Audio on ARM Cortex-M

By Pradeep D, Ittiam Systems Pvt. Ltd.

Portable Music Players have become immensely popular in the recent years. The success of these music players can be attributed to various factors ranging from availability of massive amount of compressed music content, complemented by declining cost for memory storage, ease of use, and easy access to online music content of different genres making them popular. The expectations from the next generation of music players are even more demanding with listeners wishing for longer audio playback time coupled with enriching listening experience.

Cortex-M series, the new generation of low cost microcontrollers from ARM®, are low power by design. Cortex™-M3 core and the recently launched Cortex™-M4 core are based on Harvard architecture with a 3-stage pipeline and implement Thumb®-2 Instruction Set Architecture (ISA) to ensure lower memory requirements. But are these MCUs capable of audio processing? Can they match the expectation of better listening experience?

To analyze the suitability of these processors for audio, first an overview of the implementation of audio processing modules commonly used in audio components like audio decoders and post processing will be provided, with examples from MP3 decoder and Equalizer. These modules will be then categorized based on the requirements they place on a processor's Instruction Set Architecture for an efficient implementation.

The features of the Cortex-M3 and Cortex-M4 cores will be discussed with emphasis on the advantages for audio processing from an instruction set perspective. To illustrate, few code examples of modules from different audio components are discussed followed by an analysis of the cycle and instruction advantage on these processors. Finally, the typical performance numbers for popular audio codecs and audio post processing components will be shown to prove the audio capability of these processor cores.

Functional Blocks of Audio Processing Modules

To enable an analysis on the requirements for audio processing, a discussion on the functional blocks involved in audio processing modules, an audio codec and an audio post processing component, will be taken up now. The modules in the block diagrams featured in this section are coloured to represent 3 different categories.

- **Green** represents Multiply and Accumulate (MAC) Intensive modules
- **Red** represents Mix of MAC and Control code modules
- **Blue** represents modules which are Control code in nature

Audio Codecs

The huge amount of compressed audio data available on the internet necessiate the need for supporting many popular audio decoders in the music player. The processing in the audio codecs involves a set of functional blocks which will be reviewed next.

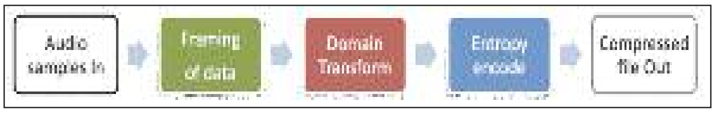


Figure 1: Block diagram of a typical Audio Encoder

Audio Encoder: The goal of an audio codec is to achieve compression of audio data while striving for perceptually transparent quality given the bitrate constraint. A diagram of a typical encoder is shown in Figure 1. Broadly speaking, compression in an audio encoder is achieved in three stages as desribed:

- First, the audio data to be encoded is conditioned by filtering the data. Then the audio data is framed by windowing and overlapping to enable further processing.
- Next, this data is transformed to frequency domain to exploit the redundancy in time domain. Using psychoacoustic principles, the amount of inaudible quantization noise that can be introduced is caculated.
- Finally, this data is quantized and further losslessly compressed through entropy encoding. This encoded data is packed compactly into a bitstream. Though the compression achieved is lossy, it is perceptually transparent.

Audio Decoder: The reverse of the process discussed for audio encoder is applied at the audio decoder to generate the audio data back from the bitstream. As an example, the MP3 decoder is discussed here and the high level module breakup is shown in Figure 2. The block diagram takes compressed audio in the form of MP3 file as input and outputs uncompressed audio in PCM audio.

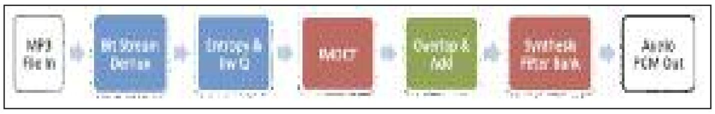


Figure 2: Block diagram of MP3 Decoder

The modules are named by their functionality and are representative of the operations performed

- 'Bit Stream Demux' module parses the MP3 bit stream
- 'Entropy & Inv Q' module performs Huffman decode and inverse quantization
- 'IMDCT' module applies inverse Modified Discrete Cosine Transform on this data

- 'Overlap and Add' module performs windowing, overlap and add operations
- 'Synthesis Filter Bank' reconstructs the time domain samples from filter bank domain data

Audio Post Processing

Post processing of audio data is necessary in two scenarios. First, it is required for providing an enriching listening experience. For instance, a spatializer like stereo widening, can be used to eliminate the listening fatigue while listenting over a headphone. Post processing is also needed even for basic audio playback. For instance, resampling the audio data might be required in a music player whose DAC's supports a limited set of sampling rates.

