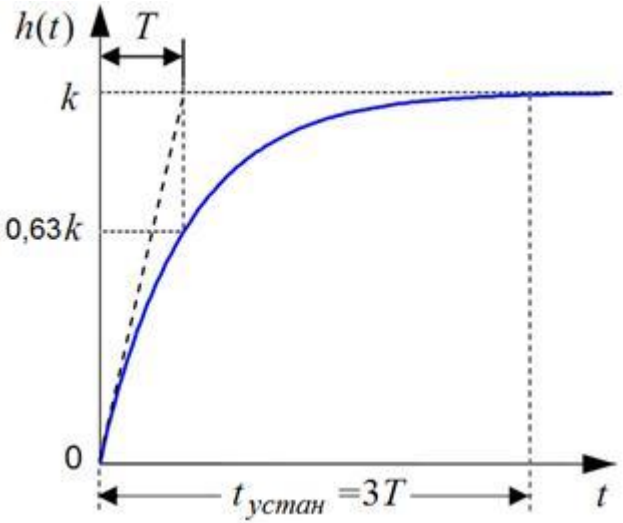
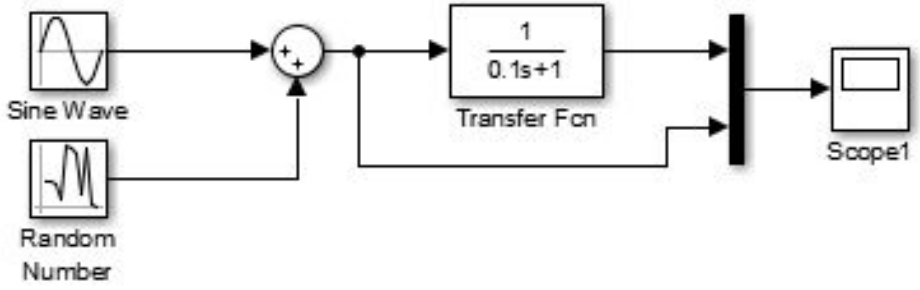
Структура и программная реализация цифровых фильтров и регуляторов

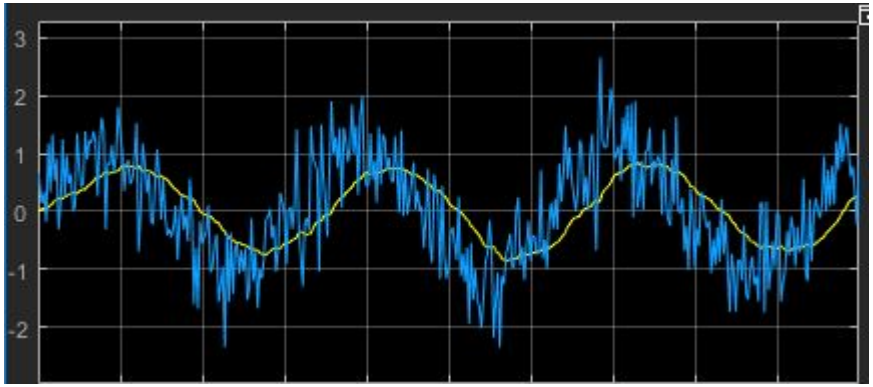
Инерционное звено первого порядка (фильтр)

$$W(p)=\frac{Y(p)}{X(p)}=\frac{1}{T_{\Phi}p+1};$$



Где $T_ф$ – постоянная времени фильтра





Пример разработки и отладки программы инерционного звена первого порядка (фильтра)

$$W(p)=\frac{Y(p)}{X(p)}=\frac{1}{T_Ф p+1};\qquad T_Ф yp+y=x;$$

Для преобразования уравнения фильтра из непрерывной области в дискретную, введем интервал дискретизации по времени h. Тогда производная в уравнении может быть представлена в виде разности первого порядка:

$$T_Ф\frac{y_k-y_{k-1}}{h}+y_k=x_k;$$

$$\left(1+\frac{T_Ф}{h}\right)y_k=\left(\frac{T_Ф}{h}\right)y_{k-1}+x_k;$$

$$y_k=\frac{\left(\frac{T_Ф}{h}\right)}{\left(1+\frac{T_Ф}{h}\right)}y_{k-1}+\frac{1}{\left(1+\frac{T_Ф}{h}\right)}x_k.$$

