/N) - j\sin(2\pi nk/N)]$$

8-точечное ДПФ (N=8)

$$X(k) = \frac{1}{N} \sum_{n=0}^{N-1} x(n)\, e^{\frac{-j2\pi nk}{N}} = \frac{1}{N} \sum_{n=0}^{N-1} x(n)\, W_N^{nk}$$

$$W_N = e^{\frac{-j2\pi}{N}}$$

$X(0) =$	$x(0)W_8^0$	$+ x(1)W_8^0$	$+ x(2)W_8^0$	$+ x(3)W_8^0$	$+ x(4)W_8^0$	$+ x(5)W_8^0$	$+ x(6)W_8^0$	$+ x(7)W_8^0$
$X(1) =$	$x(0)W_8^0$	$+ x(1)W_8^1$	$+ x(2)W_8^2$	$+ x(3)W_8^3$	$+ x(4)W_8^4$	$+ x(5)W_8^5$	$+ x(6)W_8^6$	$+ x(7)W_8^7$
$X(2) =$	$x(0)W_8^0$	$+ x(1)W_8^2$	$+ x(2)W_8^4$	$+ x(3)W_8^6$	$+ x(4)W_8^8$	$+ x(5)W_8^{10}$	$+ x(6)W_8^{12}$	$+ x(7)W_8^{14}$
$X(3) =$	$x(0)W_8^0$	$+ x(1)W_8^3$	$+ x(2)W_8^6$	$+ x(3)W_8^9$	$+ x(4)W_8^{12}$	$+ x(5)W_8^{15}$	$+ x(6)W_8^{18}$	$+ x(7)W_8^{21}$
$X(4) =$	$x(0)W_8^0$	$+ x(1)W_8^4$	$+ x(2)W_8^8$	$+ x(3)W_8^{12}$	$+ x(4)W_8^{16}$	$+ x(5)W_8^{20}$	$+ x(6)W_8^{24}$	$+ x(7)W_8^{28}$
$X(5) =$	$x(0)W_8^0$	$+ x(1)W_8^5$	$+ x(2)W_8^{10}$	$+ x(3)W_8^{15}$	$+ x(4)W_8^{20}$	$+ x(5)W_8^{25}$	$+ x(6)W_8^{30}$	$+ x(7)W_8^{35}$
$X(6) =$	$x(0)W_8^0$	$+ x(1)W_8^6$	$+ x(2)W_8^{12}$	$+ x(3)W_8^{18}$	$+ x(4)W_8^{24}$	$+ x(5)W_8^{30}$	$+ x(6)W_8^{36}$	$+ x(7)W_8^{42}$
$X(7) =$	$x(0)W_8^0$	$+ x(1)W_8^7$	$+ x(2)W_8^{14}$	$+ x(3)W_8^{21}$	$+ x(4)W_8^{28}$	$+ x(5)W_8^{35}$	$+ x(6)W_8^{42}$	$+ x(7)W_8^{49}$

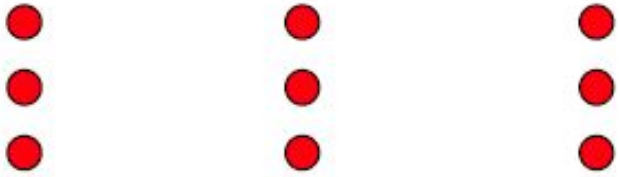
N^2 умножений с комплексными числами

$\frac{1}{N}$ Не учтенный масштабный коэффициент

симметричность: $W_N^{r+N/2} = -W_N^{r}$, периодичность: $W_N^{r+N} = W_N^{r}$

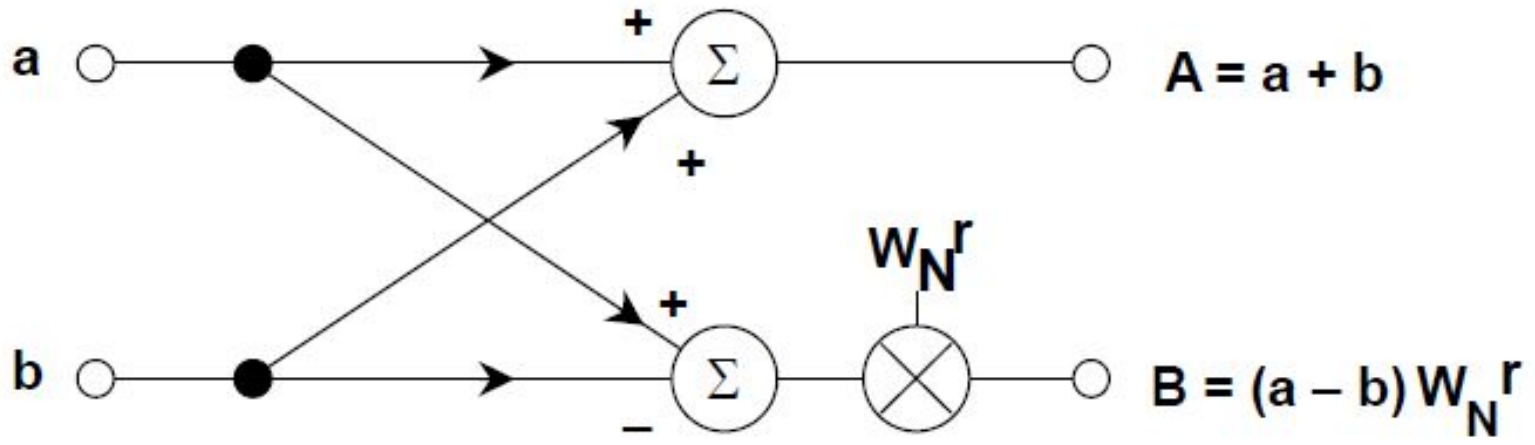
$N = 8$

$$W_8^4 = W_8^{0+4} = -W_8^0 = -1$$
$$W_8^5 = W_8^{1+4} = -W_8^1$$
$$W_8^6 = W_8^{2+4} = -W_8^2$$
$$W_8^7 = W_8^{3+4} = -W_8^3$$
$$W_8^8 = W_8^{0+8} = +W_8^0 = +1$$
$$W_8^9 = W_8^{1+8} = +W_8^1$$
$$W_8^{10} = W_8^{2+8} = +W_8^2$$
$$W_8^{11} = W_8^{3+8} = +W_8^3$$

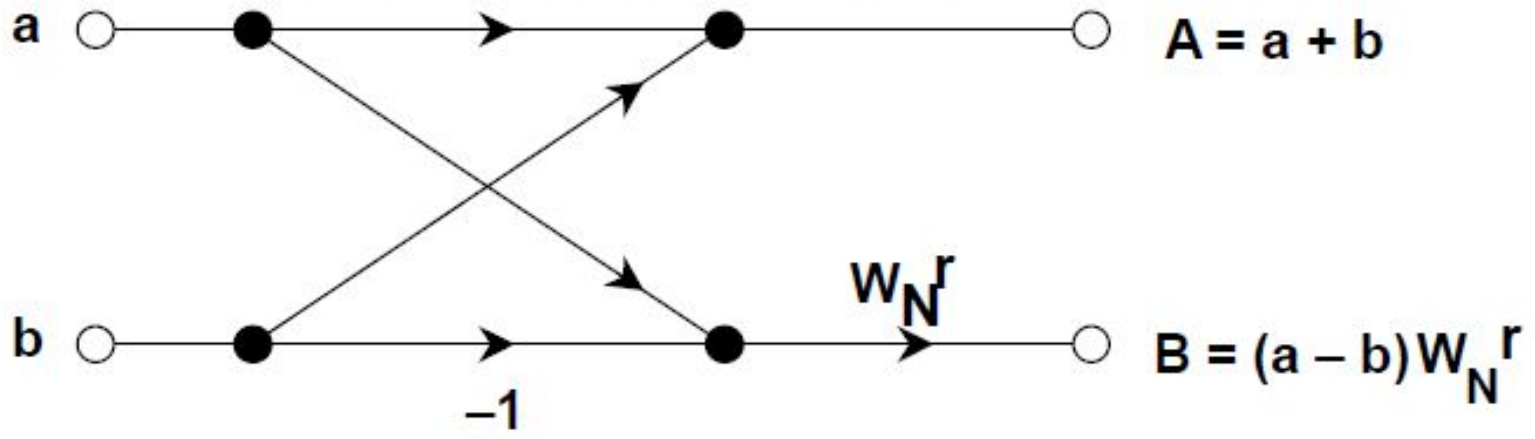



x(0)
x(1)
x(2)
x(3)
x(4)
x(5)
x(6)
x(7)
РАЗБИЕНИЕ НА 4-ТОЧЕЧНЫЕ ДПФ
РАЗБИЕНИЕ НА 2-ТОЧЕЧНЫЕ ДПФ
РАЗБИЕНИЕ НА 2-ТОЧЕЧНЫЕ ДПФ
2-ТОЧЕЧНОЕ ДПФ
2-ТОЧЕЧНОЕ ДПФ
2-ТОЧЕЧНОЕ ДПФ
2-ТОЧЕЧНОЕ ДПФ
X(0)
X(4)
X(2)
X(6)
X(1)
X(5)
X(3)
X(7)

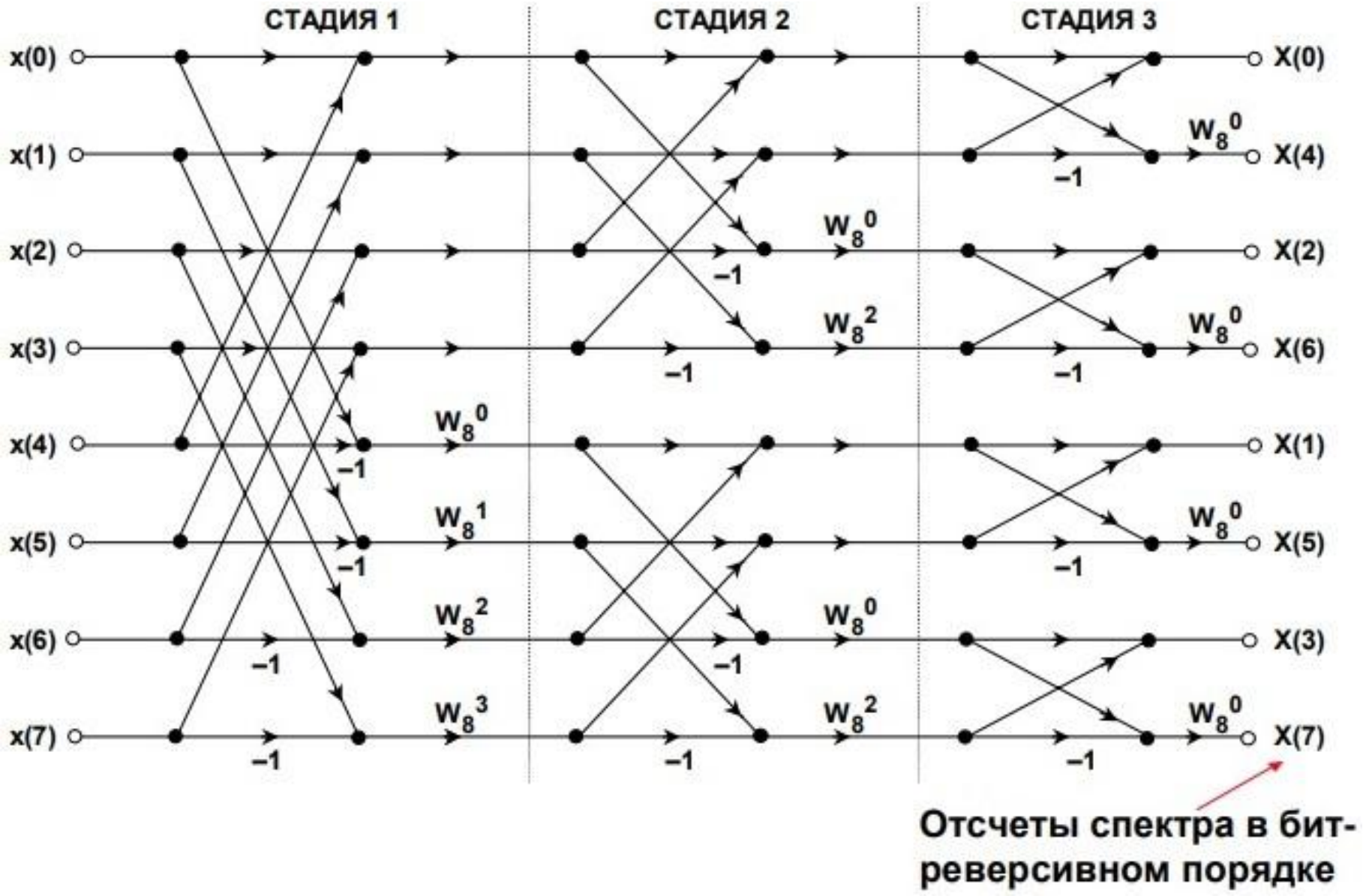
Операция «бабочка»



УПРОЩЕННОЕ ПРЕДСТАВЛЕНИЕ



Алгоритм 8-точечного БПФ



Упорядочивание элементов массива (бит-реверсная адресация)

■ Десятичное число:	0	1	2	3	4	5	6	7
■ Двоичный эквивалент:	000	001	010	011	100	101	110	111
■ Дв. с реверсированием:	000	100	010	110	001	101	011	111
■ Десятичный эквивалент:	0	4	2	6	1	5	3	7

