Digital Signal Processing

Higher Order Recursive Filters Using Second Order Stages

Today's Topics

- Review higher order IIR Direct Filters
- Demonstrate stability issues with the higher order filter
- Introduce 2nd order stages (SOS) approach
- Discuss Implementation and demonstrate results

Higher Order Recursive (IIR) Filters

- Adding poles and zeros to the transfer function can improve the frequency response of the filter

Zeros are the roots of the numerator

$$H(z) = \frac{\sum_{k=0}^{M-1} a_k z^{-1}}{1 - \sum_{k=0}^{N-1} b_1 z^{-1}}$$

Poles are the roots of the denominator

Implementing the High Order Recursive Filters

- Recursive filters compute the next output using:
  - The filter input values $x[n], x[n-1]$ ...
  - *and* the past filter output values $y[n-1], y[n-2]$, ...

- The difference equation is:

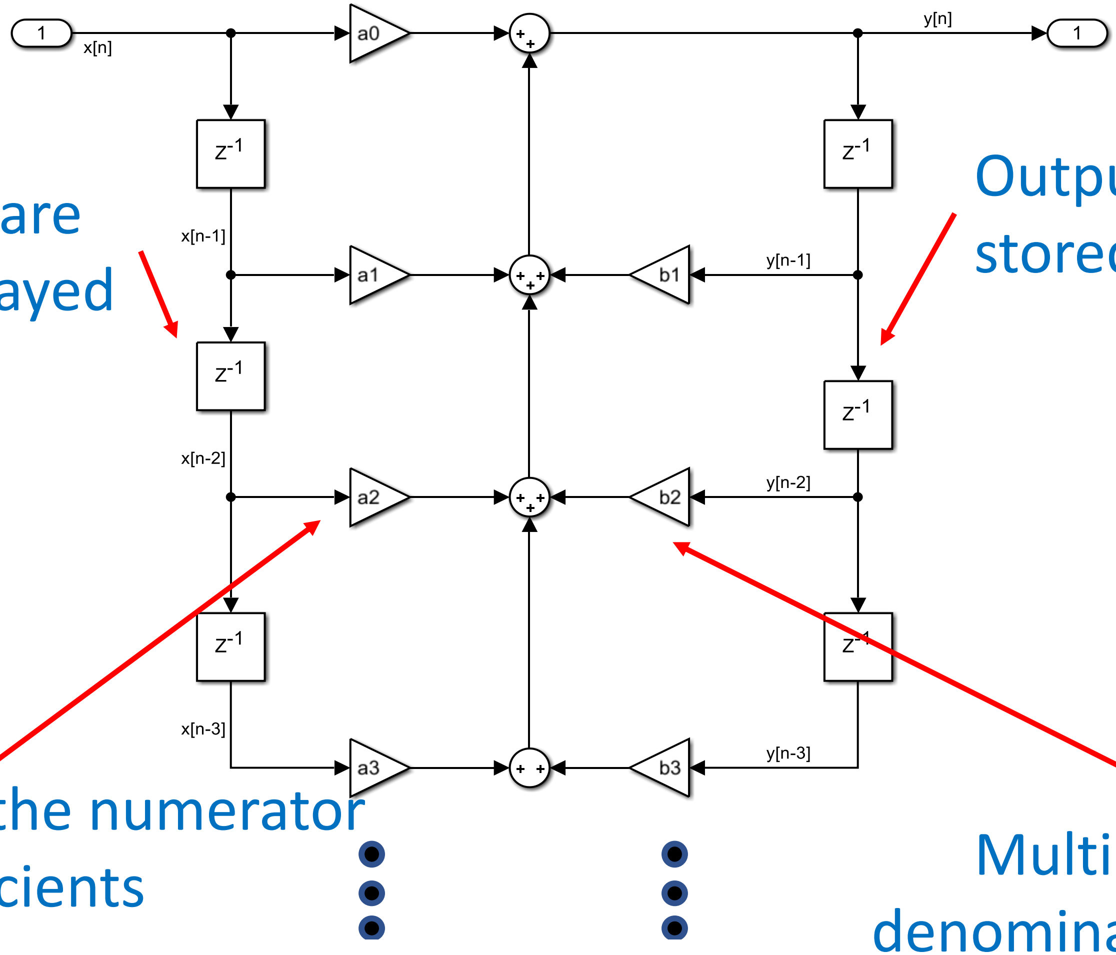
$$y[n] = a_0 x[n] + a_1 x[n-1] + a_2 x[n-2] + a_3 x[n-3] + \cdots$$
$$b_1 y[n-1] + b_2 y[n-2] + b_3 y[n-3] + \cdots$$