Пример разработки и отладки программы инерционного звена первого порядка (фильтра)

$$y_k = \frac{T_Ф}{(T_Ф + h)} y_{k-1} + \frac{h}{(T_Ф + h)} x_k.$$

$$y_k = k_{y1} y_{k-1} + k_{x0} x_k,$$

Пример разработки и отладки программы инерционного звена первого порядка (фильтра)

Пусть $Th = \frac{T_\Phi}{h}$, тогда программный код:

```
 void Filter_Init()
{
    ky1=_IQdiv(Th,Th+_IQ(1));
    kx0=_IQdiv(_IQ(1),Th+_IQ(1));
}

void Filter_Execute()
{
    y=_IQmpy(kx0,x)+_IQmpy(ky1,y);
}
```

Пример разработки и отладки программы инерционного звена первого порядка (фильтра)

$$T_Ф \frac{y_k - y_{k-1}}{h} + y_k = x_k;$$

$$y_k = x_k - T_Ф \frac{y_k - y_{k-1}}{h};$$

$$hy_k = hx_k - T_Ф y_k + T_Ф y_{k-1};$$

$$T_Ф y_k = hx_k - hy_k + T_Ф y_{k-1};$$

$$y_k = \frac{hx_k - hy_k + T_Ф y_{k-1}}{T_Ф};$$

$$y_k = y_{k-1} + h\frac{x_k - y_k}{T_Ф};$$

$$y_k = y_{k-1} + \frac{h}{T_Ф}(x_k - y_{k-1});$$

Можно принять допущение и упростить формулы:

Приблизительно равен y_{k-1}

Пример разработки и отладки программы инерционного звена первого порядка (фильтра)

Упрощенное выражение фильтра

$$y_k = y_{k-1} + \frac{h}{T_\Phi}\left(x_k - y_{k-1}\right);$$

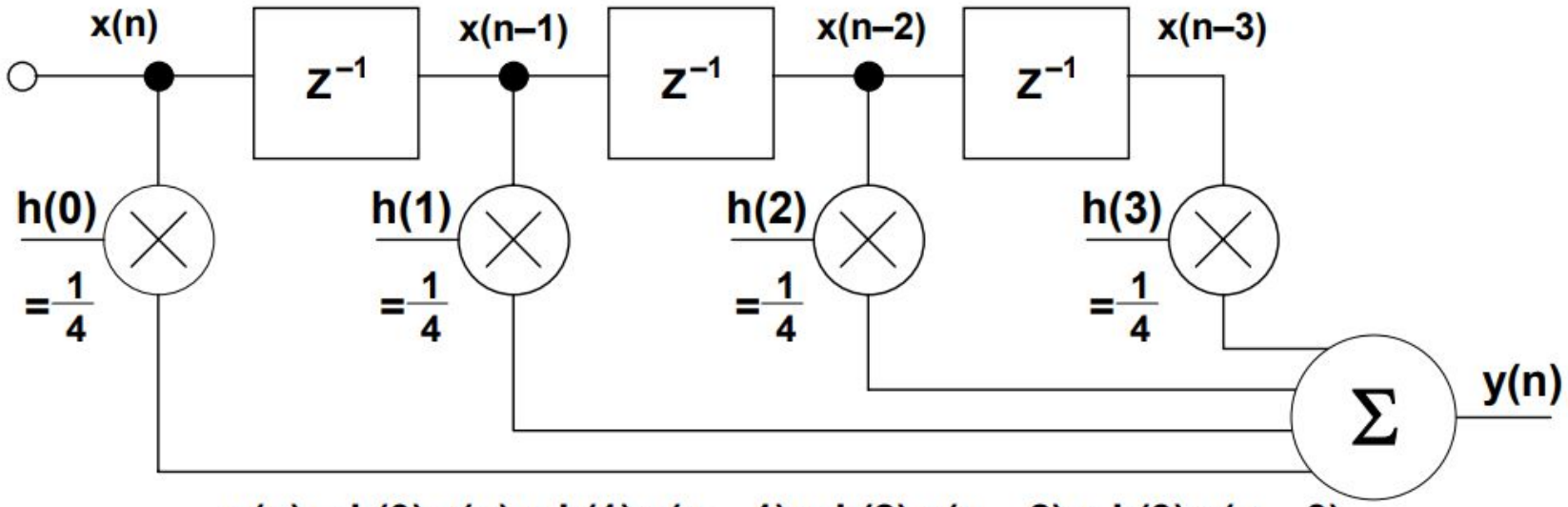
Это выражение имеет интуитивно-понятный смысл:
На данном шаге расчета отфильтрованное значение (y_k) равно тому же самому, чему было равно в прошлый раз (y_{k-1}) плюс коррекция, которая пропорциональна разнице между входом фильтра и его выходным значением. Чем отношение $h/T_ф$ больше, тем коррекция сильнее, соответственно фильтр работает «быстрее» и фильтрует слабее.

