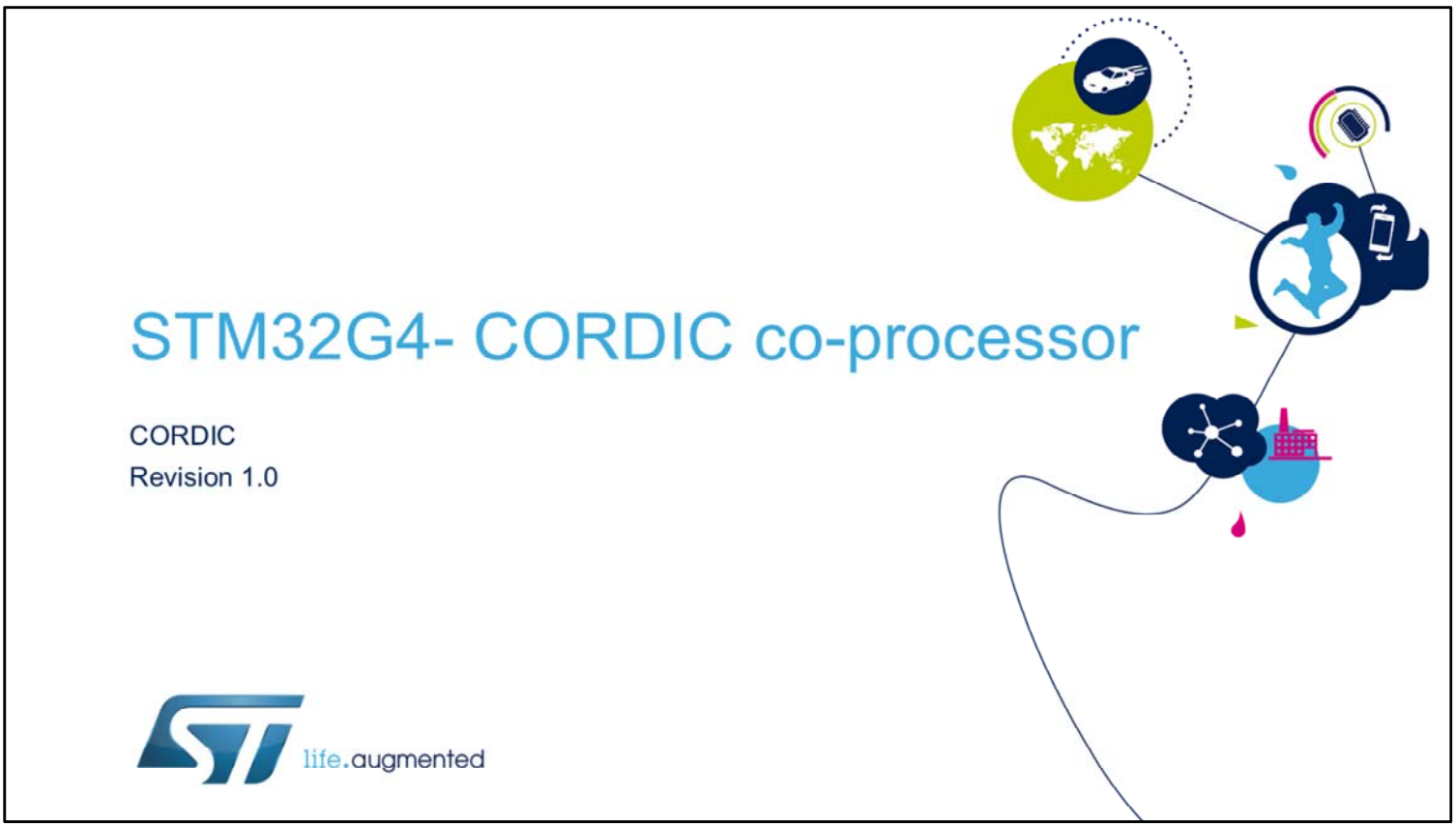
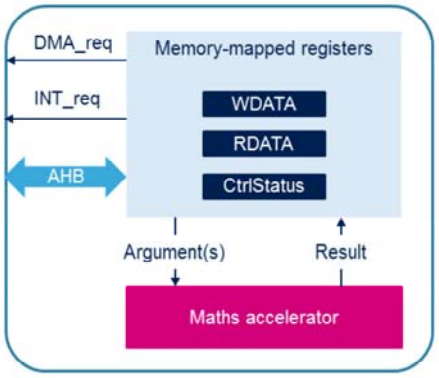


Hello and welcome to this presentation of the STM32G4 Cordic co-processor block.
It will cover the main features of this block, which is used to accelerate trigonometric functions.

Overview



- Cordic (COordinate Rotation DIgital Computer) provides hardware acceleration of mathematical functions
  - Trigonometric, logarithm, square root
- Transparent synchronization with the software
  - Wait states are inserted in the AHB slave interface until result is available
  - Interrupt and DMA channel can also be used

Application benefits

- Fixed point
- Pipelined operation
- Offloads the CPU, thus reducing the consumption and improving the performance

life.augmented

The CORDIC co-processor provides hardware acceleration of certain mathematical functions, notably trigonometric, commonly used in motor control, metering, signal processing and many other applications.
It speeds up the calculation of these functions compared to a software implementation, allowing a lower operating frequency, or freeing up processor cycles in order to perform other tasks.
The Cordic block is an AHB slave that inserts wait state when the Cortex-M4 requests the result until the operation is completed. No input/output driver is therefore needed.
Another approach consists in enabling the Cortex-M4 to handle other processing while the Cordic calculation is in progress. In this case an interrupt request indicates that the result is available.
DMA channels can be implemented to provide the

arguments from memory and to write the result to memory.
The Cordic block supports a pipelined operation: next arguments can be provided while the calculation with the current arguments is in progress.
Note that the Cordic block is a fixed-point arithmetic accelerator.

