
Real-time Filtering

CPE 381 Fundamentals of Signals and Systems
for Computer Engineers

Dr. Emil Jovanov

Matlab environment

- ◆ De-facto standard for scientific computing
 - <http://www.mathworks.com/>
- ◆ Mathematical computation, analysis, visualization, and algorithm development
- ◆ Toolboxes
 - Signal Processing, Image Processing, Data Acquisition
- ◆ Vector/Array oriented computation
 - Custom procedures
- ◆ Simulink
- ◆ Real-time interfaces & hardware

Signal processing environment


- ◆ Function generators
 - Convenient preparation of constant arrays
 - ◆ DA conversion
- ◆ Signal processing tool
 - `sptool`
- ◆ Filter Design and Analysis tool
 - `fdatool`
 - Generate C-header files with your filter coefficients
 - ◆ Targets/Generate C header
 - Export filters to Simulink modules
- ◆ Wavelet toolbox
 - `wavemenu`

Real-time Processing

- ◆ What is real-time?
 - Processing your samples fast enough to influence the system you control
- ◆ Hard real time and soft real time systems
 - Always evaluate performance
- ◆ Matlab processes the whole array
- ◆ Real-time systems receive data sample-by-sample from signal generators, such as AD converter
- ◆ Buffering, processing, output
 - Latency handling – signal processing

Digital filters

- ◆ Processing previous inputs ($X[i]$) and outputs ($Y[i]$) to evaluate the value of the current output sample
 - nb input samples and coefficients
 - na previous output samples and coefficients
 - current output is the sum of products of previous samples and coefficients
- ◆ Types of filters
 - IIR (Infinite Impulse Response) → general form
 - FIR (Finite Impulse Response) → only input samples $X[i]$
- ◆ Matlab function – filter
 - run Matlab “filtering” demo



`FILTER` One-dimensional digital filter.

`Y = FILTER(B,A,X)` filters the data in vector `X` with the filter described by vectors `A` and `B` to create the filtered data `Y`. The filter is a "Direct Form II Transposed" implementation of the standard difference equation:

$$a(1)*y(n) = b(1)*x(n) + b(2)*x(n-1) + \dots + b(nb+1)*x(n-nb) - a(2)*y(n-1) - \dots - a(na+1)*y(n-na)$$