Программная реализация также проста и делается одной строкой:

```
output = output + _IQmpy(kf,(input-output));
```

Где *kf* – коэффициент фильтра, равный $h/T_ф$: отношению частоты вызова функции фильтра h к желаемой постоянной времени $T_ф$.

Структура фильтра скользящего среднего



$$y(n) = h(0)\,x(n) + h(1)\,x(n-1) + h(2)\,x(n-2) + h(3)\,x(n-3)$$

$$= \frac{1}{4}x(n) + \frac{1}{4}x(n-1) + \frac{1}{4}x(n-2) + \frac{1}{4}x(n-3)$$

$$= \frac{1}{4}\left[x(n) + x(n-1) + x(n-2) + x(n-3)\right]$$

Для N-точечного фильтра скользящего среднего $y(n) = \frac{1}{N}\sum_{k=0}^{N-1} x(n-k)$

ВЫЧИСЛЕНИЕ ВЫХОДНОГО СИГНАЛА 4-ТОЧЕЧНОГО ФИЛЬТРА СКОЛЬЗЯЩЕГО СРЕДНЕГО

$y(3) = 0.25\,[x(3) + x(2) + x(1) + x(0)]$

$y(4) = 0.25\,[x(4) + x(3) + x(2) + x(1)]$

$y(5) = 0.25\,[x(5) + x(4) + x(3) + x(2)]$

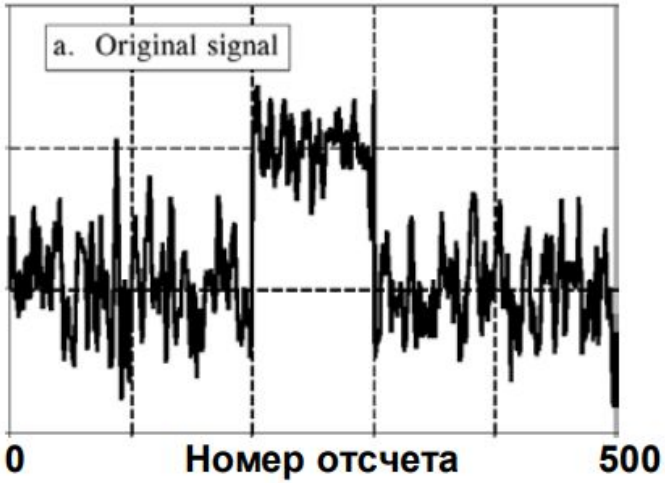
$y(6) = 0.25\,[x(6) + x(5) + x(4) + x(3)]$