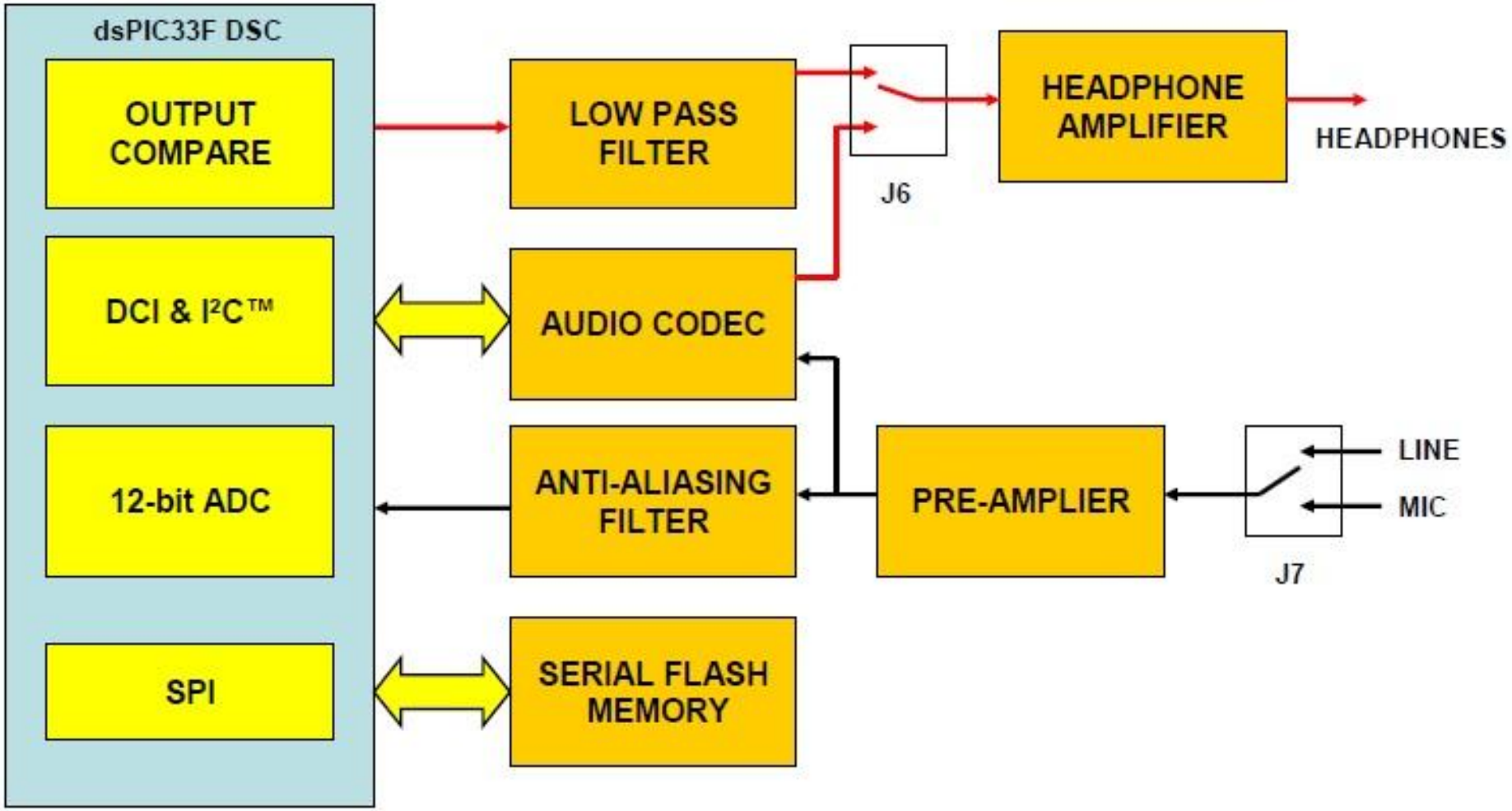
Структура данных

```
int output[64];                    //  выходной массив фильтра
int i;                             //  переменная цикла

#define FFT_BLOCK_LENGTH    64      // длина блока
#define LOG2_BLOCK_LENGTH   6       // log2(64) – количество стадий

// рабочие массивы алгоритма БПФ
fractcomplex fftInputOutput[FFT_BLOCK_LENGTH]  //массив выборок
_YBSS(FFT_BLOCK_LENGTH * 2 * 2);

fractcomplex twiddleFactors[FFT_BLOCK_LENGTH/2]  //массив коэффиц.
_XBSS(FFT_BLOCK_LENGTH * 2);

fractional preFilterFFTMag[FFT_BLOCK_LENGTH];  // модули гармоник
fractional postFilterFFTMag[FFT_BLOCK_LENGTH]; // модули гармоник
```

Функции программы (API)

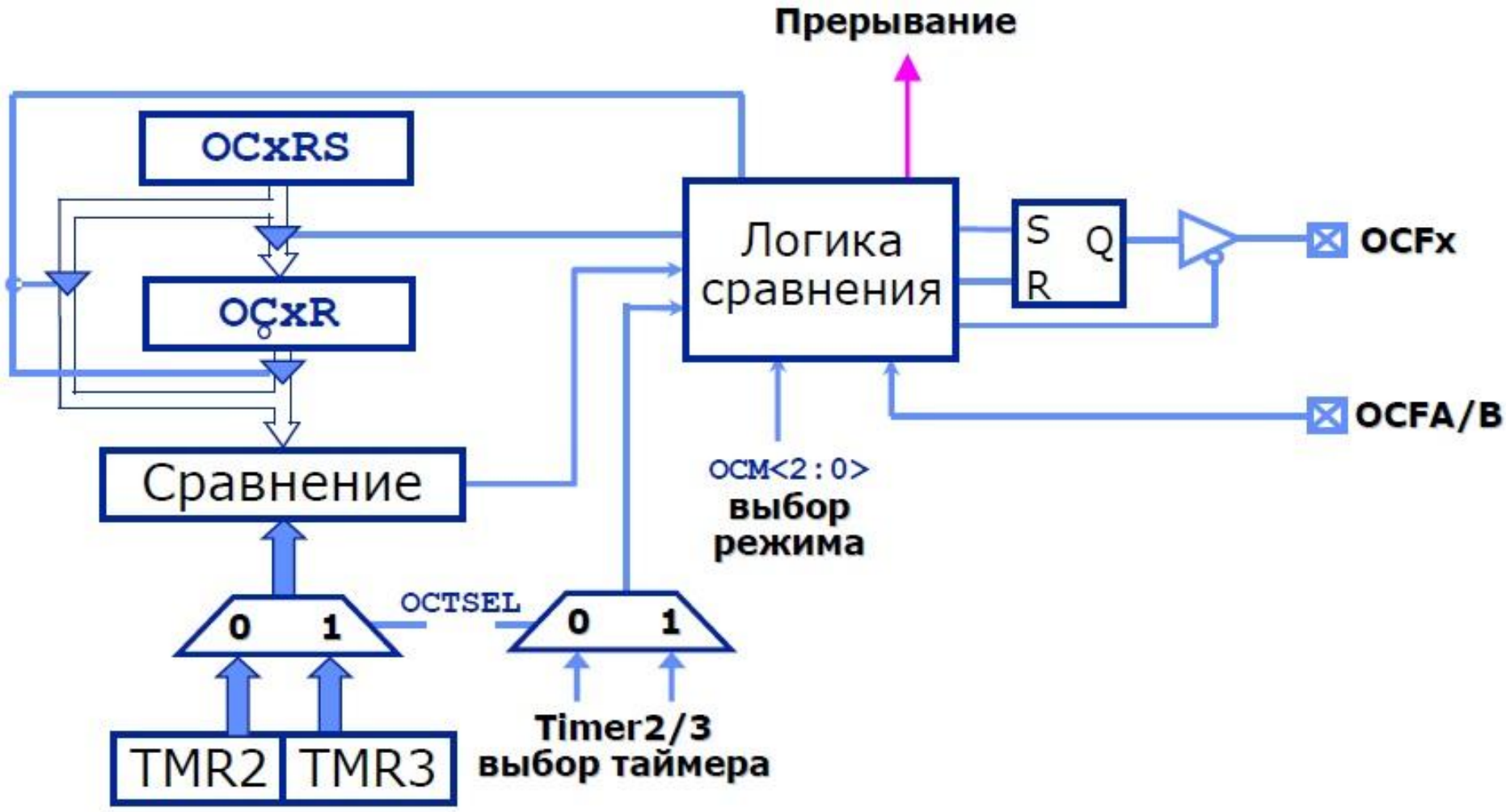
```
IIRTransposedInit(&Filter);          //инициализация структуры фильтра
TwidFactorInit (LOG2_BLOCK_LENGTH, &twiddleFactors[0], 0);
                                  //инициализация массива коэффициентов
 IIRTransposed(64, output, composite, &Filter);  КИХ фильтр

 FFTComplexIP(LOG2_BLOCK_LENGTH,  fftInputOutput, twiddleFactors,
COEFFS_IN_DATA);                  // стадии БПФ
 BitReverseComplex(LOG2_BLOCK_LENGTH, fftInputOutput);
                                 // перестановка элементов
 SquareMagnitudeCplx(FFT_BLOCK_LENGTH, fftInputOutput,
preFilterFFTMag);                 //  вычисление квадрата модулей
```

Структура стенда Starter Kit for dsPIC DSC



Периферийные модули dsPIC

Модуль сравнения в режиме ШИМ



REGISTER 15-1: OCxCON: OUTPUT COMPARE x CONTROL REGISTER

U-0	U-0	R/W-0	U-0	U-0	U-0	U-0	U-0
—	—	OCSIDL	—	—	—	—	—
bit 15							bit 8

U-0	U-0	U-0	R-0, HC	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	OCFLT	OCTSEL	OCM<2:0>		
bit 7							bit 0

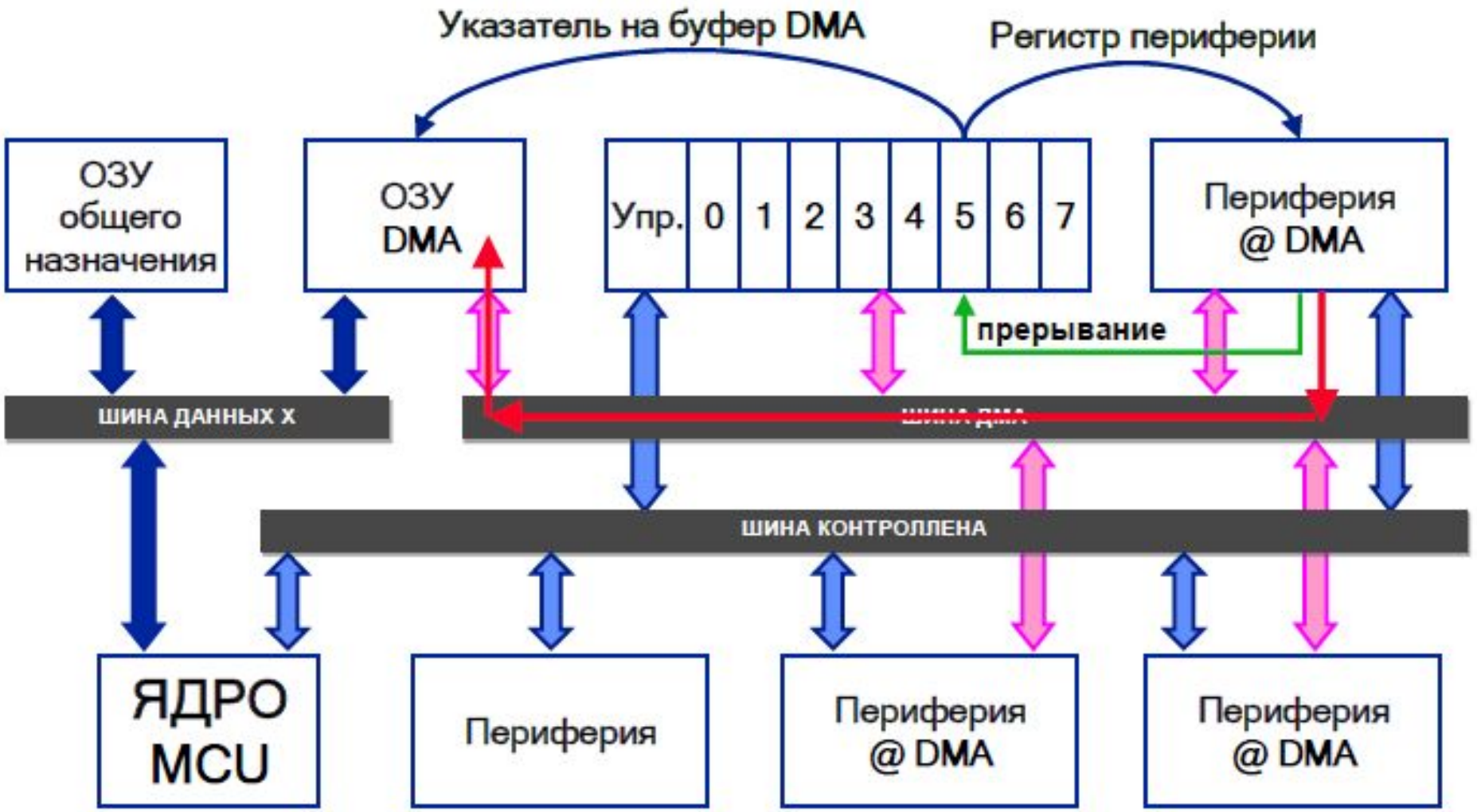
bit 13 **OCSIDL:** Stop Output Compare in Idle Mode Control bit
1 = Output Compare x halts in CPU Idle mode
0 = Output Compare x continues to operate in CPU Idle mode

bit 4 **OCFLT:** PWM Fault Condition Status bit
1 = PWM Fault condition has occurred (cleared in hardware only)
0 = No PWM Fault condition has occurred (this bit is only used when OCM<2:0> = 111)

bit 3 **OCTSEL:** Output Compare Timer Select bit
1 = Timer3 is the clock source for Compare x
0 = Timer2 is the clock source for Compare x

bit 2-0 **OCM<2:0>:** Output Compare Mode Select bits
111 = PWM mode on OCx, Fault pin enabled
110 = PWM mode on OCx, Fault pin disabled
101 = Initialize OCx pin low, generate continuous output pulses on OCx pin
100 = Initialize OCx pin low, generate single output pulse on OCx pin
011 = Compare event toggles OCx pin
010 = Initialize OCx pin high, compare event forces OCx pin low
001 = Initialize OCx pin low, compare event forces OCx pin high
000 = Output compare channel is disabled

