ASN Filter Designer ANSI C SDK framework performance benchmarks on 32-bit embedded platforms

Application note (ASN-AN028) **August 2021 (Rev 6)**

Synopsis

Digital Filters are one of the fundamental blocks in digital signal processing. Applications of this type of technology, include removing powerline interference from biomedical ECG, EMG data, eliminating any glitches and unwanted artefacts from IoT sensor data, the list goes on.

Digital filters are divided into the following two categories:

- **Infinite impulse response (IIR)**: generally chosen for applications where linear phase is not too important and memory is limited, as such they have a lower overall implementation cost than that of FIR filters. They have been widely deployed in audio equalisation, biomedical sensor signal processing, IoT/IIoT smart sensors and high-speed telecommunication/RF applications.

- **Finite impulse response (FIR)**: generally chosen for applications where linear phase is important and a decent amount of memory and computational performance are available. They have a widely deployed in audio and biomedical signal enhancement applications. Their architecture ensures that they never become unstable for any type of input signal, which gives them a distinct advantage over the IIR.

Suffice to say, each type of filter has its own advantages and disadvantages, and choosing the right type of filter is highly dependent on the application, target price, processor performance and available tooling. For a more detailed explanation, the reader is referred to the following article, that provides an excellent overview of differences between FIR and IIR filters.

In this application note, we show the benchmark performance of ASN's C SDK on a variety of 32-bit embedded platforms, including Arm® Cortex®-M microcontrollers and the very popular ESP32-WROOM-32 SoC from Espressif. As Arm has its own DSP library, we will also compare the performance of ASN's C SDK with Arm's optimised CMSIS-DSP library. As an example, we will design an 8th order lowpass Biquad IIR filter designed using the ASN Filter Designer. The ASN Filter Designer software will then be used to generate all of the necessary C code required for benchmarking the performance of the filter.

The following microcontrollers were used for the benchmark testing:

1. STM32F103C8 – Arm Cortex-M3
2. STM32F469NI – Arm Cortex-M4F
3. STM32F769NI – Arm Cortex-M7F
4. ESP32-WROOM-32 – Espressif (Dual-core Xtensa 32-bit LX6)

The F suffix signifies that the device has an FPU (floating point unit).

ASN Filter Designer's new **ANSI C SDK framework** provides developers with a comprehensive ANSI C code base that can be used to deploy developed filtering applications to any C/C++ embedded platform. This agnostic feature, allows product developers to quickly integrate digital filtering functionality into their existing designs with the minimum amount of effort.

In this application note, we provide the reader with benchmarks on microcontrollers such as the STM32 and ESP32 for an 8th order IIR lowpass filter. Using the ASN Filter Designer software tool, **you can get working C code in few steps** which you can use along with the supporting SDK for your project. This allows developers to directly deploy their filtering application from within the tool to any **STM32, Arduino, ESP32, Beagle Bone platform** for direct use in their application.

The Arm CMSIS (Cortex Microcontroller Software Interface Standard) DSP software framework developed by Arm Ltd, that provides a rich collection DSP/ML function (including various mathematical functions, such as sine and cosine; IIR/FIR filtering functions, complex math functions, and data types) developed by Arm Ltd. that have been optimised for their range of Cortex-M processor cores. This library has the advantage of utilising hardware acceleration, such as SIMD operations, but is limited in its support for different types of filters, typically required for many IoT applications.

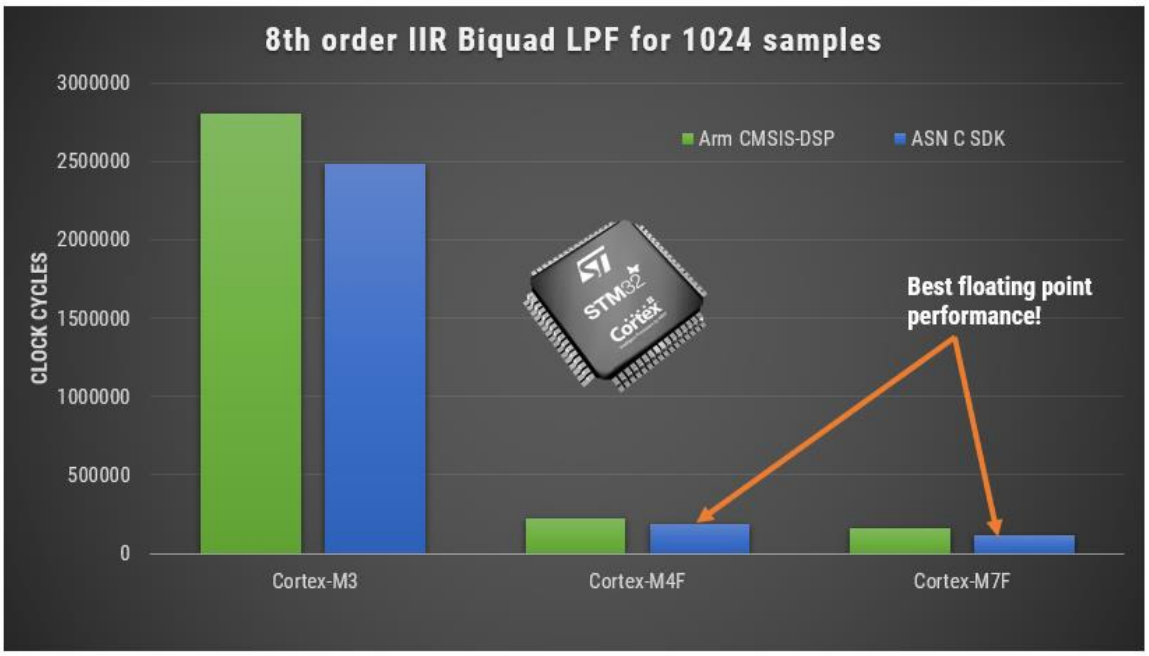


Figure 1 – overview of libraries (ASN and Arm) performance on different Arm cores

The following sections will describe these results in detail.

1. Advantages of using the ASN-DSP SDK framework

1. A developer can now develop, test and deploy a complete DSP filtering application within the ASN Filter Designer within a few hours. This is very different from a traditional R&D approach that assigns a team of developers for several days in order to achieve the same level of accuracy required for the application.