- This is referred to as the direct form of the IIR

High Order Recursive Filters
Block Diagram

Input samples are stored and delayed

Output samples are stored and delayed



Multiplied by the numerator coefficients

Multiplied by the denominator coefficients

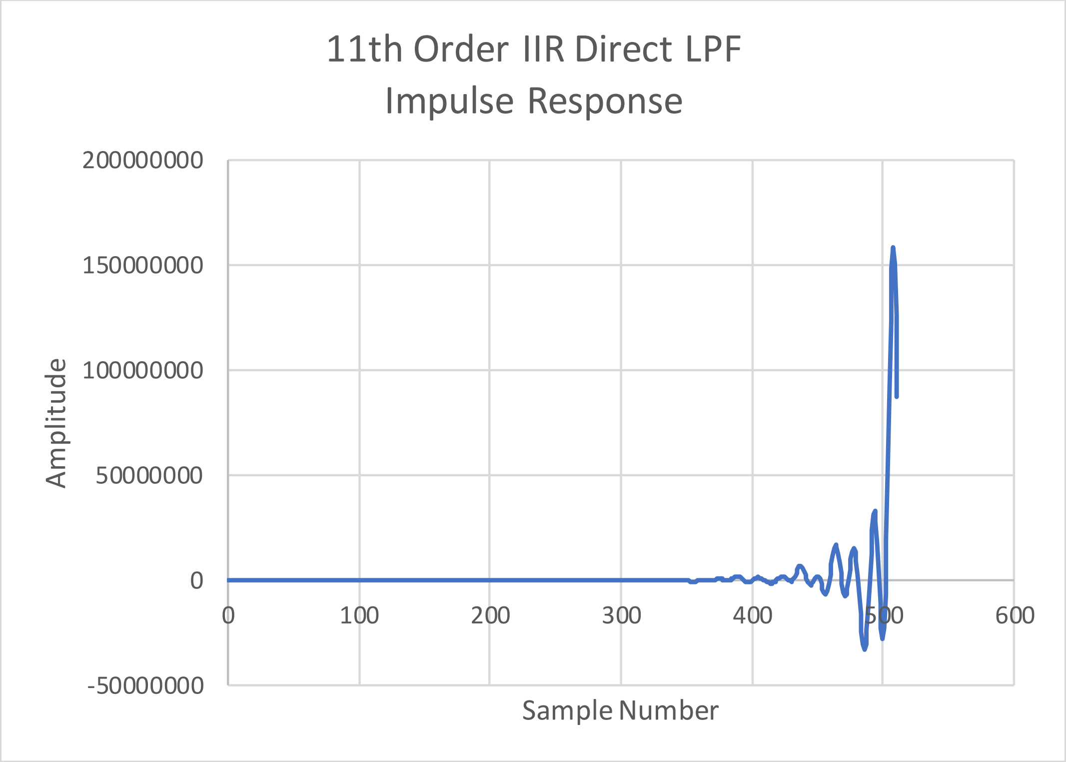
LPF Filter Demonstration

- Implement an 11th order direct IIR filter using coefficients from MATLAB
  - 11th order
  - Ripple = 0.5 dB
  - Corner Frequency = 50 BPM

```
// CHEBY LOW, order 11, R = 0.5, 50 BPM
const int MFILT = 12;
static float GAIN = 4.04401e-07;
static float b[] = {0.0021645, 0.0238095, 0.1190476, 0.3571429, 0.7142857, 1.0000000, 1.0
static float a[] = {1.0000000, -9.6818650, 43.2839718, -117.8814588, 217.2145859, -284.25
```

High Order Filter Impulse Response – IIR Direct

- 11^{th} Order Chebyshev LPF



Filter becomes unstable and impulse response grows without bounds

How Can we Make the Filter Stable?

- Factor the higher order transfer function into quadratic polynomials

$$H(z) = \frac{\sum_{k=0}^{M-1} a_k z^{-1}}{1 - \sum_{k=1}^{N-1} b_1 z^{-1}}$$

Second Order Section #1

Second Order Section #2

$$H(z) = \boxed{g_1 \frac{1 + a_{11} z^{-1} + a_{12} z^{-2}}{1 + b_{11} z^{-1} + b_{12} z^{-2}}} \times \boxed{g_2 \frac{1 + a_{21} z^{-1} + a_{22} z^{-2}}{1 + b_{21} z^{-1} + b_{22} z^{-2}}} \times$$

$$\boxed{g_3 \frac{1 + a_{31} z^{-1} + a_{32} z^{-2}}{1 + b_{31} z^{-1} + b_{32} z^{-2}}} \times \cdots$$

Continues until all sets of 2 poles and zeros are accounted for.
Could have an additional First Order Section

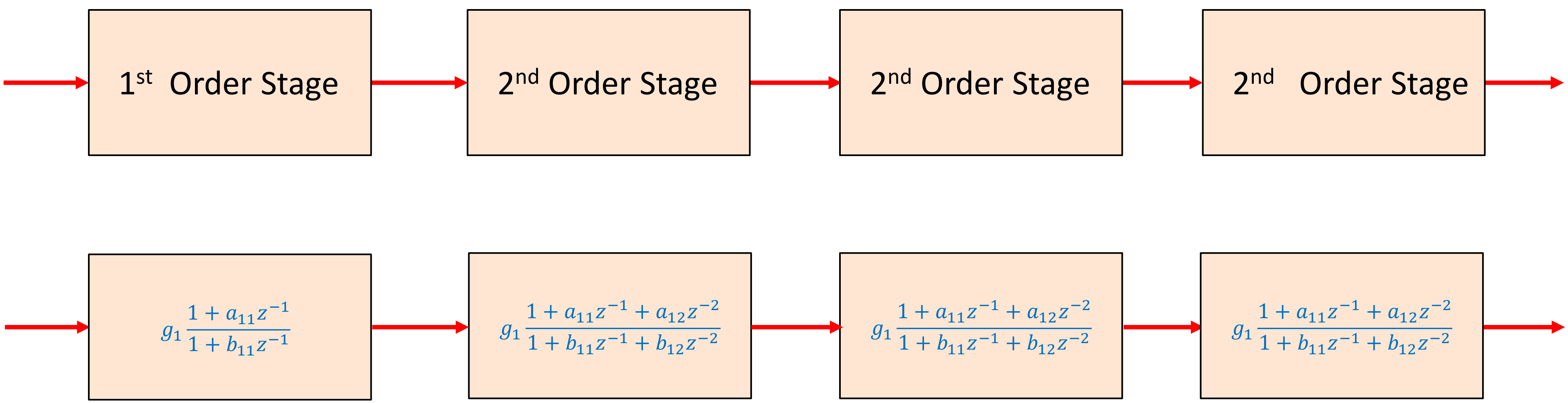
Second Order Section #3

How Many Sections?

- Each set of 2 poles and zeros are a second order stage
- If the filter is an odd order then there is an additional 1st order section
- Example 5th order filter will have two second order stages and one first order stage

Cascade the second order stages

- Cascade multiple stages together. Each stage is stable by itself creating a stable system
- Example 7th order system



How do we find the coefficients for each stage?