Структурная схема DMA

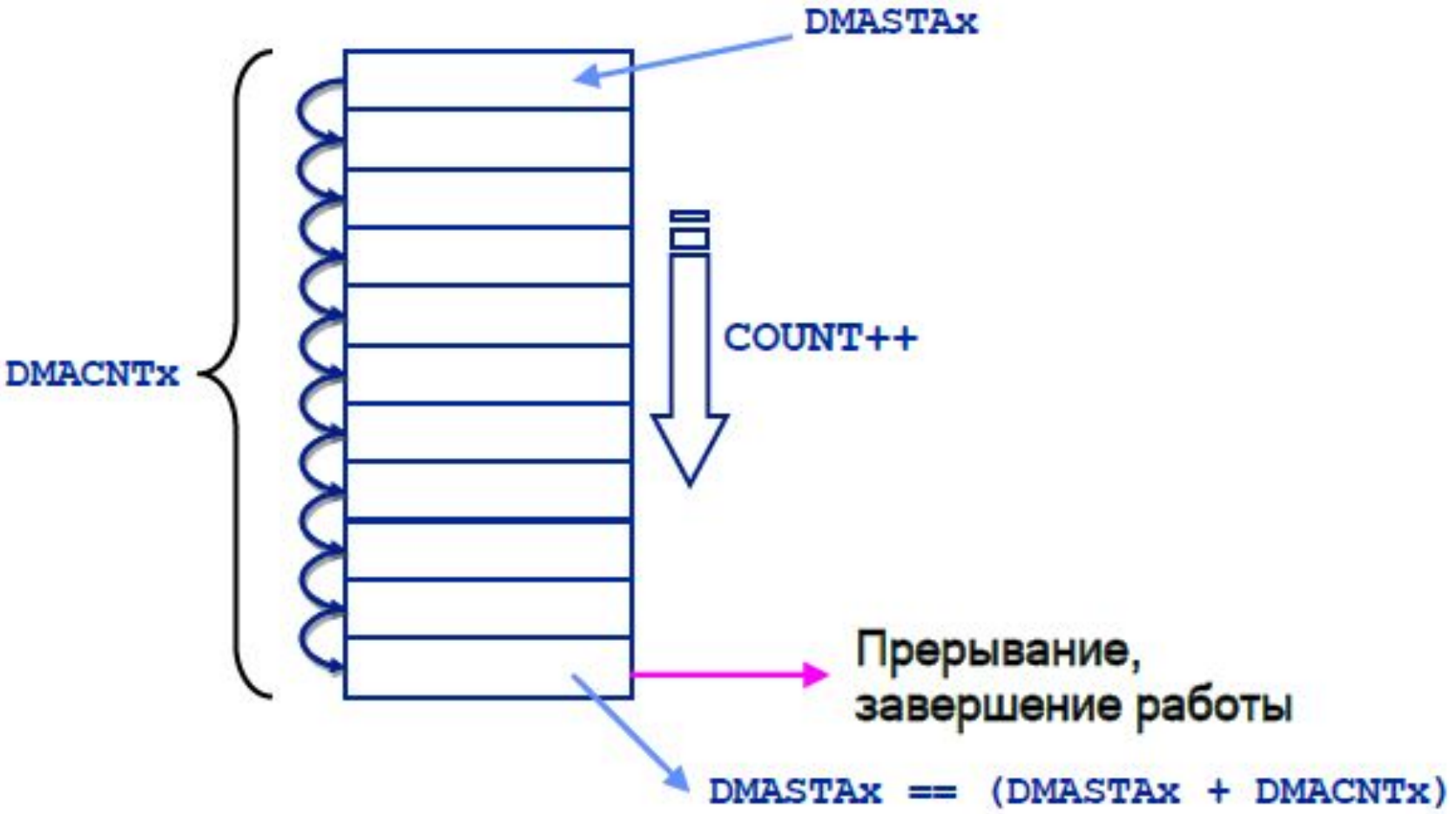


Режимы работы DMA

- **Однократная передача**
  - канал DMA необходимо переконфигурировать для продолжения обмена
- **Автоматический повтор**
  - возврат указателя на начало буфера DMA
- **Автоматический повтор с половинным заполнением**
  - прерывание возникает при половинном заполнении буфера
- **Режим «ping-pong»**
  - используется два буфера
- **Режим косвенной адресации периферии**
  - периферийный модуль формирует смещение для указателя на буфер DMA
- **Режим дополнения**
  - Автоматическая передача после приема (для SPI)