2. Open source and agnostic code base: In order to allow developers to get the maximum performance for their applications, the ASN-DSP SDK is provided as open source and is written in ANSI C. This means that any embedded processor and any level of compiler optimisation can be used.

3. The ASN-DSP SDK has full support for **complex filters**, which is seldom supported by many vendors due to their conceptual difficulty. This now allows developers to quickly design and implement solutions for I4.0 applications, such as Coriolis flow metering and powerline harmonic frequency tracking applications.

4. Memory size required for the ASN-DSP SDK is relativity lower than other standard DSP libraries, which makes the ASN-DSP SDK extremely suitable for microcontrollers that have memory constrains.

5. Using the ASN Filter Designer's signal analyser tool, developers now can test the performance, accuracy and assess the frequency response of their designed filter and get optimised C code which they can directly use in their application.

6. The SDK also supports some **extra filtering functions**, such as: a median filter, a moving average filter, all-pass, single section IIR filters, a TKEO biomedical filter, and various non-linear functions, including RMS, Abs, Log and Sqrt. These functions form the filter cascade within the tool, and can be used to build signal processing applications, such as EMG and ECG biomedical applications.

7. The ASN-DSP SDK supports both **single and double precision floating point arithmetic**, providing excellent numerical accuracy and wide dynamic range. The library is unique in the sense that it supports double precision arithmetic, which although is not the most optimal for microcontrollers, allows for the implementation of high-fidelity filtering applications.

2. Filter specification used for benchmarking

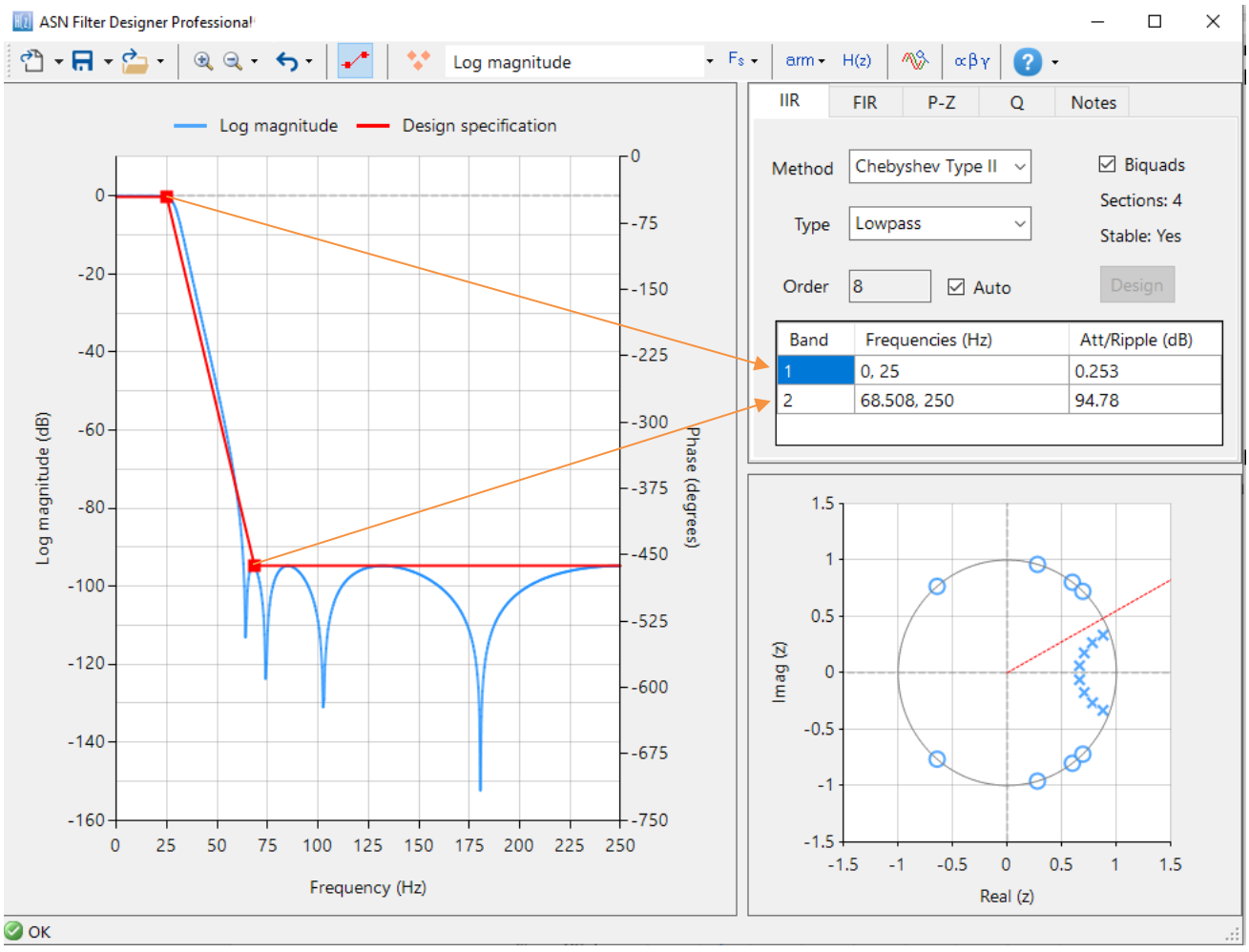
ASN Filter Designer provides engineers with an easy to use, intuitive graphical design development platform for both IIR and FIR digital filter design. The tool's real-time design paradigm makes use of *graphical design markers*, allowing designers to simply draw and modify their magnitude frequency response requirements in real-time while allowing the tool to automatically fill in the exact specifications for them.

The following technical specification was used for designing the IIR filter used for benchmarking:

Fs:	500Hz
Passband frequency:	0-25Hz
Type:	Lowpass
Method:	Chebyshev Type II
Stopband attenuation @ 50Hz:	≥ 40 dB
Passband ripple:	< 0.001dB
Order:	8

Graphically entering the specifications into the ASN Filter Designer, and fine tuning the design marker positions, the tool automatically designs the filter, automatically choosing the required filter order, and in essence - automatically producing the filter's exact technical specification!

