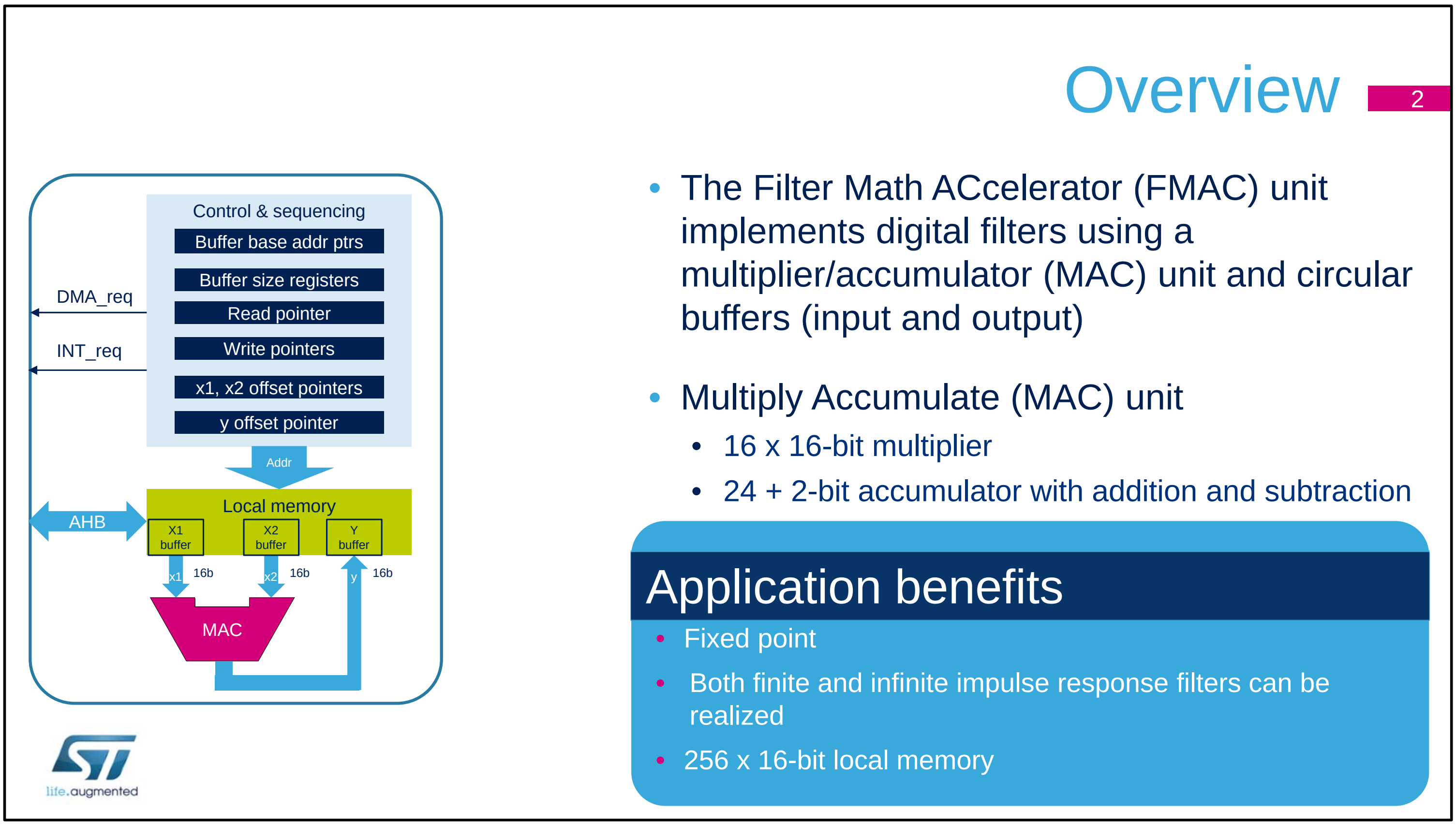
STM32H7 - FMAC

Filter Math Accelerator

Revision 1.0

life.augmented

Hello and welcome to this presentation of the STM32H7 FMAC block.
It covers the main features of this block, which is used to perform background signal filtering tasks autonomously.



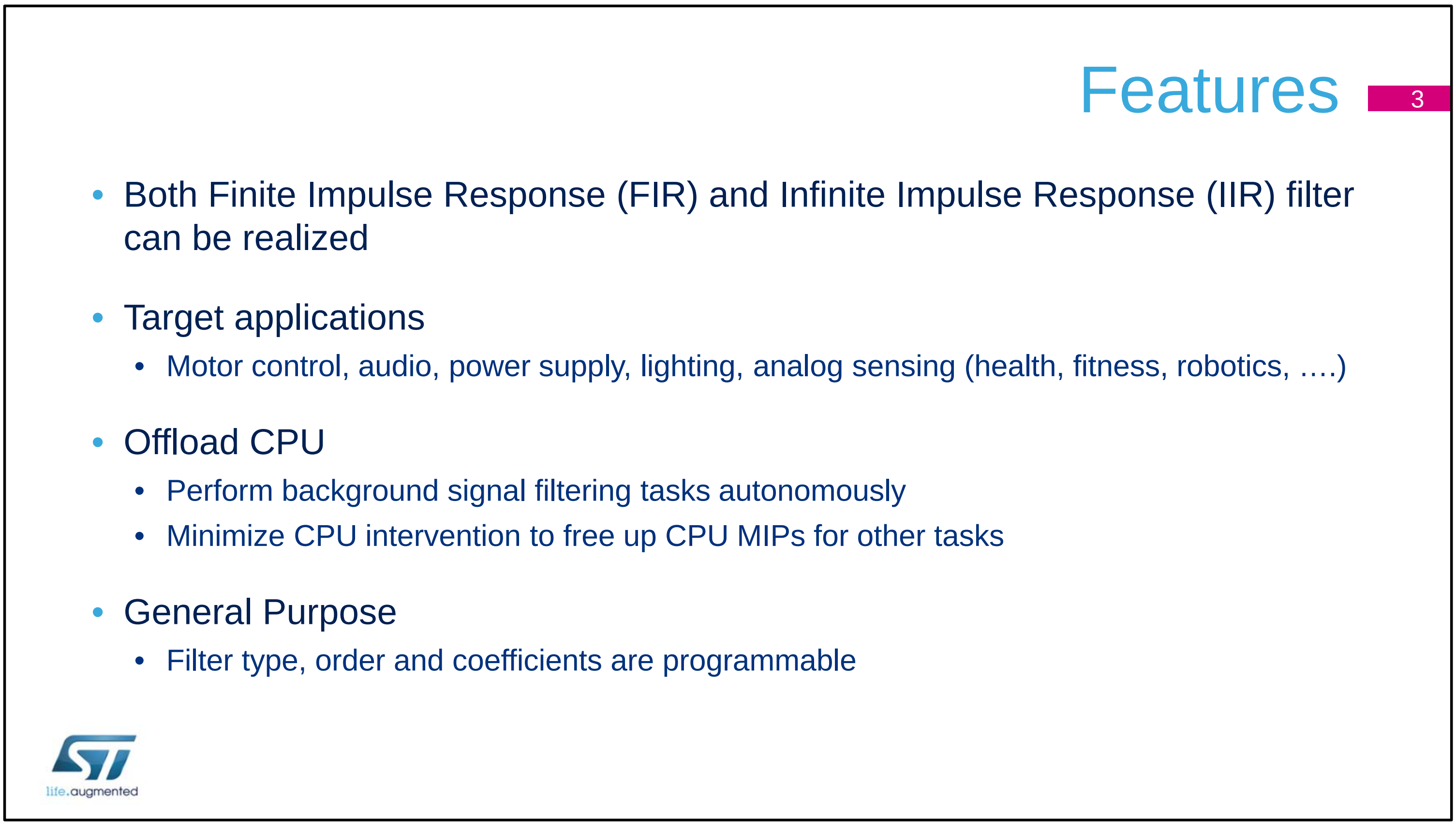
Overview

- The Filter Math ACcelerator (FMAC) unit implements digital filters using a multiplier/accumulator (MAC) unit and circular buffers (input and output)
- Multiply Accumulate (MAC) unit
  - 16 x 16-bit multiplier
  - 24 + 2-bit accumulator with addition and subtraction

Application benefits

- Fixed point
- Both finite and infinite impulse response filters can be realized
- 256 x 16-bit local memory

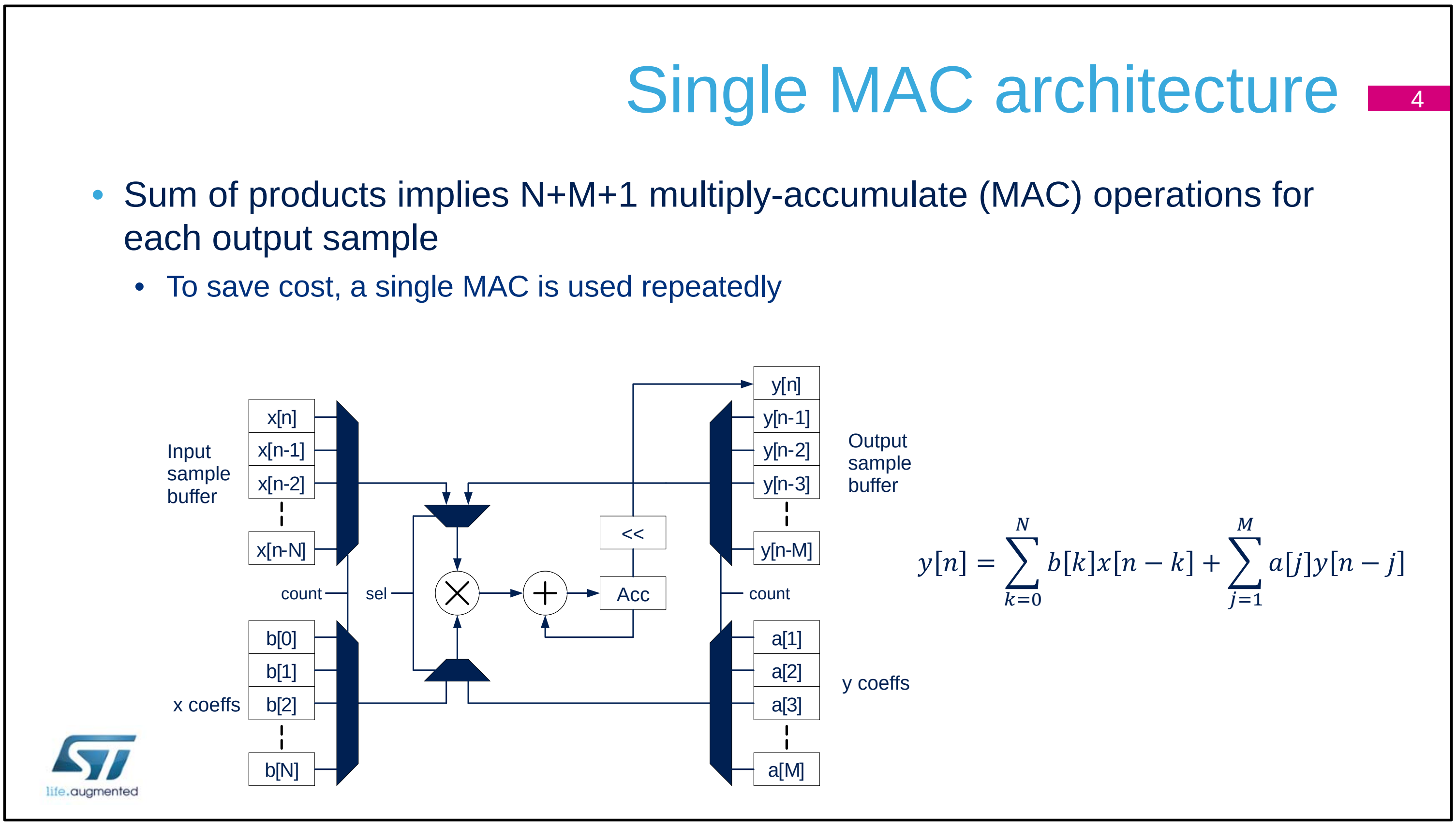
The FMAC unit is built around a fixed point multiplier and accumulator (MAC).
The MAC units receives two fixed-point 16-bit operands from an internal 256x16-bit RAM and write the result back to this memory.
The address of the input values in local memory is determined using a set of pointers.
These pointers can be loaded, incremented, decremented or reset by the internal hardware.
Software does not access them directly.
The unit allows frequent or lengthy filtering operations to be offloaded from the CPU, freeing up the processor for other tasks.



Features

- Both Finite Impulse Response (FIR) and Infinite Impulse Response (IIR) filter can be realized
- 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 and coefficients are programmable

Filter functions FIR and IIR can be realized by the FMAC.
Typical applications requiring these filters are motor control, audio, power supply, lighting and analog sensing.
The FMAC offloads the CPU by executing background signal filtering tasks autonomously, thus freeing up the CPU MIPs for other tasks.
The FMAC unit enables the user to select the filter type, the filter order and the coefficients, which are all programmable.



