HSP processing functions on STM32 MCUs

Introduction

The HSP is a hardware signal processor engine of STM32 MCUs. This document describes the HSP processing functions and their programming interfaces (HSP firmware API). In general, it is not required for a user to directly program the HSP firmware. A large variety of efficient high-level C functions executed by the CPU are available on STM32Cube ecosystem to ease the call of HSP firmware functions.

Several software packages are a available:

- HAL functions to control and configure all the HSP hardware subblocks
- A set of middleware functions to handle the accelerator and sequencer mode
- A set of CNN functions allowing acceleration of neural networks

Note: Cube-AI supports the HSP as accelerator and can map execution of neural network layers into HSP.

Below is the list of products to which this document applies.

Table 1. Applicable products

Type	Products
Microcontrollers	STM32U3B5/3C5 line
	STM32H7P4/P5 line

1 General information

This document applies to STM32 Arm® Cortex®-based MCUs.

arm

Note: Arm and Cortex are registered trademarks of Arm Limited (or its subsidiaries or affiliates) in the US and/or elsewhere.

2 HSP overview

The hardware signal processor (HSP) is a dedicated signal processor engine. It can offload and speed-up processing initially handled by the CPU.

It offers two categories of processing functions, standard functions, generally targeting the following applications:

- Control loops
- Metering
- General purpose signal processing

and CNN functions, capable of executing small neural networks.

The HSP is built around a dedicated high-performance signal processor engine (SPE) working in 32-bit floating-point for standard signal processing, and 8-bit integer for CNN. A part of its microcode is embedded into a private CROM, while the other part is loaded into the CRAM during the initialization phase. This code is generally named HSP firmware.

Standard processing and CNN functions are called **processing functions** in this document.

As shown in Figure 1, the HSP proposes a sequencer to handle the processing lists, and a direct command interface for the accelerator function.

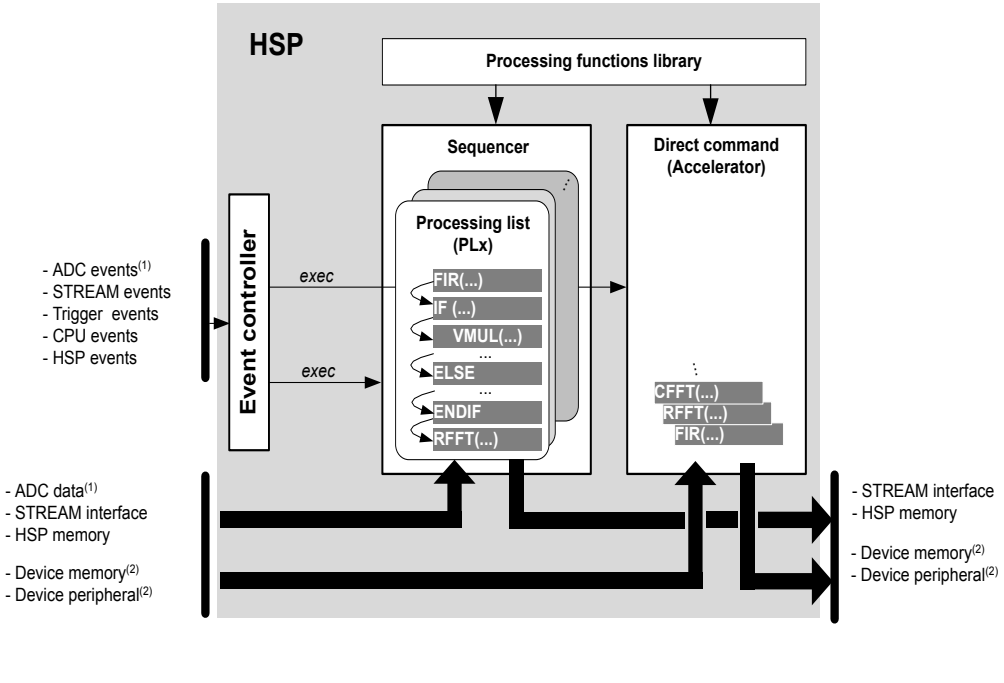
- In sequencer mode, the application can record **processing lists** at any moment. Each processing list (PL) contains the sequence of processing functions to be executed.
  Each processing list execution is triggered by an event. The application can select the event triggering the execution of a processing list.
  Events can be generated by peripherals or internal HSP events. The processing lists allow the application to automate computation sequences, making the HSP autonomous with respect to the main CPU.
- The accelerator mode allows the application to execute a processing function immediately.

The HSP embeds efficient integer-to-float and float-to-integer functions allowing the processing of 32-bit and 16-bit integers.

As shown in the figure, when an event associated with a processing list occurs, the event controller (EVTC) asks the SPE to execute the corresponding processing list. The processing lists take data from various sources. If the data type is different from 32-bit floating-point, the processing list converts them into floating-point if necessary. The result of the processing list computation can also be stored at various locations. Data can be converted to integer if needed.

Note: In this document the wording float32 is used instead of 32-bit floating-point.

Figure 1. HSP way of working



1. *The ADCIN interface is not present in all products.*
2. *The HSPDMA and STI are not present in all products.*

2.1 Sequencer and accelerator modes

The HSP is working in sequencer mode when it executes processing lists. The HSP is working in accelerator mode when executing direct commands.

The sequencer mode offers several advantages:

- Possibility to automate a sequence of operations, making the HSP autonomous.
- Very low latency: during the recording phase of a processing list, the HSP translates each processing function into an efficient code for the SPE. During the execution phase it is not necessary to decode any command.

The benefit of using the accelerator mode is the flexibility and simplicity:

- All function arguments can be changed every time the function is executed.
- The application writes the command and waits for the result or performs other operations while waiting for the command.

