NEW Math Accelerator

- Cordic trigonometric co-processor
- Filter Math Accelerator (FMAC)

life.augmented

Cordic trigonometric co-processor

Cordic (Trigo) - Main features

- Cordic can be used for the following functions:
  - Cosine (cos θ)
  - Sine (sin θ)
  - Phase (atan2 y,x)
  - Modulus ($\sqrt{x^2 + y^2}$)
  - Arctangent ($\tan^{-1} x$)
  - Hyperbolic sin (sinh x)
  - Hyperbolic cosine (cosh x)
  - Hyperbolic arctangent ($\tanh^{-1} x$)
  - Natural logarithm (ln x)
  - Square root ($\sqrt{x}$)

- Cordic provides the best compromise between
  - Hardware complexity
  - Range of functions supported → Supports large set of math.h functions
  - **Speed-up relative to software (e.g. Sin and Cos are 5x faster than SW execution)**

→ In Motor Control (real application) = ~12% gain in control loop execution !

Function parameters

- Arguments
  - Cordic functions can take one or two arguments (ARG1, ARG2)
  - The domain (input range) is limited by the function, or else by the range of convergence of the Cordic algorithm.
- Results
  - Cordic functions deliver one or two results (RES1, RES2)

Cordic - Precision

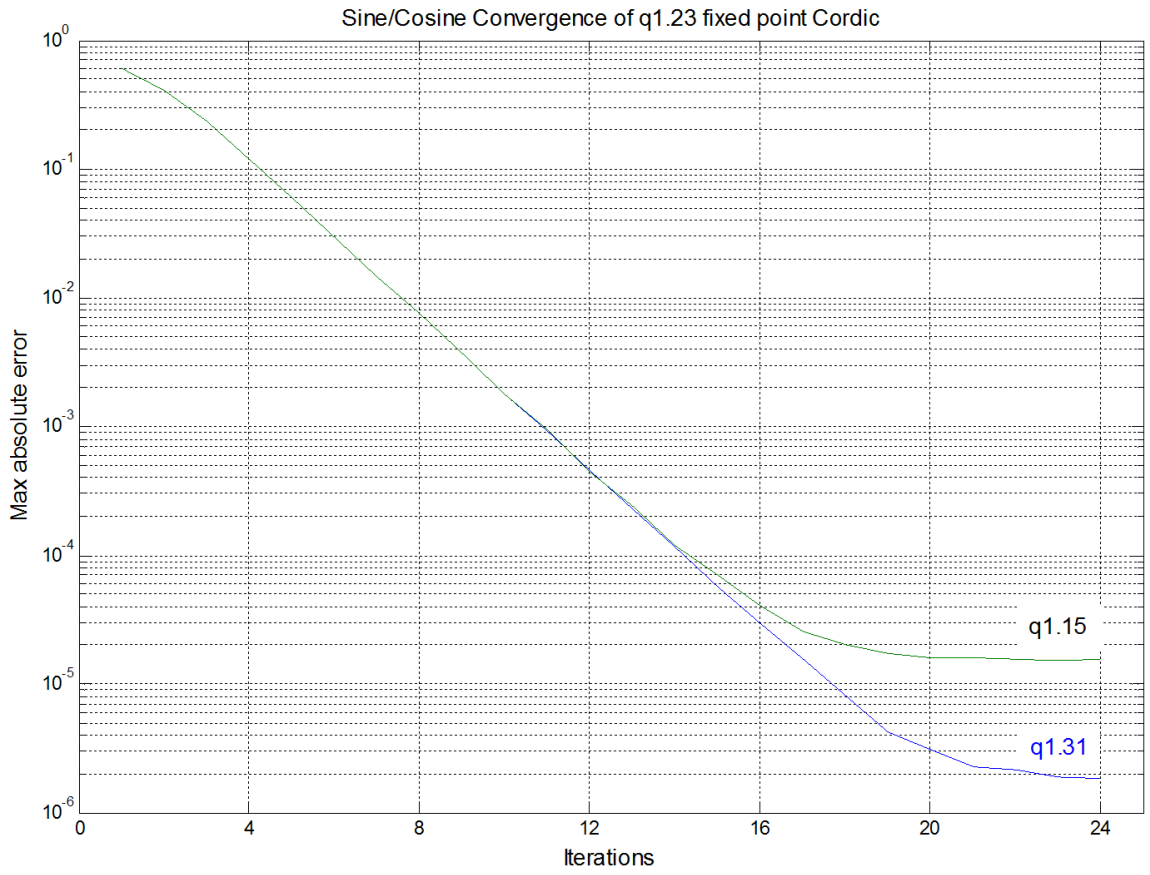
Function	Number of clock cycles for (at least) 16-bit precision	Maximum precision
Sin,Cos,Phase,Mod,Atan	4	20-bit (6 cycles)
Sinh,Cosh,Atanh,Ln	5	19-bit (6 cycles)
Sqrt	3	20-bit (3 cycles)

Numerical representation

- Fixed point representation
  - The CORDIC operates in fixed point signed integer format. Input and output values can be either q1.31 or q1.15.
  - In q1.31 format, numbers are represented by one sign bit and 31 fractional bits (binary places). The numeric range is therefore -1 (0x80000000) to 1 - 2^{-31} (0x7FFFFFFF). The precision is 2^{-31} (around 5 x 10^{-10}).
  - In q1.15 format, the numeric range is 1 (0x8000) to 1 - 2^{-15} (0x7FFF). This format has the advantage that two input arguments can be packed into a single 32-bit write, and two results can be fetched in one 32-bit read. However the precision is reduced to 2^{-15} (around 3 x 10^{-5}).
- Angle representation
  - Angles are divided by π in order to represent them efficiently in fixed point format. Hence -1 corresponds to the angle $-\pi$ and +1 corresponds to $+\pi$. Increasing the angle past $+\pi$ automatically causes it to wrap to $-\pi$.
- Scaling factor
  - Some of the functions have a domain which exceeds the fixed point numerical range. In this case a power of two scaling factor (right shift) can be applied. The scaling factor must be accounted for in the Cordic input parameters (SCALE) and undone at the output.

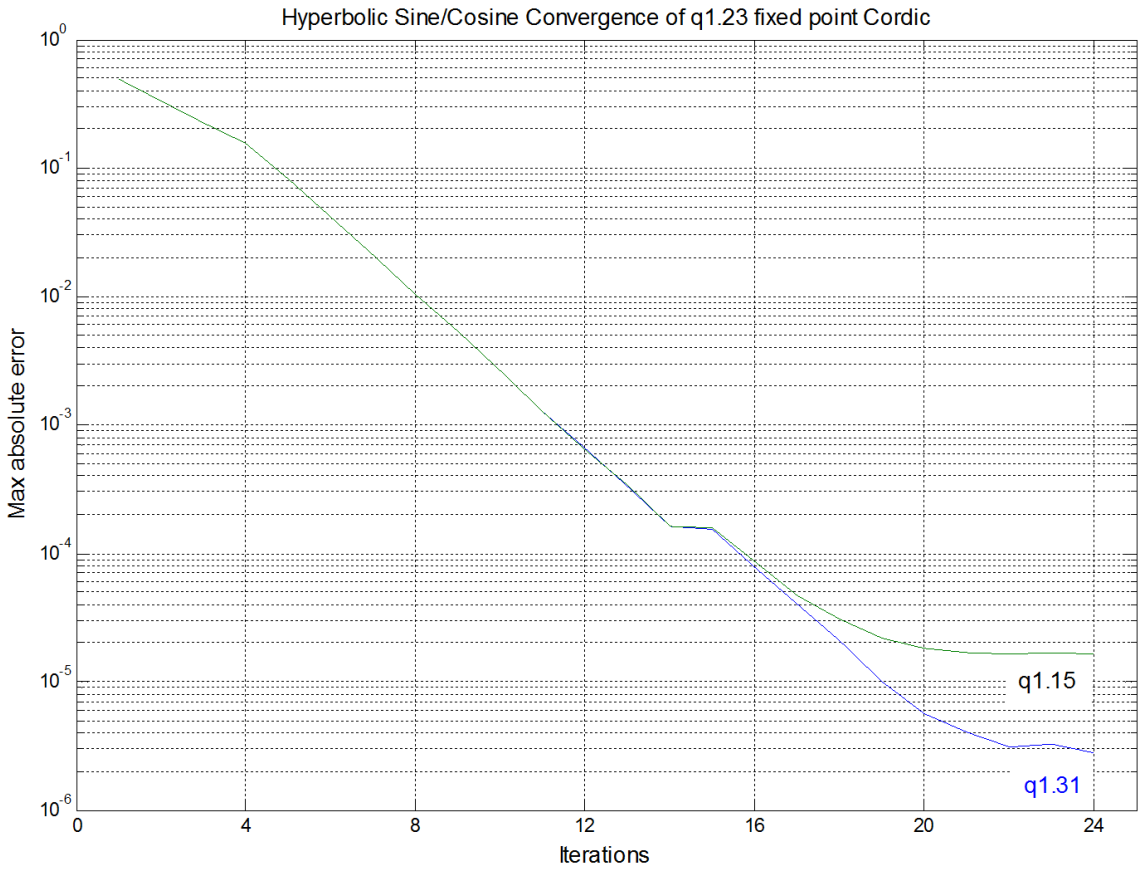
Precision

- Trigonometric functions (sine, cosine, phase, modulus)
  - The algorithm converges at a constant rate of one binary digit per iteration



