IIR Filters in FPGAs

Dominic Meads

Outline

Part 1

- Prerequisties to know before the video
- IIR filter structures
- Biquads: how/why they are used
- MATLAB code to generate filter coefficients
- Simple IIR biquad in Verilog
- Test results of the simple IIR filter

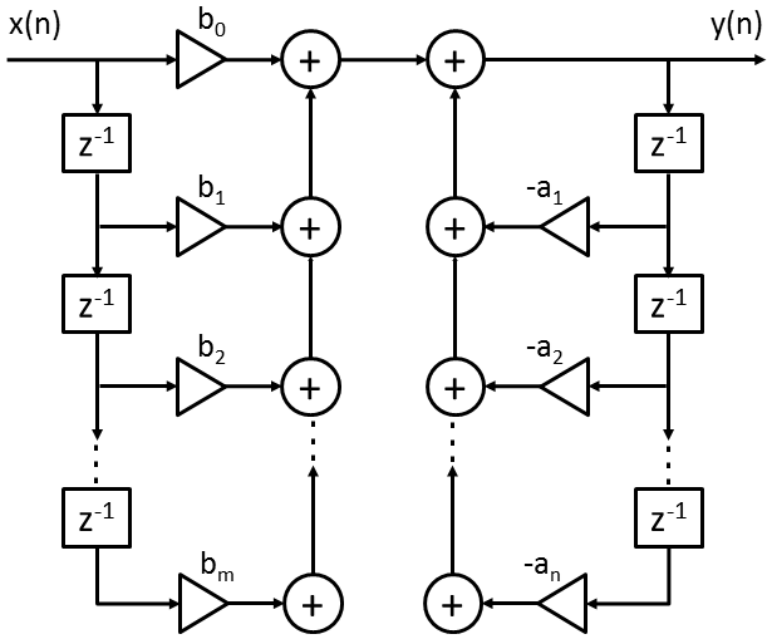
Parts 2 and 3

- Modify biquad for use with AXI-Stream
- Cascade multiple biquads to create a higher order filter

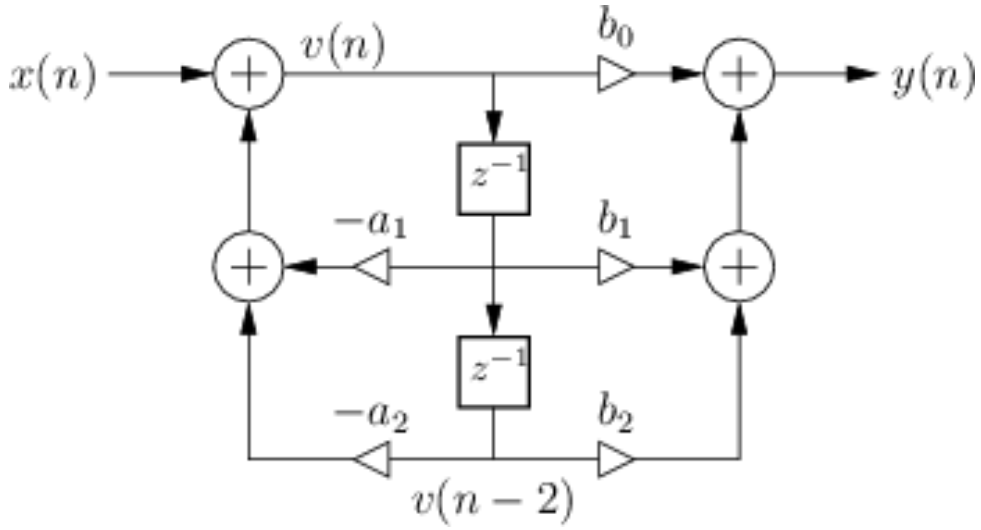
Prerequisites

- Basic signal processing (FIR vs IIR, impulse response, etc.)
- Knowledge of Verilog
- Access to MATLAB if desired
- Access to FPGA simulator of your choice (I’m using AMD Vivado)