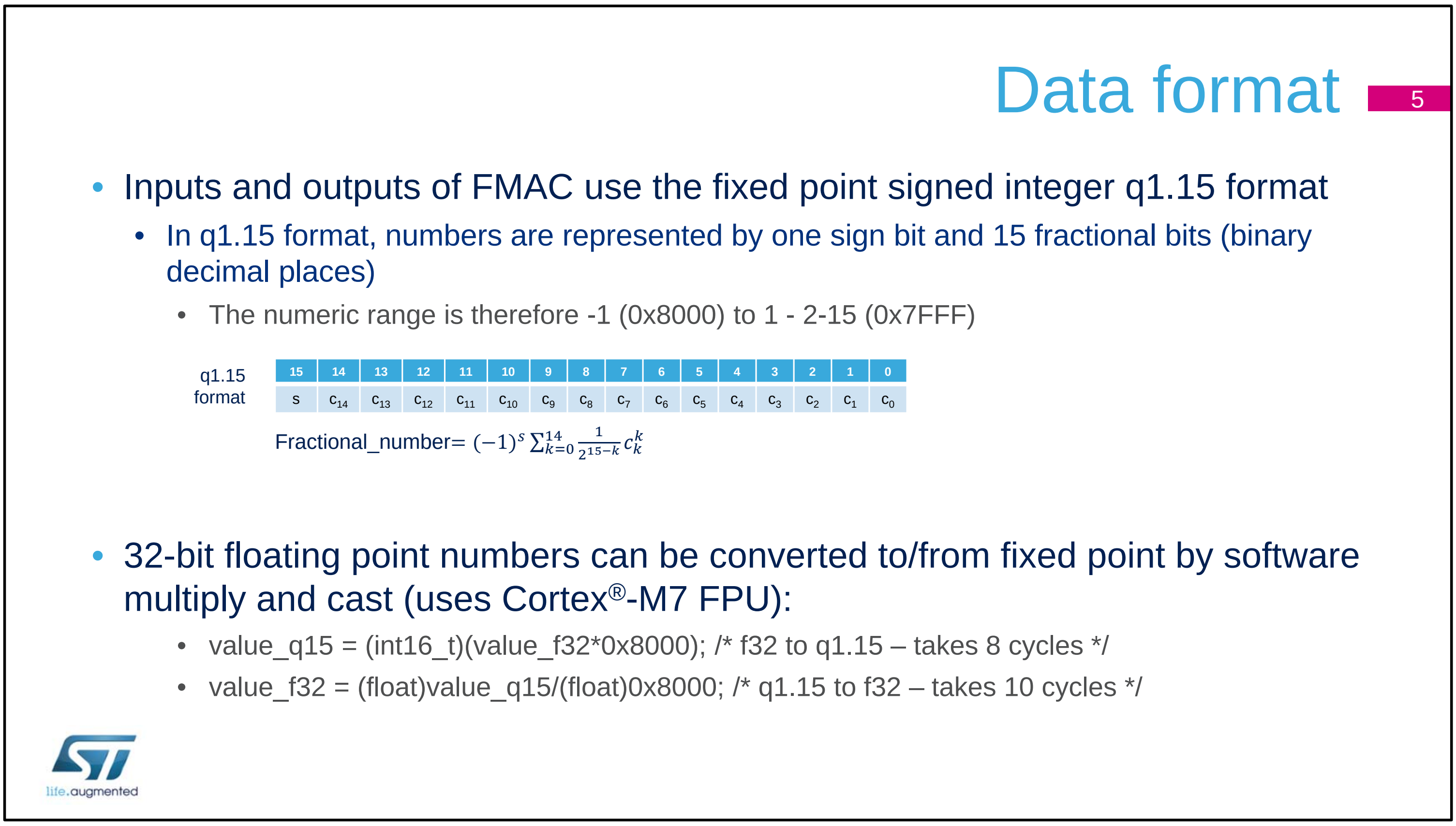
Single MAC architecture

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

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

This figure details the architecture of the MAC unit.
X is the input sample buffer containing the raw samples to be filtered.
B is the array of coefficients of the filter to be applied to X samples.
X and B have the same size: N+1 entries.
Y is the output sample buffer containing the results of the filtering.
A is the array of coefficients of the filter to be applied to Y samples.
Y and A have the same size: M+1 entries.
The number of MAC operations to obtain y[n] = (N+1) + M:

- N+1 MACs to multiply accumulate vector X and vector B.
- M MACs to multiply accumulate vector Y[n-1:n-M] and vector A.



Data format

5

- Inputs and outputs of FMAC use the fixed point signed integer q1.15 format
  - 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)

q1.15 format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
s	c_{14}	c_{13}	c_{12}	c_{11}	c_{10}	c_9	c_8	c_7	c_6	c_5	c_4	c_3	c_2	c_1	c_0

Fractional_number= $(-1)^s \sum_{k=0}^{14} \frac{1}{2^{15-k}} c_k^k$

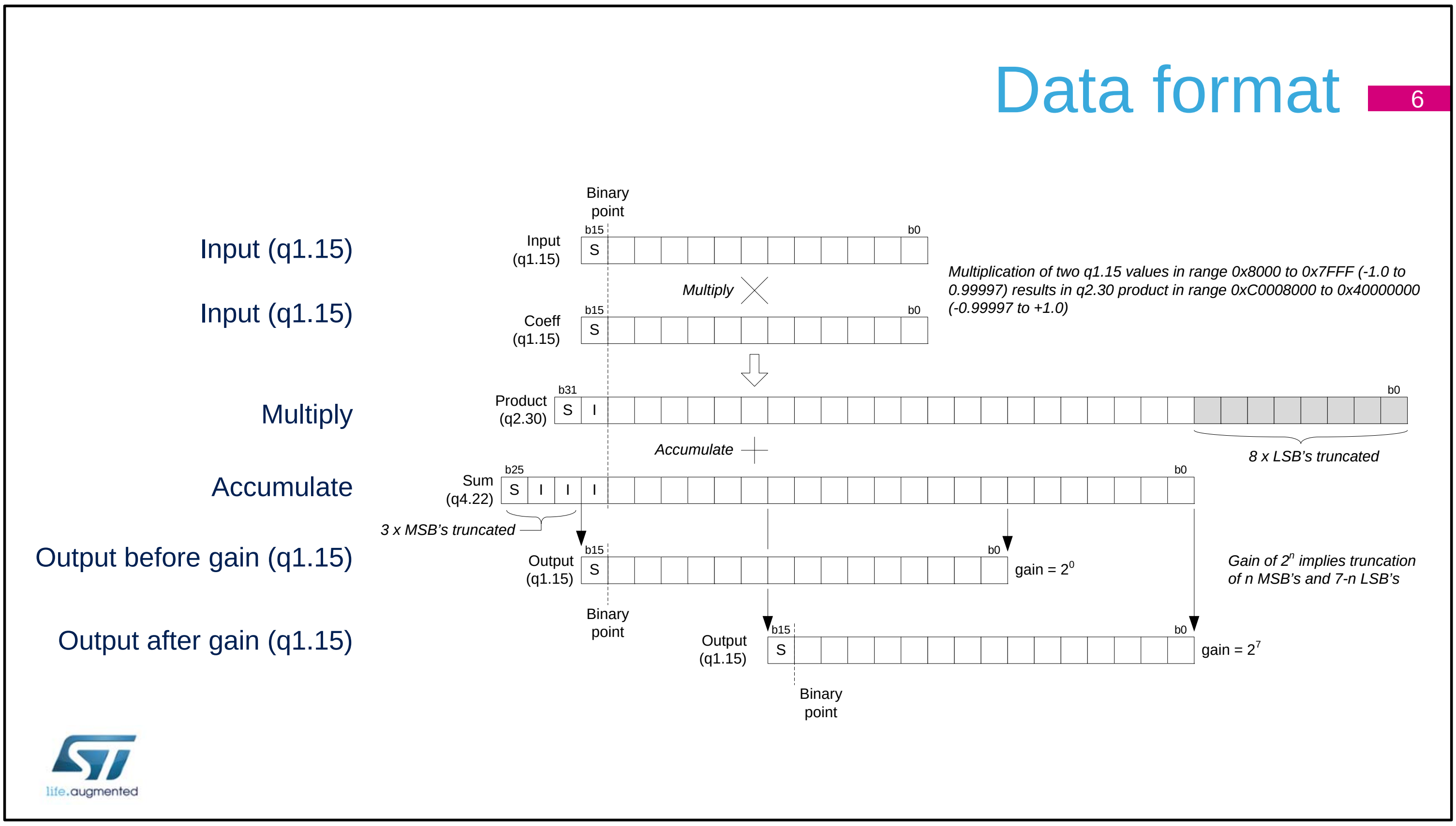
- 32-bit floating point numbers can be converted to/from fixed point by software multiply and cast (uses Cortex®-M7 FPU):
    - value_q15 = (int16_t)(value_f32*0x8000); /* f32 to q1.15 – takes 8 cycles */
    - value_f32 = (float)value_q15/(float)0x8000; /* q1.15 to f32 – takes 10 cycles */

life.augmented

Inputs and outputs of FMAC use the fixed-point signed integer q1.15 format

In q1.15 format, the numeric range is 1 (0x8000) to 1 - 2^{-15} (0x7FFF).

72-bit single precision floating-point numbers can be converted to or from q1.15 format by dedicated conversion instructions that are executed in the Cortex-M7 FPU.



Data format

6

Input (q1.15)

Input (q1.15)

Multiply

Accumulate

Output before gain (q1.15)

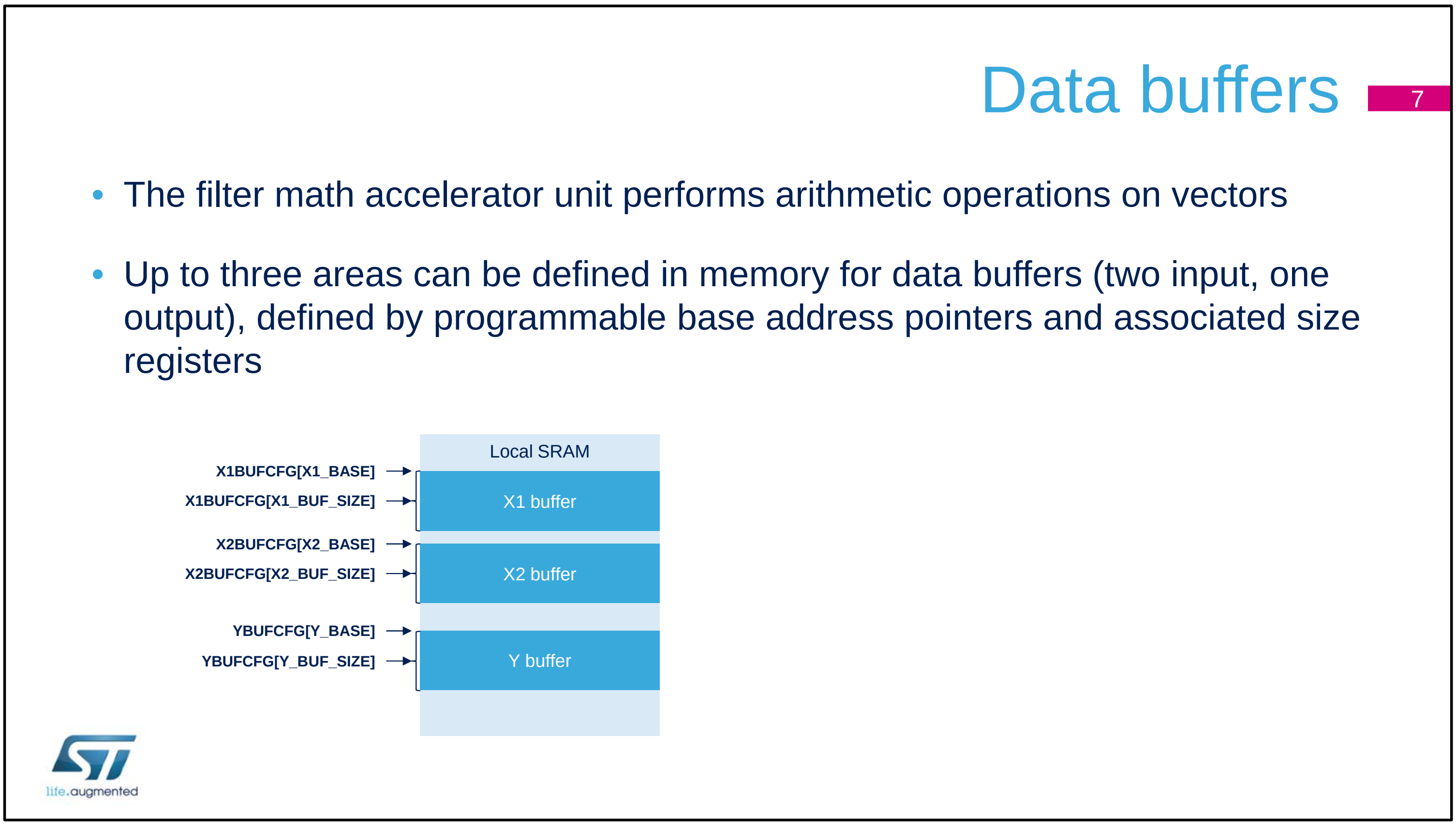
Output after gain (q1.15)

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)

Gain of 2^n implies truncation of n MSB's and 7-n LSB's

This figure details the various formats used internally by the FMAC.
The output of the multiplier, in q2.30 format, is truncated to q2.22 and added to the accumulator LSB aligned.
The accumulator has 26 bits, of which 22 are fractional and 4 are integer/sign (q4.22).
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.
When the filter gain is less than unity for all frequencies, the accumulator value always returns 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

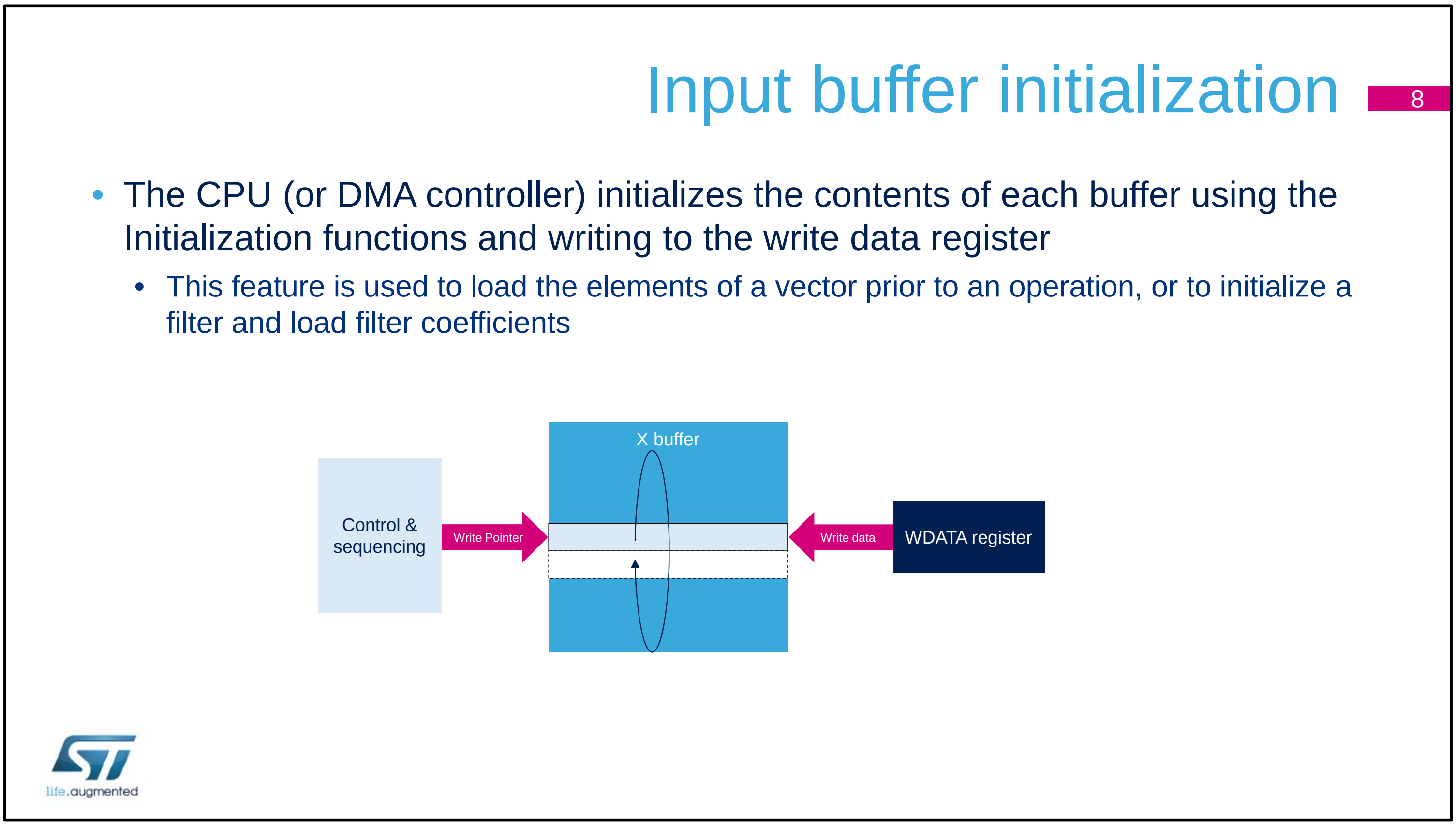


Data buffers

7

- The filter math accelerator unit performs arithmetic operations on vectors
- Up to three areas can be defined in memory for data buffers (two input, one output), defined by programmable base address pointers and associated size registers

The FMAC unit performs arithmetic operations on vectors, which are arrays of 16-bit fixed-point scalar values.
These vectors are allocated in the local SRAM.
Software is in charge of configuring the X1 and X2 operand buffers and Y output buffer through X1BUFCFG, X2BUFCFG and YBUFCFG registers.
The base addresses can be chosen anywhere in internal memory, provided that all buffers fit within the internal memory address range (0x00 to 0xFF).
Buffer base address and size have to be programmed.
Note that X1, X2 and Y buffers may overlap.
These buffers are not visible in the CPU mapping.