- We will use MATLAB to take the transfer function and convert it into second order stages

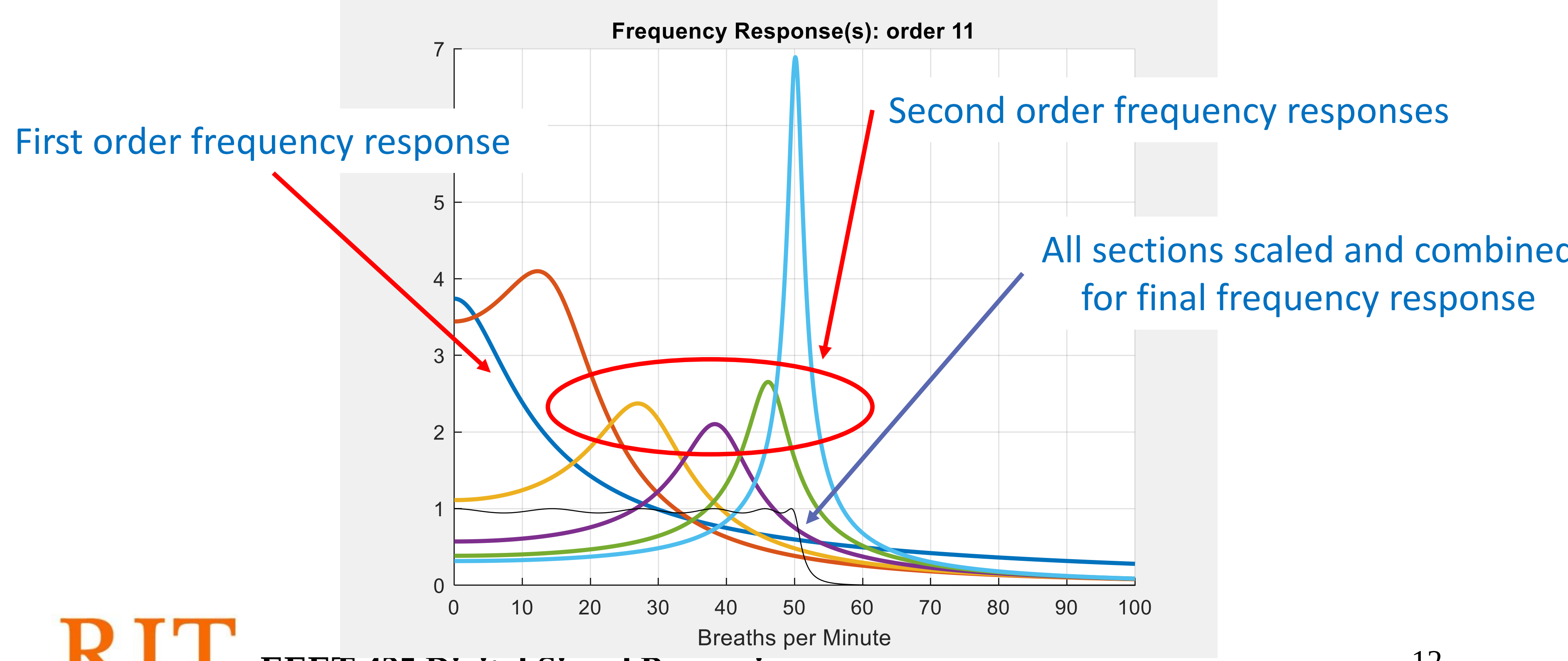
```
%-----------------------------------------------------
% Plot cascade and sos component frequency response

[sos,G1] = tf2sos(b,a); % display second order systems
[rsos,~] = size(sos);
```

The MATLAB command tf2sos(b,a) takes the denominator and numerator coefficients and breaks them up into second order stages

Frequency Response of Each Section

- Each Individual Section has its own frequency response
- Cascading in the time domain is convolution
- Frequency domain multiplication