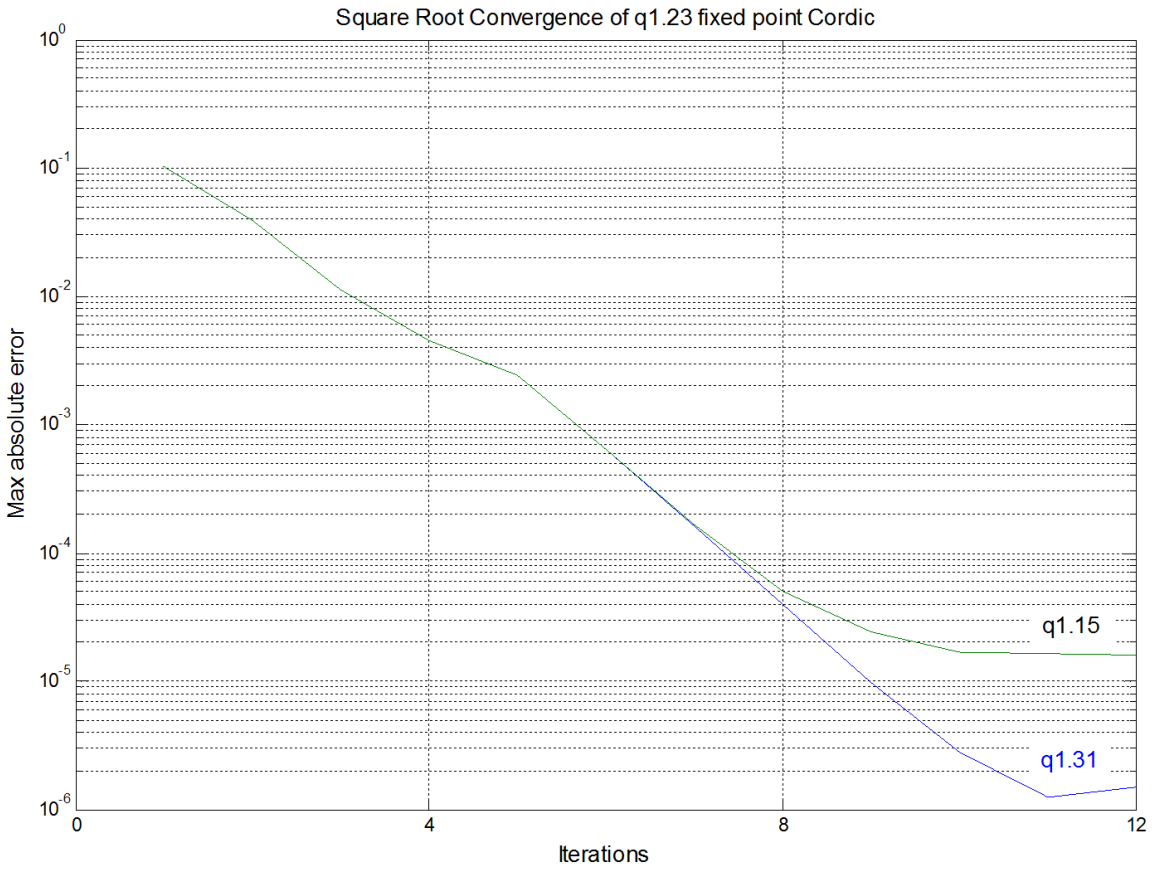
Precision

- Hyperbolic functions (hyperbolic sine, hyperbolic cosine, natural logarithm)
  - The algorithm converges at <1 binary digit per iteration



Precision

- Square root function
  - The algorithm converges at approximately twice the rate of the hyperbolic functions



Precision

- Internal word length
  - Numbers are represented internally in q1.23 format. This means that rounding errors start to become significant at a precision of 2^{-19}
  - Continuing Cordic iteration after the maximum precision has been reached will degrade the precision gradually.
- Input/output word length
  - For maximum precision, q1.31 format should be used for input and output (but output will be limited to 20-bit precision at best)
  - If q1.15 format is used for input, the precision will be limited to q1.15, whatever the output format.
- Iterations
  - The number of iterations can be specified when programming the Cordic, in multiples of 4. Four iterations can be executed in each clock cycle. For maximum speed, the minimum number of iterations for the required precision should be programmed.

Registers

- Control/Status register (CORDIC_CSR)
  - FUNC - select cordic function (4-bit)

    Sin, Cos, Mod, Atan2, Atan, Acos, Asin, Sinh/Cosh, Atanh, Ln, Sqrt
  - PRECISION – set required precision (4-bit)

    Number of clock cycles for the calculation (= number of iterations/4). This determines the precision.
  - SCALE - scaling factor (4-bit)

    Contains scale factor for arguments and results. A value n implies that the arguments have been multiplied by a factor 2^{-n}, and/or the results need to be multiplied by 2^{n}.
  - IEN - interrupt enable (1-bit)

    Enables the generation of an interrupt when a cordic operation finishes (RRDY flag is set).
  - DMAREN - DMA read channel enable (1-bit)

    Enables the generation of a DMA request to transfer the result to memory when a cordic operation finishes
  - DMAWEN - DMA write channel enable (1-bit)

    Enables the generation of a DMA request to start a new operation when the result of the previous operation has been read

Registers

- Control/Status register (CORDIC_CSR) continued
  - NRES – Number of results (1-bit)

    Selects whether one or two reads are required to the RDATA register to clear the RRDY flag
  - NARGS – Number of arguments (1-bit)

    Selects whether one or two writes are required to the WDATA register to start the calculation
  - RESSIZE – Width of output data (1-bit)

    Selects whether the RDATA register contains 32-bit (q1.31) data or 16-bit (q1.15) data
  - ARGSIZE – Width of input data (1-bit)

    Selects whether the WDATA register expects 32-bit (q1.31) data or 16-bit (q1.15) arguments
  - RRDY – Result ready flag (1-bit)

    Set when a cordic operation terminates, to indicate that a valid result is in the RDATA register. Reset by a read access to the RDATA register, if NRES = 0, or two reads if NRES = 1.

Registers

- Arguments
  - Cordic functions can take one or two arguments (ARG1, ARG2)
  - The domain (input range) is limited by the function, or else by the range of convergence of the Cordic algorithm.
- Results
  - Cordic functions deliver one or two results (RES1, RES2)

Fast math speed on Cortex-M4

- arm_sin_f32() : 52 cycles
- arm_sin_q31() : 41 cycles
- arm_sin_q15() : 36 cycles
- arm_sqrt_f32() : 25 cycles
- arm_sqrt_q31() : 98 cycles
- arm_sqrt_q15() : 101 cycles

using IAR EWARM v7.60 with optimisation: high speed, no size constraints

Note: arm_sqrt_f32() uses the VSQRT function of the FPU which takes 14 cycles

Cordic vs ARM fast math

- Buffered conversion
  - Convert a buffer of 3024 angles stored in memory to samples representing a 1kHz sine wave sampled at 48ksps, and store them back in memory.
  - Angles are pre-calculated by the CPU.

q1.15 fixed point: (precision = 4)

ARM fast math arm_sin_q15()	Cordic: zero-overhead mode	Cordic: DMA in/out mode
36 cycles/sample	7 cycles/sample	11 cycles/sample
-	x5 acceleration	x3 acceleration
100%CPU	100% CPU	0% CPU
Max error: 0.00012	Max error: 0.00004	Max error: 0.00004
-	x3 precision	x3 precision

q1.31 fixed point: (precision = 6)