An overview of the UI meeting this specification is shown below:



For more information related to designing IIR/FIR filters with the ASN Filter Designer, please visit the following link

3. Benchmarks on real hardware

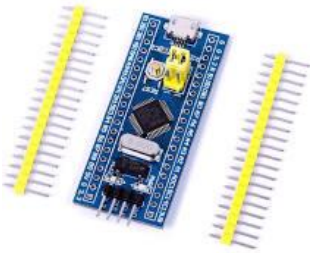
In order to objectively benchmark the libraries, the Arm CMSIS-DSP and ASN C SDK libraries were tested on microcontrollers with different Arm Cortex-M cores and an ESP-32 SoC to evaluate their performance. The base test was to run 1024 words through the filter and measure the number of CPU cycles it takes to complete the whole filtering operation. In all cases, single-precision floating-point arithmetic was used and the CMSIS-DSP library's `BLOCKSIZE=32`.

Summary of development boards used

a) STM32F769NI: An Arm Cortex-M7 with FPU (M7F), 512 kB RAM and clocked at 216MHz with D/I Cache enabled.

b) STM32F469NI: An Arm Cortex-M4 with FPU (M4F) and no-cache, 256 kB RAM and clocked at 180MHz

c) STM32F103: An Arm Cortex-M3 with no FPU and no cache, 20 kB RAM and clocked at 72MHz.



d) ESP32-WROOM-32: Dual-core Xtensa® 32-bit LX6 microprocessors (cache enabled), 520kB RAM clocked at 240MHz.

3.1. Benchmarks from different boards

The following charts show the performance test results between Arm's CMSIS-DSP v1.8 library (green) and ASN's C SDK (blue) using the 8th order IIR Biquad filter described in section 2 for different levels of compiler optimisation with an GNU Arm Embedded GCC compiler (v9.3.1). The graphs show the number of CPU clock cycles required to complete the filtering operation for 1024 samples using single-precision arithmetic - where, a lower value indicates higher performance from the library.