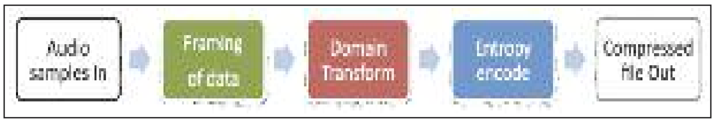


Figure 3: Block diagram of Equalizer

As an example, an equalizer is considered to discuss the typical modules involved in post processing. An equalizer is typically applied to suit the listener's preference, while it can also be used for correcting the response of headphones or speakers. The high level module breakup is shown in Figure 3 for a basic parametric equalizer. The modules listed below are used for modifying only a section of audio spectrum by giving a boost or a cut with controllable parameters for gain value, bandwidth, and center/cuttoff frequency. Multiple such biquad IIR filters can be cascaded in series to achieve a desired response.

- Low Frequency (LF) Shelving Filter – A filter used to modify the bass section of audio spectrum
- Peaking Filter – A filter used to modify the mid range section of audio spectrum
- High Frequency (HF) Shelving Filter – A filter used to modify the treble section of audio spectrum

Categories of Audio Processing Modules

The audio processing modules can be broadly classified into three categories based on their traits. Each category of modules and their traits will be discussed in detail, and the processor requirements for their efficient implementation will be analyzed. The summary of processor requirements for audio processing discussed in this section is tabulated in Table 1.

Audio processing modules		
Category of Modules	**Example**	**Processor Requirements**
MAC intensive modules	Signal Processing routines	High precision MAC
	Filtering modules	SIMD capability
	Convolution, Correlation	Branch speculation
	Windowing modules	Saturation logic
Mix of MAC and control code modules	Transform modules	DSP instructions
	E.g. Filter bank	Mixed bit-width arithmetic
	E.g. FFT, MLT	Packed data processing
Control code modules	Entropy decoding	Bit manipulation operations
	E.g. Huffman, Run length	Unsigned operations
	Bit stream processing	Compact code size

Table 1: Audio processing modules and their processor requirements

MAC Intensive modules

These modules are computationally intensive and are characterized by Multiply and Accumulate (MAC) operations and usually have a predetermined code flow. Signal processing operations like filtering (FIR, IIR filters), windowing, correlation, and convolution are typical examples of this category. The processor requirements for this category of modules are discussed next.

High precision MAC: A high precision multiply and accumulate would be desirable for proper functioning of these modules. For instance the stability of an IIR filter requires high precision operations. The fidelity and accuracy of the output depends directly on the precision of the MAC instructions. Though high precision MAC instructions, for e.g. 32-bit multiplied by 32-bit accumulating in 64-bit result, can be simulated using lower precision instructions, it is desirable to have high precision MAC instructions for an efficient implementation.

SIMD capability: The operations performed in these modules are repeated execution of few instructions yielding a compact inner loop kernel. Single Instruction Multiple Data (SIMD) capability is ideally suited for efficient execution of such modules. A processor with this feature is a added benefit for processing such modules.

Saturation Arithmetic: Certain operations require the use of saturated arithmetic. For instance, when generating window coefficients at runtime or during creation of PCM output data to a desired bitwidth. The need for saturation arithmetic can be reduced, but cannot be avoided altogether due to the higher bitwidth of intermediate result. So instructions supporting saturated arithmetic are desirable for efficient implementation.

Branch speculation: These modules have a predetermined code flow. Since the operations performed in these modules is repeated execution of instructions, a simple branch speculation is helpful to reduce the overhead due to pipeline flush.