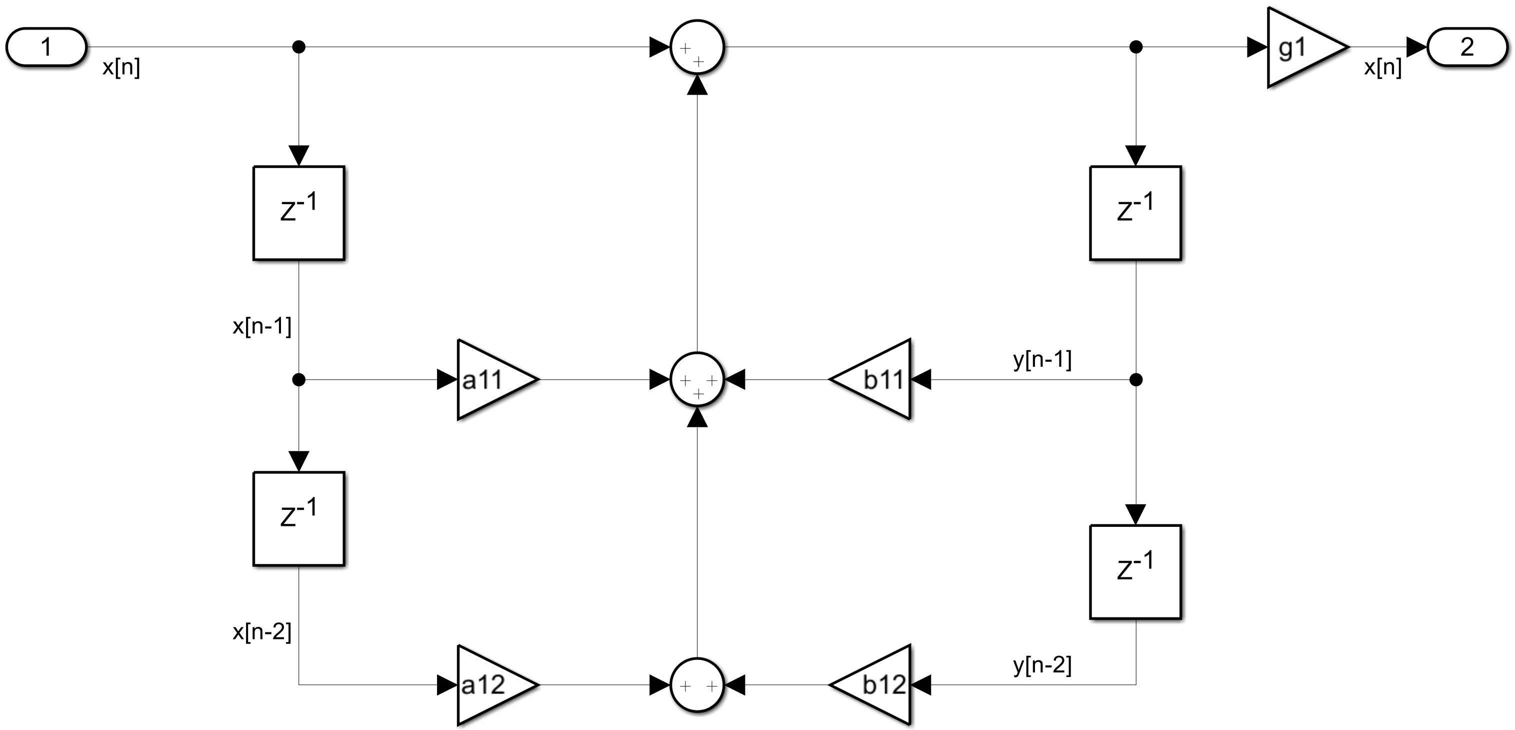
The Stage Coefficients

- MATLAB script prints the coefficients for each stage

```
// CHEBY LOW, order 11, R = 0.5, 50 BPM
G[0] = 0.0309287;
b[0][0] = 1.0000000; b[0][1] = 1.0724489; b[0][2]= 0.0000000;
a[0][0] = 1.0000000; a[0][1] =  -0.9168034; a[0][2] =  0.0000000;
G[1] = 0.0309287;
b[1][0] = 1.0000000; b[1][1] = 2.1201192; b[1][2]= 1.1253206;
a[1][0] = 1.0000000; a[1][1] =  -1.8194906; a[1][2] =  0.8472945;
G[2] = 0.0309287;
b[2][0] = 1.0000000; b[2][1] = 2.0557874; b[2][2]= 1.0608632;
a[2][0] = 1.0000000; a[2][1] =  -1.7832527; a[2][2] =  0.8667261;
G[3] = 0.0309287;
b[3][0] = 1.0000000; b[3][1] = 1.9761101; b[3][2]= 0.9810257;
a[3][0] = 1.0000000; a[3][1] =  -1.7406125; a[3][2] =  0.8966174;
G[4] = 0.0309287;
b[4][0] = 1.0000000; b[4][1] = 1.9072737; b[4][2]= 0.9120469;
a[4][0] = 1.0000000; a[4][1] =  -1.7108628; a[4][2] =  0.9342760;
G[5] = 0.0309287;
b[5][0] = 1.0000000; b[5][1] = 1.8682606; b[5][2]= 0.8729515;
a[5][0] = 1.0000000; a[5][1] =  -1.7108429; a[5][2] =  0.9772188;
..
```

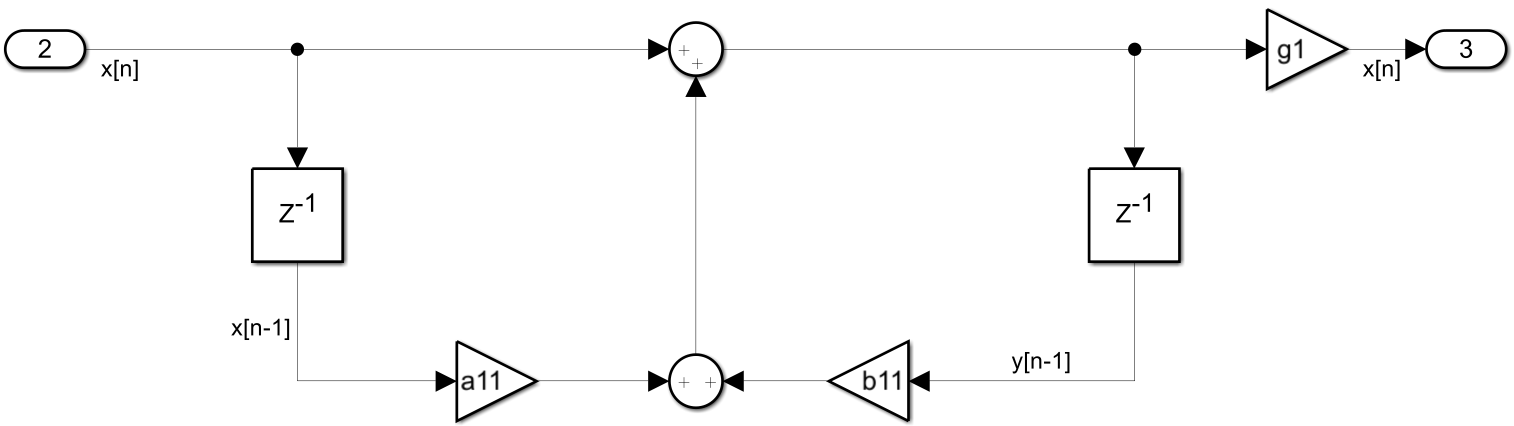
What does the 2nd order section look like?

- Each 2nd order stage is implemented as shown as a direct IIR 2nd order stage



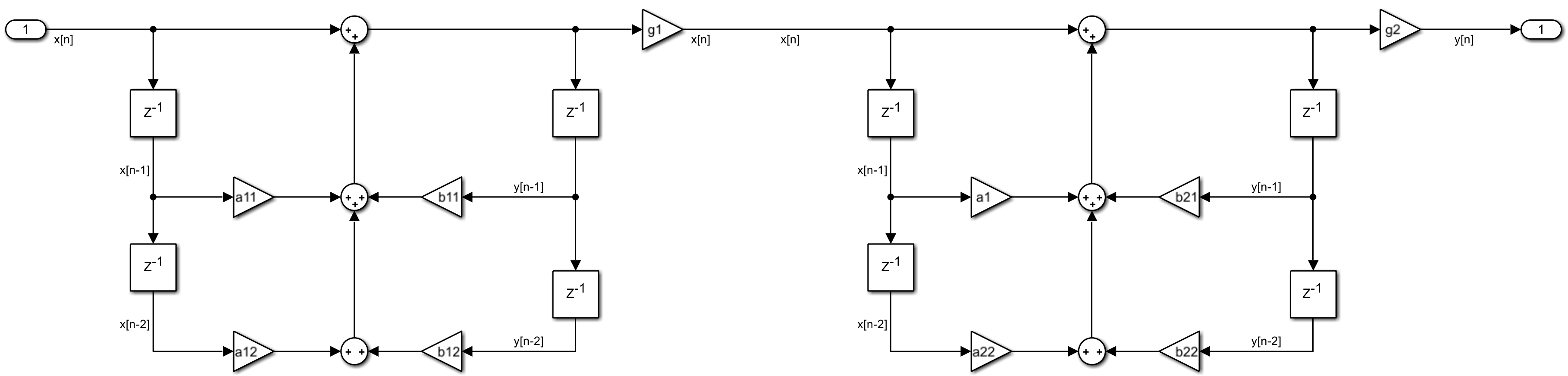
Add a 1st Order Stage for Odd Order Filters