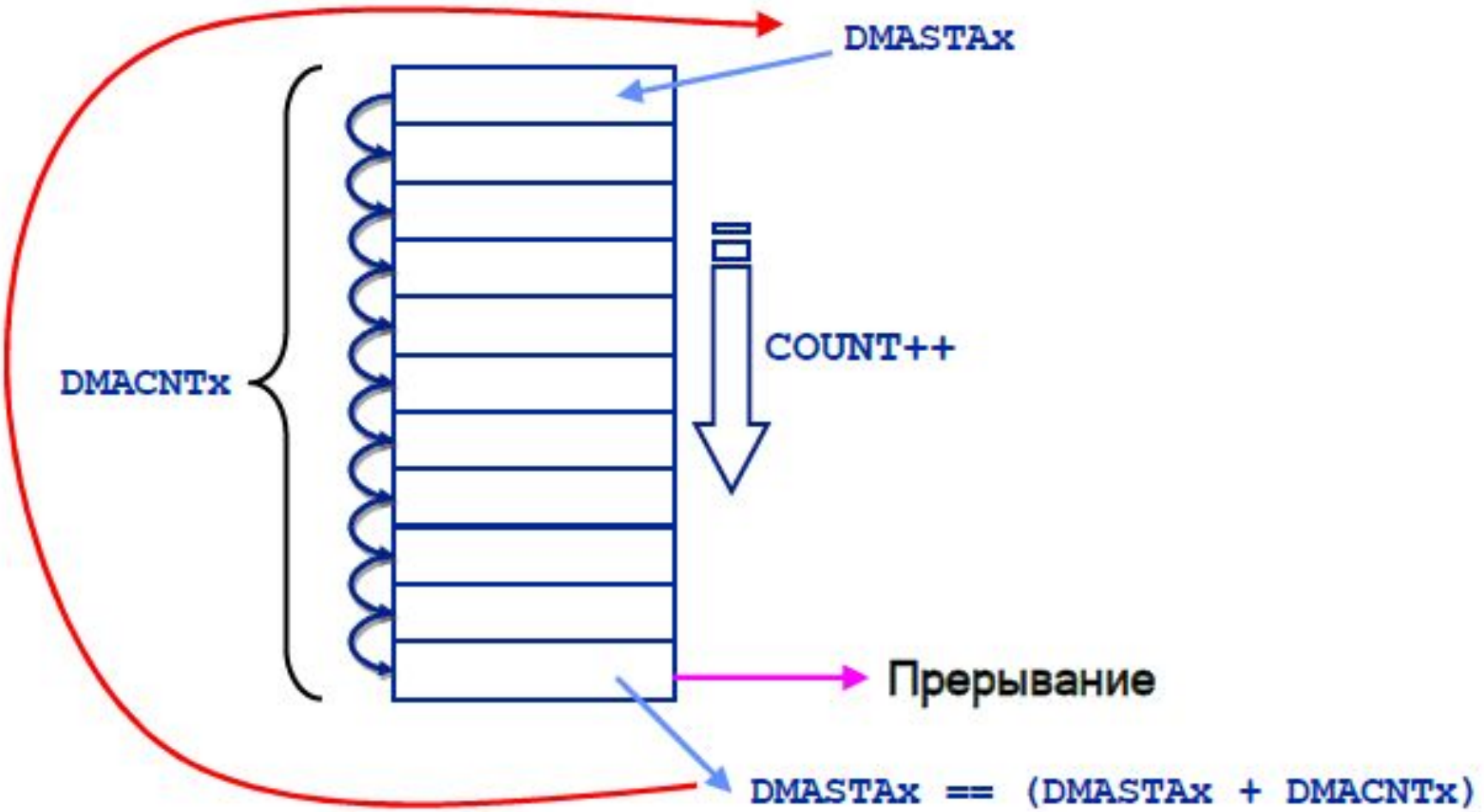
Режимы работы DMA

Однократная передача



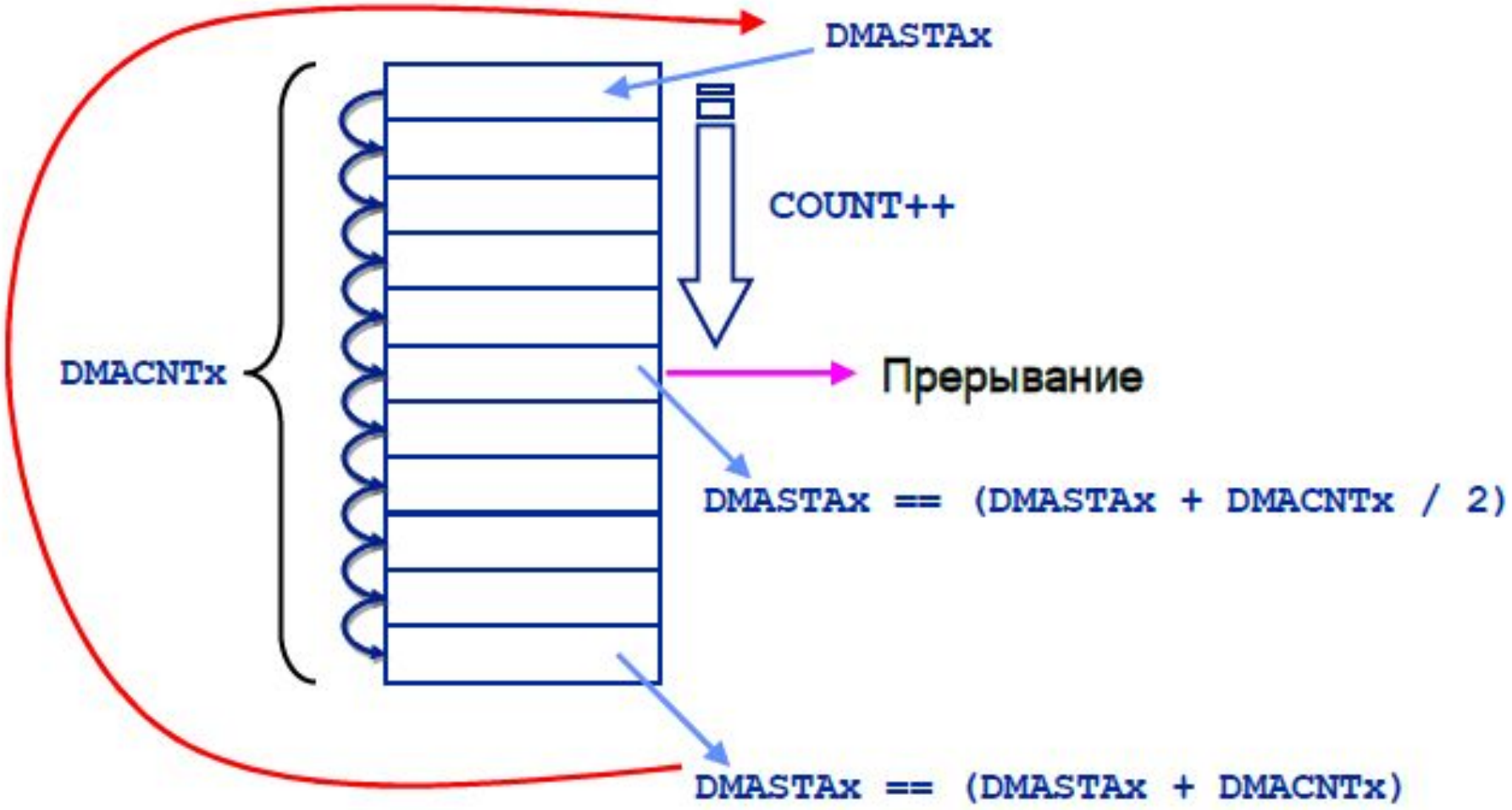
Режимы работы DMA

Автоматический повтор



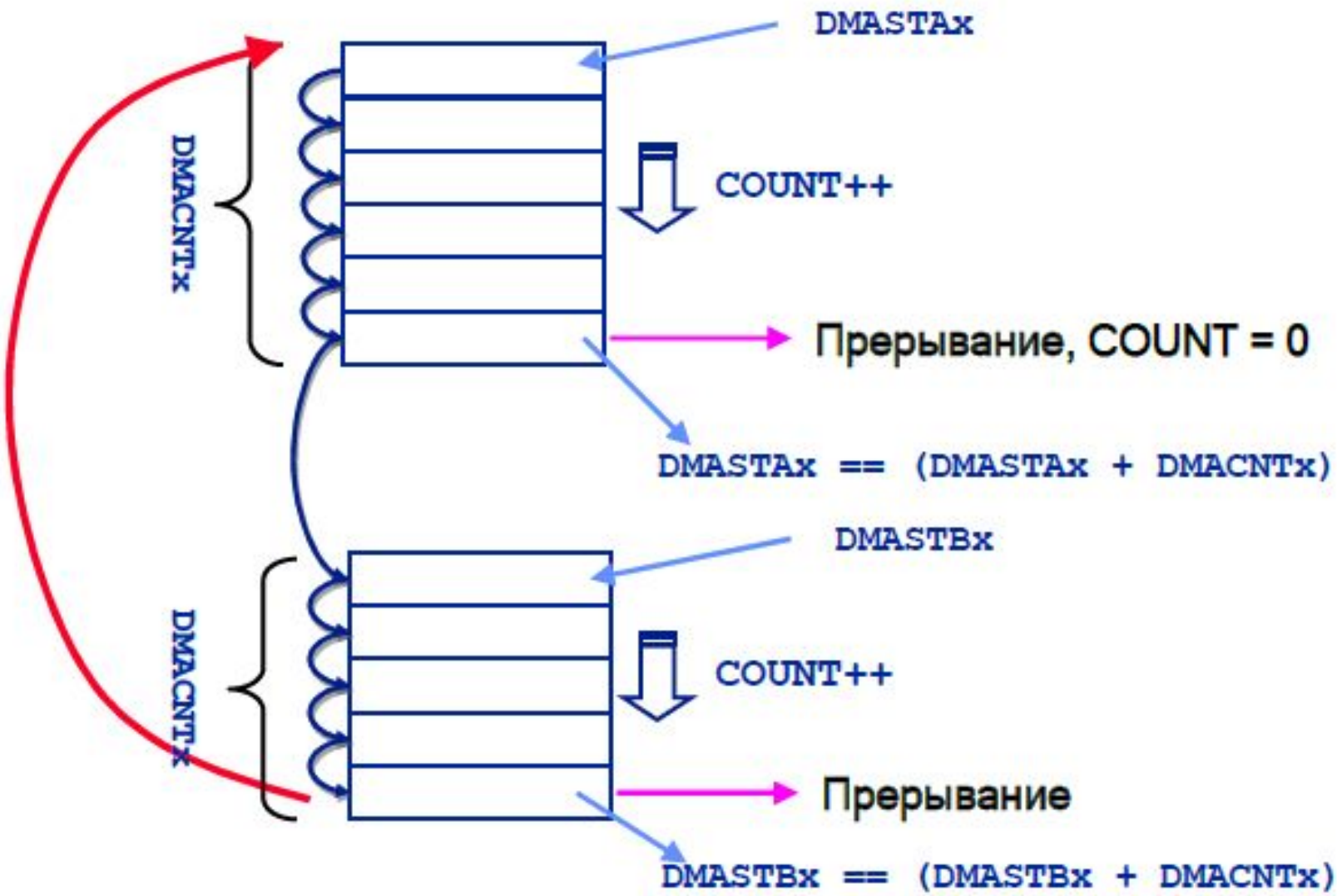
Режимы работы DMA

Автоматический повтор с половинным заполнением



Режимы работы DMA

Режим Ping-Pong



DMAxCON: DMA CHANNEL x CONTROL REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	U-0	U-0
CHEN	SIZE	DIR	HALF	NULLW	—	—	—
bit 15							bit 8

U-0	U-0	R/W-0	R/W-0	U-0	U-0	R/W-0	R/W-0
—	—	AMODE<1:0>		—	—	MODE<1:0>	
bit 7							bit 0

bit 15 **CHEN:** Channel Enable bit
1 = Channel enabled
0 = Channel disabled

bit 14 **SIZE:** Data Transfer Size bit
1 = Byte
0 = Word

bit 13 **DIR**: Transfer Direction bit (source/destination bus select)
1 = Read from DMA RAM address, write to peripheral address
0 = Read from peripheral address, write to DMA RAM address

bit 12 **HALF:** Early Block Transfer Complete Interrupt Select bit
1 = Initiate block transfer complete interrupt when half of the data has been moved
0 = Initiate block transfer complete interrupt when all of the data has been moved

bit 11 **NULLW:** Null Data Peripheral Write Mode Select bit
1 = Null data write to peripheral in addition to DMA RAM write (DIR bit must also be clear)
0 = Normal operation

bit 10-6 **Unimplemented:** Read as '0'

bit 5-4 **AMODE<1:0>:** DMA Channel Operating Mode Select bits
11 = Reserved
10 = Peripheral Indirect Addressing mode
01 = Register Indirect without Post-Increment mode
00 = Register Indirect with Post-Increment mode

bit 3-2 **Unimplemented:** Read as '0'

bit 1-0 **MODE<1:0>:** DMA Channel Operating Mode Select bits
11 = One-Shot, Ping-Pong modes enabled (one block transfer from/to each DMA RAM buffer)
10 = Continuous, Ping-Pong modes enabled
01 = One-Shot, Ping-Pong modes disabled
00 = Continuous, Ping-Pong modes disabled

PERIPHERALS WITH DMA SUPPORT

Peripheral	IRQ Number
INT0	0
Input Capture 1	1
Input Capture 2	5
Output Compare 1	2
Output Compare 2	6
Timer2	7
Timer3	8
SPI1	10
SPI2	33
UART1 Reception	11
UART1 Transmission	12
UART2 Reception	30
UART2 Transmission	31
ADC1	13
ADC2	21
DCI	60
ECAN1 Reception	34
ECAN1 Transmission	70
ECAN2 Reception	55
ECAN2 Transmission	71

Лабораторная работа 6

//Структура для обработчика sOCPWMHandle

```
typedef struct sOCPWMHandle {
int * buffer1;          /* указатель на Ping Pong buffer 1 */
int * buffer2;          /* указатель на Ping Pong buffer 2 */
volatile int bufferIndicator;  /* индикатор активности буферов ping pong */
volatile int isWriteBusy;      /* индикатор, что буферы заняты */
int currentFrameSize;          /* размер текущего кадра      */
int newFrameSize;       /* размер следующего кадра */
int currentSampleIndex;     /* индекс текущей выборки */
}OCPWMHandle;
```

// Функции API

```
void OCPWMInit (OCPWMHandle * pHandle,int * pBufferInDMA);
       OCPWMInit  (pOCPWMHandle, ocPWMBuffer);
                                  // инициализация OCPWM
void OCPWMStart (OCPWMHandle * pHandle);
        OCPWMStart  (pOCPWMHandle);        // запуск OCPWM

void OCPWMWrite (OCPWMHandle * pHandle,int *buffer,int size, unsigned
char gain);
       OCPWMWrite  (pOCPWMHandle,composite,FRAME_SIZE, gain);

int  OCPWMIsBusy (OCPWMHandle * pHandle);
     (OCPWMIsBusy(pOCPWMHandle)

void OCPWMStop (OCPWMHandle * pHandle);  // не используется

void __attribute__((__interrupt__,no_auto_psv)) _DMA1Interrupt(void)
                        //обработка прерывания от канала DMA1
```

/* Функция инициализации DMA1 и ШИМ */

```
void OCPWMInit  (OCPWMHandle * pHandle,int * pBufferInDMA)
{     thisOCPWM = pHandle;
      pHandle->buffer1 = pBufferInDMA;  /* Указатели буферов "пинг-понг" */
      pHandle->buffer2 =(int *) (pBufferInDMA) +  OCPWM_FRAME_SIZE;

      DMA1CON = 0x2002; /* Word transfers  */
                        /* From DMA to OC1RS*/
                        /* Interrupt when all the data has been moved*/
                        /* No NULL writes - Normal Operation*/
                        /* Register Indirect with post-increment mode*/
                        /* Continuous ping pong mode*/
      DMA1REQ= 0x7;        /* Timer 2 Interrupt*/

      DMA1STA = (int)(pHandle->buffer1) - (int)&_DMA_BASE;
      DMA1STB = (int)(pHandle->buffer2) - (int)&_DMA_BASE;        DMA1PAD = (int
)&OC1RS;
      DMA1CNT = OCPWM_FRAME_SIZE - 1;

      T2CON  = 0;
      TMR2    = 0;
      PR2= 0;
}
```

// Функция включения модуля OCPWM

```
void OCPWMStart      (OCPWMHandle * pHandle)
{
      pHandle-> bufferIndicator= 0;
      pHandle-> currentSampleIndex = 0;
      pHandle-> currentFrameSize = OCPWM_FRAME_SIZE;
      pHandle-> newFrameSize = OCPWM_FRAME_SIZE;
      pHandle-> isWriteBusy      = 0;
      _DMA1IF             = 0;
      _DMA1IE             = 1;
      DMA1CONbits.CHEN = 1; /* Enable the DMA1 Channel */

      PR2 = OCPWM_MAX_PWM_PERIOD;  /* PWM Period     */
      OC1RS  = ((OCPWM_MAX_PWM_PERIOD)/2);/* Initial Duty Cycle at 50% */
      OC1R    = ((OCPWM_MAX_PWM_PERIOD)/2);
      OC1CON= OCPWM_OCCON_WORD;  /* Turn module on*/
      T2CONbits.TON  = 1;              /* Enable Timer2  */
}
```

// Функция вывода ШИМ сигнала

```
void OCPWMWrite (OCPWMHandle * pHandle,int *data,int size, unsigned char gain)
{
  int *dest;                    // указатель на буфер
  int   i;
  unsigned int sample;
  unsigned int pwmDutyCycle;
  /* if the buffer indicator bit is 1, then write buffer 1 else use buffer2  */
  dest = (pHandle->bufferIndicator) ? pHandle->buffer1 : pHandle->buffer2;
  if (size >OCPWM_FRAME_SIZE)  { size = OCPWM_FRAME_SIZE; }
  for(i = 0; i < size ; i++) { /* Compute Duty cycle values for every input sample      */
     sample = data[i] - (OCPWM_LOWEST_INPUT_VALUE);
     pwmDutyCycle = ((sample * OCPWM_MAX_PWM_PERIOD))
                    /OCPWM_INPUT_RANGE;
      if ( pwmDutyCycle > OCPWM_MAX_PWM_PERIOD ) {
           pwmDutyCycle = OCPWM_MAX_PWM_PERIOD - 1;  }
           if ( pwmDutyCycle <= 0) {  pwmDutyCycle = 1; }
           dest[i] = (pwmDutyCycle*gain)/16;
      }
      pHandle->newFrameSize = size;     /* Update the frame size*/
      __asm__ volatile("disi #0x4");                /* disable Interrupts */
      pHandle->isWriteBusy = 1;
      __asm__ volatile("disi #0x0");         /* enable Interrupts */
}
```

// Функция анализа состояния канала вывода

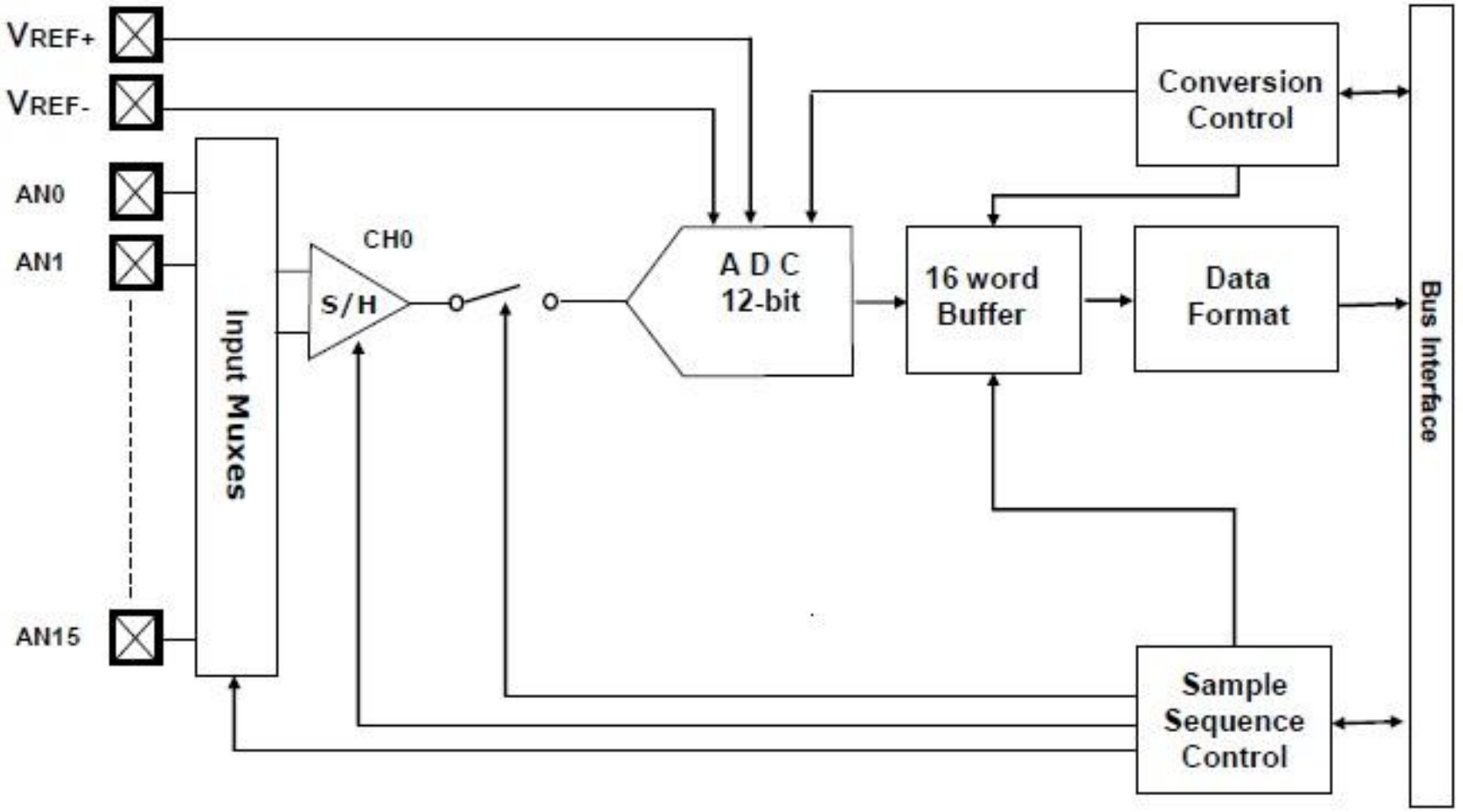
```
int OCPWMIsBusy (OCPWMHandle * pHandle)
{
   return(pHandle->isWriteBusy);
}
```