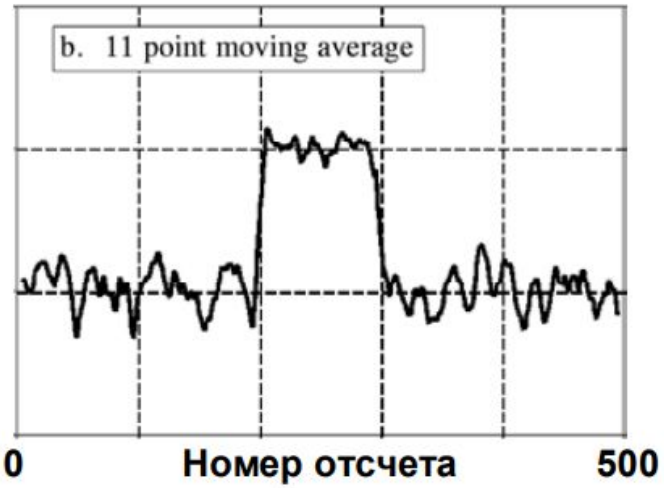
$y(7) = 0.25\,[x(7) + x(6) + x(5) + x(4)]$

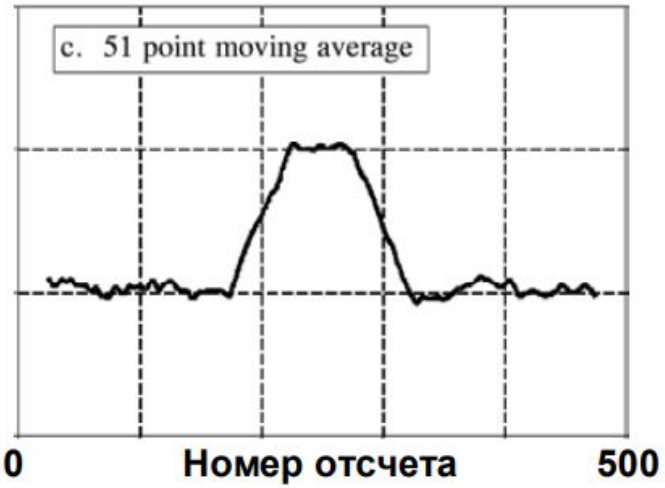
- **Вычисление каждого выходного значения требует 1 умножения, 1 сложения и 1 вычитания**

Текущий проход (скан) фильтра		Новый проход фильтра
x(k)	Свободно для записи	← x(k)
x(k-1)	x(k)	x(k-1)
x(k-2)	x(k-1)	x(k-2)
x(k-3)	x(k-2)	x(k-3)
….	x(k-3)	….
x(k-n)	x(k-n-1)	x(k-n)
«Мусорное ведро»	x(k-n) «Мусорное ведро»	«Мусорное ведро»