Figure 2 shows a detailed view of the HSP firmware data organization.

To be as efficient as possible, during the recording of a processing list, the HSP performs several operations:

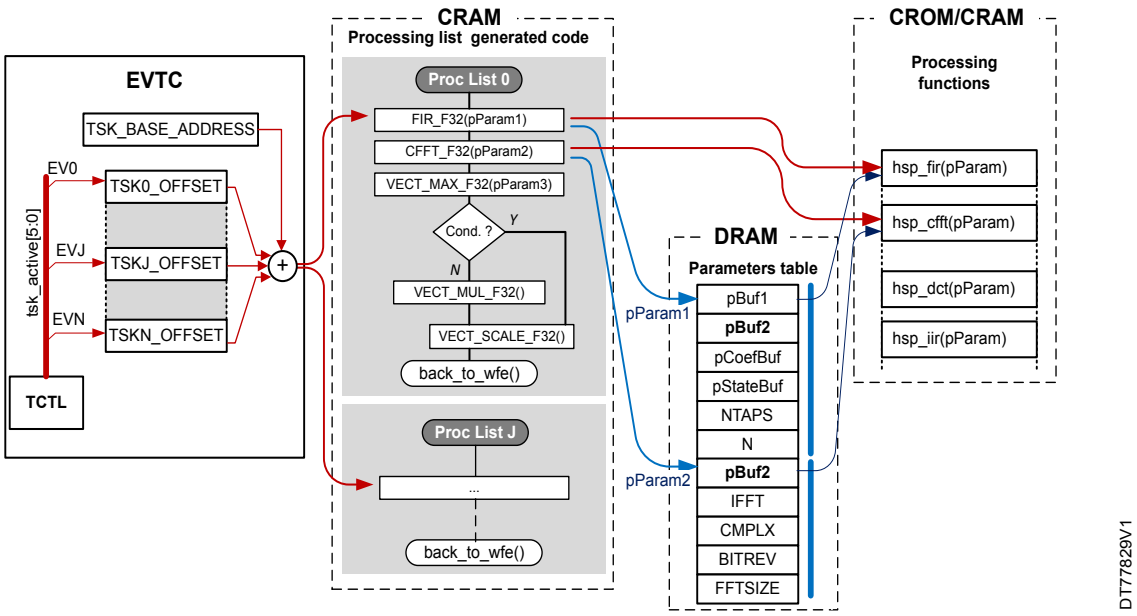
- It builds up the processing list code.
- It builds up the parameter table (**ParamTab**).
- It programs the start address of each processing list into the EVTC.

The processing lists are stored into the CRAM, and the parameter table into the DRAM.

As soon as an event requests the execution of a processing list, the SPE jumps to the base address of the processing list using to the address provided by the EVTC. Each command of a processing list calls a processing function, with a floating-pointer giving the location where the parameters can be found.

In the figure below, the buffer pBuf2, which is the output of the FIR operation, is reused as input for the CFFT.

Figure 2. HSP firmware processing list sequencer overview



2.2 Data buffer handling for standard functions

The HSP working buffers are located in the BRAM. Computation is always performed in the BRAM-AB memory, the data exchange between the HSP and application can use the BRAM-AB or STREAM interface.

The data sharing or exchange through the BRAM must be done through the BRAM-AB section. Using the BRAM-AB section means that the content of BRAM-A is the same as BRAM-B. BRAM-AB allows the exchange of data blocks.

The STREAM interface is dedicated to the processing that is executed every time new data is available, such as loop control.

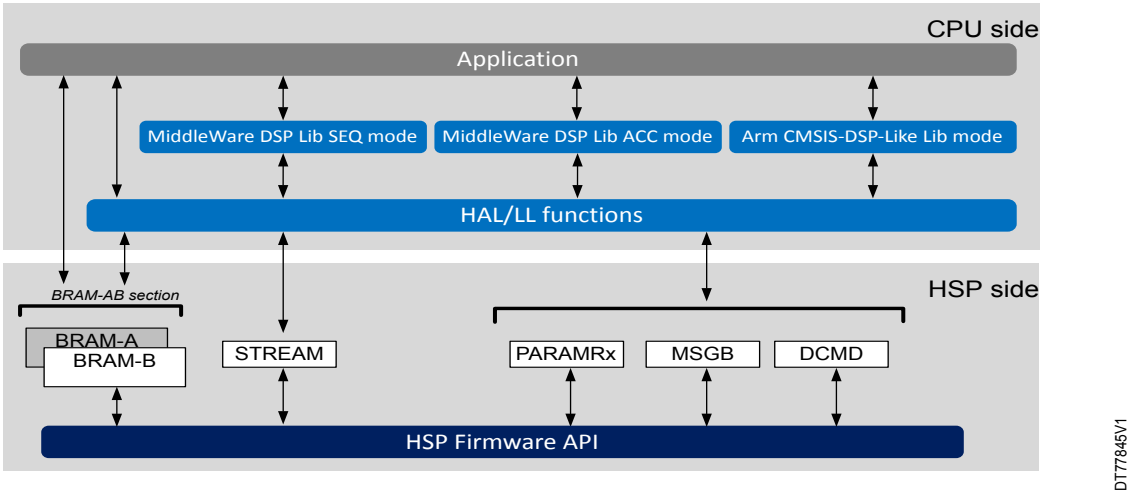
Figure 3 shows the hardware blocks involved for the accelerator mode or sequence recording. The figure also shows a simplified view of the software architecture for middleware and HAL/LL layers.

The HAL/LL functions directly interface with the HSP hardware blocks. The application therefore directly access BRAM-AB.

It is recommended for the application to use middleware functions to record a processing list, or to execute a processing function in accelerator mode. Note that middleware functions rely on the HAL/LL functions to access HSP hardware resources.