//Обработчик прерывания DMA1

```
int  value = 0;
void __attribute__((__interrupt__,no_auto_psv)) _DMA1Interrupt(void)
{
      _DMA1IF = 0;
      thisOCPWM->bufferIndicator ^= 1;   /* Flip the indicator bit */
      thisOCPWM->isWriteBusy  = 0;/* New frame is available    */

}
```

```
// Главная функция программы
int main(void)
{
    /* Операторы настройки тактового генератора на частоту 40MHz. */
__builtin_write_OSCCONH(0x01);          /*Внутренний FRC с PLL*/
     __builtin_write_OSCCONL(0x01);
     while (OSCCONbits.COSC != 0b01);/*Ожидание переключения и захвата*/
     while(!OSCCONbits.LOCK);
     OCPWMInit    (pOCPWMHandle,ocPWMBuffer);    // инициализация OCPWM
     OCPWMStart (pOCPWMHandle);     // запуск OCPWM
     SASKInit();                          //инициализация портов
     gain = 16;

     while(1)                         //основной цикл для каждого кадра
      { /* Ожидание доступности ОС для нового кадра*/
        while (OCPWMIsBusy(pOCPWMHandle));
        /* Запись кадра на вывод*/
         OCPWMWrite (pOCPWMHandle, composite, FRAME_SIZE, gain);
               if((CheckSwitchS1()) == 1)     //коррекция усиления
             { if (gain < 17)  gain = ++gain; }
         if((CheckSwitchS2()) == 1)
             { if (gain > 0)    gain = --gain;   }
       }
}
```

Аналого-цифровой преобразователь (режим 12 разрядов)



ADxCON1: ADCx CONTROL REGISTER 1

R/W-0	U-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0	R/W-0
ADON	—	ADSIDL	ADDMABM	—	AD12B	FORM<1:0>	
bit 15							bit 8

bit 15 **ADON:** ADC Operating Mode bit

1 = ADC module is operating
0 = ADC is off

bit 14 **Unimplemented:** Read as '0'

bit 13 **ADSIDL:** Stop in Idle Mode bit

1 = Discontinue module operation when device enters Idle mode
0 = Continue module operation in Idle mode

bit 12 **ADDMABM:** DMA Buffer Build Mode bit

1 = DMA buffers are written in the order of conversion. The module will provide an address to the DMA channel that is the same as the address used for the non-DMA stand-alone buffer
0 = DMA buffers are written in Scatter/Gather mode. The module will provide a scatter/gather address to the DMA channel, based on the index of the analog input and the size of the DMA buffer

bit 10 **AD12B:** 10-Bit or 12-Bit Operation Mode bit

1 = 12-bit, 1-channel ADC operation
0 = 10-bit, 4-channel ADC operation

bit 9-8 **FORM<1:0>:** Data Output Format bits

For 10-bit operation:

11 = Signed fractional (DOUT = sddd dddd dd00 0000, where s = .NOT.d<9>)
10 = Fractional (DOUT = dddd dddd dd00 0000)
01 = Signed integer (DOUT = ssss sssd dddd dddd, where s = .NOT.d<9>)
00 = Integer (DOUT = 0000 00dd dddd dddd)

For 12-bit operation:

11 = Signed fractional (DOUT = sddd dddd dddd 0000, where s = .NOT.d<11>)
10 = Fractional (DOUT = dddd dddd dddd 0000)
01 = Signed Integer (DOUT = ssss sddd dddd dddd, where s = .NOT.d<11>)
00 = Integer (DOUT = 0000 dddd dddd dddd)

R/W-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0	R/W-0 HC,HS	R/C-0 HC, HS
	SSRC<2:0>		—	SIMSAM	ASAM	SAMP	DONE
bit 7							bit 0

bit 7-5 **SSRC<2:0>:** Sample Clock Source Select bits

111 = Internal counter ends sampling and starts conversion (auto-convert)
110 101 100 = Reserved
011 = MPWM interval ends sampling and starts conversion
010 = GP timer (Timer3 for ADC1, Timer5 for ADC2) compare ends sampling and starts conversion
001 = Active transition on INT0 pin ends sampling and starts conversion
000 = Clearing sample bit ends sampling and starts conversion

bit 3 **SIMSAM:** Simultaneous Sample Select bit (only applicable when CHPS<1:0> = 01 or 1x)

When AD12B = 1, SIMSAM is: U-0, Unimplemented, Read as '0'

1 = Samples CH0, CH1, CH2, CH3 simultaneously (when CHPS<1:0> = 1x); or
Samples CH0 and CH1 simultaneously (when CHPS<1:0> = 01)
0 = Samples multiple channels individually in sequence

bit 2 **ASAM:** ADC Sample Auto-Start bit

1 = Sampling begins immediately after last conversion. SAMP bit is auto-set
0 = Sampling begins when SAMP bit is set

bit 1 **SAMP:** ADC Sample Enable bit

1 = ADC sample/hold amplifiers are sampling
0 = ADC sample/hold amplifiers are holding
If ASAM = 0, software may write '1' to begin sampling. Automatically set by hardware if ASAM = 1.
If SSRC = 000, software may write '0' to end sampling and start conversion. If SSRC ≠ 000, automatically cleared by hardware to end sampling and start conversion.

bit 0 **DONE:** ADC Conversion Status bit

1 = ADC conversion cycle is completed
0 = ADC conversion not started or in progress

Лабораторная работа 7

//Константы

```
#define ADC_CHANNEL_FCY  40000000
#define ADC_FSAMP        8000      /* Sampling Frequency       */
#define ADC_BUFFER_SIZE    128 /* This is the size of each buffer*/
#define ADC_CHANNEL_DMA_BUFSIZE (ADC_BUFFER_SIZE*2)
```

//Структура обработчика прерывания от модуля АЦП

```
typedef struct sADCChannelHandle {
      int * buffer1;
      int * buffer2;
      volatile int bufferIndicator;
      volatile int isReadBusy;

}ADCChannelHandle;
```

// Функции API

```
void ADCChannelInit   (ADCChannelHandle * pHandle,int * pBufferInDMA);
void ADCChannelStart      (ADCChannelHandle * pHandle);
void ADCChannelRead       (ADCChannelHandle * pHandle,int *buffer,int size);
int ADCChannelIsBusy      (ADCChannelHandle * pHandle);
void ADCChannelStop(ADCChannelHandle * pHandle);
```

// Константы настройки модуля АЦП

```
#define ADCON1VAL  0x0744     /* 12 bit ADC with signed fractional format
                              * Triggered by Timer 3 and auto start
                              * sampling after conversion complete. */
#define ADCON2VAL  0x0000     /* AVdd and AVss voltage reference,
                               * use channel 0 with no scan */
#define ADCON3VAL  0x0010     /* Tad is 16 Tcy                */
#define ADCHSVAL    0x0000    /* AN0 input on channel 0       */
#define ADPCFGVAL  0xFFFE     /* AN0 input is Analog          */
#define ADCSSLVAL   0x0000    /* No channel scanning          */
```

// Функция инициализации модуля АЦП и канала DMA

```
static ADCChannelHandle * thisADCChannel;

void ADCChannelInit   (ADCChannelHandle * pHandle,int * pBufferInDMA)
{
      /* This function will intialize the DMA              */
      /* DMA0 is used to read from the ADC          */
      thisADCChannel = pHandle;
      pHandle->buffer1 = pBufferInDMA;
                        /* Assign the ping pong buffers for the ADC DMA*/
      pHandle->buffer2 = (int *)((int)pBufferInDMA + ADC_BUFFER_SIZE);

      DMA0CONbits.SIZE  = 0;  /* Word transfers */
      DMA0CONbits.DIR    = 0;  /* From ADC1BUF to DMA  */
      DMA0CONbits.HALF = 0; /* Interrupt when all the data has been moved */
      DMA0CONbits.NULLW  = 0; /* No NULL writes - Normal Operation  */
      DMA0CONbits.AMODE  = 0; /* Register Indirect with post-increment mode */
      DMA0CONbits.MODE    = 2; /* Continuous  ping pong mode enabled */
      DMA0REQbits.FORCE  = 0; /* Automatic transfer  */
      DMA0REQbits.IRQSEL = 0xD; /* ADC conversion complete */
```

```
    DMA0STA  = (int)(pHandle->buffer1) - (int)&_DMA_BASE;
    DMA0STB = (int)(pHandle->buffer2) - (int)&_DMA_BASE;

    DMA0PAD  = (int )&ADC1BUF0;
    DMA0CNT  = ADC_BUFFER_SIZE - 1;

    AD1CON1  = ADCON1VAL;    /* Load the ADC registers with  value */
    AD1CON2  = ADCON2VAL;    /* specified in 12bitADCDriver.h  */
    AD1CON3  = ADCON3VAL;
    AD1CHS0  = ADCHSVAL;
    AD1PCFGLbits.PCFG0 = 0;
    AD1CSSL  = ADCSSLVAL;

    TMR3        = 0;
    PR3    = (ADC_CHANNEL_FCY/ADC_FSAMP) - 1;

}
```

// Функция включения модуля АЦП

```
void ADCChannelStart      (ADCChannelHandle * pHandle)
{
      pHandle->bufferIndicator   = 0;
      pHandle->isReadBusy        = 1;
      _DMA0IF = 0;
      _DMA0IE = 1;
      DMA0CONbits.CHEN = 1;         /* Enable the DMA Channel      */
      AD1CON1bits.ADON = 1;         /* Enable ADC module           */
      T3CONbits.TON    = 1;       /* Enable Timer 3              */
}
```

// Функция выключения модуля АЦП

```
void ADCChannelStop(ADCChannelHandle * pHandle)
{
      _DMA0IE = 0;                  /* Disable the DMA interrupt   */
      DMA0CONbits.CHEN = 0;         /* Disable the DMA Channel     */
      AD1CON1bits.ADON = 0;         /* Disable ADC module          */
      T3CONbits.TON    = 0;       /* Disable Timer 3             */
}
```

// Функция аналогового ввода

```
void ADCChannelRead      (ADCChannelHandle * pHandle,int *data,int size)
{
      int   *source;
      int   i;

      /* if the buffer indicator bit is 1, then use buffer 1 else use buffer2    */
      /* Since the DMA ping pongs between these buffer, you must know */
      /* which one to read. The bufferIndicators keep track of this              */

      source = (pHandle->bufferIndicator) ? pHandle->buffer1 : pHandle->buffer2;
      if (size > ADC_BUFFER_SIZE) size = ADC_BUFFER_SIZE;

      for(i = 0; i < size; i++)
      {
          data[i] = source[i];
      }
      __asm__ volatile("disi #0x4"); /* disable interrupts */
      pHandle->isReadBusy = 1;
      __asm__ volatile("disi #0x0"); /* enable interrupts */

}
```

// Функция анализа состояния канала ввода

```
int ADCChannelIsBusy       (ADCChannelHandle * pHandle)
{
      return(pHandle->isReadBusy);
}
```

// Функция обработчика прерывания от канала DMA

```
void __attribute__((__interrupt__,no_auto_psv)) _DMA0Interrupt(void)
{

      _DMA0IF = 0;
      thisADCChannel->bufferIndicator     ^= 1;       /* Flip the indicator bit  */
      thisADCChannel->isReadBusy    = 0;       /* New frame is available */
}
```