Input buffer initialization

- The CPU (or DMA controller) initializes the contents of each buffer using the Initialization functions and writing to the write data register
  - This feature is used to load the elements of a vector prior to an operation, or to initialize a filter and load filter coefficients

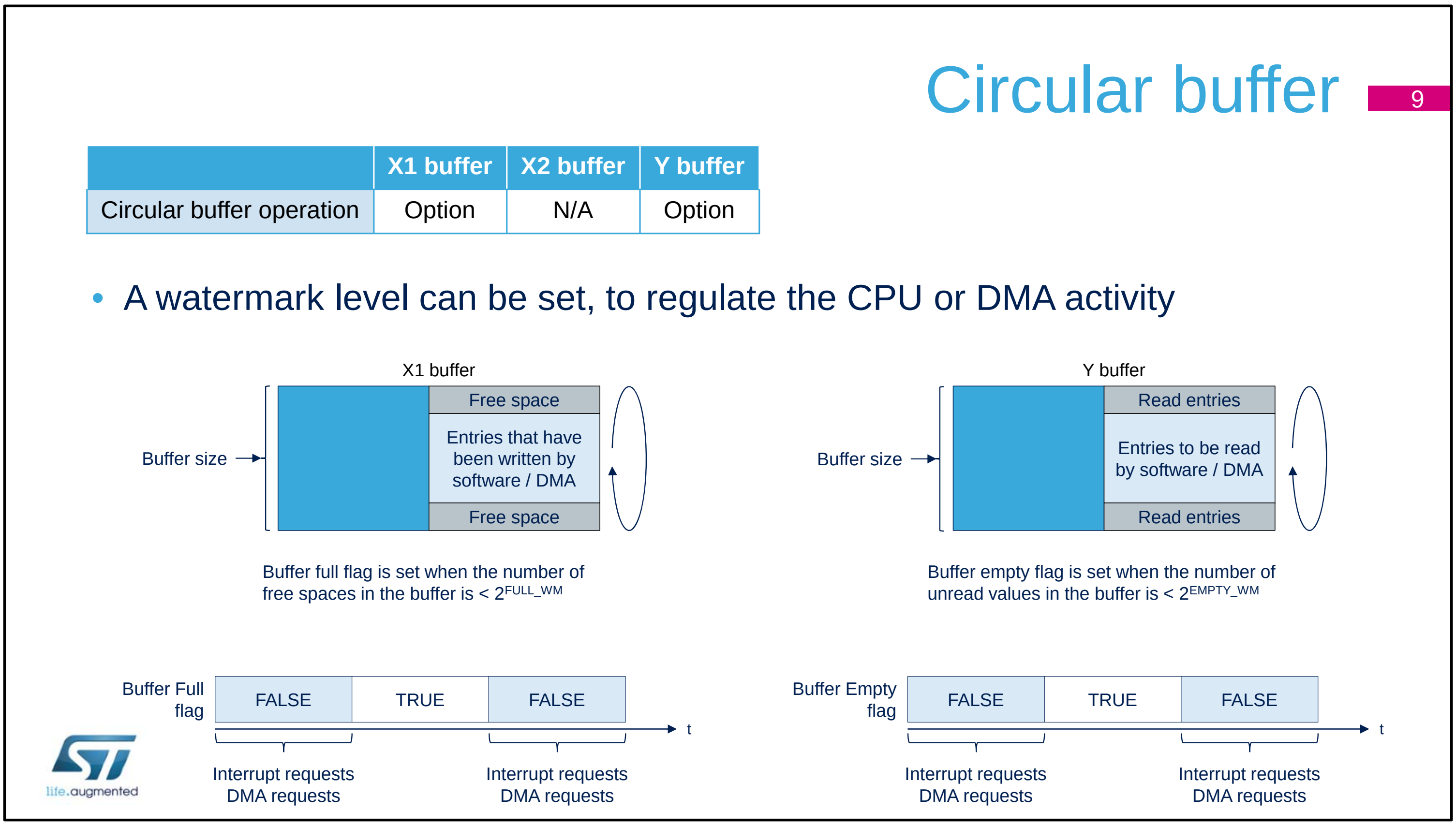
Before starting a filtering operation, the CPU or DMA controller initializes the contents of input buffers using the Initialization functions and writing to the WDATA register.

The contents of input buffers can be either data to be filtered or filter coefficients.

The data is transferred to the location within the target buffer indicated by a write pointer.

After each new write, the write pointer is incremented.

When the write pointer reaches the end of the allocated buffer space, it wraps back to the base address.



Circular buffer

9

	X1 buffer	X2 buffer	Y buffer
Circular buffer operation	Option	N/A	Option

- A watermark level can be set, to regulate the CPU or DMA activity

X1 buffer

Buffer size

Free space

Entries that have been written by software / DMA

Free space

Buffer full flag is set when the number of free spaces in the buffer is < 2^{FULL_WM}

Y buffer

Buffer size

Read entries

Entries to be read by software / DMA

Read entries

Buffer empty flag is set when the number of unread values in the buffer is < 2^{EMPTY_WM}

Buffer Full flag

FALSE	TRUE	FALSE

t

Interrupt requests DMA requests

Interrupt requests DMA requests

Buffer Empty flag

FALSE	TRUE	FALSE

t

Interrupt requests DMA requests

Interrupt requests DMA requests

life.augmented

Regarding the X1 buffer, if the number of free spaces in the buffer is less than the watermark threshold programmed in the FULL_WM field of the FMAC_X1BUFCFG register, the buffer is flagged as full As long as the full flag is not set, interrupts or DMA requests are generated, if enabled, to request more data for the buffer.

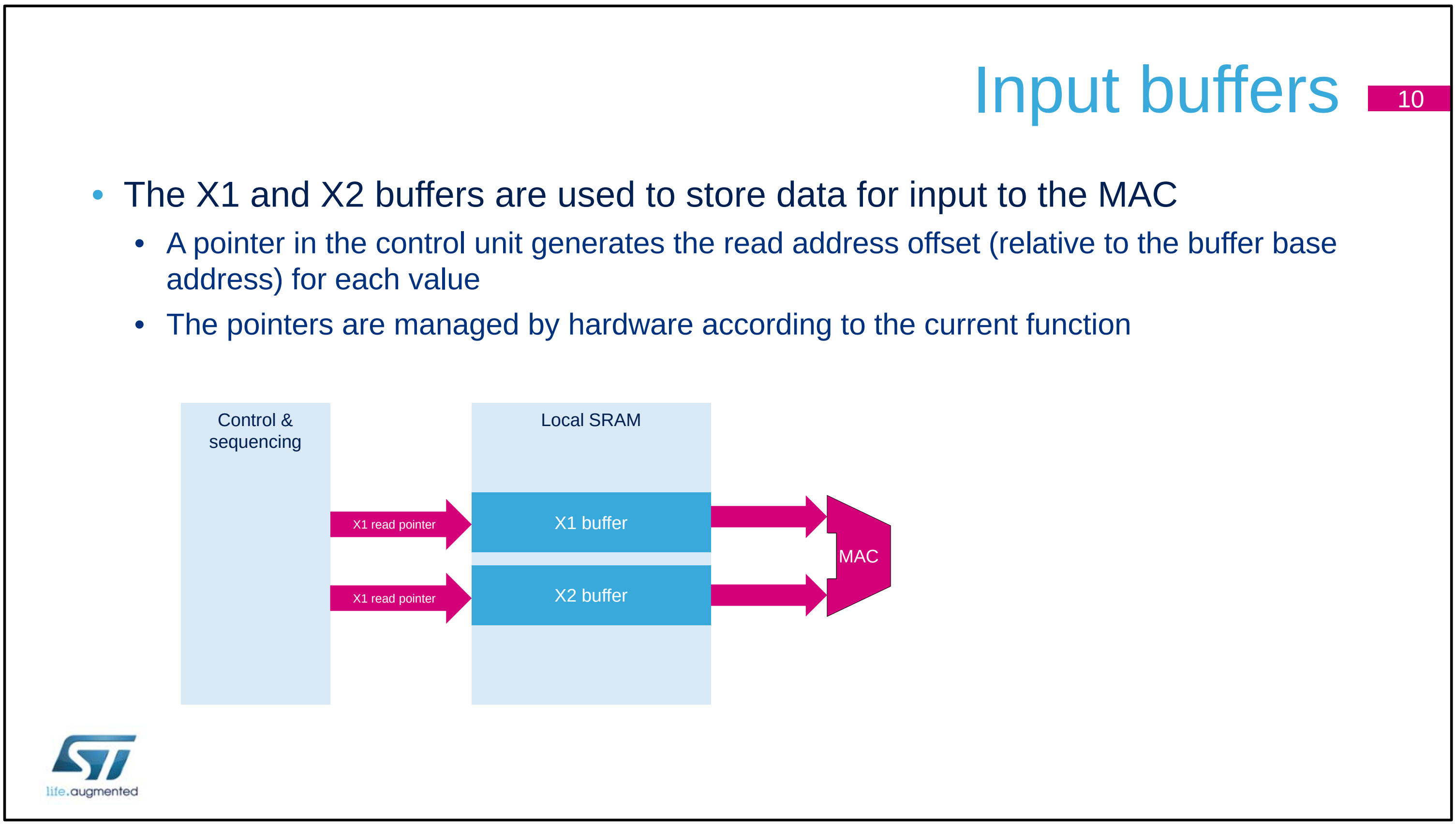
The watermark allows several data to be transferred under one interrupt, without danger of overflow.

Nevertheless, if an overflow does occur, the OVFL error flag is set and the write data is ignored.

The write pointer is not incremented in the event of an overflow.

Regarding the Y buffer, if the number of unread data in the buffer is less than the watermark threshold programmed in the EMPTY_WM field of the FMAC_YBUFCFG register, the buffer is flagged as

empty.
As long as the empty flag is not set, interrupts or DMA requests are generated, if enabled, to request reads from the buffer.
The watermark allows several data to be transferred under one interrupt, without danger of underflow.
Nevertheless, if an underflow does occur, the UNFL error flag is set.
In this case, the read pointer is not incremented and the read operation returns the content of the memory at the read pointer address.



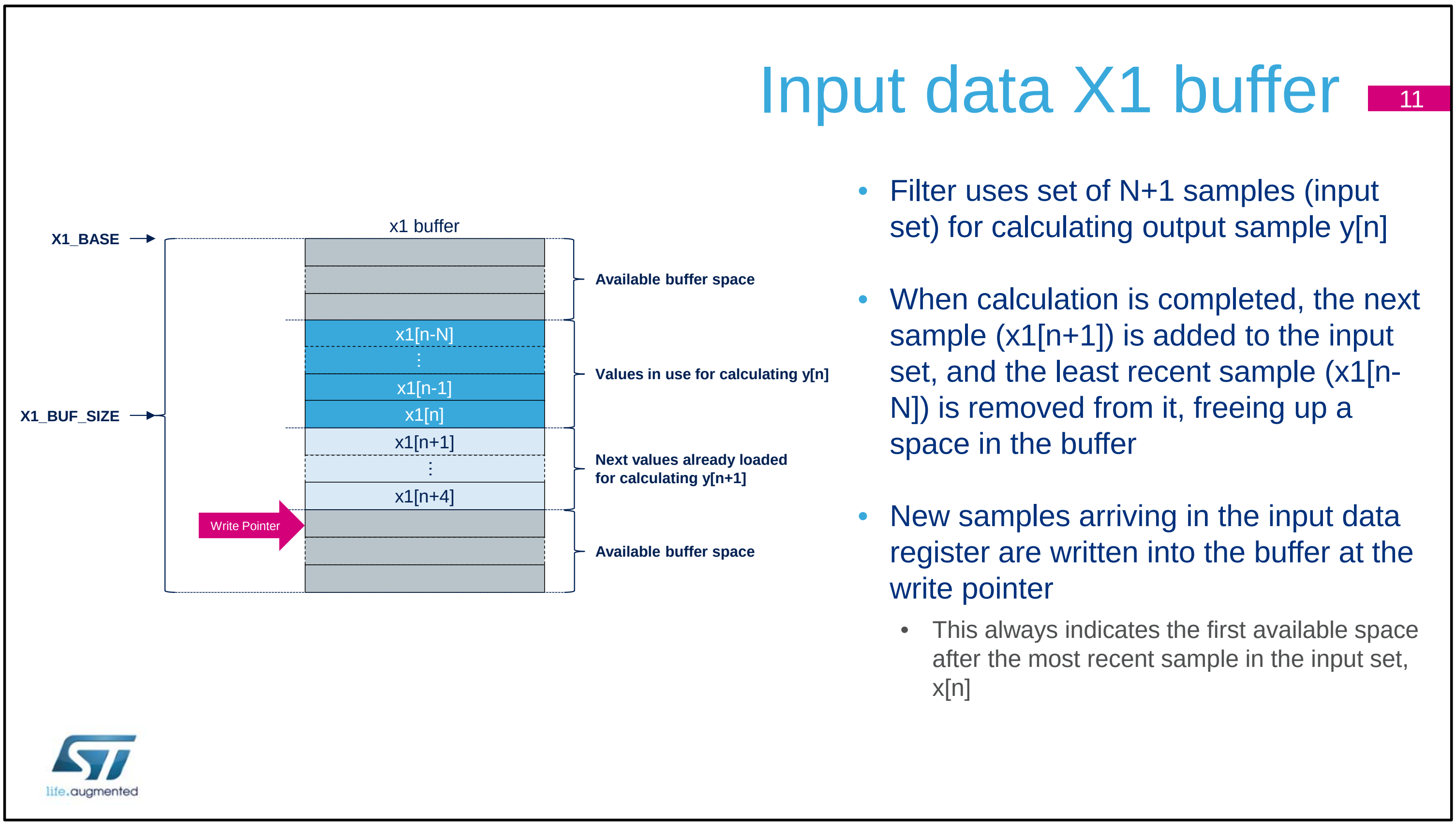
Input buffers

10

- The X1 and X2 buffers are used to store data for input to the MAC
  - A pointer in the control unit generates the read address offset (relative to the buffer base address) for each value
  - The pointers are managed by hardware according to the current function

