

Matus Plachy, Freescale Semiconductor
Anders Lundgren, IAR Systems
Lotta Frimanson, IAR Systems

Designing advanced DSP applications on the Kinetis ARM Cortex-M4 MCU—Part 1

1 Introduction

Signal processing needs in low cost MCUs are increasing. The microcontroller domain tackles many control applications in the analog and regulator domain like AC motor control and PID regulator designs. The lack of DSP processing power in current MCU devices has severely limited the possibility to perform high speed complex DSP algorithms in low cost designs. The latest ARM Cortex-M4 is targeted as a solution to these needs as it comes with an extensive set of DSP instructions.

This article introduces the basics behind DSP technology and discusses how advanced algorithms like digital filters, FFT's and control loops can be efficiently implemented without having to go into low level assembly programming. In part 2, the design of a motor control application using a sensorless vector control algorithm is discussed.

2 Cortex-M4 MCU introduction

Cortex-M4 is the latest embedded core by ARM. The ARM Cortex-M4 core retains all the advantages of the ARM Cortex-M3 core and adds new digital signal processing capability in the form of DSP extensions, a single cycle MAC unit and an optional single precision floating point unit. Cortex-M4 is using the ARMv7E-M instruction set. Figure 2-1 lists the main architectural features.

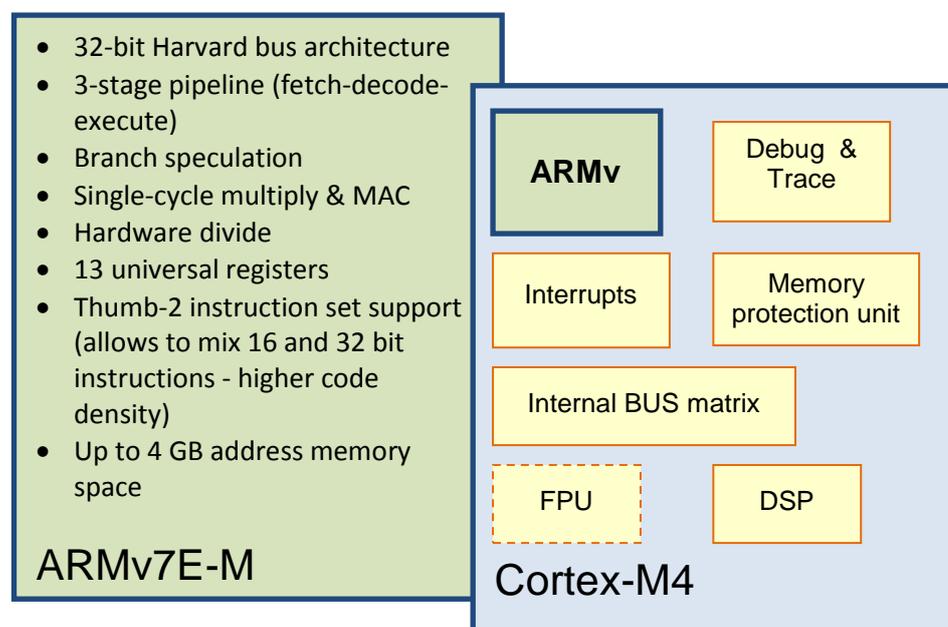


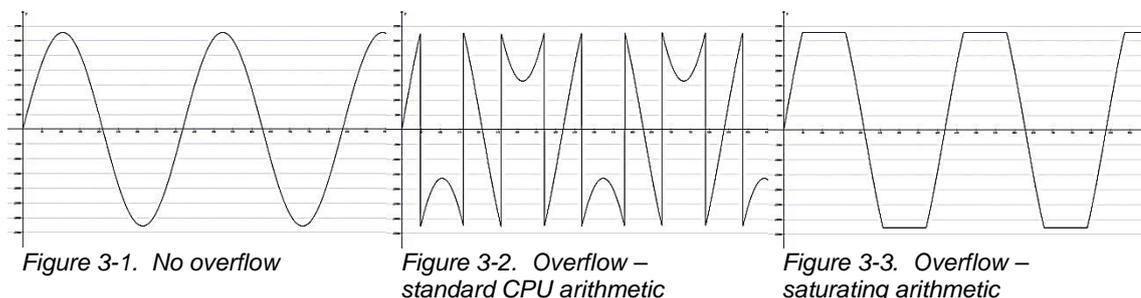
Figure 2-1. Cortex-M4 architecture

3 DSP lecture lite

3.1 Saturating arithmetic's

Saturating arithmetic is the most important requirement for digital signal processing. The term “saturating” comes from analog electronics, in which an amplifier output will be limited (clipped), between fixed values when a large input is applied. Audio, graphics, video and control loops all behave reasonable well if overflow conditions limit the signal to min and max instead of wrapping around to some unexpected value. A graphics pixel is a good example; adding a large value to an almost white pixel using saturating arithmetic will give a maximum white pixel. Doing the same operation with standard CPU arithmetic could give any color as the result of the wrap around. The saturating behavior is clearly the preferred choice here.

To illustrate the difference between saturating arithmetic and standard CPU arithmetic we take a sine wave as shown in Figure 3-1. The sine wave is scaled to fit within the 16-bit signed range of -32768 to 32767. Processing the sine wave by scaling it by a factor 1.5 will cause overflow, the resulting waveform for standard CPU arithmetic is shown in figure 3-2, and for saturating arithmetic in Figure 3-3. In both overflow cases, the scaled waveforms show wrong behavior, but whereas Figure 3-2 is merely clipped/distorted, the waveform in Figure 3-2 is completely incorrect. Saturating arithmetic will preserve the characteristics of a waveform when overdriven; the resulting system will be robust to large signals. The effect is similar to turning the volume knob to high on a stereo amplifier, the sound quality will decrease but the music will still be discernible.