// Основной модуль программы

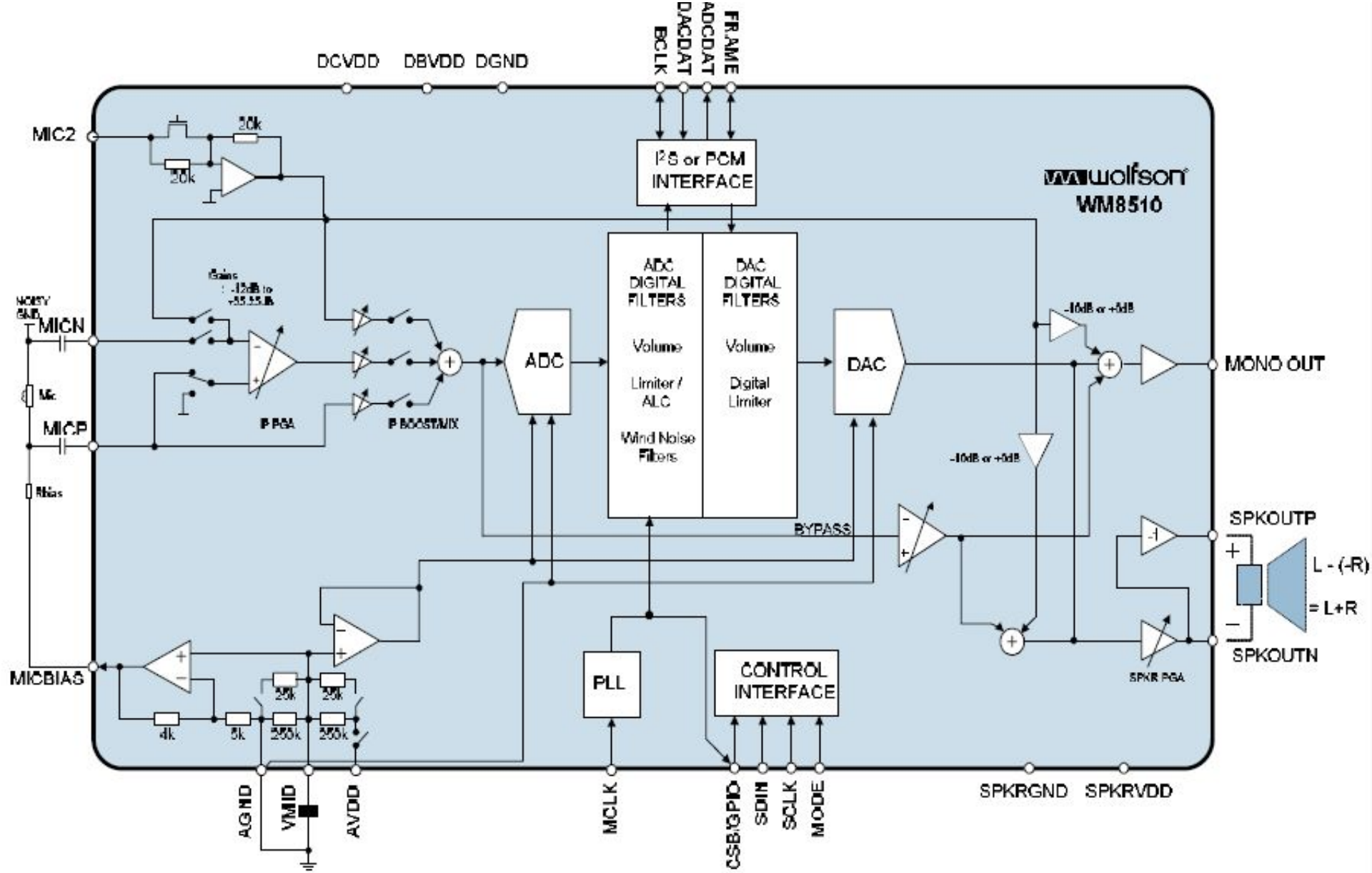
```
_FGS(GWRP_OFF & GCP_OFF);
_FOSCSEL(FNOSC_FRC);
_FOSC(FCKSM_CSECMD & OSCIOFNC_ON & POSCMD_NONE);
_FWDT(FWDTEN_OFF);

#define FRAME_SIZE        128        // длина кадра
#define BLOCK_LENGTH    128             //длина блока

/* Выделение памяти для буферов, переменных и драйверов     */
int   adcBuffer   [ADC_CHANNEL_DMA_BUFSIZE]     __attribute__((space(dma)));
int   ocPWMBuffer  [OCPWM_DMA_BUFSIZE]     __attribute__((space(dma)));
int   samples    [FRAME_SIZE];

ADCChannelHandle adcChannelHandle;       // объявление структуры драйверов
OCPWMHandle    ocPWMHandle;
ADCChannelHandle *pADCChannelHandle = &adcChannelHandle;
                              //указатели на структуры
OCPWMHandle    *pOCPWMHandle  = &ocPWMHandle;

extern FIRStruct LowPassFilter;
int FilterOut[BLOCK_LENGTH];     /*Output array where filtered output will be stored */
```

```
// Главная функция программы
int main(void)
{
    /* Настройка тактового генератора на частоту 40MHz.
     * Fosc= Fin*M/(N1*N2), Fcy=Fosc/2
     * Fosc= 7.37M*43/(2*2)=80Mhz for 7.37M input clock */

    PLLFBD=41;              /* M=41+2*/
    CLKDIVbits.PLLPOST=0;       /* N1=2  */
    CLKDIVbits.PLLPRE=0;        /* N2=2  */
    OSCTUN=0;               //центральная частота 7.37МГц

    __builtin_write_OSCCONH(0x01);      /*Внутренний FRC с PLL*/
    __builtin_write_OSCCONL(0x01);
        while (OSCCONbits.COSC != 0b01); /*Ожидание переключения и захвата*/
        while(!OSCCONbits.LOCK);

    ADCChannelInit   (pADCChannelHandle,adcBuffer);   //инициализация АЦП
    OCPWMInit   (pOCPWMHandle,ocPWMBuffer);   //иниц.  OCPWM
    ADCChannelStart(pADCChannelHandle);           // запуск АЦП
    OCPWMStart      (pOCPWMHandle);           // запуск OCPWM
    FIRDelayInit(&Filter);                    //инициализация фильтра
// основной цикл
}
```

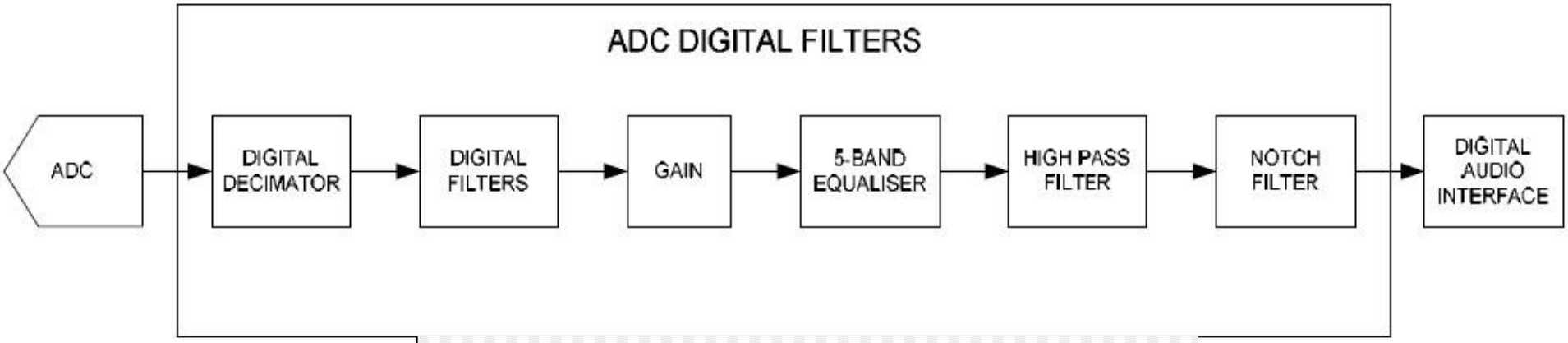
// основной цикл

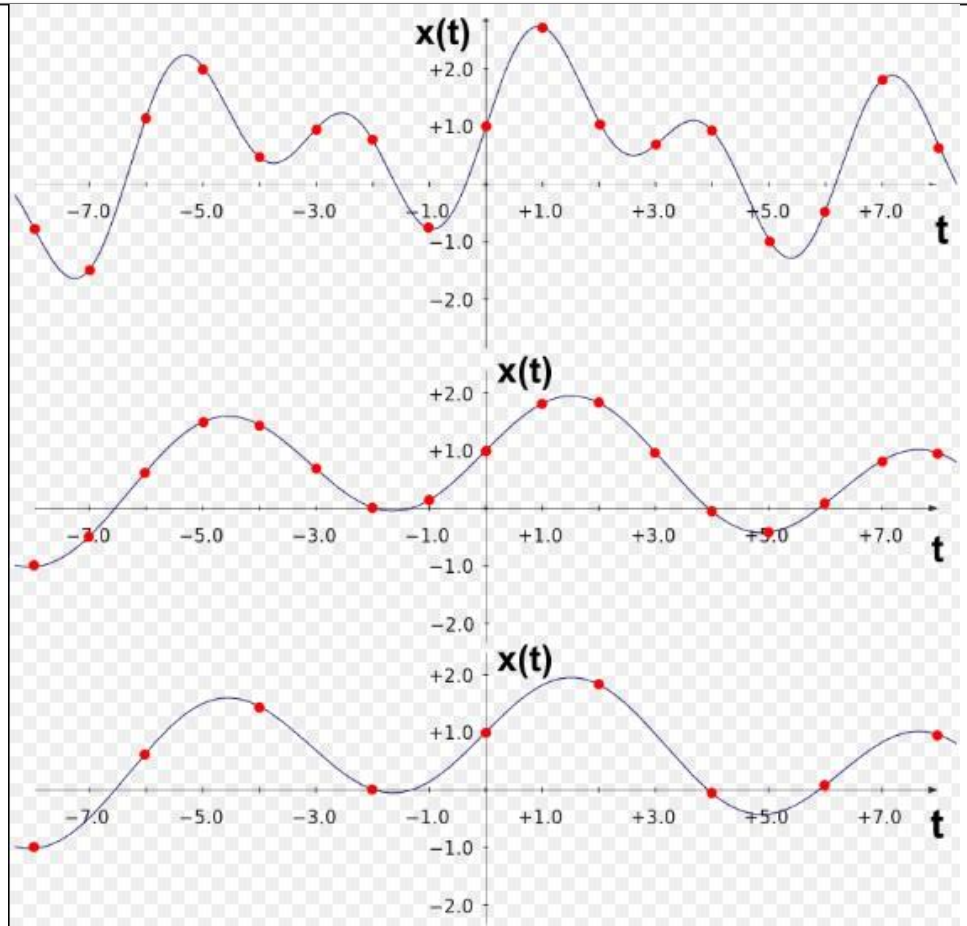
```
while(1)
{
    /* Ожидание доступности канала ввода для нового кадра     */
    while(ADCChannelIsBusy(pADCChannelHandle));

    /* Заполнение массива samples */
    ADCChannelRead (pADCChannelHandle,samples,FRAME_SIZE);

    /* Функция КИХ-фильтра  */
    FIR(BLOCK_LENGTH,&FilterOut[0],&samples[0],&LowPassFilter);

    /* Ожидание доступности ОС для нового кадра*/
    while(OCPWMIsBusy(pOCPWMHandle));

    /* Запись кадра на вывод */
    OCPWMWrite (pOCPWMHandle,FilterOut,FRAME_SIZE);
}
```