Mix of MAC and Control code modules

These modules are also computationally intensive but they involve control code operations amidst MAC intensive operations. Transforms operations like Fast Fourier Transform (FFT), Filterbank are typical examples of this category. The processor requirements for this category of modules are:

DSP instructions: Since both the above category of modules, viz. MAC intensive and mix of MAC and control code modules, primarily involve MAC operations, it will be efficient to have MAC and MLS (Multiply and Subtract) instructions preferably without overhead for performing accumulation.

Mixed bitwidth operations: The minimum bitwidth of the operands are determined by various factors. For instance, in filtering operations the output precision and filter stability determines the bit width of the filter coefficients. So instructions capable of handling mixed bitwidth operands are desirable for an efficient implementation.

Packed Data processing: These modules like FFT typically operate using 16 bit coefficients called twiddles. Higher the radix of the FFT, more twiddles are involved but lesser is the relative complexity. These twiddles can be used by packing into 32 bit data to reduce any register crunch in the core loop. Or if possible overhead of repeated loads can be avoided altogether by loading the twiddles onto registers outside the core loop. So instructions capable of operating on packed data are desirable for efficiency.

Bit Reversal: Instructions capable of performing bit reversal are beneficial for key modules, for instance FFT, where bit reversed addressing is needed.

Control code modules

These modules mainly involve control code and the code flow is data dependent. Bistream demux and entropy decoding operations are examples of this category. For instance, MP3 decoder has Huffman decoding and WMA Decoder has Run length decoding. The processor requirements for this category of modules are discussed next.

Unsigned and Bitwise operations: These modules operate on the packed bitstream, which require data extraction. Instructions for performing bitwise operations, extract, packing, unpacking and instructions capable of handling unsigned operands are desirable for an efficient implementation.

Compact footprint: The code flow is not predetermined in these modules, since it is mainly data dependent. So having a smaller footprint helps reduce any overhead due to cache flush.

ARM Cortex-M3 and Cortex-M4 Processors

In this section, the features of Cortex-M3 core, the low power yet high performance microcontroller from ARM, will be explored and then Cortex-M4, the latest processor core in the Cortex-M series, will be discussed. The features of Cortex-M processors with example instructions and cycles are listed in Table 2.

Cortex-M3 Features: The high precision MAC unit with a precision for multiplying upto 32-bit with 32-bit data giving 64-bit accumulated result is present. Other advantageous features like DSP instructions, instruction to assist norm calculation, branch speculation, high code density due to Thumb®-2 and hardware divide instructions are present. These features are advantageous for audio processing.

Cortex-M4 Features: The Cortex-M4 core, successor to Cortex-M3, inherits all its features, and has additional ones like single cycle MAC unit and floating point unit (FPU). The MAC unit takes just 1 cycle for even highest precision operations, multiplying 32-bit by 32-bit data giving 64-bit accumulated result. With new features like DSP instructions with SIMD capability, high precision MAC unit, Cortex-M4 is a microcontroller with DSP capabilities, a Digital Signal Controller (DSC). These features are advantageous for audio processing, where high precision is desirable to maintain transparent audio quality.

ARM® Cortex™-M Processor features			
Feature	**Example Instructions**	**Cortex-M3 cycles**	**Cortex-M4 cycles**
High precision MAC	SMULL, SMLAL UMAAL	3-7 -	1 1
DSP Instructions	MUL, MLA SMMUL, SMMLSR	1-2 -	1 1
Saturated arithmetic	SSAT, USAT QADD, QSUB	1 -	1 1
Bitwise operations	SXTH, UBFX PKHTB	1 -	1 1
Mixed bit-width capability	SMULWB, SMLAWT	-	1
Packed data processing	SMULBB, SMLABT	-	1
SIMD capability	SMLAD, SMUADx	-	1

Table 2: Cortex-M3 and Cortex-M4 features, example instructions and cycles

Traditionally audio processing is implemented on a Digital Signal Processor (DSP) while the General Purpose Processor (GPP) takes care of the system related features. With the introduction of the Cortex-M series, audio can be efficiently implemented on these cores without the need for a DSP, since these processors are ideally suited for audio processing application.