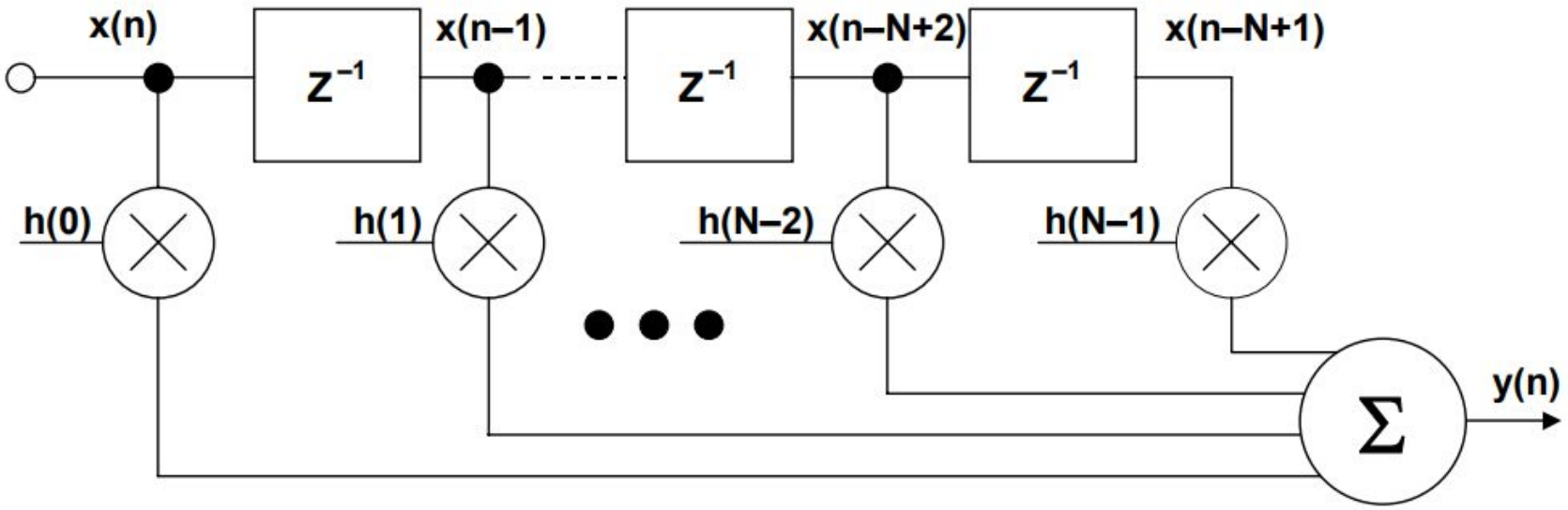
РЕАКЦИЯ ФИЛЬТРА СКОЛЬЗЯЩЕГО СРЕДНЕГО НА ВОЗДЕЙСТВИЕ В ВИДЕ СМЕСИ ШУМА И СТУПЕНЧАТОГО СИГНАЛА