Also note that CMSIS-DSP-like functions are fully compatible with cmsis-dsp functions from Arm®, offering a simple way to benefit from HSP acceleration.

Figure 3. Data sharing models for standard functions



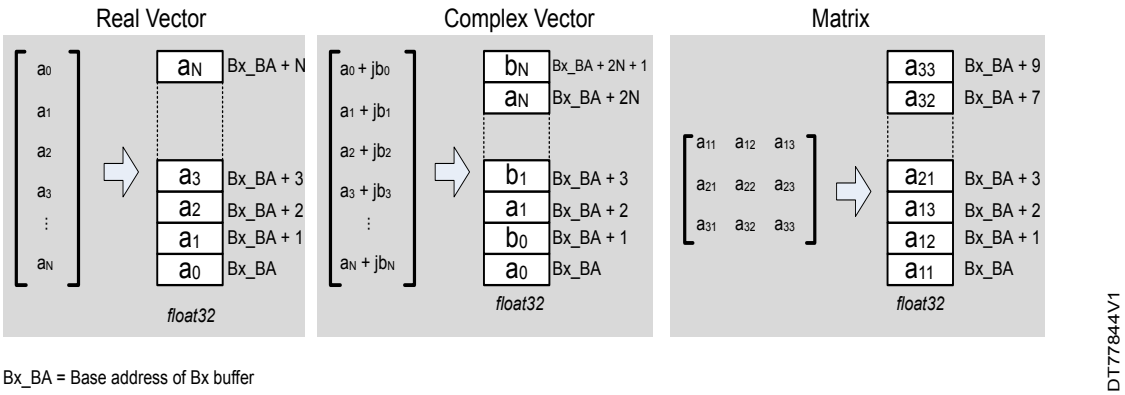
1. *System memory path can be used only for HSP version with HSPDMA & STI. ADCIN is not present in all HSP versions.*

2.2.1 Vector, matrix operations, and formats

The processing functions can directly use data from internal data buffers having the expected format. The expected format must be as follows:

- Generally, data buffer elements are float32. There are some exceptions such as functions translating from integer-to-float or float-to-integer. Refer to Section 5: Processing functions description.
- In the case of real vectors, all elements must be contiguous, and arranged in ascending order of address (see Figure 4).
- In the case of complex vectors ($a_x + jb_x$), real and imaginary parts are interleaved, and elements are arranged in ascending order of address.
- In the case of real matrix, the elements must be arranged per line, as shown in the figure below.

Figure 4. Format accepted by processing functions



If the data buffer does not respect this format, it must be reformatted using formatting functions of the HSP, or CPU capabilities.

When a processing function performs an operation on data buffers, there is no need to have the same size for all data buffers. The only requirement is that they are large enough to support the operation.

2.2.2 Supported data pointers

Some of the parameters provided to the processing functions indicate the location or value of input/output data buffers. The same parameter, may contain one of the following objects:

- A pointer on a data buffer located into the BRAM-AB memory (ABMP)
- A pointer on a pointer of a data buffer located into the BRAM-AB (POP)
- A reference to data located into a peripheral inside the HSP (STREAM) also called HSP interface buffer pointer (IBP)
- A reference to an immediate data (IMM)

For parameters supporting several types of objects, the HSP must know which kind of object is used. For that purpose, a specific parameter named IOTYPE, must indicate the type of those parameters.

The data buffers also allow data exchange between the various processing functions. Data buffers are visible by all processing functions even if they are located in different processing lists.

A data buffer can represent a single value or a vector, a matrix with a size limited to 4096 elements. It is up to the application to ensure that the processing functions are using data with the expected format.

It is up to the application to instantiate data buffers into the BRAM-AB section for processing functions.

2.2.3 Data pointer handling

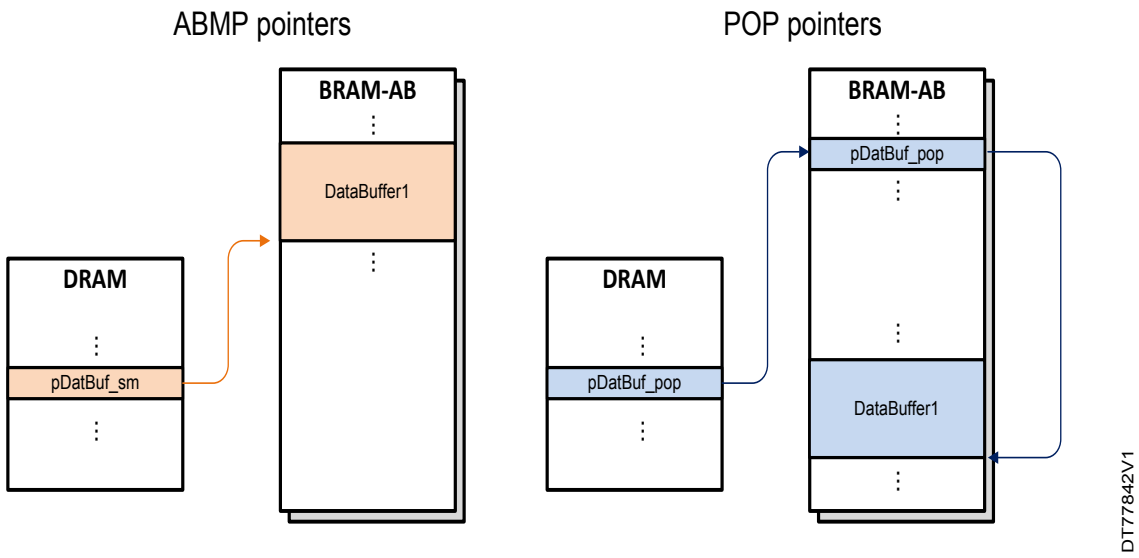
2.2.3.1 BRAM-AB memory and pointer-on-pointer (ABMP, POP)

The data in the BRAM-AB can be read or written directly by the CPU or by any other master of the product. An arbiter inside the MMC avoids contention between SPE and other masters. This arbiter gives the priority to SPE internal accesses, and other master accesses can be delayed if the SPE has an important activity on this memory. It is possible to balance the bandwidth between the other master accesses and SPE, refer to the HSP user specification for details.

