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

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

Introduction

This laboratory exercise focuses on the real-time implementation of recursive digital filters on modern ARM Cortex-M microcontrollers. The target platform is the NUCLEO-G474RE development board, featuring an STM32G4 device from STMicroelectronics.

The objective is to design, analyze, and implement sample-based digital filters that operate in real time within a timer-driven Interrupt Service Routine (ISR) at a fixed sampling frequency of 48 kHz. The focus is on the following types of filters:

- First-order Exponential Moving Average (EMA) filters in Low-and High-pass configuration

Both filters are implemented using floating-point arithmetic.

Learning Objectives

By the end of this lab, you will be able to:

- Design and analyze first- and second-order recursive filters in MATLAB
- Implement real-time digital filters in C on an STM32G4 MCUusing a timer interrupt based processing scheme.
- Generate and manage test signals using lookup tables (LUTs)
- Convert between integer and floating-point data formats
- Output signals using the DAC and verify the filter's behavior via oscilloscope
- Measure and analyze execution time to validate real-time performance

Preparation Steps

To complete this laboratory exercise, the following software tools must be installed.

STM32CubeIDE (latest version)

- Register on st.com
- Integrated development environment for STM32 microcontrollers, which Includes compiler, debugger, and project configuration tools

PicoScope Software (latest version)

- Required for the use with Pico Technology USB oscilloscopes

In addition, please download

- ST NUCLEO-G474RE User Manual **UM2505** for pinout documentation
- STM32G474RE datasheet

Methodology

The following section describes a structured approach used to develop and test digital filters, covering simulation, implementation, and real-time evaluation.

Filter Design and Simulation using MATLAB

- Each filter is mathematically modeled and simulated using MATLAB.
- Magnitude and phase responses are analyzed.
- Sensitivities to design parameters (e.g., cutoff frequency, quality factor) are evaluated.

STM32G4 Real-Time Implementation

- A hardware timer (**TIM6 for simplicity**) triggers the ISR at 48 kHz.
- At each interrupt, one input sample is processed by the filter.
- Filter output and timing diagnostics are analysed.

Test Signal Generation

- Input signals include:
  - Dual-tone signals
  - Sine waves with additive white noise
- All signals are precomputed at startup and stored in fixed-size lookup tables (LUTs).
- LUTs replicate typical ADC output formats (e.g., 12-bit unsigned integers).

Data Conversion

- LUT values are converted to floating-point format on-the-fly before filtering.
- Filter outputs are then reconverted to integer format for DAC output.

Signal Output and Evaluation

- Both filtered and unfiltered signals are output simultaneously via the STM32G4 dual-channel internal DAC.
- An oscilloscope is used to visually compare both waveforms.
- Filter behavior, such as the attenuation for certain signal components, is observed in real time.

Performance Analysis

- Filter execution time is measured using hardware cycle counters or GPIO toggling.
- If the processing time exceeds the ISR interval (in out case: 20.83 µs), real-time operation is compromised.
- This offers practical insights into the computational cost of each filter.

Basic STM32G4 Framework

This section shortly describes how to configure an STM32 project in STM32CubeIDE for executing real-time digital filters using a 48 kHz timer interrupt. The DAC outputs the processed and reference signals.

1. Create a New STM32 Project

1. Open **STM32CubeIDE.**
2. Go to **File > New > STM32 Project.**
3. In the **Board Selector**, choose NUCLEO-G474RE (or select the specific STM32G4 MCU from the MCU selector).
4. Click **Next**, name your project (e.g., ESPSLab1), and choose **"Empty"** as project template (i.e., **Do not initialize all peripherals with default mode**).
5. Finish and let the IDE generate the project structure.