- A first order section is similar to the second order with one section removed



Cascading the Stages

- Each 2nd Order section is cascaded together
- A first order stage is included if the filter has an odd order



Variable Declarations
Coefficient and Input Storage

```
float IIR_SOS(float xv)
{

  //  ***  Copy variable declarations from MATLAB generator to here  ****

//Filter specific variable declarations
const int numStages = 6;
static float G[numStages];
static float b[numStages][3];
static float a[numStages][3];

//  *** Stop copying MATLAB variable declarations here

  int stage;
  int i;
  static float xM0[numStages] = {0.0}, xM1[numStages] = {0.0}, xM2[numStages] = {0.0};
  static float yM0[numStages] = {0.0}, yM1[numStages] = {0.0}, yM2[numStages] = {0.0};

  float yv = 0.0;
  unsigned long startTime;
```

numStages changes with filter length
Copy from MATLAB code

Define the number of stages (sections)
Declare storage for the coefficients

Declare storage for saved input and output values $x[n], x[n-1], \ldots, y[n-1], y[n-2], \ldots$.

Defining the Recursion Coefficients

- Copy these values in from MATLAB

These two value are zero

```
//   ***  Copy variable initialization code from MATLAB generator to here  ****

// CHEBY LOW, order 11, R = 0.5, 50 BPM
G[0] = 0.0309287;
b[0][0] = 1.0000000; b[0][1] = 1.0724489; b[0][2]= 0.0000000;
a[0][0] = 1.0000000; a[0][1] =  -0.9168034; a[0][2] =  0.0000000;
G[1] = 0.0309287;
b[1][0] = 1.0000000; b[1][1] = 2.1201192; b[1][2]= 1.1253206;
a[1][0] = 1.0000000; a[1][1] =  -1.8194906; a[1][2] =  0.8472945;
G[2] = 0.0309287;
b[2][0] = 1.0000000; b[2][1] = 2.0557874; b[2][2]= 1.0608632;
a[2][0] = 1.0000000; a[2][1] =  -1.7832527; a[2][2] =  0.8667261;
G[3] = 0.0309287;
b[3][0] = 1.0000000; b[3][1] = 1.9761101; b[3][2]= 0.9810257;
a[3][0] = 1.0000000; a[3][1] =  -1.7406125; a[3][2] =  0.8966174;
G[4] = 0.0309287;
b[4][0] = 1.0000000; b[4][1] = 1.9072737; b[4][2]= 0.9120469;
a[4][0] = 1.0000000; a[4][1] =  -1.7108628; a[4][2] =  0.9342760;
G[5] = 0.0309287;
b[5][0] = 1.0000000; b[5][1] = 1.8682606; b[5][2]= 0.8729515;
a[5][0] = 1.0000000; a[5][1] =  -1.7108429; a[5][2] =  0.9772188;

//   **** Stop copying MATLAB code here   ****
```

Stage 0 coefficients (1st order)

Stage 1 coefficients (2nd order)

Stage 2 coefficients (2nd order)

Stage 3 coefficients (2nd order)

Stage 4 coefficients (2nd order)

Stage 5 coefficients (2nd order)

Implementing the Filter

- The recursion equation does all the work

```
//  Iterate over each second order stage.  For each stage shift the input data
//  buffer ( x[kk] ) by one and the output data buffer by one ( y[k] ).  Then bring in
//  a new sample xv into the buffer;
//
//  Then execute the recusive filter on the buffer
//
//  y[k] = -a[2]*y[k-2] + -a[1]*y[k-1] + g*b[0]*x[k] + b[1]*x[k-1] + b[2]*x[k-2]
//
//  Pass the output from this stage to the next stage by setting the input
//  variable to the next stage x to the outp
//
//  Repeat this for each second order stage


for (i =0; i<numStages; i++)
  {
    yM2[i] = yM1[i]; yM1[i] = yM0[i];  xM2[i] = xM1[i]; xM1[i] = xM0[i], xM0[i] = G[i]*xv;
    yv = -a[i][2]*yM2[i] - a[i][1]*yM1[i] + b[i][2]*xM2[i] + b[i][1]*xM1[i] + b[i][0]*xM0[i];
    yM0[i] = yv;
    xv = yv;
  }
```