The BRAM-AB memory pointers (noted ABMP) must be used to address data buffers located into the BRAM-AB. Once the processing function is recorded, the value of these pointers cannot be changed.

The pointers-on-pointers (noted POP) can also be used to address data buffers located into the BRAM-AB. In the case of POP objects, the parameter passed to the processing function is not a pointer on a data buffer, but a pointer on a BRAM-AB location containing a pointer to the data buffer. The main difference is that the value of this BRAM-AB location representing a pointer can be changed by the processing functions or application.

Figure 5. **ABMP and POP pointers**



For standard functions, the size of each element in the BRAM-AB must be 16 or 32 bits. The data buffers must be converted to the format accepted by the processing functions before executing them. The size of an ABMP data buffer can be 1 to 4096 elements if the BRAM-AB size is large enough.

Using the BRAM-AB gives the best HSP performance and must be preferred.

2.2.3.2 *HSP interface buffers pointers (IBP)*

The HSP interface buffer pointers (IBP) are mainly used in streaming mode in order to get data from RXBUFF[3:0] or write data to TXBUFF[3:0]. Data read from RXBUFF[3:0] can be directly converted into float32 by the hardware.

For additional information, refer to Section 7.5: Programming a simple processing list using IBP pointers.

Note: IBP cannot be used when running a processing function as a direct command.

2.2.4 Buffer pointer transmission on processing lists

2.2.4.1 *Using BRAM-AB memory pointers (ABMP)*

The BRAM-AB location is seen at a different address by the CPU and by the SPE. When the CPU programs a processing function using pointers to buffers located into the BRAM-AB, it must translate the pointers into pointers directly usable by the SPE, and set the corresponding field of IOTYPE to 0.

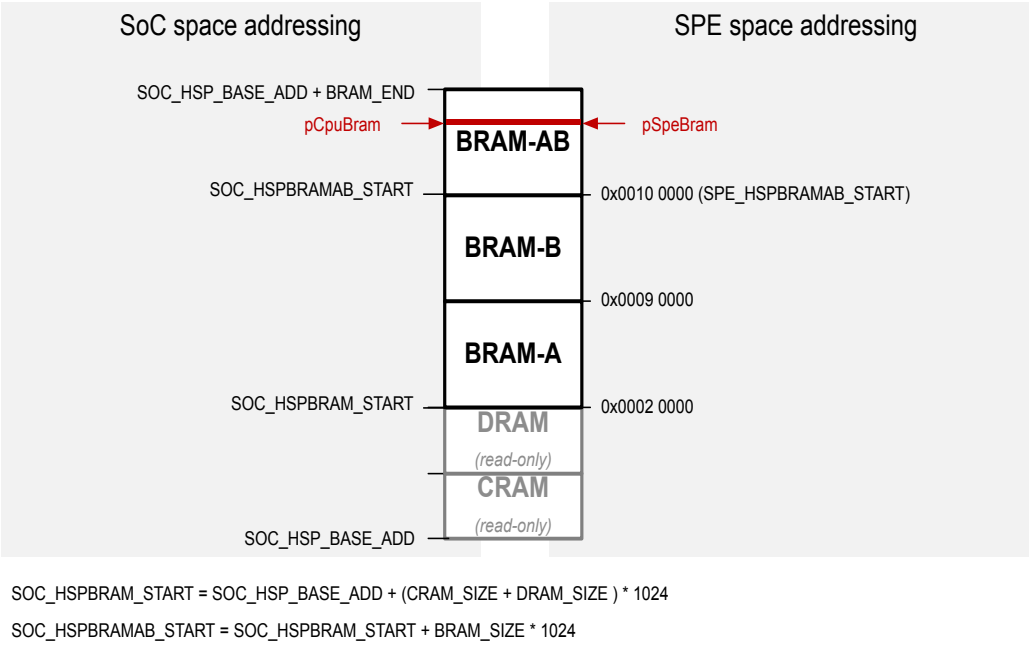
To translate a pointer on BRAM-AB used by the CPU to the same pointer used by the SPE, the application must perform the following operation:

pSpeBram = pCpuBram - SOC_HSPBRAMAB_START + SPE_HSPBRAMAB_START

Where SOC_HSPBRAMAB_START is the BRAM-AB start address in byte, and SPE_HSPBRAMAB_START is a constant fixed to 0x0010 0000.

pSpeBram and pCpuBram are pointers to the same BRAM location seen respectively by the SPE and CPU.

Figure 6. **BRAM mapping on SoC and SPE side**



Note: For additional information, refer to Section 7.4: Programming a simple processing list using ABMP pointers.

When middleware functions are used to program a processing list, the application does not need to consider this address translation, the middleware functions handle it.

During the runtime, the sequencer simply runs the processing function. Overhead is reduced to its minimum.

2.2.4.2 *Using pointers-on-pointers (POP)*

To use a pointer-on-pointer, the application must create two objects inside the BRAM-AB:

- The data buffer
- A pointer on the above data buffer

Similarly to ABMP pointers, when the CPU programs a processing function using POP, it must translate the address location containing the pointer into an address directly usable by the SPE, and set the corresponding field of IOTYPE to the value identifying POP pointers.