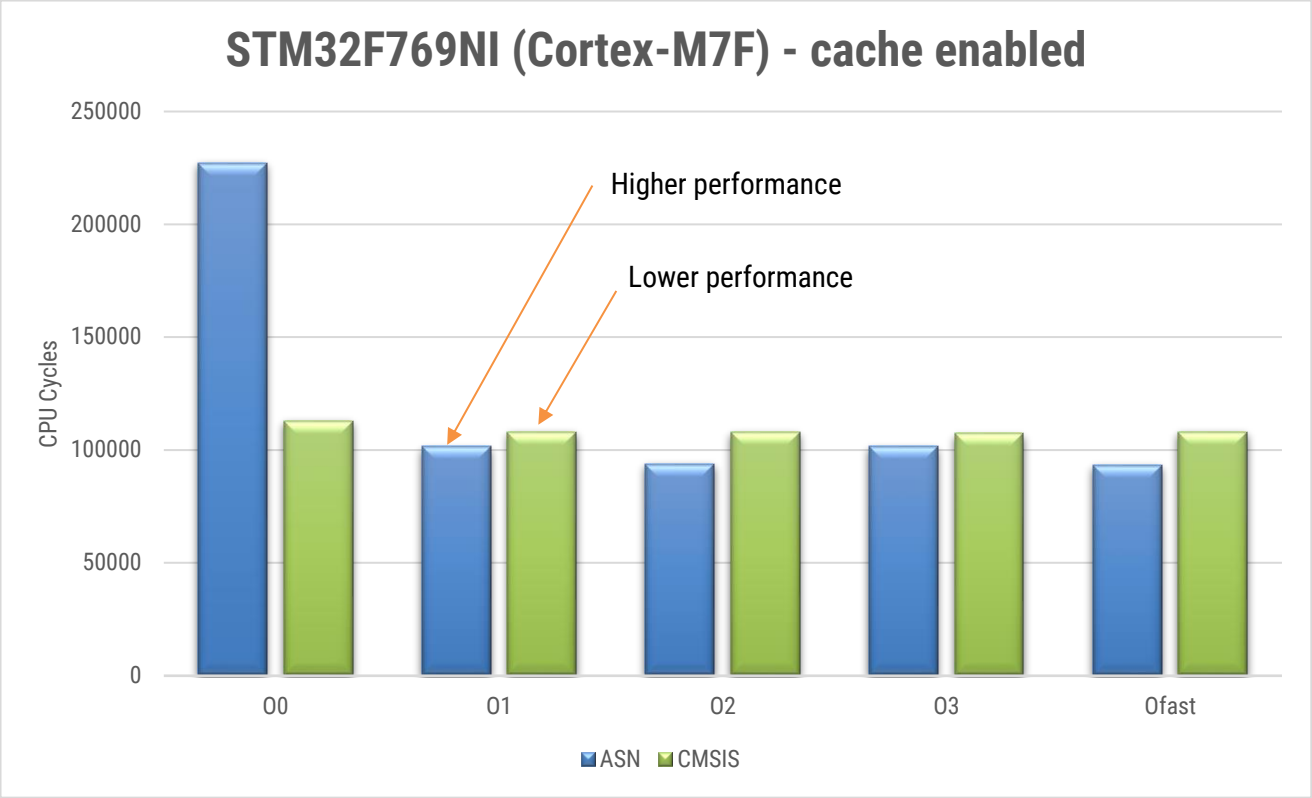


Figure 2: STM32F769NI with D/I-Cache enabled

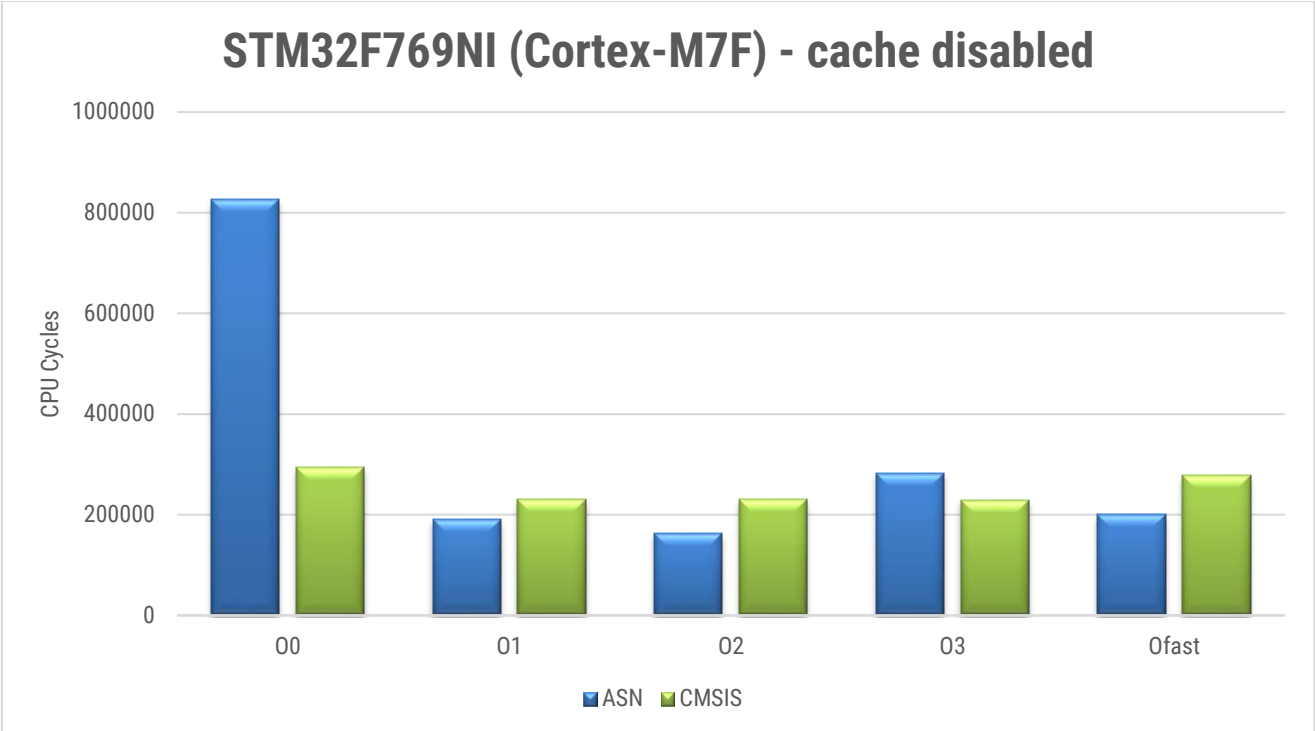


Figure 3: STM32F769NI without D/I-Cache enabled

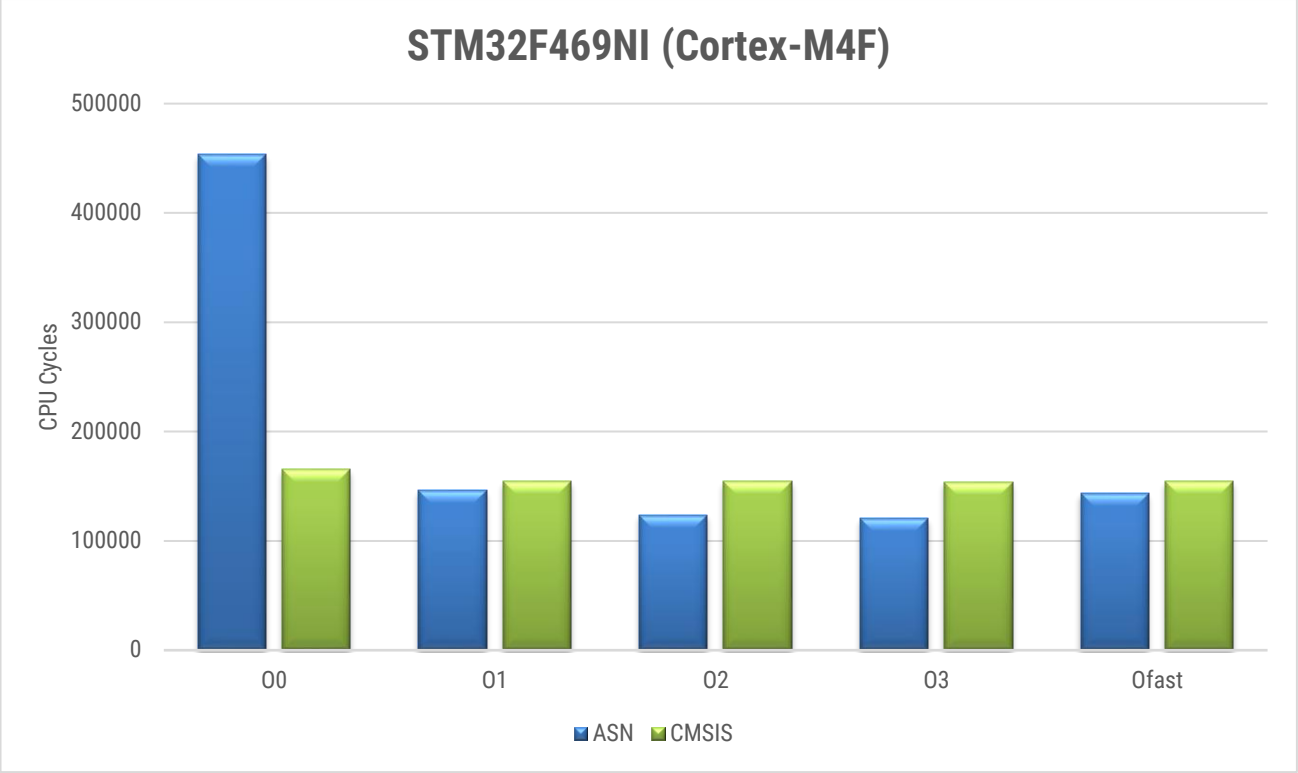


Figure 4: STM32F469NI

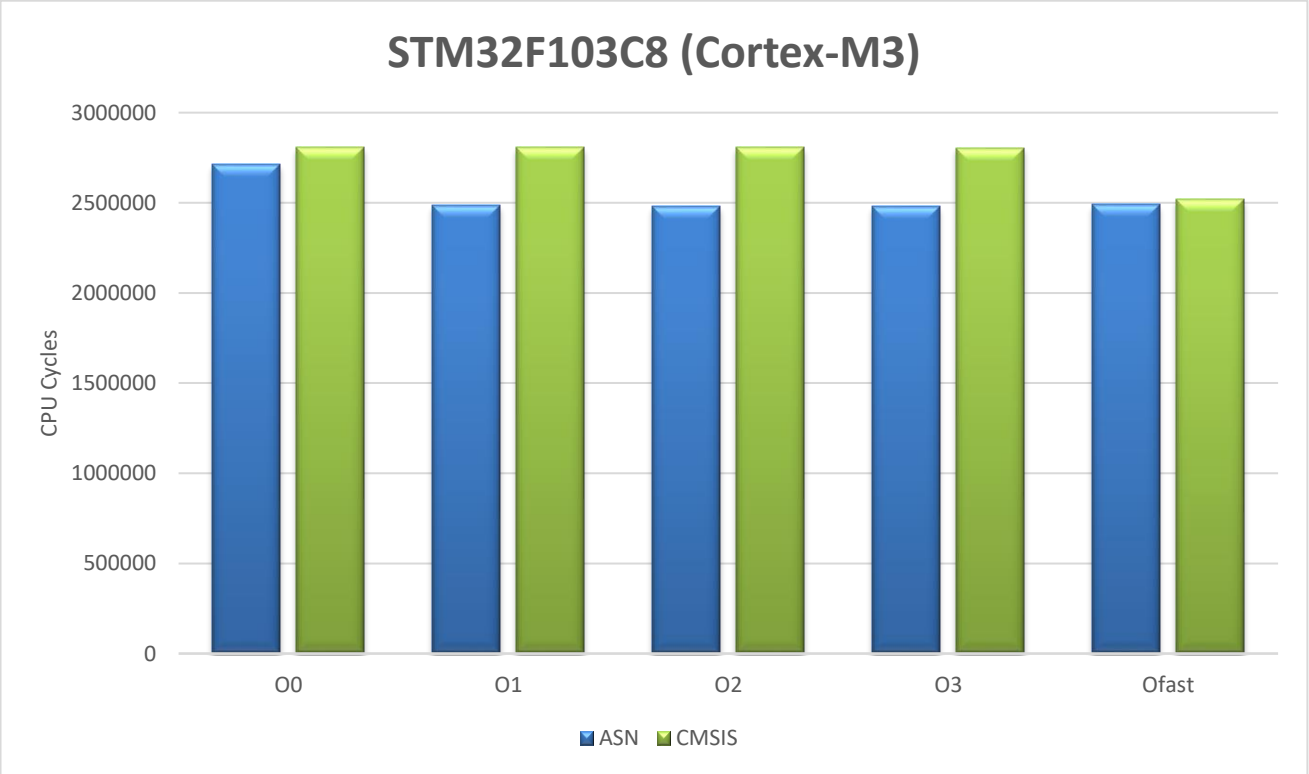


Figure 5: STM32F103

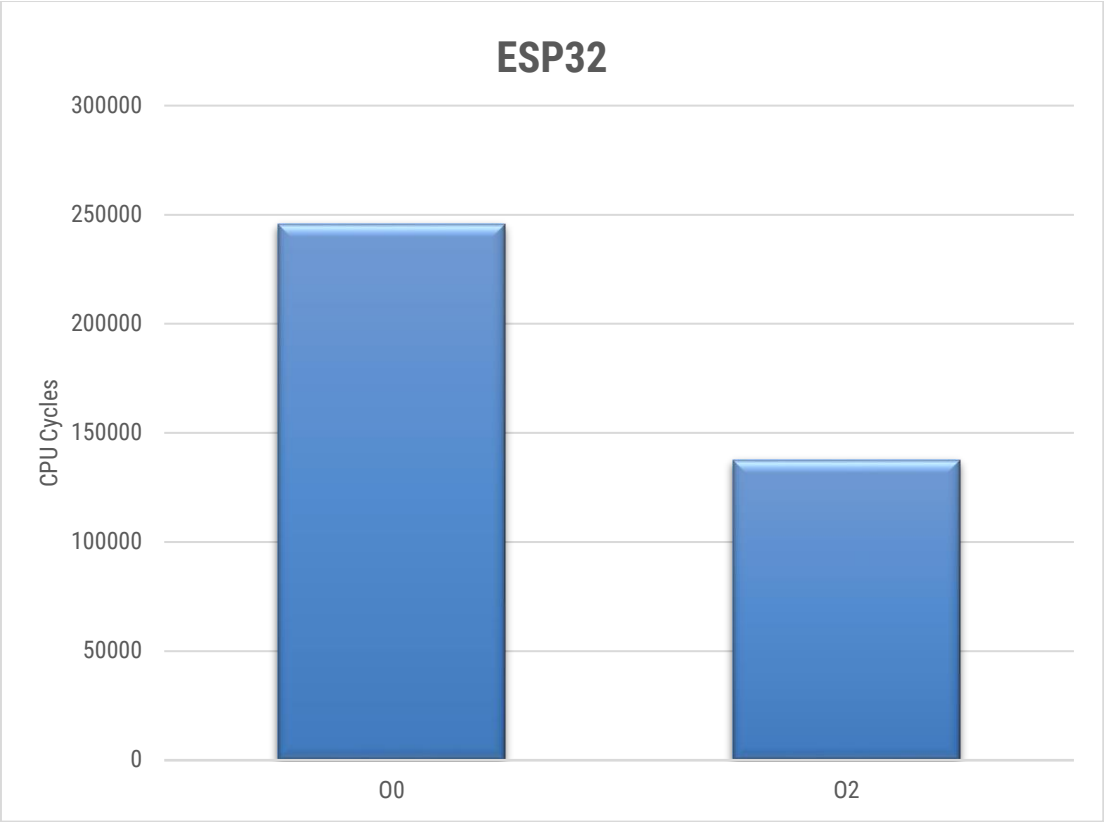


Figure 6: ESP32-WROOM-32

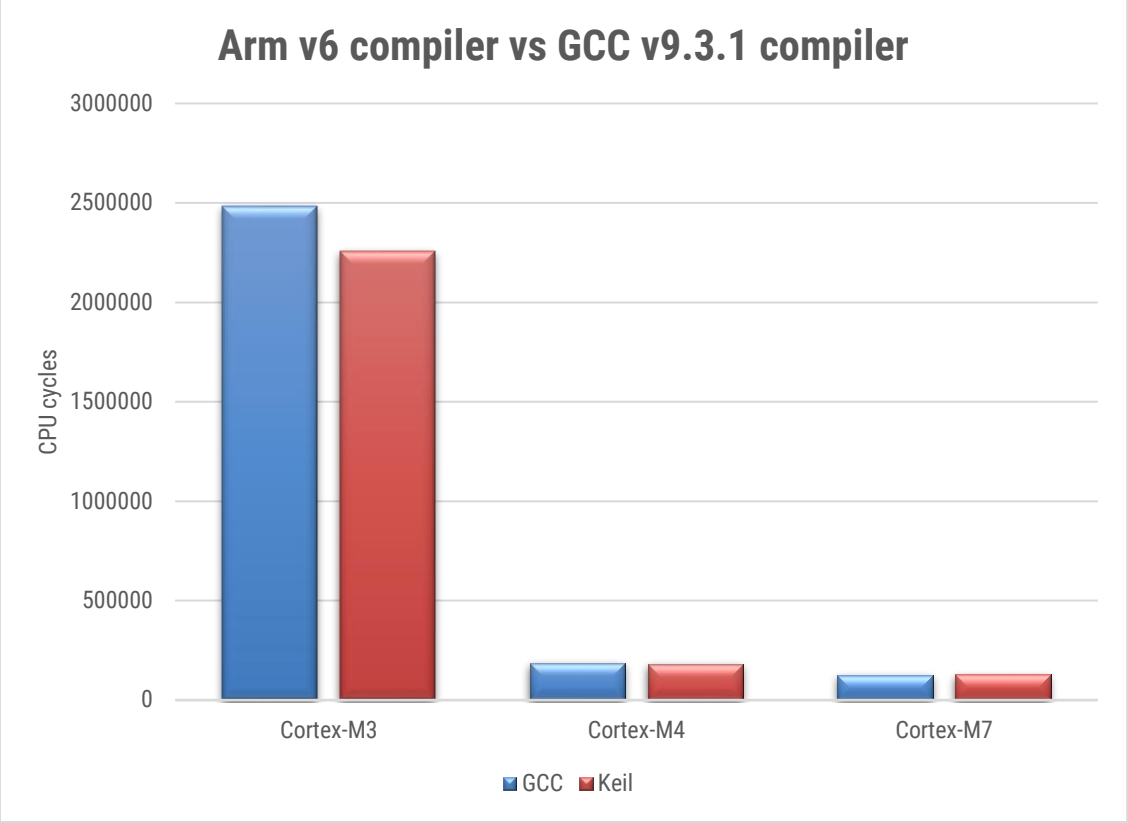


Figure 7 - Arm v6 compiler vs GNU Arm Embedded GCC compiler (v9.3.1)

3.2. Key observations

a) Performance of the ASN-SDK is better at compiler optimisation level ≥ o1, since the o1 compiler optimisation level reduces the debugging information, resulting in less stack consumption and it also enables tail calls, meaning that the CPU does not have to load the stack consecutively.

b) Since Cortex-M0/M3 based microcontrollers do not have a hardware floating-point unit, any arithmetic operations using *float* or *double* will be slow as they will be undertaken in software. The result of