Shift the input and output values in the array

$$x[n-1] = x[n], x[n-2] = x[n-1]$$
$$y[n-1] = y[n], y[n-2] = y[n-1]$$

$$y[n] = -a_{12}y[m-2] - a_{11}y[m-1] + b_{12}y[m-2] + b_{11}y[m-1]$$

 Signs are different for a coefficients. It's just how MATLAB defines the signs

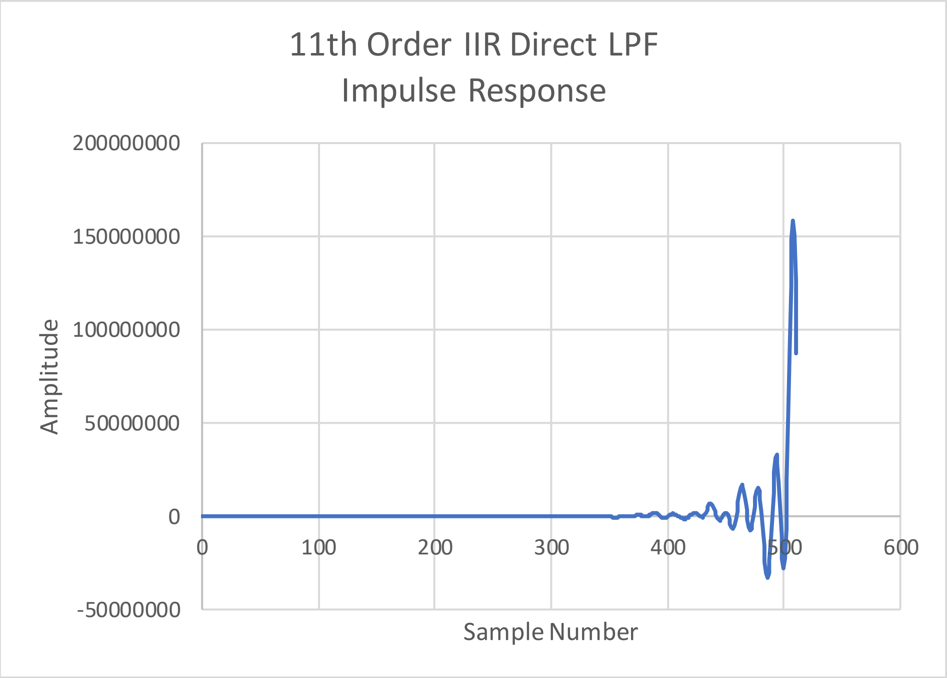
11^{th} Order SOS Filter Demonstration

- Use MATLAB to create the coefficients of the11th order LPF. Fcorner = 50 bpm
- Demonstrate on the Arduino Platform

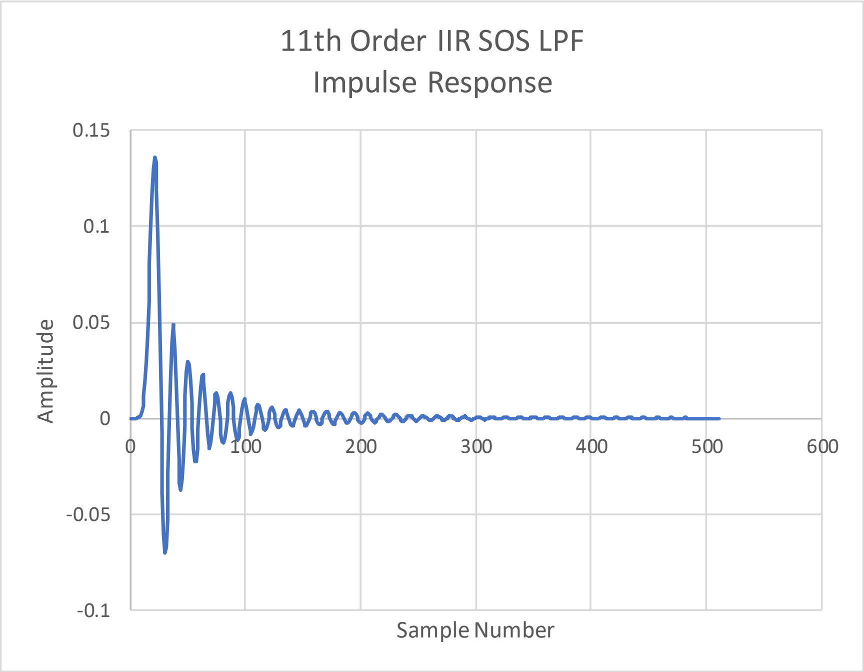
Impulse Response Comparison

- IIR Direct and IIR SOS Comparison

IIR Direct Unstable

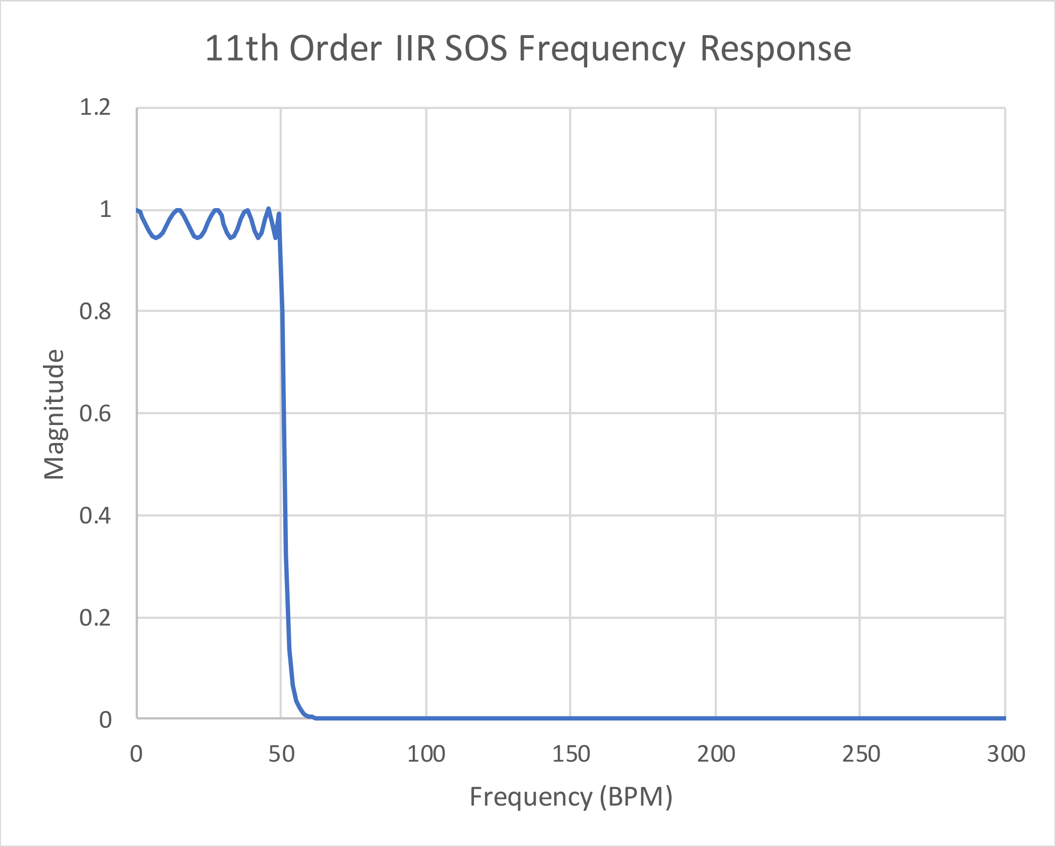


IIR SOS Stable



IIR SOS Frequency Response

- Frequency Response is well behaved



IIR SOS Filter ICP

- Design a 9th order Chebyshev filter using cascaded second order stages using MATLAB design script

- 9th order
- R = 0.5 dB
- F_{corner} = 12 BPM

- How many total sections will there be?
- How many 2nd order sections will there be?
- How many 1st order sections?