Cordic basics

3

- Cordic (COordinate Rotation DIgital Computer) is a low-cost successive approximation algorithm for evaluating trigonometric and hyperbolic functions
  - In trigonometric (circular) mode, the sine and cosine of an angle θ are determined by rotating the unit vector [1, 0] through decreasing angles until the cumulative sum of the rotation angles equals the input angle
  - Inversely, the angle of a vector [x, y], corresponding to arctangent (y/x), is determined by rotating [x, y] through successively decreasing angles to obtain the unit vector [1, 0]
- The CORDIC algorithm can also be used for calculating hyperbolic functions (sinh, cosh,atanh) by replacing the successive circular rotations by steps along a hyperbola

life.augmented

CORDIC, which means coordinate rotation digital computer, is a hardware-efficient iterative method which uses rotations to calculate a wide range of elementary functions.
In trigonometric (circular) mode, the sine and cosine of an angle θ are determined by rotating the unit vector [1, 0] through decreasing angles until the cumulative sum of the rotation angles equals the input angle.
The x and y Cartesian components of the rotated vector then correspond respectively to the cosine and sine of θ.
Inversely, the angle of a vector [x, y], corresponding to arctangent (y/x), is determined by rotating [x, y] through successively decreasing angles to obtain the unit vector [1, 0].
The cumulative sum of the rotation angles gives the angle of the original vector.
The CORDIC algorithm can also be used for calculating

hyperbolic functions (sinh, cosh,atanh) by replacing the successive circular rotations by steps along a hyperbola.

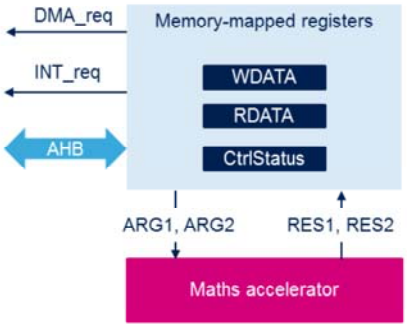
Cordic functions

- STM32G4 Cordic unit supports 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 ($\sqrt{x}$)
- Selection is done in the FUNC field of the CORDIC_CR register

life.augmented

This slide indicates the list of the 10 supported mathematical functions.
The first step when using the co-processor is to select the required function, by programming the FUNC field of the CORDIC_CSR register accordingly.
Consequently only one function is active at a time.

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)

life.augmented

Several functions take two input arguments, ARG1 and ARG2, and some generate two results simultaneously, RES1 and RES2.
This is a side-effect of the Cordic algorithm and means that only one operation is needed to obtain two values.
This is the case, for example, when performing polar-to-rectangular conversion: sin θ also generates cos θ, while cos θ also generates sin θ.
Similarly for rectangular-to-polar conversion (phase(x,y), modulus(x,y)) and for hyperbolic functions (cosh θ, sinh θ).

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

q1.31 format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
s	c_{30}	c_{29}	c_{28}	c_{27}	c_{26}	c_{25}	c_{24}	c_{23}	c_{22}	c_{21}	c_{20}	c_{19}	c_{18}	c_{17}	c_{16}	c_{15}	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

$$\text{Fractional\_number} = (-1)^s \sum_{k=0}^{30} \frac{1}{2^{31-k}} c_k$$

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

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

- Packing, unpacking q1.15 numbers

life.augmented

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 5×10^{-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 3×10^{-5}).

Numerical representation

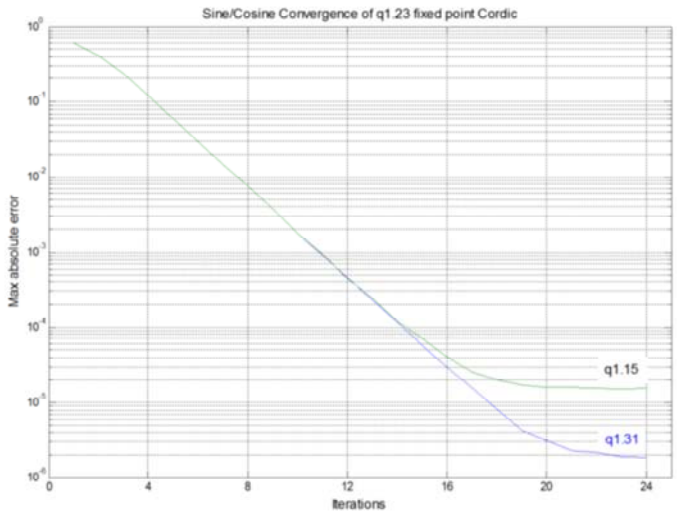
- Angle representation
  - Angles are divided by π in order to represent them efficiently in fixed point format
  - Hence -1 corresponds to the angle -π and +1 corresponds to +π
  - Increasing the angle past +π automatically causes it to wrap to -π
- Scaling factor
  - Some of the functions have a domain 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

life.augmented

Angles are expressed in radian, divided by π.
Consequently, only the interval [-1,+1] is used.
Several of the functions specify a scaling factor, SCALE. This allows the function input range to be extended to cover the full range of values supported by the Cordic, without saturating the input, output or internal registers.
If the scaling factor is required, it should be calculated in software and programmed into the SCALE field of the CORDIC_CSR register.
The input arguments should be scaled accordingly before programming the scaled values in the CORDIC_WDATA register.
The scaling should also be undone on the results read from the CORDIC_RDATA register.
Note that the scaling factor entails a loss of precision due to truncation of the scaled value.