this can be seen for both libraries in the STM32F103 benchmark (Figure 5), which is much worse than the Cortex-M4F and M7F cores. Hence, it is always recommended to consider a microcontroller based on the Cortex-M4F or higher in applications where speed and ultimately sampling rate play a crucial role.

   Many developers have traditionally considered devices without an FPU (e.g., Cortex-M0/M3) as the best choice for low power battery applications. However, when comparing a modern Cortex-M7 device manufactured using 40nm semiconductor process technology, to that of a ten-year-old Cortex-M3 using 180nm process technology, the Cortex-M7 device will likely have a lower power profile.

c) To optimise the performance of Cortex-M7F based microcontrollers, the architecture supports the I/D cache. The I and D caches are coupled to the Cortex-M7 system bus interface and when the caches are enabled, all read access to cacheable memory regions are looked up in the caches. As seen in Figure 2, this results in 37% higher performance (o1 compiler optimisation) than that of the Cortex-M4F. However, in cases where the I/D cache is disabled, the CPU requires a greater number of instructions to access the memory using the AHB bus. As such, this results in the worst performance using the Cortex-M7F, as seen in Figure 3.

   As embedded flash memories contains both instructions and read-only data, and the access time of embedded flash is normally slower than the processor's speed. As such, it is **recommended to enable both I and D caches on the Cortex-M7 microcontrollers** when running programs that are stored in embedded flash memories.

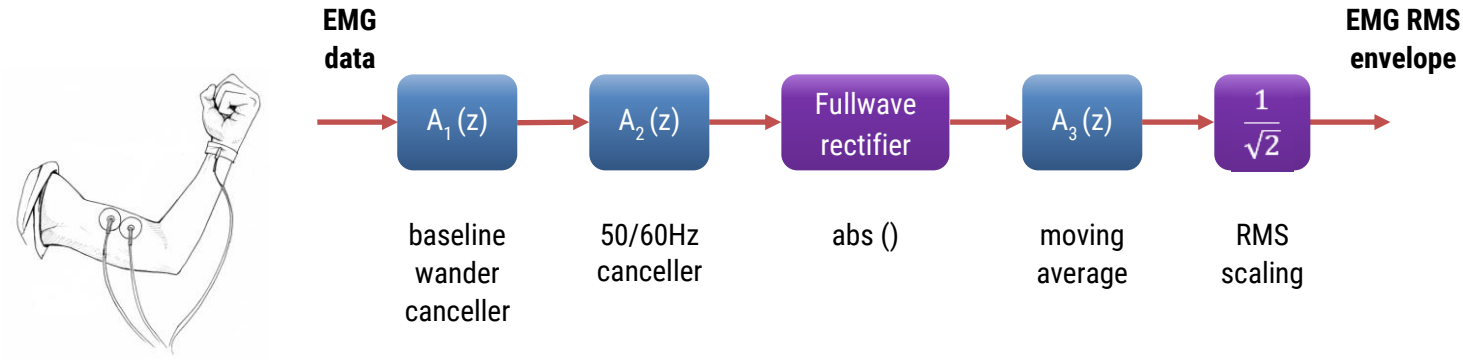
d) Testing the ASN SDK library on the ESP-32, which has two Xtensa® 32-bit LX6 cores, we found the performance to be 39% worse to that of the Cortex-M4 using o2 optimisation – see Figure 6. However, the library only used a single core, but is nevertheless a good contender to Cortex-M4F based microprocessors for filtering applications.