Matlab filter Design and Analysis

◆ Matlab tools

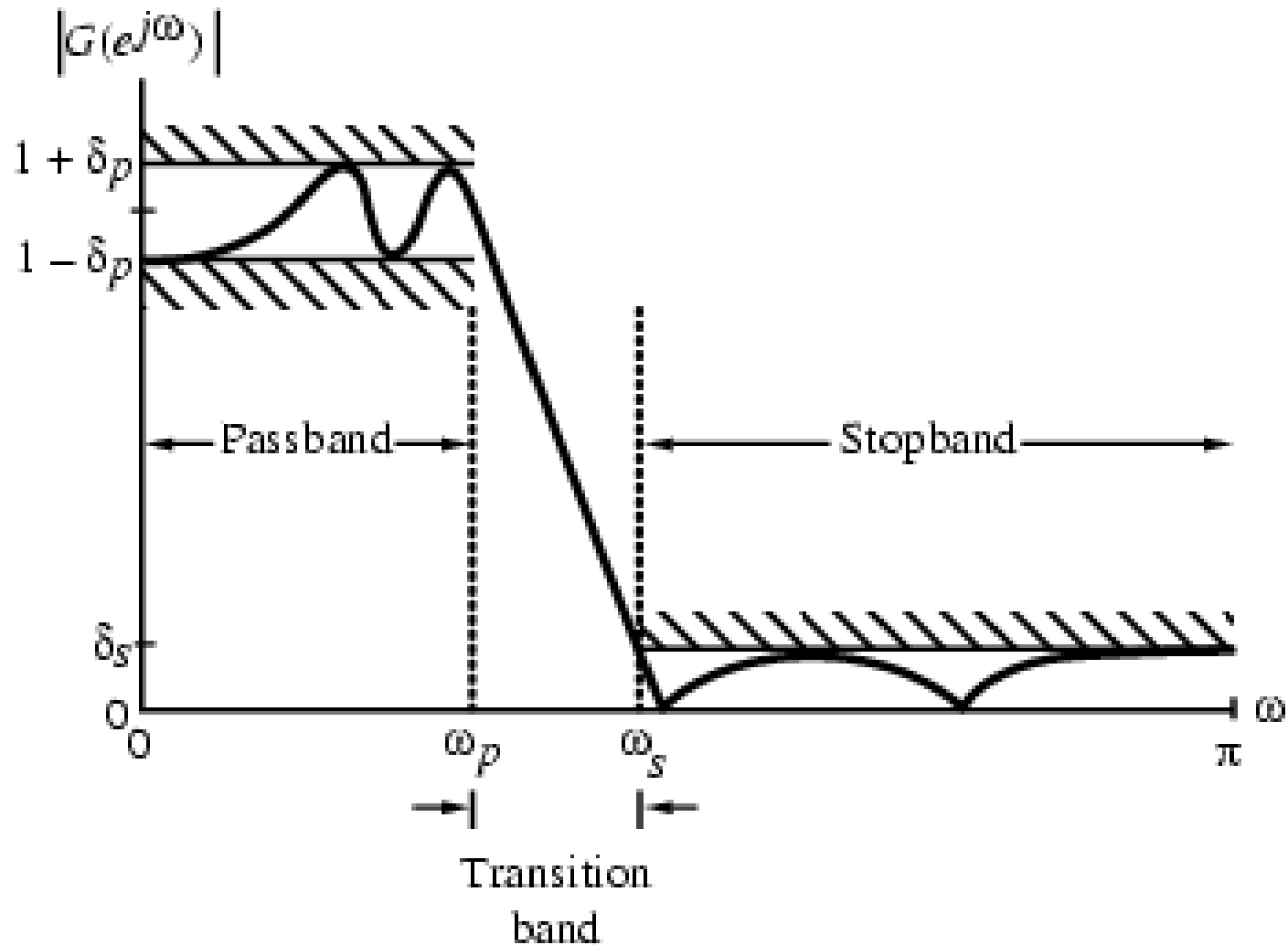
- sptool for preliminary signal processing and analysis
 - ◆ Example 3D accelerometer with LP and HP filtering
 - ↗ Signal viewer
 - ↗ Filter design and testing
 - ↗ Spectral analysis
- fdatool for filter design and export
 - ◆ Tools → generate C header file

◆ Other tools ...

◆ Specialized filter controllers

- <http://www.quickfiltertech.com/index.php>

Filter Specification



Filter Order Estimation

- ◆ Filter order (Fred Harris' guide):

$$N \cong \frac{A}{20(\omega_s - \omega_p) / 2\pi}$$

- A – attenuation
- Pass and stop band
- Add 10%

Example

- ◆ Generate a signal that consists of two sinewaves with frequency of 5Hz and 40Hz and sampling frequency of 200Hz.
- ◆ Design FIR and IIR low pass filter with cut-off frequency of 7 Hz and attenuation of 40 dB at 40 Hz.

Fixed point filter implementation

- ◆ Microcontrollers emulate floating point operations
 - Running fixed point operations much faster
 - The precision may not be sufficient for some applications
 - Example `ffilt.c` on our web-site
- ◆ Representing floating point numbers using fixed point values (arithmetic operations)
- ◆ Assume:
 - $\text{max}(\text{coefficient value}) = \text{MAX_INT}$
 - scale all coefficients to `MAX_INT`
- ◆ Optimize individual terms

Filter implementation - example

◆ Optimized processing

◆ Example - low pass IIR filter, coefficients:


b(1) -> 0.009236

b(2) -> 0.018472

b(3) -> 0.009236

a(2) -> -1.710329

a(3) -> 0.747274

- filter coefficient: 0.009236
- fixed point coef. value: $65536 \times () = 605.290496$ (0x25D)
- binary value 0b 0000 00 10 0101 1101 010010
- Loosing 6-bits of precision!!! 
- Old value: 0b0000001001011101
- New value: 0b1001011101010010 (dec. 38738)