Precision

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

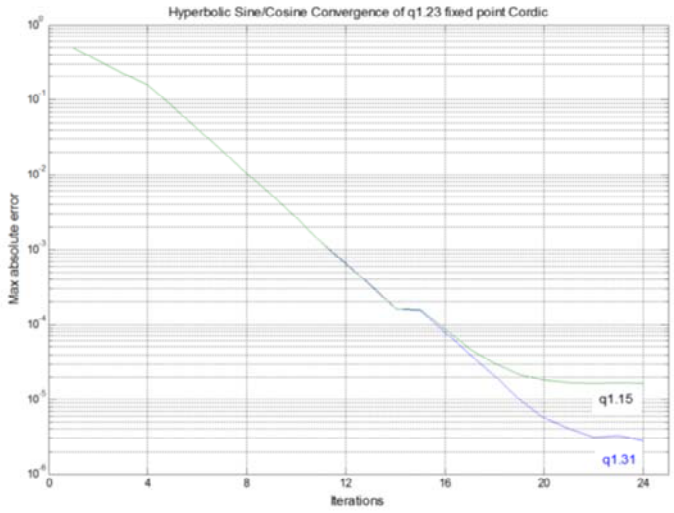


life.augmented

The precision of the result is dependent on the number of CORDIC iterations.
The algorithm converges at a constant rate of one binary digit per iteration for trigonometric functions

Precision

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

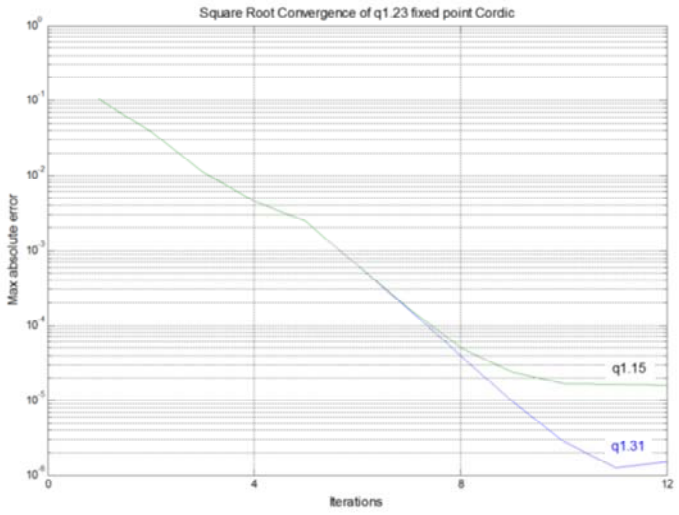


life.augmented

For hyperbolic functions (hyperbolic sine, hyperbolic cosine, natural logarithm), the convergence rate is less constant due to the peculiarities of the CORDIC algorithm

