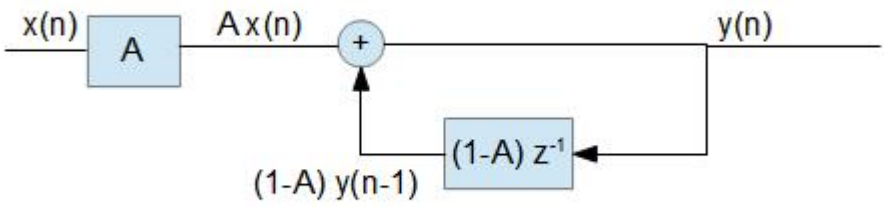
Simple Low Pass IIR Filter Implementation Using the C Language

Simple Low Pass IIR Filter Implementation Using the C Language



2020-09-01 Weimich

Simple Low Pass IIR Filter Implementation Using the C Language

1. Abbreviation

IIR – Infinite Impulse Response Filter

RISC – Reduced Instruction Set Computing

2. Introduction

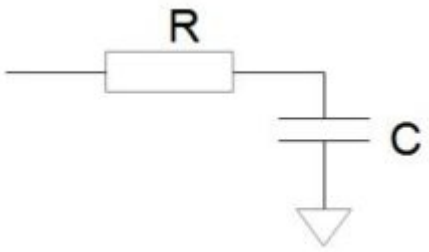


Figure 1: RC analog first order low pass filter with the T=RC

The linear analog circuit is described with the linear differential equation or with the Laplace transformation (s plane). For example: the RC circuit (low pass filter) is described with the first

order differential equation:

$$u_{input}(t) = RC\frac{du_{out}(t)}{dt} + u_{out}(t) \quad (1)$$

or in the Laplace form:

$$H(s) = \frac{1}{1 + sRC} \quad (2)$$

The analog method experience can be used for the digital technical. The linear digital methods are used the difference equation and z transformation (z plane).

$$y(n) = \sum_{i=0}^{M} b_i \cdot x(n-i) + \sum_{k=1}^{N} a_k \cdot y(n-k) \quad (3)$$

$$H(z) = \frac{\sum_{i=0}^{M} b_i \cdot z^{-i}}{1 - \sum_{k=1}^{N} a_k \cdot z^{-k}} \quad (4)$$

There are few transformation methods of the s-plane (analog space) to z-plane (digital space): backward Euler, impulse invariance, bilinear. The article is described the simple first order filter. Backward Euler method is suitable for the case.

$$s = \frac{1}{T_s}(1 - z^{-1}) \quad (5) \text{ where Ts-sampling time}$$

Insert the (5) in the (2)

$$H(z) = \frac{1}{1 + \frac{1}{T_s}(1 - z^{-1})RC} = \frac{T_s}{T_s + R \cdot C} \cdot \frac{1}{1 - \frac{R \cdot C}{T_s + R \cdot C} \cdot z^{-1}} \quad (6)$$

Specify

$$A=\frac{T_s}{T_s+R\cdot C} \quad (7)$$

Then the z-transformation

$$H(z)=\frac{A}{1-(1-A)\cdot z^{-1}} \quad (8)$$

And the difference equation

$$y(n)=(1-A)\cdot y(n-1)+A\cdot x(n)=y(n-1)+A\cdot(x(n)-y(n-1)) \quad (9)$$

3. First Order Digital IIR Filter

The low pass IIR filter first order using the backward Euler (or impulse invariance) transformation is described with the (8) and (9).

Example of the A value calculation

Sampling fs=20kHz
Filter passband fb=1kHz

Then

*RC=1/(2πfb)=1/(2*3.14*1000Hz) ≈ 1.6e-4s*
Ts=1/fs=1/20000Hz=0.5e-4s
A=Ts/(Ts+RC)=0.5e-4/(0.5e-4+1,6e-4) ≈ 0.24
y(n)=y(n-1)+0.24(x(n)-y(n-1))*

Notes

1. The value A is less as 1 => A < 1
2. The filter can be interpreted as weighting filter: y(n)=(1-A)y(n-1)+Ax(n). The input x(n) signal will be taken with wight A and previous history y(n-1) with the weight (1-A). If the input x(n) signal is not trusted then the value A shall be taken lower value.
3. The filter can be realized only with the one multiplication: y(n)=y(n-1)+A(x(n)-y(n-1)
4. IIR filter realization with fixed point arithmetic: output history y(n-1) shall be saved with the double accuracy