life.augmented

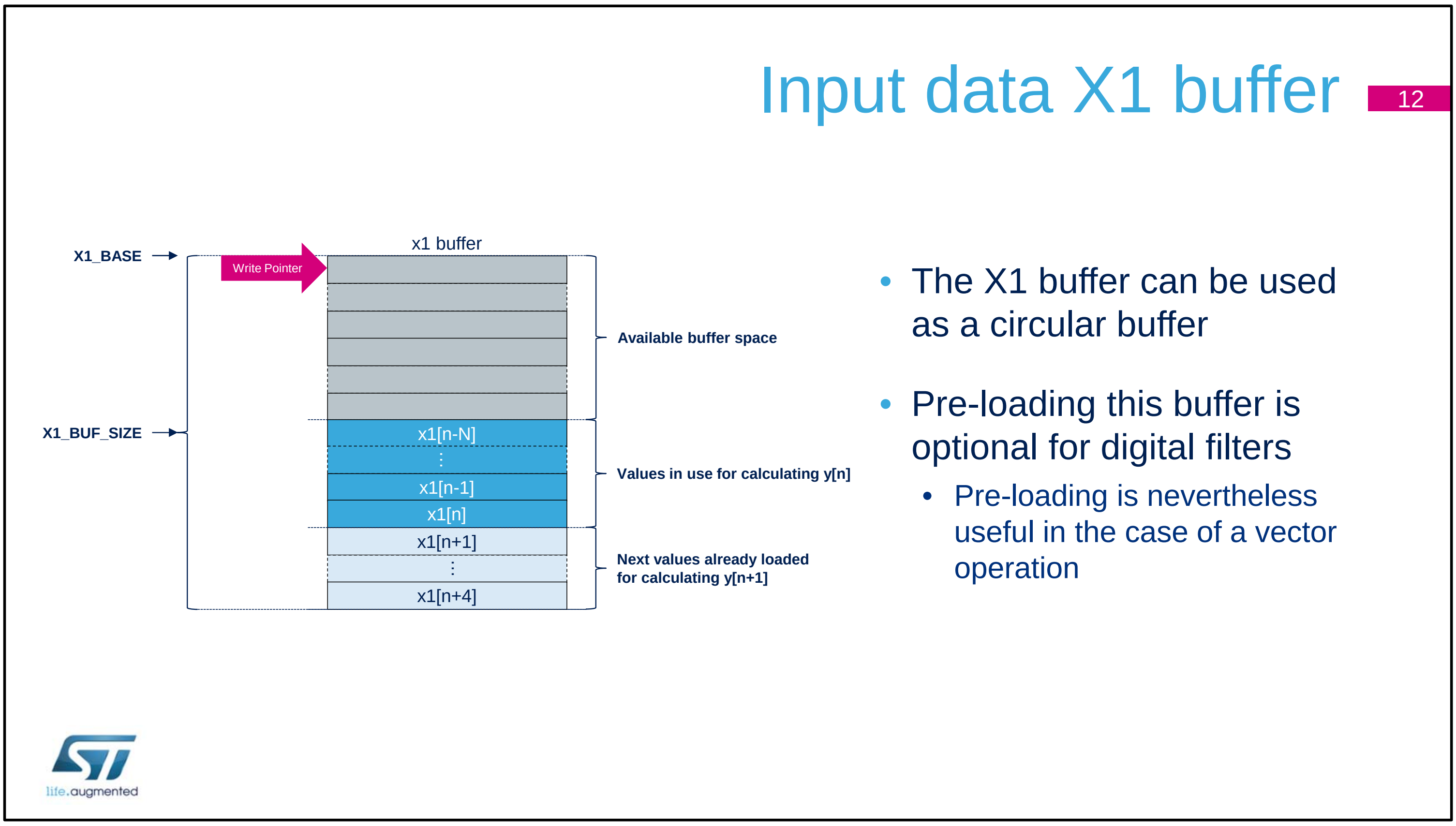
Each multiplication takes a value from the X1 buffer and a value from the X2 buffer and multiplies them together.
A pointer in the control unit generates the read address offset (relative to the buffer base address) for each value.
The pointers are managed by hardware according to the current function.



Input data X1 buffer

- Filter uses set of N+1 samples (input set) for calculating output sample y[n]
- When calculation is completed, the next sample (x1[n+1]) is added to the input set, and the least recent sample (x1[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]

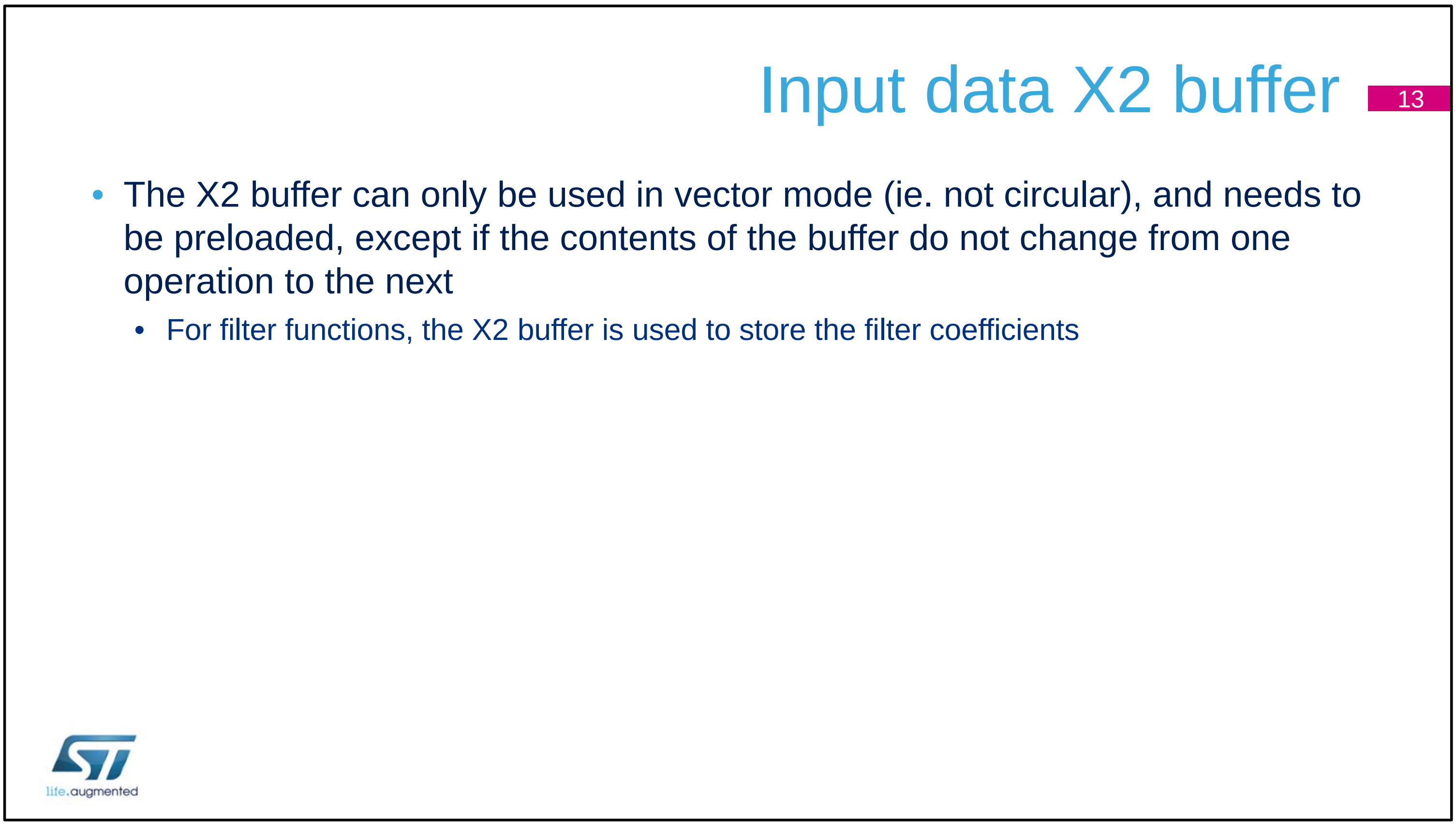
This figure explains the X1 buffer operation.
When the write pointer reaches the end of the buffer it wraps back to the beginning.
If available space in the buffer is less than the transfer size, the input buffer full flag is activated.
If the top of the input set, x[n] equals the write pointer (ie. no new sample available), the filter stalls until a new sample is available.
The processor, or DMA controller, must ensure that the new sample x[n+1] is available in the buffer space when required.
If not, the buffer is flagged as empty, which stalls the execution of the unit until a new sample is added.
No underflow condition is signaled on the X1 buffer.



Input data X1 buffer

- The X1 buffer can be used as a circular buffer
- Pre-loading this buffer is optional for digital filters
  - Pre-loading is nevertheless useful in the case of a vector operation

The X1 buffer can be used as a circular buffer.
New data are continually transferred into the input buffer whenever space is available.
The write pointer automatically wraps around when it reaches the last 16-bit entry in the buffer, as shown in the figure.
Pre-loading this buffer is optional for digital filters since if no input samples have been written in the buffer when the operation is started, it is flagged as empty, which triggers the CPU or DMA to load new
samples until there are enough to begin operation.
Pre-loading is nevertheless useful in the case of a vector operation, that is, the input data is already available in system memory and circular operation is not required.



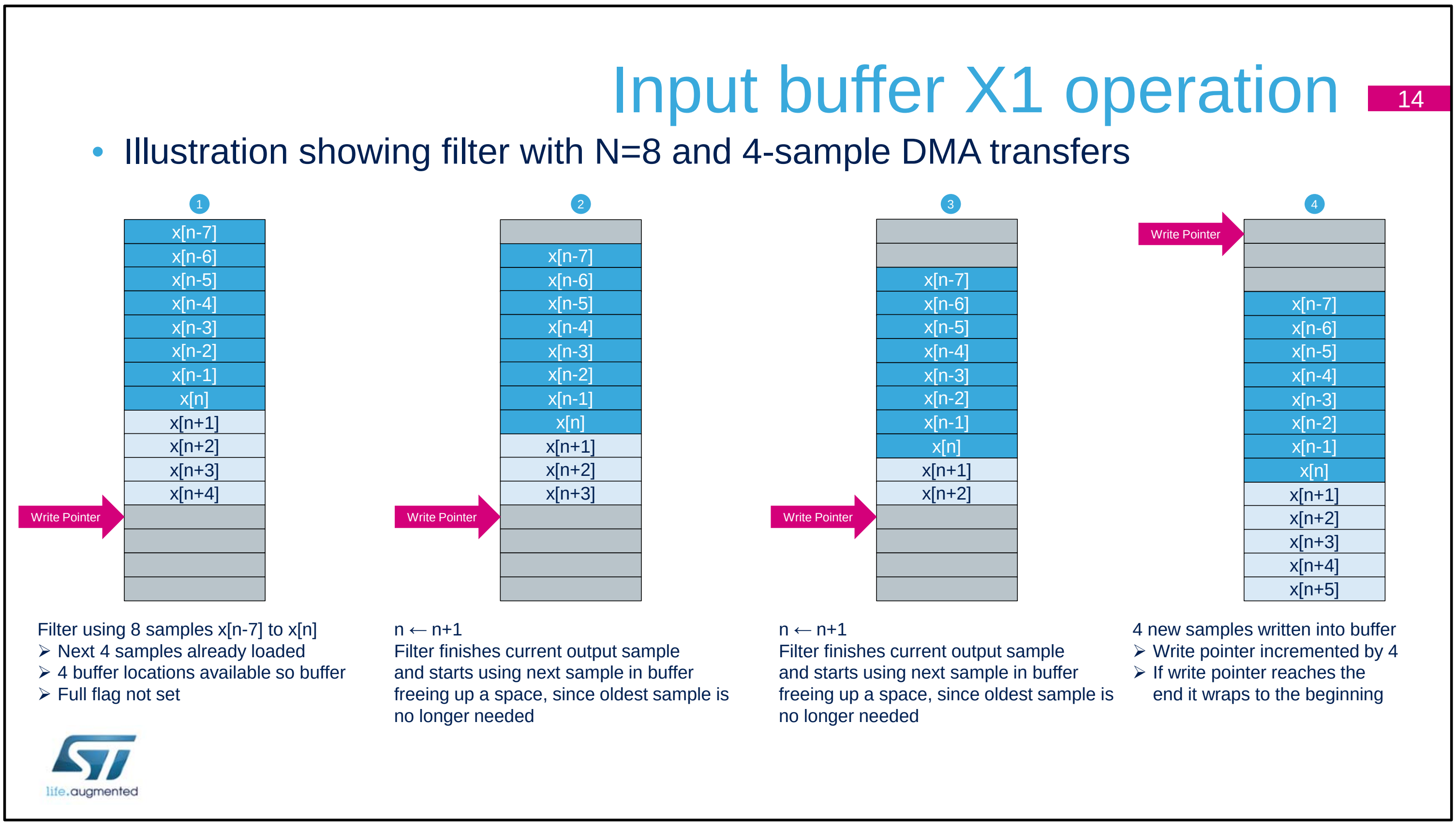
Input data X2 buffer

13

- The X2 buffer can only be used in vector mode (ie. not circular), and needs to be preloaded, except if the contents of the buffer do not change from one operation to the next
  - For filter functions, the X2 buffer is used to store the filter coefficients

life.augmented

The X2 buffer is used to store coefficients. It is usually loaded once during the initialization of the FMAC. Consequently, it does not support the circular addressing mode.