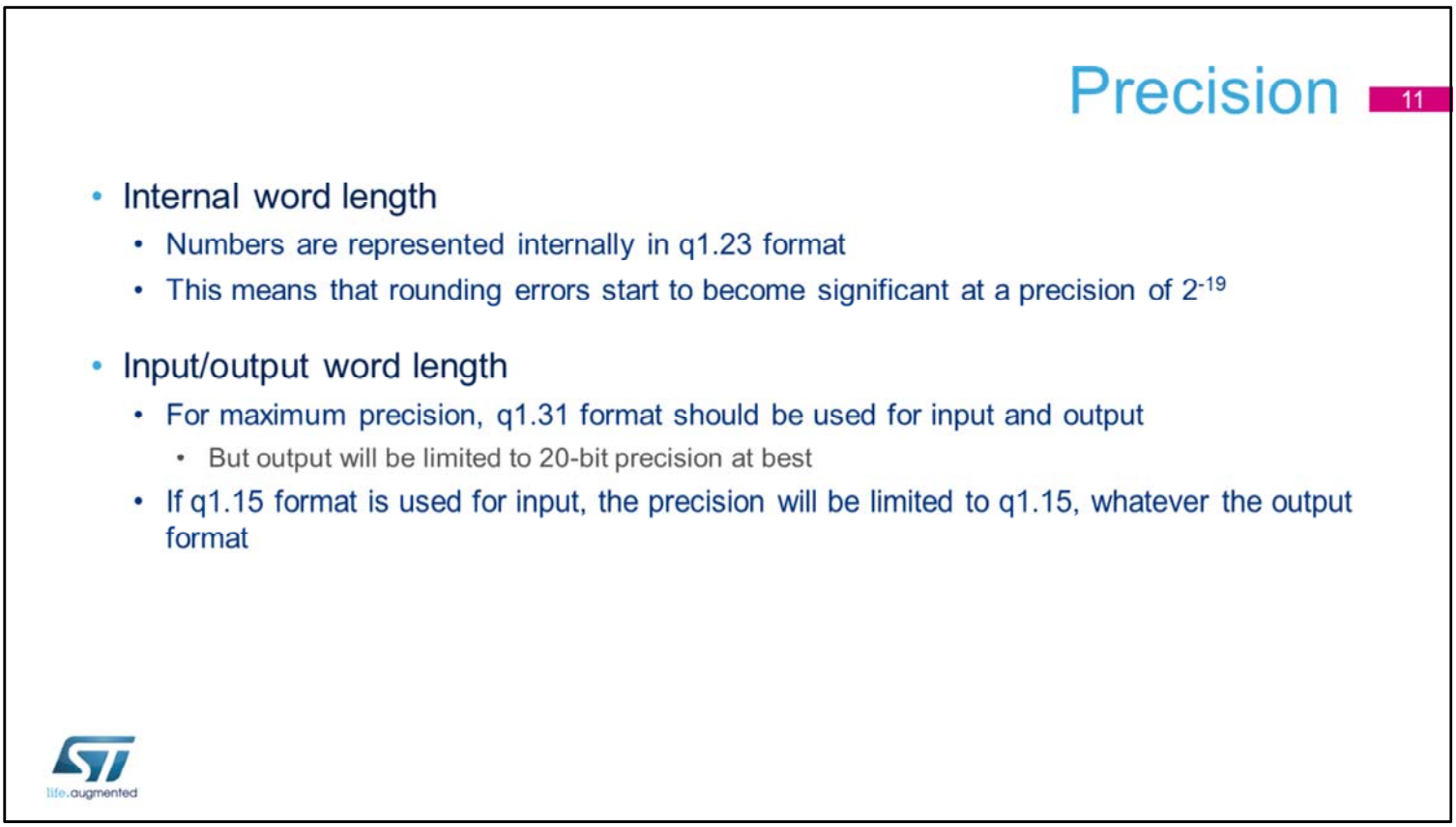
Precision

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



life.augmented

The square root function converges at roughly twice the speed of the hyperbolic functions.



The format of arguments and results is independently programmed in the fields ARGSIZE and RESSIZE of the CORDIC_CSR register: either q1.15 or q1.31.
Internally, the Cordic accelerator implements the 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.
For maximum precision, q1.31 format should be used for input and output.
However, given the format implemented internally, the output is 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.

Precision

- 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

life.augmented

The precision required depends on the number of iterations, which has to be programmed in the field PRECISION of the CORDIC_CSR register.
The number of iterations is equal to the value programmed in this field multiplied by four.
For maximum speed, the minimum number of iterations for the required precision should be programmed.
Note that for most functions, the recommended range for this field is 3 to 6.

Cosine function

Parameter	Description	Range
ARG1	Angle θ in radians, dvided by π	[-1, 1]
ARG2	Modulus m	[0, 1]
RES1	m·cosθ	[-1, 1]
RES2	m·sinθ	[-1, 1]
SCALE	Not applicable	0

- This function calculates the cosine of an angle in the range -π to π
  - It can also be used to perform polar to rectangular conversion

life.augmented

This slide describes the features of the cosine function.
The primary argument is the angle θ in radians.
It must be divided by π before programming ARG1.
The secondary argument m is the modulus.
If m is greater than 1, a scaling must be applied in software to adapt it to the q1.31 range of ARG2.
The primary result, RES1, is the cosine of the angle, multiplied by the modulus.
The secondary result, RES2, is the sine of the angle, multiplied by the modulus.

Sine function

Parameter	Description	Range
ARG1	Angle θ in radians, dvided by π	[-1, 1]
ARG2	Modulus m	[0, 1]
RES1	m·sinθ	[-1, 1]
RES2	m·cosθ	[-1, 1]
SCALE	Not applicable	0

- This function calculates the sine of an angle in the range -π to π
  - It can also be used to perform polar to rectangular conversion

life.augmented

This slide describes the features of the sine function.
The primary argument is the angle θ in radians.
It must be divided by π before programming ARG1.
The secondary argument m is the modulus.
If m is greater than 1, a scaling must be applied in software to adapt it to the q1.31 range of ARG2.
The primary result, RES1, is the sine of the angle, multiplied by the modulus.
The secondary result, RES2, is the cosine of the angle, multiplied by the modulus.

Phase function

Parameter	Description	Range
ARG1	x coordinate	[-1, 1]
ARG2	y coordinate	[-1, 1]
RES1	Phase angle θ in radians, divided by π	[-1, 1]
RES2	Modulus m	[0, 1]
SCALE	Not applicable	0

- This function calculates the phase angle in the range -π to π of a vector v = [x y]
  - It can also be used to perform rectangular to polar conversion

life.augmented

This slide describes the features of the Phase function.
The primary argument is the x co-ordinate, that is, the magnitude of the vector in the direction of the x axis.
If |x| > 1, a scaling must be applied in software to adapt it to the q1.31 range of ARG1.
The secondary argument is the y co-ordinate, that is, the magnitude of the vector in the direction of the y axis.
If |y| > 1, a scaling must be applied in software to adapt it to the q1.31 range of ARG2.
The primary result, RES1, is the phase angle θ of the vector v.
RES1 must be multiplied by π to obtain the angle in radians.
Note that values close to π may sometimes wrap to -π due to the circular nature of the phase angle.
The secondary result, RES2, is the modulus, given by: $|\mathbf{v}|=\sqrt{(x^2+y^2}$).