ARM fast math arm_sin_q31()	Cordic: zero-overhead mode	Cordic: DMA in/out mode
41 cycles/sample	8 cycles/sample	12 cycles/sample
-	x5 acceleration	x3 acceleration
100%CPU	100% CPU	0% CPU
Max error: 0.00002	Max error: 0.000002	Max error: 0.000002
-	x10 precision	x10 precision

Cordic vs ARM fast math

- 32-bit floating point numbers can be converted to/from fixed point by software multiply and cast (uses Cortex-M4 FPU):

```
value_q31 = (int31_t)(value_f32*0x80000000); /* f32 to q1.31 */
value_f32 = (float)value_q31/(float)0x8000000; /* q1.31 to f32 */
```

32-bit floating point:
(precision = 6)

ARM fast math arm_sin_f32()	Cordic: zero-overhead mode
66 cycles/sample	21 cycles/sample*
-	x3 acceleration
100%CPU	100% CPU
Max error: 0.00002	Max error: 0.000002
-	x10 precision

*includes conversion from float32 to int32 and back

Cordic vs ARM fast math

- Buffered square root

Calculate the square roots of a buffer of 3024 values stored in memory and store them back in memory.

q1.15 fixed point:
(precision = 3)

ARM fast math arm_sqrt_q15()	Cordic: zero-overhead mode
101 cycles/sample	7 cycles/sample
-	x14 acceleration
100%CPU	100% CPU
Max error: 0.0002	Max error: 0.000015
-	x13 precision

q1.31 fixed point:
(precision = 3)

ARM fast math arm_sqrt_q31()	Cordic: zero-overhead mode
98 cycles/sample	7 cycles/sample
-	x14 acceleration
100%CPU	100% CPU
Max error: 0.000000004	Max error: 0.0000015
-	x0.003 precision

Cordic vs ARM fast math

- Buffered square root

32-bit floating point:
(precision = 3)

ARM fast math arm_sqrt_f32()	Cordic: zero-overhead mode
27 cycles/sample	21 cycles/sample*
-	x1.3 acceleration
100%CPU	100% CPU
Max error: 0.00000003	Max error: 0.0000015
-	x0.02 precision

*includes conversion from float32 to int32 and back

For floating point square root operations, there is little advantage in using the Cordic over the ARM fast math function (NB. sqrtf() function in math.h has similar performance).

Cordic vs ARM fast math

- Park transform (one-off calculation)

The Park transform is widely used in motor control applications:

$X = D.\cos\theta - Q.\sin\theta$

$Y = D.\sin\theta + Q.\cos\theta$

q1.15-bit fixed point:
(precision = 4)

ARM fast math arm_sqrt_q15()	Cordic: zero-overhead mode
243 cycles	48 cycles
-	x5 acceleration

Filter Math Accelerator (FMAC)

Requirements

- Target applications
  - Motor control
  - Audio
  - Power supply
  - Lighting
  - Analog sensing (health, fitness, robotics, ….)
- Offload CPU
  - Perform background signal filtering tasks autonomously
  - Minimize CPU intervention to free up CPU MIPs for other tasks
- General Purpose
  - Filter type, order, co-efficients etc should be programmable

Filter functions

- Classical filters
  - FIR, IIR
  - Low-pass, high-pass, band-pass, band-stop, compensator
- Correlator
  - Synchronization, radar/sonar pulse detection
  - Co-variance (for adaptive filter)
  - DFT, matched filter (optional)

Generic digital filter

- Filter equation

$$y[n] = \sum_{k=0}^{N} b[k]x[n-k] + \sum_{j=1}^{N} a[j]y[n-j]$$

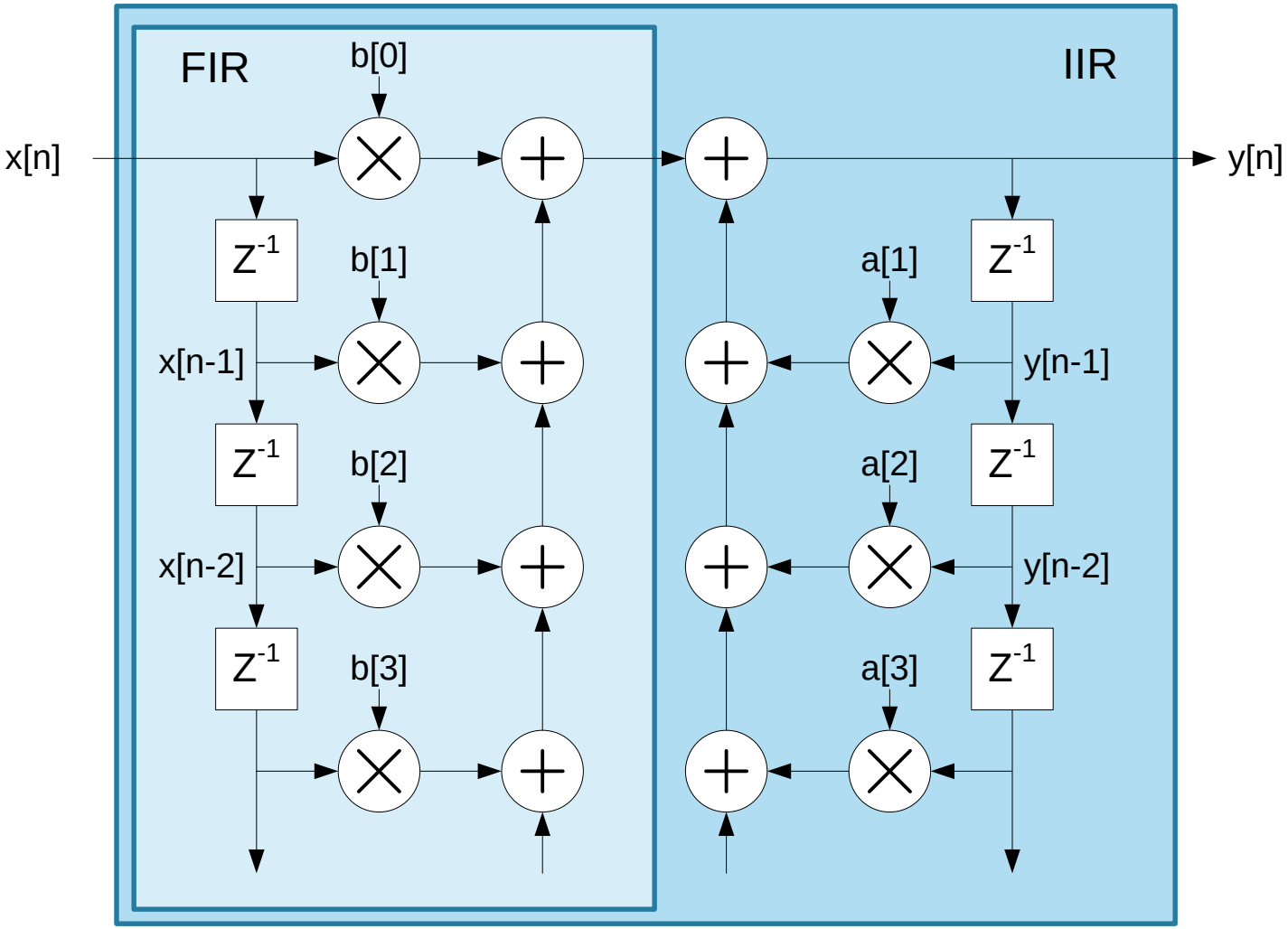
  - x[n] : n^{th} input sample
  - y[n] : n^{th} output sample
  - b[k] : k^{th} feed-forward coefficient
  - a[j] : j^{th} feed-back coefficient

- A finite impulse response (FIR) digital filter is a convolution between an input sampled data stream, x[n], and a set of coefficients b[k]

- An infinite impulse response (IIR) filter is the combination of an FIR with a second convolution, between the FIR output, y[n], and a second set of coefficients, a[j]