Input buffer X1 operation

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

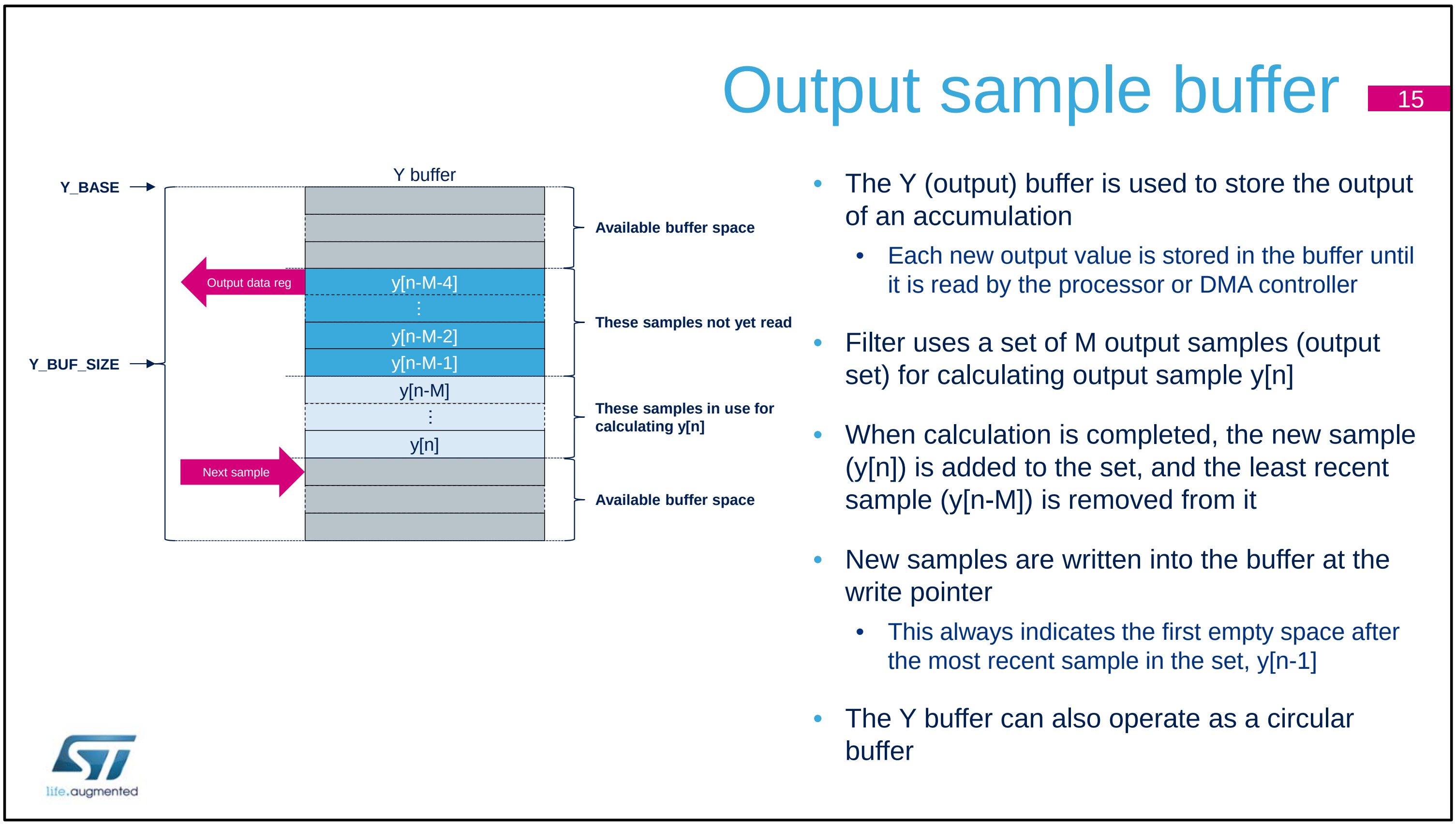
Filter using 8 samples x[n-7] to x[n]
- ➢ Next 4 samples already loaded
- ➢ 4 buffer locations available so buffer
- ➢ Full flag not set

n ← n+1
Filter finishes current output sample and starts using next sample in buffer freeing up a space, since oldest sample is no longer needed

n ← n+1
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

This figure summarizes the operation of the input buffers.
During step 1, the filter calculates y[n] from x[n-7] to x[n] and loads the next four samples.
During step 2, y[n] is now calculated. The sample x[n-7] is removed. Then n is incremented. The filter calculates y[n] from x[n-7] to x[n]. No new sample is loaded.
During step 3, y[n] is now calculated. The sample x[n-7] is removed. Then n is incremented. The filter calculates y[n] from x[n-7] to x[n]. No new sample is loaded.
During step 4, y[n] is now calculated. The sample x[n-7] is removed. Then n is incremented. The filter calculates y[n] from x[n-7] to x[n]. Four new samples are loaded. Since the upper address of the buffer has been reached, a wrap around to the beginning of the buffer occurs.

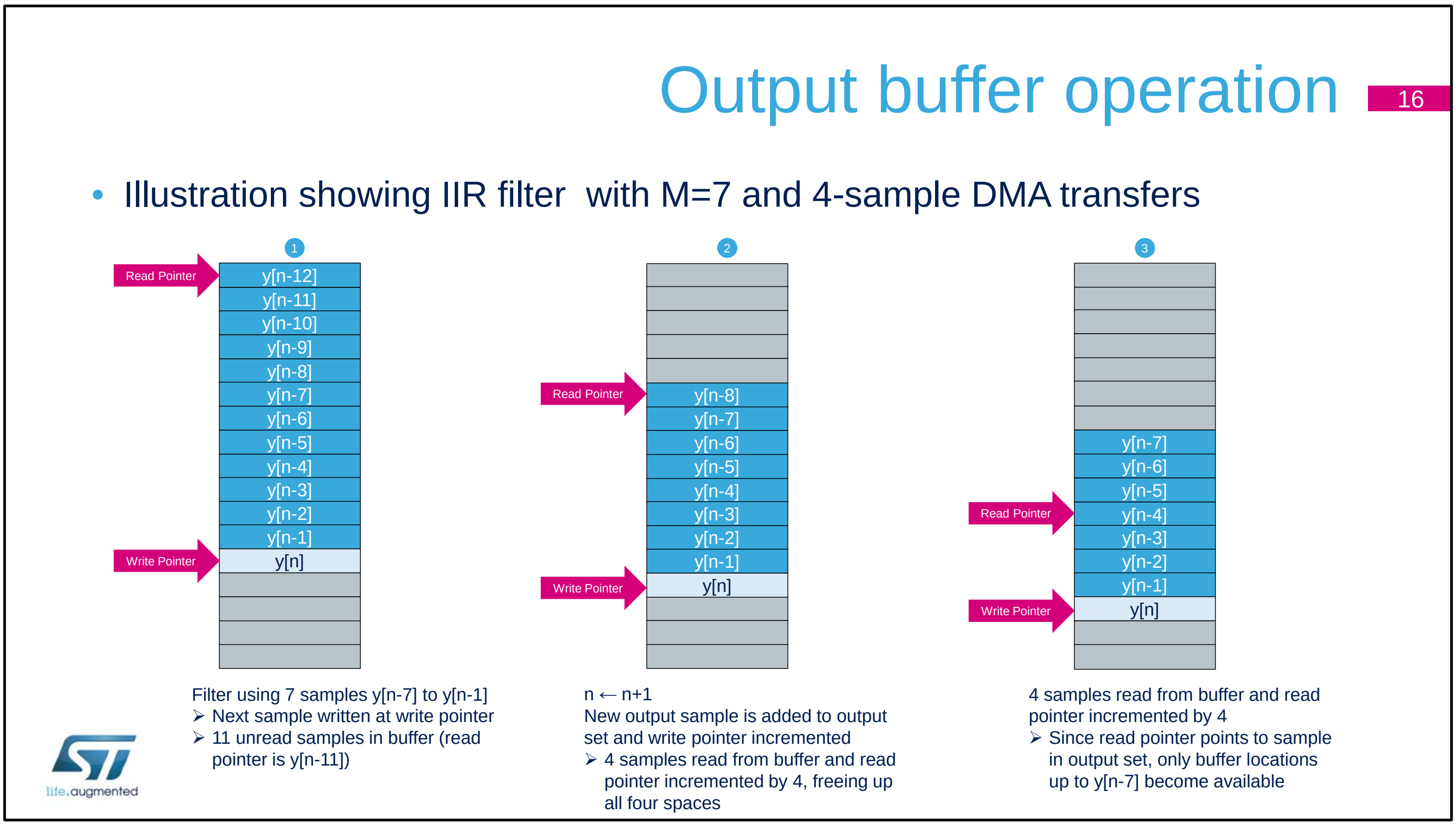


Output sample buffer

15

- The Y (output) buffer is used to store the output of an accumulation
  - Each new output value is stored in the buffer until it is read by the processor or DMA controller
- 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]
- The Y buffer can also operate as a circular buffer

This figure explains the Y buffer operation.
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

16

- 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])

n ← n+1
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

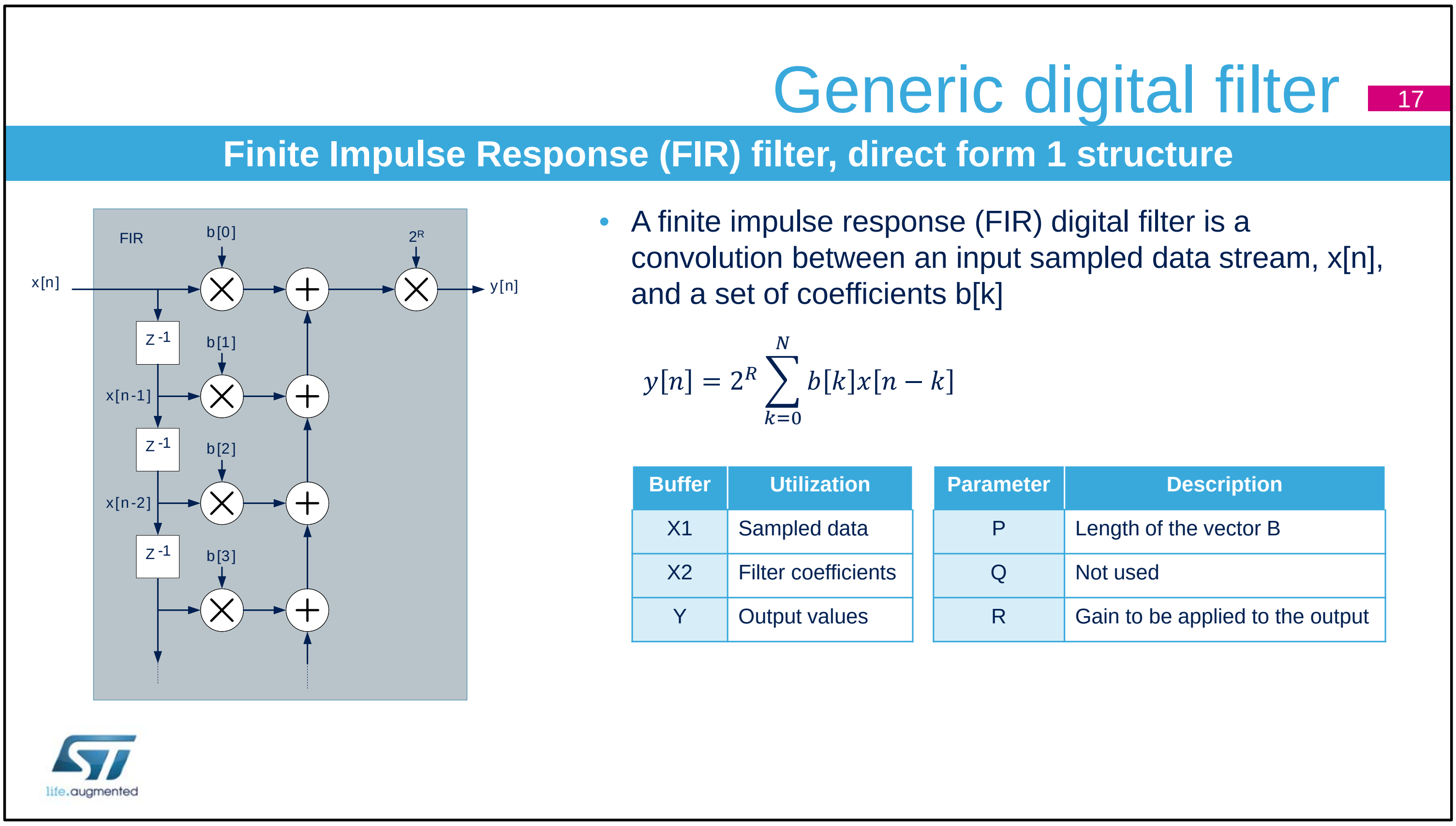
4 samples read from buffer and read pointer incremented by 4
- ➢ Since read pointer points to sample in output set, only buffer locations up to y[n-7] become available

This figure summarizes the operation of the output buffer.

During step 1, the filter calculates y[n] from y[n-7] to y[n-1]. Eleven samples are unread.

During step 2, n is incremented, the filter calculates y[n] from y[n-7] to y[n-1]. Software or DMA reads four samples, which shifts the Read Pointer to the oldest sample.

During step 3, n is incremented, the filter calculates y[n] from y[n-7] to y[n-1]. Software or DMA again reads four samples, which shifts the Read Pointer to the oldest sample. However samples y[n-7] to y[n-5] are not deallocated, because they are used in the current calculation.



Generic digital filter

Finite Impulse Response (FIR) filter, direct form 1 structure

- 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]

$$y[n] = 2^R \sum_{k=0}^{N} b[k]x[n-k]$$

Buffer	Utilization
X1	Sampled data
X2	Filter coefficients
Y	Output values

Parameter	Description
P	Length of the vector B
Q	Not used
R	Gain to be applied to the output

The FIR function performs a convolution of a vector B of length N+1 containing the filter coefficients and a vector X of indefinite length containing the sampled data.