Аудиокодек WM8510

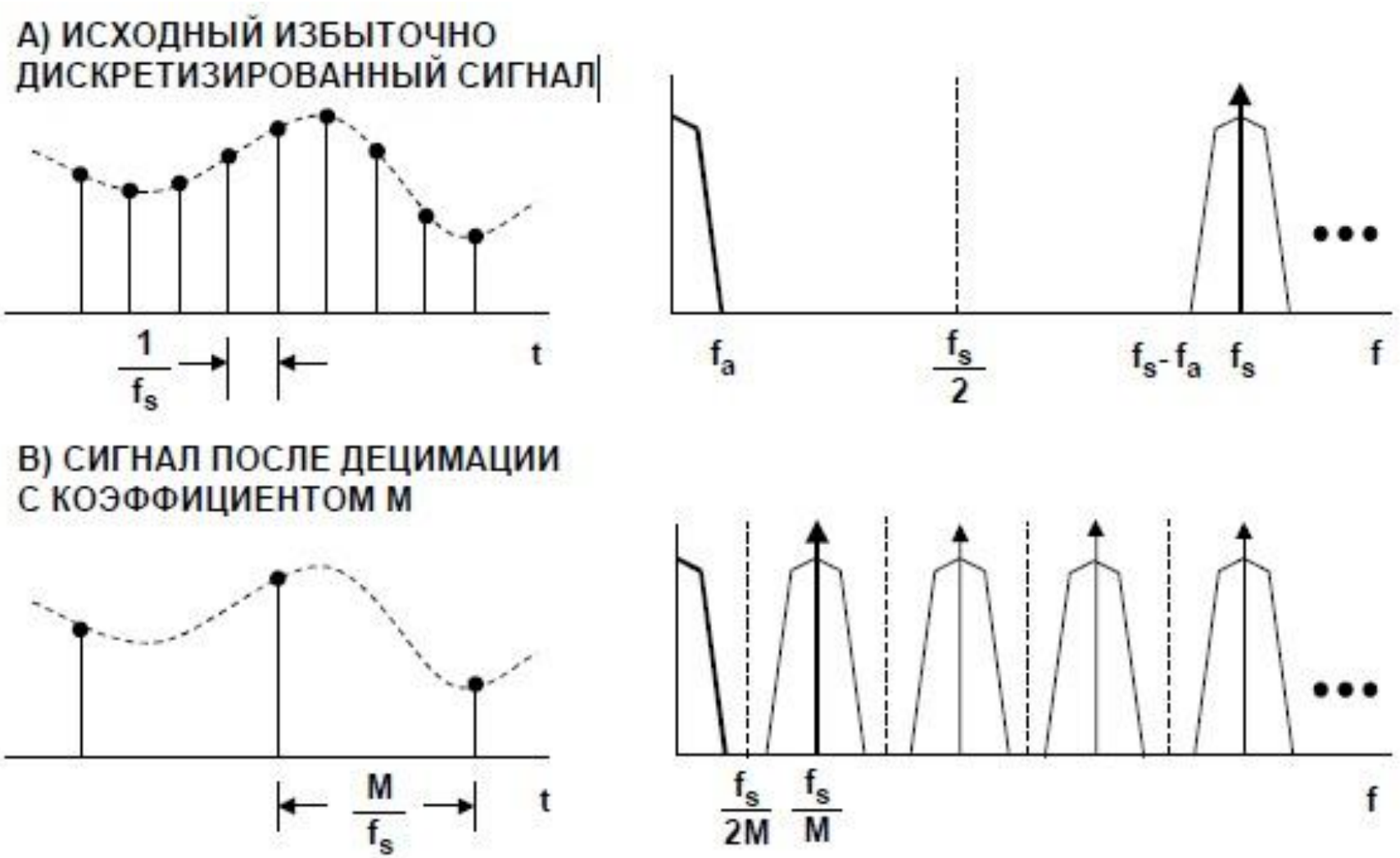


Передискретизация

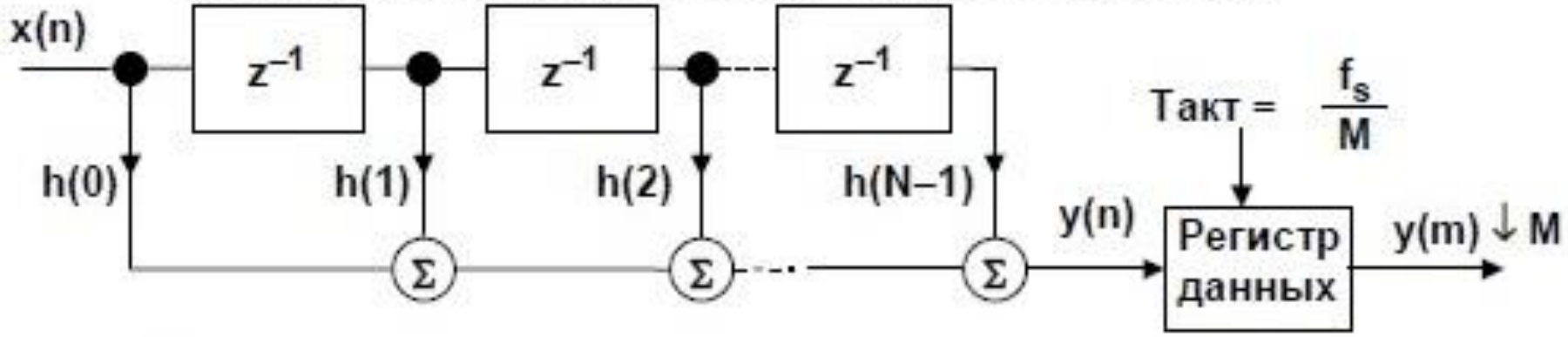




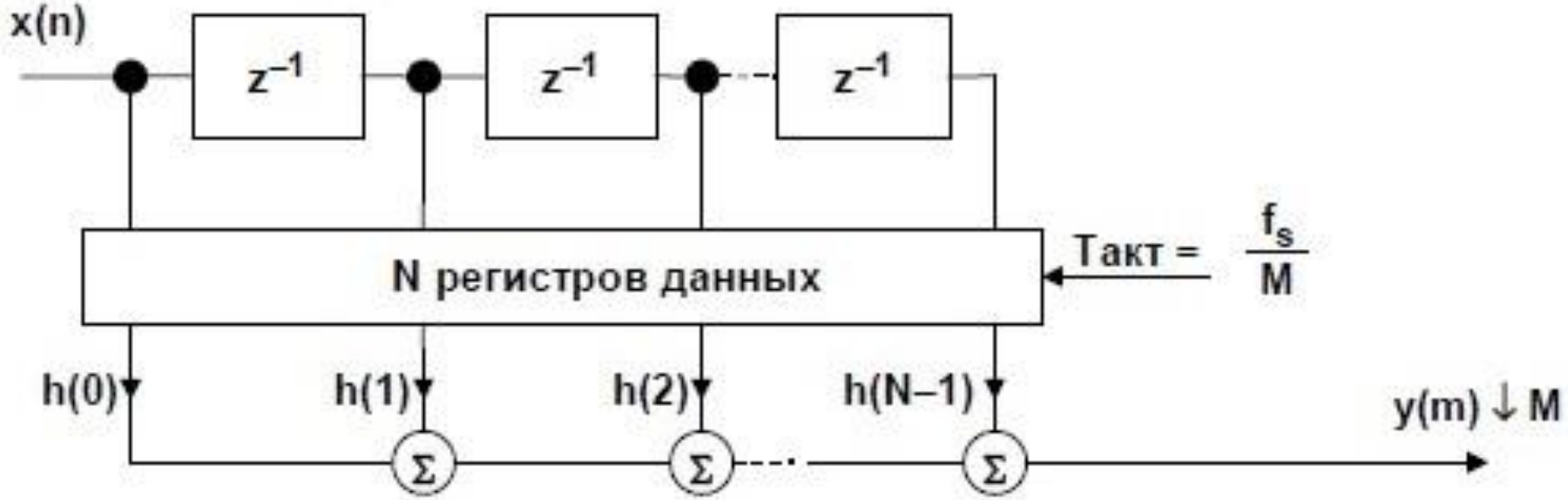
Децимация

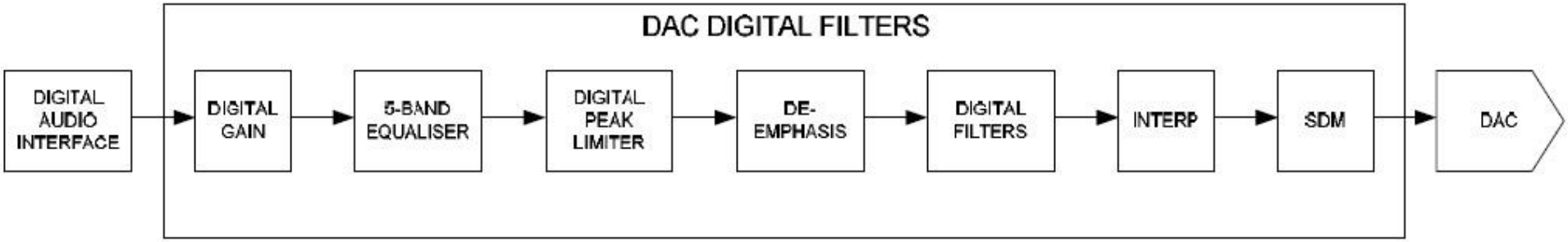

А) ИСХОДНЫЙ ИЗБЫТОЧНО ДИСКРЕТИЗИРОВАННЫЙ СИГНАЛ
1/fs
t
fa
fs/2
fs-fa
fs
f
В) СИГНАЛ ПОСЛЕ ДЕЦИМАЦИИ С КОЭФФИЦИЕНТОМ М
M/fs
t
fs/2M
fs/M
f

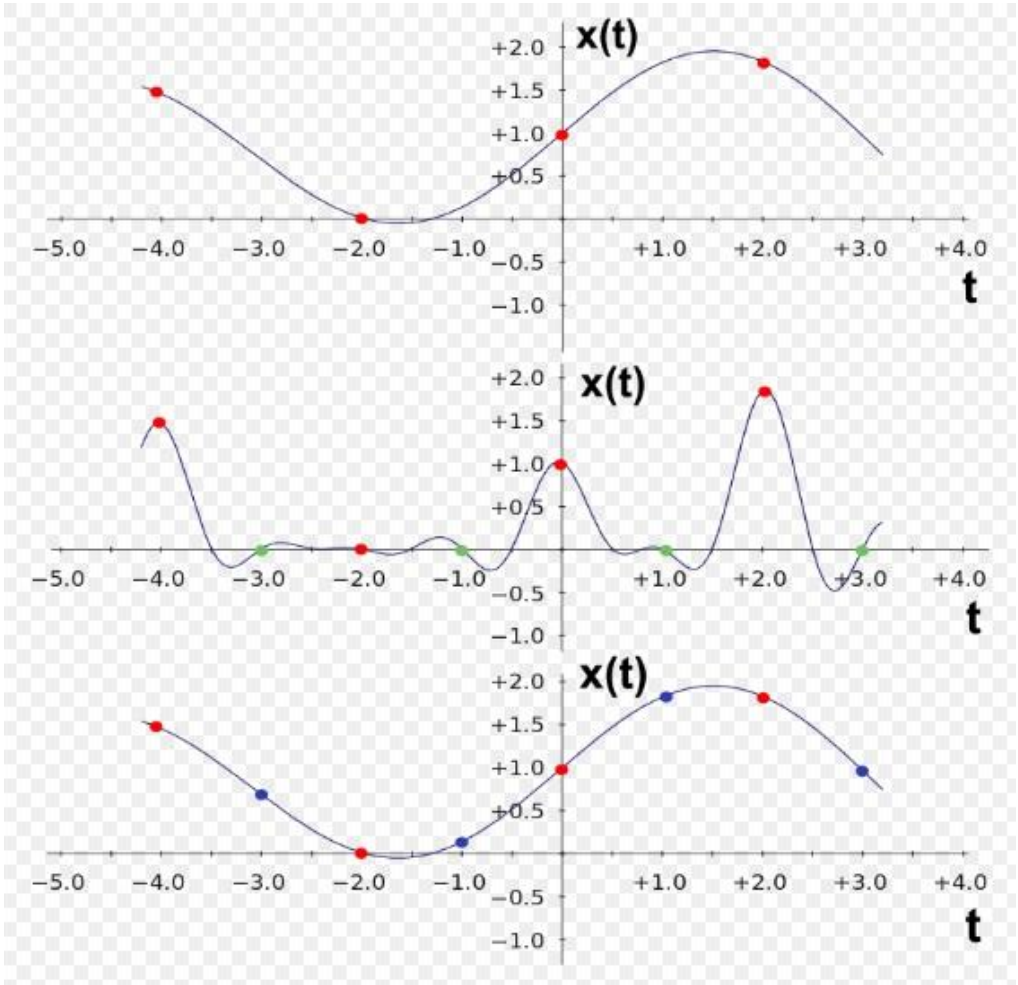
А) нет изменений в требуемом объеме вычислений



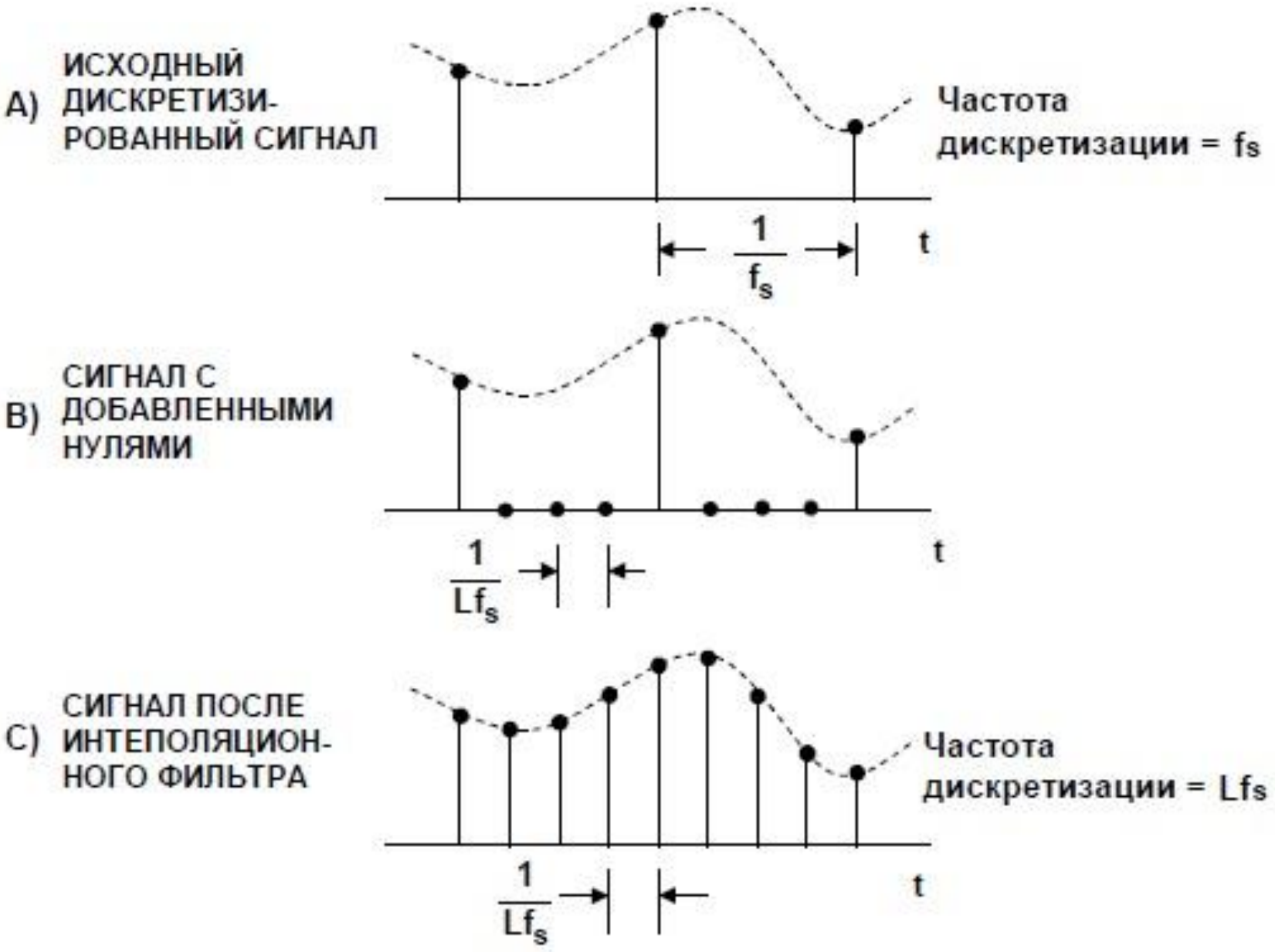
В) требуемый объем вычислений уменьшается пропорционально коэффициенту M



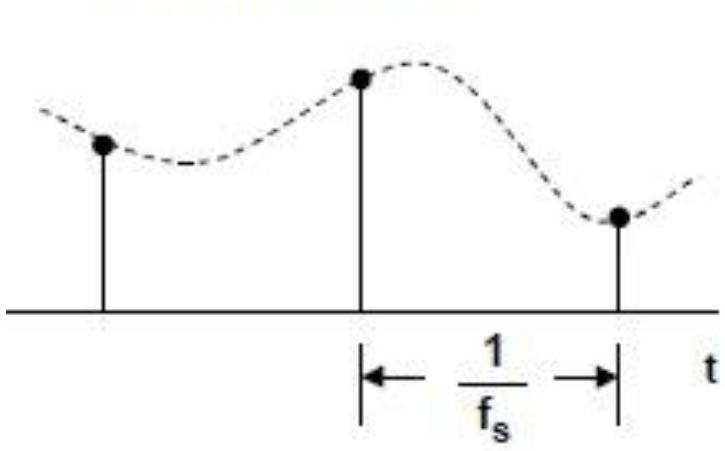

DAC DIGITAL FILTERS
DIGITAL AUDIO INTERFACE
DIGITAL GAIN
5-BAND EQUALISER
DIGITAL PEAK LIMITER
DE-EMPHASIS
DIGITAL FILTERS
INTERP
SDM
DAC