2. Configure TIM6 as Base Timer for 48 kHz Interrupt

1. In the **.ioc configuration file**, open the **Peripherals > Timers > TIM6.**
2. Enable **TIM6** in **"Internal Clock"** mode.
3. In **Configuration > Parameter Settings**, set:
   - **Prescaler** and **Auto-Reload Register (ARR)** to generate exactly **48 kHz update events**:
     - For example, if your system clock is 170 MHz:
       - Prescaler: 170 - 1 → gives 1 MHz timer clock
       - ARR: 1000000 / 48000 - 1 = 20.83 ⇒ 20
     - This results in a 48 kHz update event. Other configuration are also possible.
4. In **NVIC Settings, enable the TIM6 global interrupt.**

3. Configure the DAC (Dual-Channel)

1. Under **Peripherals > DAC1**, enable **Channel 1** and **Channel 2.**
2. In the **DAC configuration**:
   - Mode: **Normal (Non-triggered)** (we will write manually in the ISR)
   - Output buffer: **Enable** (default)
3. In the **GPIO Configuration,** check that both DAC channels are mapped to correct analog pins:
   - **PA4** → DAC_OUT1
   - **PA5** → DAC_OUT2

4. Clock Configuration (Verify Timing)

1. Open the **Clock Configuration** tab.
2. Ensure **HCLK = 170 MHz** (default on G474RE Nucleo).
3. Confirm the timer clock input is correctly derived from APB1 Timer clock (TIM6 is on APB1).

5. Generate Initialization Code

1. Click the **"Generate Code"** in the Project section.
2. Make sure to understand the structure of the STM32Cube-generated code. Moreover, understand how peripherals are initialized and how the HAL library manages low-level operations.

6. Enable TIM6 Interrupt and Start DAC in main.c

1. Examine the `main.c` source file. Start with the initialization section in `main()`:

```c
HAL_TIM_Base_Start_IT(&htim6);          // Start TIM6 with interrupts

HAL_DAC_Start(&hdac1, DAC_CHANNEL_1);   // Enable DAC channel 1
HAL_DAC_Start(&hdac1, DAC_CHANNEL_2);   // Enable DAC channel 2
```

[C]

7. ISR Handler

2. Scroll down to stm32g4xx_it.c, and locate the TIM6 ISR handler. This code segment is used to implement the actual filter kernel.

```c
void TIM6_DAC_IRQHandler(void)
{

}
```

[C]

Within, the ISR, delete any kind of "unnecessary HAL code"

8. Test Setup Verification #1

Start with verifying the basic system setup to ensure that the timer interrupt is being triggered at 48 kHz. To do this, configure a free GPIO pin as a digital output and toggle its state inside the timer ISR. The resulting square wave can be observed using an oscilloscope to confirm the correct interrupt frequency.

9. Test Setup Verification #2

In the next step, a sine wave consisting of 256 samples should be generated and stored in a lookup table. This signal is continuously read out within the ISR in a circular manner and sent to the DAC. As a result, a continuous analog sine wave should be observable at the DAC output for verification using an oscilloscope.

Low-Pass Exponential Moving Average Filter

Low-Pass Configuration

The exponential moving average (EMA) filter in **low-pass configuration** is defined by the following difference equation:

$$y(n) = \alpha x(n) + (1 - \alpha)y(n - 1)$$

where:

- $y(n)$ is the filter output at time step n, $x(n)$ is the input signal at time step n and α is the smoothing factor in the range $0 < \alpha < 1$

EMA Low-pass filter - MATLAB related

The intention of the MATLAB tasks is to develop a clear understanding of the EMA filter's behavior in both the time and frequency domains before implementing it on the embedded target.

a) Implement the EMA low-pass filter in MATLAB using the difference equation above: Design your function to accept the input signal $x(n)$, the smoothing factor α, and return the filtered signal $y(n)$. Check which representation is required for using MATLAB functions such as `freqz`, `filter`, or similar.

b) For a given set of α values (e.g., 0.1, 0.3, 0.5, 0.9), compute and plot the **magnitude** and **phase** response of the filter: Use MATLAB's `freqz` function to visualize the filter behavior in the frequency domain. Label axes and clearly indicate the -3 dB cutoff frequency for each case.

c) Generate a dual-tone test signal composed of two sinusoidal components:

- One frequency in the **passband** of the EMA filter (e.g., 500 Hz)
- One frequency in the **stopband** (e.g., 8,000 Hz)

d) The signal should be sampled at **48 kHz** and last for at least 0.1 seconds. Filter the signal using the implemented EMA filter. Plot the input and output signals in the time domain.

e) Discuss the observed filtering effect: Which frequency component is preserved, and which is attenuated?

f) Investigate the effect of varying α on the filter behavior: Describe how increasing or decreasing α affects the cutoff frequency and the transient response.