Code Examples of Audio Modules

Now, a few code examples of routines from different audio processing components – an audio decoder, encoder, post processing, and general signal processing will be shown. The instructions in the code segment in **bold typeface** are newly introduced in Cortex-M4. The number of cycles taken by each instruction is given in braces. Summary of the total cycles taken, number of instructions required, and registers used are tabulated for each example.

Window Overlap Add module

Windowing with Overlap add is a typical module present in most audio decoders. It is computationally intensive and characterized by MAC operations. Figure-4 illustrates the processing of this module. The implementation in this example is given in Equation 1.

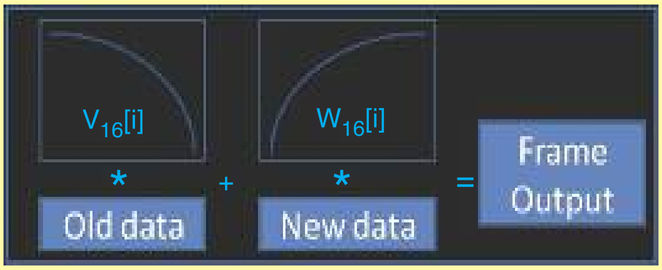


Figure 4: Diagram of Window Overlap Add module

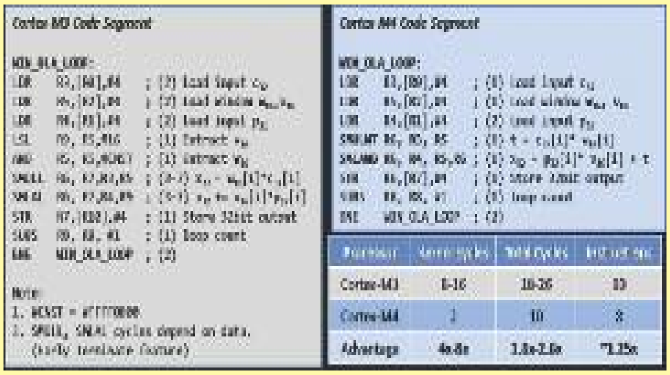
continued

$$x_{32}[i] = c_{32}[i] * w_{16}[i] + p_{32}[i] * v_{16}[i];$$

w_{16}, v_{16} – 16 bit window coefficients; $0 < i <$ frame length

c_{32}, p_{32} – 32 bit input data from current and previous frame

Equation 1: Window Overlap Add module



Cortex M3 Code Segment

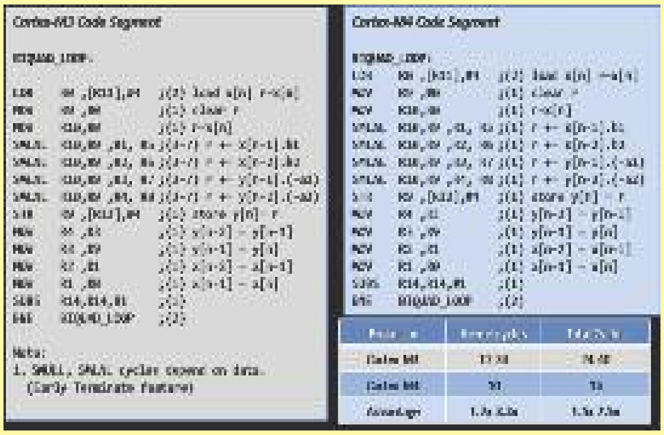
```
WIN_OLA_LOOP:
LDR    R3, [R0],#4     ; (2) Load input c32
LDR    R4, [R2],#4     ; (2) Load window w16, v16
LDR    R6, [R1],#4     ; (2) Load input p32
LSL    R5, R5,#16      ; (1) Extract v16
AND    R5, R5,#CNST    ; (1) Extract w16
SMULL  R6, R7,R3,R5    ; (3-7) x32 = w16[i]*c32[i]
SMLAL  R6, R7,R8,R9    ; (3-7) x32 += v16[i]*p32[i]
STR    R7, [R10],#4    ; (1) Store 32bit output
SUBS   R9, R9, #1      ; (1) Loop count
BNE    WIN_OLA_LOOP    ; (2)

Note:
1. #CNST = #FFFF0000
2. SMULL, SMLAL cycles depend on data.
   (Early terminate feature)
```