IIR SOS Filter ICP

NO PRINT

- For an 9th order filter, there will be 5 sections

```
**** C Code for SOS Filter -- Copy to Arduino IDE ****

//Filter specific variable declarations
const int numStages = 5;
static float G[numStages];
static float b[numStages][3];
static float a[numStages][3];
```

Number of stages

Stage Gains

Numerator Coefficients

Denominator Coefficients

IIR SOS Filter ICP

NO PRINT

- 5 sections
  - 4 - 2nd order sections (8-poles)
  - 1 - 1st order section (1-pole)

Stages

Stage Gains

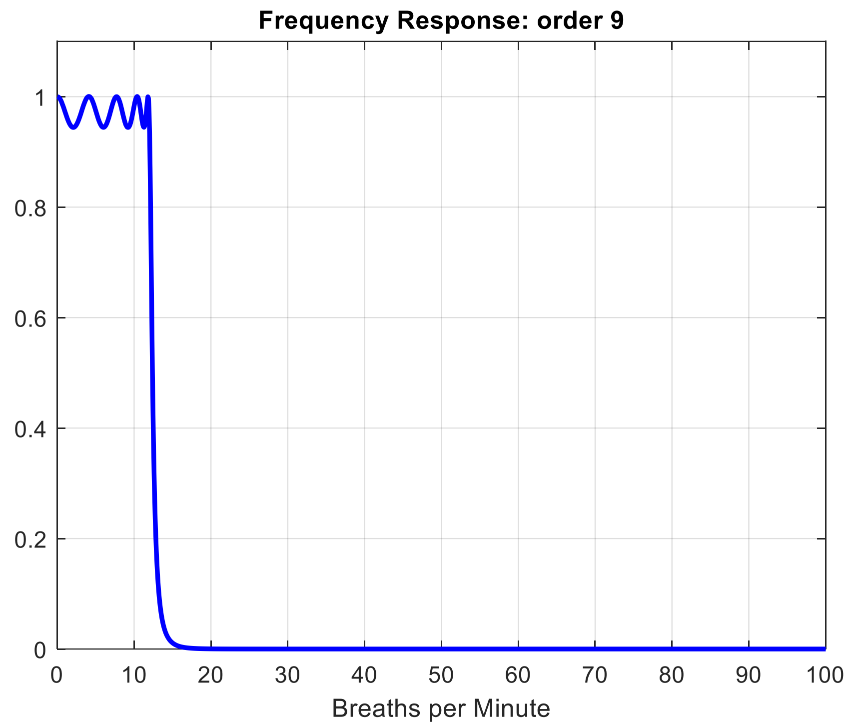
Numerator Coefficients

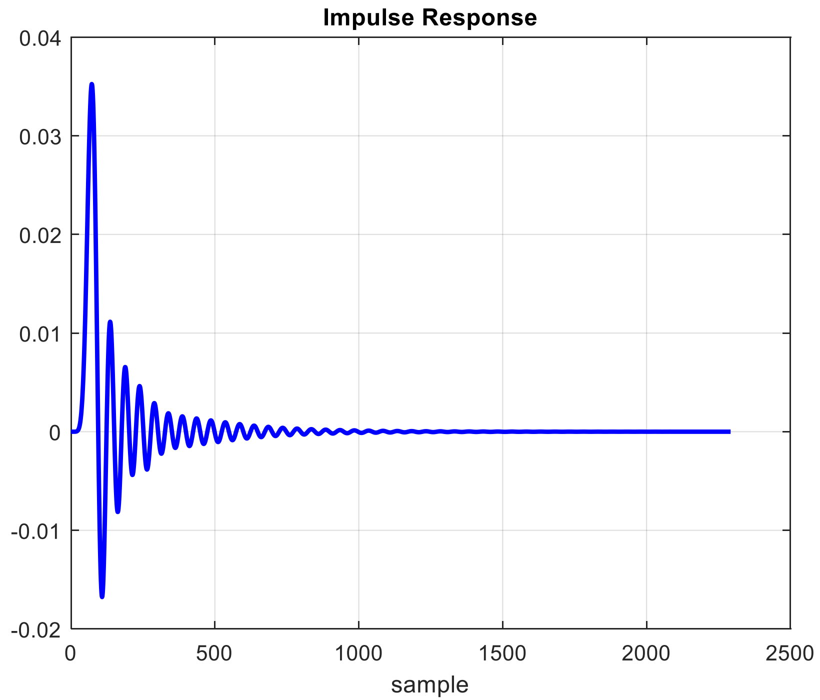
Denominator Coefficients

```
// CHEBY LOW, order 9, R = 0.5, 12 BPM
G[0] = 0.0027572;
b[0][0] = 1.0000000; b[0][1] = 1.0346697; b[0][2]= 0.0000000;
a[0][0] = 1.0000000; a[0][1] =  -0.9753691; a[0][2] =  0.0(
G[1] = 0.0027572;
b[1][0] = 1.0000000; b[1][1] = 2.0524836; b[1][2]= 1.0536768;
a[1][0] = 1.0000000; a[1][1] =  -1.9517205; a[1][2] =  (
G[2] = 0.0027572;
b[2][0] = 1.0000000; b[2][1] = 2.0108794; b[2][2]= 1.0120511;
a[2][0] = 1.0000000; a[2][1] =  -1.9555455; a[2][2] =  0.9625653;
G[3] = 0.0027572;
b[3][0] = 1.0000000; b[3][1] = 1.9653564; b[3][2]= 0.9665046;
a[3][0] = 1.0000000; a[3][1] =  -1.9631027; a[3][2] =  0.9754065;
G[4] = 0.0027572;
b[4][0] = 1.0000000; b[4][1] = 1.9366110; b[4][2]= 0.9377444;
a[4][0] = 1.0000000; a[4][1] =  -1.9755549; a[4][2] =  0.9914027;
```