If |v| > 1 the result in RES2 will be saturated to 1.

Modulus function

Parameter	Description	Range
ARG1	x coordinate	[-1, 1]
ARG2	y coordinate	[-1, 1]
RES1	Modulus m	[0, 1]
RES2	Phase angle θ in radians, divided by π	[-1, 1]
SCALE	Not applicable	0

- This function calculates the magnitude, or modulus, of a vector v = [x y]
  - It can also be used to perform rectangular to polar conversion

life.augmented

This slide describes the features of the Modulus function.
The primary argument is the x co-ordinate, that is, the magnitude of the vector in the direction of the x axis.
If |x| > 1, a scaling must be applied in software to adapt it to the q1.31 range of ARG1.
The secondary argument is the y co-ordinate, that is, the magnitude of the vector in the direction of the y axis.
If |y| > 1, a scaling must be applied in software to adapt it to the q1.31 range of ARG2.
The primary result, RES1, is the modulus, given by:
$|\mathbf{v}|=\sqrt{(x\text{^}2+y\text{^}2}$).
If |**v**| > 1 the result in RES1 will be saturated to 1.
The secondary result, RES2, is the phase angle θ of the vector **v**.
RES2 must be multiplied by π to obtain the angle in radians.
Note that values close to π may sometimes wrap to -π

due to the circular nature of the phase angle.

Arctangent function

Parameter	Description	Range
ARG1	$x.2^{-n}$	[-1, 1]
ARG2	N/A	-
RES1	$2^{-n}\tan^{-1} x$, in radians, dvided by π	[-1, 1]
RES2	N/A	-
SCALE	n	[0, 7]

- This function calculates the arctangent, or inverse tangent, of the input argument x

life.augmented

This slide describes the features of the Arc tangent function.

The primary argument, ARG1, is the input value, $x = \tan \theta$.

If $|x| > 1$, a scaling factor of 2^{-n} must be applied in software such that $-1 < x * 2^{-n} < 1$.

The scaled value $x * 2^{-n}$ is programmed in ARG1 and the scale factor n must be programmed in the SCALE parameter.

Note that the maximum input value allowed is $\tan \theta = 128$, which corresponds to an angle $\theta = 89.55$ degrees.

For $|x| > 128$, a software method must be used to find $\tan^{-1} x$.

The secondary argument, ARG2, is unused.

The primary result, RES1, is the angle $\theta = \tan^{-1} x$. RES1 must be multiplied by $2^n.\pi$ to obtain the angle in radians.

The secondary result, RES2, is unused.

Hyperbolic Cosine function

Parameter	Description	Range
ARG1	$x.2^{-n}$	[-0.559, 0559]
ARG2	N/A	-
RES1	$2^{-n}\cosh x$	[0.5, 0846]
RES2	$2^{-n}\sinh x$	[-0.683, 0683]
SCALE	n	1

- This function calculates the hyperbolic cosine of a hyperbolic angle x
  - It can also be used to calculate:
    - The exponential functions $e^x = \cosh x + \sinh x$
    - and $e^{-x} = \cosh x - \sinh x$

life.augmented

This slide describes the features of the Hyperbolic Cosine function.
The primary argument is the hyperbolic angle x.
Only values of x in the range -1.118 to +1.118 are supported.
Since the minimum value of cosh x is 1, which is beyond the range of the q1.31 format, a scaling factor of 2^{-n} must be applied in software.
The factor n = 1 must be programmed in the SCALE parameter.
The secondary argument, ARG2, is unused.
The primary result, RES1, is the hyperbolic cosine, cosh x.
RES1 must be multiplied by 2 to obtain the correct result.
The secondary result, RES2, is the hyperbolic sine, sinh x.
RES2 must be multiplied by 2 to obtain the correct result.

Hyperbolic Sine function

Parameter	Description	Range
ARG1	$x2^{-n}$	[-0.559, 0559]
ARG2	N/A	-
RES1	$2^{-n}\sinh x$	[-0.683, 0683]
RES2	$2^{-n}\cosh x$	[0.5, 0846]
SCALE	n	1

- This function calculates the hyperbolic sine of a hyperbolic angle x
  - It can also be used to calculate:
    - The exponential functions $e^x = \cosh x + \sinh x$
    - and $e^{-x} = \cosh x - \sinh x$

life.augmented

This slide describes the features of the Hyperbolic Sine function.
The primary argument is the hyperbolic angle x.
Only values of x in the range -1.118 to +1.118 are supported.
For all input values, a scaling factor of 2^{-n} must be applied in software, where n = 1.
The scaled value $x \cdot 0.5$ is programmed in ARG1 and the factor n = 1 must be programmed in the SCALE parameter.
The secondary argument, ARG2, is unused.
The primary result, RES1, is the hyperbolic sine, sinh x.
RES1 must be multiplied by 2 to obtain the correct result.
The secondary result, RES2, is the hyperbolic cosine, cosh x.
RES2 must be multiplied by 2 to obtain the correct result.