Cortex M4 Code Segment

```
WIN_OLA_LOOP:
LDR     R3, [R0],#4    ; (2) Load input c32
LDR     R5, [R2],#4    ; (2) Load window w16, v16
LDR     R4, [R1],#4    ; (2) Load input p32
SMULWT  R6, R3, R5     ; (1) t = c32[i]* w16[i]
SMLAWB  R6, R4, R5,R6  ; (1) x32 = p32[i]* v16[i] + t
STR     R6, [R7],#4    ; (1) Store 32bit output
SUBS    R8, R8, #1     ; (1) loop count
BNE     WIN_OLA_LOOP   ; (2)
```

Processor	Kernel cycles	Total cycles	Instructions
Cortex-M3	8-16	18-26	10
Cortex-M4	2	10	8
Advantage	4x-8x	1.8x-2.6x	~1.25x



Cortex-M3 Code Segment

```
BIQUAD_LOOP:
LDR    R9 ,[R11],#4     ;(2) load x[n] r=x[n]
MOV    R8 ,#0           ;(1) clear r
MOV    R10,R9           ;(1) r=x[n]
SMLAL  R10,R8 ,R1, R4   ;(4-7) r += x[n-1].b1
SMLAL  R10,R8 ,R2, R5   ;(4-7) r += x[n-2].b2
SMLAL  R10,R8 ,R3, R7   ;(4-7) r += y[n-1].(-a1)
SMLAL  R10,R8 ,R4, R8   ;(4-7) r += y[n-2].(-a2)
STR    R8 ,[R12],#4     ;(2) store y[n] = r
MOV    R4 ,R3           ;(1) y[n-2] = y[n-1]
MOV    R3 ,R8           ;(1) y[n-1] = y[n]
MOV    R2 ,R1           ;(1) x[n-2] = x[n-1]
MOV    R1 ,R9           ;(1) x[n-1] = x[n]
SUBS   R14,R14,#1       ;(1)
BNE    BIQUAD_LOOP      ;(2)

Note:
1. SMULL, SMLAL cycles depend on data.
   (Early Terminate feature)
```

Cortex-M4 Code Segment

```
BIQUAD_LOOP:
LDR    R9 ,[R11],#4     ;(2) load x[n] r=x[n]
MOV    R8 ,#0           ;(1) clear r
MOV    R10,R9           ;(1) r=x[n]
SMLAL  R10,R8 ,R1, R4   ;(1) r += x[n-1].b1
SMLAL  R10,R8 ,R2, R5   ;(1) r += x[n-2].b2
SMLAL  R10,R8 ,R3, R7   ;(1) r += y[n-1].(-a1)
SMLAL  R10,R8 ,R4, R8   ;(1) r += y[n-2].(-a2)
STR    R8 ,[R12],#4     ;(1) store y[n] = r
MOV    R4 ,R3           ;(1) y[n-2] = y[n-1]
MOV    R3 ,R8           ;(1) y[n-1] = y[n]
MOV    R2 ,R1           ;(1) x[n-2] = x[n-1]
MOV    R1 ,R9           ;(1) x[n-1] = x[n]
SUBS   R14,R14,#1       ;(1)
BNE    BIQUAD_LOOP      ;(2)
```

Processor	Kernel cycles	Total cycles
Cortex-M3	17-28	24-40
Cortex-M4	10	16
Advantage	1.7x-2.8x	1.5x-2.5x

Biquad IIR Filter module

Biquad is the common name given to second order IIR filter, a typical routine present in the audio encoders and post processing modules like equalizer. It is computationally intensive in nature and characterized by MAC operations. Figure-5 illustrates the Direct Form-1 version of this module. The implementation in this example is given in Equation-2. In this example, the register usage and number of instructions needed are same on both the processors.