3.2 Data types

DSP operations can use either floating-point or fixed-point format. The Cortex-M4 core has optional hardware support for 32-bit floating-point, but is not optimized for DSP algorithms and too slow for most DSP applications. Available fixed-point data sizes in Cortex-M4 are 8, 16, 32 and 64 bits. Saturating arithmetic is supported on 8, 16 and 32 bits.

The fixed-point format can be integer, fractional or a mix of integer and fractional. The notation used to describe the format of a fixed point number is $Q_{m.f}$, where m is the number of integer bits and f is the number of fractional bits. $Q_{3.12}$ for example denotes a fixed point data type with 3 integer bits and 12 fractional bits, it will fit in a 16-bit register with the remaining bit as a sign bit. The most common format used for DSP operations is $Q_{0.15}$ and $Q_{0.31}$, with only fractional bits to represent numbers between -1.0 and +1.0. It is common to omit the leading '0.' and simply write Q_{15} and Q_{31} . The representation of a Q_{15} number is:

$$value = b_{14} * 2^{-1} + b_{13} * 2^{-2} + \dots + b_1 * 2^{-14} + b_0 * 2^{-15}$$

For example the number 0.5 will be encoded in Q_{15} as 0x4000.

Integer arithmetic (saturating and non-saturating) instructions can be used on $Q_{m.f}$ numbers for addition and subtraction. For multiplication and division, scaling is needed, for example multiplying $Q_{m.f}$ with $Q_{n.g}$ will result in a $Q_{(m+n).(f+g)}$ number.

3.3 DSP instructions

The Cortex-M4 has a number of instructions to support DSP algorithms in an efficient way. Let's take look at the most important ones.

3.3.1 Saturating instructions

A set of instructions are available for saturating addition and subtraction of 8, 16 and 32-bit values; QADD8, QSUB8, QADD16, QSUB16, QADD and QSUB.

The SSAT instruction is available to scale and saturate a value to any bit width that fits a 32-bit value.

3.3.2 MAC instructions

Most DSP algorithms rely heavily on accumulating the result of long series of multiplications. This is the case for FIR and IIR filters, but also for complex algorithms like FFT and DCT. Performing the multiplication and accumulation by a single cycle instruction is key to high performance. Some of the Cortex-M4 MAC instructions are:

- SMLA – is used for Q15 calculations; it performs the operation $32 \text{ += } 16 \times 16$ (multiplies two 16-bit values and adds the result to a 32-bit register)
- SMLAL – is used for Q31 calculations; it performs the operation $64 \text{ += } 32 \times 32$. There is also a variant of the SMLAL instruction to perform the operation $64 \text{ += } 16 \times 16$