NO PRINT

Frequency and Impulse Response

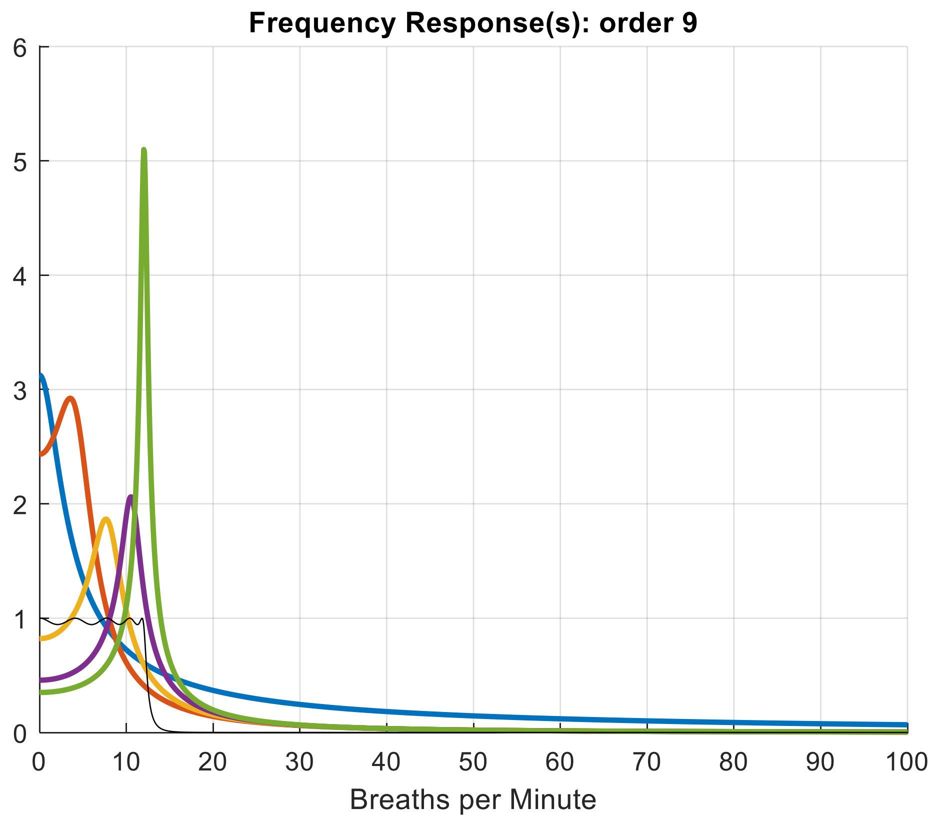




NO PRINT

Individual Stage Response

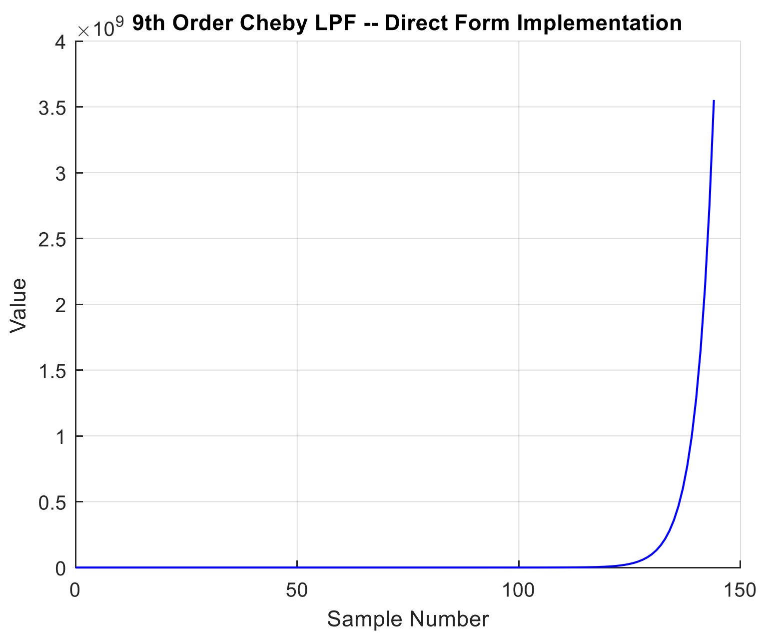
- Stage response cascaded together

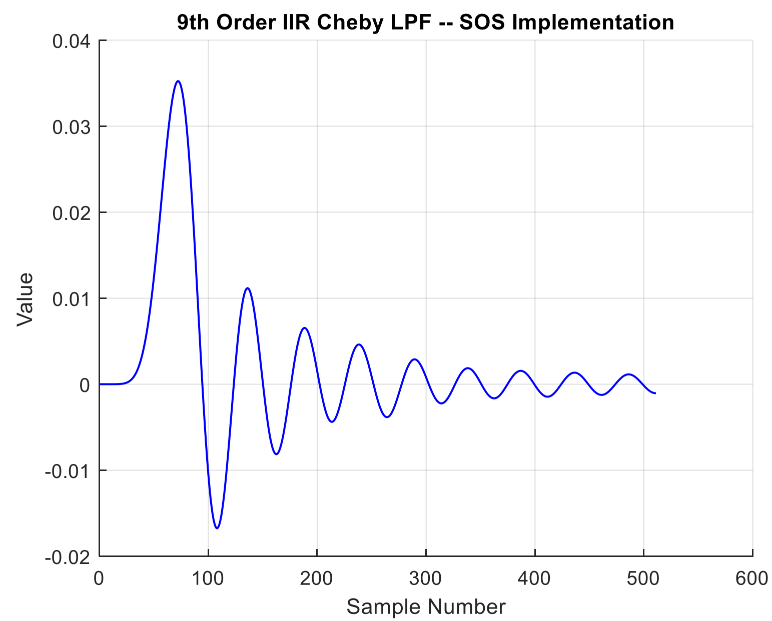


9^{th} Order Chebyshev LPF Arduino Implementation

NO PRINT

- SOS Implementation has a stable impulse response





Summary

- Reviewed higher order IIR Direct Filters
- Demonstrated stability issues with the higher order filter
- Introduced 2nd order stages (SOS) approach
- Discussed Implementation and demonstrated results