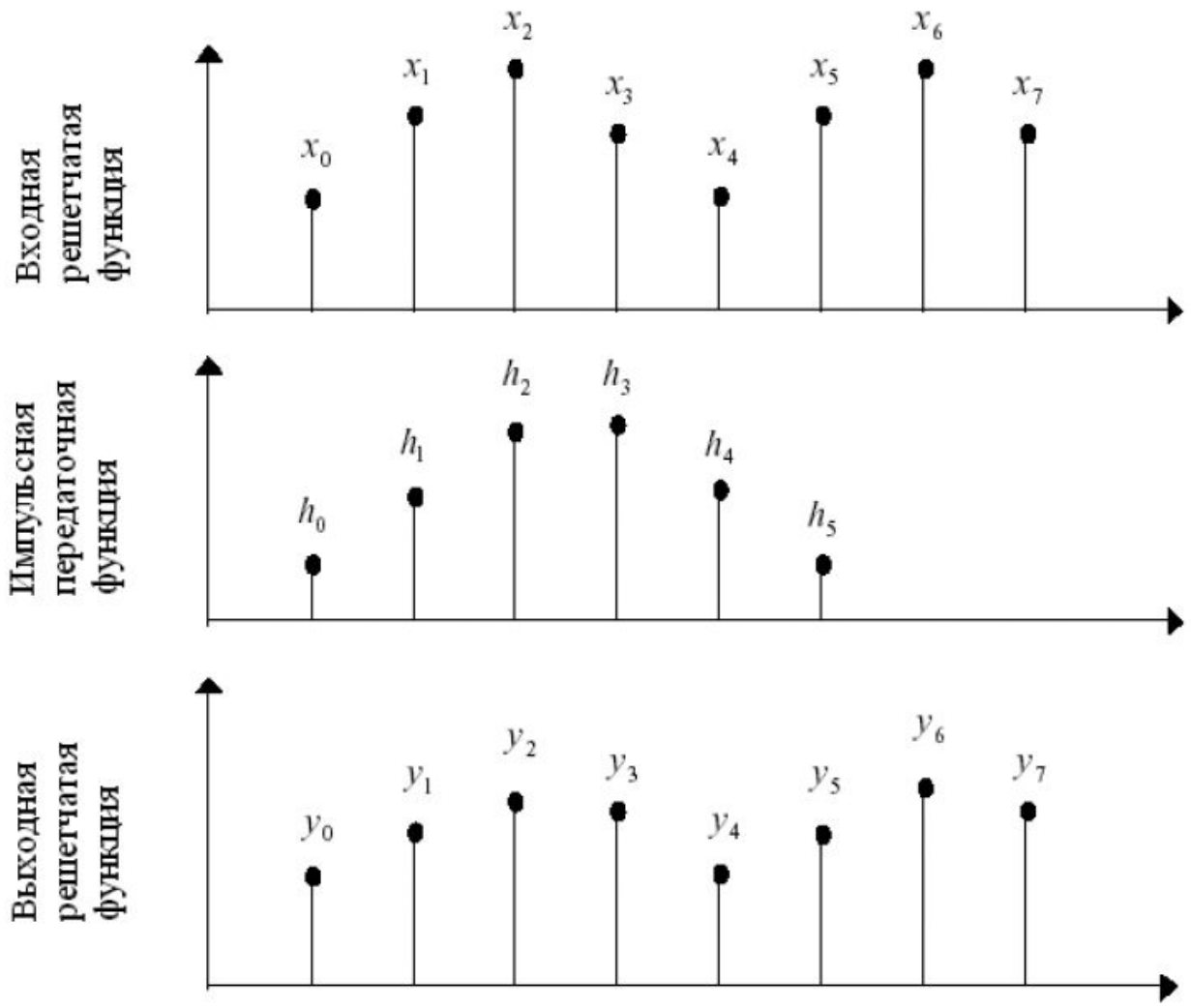


Структура КИХ-фильтра



- $y(n) = h(n) * x(n) = \sum_{k=0}^{N-1} h(k)\, x(n-k)$
- $*$ = символ свертки
- Требуется N операций умножения с накоплением для каждого выходного отсчета