3.3.3 SIMD instructions

The availability of suitable instructions for implementing DSP algorithms is a real advantage of the Cortex-M4. To increase performance even more it would be nice to perform several operations in parallel. As discussed above, the MAC instructions performs a multiply and an addition in a single cycle. On top of that there is SIMD (Single Instruction Multiple Data) to perform multiple identical operations in a single cycle instruction. Some of the available SIMD instructions are:

- QADD8 – performs four 8-bit saturating additions
- QADD16 – performs two 16-bit saturating additions
- SMLAD – performs a dual MAC operation $32 \text{ += } 16 \times 16 + 16 \times 16$
- SMLALD – performs a dual MAC operation $64 \text{ += } 16 \times 16 + 16 \times 16$
- SSAT16 – performs two scale and saturate operations

3.4 Algorithms

3.4.1 Filters

The two most common digital filters are FIR (Finite Impulse Response) and IIR (Infinite Impulse Response).

- FIR filters use no feedback and are therefore inherently stable. In the FIR equation below x is the input samples vector, y is the output vector, N is the filter order, and b is the filter coefficients.

$$y[n] = \sum_{i=0}^N b_i x[n - i]$$

- IIR filters use feedback and can show instability if not designed properly. IIR filter can be used to implement the well know analog filter types Butterworth, Chebyshev, Bessel and others. In the IIR equation below, x is the input samples vector, y is the output vector, P is the feedforward filter order, Q is the feedback filter order, and a/b are the filter coefficients.

$$y[n] = \frac{1}{a_0} \sum_{i=0}^P b_i x[n - i] - \frac{1}{a_0} \sum_{j=0}^Q a_j y[n - j]$$

3.4.2 Transforms

A transform is a function that maps data from its original domain into another domain. Typical examples of transforms are FFT (Fast Fourier Transform) and DCT (Discrete Cosine Transform). Both the FFT and DCT can map data from the time domain to the frequency domain, and vice versa.

In our motor control example in section 5 we will use the Clarke and Park transforms. Combined they will transform the motor rotating 3-phase field current to a rotor magnetizing flux and torque component.

Transforms are computational intensive, especially for large input vectors/matrixes. Considerable performance gains can be obtained using the Cortex-M4 DSP specific instructions.

3.4.3 Control loops

Motor control requires fast and stable control loops as will be more discussed in section 5. An example of a common control loop is the PID controller.

4. DSP application development support

4.1 Development tool requirements

To help you use the features on the Cortex-M4 core, there is a multitude of support from the development tools domain. Complex hardware needs to be easy to program and easy-to-use tools are very important. Executions speed is very essential in DSP applications and that puts requirements on the build chain. The compiler must be very good at optimizing the code for speed.

Software development costs and time-to-market requirements are other important factors in all projects. Development tools therefor must comply and support both software standards and programming standards. To reuse and port code, compiler extensions like intrinsic functions need to follow a standard.

4.1.1 CMSIS

The ARM® Cortex™ Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for all Cortex-M processor based devices. CMSIS has been developed by ARM in conjunction with silicon, tools and middleware partners. The idea behind CMSIS is to provide a consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware, simplifying software re-use, reducing the learning curve for new microcontroller developers and reducing the time to market for new devices.

Every CMSIS-compliant C compiler supports the Cortex-M4 extensions defined by ARM. This means for example that if you use a CMSIS-compliant compiler, the DSP related intrinsic functions are named according to the standard and your code will be portable between different compilers.

4.2 DSP algorithm implementation

4.2.1 Assembler

One way of implementing DSP functions is of course to write the algorithms in assembler. This can result in a very high performance when it comes to fast and efficient execution, but assembly optimization is hard work and extends the development time.

4.2.2 Intrinsic functions

Another alternative is to implement DSP algorithms in C or C++. The Cortex-M4 processors can be fully programmed in C and code written in C is easy to write, maintain, and can easily be ported.