To translate a pointer on BRAM-AB used by the CPU to the same pointer used by the SPE, the application must perform the following operation:

pSpeBramPoP = pCpuBramPoP - SOC_HSPBRAMAB_START + SPE_HSPBRAMAB_START

Where SOC_HSPBRAMAB_START is the BRAMAB start address in byte, and SPE_HSPBRAMAB_START is a constant fixed to 0x0010 0000.

pSpeBramPoP and pCpuBramPoP are pointers to the same BRAM location seen respectively by the SPE and CPU.

The pointer located into the BRAM-AB represents a SoC address. During the runtime, before executing a processing function containing POP pointers, the sequencer of the HSP reads the pointer on the BRAM-AB, translates it to an address directly usable by the SPE, and runs the processing function.When the CPU records a processing function using pointers to HSP interface buffers, it must:

- Provide a specific value identifying the register to be accessed.
- Set the corresponding field of IOTYPE to 0.

The allowed register addresses for HSP interface buffer pointers are listed in the table below:

Table 2. **HSP interface buffer identification values**

Register names	Identification values
HSP_REG_SPE_BUFF0DR	0x0020 10F8

Register names	Identification values
HSP_REG_SPE_BUFF1DR	0x0020 10FC
HSP_REG_SPE_BUFF2DR	0x0020 1100
HSP_REG_SPE_BUFF3DR	0x0020 1104

2.2.5 Buffer pointer transmission on direct commands

The execution of a direct command uses the DCMD interface and HSP_PARAMRx registers. DCMD optimizes the latency, and the way to program it is slightly different than recording a processing function into a processing list.

For direct commands, the ABMP pointer must not be translated by the CPU. For performances reasons, the SPE does the translation.

Note: It is not necessary to provide IOTYPE.

2.2.6 IOTYPE description

The IOTYPE field, when present, is located in the HSP_PARAMR15 register. The processing function parameters generally contain pointers to input/output buffers or immediate values. In addition, a same parameter may contain a pointer or an immediate value. IOTYPE must be used to inform the HSP about the type of object contained in the processing functions parameters.

IOTYPE specifies the type or target of up to three pointers. It is not allowed to combine types and target for a same pointer. The table below shows the allowed combinations for each pointer.

Table 3. Allowed IOTYPE combinations

POPx	IMM	Target
0	0	The pointer target is a ABMP or IBP. The value of the pointer defines the target. The sequence-builder just inserts inside the processing list the selected processing function.
0	1	It is an immediate value. The sequence-builder stores the immediate value and inserting the call to the selected processing function with a link to this immediate value.
1	0	The pointer target is POP. The sequence-builder must insert inside the processing list a function reading the pointer on BRAM-AB before inserting the call to the selected processing function.

The table below gives the IOTYPE format.

Table 4. IOTYPE 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
res	res	res	res	res	res	res	res	res	res	res	res	res	res	res	res	res	res	res	IMM	res	res	res	res	res	POP2	POP1	POP0	res	res	res	res

IMM	Defines if the scalar parameter is an immediate or a pointer 0: Is a pointer 1: Is an immediate value
POP2	Defines if the third pointer is a pointer-on-pointer If this field is different from 0, GMP2 must be set to 0. 0: It is not a POP 1: It is a POP

POP1	Defines if the second pointer is a pointer-on-pointer If this field is different from 0, GMP1 must be set to 0. 0: It is not a POP 1: It is a POP
POP0	Defines if the first pointer is a pointer-on-pointer If this field is different from 0, GMP0 must be set to 0. 0: It is not a POP 1: It is a POP

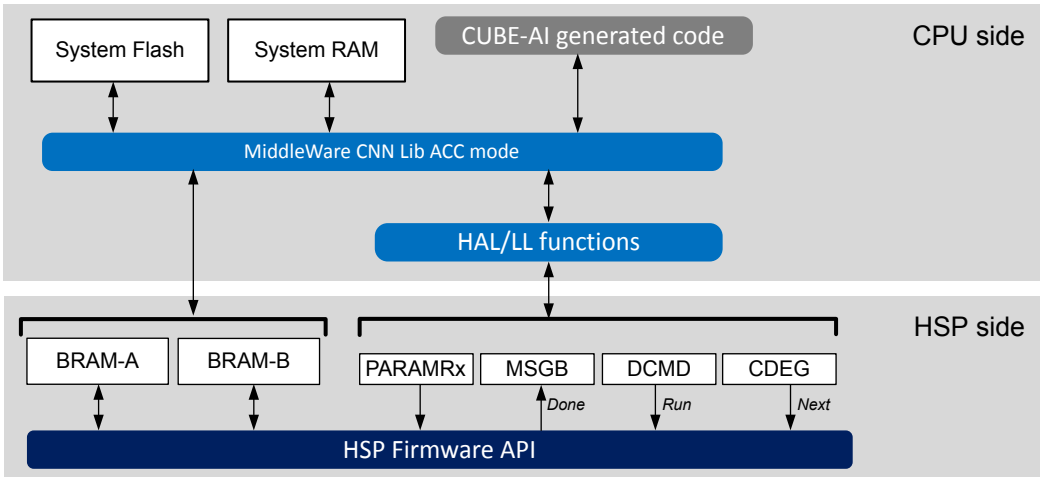
3 Data buffer handling for CNN functions

When the HSP is executing CNN functions, all events must be disabled except the DCMD interface and the CDEG block. The execution of a neural network cannot be interrupted by the execution of a standard processing list.

For CNN functions, the HSP working buffers are the BRAM-A and BRAM-B sections (not BRAM-AB). Compared to standard processing functions, the CNN functions request a synchronization mechanism between the code executed in the HSP and the code executed on the CPU side. It is strongly recommended to use Cube-AI code generator.

Note: The CUBE-AI can generate a code for complete neural networks, using the HSP as accelerator every time it is possible. Execution of a neural network generally mixes functions executed by the CPU and functions executed by the HSP.