Hyperbolic Arctangent function

Parameter	Description	Range
ARG1	$x2^{-n}$	[-0.403, 0403]
ARG2	N/A	-
RES1	2^{-n} atanh x	[-0.559, 0559]
RES2	N/A	-
SCALE	n	1

- This function calculates the hyperbolic arctangent of the input argument x

This slide describes the features of the Hyperbolic arc tangent function.
The primary argument is the input value x.
Only values of x in the range -0.806 to +0.806 are supported.
The value x must be scaled by a factor 2^{-n}, where n = 1.
The scaled value x · 0.5 is programmed in ARG1 and the factor n = 1 must be programmed in the SCALE parameter.
The secondary argument, ARG2, is unused.
The primary result, RES1, is the hyperbolic arctangent, atanh x.
RES1 must be multiplied by 2 to obtain the correct value.
The secondary result is not used.

Natural Logarithm function

Parameter	Description	Range
ARG1	$x2^{-n}$	[0.054, 0875]
ARG2	N/A	-
RES1	$2^{-(n+1)} \ln x$	[-0.279, 0.137]
RES2	N/A	-
SCALE	n	[1, 4]

- This function calculates the natural logarithm of the input argument x

This slide describes the features of the natural logarithm function.
The primary argument is the input value x.
Only values of x in the range 0.107 to +9.35 are supported.
The value x must be scaled by a factor 2^{-n}, such that $x \cdot 2^{-n} < 1 - 2^{-n}$.
The scaled value $x \cdot 2^{-n}$ is programmed in ARG1 and the factor n = 1 must be programmed in the SCALE parameter.
The secondary argument is unused.
The primary result, RES1, is the natural logarithm.
RES1 must be multiplied by $2^{(n+1)}$ to obtain the correct value.
The secondary result is not used.

Square root function

Parameter	Description	Range
ARG1	$x2^{-n}$	[0.027, 0703]
ARG2	N/A	-
RES1	2^{-n}.	[0.04, 1]
RES2	N/A	-
SCALE	n	[0, 2]

- This function calculates the square root of the input argument x

life.augmented

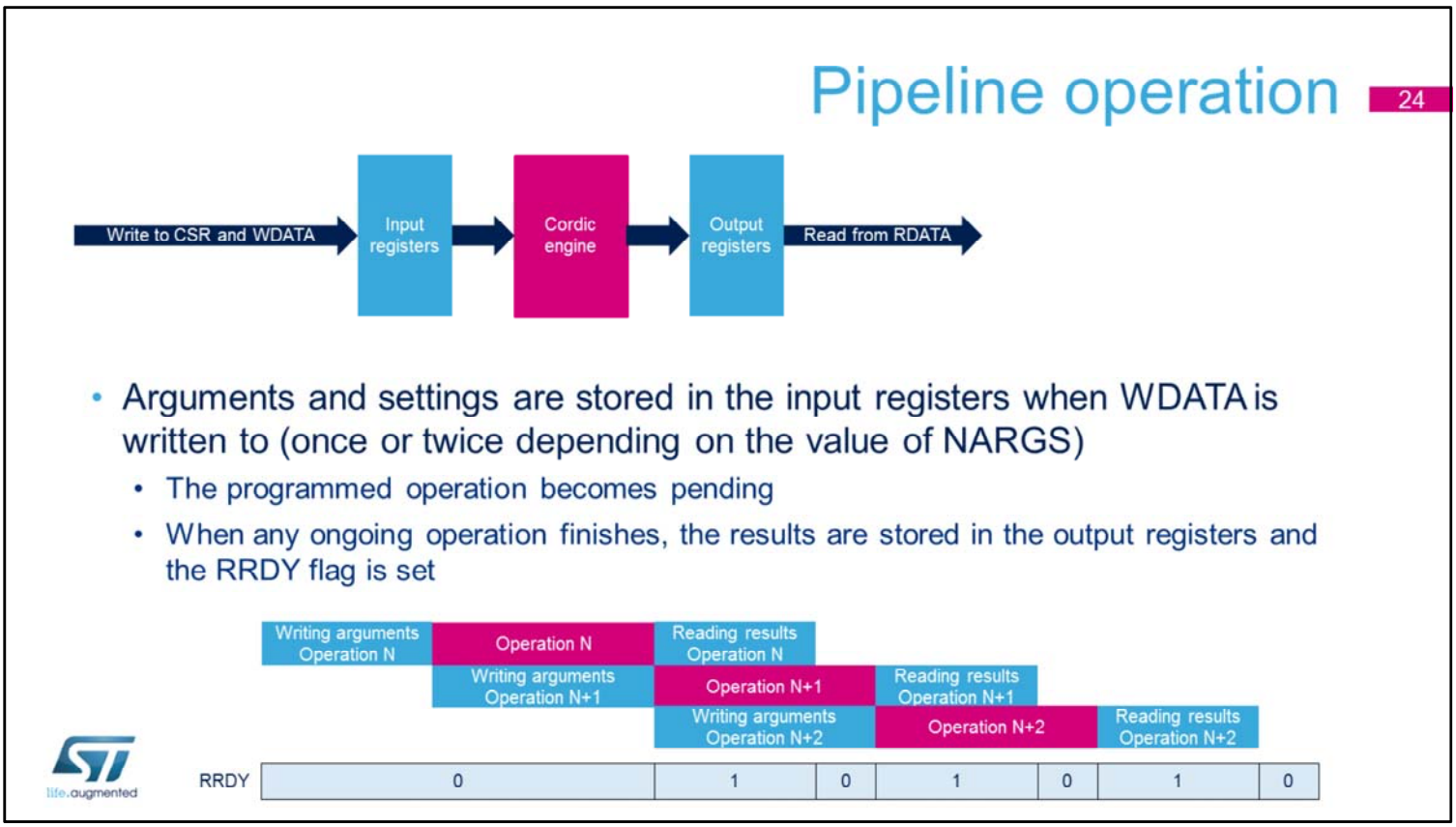
This slide describes the features of the square root function.
The primary argument is the input value x.
Only values of x in the range 0.027 to 2.34 are supported.
The value x must be scaled by a factor 2^{-n}, such that $x \cdot 2^{-n} < 1 - 2^{(-n-2)}$.
The scaled value $x \cdot 2^{-n}$ is programmed in ARG1 and the factor n = 1 must be programmed in the SCALE parameter.
The secondary argument is unused.
The primary result, RES1, is the square root of x.
RES1 must be multiplied by 2^n to obtain the correct value.
The secondary result is not used.