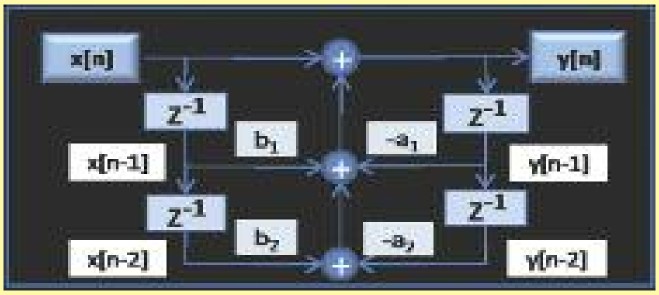


Figure 5: Biquad Direct Form 1 IIR Filter

$$H(z)=\frac{1+b_1\cdot z^{-1}+b_2\cdot z^{-2}}{1+a_1\cdot z^{-1}+b_2\cdot z^{-2}};\qquad H(z)\text{ – Transfer function}$$

$$y[n]=x[n]+b_1x[n-1]+b_2x[n-2]-a_1y[n-1]-a_2y[n-2]$$

y[n] – 32 bit filter output; x[n] – 32 bit input data;

a_i – 32 bit filter coeff; b_i – 32 bit filter coeff

Equation 2: Window Overlap Add module

continued

Radix-2 FFT Butterfly module

FFT is a general signal processing operation. It is typical of audio processing modules which need FFT operation for frequency domain transformation. It is computationally intensive and characterized by mix of MAC and control code operations. This example only illustrates the Radix 2 decimation-in-time (DIT) butterfly. The input and output data are in 16 bit precision. Figure 6 illustrates the processing of this module. The implementation in this example is given by Equation-3.

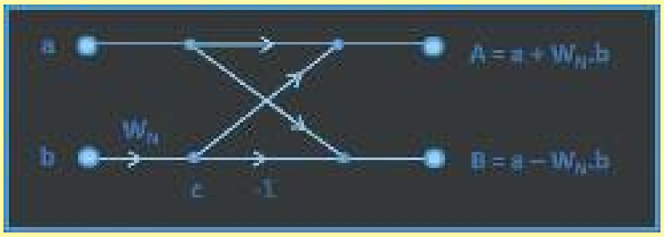
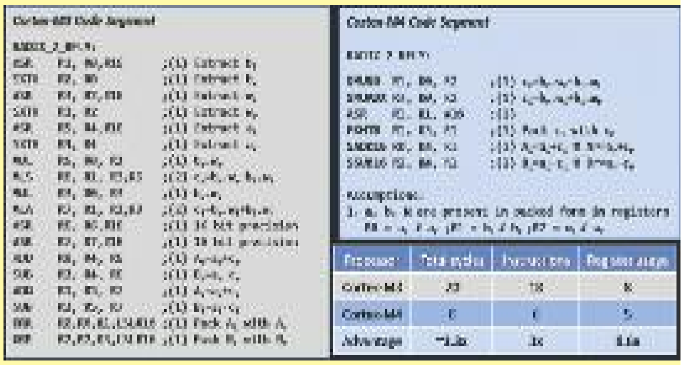


Figure 6: Radix-2 DIT FFT Butterfly

$$a=a_r+ja_i;\quad b=b_r+jb_i;\quad W_N=w_r+jw_i$$
$$c=c_r+jc_i;\quad A=A_r+jA_i;\quad B=B_r+jB_i;$$
$$A=a+W_N\bullet b;\quad B=a-W_N\bullet b;\quad c=W_N\bullet b$$
$$c_r=w_rb_r-w_ib_i;\qquad c_i=w_rb_i+w_ib_r$$
$$A_r=a_r+c_r;\quad A_i=a_i+c_i;\quad B_r=a_r-c_r;\quad B_i=a_i-c_i$$