e) Comparing the ASN SDK library performance on Arm's v6 compiler vs GNU Arm Embedded GCC compiler (v9.3.1), it can be seen that the Arm compiler outperforms the GCC compiler on all cores considered herein. This is as expected, as the compiler makes use of Arm's inside knowledge and expertise.

4. Benchmarking an EMG biomedical application

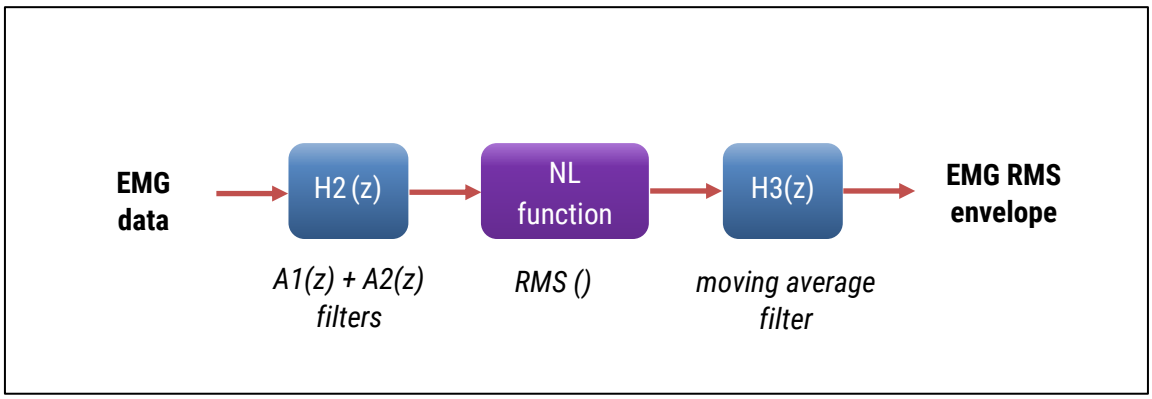
As discussed in section 1, the ASN-SDK also supports some extra filtering functions, such as: a median filter, a moving average filter, all-pass, single section IIR filters, a TKEO biomedical filter and various non-linear functions, including RMS, Abs, Log and Sqrt. These functions form the filter cascade with the tool, and can be used to build signal processing applications, such as EMG and ECG biomedical signal processing applications.

As an example, we will benchmark an Electromyography (EMG) filtering chain developed using the using the ASN filter designer. A full description of the reference design can be found here.



Building block	Equation	Description
A1(z)	$\frac{1 - z^{-1}}{(w_c + 1) + (w_c - 1)\, z^{-1}}$	A baseline removal filter (DC removal) can be realised with a simple first order pole-zero filter. Where, Where, $w_c = tan(\frac{\pi f_c}{fs})$ is used to set the highpass filter's bandwidth.
A2(z)	$\frac{1 - 2\cos w_c z^{-1} + z^{-2}}{1 - 2r\cos w_c z^{-1} + r^2 z^{-2}}$	A notch filter. Where, $w_c = \frac{2\pi f_c}{fs}$ controls the centre frequency, f_c (50 or 60Hz) of the notch, and r controls the bandwidth of the notch.
Fullwave rectification	abs()	`Abs()` function.
A3(z)	$1 + z^{-1} + z^{-2} + \ldots z^{-M}$	An Mth order moving average filter.

These blocks can be translated into ASN Filter Designer filtering and signal processing blocks, as follows:



As seen, the A1(z) and A2(z) filters are combined into a single H2(z) filter, the RMS() function implements the Abs() and amplitude scaling, and finally the H3(z) or post filter implements the moving average filter.

4.1. Benchmarking the complete signal chain

The benchmarks for the complete EMG biomedical application on different processors are shown below for 1024 samples using single-precision arithmetic with level o2 compiler optimisation. As expected, the Cortex-M7F with cache enabled outperforms the other devices by factor of 2.2, and the ESP-32 performs slightly worse than the Cortex-M4F.