Figure 7. Data sharing models for CNN functions



The use of CNN functions requires the following HSP hardware resources:

- BRAM-A, BRAM-B
- MSGB
- PARAMRx
- DCMD
- CDEG

CNN processing functions cannot be recorded into processing list.

3.1 Memory mapping

For CNN functions, the data exchange between the HSP and CNN functions running on the CPU side must be done through the BRAM-A and BRAM-B sections.

The BRAM memory mapping for the execution of each layer depends on the operation type, and on the amount of data to process.

As shown in the figure below, the BRAM contains the following elements:

- For BRAM-A
  - A circular buffer for data input
  - A ping-pong buffer for data output (depends on the layer characteristics)
  - A buffer for bias values
  - A buffer for quantization values
- BRAM-B contains kernels
  - A buffer for data output, line per line (depends on the layer characteristics)
  - A single or ping-pong buffer for kernels (that is, weights)

Note: The output, input and kernel data format is int8.

4 Processing functions summary

In sequencer or accelerator mode, the HSP is using optimized processing functions stored into its CROM. These are parts of the HSP firmware API.

To optimize the execution time of some basic processing functions such as scalar addition and subtraction, the sequence builder of the HSP is using ‘in-line’ functions. In-line functions are directly inserted into the processing lists. They are only used in sequencer mode.

Functions with a mnemonic starting by “__” are in-line functions. In-line functions are optimized to reduce the overhead as much as possible. As a consequence, they cannot be traced by HSP_SNP bus.

The middleware functions requesting HSP services generally rely on processing functions.

Processing functions are identified in this document by a mnemonic, and each processing function has a unique function ID, see Section 8: Function ID list for details.

The processing function FIR_F32 is used in several middleware functions, for example:

- HSP_SEQ_Fir_f32() is used to record inside a processing list the processing function FIR_F32. The processing list is executed only if its associated event becomes active.
- HSP_ACC_Fir_f32() and stm32_hsp_fir_f32() trigger immediately the execution of FIR_F32.
- HSP_ACC_Fir_f32() is the generic middleware function used in accelerator mode, and stm32_hsp_fir_f32() is fully compatible with the arm_fir_f32() function and uses the HSP as accelerator

Table 5. List of functions gives a summary of all functions the application can use to program processing lists, or to execute in accelerator mode.

In the table below:

- The column ‘Type’ indicates if the function can be executed directly in accelerator mode (ACC) or recorded into a processing list (SEQ).
- The column ‘STREAM’ indicates if input or output data can be taken from the STREAM interface: ‘Y’ means yes for sequencer mode, and ‘N’ means not possible.
- The column ‘BRAM’ indicates which BRAM section must be used to exchange data: DP means that BRAM-AB section must be used, SP means BRAMA, and/or BRAM-B sections must be used.

Each processing function has a unique function ID, see Section 8: Function ID list for details.

Table 5. List of functions

Processing functions mnemonic	Description	Type	STREAM	BRAM	GMP
Configuration functions					
FW_INIT	Firmware initialization command	-	-	-	-
PROCFG_START	Starts the creation of a processing list	-	-	-	-
PROCFG_END	Ends the creation of a processing list	-	-	-	-
PL_RESET	Processing list reset	-	-	-	-
Filter functions					
BIQDF1_CASC_F32	Cascaded biquad direct form 1 filtering	SEQ, ACC	Y	DP	Y
BIQDF2T_CASC_F32	Cascaded biquad direct form 2 transposed filtering	SEQ, ACC	Y	DP	Y
FIR_F32	Optimal FIR filtering	SEQ, ACC	Y	DP	Y
FIRDEC_F32	FIR filtering with decimation	SEQ, ACC	N	DP	Y
FIRLMS_F32	FIR filtering with adaptive LMS algorithm	SEQ, ACC	Y	DP	Y

Processing functions mnemonic	Description	Type	STREAM	BRAM	GMP
IIRDF1_F32	Optimal IIR filtering using the direct form 1	SEQ, ACC	Y	DP	Y
IIR_LATTICE_F32	Performs an IIR filtering using the Lattice form	SEQ, ACC	Y	DP	Y
FLTBANK_F32	Performs frequency filtering using internal filter banks	SEQ, ACC	N	DP	-
CONV_F32	Performs the convolution between two real vectors	SEQ, ACC	N	DP	-
CORR_F32	Performs the correlation between two real vectors	SEQ, ACC	N	DP	-
GET_STATE	Gets the filter state for single stage filters	ACC	N	DP	-
GET_STATE_IIRDF1	Gets the filter state for IIRDF1	ACC	N	DP	-
GET_STATE_BIQ	Gets the filter state for biquads	ACC	N	DP	-
SET_STATE	Sets the filter state for single stage filters	ACC	N	DP	-
SET_STATE_IIRDF1	Sets the filter state for IIRDF1	ACC	N	DP	-
SET_STATE_BIQ	Sets the filter state for biquads	ACC	N	DP	-
RST_STATE	Reset the filter state	ACC	N	DP	-
Vector functions on real numbers					
VECT_ABS_F32	Element-related absolute value	SEQ, ACC	N	DP	Y
VECT_ABSMAX_F32	Maximum absolute value of a vector	SEQ, ACC	N	DP	Y
VECT_ADD_F32	Element-related addition	SEQ, ACC	N	DP	Y
VECT_ATAN2_F32	Element-related arc tangent 2	SEQ, ACC	N	DP	Y
VECT_AVG_F32	Average value	SEQ, ACC	N	DP	Y
VECT_COPY	Vector copy	SEQ, ACC	N	DP	Y
VECT_COS_F32	Element-related cosine	SEQ, ACC	N	DP	Y
VECT_DEC	Vector decimation	SEQ, ACC	N	DP	Y
VECT_DIV_F32	Element-related division	SEQ, ACC	N	DP	Y
VECT_DOTP_F32	Dot product	SEQ, ACC	N	DP	Y
VECT_EXP_F32	Element-related exponential	SEQ, ACC	N	DP	Y
VECT_EXP10_F32	Element-related 10x	SEQ, ACC	N	DP	Y
VECT_FTOI	Conversion of float32 to 32-bit signed integers	SEQ, ACC	N	DP	Y
VECT_FTOU	Conversion of float32 to 32-bit unsigned integers	SEQ, ACC	N	DP	Y