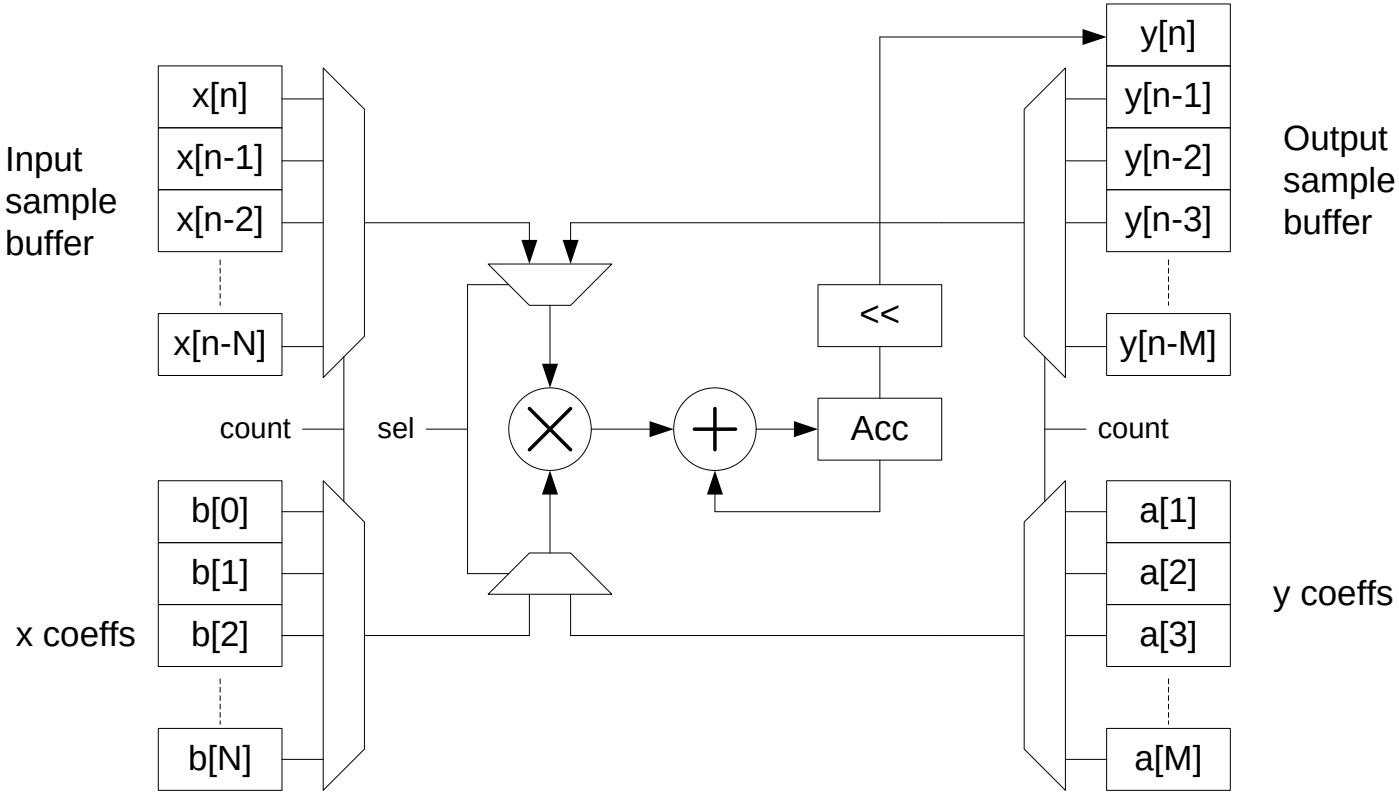
Generic digital filter

Direct Form 1 structure:

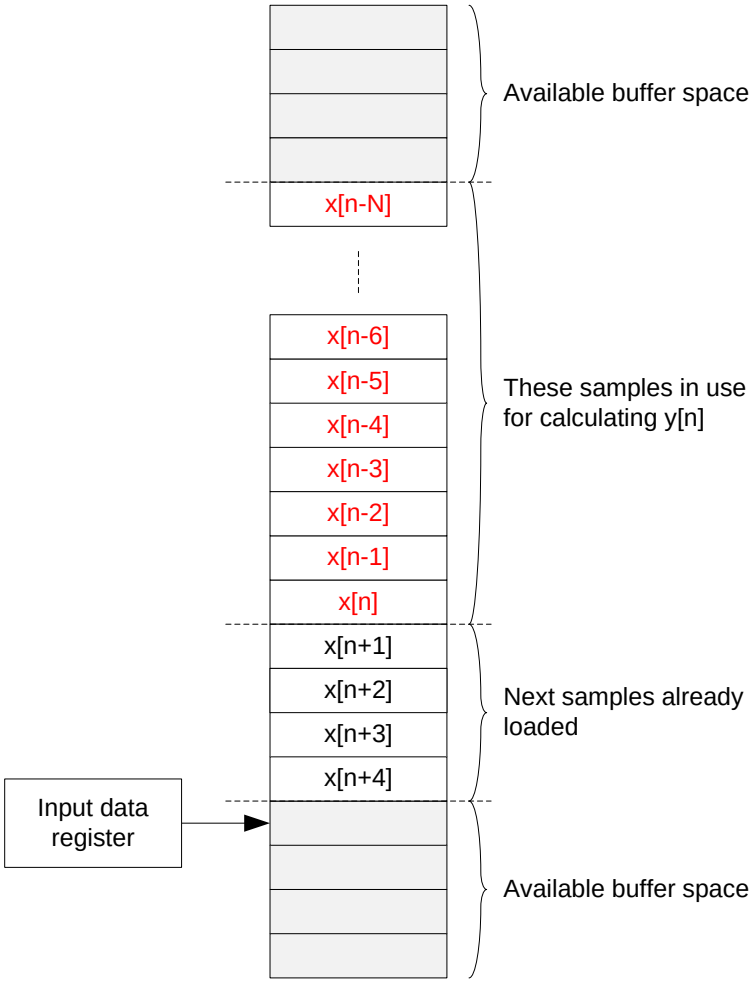


Single MAC architecture

- Sum of products implies N+M+1 multiply-accumulate (MAC) operations for each output sample
  - To save cost, a single MAC is used repeatedly
  - Requires N+M cycles per output sample



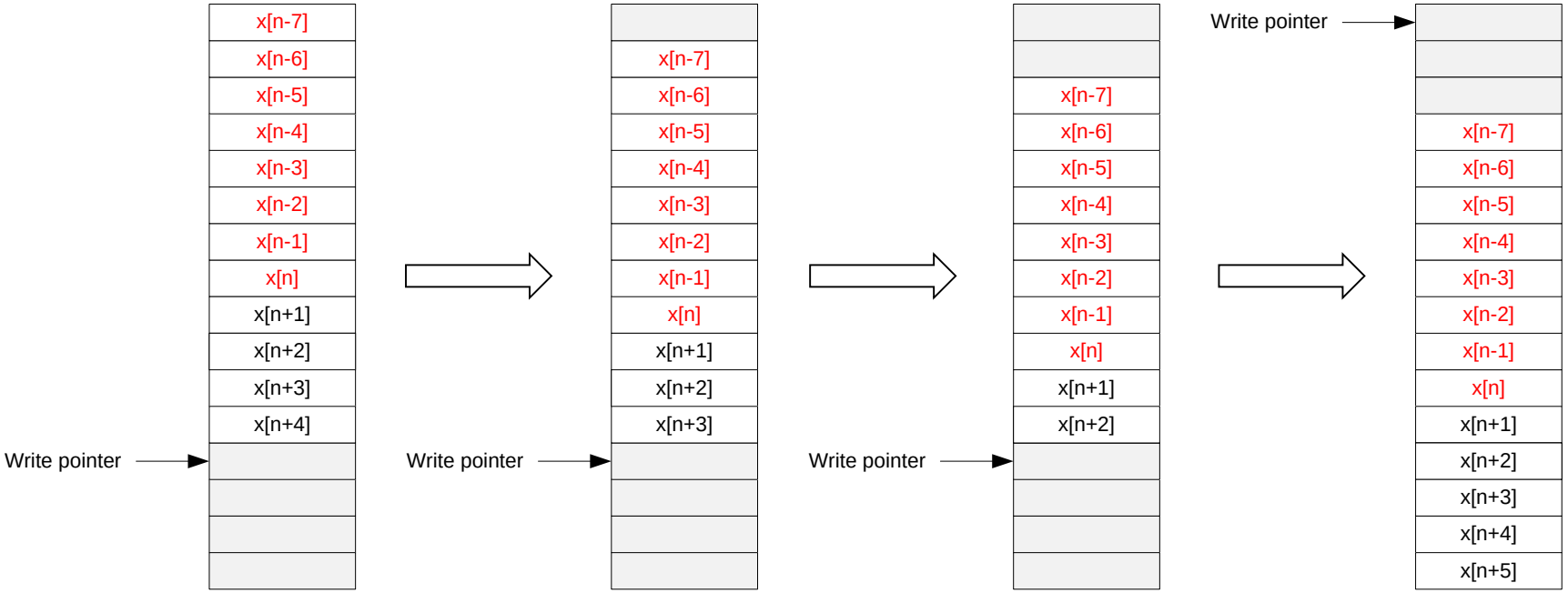
Input sample buffer



- Filter uses set of N+1 samples (input set) for calculating output sample y[n]
- When calculation is completed, the next sample (x[n+1]) is added to the input set, and the least recent sample (x[n-N]) is removed from it, freeing up a space in the buffer
- New samples arriving in the input data register are written into the buffer at the write pointer. This always indicates the first available space after the most recent sample in the input set, x[n]
- When the write pointer reaches the end of the buffer it wraps back to the beginning
- If available space in the buffer is less than the transfer size, the input buffer full flag is activated
- If the top of the input set, x[n] equals the write pointer (ie. no new sample available), the filter stalls until a new sample is available