Equation 3: Radix-2 Decimation-in-Time Butterfly



Cortex-M3 Code Segment

```
RADIX_2_BFLY:
ASR    R3, R0,#16        ;(1) Extract bi
SXTH   R7, R0            ;(1) Extract br
ASR    R4, R2,#16        ;(1) Extract wi
SXTH   R2, R2            ;(1) Extract wr
ASR    R5, R1,#16        ;(1) Extract ai
SXTH   R1, R1            ;(1) Extract ar
MUL    R6, R0, R2        ;(1) br.wr
MLS    R6, R3, R4,R6     ;(2) cr=br.wr-bi.wi
MUL    R9, R0, R4        ;(1) br.wi
MLA    R2, R3, R2,R9     ;(2) ci=bi.wr+br.wi
ASR    R6, R6,#16        ;(1) 16 bit precision
ASR    R2, R2,#16        ;(1) 16 bit precision
ADD    R0, R1, R6        ;(1) Ar=ar+cr
SUB    R3, R1, R6        ;(1) Br=ar-cr
ADD    R1, R5, R2        ;(1) Ai=ai+ci
SUB    R2, R5, R2        ;(1) Bi=ai-ci
PKH    R0,R0,R1,LSL#16   ;(1) Pack Ar with Ai
PKH    R2,R3,R2,LSL#16   ;(1) Pack Br with Bi
```

Cortex-M4 Code Segment

```
RADIX_2_BFLY:
SMUAD   R1, R6, R2       ;(1) ci=bi.wr+br.wi
SMUSDX  R2, R6, R2       ;(1) cr=br.wr-bi.wi
ASR     R1, R1, #16      ;(1)
PKHTB   R1, R1, R2       ;(1) Pack ci with cr
SADD16  R0, R5, R1       ;(1) Ar=ar+cr, Ai=ai+ci
SSUB16  R2, R5, R1       ;(1) Br=ar-cr, Bi=ai-ci

Assumption:
1. a, b, W are present in packed form in registers
   R5 = ai / ar, R6 = bi / br, R2 = wi / wr
```

Processor	Total cycles	Instructions	Register usage
Cortex-M3	20	18	8
Cortex-M4	6	6	5
Advantage	~3.3x	3x	1.6x

FIR Filter module

FIR filtering is a typical signal processing operation and usual of post processing modules. This module is computationally intensive and characterized by MAC operations. Figure-7 illustrates the processing of this module. The implementation in this code example is given in Equation-4.

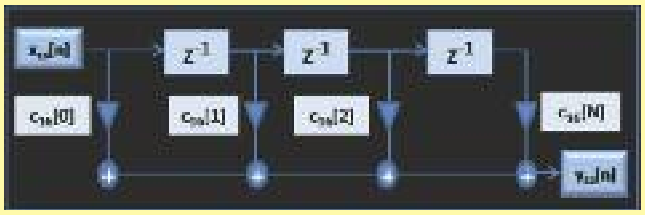


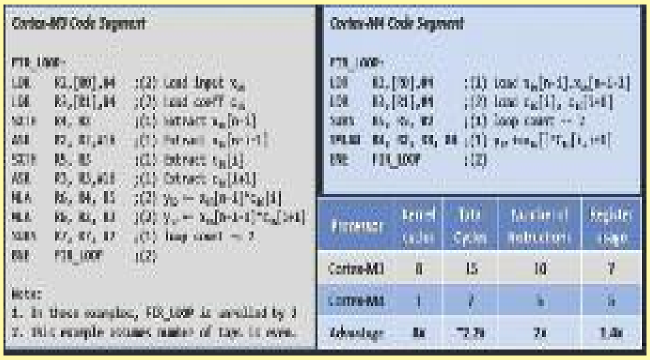
Figure 7: Diagram of FIR Filter module

$$y_{32}[n] = \sum_{i=0}^{N} x_{16}[n-i] \cdot c_{16}[i]$$

y_{32} – 32 bit filter output; N - filter order

x_{16} – 16 bit input data; c_{16} – 16 bit filter coeff

Equation 4: FIR filter equation



Cortex-M3 Code Segment

```
FIR_LOOP:
LDR    R2,[R0],#4    ;(2) Load input x16
LDR    R3,[R1],#4    ;(2) Load coeff c16
SXTH   R4, R2        ;(1) Extract x16[n-i]
ASR    R2, R2,#16    ;(1) Extract x16[n-i-1]
SXTH   R5, R3        ;(1) Extract c16[i]
ASR    R3, R3,#16    ;(1) Extract c16[i+1]
MLA    R6, R4, R5    ;(2) y32 += x16[n-i]*c16[i]
MLA    R6, R2, R3    ;(2) y32 += x16[n-i-1]*c16[i+1]
SUBS   R7, R7, #2    ;(1) loop count -= 2
BNE    FIR_LOOP      ;(2)

Notes:
1. In these examples, FIR_LOOP is unrolled by 2
2. This example assumes number of taps is even.
```