Processing functions mnemonic	Description	Type	STREAM	BRAM	GMP
VECT_ITOF	Conversion of signed 32-bit integers to float32	SEQ, ACC	N	DP	Y
VECT_I24TOF	Conversion of signed 24-bit integers to float32	SEQ, ACC	N	DP	Y
VECT_Q31TOF	Conversion of signed Q31 to float32	SEQ, ACC	N	DP	Y
VECT_FTOQ31	Conversion of float32 to signed Q31	SEQ, ACC	N	DP	Y
VECT_FTOQ15	Convert float32 to Q15 format	SEQ, ACC	N	DP	Y
VECT_Q15TOF	Convert Q15 to float32 format	SEQ, ACC	N	DP	Y
VECT_LN_F32	Element-related natural logarithm	SEQ, ACC	N	DP	Y
VECT_LOG10_F32	Element-related logarithm base 10	SEQ, ACC	N	DP	Y
VECT_MAX_F32	Search for the value and position of the biggest element	SEQ, ACC	N	DP	Y
VECT_MIN_F32	Search for the value and position of the smallest element	SEQ, ACC	N	DP	Y
VECT_MUL_F32	Element-related product	SEQ, ACC	N	DP	Y
VECT_MULCOS_F32	Element-related product with a cosine	SEQ, ACC	N	DP	Y
VECT_MULSIN_F32	Element-related product with a sinus	SEQ, ACC	N	DP	Y
VECT_OFFSET_F32	Add an offset to each element	SEQ, ACC	Y	DP	Y
VECT_RMS_F32	Add an immediate offset to each element	SEQ, ACC	N	DP	Y
VECT_SIN_F32	Element-related sinus	SEQ, ACC	N	DP	Y
VECT_SINCOS_F32	Element-related sinus and cosine	SEQ, ACC	N	DP	Y
VECT_SCALE_F32	Multiply each element by a value	SEQ, ACC	Y	DP	Y
VECT_SET	Sets a value to each element	SEQ, ACC	Y	DP	Y
VECT_SQRT_F32	Element-related square-root	SEQ, ACC	N	DP	Y
VECT_SUB_F32	Element-related subtraction	SEQ, ACC	N	DP	Y
VECT_UTOF	Conversion of unsigned 32-bit integer to float32	SEQ, ACC	N	DP	Y
VECT_ZINS	Performs the interpolation by inserting zeros	SEQ, ACC	N	DP	Y
Functions operating on complex vectors					
CMPLX_CONJ_F32	Element-related conjugate	SEQ, ACC	N	DP	Y

Processing functions mnemonic	Description	Type	STREAM	BRAM	GMP
CMPLX_DOTP_F32	Dot product	SEQ, ACC	N	DP	Y
CMPLX_MAG_F32	Magnitude	SEQ, ACC	N	DP	Y
CMPLX_MAGSQR_F32	Magnitude squared	SEQ, ACC	N	DP	Y
CMPLX_MUL_F32	Element-related product	SEQ, ACC	N	DP	Y
CMPLX_MULEXP_F32	Element-related product with an exponential complex	SEQ, ACC	N	DP	Y
CMPLX_RMUL_F32	Element-related product of a complex with a real vector	SEQ, ACC	N	DP	Y
Scalar functions on real					
__SCA_ABS_F32	In-line absolute value	SEQ	Y	DP	-
__SCA_ADD_F32	In-line addition	SEQ	Y	DP	-
SCA_ATAN2_F32	Arc tangent 2	SEQ	Y	DP	Y
SCA_COS_F32	Cosine	SEQ	Y	DP	Y
__SCA_DIV_F32	In-line division	SEQ	Y	DP	-
SCA_EXP_F32	Exponential	SEQ	Y	DP	Y
SCA_EXP10_F32	10x	SEQ	Y	DP	Y
SCA_FATAN2_F32	Low-precision Arc tangent 2	SEQ	Y	DP	Y
SCA_FCOS_F32	Low-precision cosine	SEQ	Y	DP	Y
SCA_FSIN_F32	Low-precision sinus	SEQ	Y	DP	Y
__SCA_FTOI	In-line float32 to signed 32-bit integer conversion	SEQ	N	DP	Y
SCA_FTOQ31	Float32 to Q31 conversion	SEQ	N	DP	Y
__SCA_FTOU	In-line float32 to unsigned 32-bit integer conversion	SEQ	N	DP	Y
__SCA_I24TOF	In-line signed 24-bit integer to float32 conversion	SEQ	N	DP	Y
__SCA_ITOF	In-line signed 32-bit integer to float32 conversion	SEQ	N	DP	Y
SCA_Q31TOF	Conversion of a Q31 to float32	SEQ	N	DP	Y
SCA_LN_F32	Natural logarithm	SEQ	Y	DP	Y
SCA_LOG10_F32	Logarithm base 10	SEQ	Y	DP	Y
__SCA_MAC_F32	In-line multiply-accumulate	SEQ	Y	DP	Y
__SCA_MUL_F32	In-line product	SEQ	Y	DP	Y
__SCA_NEG_F32	In-line negate	SEQ	Y	DP	Y
__SCA_SAT_F32	In-line saturation function	SEQ	Y	DP	Y
SCA_SIN_F32	Sinus	SEQ	Y	DP	Y