To implement the FIR in the FMAC, the buffers are used as follows:

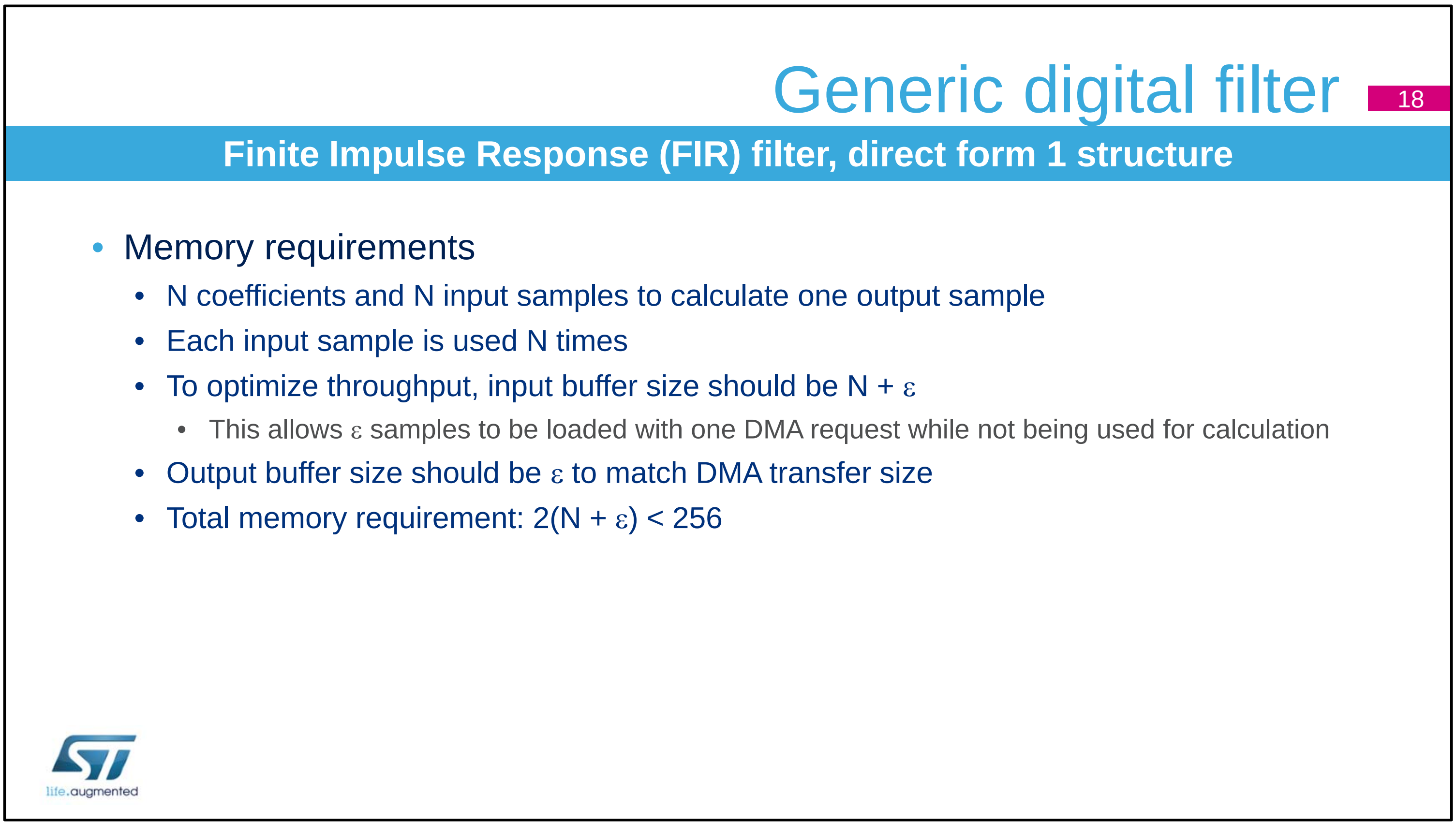
- X1 buffer contains the elements of vector X. It is a circular buffer of length N + 1 + d.
- X2 buffer contains the elements of vector B. It is a fixed buffer of length N + 1.
- Y buffer contains the output values, yn. It is a circular buffer of length d.

Here are the parameters:

- The parameter P contains the length, N+1, of the coefficient vector B in the range [2:127].
- The parameter R contains the gain to be applied to the accumulator output. The value output to the Y buffer is multiplied by 2^R, where R is in the range [0:7]

- The parameter Q is not used.

The function completes when the START bit in the FMAC_PARAM register is reset by software.

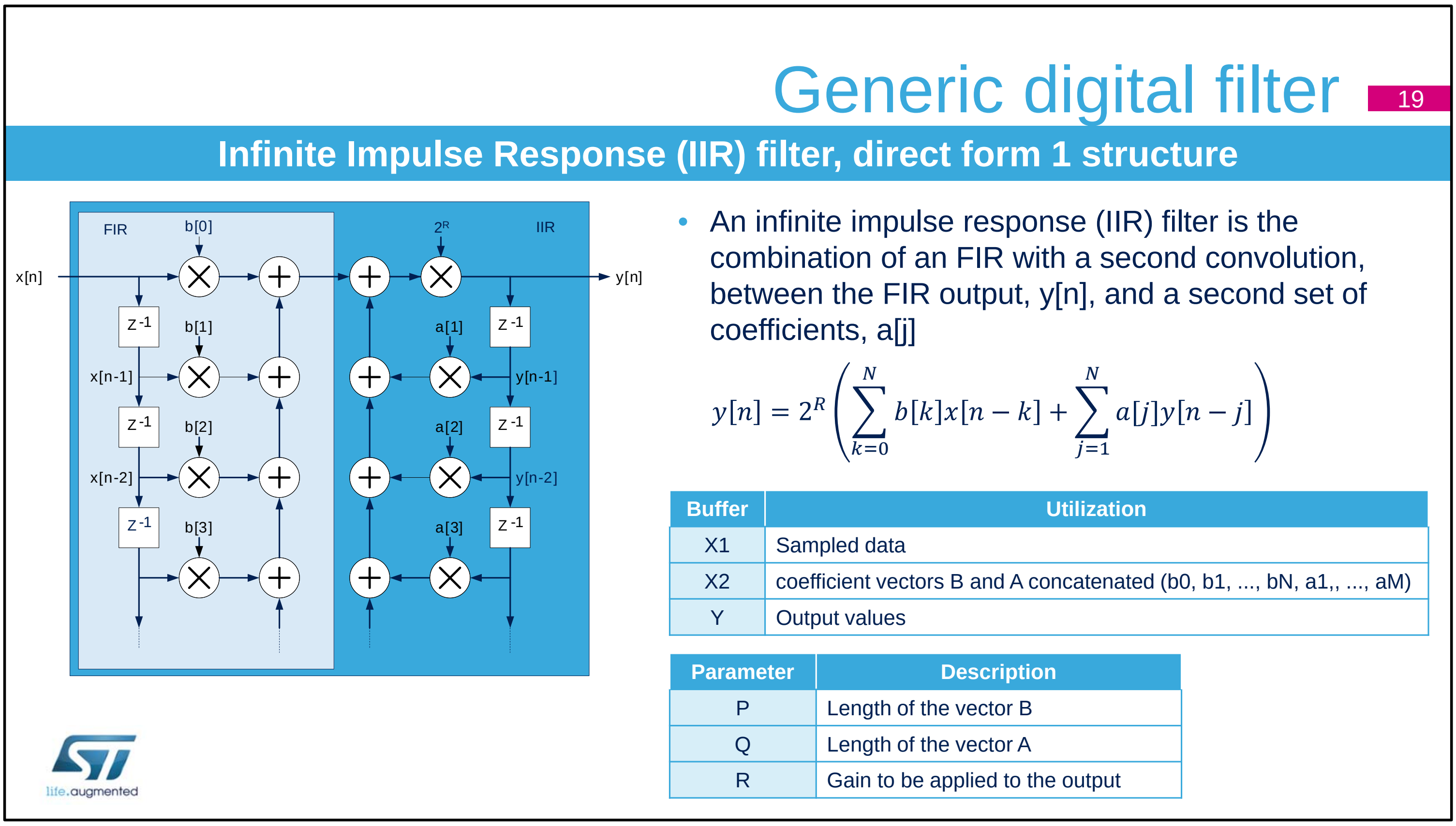


Generic digital filter

Finite Impulse Response (FIR) filter, direct form 1 structure

- Memory requirements
  - 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: $2(N + \varepsilon) < 256$

The FIR requires N coefficients and N input samples to calculate one output sample.
To optimize throughput, the input buffer size should be larger than N in order to load the next samples while the filter is working on the current set.
For example, when using 4-beat DMA transfers, epsilon should be set to 4.
Also the size of the output buffer should be set to epsilon to transfer resulting samples in a unique AHB burst transaction.



Generic digital filter

19

Infinite Impulse Response (IIR) filter, direct form 1 structure

- 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]

$$y[n] = 2^R\left(\sum_{k=0}^{N} b[k]x[n-k] + \sum_{j=1}^{N} a[j]y[n-j]\right)$$

Buffer	Utilization
X1	Sampled data
X2	coefficient vectors B and A concatenated (b0, b1, ..., bN, a1,, ..., aM)
Y	Output values

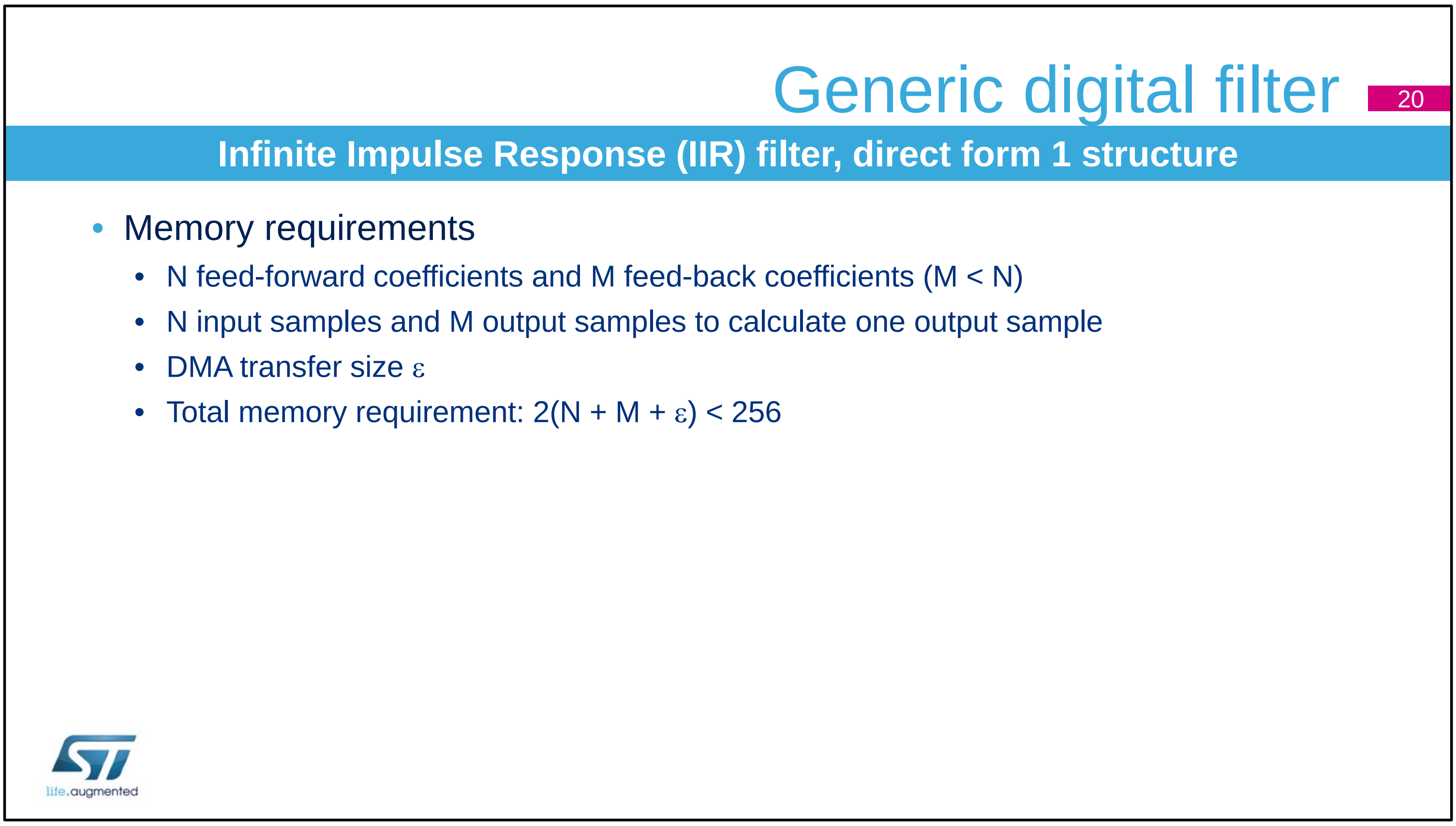
Parameter	Description
P	Length of the vector B
Q	Length of the vector A
R	Gain to be applied to the output

life.augmented

The IIR filter output vector Y is the convolution of a coefficient vector B of length N+1 and a vector X of indefinite length, plus the convolution of the delayed output vector Y’ with a second coefficient vector A, of length M.
To implement the IIR in the FMAC, the buffers are used as follows:
• X1 buffer contains the elements of vector X. It is a circular buffer of length N + 1+ d.
• X2 buffer contains the elements of coefficient vectors B and A concatenated (b0, b1, b2..., bN, a1, a2, ..., aM). It is a fixed buffer of length M+N+1.
• Y buffer contains the output values, yn. It is a circular buffer of length M + d.
Here are the parameters:
• The parameter P contains the length, N + 1, of the coefficient vector B in the range [2:64].

- The parameter Q contains the length, M, of the coefficient vector A in the range [1:63].
- The parameter R contains the gain to be applied to the accumulator output. The value output to the Y buffer is multiplied by 2^R, where R is in the range [0:7].

The function completes when the START bit in the FMAC_PARAM register is reset by software.



Generic digital filter

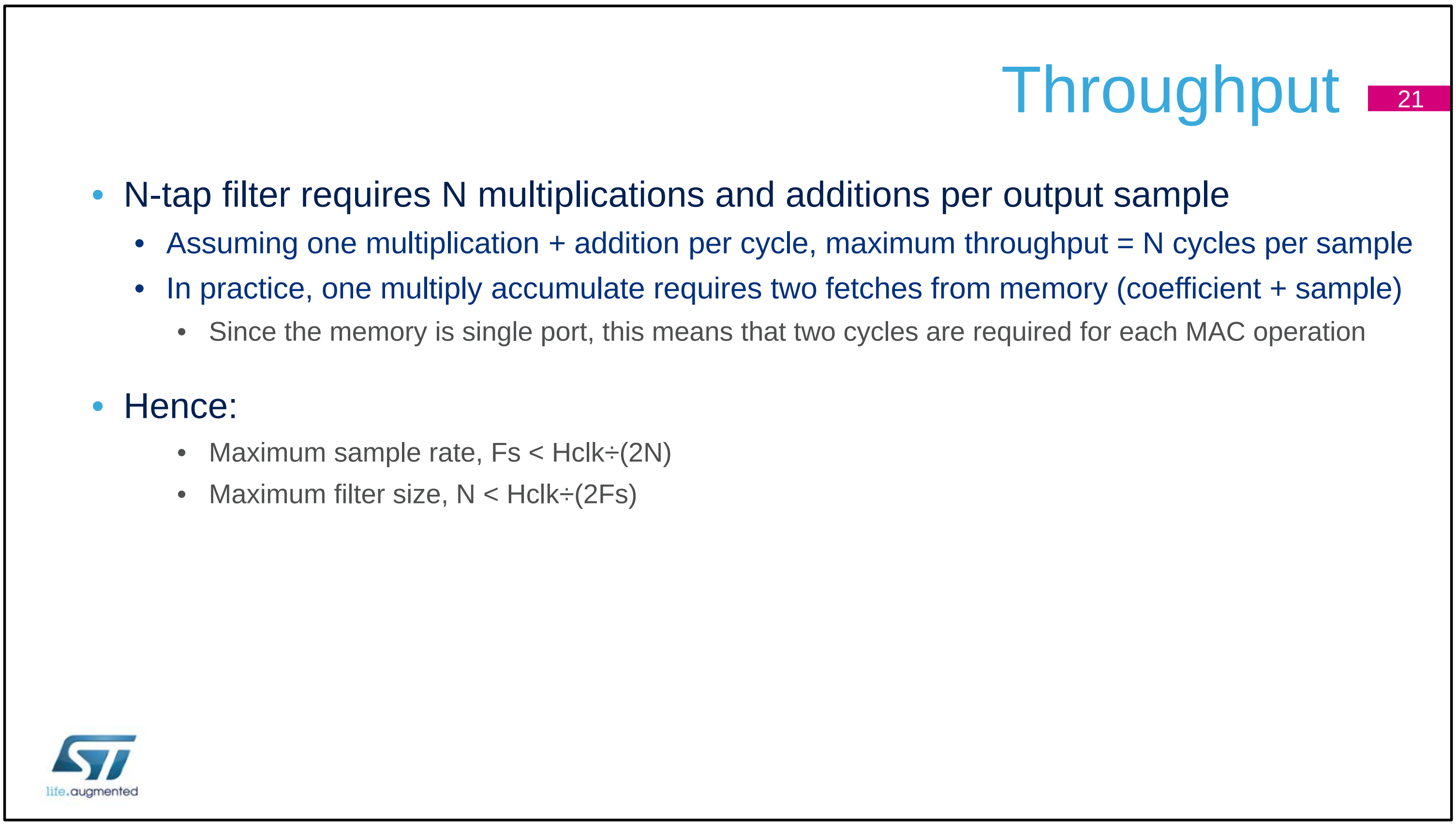
20

Infinite Impulse Response (IIR) filter, direct form 1 structure

- Memory requirements
  - 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: $2(N + M + \varepsilon) < 256$

life.augmented

The FIR requires N feed-forward coefficients and M feed-back coefficients, M being lower than N.
The input buffer size should be N+epsilon, epsilon being the number of data in a DMA burst.
The output buffer size should be M+epsilon.