Input buffer operation

Illustration showing filter with N=8 and 4-sample DMA transfers

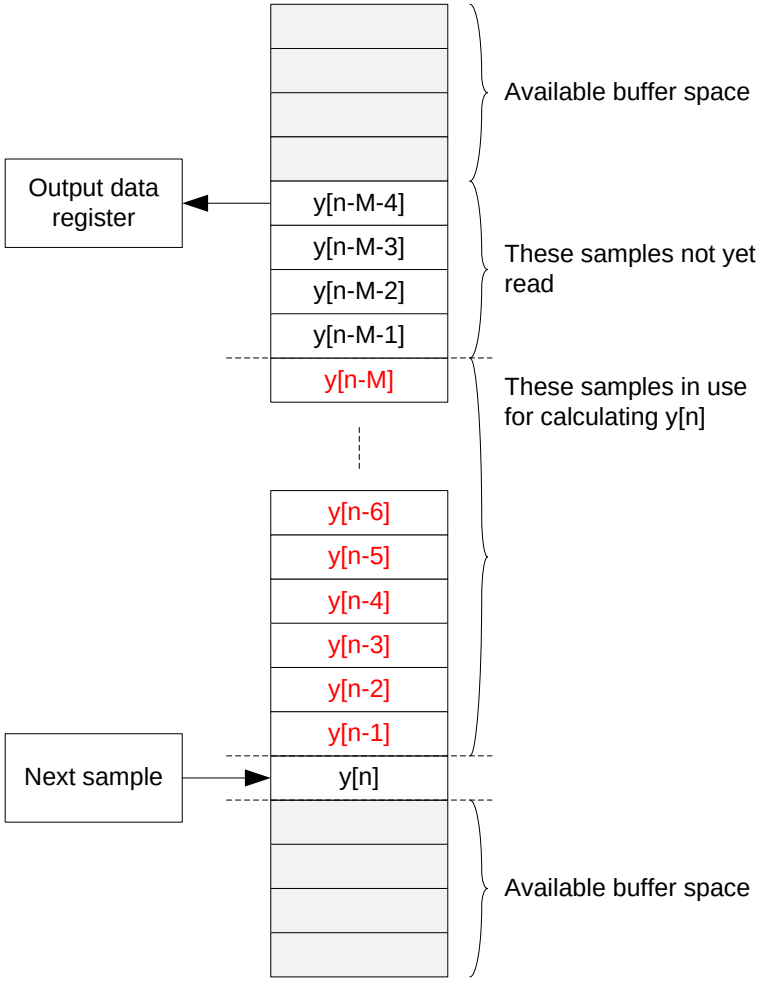


Filter using 8 samples x[n-7] to x[n]
Next 4 samples already loaded
4 buffer locations available so buffer full flag not set

Filter finishes current output sample and starts using next sample in buffer freeing up a space, since oldest sample is no longer needed

4 new samples written into buffer
Write pointer incremented by 4
If write pointer reaches the end it wraps to the beginning

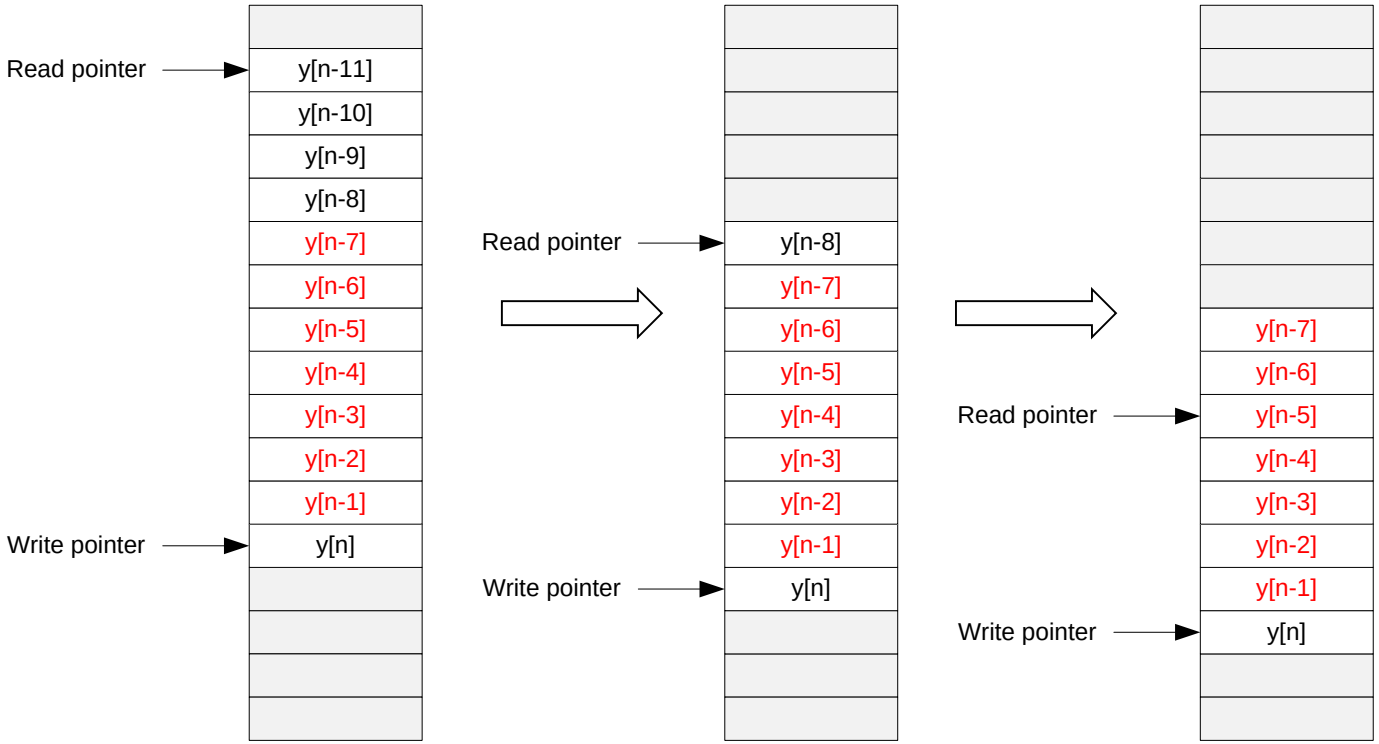
Output sample buffer



- Filter uses a set of M output samples (output set) for calculating output sample y[n]
- When calculation is completed, the new sample (y[n]) is added to the set, and the least recent sample (y[n-M]) is removed from it
- New samples are written into the buffer at the write pointer. This always indicates the first empty space after the most recent sample in the set, y[n-1]
- When the write pointer reaches the end of the buffer it wraps back to the beginning
- A read pointer designates the oldest un-read sample, corresponding to the output data register
- When a sample is read, and is not part of the output set, then the space becomes free
- If the write pointer equals the read pointer or the least recent sample in the output set (y[n-M]), the filter stalls and the output buffer full flag is set