The C language does not have any built-in specific support for expressing saturating arithmetic, such that $INT_MAX + n = INT_MAX$ and $INT_MIN - n = INT_MIN$. The operation can be expressed as:

```
uint16_t saturating_16bit_add(uint16_t a, uint16_t b)
{
    uint16_t result = a + b;
    if (((a ^ b) & 0x8000) == 0)
    {
        if ((result ^ a) & 0x8000)
        {
            result = (a < 0) ? 0x8000 : 0x7fff;
        }
    }
    return result;
}
```

The compiler could be designed to recognize this construct and generate the proper Cortex-M4 saturating instruction for those C statements. Even then, the source code is clumsy, error prone and difficult to read.

We somehow need to be able to access DSP instructions in a direct and transparent way. This can be done with the use of intrinsic functions which is a predefined set of functions available in the compiler. The intrinsic functions provide direct access to low-level processor operations and compile into inline code, either as a single instruction or as a short sequence of instructions. To make code portable, the intrinsic functions provided by the tool vendor must conform to a standard, for example CMSIS.

Using Cortex-M4 intrinsic functions, the function above can use the intrinsic function `__QADD16` and be re-written as:

```
uint16_t saturating_16bit_add(uint16_t a, uint16_t b)
{
    return __QADD16(a, b);
}
```

4.3 DSP library

4.3.1 Overview

The alternative to writing your own DSP implementation is to use a DSP library. Most DSP libraries are supplied by the chip vendor and are targeted to a specific architecture. These libraries are often written in assembler to get the best performance and are by that not portable to a different architecture.

There are also both commercial and public domain DSP libraries available that are written in C or C++, like FFTW, SigLib and others. These libraries are portable and optimized for speed.

ARM has recently launched a standardized DSP Library with highly optimized signal processing functions for Cortex-M4 processor-based systems. The library is CMSIS-compliant and is designed to make DSP programs easy to develop for MCU users. It includes general functions for math, trigonometric, and control functions as well as digital filter algorithms for filter design utilities and DSP toolkits:

- Basic math – vector and matrix mathematics
- Fast math – sin, cos, sqrt etc.
- Interpolation – linear, bilinear
- Complex math
- Statistics – max, min, RMS etc.
- Filtering – IIR, FIR, LMS etc.
- Transforms – FFT(real and complex), DCT (cosine transform) etc.
- PID Controller, Clarke and Park transforms
- Support functions – copy/fill arrays, data type conversions etc.

4.3.2 Performance

The performance varies widely depending on the adaptation/optimization of the DSP algorithm to the actual instruction set. In table 4-1 two fixed-point FFT's are compared for execution time. The general purpose `fix_fft` is compared to the ARM specific FFT. The ARM specific FFT have been specifically tuned for the Cortex-M4 DSP instructions and executes the 1024-point FFT in 5 ms, the same library optimized for Cortex-M3 executes in twice the time; 11.9 ms. The general purpose `fix_fft` executes another factor two slower; 20.1 ms at the highest compiler speed optimization level.

1024-point FFT	Cortex-M3 @ 25MHz IAR compiler (no optimization)	Cortex-M4 @ 25MHz IAR compiler (no optimization)	Cortex-M4 @ 25MHz IAR compiler (speed optimization)
<code>fix_fft</code> ¹⁾	35.7 ms	35.7 ms	20.1 ms
<code>arm_cfft_radix4_q15</code> ²⁾	11.9 ms	5.0 ms	5.0 ms

Table 4-1

- 1) written by Tom Roberts (1989), Malcolm Slaney (1994), Dimitrios P. Bouras (2006)
- 2) part of ARM CMSIS 2.0

4.4 Debugging

Debugging is an important and time consuming task of the development cycle that all software developers will have to perform. Cortex-M4 has a very powerful debug architecture that for example allows live access to the core when the application is running, which makes it possible to read and write memory and set/clear breakpoints on a running application. Full trace is provided via the Embedded Trace Macrocell (ETM) in combination with a 4-bit trace port and a low bandwidth instrumentation trace is provided from the Instrumentation Trace Macrocell (ITM) via the SWO pin.