3.1 First Order Digital IIR Filter. Fixed Point Realization with the One Multiplication.

Fixed point realization shall be scaled. For example: input x and output y will be used signed 16 bits, scaling S=2^16 =0x10000=65536 and history Sy(n-1) will be saved signed 32 bits

Sy(n)=Sy(n-1)+SA(x(n)-y(n-1)) (10)

Output value without scaling:

y(n) = Sy(n-1)+SA(x(n)-y(n-1) +/- 0.5*S)/S (11) where 0.5S is round value

C-Code:

sint16 IIR_Filter_First_Order(sint16 x_input)

```
{
  sint32 sy_round;
  sy_round = ROUND_MACRO(sy_history);
  sy_history = sy_history + SA_VALUE*((sint32)(x_input – (sy_round>>SCALING_SHIFT)));
  sy_round = ROUND_MACRO(sy_history);
  return ((sint16)(sy_round>>SCALING_SHIFT));
}
```

3.2 First Order Digital IIR Filter. Fixed Point Realization with the Shift Operation.

Instead of the multiplication can be used the arithmetic shift operation (multiplication/division on the 2^k). The A value shall be taken 2^-k. Then the (9) can be written

$$y(n)=(1-2^{-k})\cdot y(n-1)+2^{-k}\cdot x(n)=y(n-1)+2^{-k}\cdot(x(n)-y(n-1)) \quad (11)$$

Using the scaling S=2^k in the (10)

$$2^k y(n)=2^k y(n-1)+2^k 2^{-k}(x(n)-y(n-1))=2^k y(n-1)+(x(n)-y(n-1)) \quad (12)$$

The method can be used for example for the RISC microprocessor which has not the multiplication instruction in the assembler.

C-Code:

sint16 IIR_Filter_First_Order(sint16 x_input)

```
{
  sint32 sy_round;
  sy_round = ROUND_MACRO(sy_history);
  sy_history = sy_history + ((sint32)(x_input – (sy_round>>K_VALUE)));
  sy_round = ROUND_MACRO(sy_history);
  return ((sint16)(sy_round>>K_VALUE));
}
```

3.3 First Order Digital IIR Filter. Float Arithmetic.

The equation (9) shall be used in the case.

C-Code:

sint16 IIR_Filter_First_Order(sint16 x_input)

```
{
  sint16 y_round;
  y_history = y_history + A_VALUE*(((float)x_input – y_history));
  y_round = ROUND_MACRO(y_history);
  return y_round;
}
```

4. Appendix. Solution of the differential and difference first order equations

***Consider the (1) differential equation with the input step signal:**

-

$u_{input}(t) = 1(t)$ where

$$if\ t<0\ then\ 1(t)=0$$
$$if\ t\geq 0\ then\ 1(t)=1 \quad (13)$$

Then the solution of the differential equation:

$$u_{out}(t)=1-e^{-\frac{t}{RC}} \quad (14)$$

Consider the (9) difference equation with the input step signal:

$x_{input}(n) = 1(n)$ where

$$if\ n<0\ then\ 1(n)=0$$
$$if\ n\geq 0\ then\ 1(n)=1 \quad (15)$$

Then the solution of the difference equation:

$$y(n)=1-(1-A)^n=1-e^{n\cdot\ln(1-A)} \quad (16)$$

5. Appendix. C-Source Codes

You can use the follow C-Source codes for your applications:

IIRfirstOrderFixedPoint.c/h – implementation with single multiplication and fixed point arithmetic

IIRfirstOrderFloat.c/h – implementation with single multiplication and float point arithmetic

IIRfirstOrderShift.c/h – implementation with shift operation instead of multiplication and fixed point arithmetic

Download

6. Literature / References

[1] Harry Y.-F. Lam „Analog and digital filters: design and realization“, Prentice-Hall Inc, 1979

[2] Lawrence R. Rabiner, Bernard Gold „Theory and application of digital signal processing“, Prentice-Hall Inc, 1975