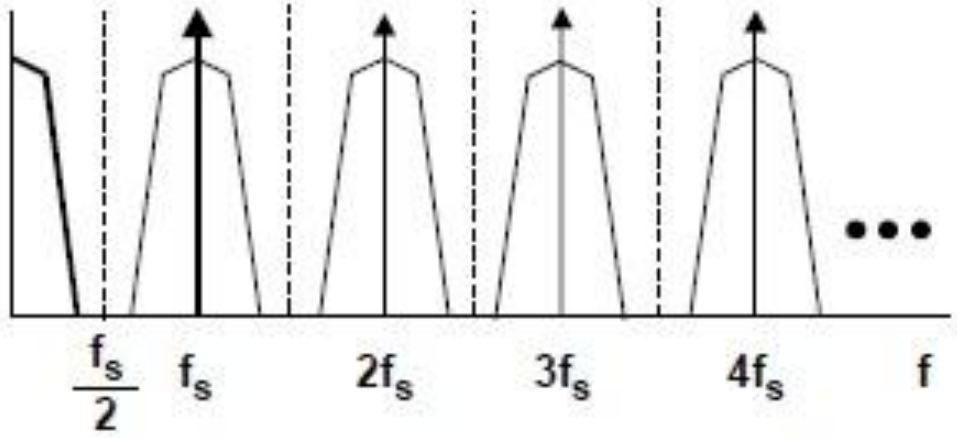

x(t)
+2.0
+1.5
+1.0
+0.5
−0.5
−1.0
−5.0
−4.0
−3.0
−2.0
−1.0
+1.0
+2.0
+3.0
+4.0
t
x(t)
t
x(t)
t

Интерполяция

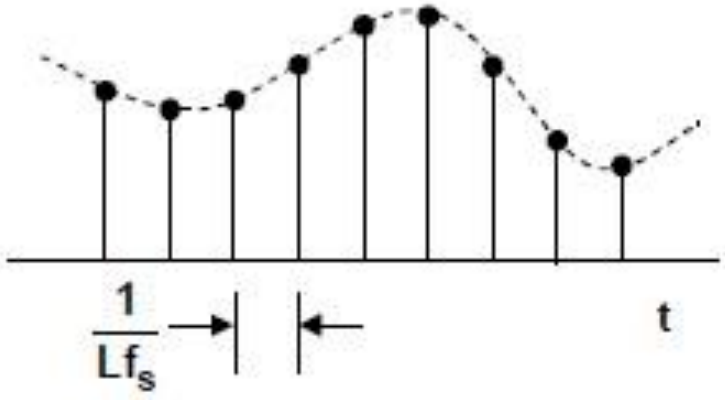


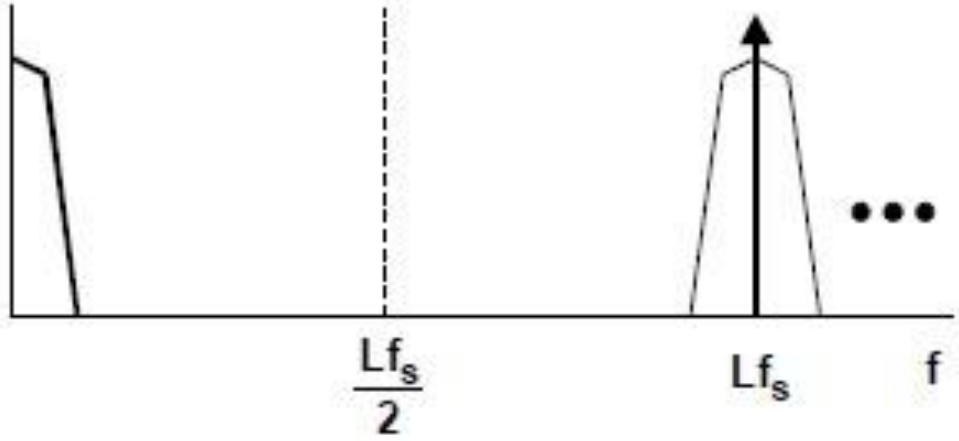
ИСХОДНЫЙ СИГНАЛ

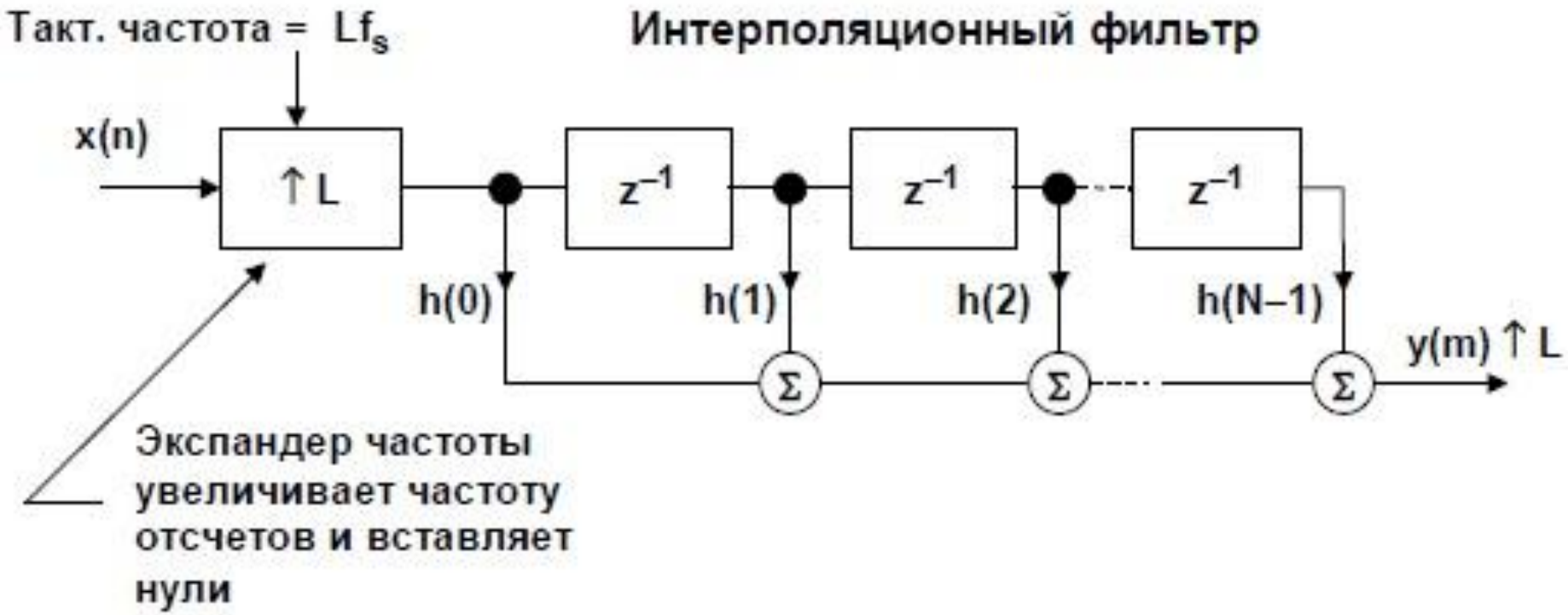




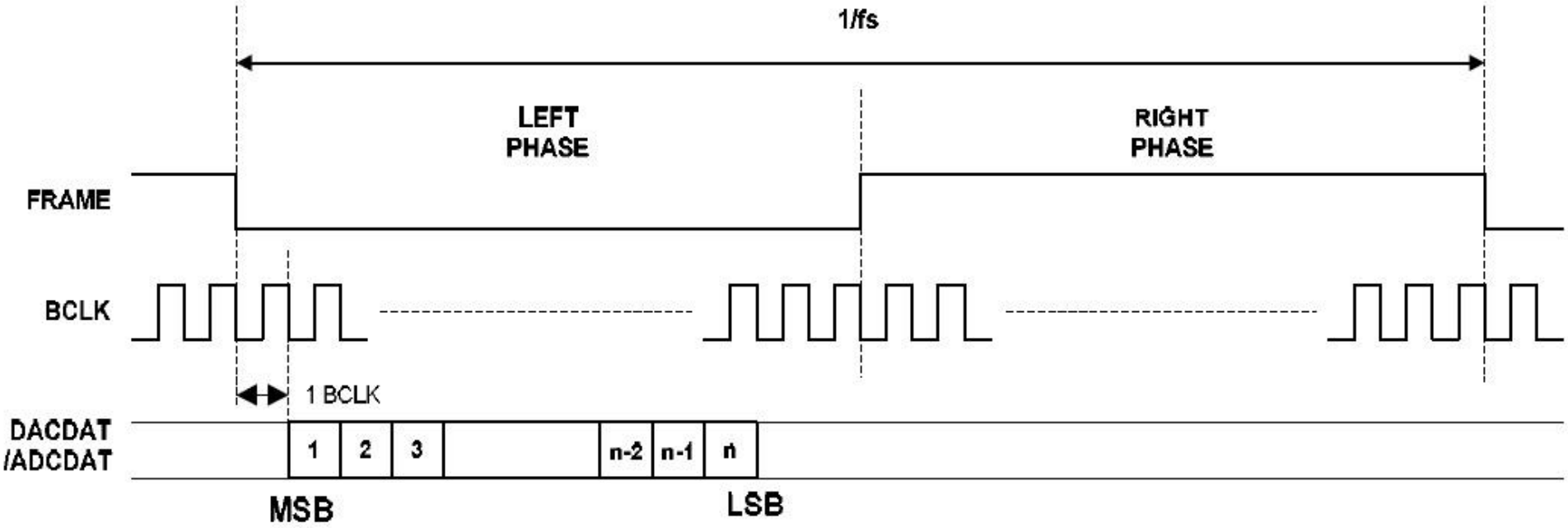
СИГНАЛ ПОСЛЕ ИНТЕРПОЛЯЦИИ С КОЭФФИЦИЕНТОМ L






Такт. частота = Lf_s
Интерполяционный фильтр
x(n)
↑ L
z^{-1}
z^{-1}
z^{-1}
h(0)
h(1)
h(2)
h(N–1)
Σ
Σ
Σ
y(m) ↑ L
Экспандер частоты увеличивает частоту отсчетов и вставляет нули

Интерфейс I2S



```
typedef struct sWM8510Handle
{
  int * inputBuffer1;                    /* Ping Pong Input Buffer 1 */
  int * inputBuffer2;                    /* Ping Pong Input Buffer 2 */
  int * outputBuffer1;                /* Ping Pong Output Buffer 1     */
  int * outputBuffer2;                /* Ping Pong Output Buffer 2     */
  volatile int  currentSampleIndex;          /* Tracks the sample being processed */
  volatile int  currentFrameSize;   /* The size of the current frame being processed - 1*/
  volatile int  newFrameSize;            /* The size of the new frame      */
  volatile int * activeInputBuffer;       /* The active ping pong input buffer  */
  volatile int * activeOutputBuffer;          /* The active ping pong output buffer */
  volatile int   statusFlag;             /* Tracks the state of the driver  */

}WM8510Handle;
```

```
void WM8510Read(WM8510Handle * pHandle, int * data, int size)
{
        int * source;
        int sampleIndex;

        if((pHandle->statusFlag & WM8510DRV_GET_BUFFER_IND) == 0)
        { source = pHandle->inputBuffer2;  }
       else  { source = pHandle->inputBuffer1;  }

       if (size > WM8510DRV_CODEC_FRAME)
       { size = WM8510DRV_CODEC_FRAME; }

       for(sampleIndex = 0; sampleIndex < size; sampleIndex++)
       { data[sampleIndex] = source[sampleIndex];  }

       /* Set the read busy flag indicating that no buffers are
        * available for reading*/

       __asm__ volatile("disi #0x4"); /* disable interrupts */
       pHandle->statusFlag |= WM8510DRV_SET_READ_BUSY;
       __asm__ volatile(
```

```
void WM8510Write(WM8510Handle * pHandle, int * data, int size)
{
       int* destination;
       int  sampleIndex;

       if((pHandle->statusFlag & WM8510DRV_GET_BUFFER_IND) == 0)
       { destination = pHandle->outputBuffer2;  }
      else { destination = pHandle->outputBuffer1; }

      if (size > WM8510DRV_CODEC_FRAME)
      { size = WM8510DRV_CODEC_FRAME; }

      pHandle->newFrameSize = size;

      for(sampleIndex = 0; sampleIndex < size; sampleIndex++)
      { destination[sampleIndex] = data[sampleIndex]; }

      /* Set the write busy flag indicating that no buffers are
       * available for writing  */
      __asm__ volatile("disi #0x4"); /* disable interrupts */
      pHandle->statusFlag |= WM8510DRV_SET_WRITE_BUSY;
      __asm__ volatile("disi #0x0"); /* enable interrupts */
}
```

```
int WM8510IsWriteBusy(WM8510Handle *pHandle)
{
   return( ((pHandle->statusFlag & WM8510DRV_GET_WRITE_BUSY) == 0) ? 0 : 1);
}

int  WM8510IsReadBusy(WM8510Handle *pHandle)
{
   return( ((pHandle->statusFlag & WM8510DRV_GET_READ_BUSY) == 0) ? 0 : 1);
}
```