IIR Filter Structures



Direct Form-I



Direct Form-II

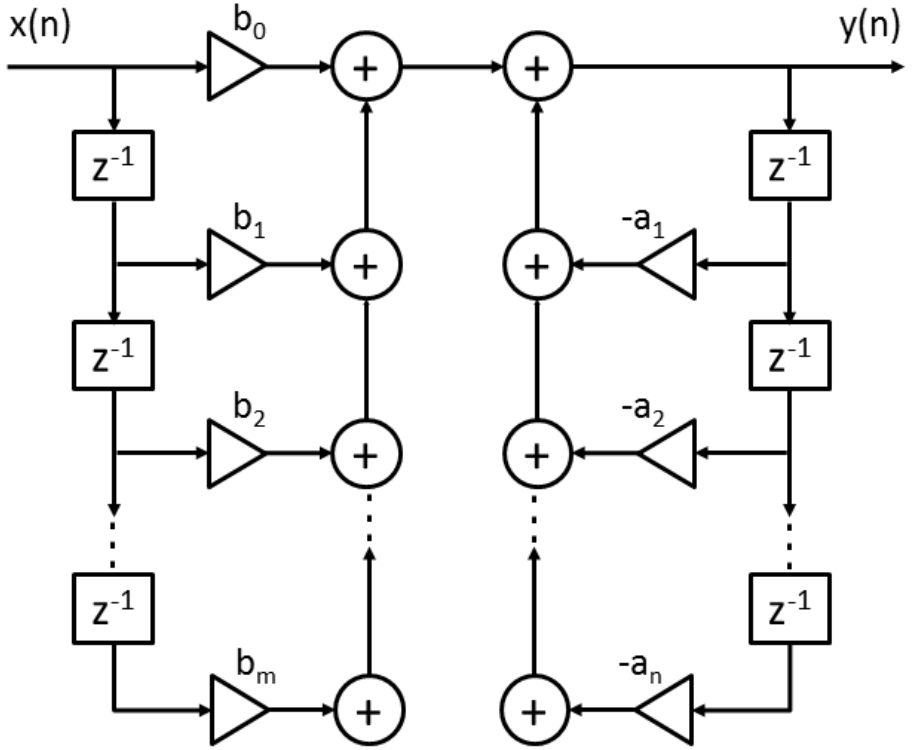
Focus: Direct Form-I

Why?

"It is a very useful property of the direct-form I implementation that it cannot overflow internally in two's complement fixed-point arithmetic:

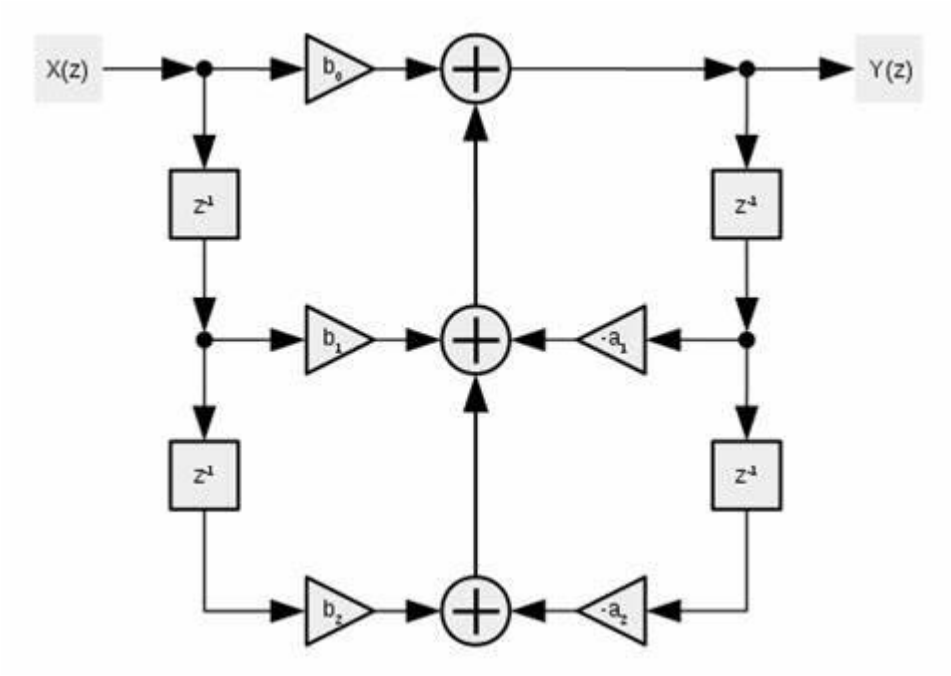
As long as the output signal is in range, the filter will be free of numerical overflow."