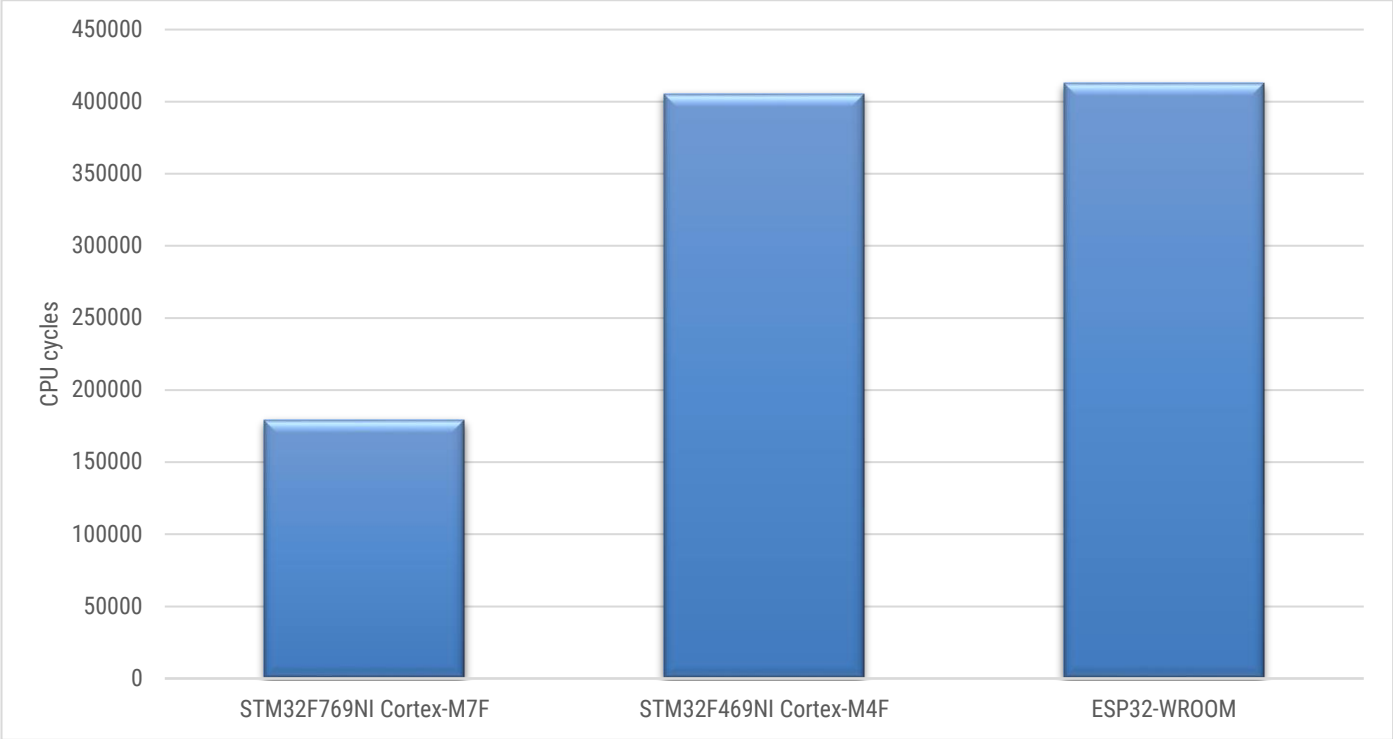


Figure 8: benchmarks for EMG biomedical application on different processors

	Optimisation level	CPU cycles	Clock	Time taken in µs
STM32F769NI (M7F)	o2	179,131	216 MHz	829
STM32F469NI (M4F)	o2	405,282	180 MHz	2251
ESP32-WROOM	o2	412,580	240 MHz	1719

The results between the Cortex-M4F and ESP32 deserve a little more scrutiny, as they are very comparable. Recall from section 3.2, that running the Biquad using the CMSIS-DSP library on the Cortex-M4F was 39% better than the ESP-32. However, for this biomedical application, the C code used on the Cortex-M4F did not make use of the CMSIS-DSP library, as it does not support the required functions needed in the signal chain. Thus, it would seem that the Cortex-M4F and ESP-32 processors running ASN-DSP library C code perform equally well.

Analysing the table, all devices take between 1-2ms to complete the filtering operation. Considering that most biomedical applications are sampled at 200-500Hz, 1024 samples result in a buffer time of at least two seconds. This means that implementing the filtering cascade via the ASN-SDK is efficient enough to accommodate multiple measurement channels, while at the same time leaving sufficient time for undertaking housekeeping tasks.

5. Conclusion

ASN Filter Designer's new ANSI C SDK framework provides developers with a comprehensive ANSI C code base that can be used to deploy filtering applications to any C/C++ embedded platform. This agnostic feature, allows product developers to quickly integrate digital filtering functionality into their existing designs with the minimum amount of effort.

Benchmarking an 8th order IIR Biquad lowpass filter on three types of STM32 Arm Cortex processors (M3, M4F, M7F), a comparison was made between Arm's CMSIS-DSP optimised library and the ASN C SDK for three levels of GCC compiler optimisation. The base test was to run 1024 words through the filter and measure the number of CPU cycles it takes to complete the filtering operation. In all cases, single-precision floating-point arithmetic was used and the CMSIS-DSP library's `BLOCKSIZE=32`. However, increasing the `BLOCKSIZE=128`, resulted in much better performance, as the number of function calls was reduced.

It was found that with a compiler optimisation ≥ o1, the **ASN-SDK matched or outperformed the Arm CMSIS-DSP library for all devices considered herein**. The M3 results in general are much worse than the M4F and M7F, as the M3 does not have hardware floating point support. However, a 37% performance increase was achieved over the Cortex-M4F on a Cortex-M7F device with D/I cache enabled.

Performing a benchmark on the popular ESP-32 SoC, that implements the Xtensa® 32-bit LX6 processor, we found the performance to be 39% worse to that of the Cortex-M4F using o2 optimisation and the Arm CMSIS-DSP library.

As expected, the ASN SDK library performance with an Arm's v6 compiler was higher than that of the GNU Arm Embedded GCC compiler (v9.3.1).

The final test involved benchmarking a complete Electromyography (EMG) biomedical application developed in the ASN Filter Designer tool. The biomedical application was comprised of various types of filters and a non-linear function, and forms a very realistic example of a biomedical signal processing application. Testing on different processors, demonstrated that the Cortex-M7F with cache enabled outperforms the other devices by factor of 2.2, and the ESP-32 performs slightly worse than the Cortex-M4F.

Thus, it can be concluded that without the optimised CMSIS-DSP library, the Cortex-M4F is very comparable to the ESP-32 for implementing a complete signal processing cascade, comprised of different building blocks. For 1024 samples, the complete signal processing cascade took between **1-2ms to complete on all devices**. This is fast enough for all biomedical applications that typically sample data between 200-500Hz. As such, this floating-point design is also suitable for implementation on lower end Arm processors, such as the M3 and M0/M0+ devices.

The ASN-DSP ANSI C SDK and ASN Filter Designer design tool are an excellent choice for developers looking to design and validate biomedical and AIoT filtering applications and deploy them to embedded platforms within a few hours. The ANSI C code base supports both **single and double precision floating point arithmetic**, providing excellent numerical accuracy and wide dynamic range.

The added bonus of **complex filters support**, allows developers to quickly design and implement solutions for I4.0 applications, such as Coriolis flow metering and powerline harmonic frequency tracking applications.