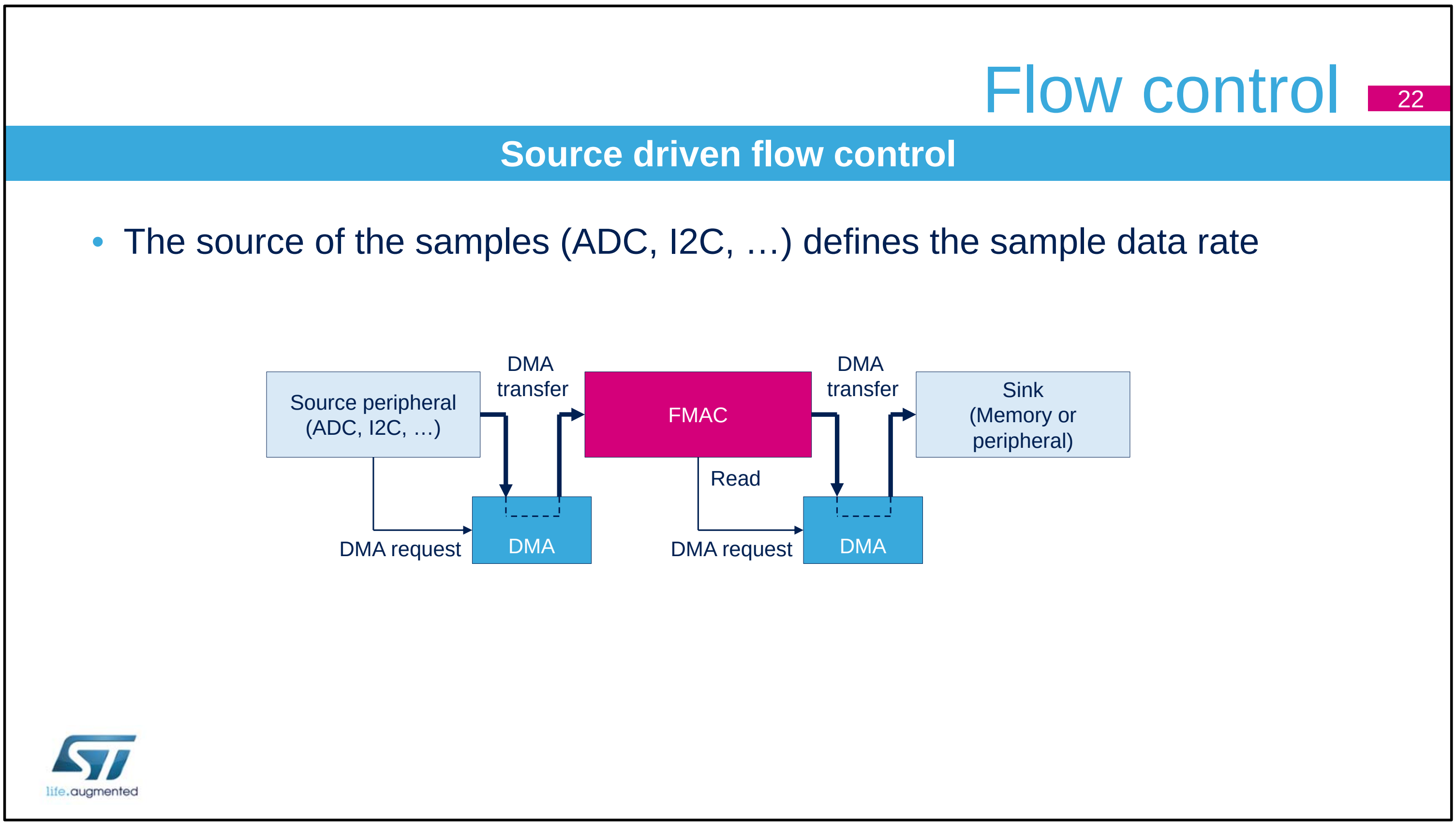
Throughput

21

- N-tap filter requires N multiplications and additions per output sample
  - Assuming one multiplication + addition per cycle, maximum throughput = N cycles per sample
  - In practice, one multiply accumulate requires two fetches from memory (coefficient + sample)
    - Since the memory is single port, this means that two cycles are required for each MAC operation
- Hence:
    - Maximum sample rate, Fs < Hclk÷(2N)
    - Maximum filter size, N < Hclk÷(2Fs)

The clock reference of the FMAC module is the AHB clock Hclk. Hclk maximum frequency is half of CPU max frequency.
N-tap filter such as FIR requires N multiplications and additions per output sample, knowing that each MAC requires two memory reads, thus 2 clock cycles.
As a consequence, the maximum sample rate is Hclk/(2*N).
N must be lower than Hclk frequency divided by 2 times the maximum sample rate frequency.
Assuming the Hclk frequency of the STM32H7 is 275MHz, we obtain:

- Maximum filter size at Fs = 2Msps is N < 68 taps
- Maximum sample rate for N = 127 taps is Fs < 1.08MHz.



Flow control

22

Source driven flow control

- The source of the samples (ADC, I2C, …) defines the sample data rate

Source peripheral (ADC, I2C, …)

DMA transfer

FMAC

DMA transfer

Sink (Memory or peripheral)

Read

DMA request

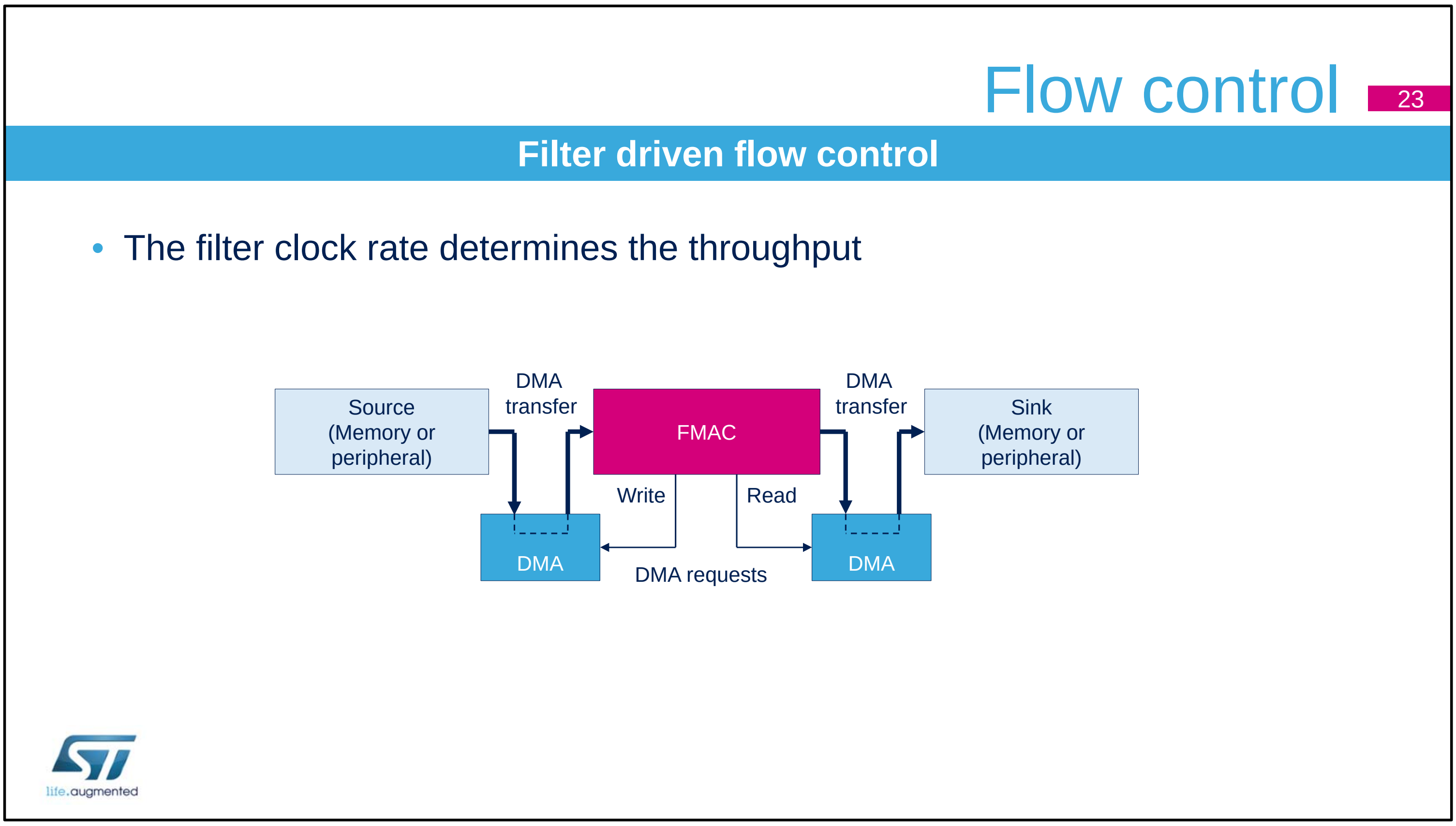
DMA

DMA request

DMA

life.augmented

Flow control can be source, sink or filter driven.
This slide describes the source driven flow control sequence.
The source of the samples (ADC, I2C, …) defines the sample data rate.
The source requests the DMA (or CPU) to transfer data to the filter input buffer.
The filter operates at a faster clock rate than 2N times the source sample rate.
When the input buffer is empty (next sample not available), the filter stalls, waiting for new data.
When the output buffer is not empty (one or more samples available) an output channel DMA request (or interrupt) is generated.
The DMA (or CPU) transfers the output samples to memory or another peripheral, such as DAC or PWM.



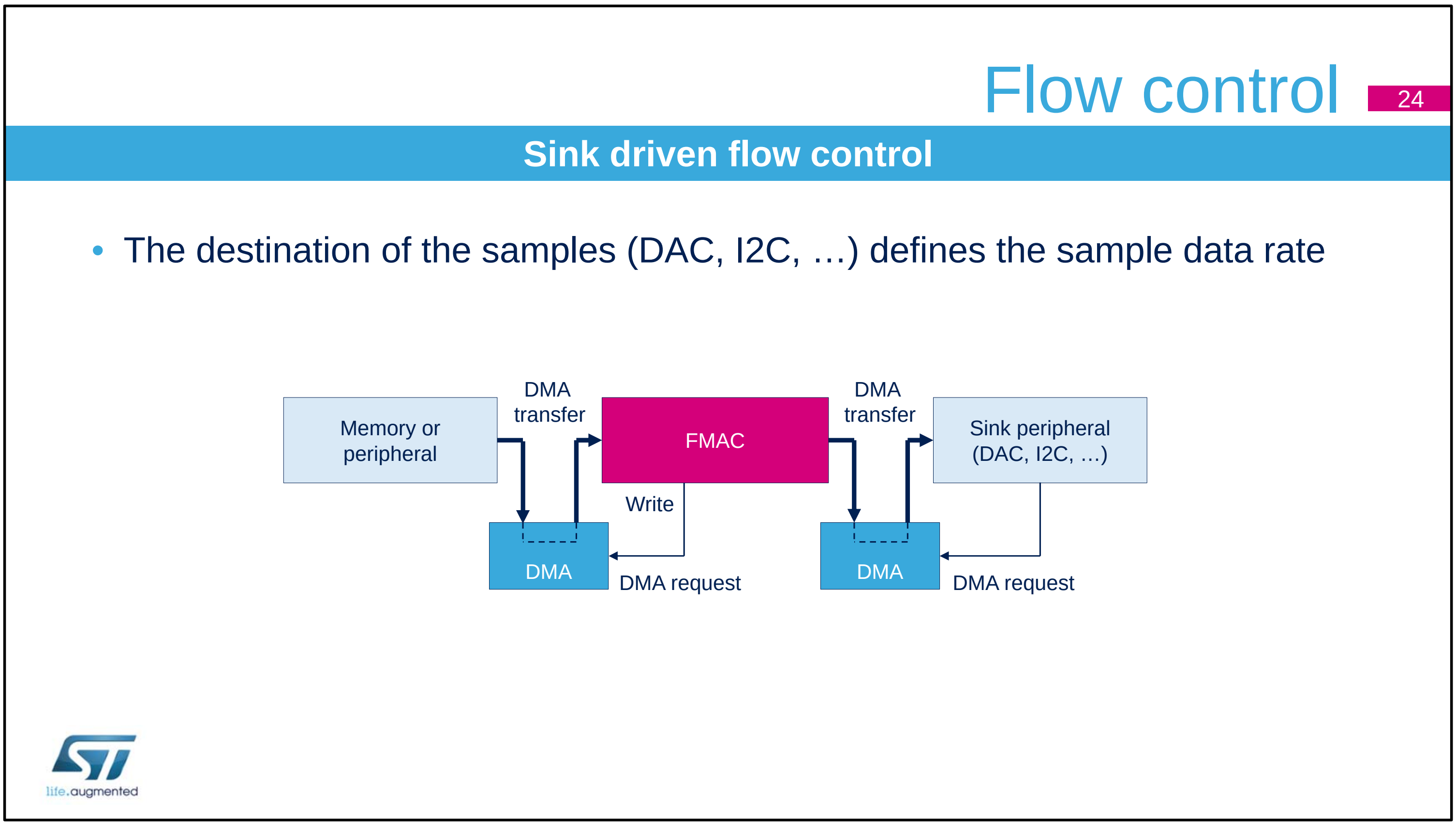
Flow control

23

Filter driven flow control

- The filter clock rate determines the throughput

This slide describes the filter driven flow control sequence.
The filter clock rate determines the throughput.
An input channel DMA request (or interrupt) is generated whenever the input buffer is not full.
The DMA (or CPU) transfers data into the input buffer from memory or another peripheral.
As long as data is available in the input buffer, the filter generates new output samples.
When the output buffer is not empty an output channel DMA request (or interrupt) is generated.
The DMA (or CPU) transfers data from the output buffer to memory or another peripheral.