Source: Direct-Form I



IIR Filters: Second-Order Sections

Second-order sections are also known as biquads (short for bi-quadratic).



Direct Form-I Biquad

"the cascaded-biquad design is less sensitive to coefficient quantization than a single high-order IIR, particularly for lower cut-off frequencies"

Design IIR Filters Using Cascaded Biquads - Neil Robertson

IIR Filters: Second-Order Sections

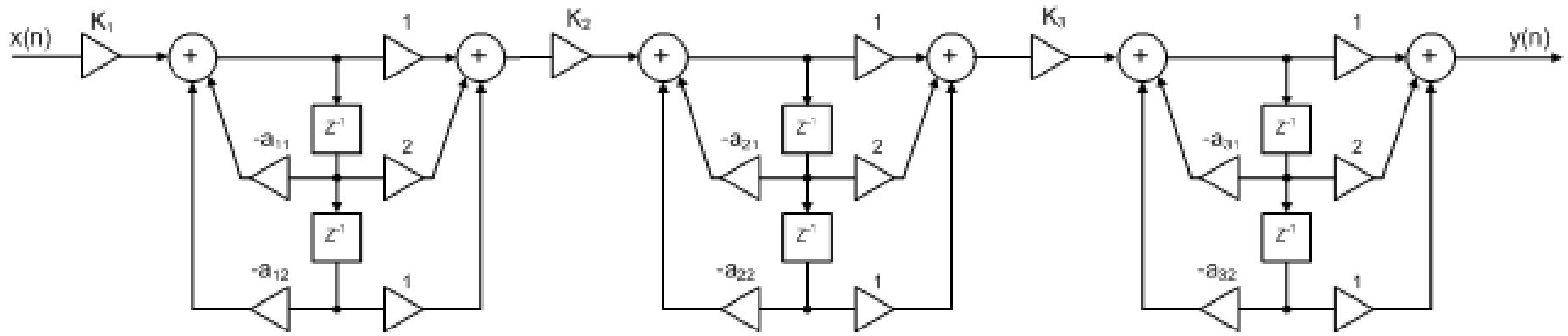
The term ``biquad" is short for ``bi-quadratic", and is a common name for a two-pole, two-zero digital filter. The *transfer function* of the biquad can be defined as

$$H(z) = g\frac{1 + \beta_1 z^{-1} + \beta_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \tag{B.8}$$

