International Master of Science in Electrical Engineering

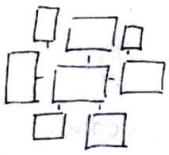
Embedded Signal Processing Systems

Real-Time Implementation of Digital Filters on Cortex-M Microcontrollers

Prof. Dr. C. Jakob

Hochschule Darmstadt
University of Applied Science Darmstadt

h_da



ESPS Laboratory #1 – Contd.

Version 1.4, 21 May 2025, created using WordTEX

Submission Deadline: -

For questions, comments or suggestions, please contact me by email:
christian.jakob@h-da.de

Darmstadt, May 2025

Digital Resonance Filter

Second Order IIR Filter

A digital resonance filter is a second-order IIR structure characterized by a pair of complex-conjugate poles located **within** the unit circle. The resonant frequency is determined by the angular position of the poles, while the bandwidth is controlled by their radial distance from the unit circle. The transfer function of a digital resonance filter with two real zeros at $z = +/-1$ is given by:

$$H(z) = \frac{b_0(z+1)(z-1)}{(z - ae^{j\omega_0})(z - ae^{-j\omega_0})}$$

with:

- b_0 being the gain factor used to normalize the peak magnitude to 1.
- a for controlling the bandwidth of the filter $0 < a < 1$.
- ω_0 the resonance frequency.

Hint: Use the MATLAB script provided to gain a clear understanding of the digital resonator in both time and frequency domain. Discuss the effect of placing two complex-conjugate poles in close proximity to the unit circle. How do the two zeros influence the filter's frequency response

STM32G4 related tasks

Implement the digital resonance filter in **floating-point arithmetic** within the **Timer Interrupt Service Routine (48 kHz ISR/TIM6)** on the STM32G4 MCU.

a) **Input Signal Source:**
   - Use a predefined array of integer values as test input, such as 12-bit unsigned samples that simulate actual ADC output. In this case, generate a dual-tone test signal with suitable frequencies. Within the ISR, retrieve the next value from the array and convert it to floating-point format for further processing.

b) **Filter Implementation:**
   - Implement the resonator filter in C using floating-point arithmetic (float only) to simplify the implementation and avoid rounding artifacts. Fixed-point arithmetic will be addressed in future lab sessions.

c) **Output Conversion and DAC Handling:**
   - After filtering, convert the floating-point output back to the **integer domain** for DAC output (e.g., scale and cast to 12-bit unsigned integer). Output **both the raw input signal** and the **filtered output** to **two separate DAC channels**.

d) **Timing and Execution:**
   - Configure the **hardware timer TIM6** to trigger the ISR at a fixed sampling rate of **48 kHz**. Ensure that the ISR executes within the sampling period without overruns.

e) **Verification:**
   - Use an oscilloscope to observe and compare the DAC outputs of the input as well as the filtered signals.