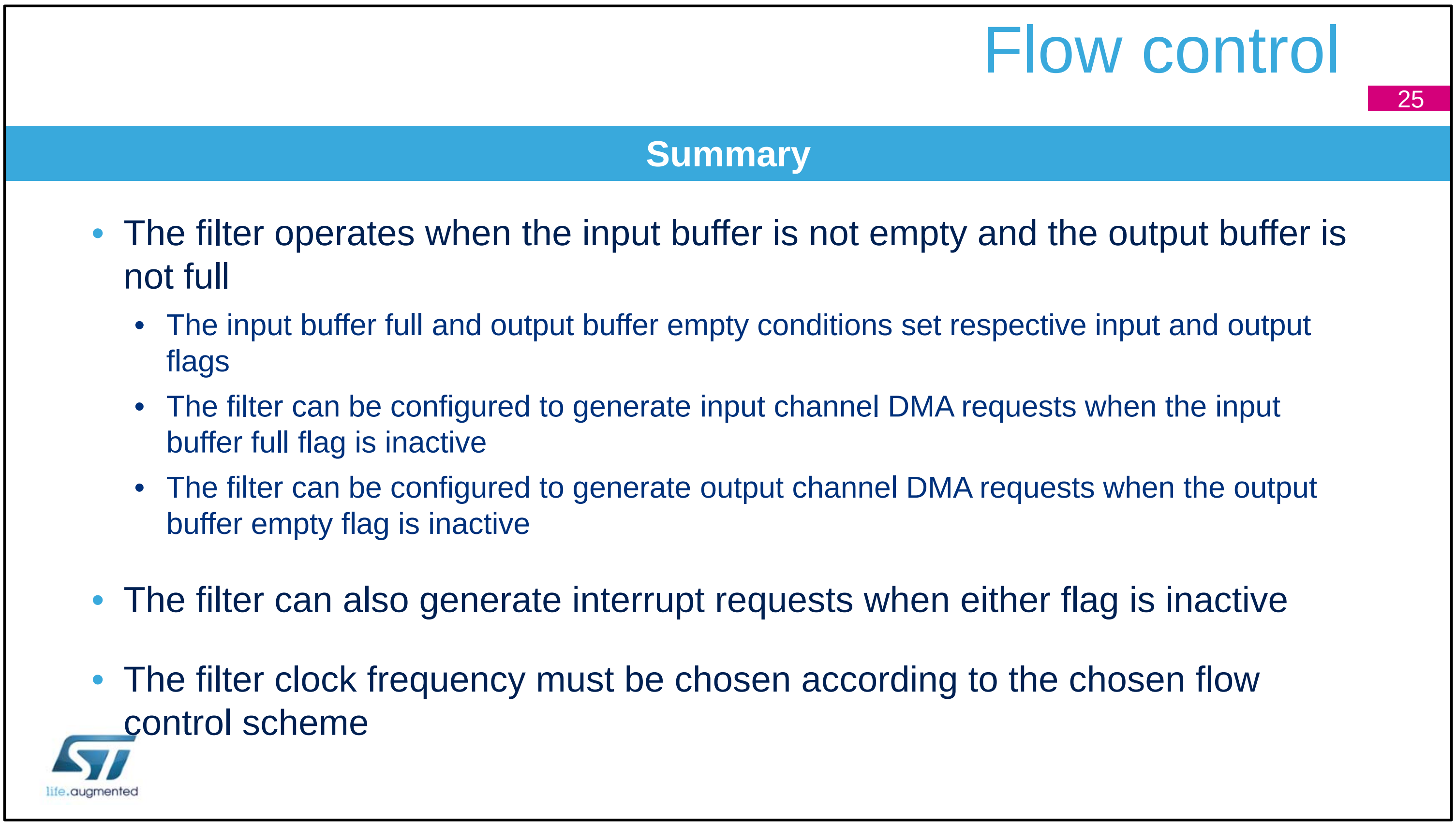


Flow control

Sink driven flow control

- The destination of the samples (DAC, I2C, …) defines the sample data rate

This slide describes the sink driven flow control sequence.
The destination of the samples (DAC, I2C, …) defines the sample data rate.
The destination requests the DMA (or CPU) to transfer data from the filter output.
The filter operates at a faster clock rate than 2N times the destination sample rate.
When the output buffer is full the filter stalls.
When the input buffer is not full, an input channel DMA request (or interrupt) is generated.
The DMA (or CPU) transfers samples from memory, or another peripheral, to the input buffer.



Flow control

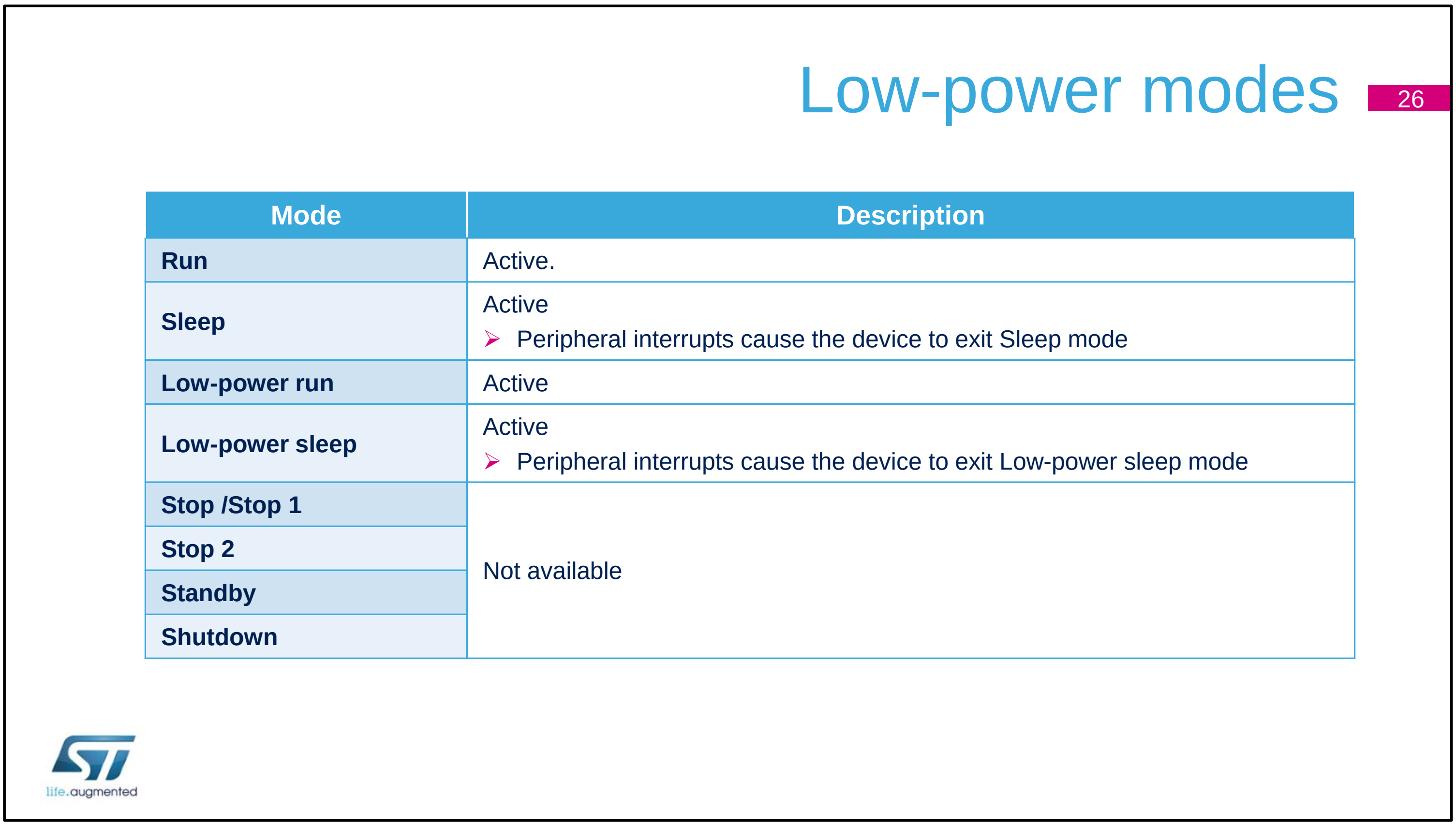
25

Summary

- The filter operates when the input buffer is not empty and the output buffer is not full
  - The input buffer full and output buffer empty conditions set respective input and output flags
  - The filter can be configured to generate input channel DMA requests when the input buffer full flag is inactive
  - The filter can be configured to generate output channel DMA requests when the output buffer empty flag is inactive
- The filter can also generate interrupt requests when either flag is inactive
- The filter clock frequency must be chosen according to the chosen flow control scheme

life.augmented

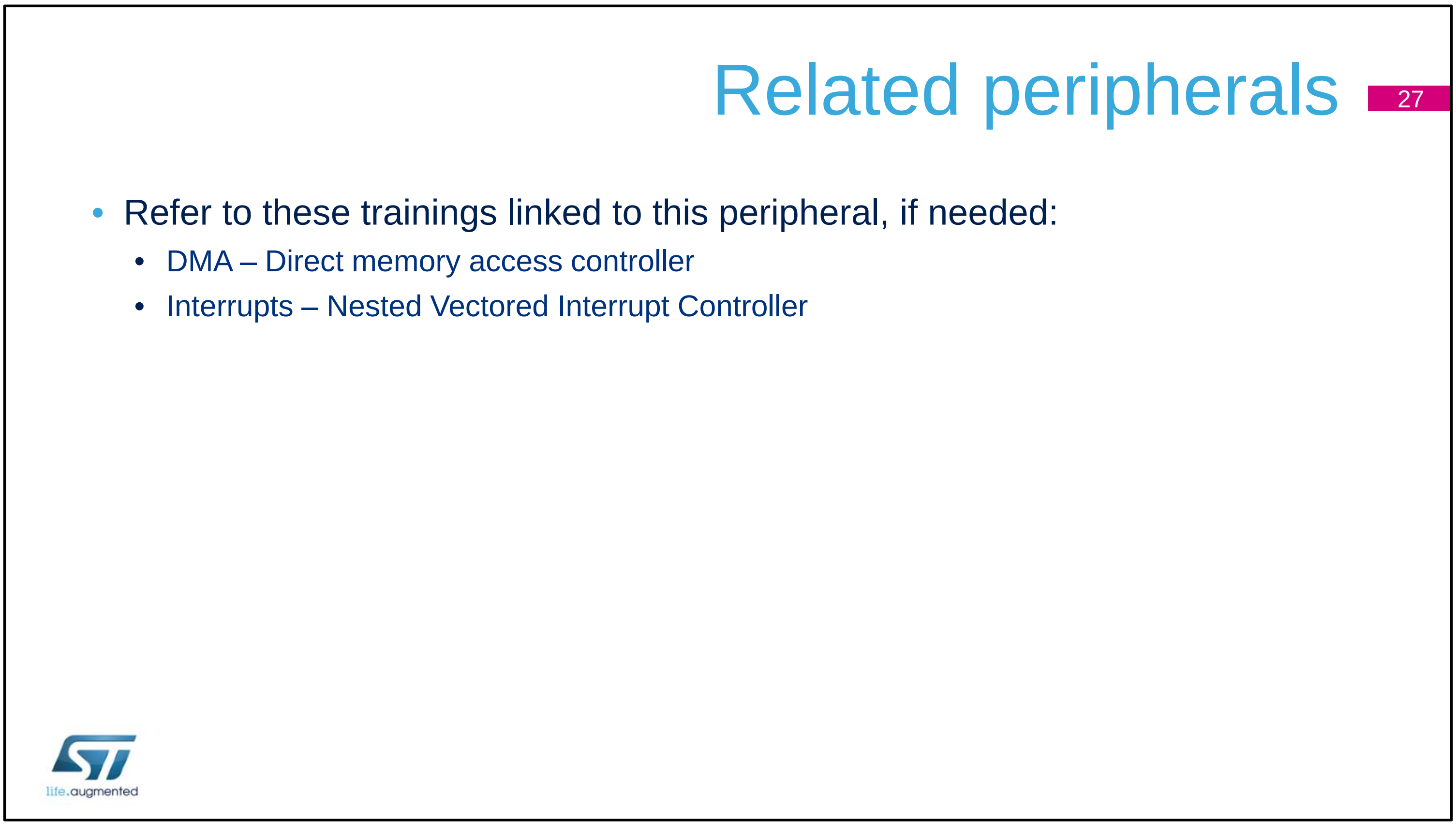
The FMAC executes the filter algorithm when the input buffer is not empty and the output buffer is not full.
A flag called X1FULL is set if the number of available spaces is X1 buffer is less than the FULL_WM threshold.
A DMA request can be generated when this flag is not set in order to fill the X1 buffer.
A flag called YEMPTY is set if the number of unread data is less than the EMPTY_WM threshold.
A DMA request can be generated when this flags is not set in order to empty the Y buffer.
The management of buffers can also be performed by software relying on interrupt requests that can be asserted when either flag inactive.
The filter clock frequency must be chosen according to the chosen flow control scheme.



Low-power modes

Mode	Description
Run	Active.
Sleep	Active ➢ Peripheral interrupts cause the device to exit Sleep mode
Low-power run	Active
Low-power sleep	Active ➢ Peripheral interrupts cause the device to exit Low-power sleep mode
Stop /Stop 1	Not available
Stop 2	
Standby	
Shutdown	

The FMAC unit is active in Run, Low-power run, Sleep and Low-power Sleep modes.
It is not available in the other low power modes.



Related peripherals

- Refer to these trainings linked to this peripheral, if needed:
  - DMA – Direct memory access controller
  - Interrupts – Nested Vectored Interrupt Controller

These peripherals may need to be specifically configured for correct use with the FMAC block.
Please refer to the corresponding peripheral training modules for more information.