Входные и выходные решетчатые функции

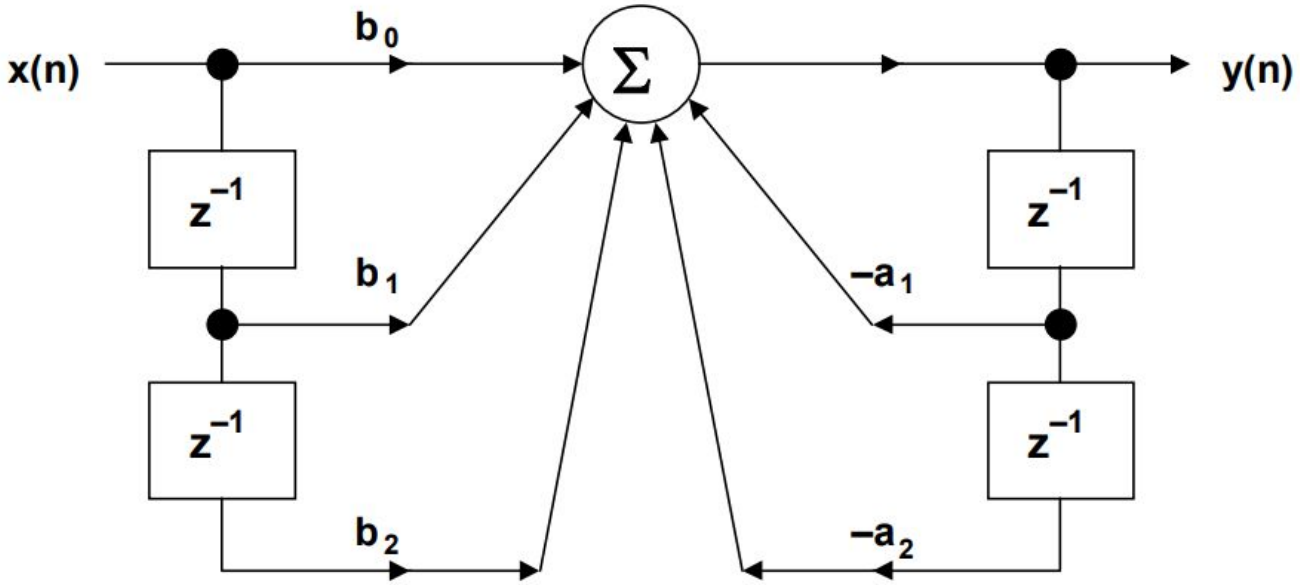


ХАРАКТЕРИСТИКИ КИХ-ФИЛЬТРОВ

Выходное управляющее воздействие (результат работы фильтра) представляет собой сумму произведений величин, сохраненных в линии задержки, на соответствующие коэффициенты. Заметьте, что число коэффициентов в фильтре всегда на единицу больше, чем порядок фильтра. Это связано с тем, что текущая выборка x(n), которая представляет собой входное воздействие, всегда участвует в процессе вычислений.
Если предположить, что буфер выборок вначале пуст, то выходное значение y_0 будет рассчитываться только на основе входной выборки x_0, выходное значение y_1 уже на основе двух выборок x_0 и x_1 и т.д. Начиная с шестой выборки x_5, все предыдущие выборки будут участвовать в расчете выходной переменной фильтра, т.е. буфер выборок окажется полностью заполненным.

- **Импульсная характеристика имеет конечную длительность (N циклов)**
- **Линейная фаза, постоянная групповая задержка (N должно быть нечетным)**
- **Нет аналогового эквивалента**
- **Безусловная устойчивость**
- **Может быть адаптивным**

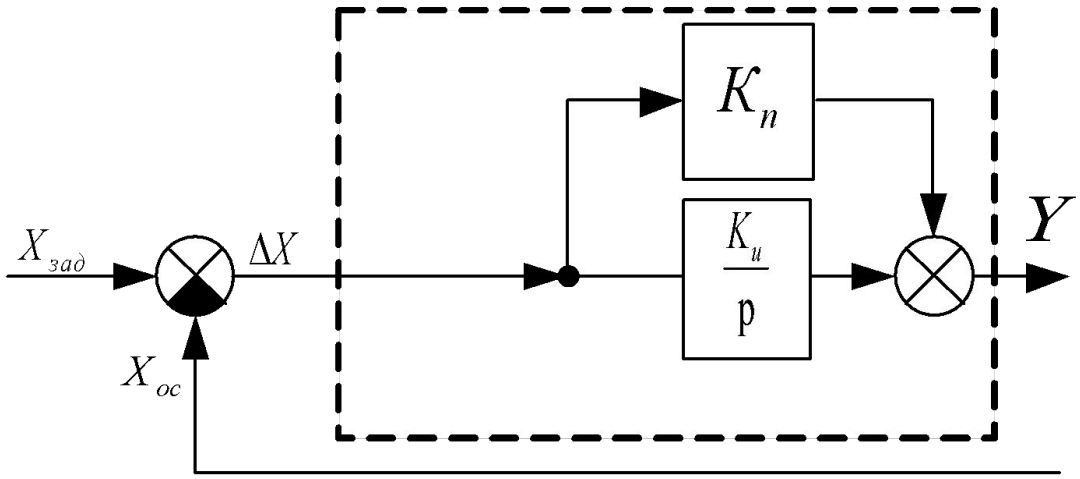
Структура БИХ-фильтра



- $y(n) = b_0x(n) + b_1x(n-1) + b_2x(n-2) - a_1y(n-1) - a_2y(n-2)$

- Имеют обратную связь (рекурсия)
- Импульсная характеристика имеет бесконечную длительность
- Потенциально нестабильны
- Нелинейная фазочастотная характеристика
- Более эффективны, чем КИХ-фильтры