◆ Processing

- temporary result: unsigned long templ;
- `templ += (38738 * x[0]) >> 6;`

Filter implementation – example (2)

◆ Matlab sequence of the previous example:

```
>> c=0.009236;      % coefficient value
>> sfact=floor(log2(1/c)) % scale factor
sfact = 6
>> cint=round(c2*65535*2^sfact) % int value
cint = 38738
```

fix_filt.c

```

/*****
file: fix_filt.c
description: Fixed point FIR and IIR filtering routines,
            all procedures manually optimized for maximum precision.
author: Emil Jovanov
date: November 3, 2001. */

// input & output samples
#define FILT_LEN 12
int NB=FILT_LEN;                // filter length

/**** FIR filter - fixed point ****/
void xfir_filter(int * x, int * y, int sample) {
    /* fixed point filter procedure, xin - input signal , yout - filtered input signal */
    long templ;
    register int ii;

    /* the latest sample is at index 0, all other are shifted */
    for (ii=NB-1;ii>0;ii--) {
        x[ii]=x[ii-1];
        y[ii] = y[ii-1];
    }

    x[0]=sample;

    /**** B coefficients */
    templ=0;
    templ += (44530 * x[0]) >> 7;    /* b(1) -> 0.005308 */
    templ += (47034 * x[1]) >> 5;    /* b(2) -> 0.022428 */
    templ += (58693 * x[2]) >> 4;    /* b(3) -> 0.055974 */
    templ += (53836 * x[3]) >> 3;    /* b(4) -> 0.102684 */
    templ += (39233 * x[4]) >> 2;    /* b(5) -> 0.149662 */
    templ += (47034 * x[5]) >> 2;    /* b(6) -> 0.179419 */
    templ += (47034 * x[6]) >> 2;    /* b(7) -> 0.179419 */
    templ += (39233 * x[7]) >> 2;    /* b(8) -> 0.149662 */
    templ += (53836 * x[8]) >> 3;    /* b(9) -> 0.102684 */
    templ += (58693 * x[9]) >> 4;    /* b(10) -> 0.055974 */
    templ += (47034 * x[10]) >> 5;   /* b(11) -> 0.022428 */
    templ += (44530 * x[11]) >> 7;   /* b(12) -> 0.005308 */

    y[0]=templ >> 16;
}

**** Filter initialization ****/
void filt_init_var(int *x, int *y) {
    register int ii;

    for (ii=0; ii<FILT_LEN; ii++)
        x[ii] = y[ii] = 0;
}

/**** IIR filter - fixed point ****/
void xiir_filter(int * x, int * y, int sample) {
    /* fixed point filter procedure
       xin - input signal
       yout - filtered input signal
    */
    long templ;
    register int ii;

    /* the latest sample is at index 0, all other are shifted */
    for (ii=NB-1;ii>0;ii--) {
        x[ii]=x[ii-1];
        y[ii]=y[ii-1];
    }
    x[0]=sample;

    /**** B coefficients */
    templ=0;
    templ += (38740 * x[0]) >> 6;    /* b(1) -> 0.009236 */
    templ += (38740 * x[1]) >> 5;    /* b(2) -> 0.018472 */
    templ += (38740 * x[2]) >> 6;    /* b(3) -> 0.009236 */
    /**** A coefficients */
    templ += (56044 * y[1]) << 1;    /* a(2) -> -1.710329 */
    templ -= (48973 * y[2]);          /* a(3) -> 0.747274 */

    y[0]=templ >> 16;
}

```

Filter Coefficients

◆ Group scaling of coefficients

- Scale factor $1/\text{max_coeff_value}$:
 - ◆ $1/1.710329 = 0.58$ ($1/2$)
- $\text{Coeff_shift} = 0.5 * 65536$