Cortex-M4 Code Segment

```
FIR_LOOP:
LDR    R2,[R0],#4           ;(1) Load x16[n-i],x16[n-i-1]
LDR    R3,[R1],#4           ;(2) Load c16[i], c16[i+1]
SUBS   R5, R5, #2           ;(1) loop count -= 2
SMLAD  R4, R2, R3, R4       ;(1) y32 += x16[]*c16[i, i+1]
BNE    FIR_LOOP             ;(2)
```

Processor	Kernel Cycles	Total Cycles	Number of Instructions	Register Usage
Cortex-M3	8	15	10	7
Cortex-M4	1	7	5	5
Advantage	8x	~2.1x	2x	1.4x

Audio Solution on ARM Cortex-M Processors

Ittiam's audio solution on Cortex-M processors comprises of low-power audio codecs and high quality post processing components. The typical performance for a few chosen set of representative audio components on both Cortex-M3 and Cortex-M4 processors are given in Table 3.

Processor features for audio: The audio codecs have been optimized by making the best use of the DSP instructions that Cortex-M3 offers and SIMD capability that Cortex-M4 processor offers. The performance optimized audio solution enables efficient use of processor MHz, resulting in longer audio playback time. The audio post processing components achieve high quality by efficiently utilizing the high precision MAC unit. These features of Cortex-M processors enable Ittiam's audio solution to achieve high quality at low power.

Audio Capability: These typical performance numbers are for stereo mode and 44.1 kHz sampling rate. The decoders and the encoder are configured for 128kbps. The parametric equalizer is configured for 3 bands. All the performance numbers are assuming zero-wait state memory configuration. The performance on hardware will vary with actual memory configuration.

ITTIAM Audio Solution		Cortex-M3		Cortex-M4	
		Typical MHz	% Load	Typical MHz	% Load
Audio Decoders	MP3 Decode	15 – 18	16%	7 – 10	6%
	AAC Decode				
	WMA Decode				
Audio Encoder	MP3 Encode	21 – 23	22%	12 – 15	9%
Audio Post Processing	Parametric Equalizer	10 – 15	12%	5 – 8	4%
	Stereo Widening				
	Dynamic Range Control				
	Loudness Compensation				

Table 3: Processor resource utilization for audio components on Cortex-M3 and Cortex-M4

For calculating the percentage loading on the processor, the Cortex-M3 is assumed to be clocked at 100MHz and the Cortex-M4 is assumed to be clocked at 150MHz. The typical processor loading for performing an audio decode and applying two effects, for instance MP3 decode followed by an equalizer and stereo widening, is around 40% on Cortex-M3, and around 15% on Cortex-M4. This shows that that these processors are very much audio capable and can provide enhanced listening experience as well.

ITTIAM Audio Components	Cortex-M3		Cortex-M4	
	RAM (kB)	ROM (kB)	RAM (kB)	ROM (kB)
MP3 Decode	12	28	12	24
WMA Decode	26	45	26	41
Parametric Equalizer	2	20	2	16

Table 4: Memory resource requirement for audio components on Cortex-M3 and Cortex-M4

Smaller Footprint: The memory requirements for a few audio components on both Cortex-M3 and Cortex-M4 processors are given in Table-4. These memory resource requirements are provided for both RAM and ROM memory types. The ROM requirements on Cortex-M4 are lower when compared over Cortex-M3 due to its advanced features. The Thumb-2 feature of Cortex-M processors enables compact memory footprint for audio components, resulting in cost savings due to lower memory requirements.

Conclusion

From the analysis of the various audio processing modules, and the processor requirements for audio processing, we can conclude that the features and capabilities of ARM Cortex-M3 and Cortex-M4 processor cores make them efficient for processing audio. The Cortex-M processors are well suited for audio applications given their low-power and superior performance. These processors are very much audio capable and they can be used for low-power design enabling longer audio play out time while simultaneously providing an enriching listening experience.

END