The BiQuad Section | Introduction to Digital Filters

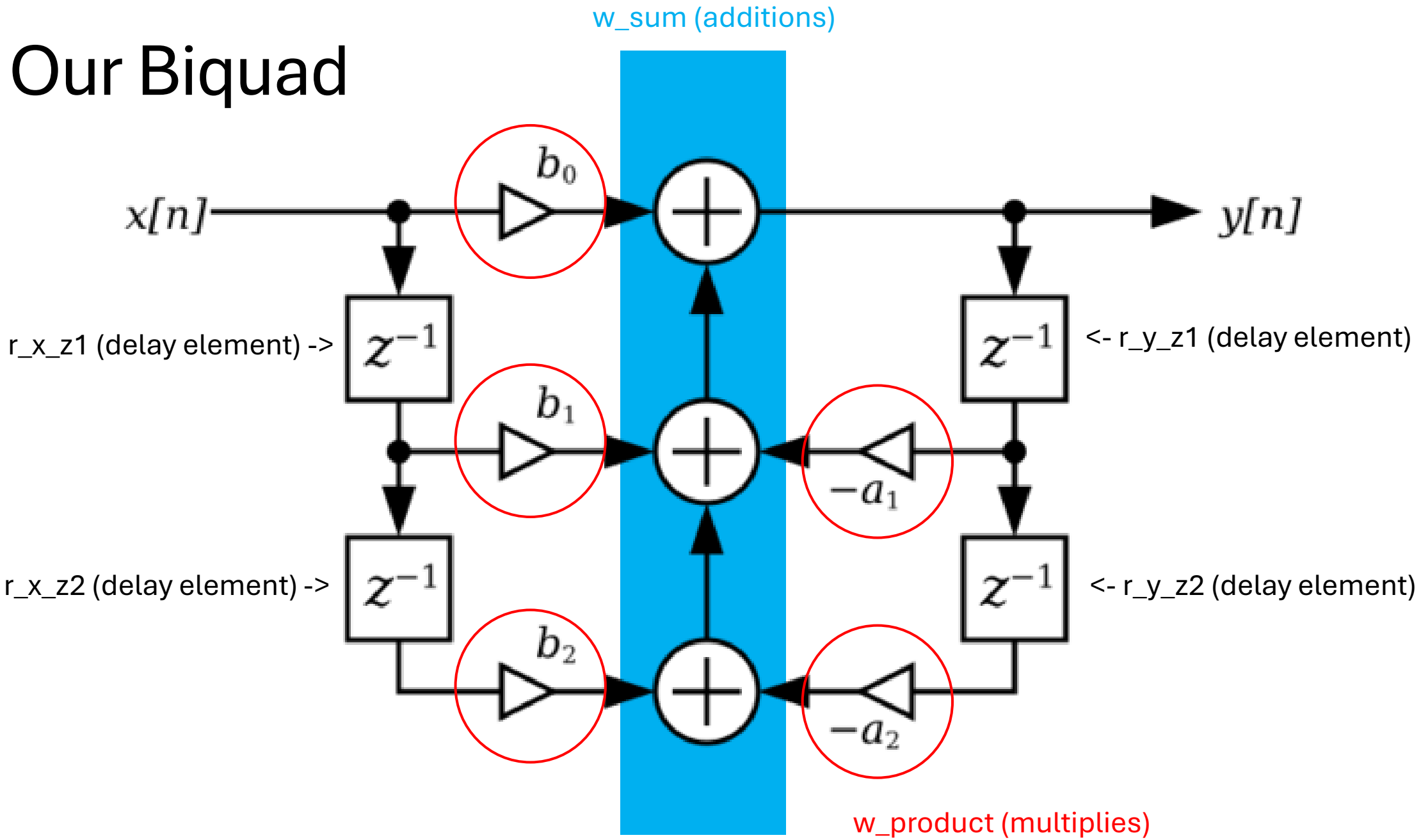
IIR Filters: Second-Order Sections

Cascade three “biquads” to create a 6th order IIR filter that is less sensitive to coefficient rounding error



Note: Picture is not Direct Form-1

Our Biquad


w_sum (additions)
x[n]
y[n]
b_0
b_1
b_2
$-a_1$
$-a_2$
z^{-1}
z^{-1}
z^{-1}
z^{-1}
r_x_z1 (delay element) ->
r_x_z2 (delay element) ->
<- r_y_z1 (delay element)
<- r_y_z2 (delay element)
w_product (multiplies)

Issues with the code

- No standard interface (AXI or similar, each sample loaded on system clock edge)
- What about increasing multiplier width for less coefficient rounding?
- No parameterization
- Poor pipelining (max around 50 MHz)