Result register

- Result register (CORDIC_RDATA)
  - A read access to CORDIC_RDATA while an operation is ongoing will wait until the results are available before completing
  - This means there is no need to poll the RRDY flag
  - The results are returned to the CPU as soon as they are available

life.augmented

The software that subcontracts a calculation to the Cordic block does not need to poll a flag to determine when this calculation is completed.
It simply initiates a read request of the RDATA register through the AHB bus.
As any AHB transaction, the slave is permitted to insert wait states by maintaining HREADY signal low.
One the results are available, the Cordic block asserts HREADY, which completes the transaction.
In the meantime, the Cortex-M4 processor is frozen.
This approach is called zero-overhead mode.



As soon as the results have been read from RDATA (one or two reads depending on the value of NRES), the pending operation is started.
A new set of arguments (and settings) can be written as long as there is no operation pending.
This means that time spent waiting for the Cordic operation to complete can be used to prepare the next operation, and the Cordic is never idle.
The CORDIC_CSR register can be re-programmed while a calculation is in progress, without affecting the result of the ongoing calculation.

Zero-overhead single-shot mode

- The zero-overhead single-shot mode is the fastest way of performing a single calculation

1. Program the CORDIC_CSR register with the appropriate settings, if necessary
2. Program the argument(s) for the calculation in the CORDIC_WDATA register
   - ➢ This will launch the calculation
3. Read the result(s) from the CORDIC_RDATA register

life.augmented

The sequence described jn this slide summarizes the use of the CORDIC_IP in zero-overhead mode assuming a single-shot operation.
No further calculation is scheduled so the processor simply waits for the completion of the current operation.

Zero-overhead pipelined mode

- The zero-overhead pipelined mode is the fastest way of performing several successive calculations

1. Program the CORDIC_CSR register with the appropriate settings
2. Program the argument(s) for the first calculation in the CORDIC_WDATA register
   - ➢ This will launch the first calculation
3. If needed, update the CORDIC_CSR register settings for the next calculation
4. Program the argument(s) for the next calculation in the CORDIC_WDATA register
5. Read the result(s) from the CORDIC_RDATA register
   - ➢ This will trigger the next calculation
6. Go to step 3
7. At the end of the sequence, an additional read is required to obtain the results of the final operation

life.augmented

The sequence described in this slide summarizes the use of the CORDIC_IP in zero-overhead mode assuming pipelined operations.
By iterating the steps 3 to 6, software can re-execute the same operation for an array of arguments.
The seventh step is required to obtain the result of the last operation.

Polling mode

- The polling mode takes longer due to the delay between reading the RRDY flag high and reading the results
  - However it allows the processor to be interrupted while waiting for the results

1. Program the CORDIC_CSR register with the appropriate settings, if necessary
2. Program the argument(s) for the calculation in the CORDIC_WDATA register
   - ➢ This will launch the calculation
3. Poll the RRDY flag in the CORDIC_CSR register until it goes high
4. Read the result(s) from the CORDIC_RDATA register

- Pipelining can also be used in polling mode

life.augmented

The sequence described in this slide summarizes the use of the CORDIC_IP in polling mode.
When a new result is available in the CORDIC_RDATA register, the RRDY flag is set in the CORDIC_CSR register.
The flag can be polled by reading this register.
It is reset by reading the CORDIC_RDATA register (once or twice depending on the NRES field of the CORDIC_CSR register).
Polling the RRDY flag takes slightly longer than reading the CORDIC_RDATA register directly, since the result is not read as soon as it is available.
However the processor and bus interface are not stalled while reading the CORDIC_CSR register, so this mode may be of interest if stalling the processor is not acceptable, for instance when low latency interrupts must be serviced.