◆ Individual scaling of coefficients

- Scale factor $1/\text{coeff_value}$:
 - ◆ $1/0.005308 = 188$
- $\text{Coeff_shift} = \text{the largest power of two} \rightarrow 128$ (2^7)
 - ◆ Coefficient value: $\text{round}(128 * 65536 * 0.005308) = 44,527$

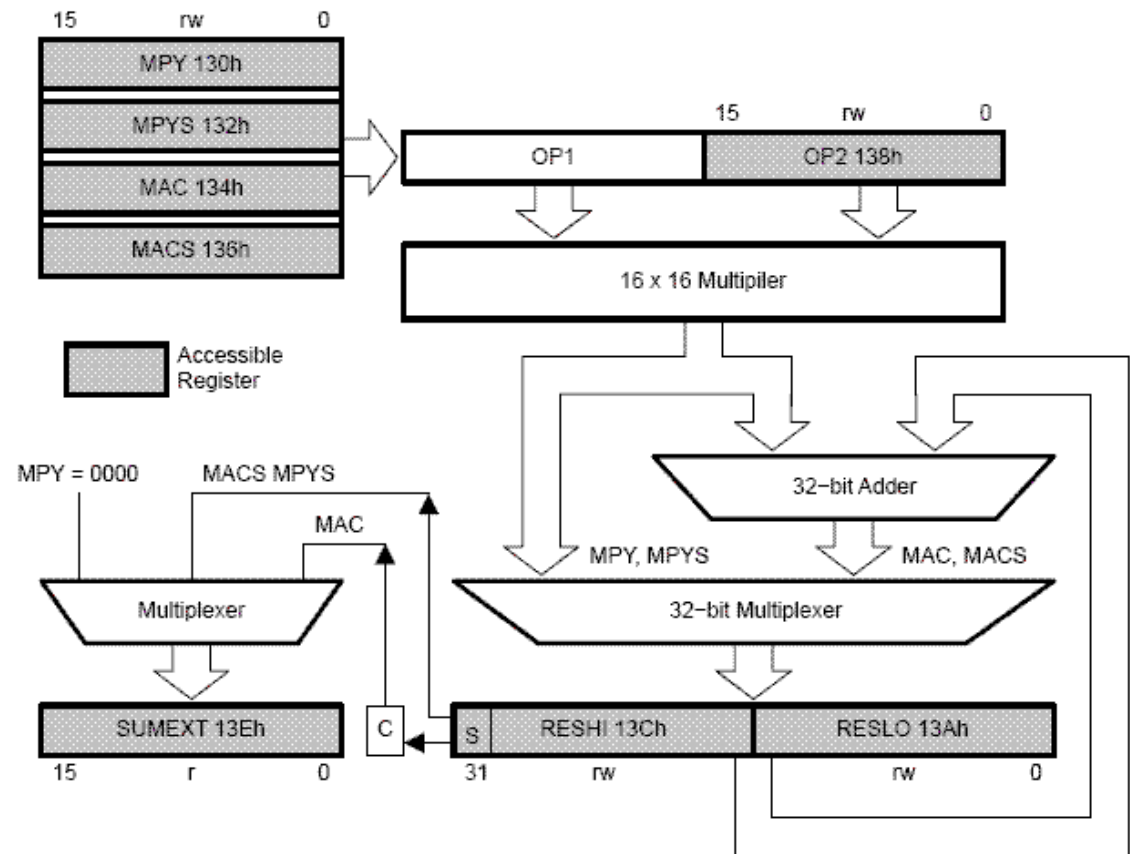
◆ Temporary result:

- $\text{templ} += (\text{sample} * \text{coeff}) \gg \text{coeff_shift}$

MSP430 Hardware multiplier

◆ The hardware multiplier supports:

- Unsigned multiply
 - ◆ MPY
- Signed multiply
 - ◆ MPYS
- Unsigned multiply accumulate
 - ◆ MAC
- Signed multiply accumulate
 - ◆ MACS
- 16×16 bits, 16×8 bits, 8×16 bits, 8×8 bits