Output buffer operation

Illustration showing IIR filter with M=7 and 4-sample DMA transfers



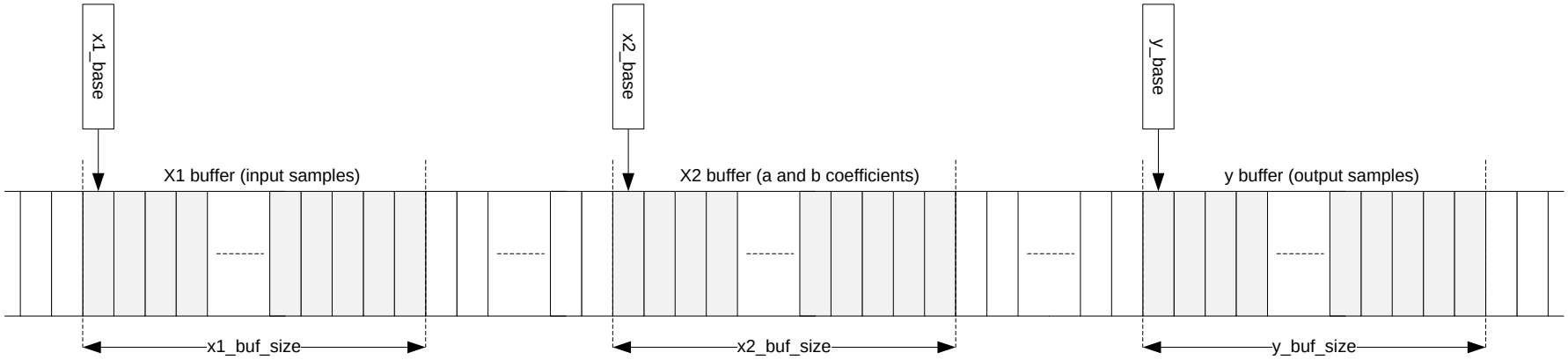
Filter using 7 samples y[n-7] to y[n-1]
Next sample written at write pointer
11 unread samples in buffer (read pointer is y[n-11])

New output sample is added to output set and write pointer incremented. 4 samples read from buffer and read pointer incremented by 4, freeing up all four spaces.

4 samples read from buffer and read pointer incremented by 4. Since read pointer points to sample in output set, only buffer locations up to y[n-7] become available.

Buffer configuration

- FMAC has 256x16-bit local storage memory for buffers
- 3 buffers are configured
  - X1 – Input samples (circular)
  - X2 – Coefficients (fixed)
  - Y – Output sample (circular)
- Base address and size of each buffer is configurable



Memory requirements

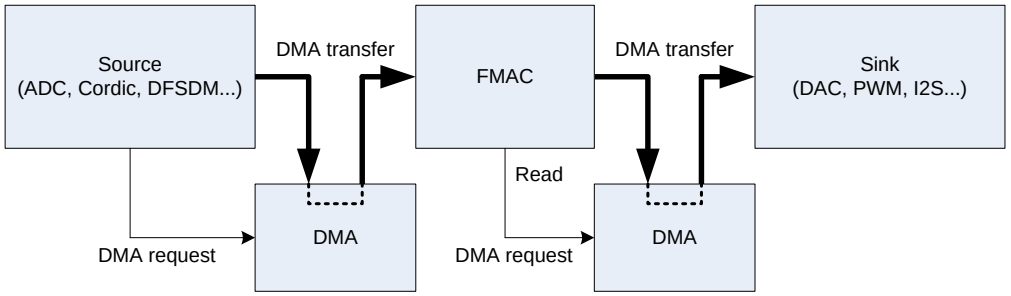
- FIR
  - N coefficients and N input samples to calculate one output sample
  - Each input sample is used N times
  - To optimize throughput, input buffer size should be N + ε. This allows ε samples to be loaded with one DMA request while not being used for calculation
  - Output buffer size should be ε to match DMA transfer size
  - Total memory requirement: $2(N + \varepsilon) < 256$
- IIR
  - N feed-forward coefficients and M feed-back coefficients (M < N)
  - N input samples and M output samples to calculate one output sample
  - DMA transfer size ε
  - Total memory requirement: $2(N + M + \varepsilon) < 256$

FMAC vs software

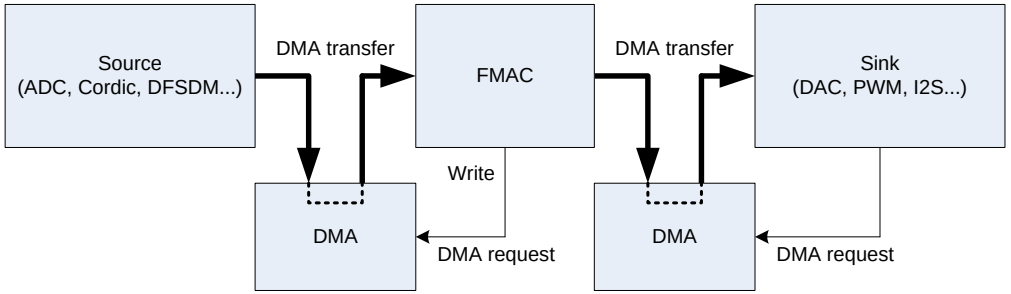
- Using ARM fast math library for comparison
  - on STM32G474 at 150MHz
  - IAR EWARM 7.82 with high speed optimization
- Single stage bi-quad/2p2z (IIR with N=3, M=2)
  - arm_biquad_cascade_df1_fast_q15():
    - 2048 samples processed in 22730 cycles ⇒ 12 cycles/sample
  - FMAC (using DMA in, DMA out):
    - 2048 samples processed in 27310 cycles ⇒ 14 cycles/sample
- 51-tap FIR (N=51)
  - arm_fir_fast_q15():
    - 2048 samples processed in 187418 cycles ⇒ 92 clock cycles/sample
  - FMAC (using DMA in, DMA out):
    - 2048 samples processed in 212683 cycles ⇒ 104 clock cycles/sample
- Software is ~15% faster due to dual MAC Cortex-M4
  - But for high sample rate/large filters, CPU spends a lot of time doing filtering task
  - With FMAC + DMA, CPU is free to perform other tasks

Flow control

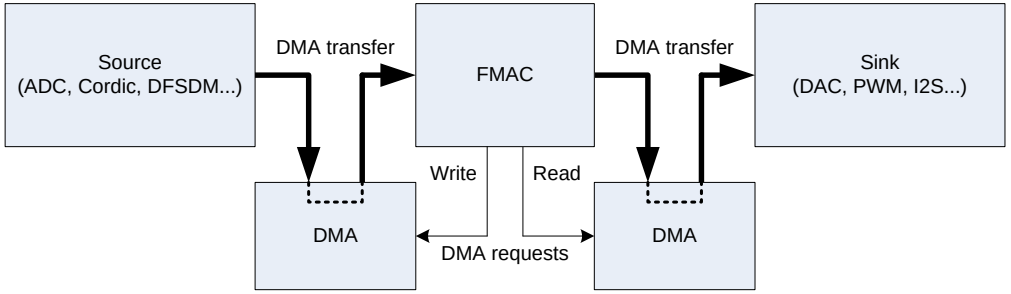
Source driven flow control



Sink driven flow control



Filter driven flow control



Data format

- The FMAC operates in fixed point signed integer format.
  - 32-bit floating point numbers can be converted to/from fixed point by software multiply and cast (uses Cortex-M4 FPU):