ПИ-регуляторы

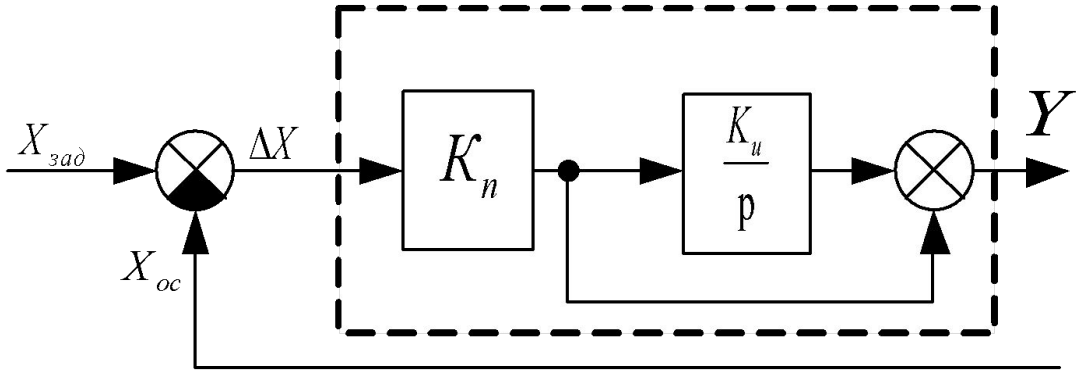


$$W(p)_{П-рег} = \frac{X_{вых}\cdot(p)_{рег}}{X_{вх}\cdot(p)_{рег}} = K_{рег}$$

$$W(p)_{И-рег} = \frac{X_{вых}\cdot(p)_{рег}}{X_{вх}\cdot(p)_{рег}} = \frac{1}{T_u \cdot p}$$

$$y_k = y_{k-1} + k_x x_k,$$

ПИ – рег



$$W(p)_{ПИ-рег} = \frac{X_{вых}\cdot(p)_{рег}}{X_{вх}\cdot(p)_{рег}}$$

$$= K_{рег}\cdot\left(1+\frac{1}{T_u \cdot p}\right)$$

$$= K_{рег} + \frac{K_{рег}}{T_u \cdot p}$$

Программная реализация ПИ-регулятора

```
65  void pid_reg3_calc(TPidReg3 *v) {
66      // Дельта сигнала
67      v->e_reg3 = v->pid_ref_reg3 - v->pid_fdb_reg3;
68
69      // Пропорциональная часть
70      v->up_reg3 = _IQmpy(v->Kp_reg3,v->e_reg3);
71
72      // Выход регулятора
73      v->uprsat_reg3 = v->up_reg3 + v->ui_reg3;
74
75      // Ограничения регулятора
76      if (v->uprsat_reg3 > v->pid_out_max)
77          v->pid_out_reg3 =  v->pid_out_max;
78      else if (v->uprsat_reg3 < v->pid_out_min)
79          v->pid_out_reg3 =  v->pid_out_min;
80      else
81          v->pid_out_reg3 = v->uprsat_reg3;
82
83      // Интегральная составляющая
84      if (v->Ki_reg3!=0)
85          v->ui_reg3 = v->ui_reg3 + _IQmpy(v->Ki_reg3,v->up_reg3);
86      else
87          v->ui_reg3=0;
88
89      // Ограничение интегральной составлющей
90      if (v->ui_reg3 > v->pid_out_max)
91          v->ui_reg3 = v->pid_out_max;
92      else if (v->ui_reg3 < v->pid_out_min)
93          v->ui_reg3 = v->pid_out_min;
94  }
```