STM32G4 related

Implement the exponential moving average (EMA) low-pass filter in **floating-point arithmetic** within the **Timer Interrupt Service Routine (ISR)** on the STM32G4 MCU.

g) Input Signal Source:

- Use a predefined array of **integer test data set** (e.g., 12-bit unsigned values simulating the actual ADC samples).
- Within the ISR, read the next value from the array and convert it to **floating-point format** for processing.

h) Filter Implementation:

- Implement the EMA filter in C. Use **floating-point operations** (float) to maintain accuracy and prevent rounding artifacts during computation.

i) Output Conversion and DAC Handling:

- After filtering, convert the floating-point output back to the **integer domain** suitable 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.**

j) 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.

k) Verification:

- Use an oscilloscope to observe and compare the DAC outputs of the input and filtered signals.

High-Pass Exponential Moving Average Filter

High-Pass Configuration

An Exponential Moving Average (EMA) high-pass filter is described by the following difference equation:

$$y_{HP}(n) = (1 - \alpha)(x(n) - y_{LP}(n-1))$$

where:

- $y(n)$ is the filter output at time step n, $x(n)$ is the input signal at time step n and α is the smoothing factor in the range $0 < \alpha < 1$

This equation is derived by subtracting the output of the EMA low-pass filter from the input signal:

$$y_{HP}(n) = x(n) - y_{LP}(n)$$

Substituting the low-pass EMA equation into the equation above and simplifying leads to the high-pass EMA form shown initially. **Please note: For implementation on the STM32G4, it is practical and efficient to first compute the low-pass EMA filter and then derive the high-pass output by subtracting it from the input. This approach simplifies the structure and minimizes computational overhead in the interrupt routine.**

EMA High-pass filter - MATLAB related

The intention of the MATLAB tasks is to develop a clear understanding of the EMA filter's behavior in both the time and frequency domains before implementing it on the embedded target.

h) Implement the EMA low-pass filter in MATLAB using the difference equation above: Design your function to accept the input signal $x(n)$, the smoothing factor α, and return the filtered signal $y(n)$. Check which representation is required for using MATLAB functions such as `freqz`, `filter`, or similar.

i) For a given set of α values (e.g., 0.1, 0.3, 0.5, 0.9), compute and plot the **magnitude** and **phase** response of the filter: Use MATLAB's `freqz` function to visualize the filter behavior in the frequency domain. Label axes and clearly indicate the -3 dB cutoff frequency for each case.

j) Generate a dual-tone test signal composed of two sinusoidal components:

- One frequency in the **passband** of the EMA filter (e.g., 500 Hz)
- One frequency in the **stopband** (e.g., 8,000 Hz)

k) The signal should be sampled at **48 kHz** and last for at least 0.1 **seconds**. Filter the signal using the implemented EMA filter. Plot the input and output signals in the time domain.

l) Discuss the observed filtering effect: Which frequency component is preserved, and which is attenuated?

m) Investigate the effect of varying α on the filter behavior: Describe how increasing or decreasing α affects the cutoff frequency and the transient response.

STM32G4 related

Implement the exponential moving average (EMA) low-pass filter in **floating-point arithmetic** within the **Timer Interrupt Service Routine (ISR)** on the STM32G4 MCU.

n) **Input Signal Source:**

- Use a predefined array of **integer test data set** (e.g., 12-bit unsigned values simulating the actual ADC samples).
- Within the ISR, read the next value from the array and convert it to **floating-point format** for processing.

i) **Filter Implementation:**

- Implement the EMA filter in C. Use **floating-point operations** (float) to maintain accuracy and prevent rounding artifacts during computation.

j) **Output Conversion and DAC Handling:**

- After filtering, convert the floating-point output back to the **integer domain** suitable 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**.

k) **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.

l) **Verification:**

- Use an oscilloscope to observe and compare the DAC outputs of the input and filtered signals.