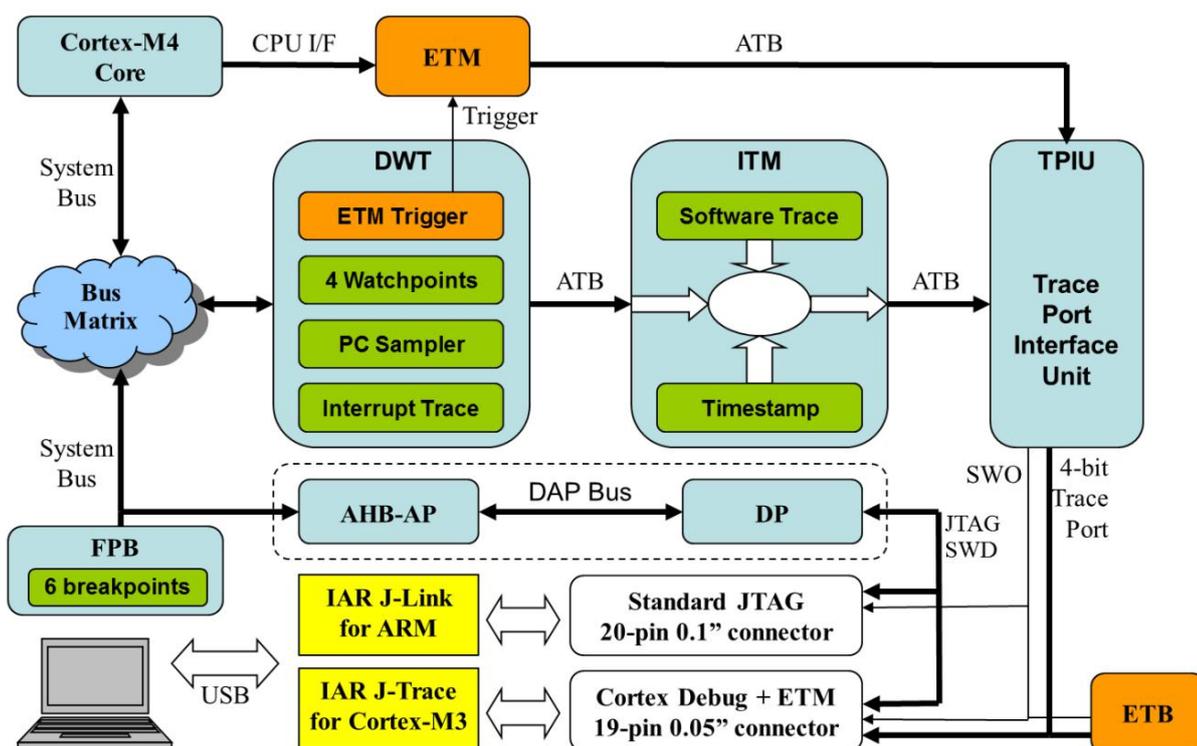


Figure 4-1. Cortex-M4 debug architecture

4.4.1 DSP-specific debugging support

When debugging a DSP application the real time aspect is very important. Single stepping through a motor control application is not possible because it will interfere with the control loops. The trace possibilities on Cortex-M4 in combination with advanced breakpoint conditions and trace filters is a much better solution.

The four watchpoints in the DWT module also provides great possibilities for debugging in real time. It is possible to log accesses to up to four different memory locations or areas, including time information.

A debugger that simulates the complete instruction set including the DSP instructions can be very useful for unit testing of the DSP algorithms before the hardware is available.

4.4.2 Performance analysis

Debugging tools can also be used during design and development for instrumentation, analysis, and performance evaluation of DSP systems. A debugger that utilizes the debugging and trace possibilities on Cortex-M4 can provide a range of useful features.

The PC sampling feature in the DWT module can be used to provide useful statistical information in a debugger, for example, function profiling and statistical data on instruction usage. The profiling information can help you find the functions where most time is spent during execution, for a given stimulus. Those functions are the parts to focus on when spending time and effort on optimizing code. This can be a great tool when optimizing time critical parts of the DSP application.

References

1. *ARM@v7-M Architecture Reference Manual*, ARM Limited, 2010
2. *Application Note 33 Fixed Point Arithmetic on the ARM*, Advanced RISC Machines Ltd (ARM), 1996
3. *Sensorless PMSM Vector Control with a Sliding Mode Observer for Compressors Using MC56F8013*, Freescale Semiconductor, 2008
4. *Sensorless PMSM Control for an H-axis Washing Machine Drive*, Freescale Semiconductor, 2010
5. *Advanced Control Library for 56800E, User reference manual*, Freescale Semiconductor, 2009