Interrupt mode

- Instead of polling the RRDY flag, the CPU can be interrupted when the results are available
  - Handling the interrupt takes extra cycles, but this mode allows the priority of the Cordic to be set with respect to other tasks

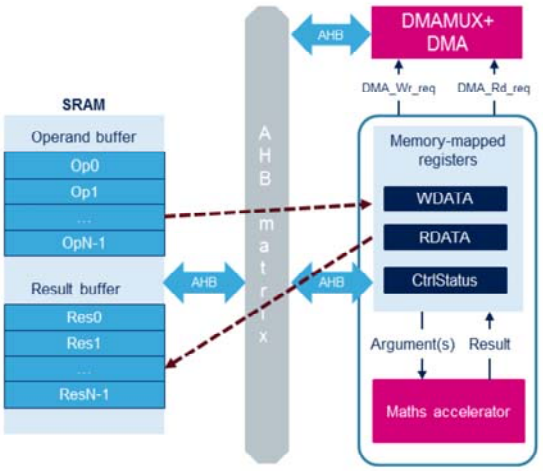
1. Program the CORDIC_CSR register with the appropriate settings, and set the IEN bit
2. Program the argument(s) for the calculation in the CORDIC_WDATA register
   - ➢ This will launch the first calculation
3. The Cordic interrupt will fire when the results are ready
4. Read the result(s) from the CORDIC_RDATA register under the interrupt handler
   - ➢ This will reset the RRDY flag and clear the interrupt request
5. Pipelining can be used in interrupt mode
   - ➢ Be careful to maintain the order of writes and reads!

life.augmented

The sequence described in this slide summarizes the use of the CORDIC_IP in interrupt mode.
By setting the interrupt enable (IEN) bit in the CORDIC_CSR register, an interrupt will be generated whenever the RRDY flag is set.
The interrupt is cleared when the flag is reset.
This mode allows the result of the calculation to be read under interrupt service routine, and hence given a priority relative to other tasks.
However it is slower than directly reading the result, or polling the flag, due to the interrupt handling delays.

DMA mode

- DMA mode can be used to perform multiple calculations using the same settings
  - Not only does it speed up the calculations compared to a software implementation, but it frees up the CPU for other tasks
- The DMA can be used to write arguments to the Cordic
  - Each time a new calculation starts, a DMA write channel request is generated to fetch the next argument(s) from memory
- Similarly, the DMA can be used to read the results
  - Each time a calculation completes (RRDY flag goes active), a DMA read channel request is generated to copy the results to memory



DMA mode is very efficient when performing multiple calculations using the same settings.
It is not possible to modify the CORDIC_CSR register by DMA.
Consequently if the settings need to be changed, the DMA should be stopped first, and restarted once the new settings have been programmed.
DMA write can be combined with DMA, polling, or interrupt read methods.
Pipelining is always used in DMA mode.
DMA write requests are enabled by setting the DMAWEN bit in the CORDIC_CSR register.
DMA read requests are enabled by setting the DMAREN bit in the CORDIC_CSR register.

Cordic vs ARM fast math

Fixed-point sine

- Convert a buffer of 3024 angles stored in memory to samples representing a 1kHz sine wave sampled at 48 ksps, 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

life.augmented

The purpose of this slide is to compare the performance of Cordic and ARM fast math when calculating a fixed-point sine.
By using the q1.15 and q1.31 formats, performance ratios are identical.
Cordic is 5 times faster in zero-overhead mode and 3 times faster in DMA in and out mode.

Cordic vs ARM fast math

Floating-point sine

- 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 ➢ Includes conversion fom foat32 to int32 and back
-	x3 acceleration
100%CPU	100% CPU
Max error: 0.00002	Max error: 0.000002
-	x10 precision

life.augmented

The purpose of this slide is to compare the performance of Cordic and ARM fast math when calculating a floating-point sine.
Cordic is 3 times faster in zero-overhead mode, including the conversion time from float32 to int32 and back.

Cordic vs ARM fast math

Square root, fixed point

- 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

life.augmented

The purpose of this slide is to compare the performance of Cordic and ARM fast math when calculating a fixed-point square root.
By using the q1.15 and q1.31 formats, performance ratios are identical.
Cordic is 14 times faster in zero-overhead mode.

Cordic vs ARM fast math

Buffered square root, floating point

32-bit floating point, precision = 3	
ARM fast math arm_sqrt_f32()	Cordic: zero-overhead mode
27 cycles/sample	21 cycles/sample ➢ Includes conversion fom foat32 to int32 and back
-	x1.3 acceleration
100%CPU	100% CPU
Max error: 0.00000003	Max error: 0.0000015
-	x0.02 precision

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

life.augmented

The purpose of this slide is to compare the performance of Cordic and ARM fast math when calculating a floating-point square root.
Cordic is 1.3 times faster in zero-overhead mode, including the conversion time from float32 to int32 and back.

Cordic vs ARM fast math

Park transform

- The Park transform is widely used in motor control applications:
  - X = D.cos θ - Q.sin θ
  - Y = D.sin θ + Q.cos θ

q1.15-bit fixed point, precision = 4	
ARM fast math arm_sqrt_q15()	Cordic: zero-overhead mode
243 cycles	48 cycles
-	x5 acceleration

life.augmented

The purpose of this slide is to compare the performance of Cordic and ARM fast math when calculating a fixed-point q1.15 park transform.
Cordic is 5 times faster in zero-overhead mode.

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 0/Stop 1	Not available
Stop 2	
Standby	
Shutdown	

life.augmented

The Cordic 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

life.augmented

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