Fiter implemenation using DMA

- ◆ Direct use of the hardware multiplier
- ◆ DMA used to access coefficients and data
- ◆ Precision/Performance

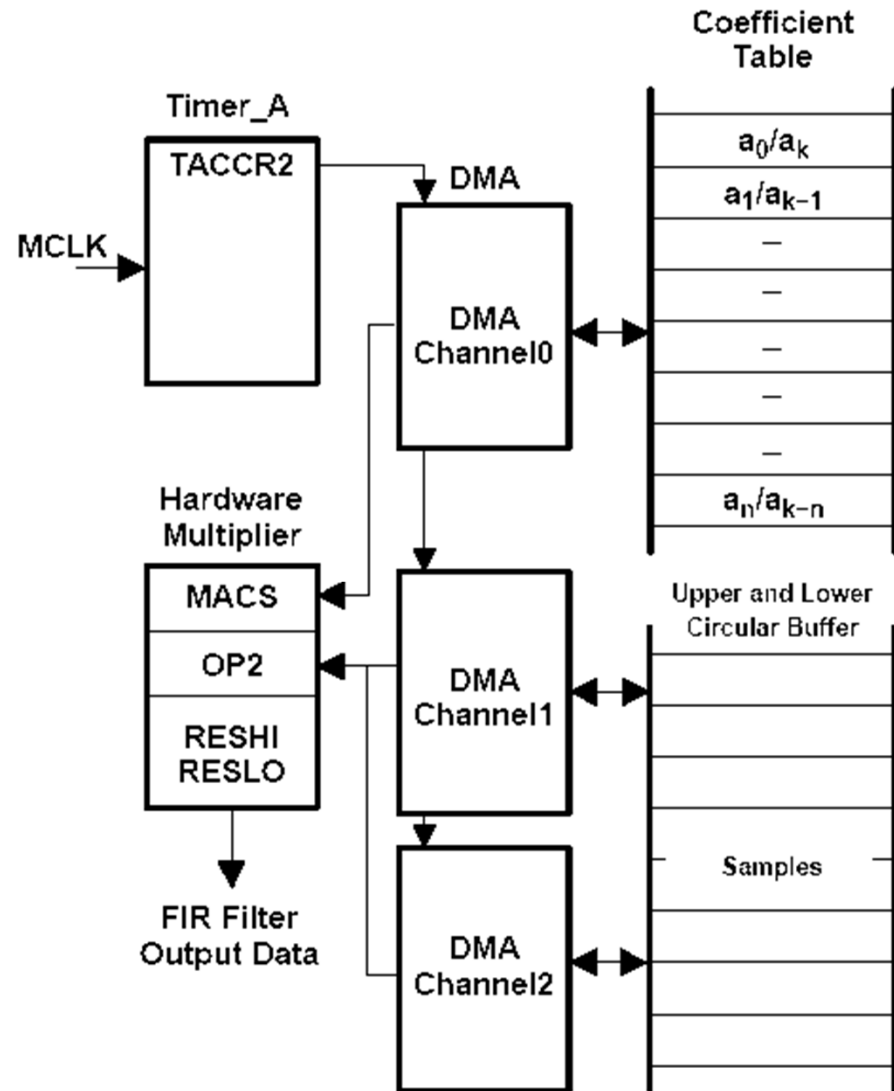


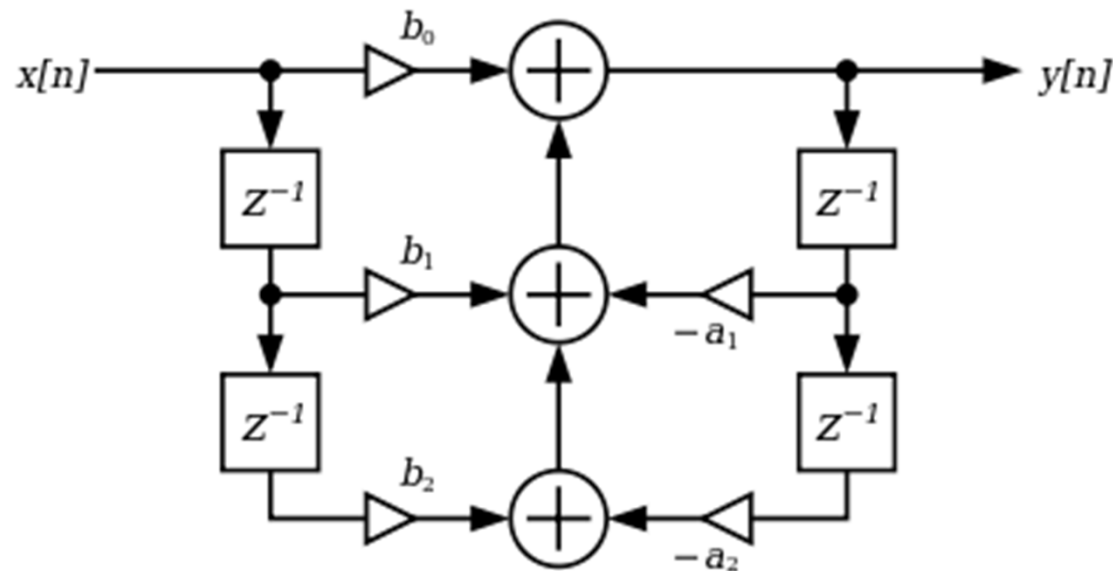
Figure 1. FIR Implementation Using the MSP430F169

Filter Implementation: Direct Form 1

The most straightforward implementation is the Direct Form 1, which has the following different equation:

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

Here the b_0 , b_1 and b_2 coefficients determine zeros, and a_1 , a_2 determine the position of the poles. Flow graph of biquad filter:



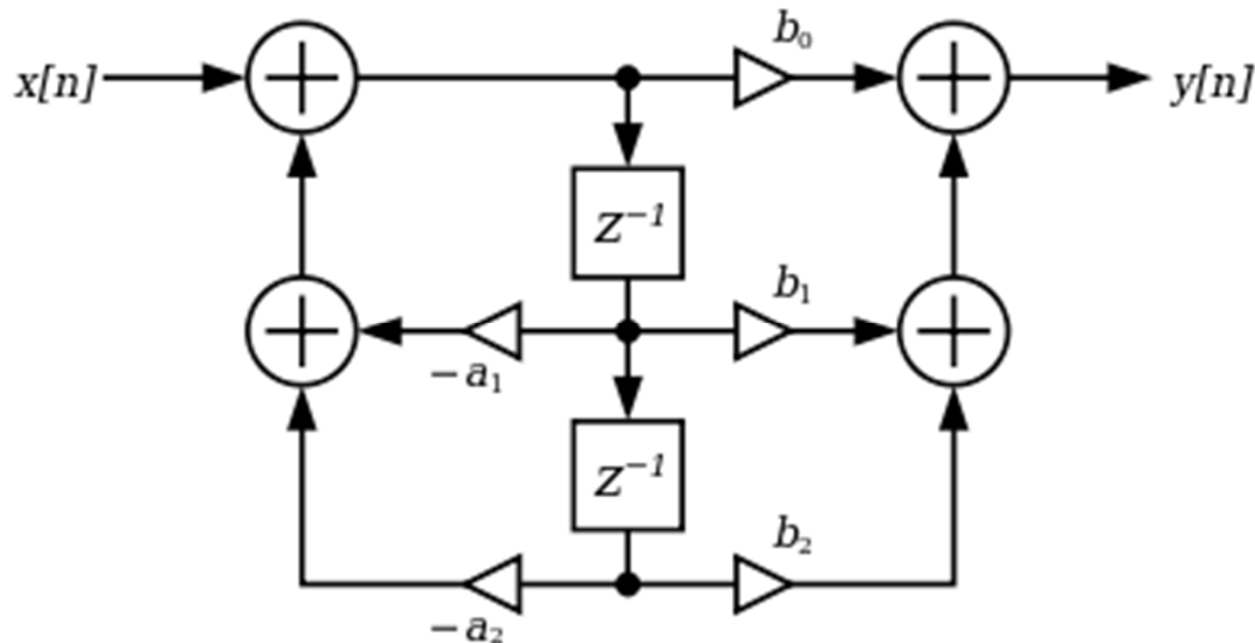
Filter Implementation: Direct Form 2

The Direct Form 1 implementation requires four delay registers. An equivalent circuit in the Direct Form 2 implementation requires two delay registers.

The Direct Form 2 implementation is called the canonical form, because it uses the minimal amount of delays, adders and multipliers, yielding in the same transfer function as the Direct Form 1 implementation. The difference equations for DF2 are:

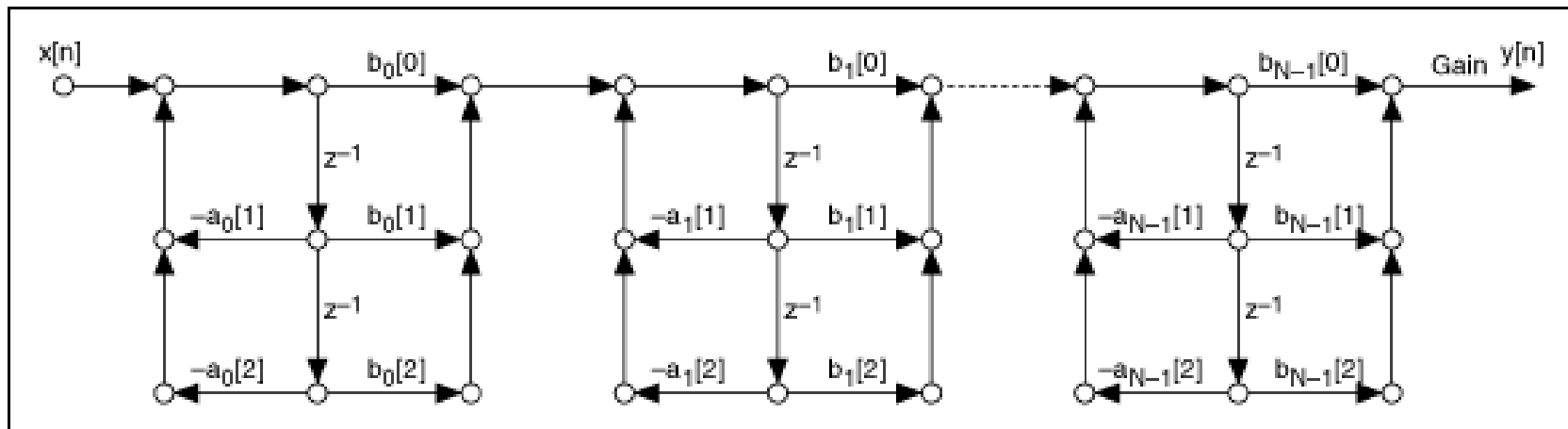
where

$$y(n) = b_0 w(n) + b_1 w(n-1) + b_2 w(n-2),$$
$$w(n) = x(n) - a_1 w(n-1) - a_2 w(n-2).$$



Filter Implementation: Second Order Sections

$$H(z) = \text{Gain} \cdot \prod_{n=0}^{N-1} \frac{b_n[0] + b_n[1]z^{-1} + b_n[2]z^{-2}}{1 + a_n[1]z^{-1} + a_n[2]z^{-2}}$$



```
function [sos,g] = tf2sos(B,A)
[z,p,g]=tf2zp(B(:)',A(:)'); % Direct form to (zeros,poles,gain)
sos=zp2sos(z,p,g); % (z,p,g) to series second-order sections
```

Conclusion

- ◆ Real-time processing is critical for many embedded systems
- ◆ Check the performance of your programs
- ◆ Support the worst case
- ◆ Optimize power after satisfying processing requirements