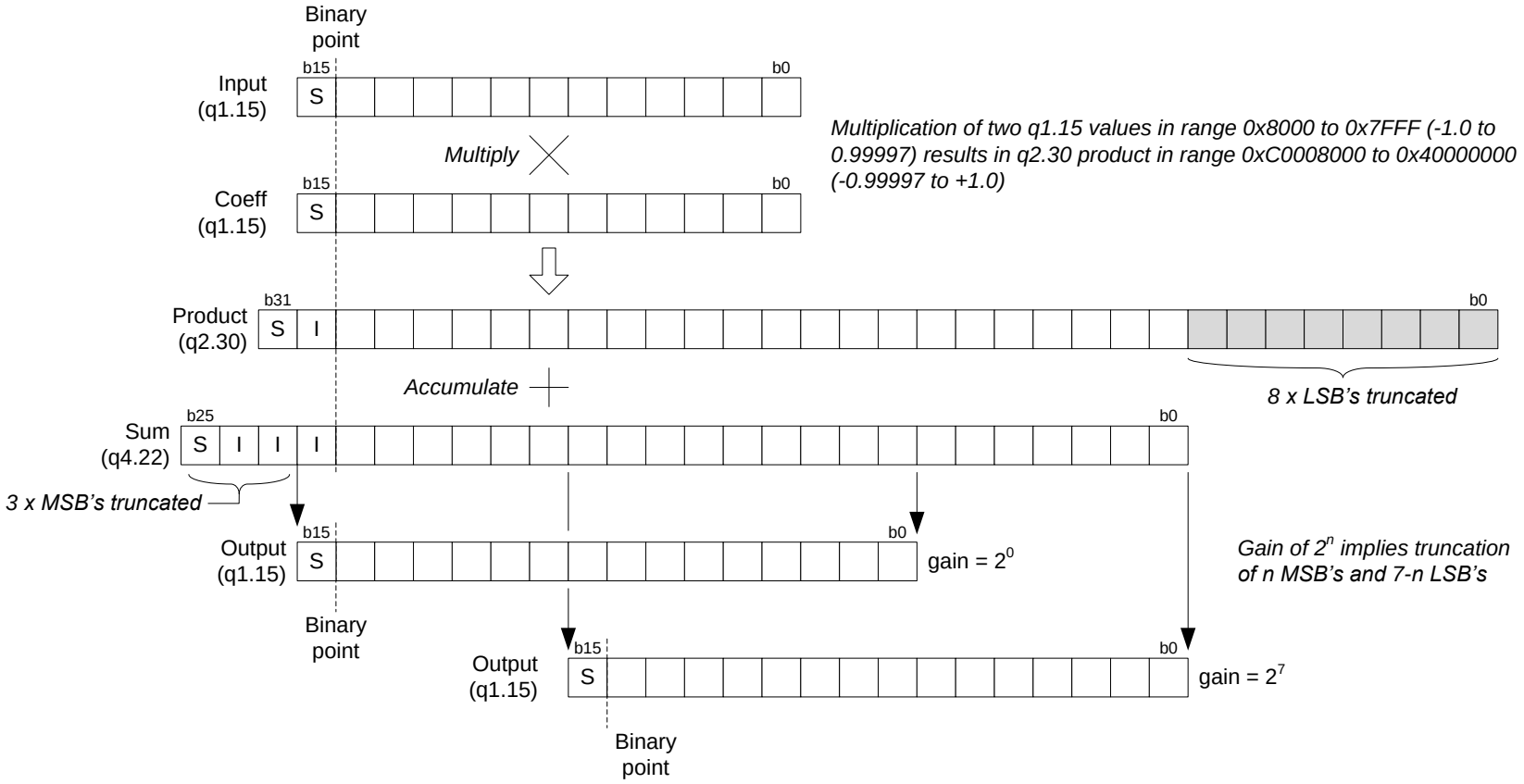
```
value_q15 = (int16_t)(value_f32*0x8000); /* f32 to q1.15 – takes 8 cycles */
value_f32 = (float)value_q15/(float)0x8000; /* q1.15 to f32 – takes 10 cycles */
```

- Input and output values are q1.15.
  - In q1.15 format, numbers are represented by one sign bit and 15 fractional bits (binary decimal places). The numeric range is therefore -1 (0x8000) to $1 - 2^{-15}$ (0x7FFF).

- The accumulator has 26 bits, of which 22 are fractional and 4 are integer/sign (q4.22).
  - The output of the multiplier, in q2.30 format, is truncated to q2.22 and added to the accumulator LSB aligned
  - The extra integer bits allow the accumulator to support partial accumulation sums in the range -8 (0x4000000) to +8 (0x3FFFFFF). This can occur if there are a large number of successive positive or negative coefficients. If the filter gain is less than unity for all frequencies, the accumulator value will always return to the range +/-1.
  - If the partial sum exceeds the accumulator numeric range (wraps), a sticky flag is set to help debugging. Nevertheless, provided subsequent additions undo the wrapping, a correct result is still obtained

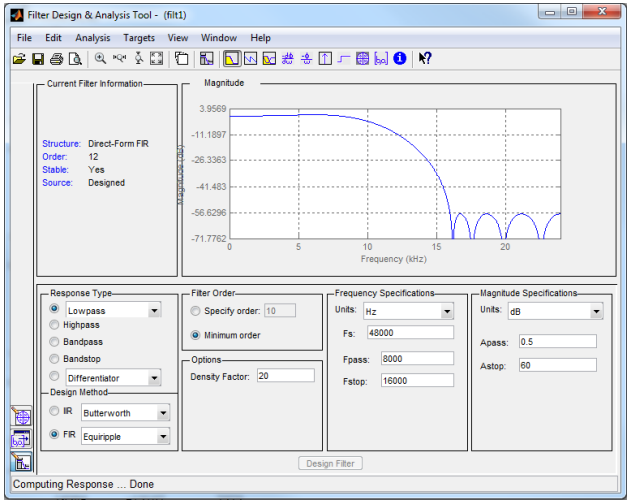
- A programmable gain can be applied at the output of the accumulator.
  - From 0dB to 42dB in steps of 6dB
  - This is necessary for IIR filter implementation

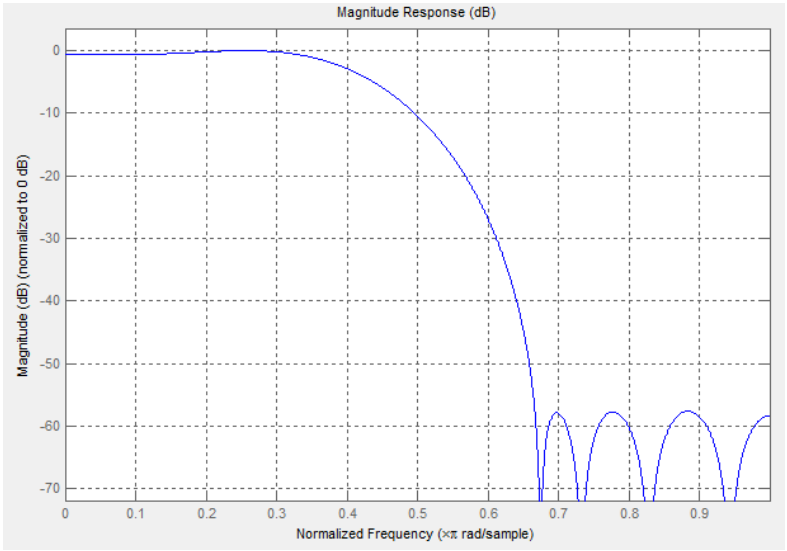
Data format


Binary point
Input (q1.15)
b15
b0
S
Multiply
Coeff (q1.15)
Product (q2.30)
b31
I
8 x LSB's truncated
Accumulate
Sum (q4.22)
b25
3 x MSB's truncated
Output (q1.15)
gain = 2^0
Gain of 2^n implies truncation of n MSB's and 7-n LSB's
gain = 2^7
Multiplication of two q1.15 values in range 0x8000 to 0x7FFF (-1.0 to 0.99997) results in q2.30 product in range 0xC0008000 to 0x40000000 (-0.99997 to +1.0)

FIR filter design

- Design the filter using Matlab or any filter design tool, and obtain the normalized coefficients (max gain = 0dB).
- Multiply the coefficients by 32768 and round to the nearest integer.
- Verify the filter still meets the specification.



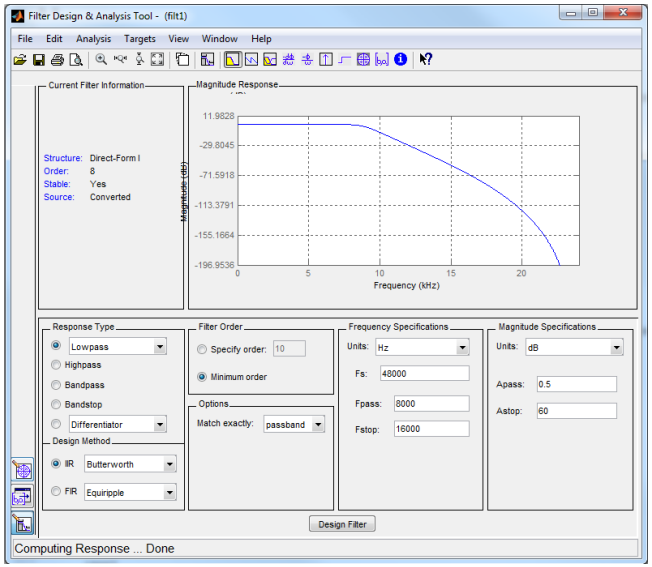


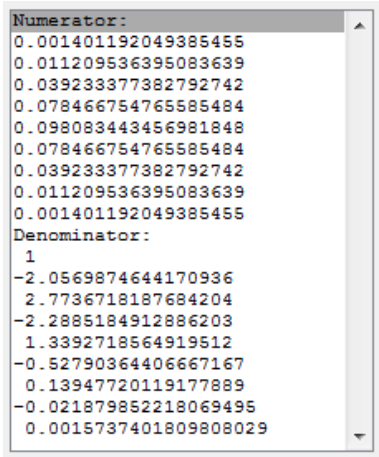

```
378
551
-994
-2620
998
9944
15027
9944
998
-2620
-994
551
378
```

```
Numerator:
 0.011540593420957625
 0.01681206074348255
-0.030331331589462183
-0.079964947624866028
 0.030468850234946283
 0.30346028569243411
 0.45858842593771432
 0.30346028569243411
 0.030468850234946283
-0.079964947624866028
-0.030331331589462183
 0.01681206074348255
 0.011540593420957625
```

IIR filter design

- Design the filter using Matlab or any filter design tool.
- Convert the structure to Direct Form 1, single section and obtain the coefficients
- Divide the denominator coefficients by 2^n (n=0,1,2…) so that all coefficients are < 1
  - Note that the 1 in the denominator becomes 2^{-n}
- Multiply all coefficients by 32768 and round to the nearest integer





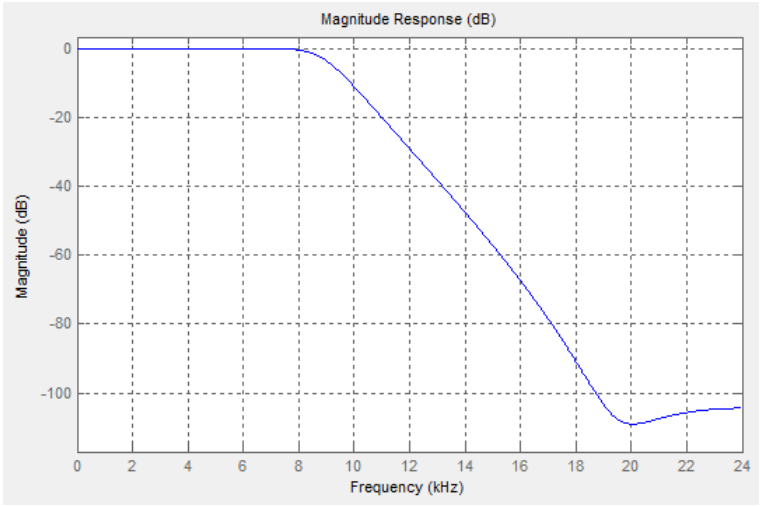
	Numerator	Denominator
46	0.001401192049385	0.250000000000000
367	0.011209536395084	-0.514246866104273
1286	0.039233377382793	0.693417954692105
2571	0.078466754765585	-0.572129622822155
3214	0.098083443456982	0.334817964122988
2571	0.078466754765585	-0.131975911016668
1286	0.039233377382793	0.034869300297945
367	0.011209536395084	-0.005469963054517
46	0.001401192049385	0.000393435045245

46	8192
367	-16851
1286	22722
2571	-18748
3214	10971
2571	-4325
1286	1143
367	-179
46	13

IIR filter design

- Verify the filter still meets the specification
- Remove the first denominator coefficient (2^{15-n}) and negate the rest:

Denominator	A coeffs
~~8192~~	
-16851	16851
22722	-22722
-18748	18748
10971	-10971
-4325	4325
1143	-1143
-179	179
13	-13



- A post-accumulator gain of 2^n must be configured
  - This is to compensate the effect of dividing the denominator by 2^n
  - The input signal should be attenuated by the same amount to avoid saturation of the output. Note that this will reduce the signal to noise ration by n x 6.02dB

Configuring the FMAC

- X1 (input sample) buffer configuration register, **FMAC_X1BUFCFG**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	FULL_WM		Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
						rw									

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X1_BUF_SIZE								X1_BASE							
rw								rw							

X1_BASE: buffer base address in the FMAC local memory

X1_BUF_SIZE: number of 16-bit words allocated to buffer. The minimum size is the number of feed forward taps (B coefficients) in the filter, plus the value of 2^{FULL_WM}.

FULL_WM: Watermark for buffer full flag. Defines the number of free spaces in the buffer below which the full flag will be set. Setting a watermark of X > 1 means that an interrupt occurs if there is space for X new data in the buffer, so X data can be transferred under one interrupt. This reduces the number of interrupts and hence improves the CPU efficiency. The watermark is not supported in DMA write mode and should be set to 1.

Configuring the FMAC

- X2 (coefficient) buffer configuration register, **FMAC_X2BUFCFG**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X2_BUF_SIZE								X2_BASE							
rw								rw							

X2_BASE: buffer base address in the FMAC local memory

X2_BUF_SIZE: number of 16-bit words allocated to buffer. The minimum size is the total number of coefficients (A and B).

Configuring the FMAC

- Y (Output sample) buffer configuration register, **FMAC_YBUFCFG**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	EMPTY_WM		Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
						rw									

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Y_BUF_SIZE								Y_BASE							
rw								rw							

Y_BASE: buffer base address in the FMAC local memory

Y_BUF_SIZE: number of 16-bit words allocated to buffer. The minimum size is the number of feed-back taps (A coefficients) in the filter, plus the value of 2^{EMPTY_WM}.

EMPTY_WM: Watermark for buffer empty flag. Defines the number of unread samples in the buffer below which the empty flag will be set. Setting a watermark of X > 1 means that an interrupt occurs if there are X new data in the buffer, so X data can be read out under one interrupt. This reduces the number of interrupts and hence improves the CPU efficiency. The watermark is not supported in DMA read mode and should be set to 1.

Configuring the FMAC

- FMAC control register, **FMAC_CR**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.		Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	RESET
															rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CLIP EN	Res.	Res.	Res.	Res.	Res.	DMA WEN	DMA REN	Res.	Res.	Res.	SAT IEN	UNFL IEN	OVFL IEN	WIEN	RIEN
rw						rw	rw				rw	rw	rw	rw	rw

RESET: Stops the filter and resets the sticky flags in the status register

CLIP_EN: Enables clipping (saturation) at the accumulator output, after applying the output gain and before truncation. Any values whose magnitude is ≥1 are set to 0x8000 (if negative) or 0x7FFF (if positive).

DMAWEN: Enable DMA mode for writing to the input buffers X1 and X2.

DMAREN: Enable DMA mode for reading from the output buffer, Y.

SATIEN: Generate an interrupt when the accumulator partial sum wraps.

UNFLIEN: Generate an interrupt when an underflow occurs on the X1 input buffer.

OVFLIEN: Generate an interrupt when an overflow occurs on the Y output buffer.

WIEN: Generate an interrupt when the input buffer is not full

RIEN: Generate an interrupt when the output buffer is not empty

Starting the FMAC

- FMAC parameter register, **FMAC_PARAM**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
START	FUNC							R							
rw	rw							rw							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Q								P							
rw								rw							

START: Starts the function selected by the FUNC field. Resetting it by software will stop any ongoing execution. It is reset automatically on completion of the initialization functions

FUNC: Selects the function to be executed: Load X1, Load X2, Load Y, FIR or IIR

P, Q and R: Input parameters. Their values depend on the function selected

Writing and reading data

- FMAC write data register, **FMAC_WDATA**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WDATA[15:0]															
w															

WDATA: 16-bit input data for writing into the memory. Depending on the function being executed, the data written to this register will be stored in one of the buffers, at the next available location.

- FMAC read data register, **FMAC_RDATA**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RDATA[15:0]															
r															

RDATA: 16-bit output data. The filter output samples are read out from this register, in the order in which they are generated.

FMAC status

- FMAC status register, **FMAC_SR**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	SAT	UNFL	OVFL	Res.	Res.	Res.	Res.	Res.	Res.	X1 FULL	Y EMPTY
					r	r	r							r	r

SAT: Saturation flag. This flag is set when the result of an accumulation exceeds the numeric range of the accumulator (0x2000000 to 0x1FFFFFF), causing it to wrap.

UNFL: Underflow flag. This flag is set when a read access is made to the output data register (FMAC_RDATA) when there is no new data in the Y buffer.

OVFL: Overflow flag. This flag is set when a write access is made to the input data register (FMAC_WDATA) when there is no free space in the X1 buffer.

X1FULL: X1 buffer full flag. This flag is set if the number of available spaces is less than the FULL_WM threshold.

YEMPTY: Y buffer empty flag. This flag is set if the number of unread data is less than the EMPTY_WM threshold.

Note: SAT, UNFL and OVFL are sticky: they can only be reset by the software writing to the RESET bit in the control register.

Releasing Your Creativity

/STM32 @ST_World community.st.com stm32g4-online-training

www.st.com/STM32G4

life.augmented