Processing functions mnemonic	Description	Type	STREAM	BRAM	GMP
SCA_SINCOS_F32	Sinus and cosine	SEQ	Y	DP	Y
__SCA_SET	In-line sets a value	SEQ	Y	DP	Y
__SCA_SQRT_F32	In-line square-root	SEQ	Y	DP	Y
__SCA_SUB_F32	In-line subtraction	SEQ	Y	DP	Y
SCA_TOVECT	Builds a vector with consecutive scalar values.	SEQ	Y	DP	Y
__SCA_UTOF	In-line unsigned 32-bit integer into float32 conversion	SEQ	Y	DP	Y
Digital control functions					
DC_CMPCNT_F32	Performs a comparison to a threshold, with a counter	SEQ	N	DP	Y
DC_CLARKE1_2P_F32	Clarke transform	SEQ	-	DP	Y
DC_PARK1_F32	Park transform	SEQ	Y	DP	Y
DC_ICLARKE1_2P_F32	Inverse Clarke transform	SEQ	N	DP	Y
DC_IPARK1_F32	Inverse Park transform	SEQ	Y	DP	Y
DC_IIRDF1_3P3Z_F32	3nd order IIR filter with saturation and gain adjust	SEQ	Y	DP	Y
DC_IIRDF1_2P2Z_F32	2nd order IIR filter with saturation and gain adjust	SEQ	Y	DP	Y
Scalar functions working on 32-bit integer					
__SCA_ADD_I32	In-line addition	SEQ	Y	DP	-
__SCA_INC_U32	In-line increment	SEQ	Y	DP	-
__SCA_MUL_I32	In-line multiplication	SEQ	Y	DP	-
SCA_MADD_I32	Addition with modulus	SEQ	Y	DP	-
__SCA_INC_U32	In-line subtraction	SEQ	Y	DP	-
__SCA_SHIFT_I32	In-line left or right shift	SEQ	Y	DP	-
__SCA_AND_U32	In-line bitwise logical AND	SEQ	Y	DP	-
__SCA_OR_U32	In-line bitwise logical OR	SEQ	Y	DP	-
__SCA_XOR_U32	In-line Bitwise logical exclusive OR	SEQ	Y	DP	-
__SCA_NOT_U32	In-line bitwise logical NOT	SEQ	Y	DP	-
Matrix functions on real					
MAT_ABS_F32	Compute the absolute value of each element. Same as VECT_ABS_F32.	SEQ, ACC	N	DP	Y
MAT_ADD_F32	Performs the addition of two matrix. Same as VECT_ADD_F32.	SEQ, ACC	N	DP	Y

Processing functions mnemonic	Description	Type	STREAM	BRAM	GMP
MAT_INV_F32	Compute the inverse of a matrix.	SEQ, ACC	N	DP	Y
MAT_SUB_F32	Performs the subtraction of two matrix. Same as VECT_SUB_F32.	SEQ, ACC	N	DP	Y
MAT_MUL_F32	Performs the product of two matrix.	SEQ, ACC	N	DP	Y
MAT_OFFSET_F32	Add an offset to each element of a matrix. Same as VECT_OFFSET_F32.	SEQ, ACC	Y	DP	Y
MAT_SCALE_F32	Multiply each element of a matrix by a scalar. Same as VECT_SCALE_F32.	SEQ, ACC	Y	DP	Y
MAT_TRANS_F32	Performs the transpose of a matrix.	SEQ, ACC	N	DP	Y
MAT_VECT2COL	Copy a vector to a matrix column	SEQ	N	DP	Y
MAT_COL2VECT	Copy a matrix column into a vector	SEQ	N	DP	Y
Transform functions					
FFT_F32	Complex or Real FFT and IFFT transform	SEQ	N	DP	-
RFFT_F32	Real FFT transform	ACC	N	DP	-
CFFT_F32	Complex FFT transform	ACC	N	DP	-
DCT_F32	Discrete Cosine Transform type II	SEQ, ACC	N	DP	-
IDCT_F32	Discrete Cosine Transform type III	SEQ, ACC	N	DP	-
Conditional functions					
__IF_ELSE	In-line IF-THEN-ELSE functions	SEQ	N	DP	-
__LOOP	In-line Loop function	SEQ	N	DP	-
__IF_LOOP	In-line IF-LOOP functions	SEQ	N	DP	-
__IF_COUNT	In-line IF for COUNT counters	SEQ	N	-	-
COUNT	General purpose counter	SEQ	N	-	-
SET_COUNT	Set counters to a specific value	ACC	N	-	-
__CLR_SATF	In-line function, clears the saturation status	SEQ	N	-	-
__GET_SATF	In-line function, reads the saturation status	SEQ	N	-	-
CNN functions					
CNN_CONV2D_I8	2D convolution	ACC	N	SP	-
CNN_CONVDW_I8	Depth-wise convolution	ACC	N	SP	-

Processing functions mnemonic	Description	Type	STREAM	BRAM	GMP
CNN_CONVPW_I8	Point-wise convolution	ACC	N	SP	-
CNN_FC_I8	Fully connected layer	ACC	N	SP	-
CNN_POOL_I8	Pooling layer	ACC	N	SP	-
Specific functions					
MULWIN_F32	element related product with window shape	SEQ	N	DP	-
COMP_F32	Compare a vector or scalar to 2 thresholds.	SEQ	N	DP	-
SAT_F32	Saturation using programmable thresholds.	SEQ	N	DP	Y
SEND_EVENT	Triggers an event to the SPE	SEQ	N	-	-
SET_FLAG	Set flags to generate interrupt to the CPU	SEQ	N	-	-
CLR_FWERR	Read and clear firmware error status	ACC	N	-	-
SET_TRGO	Set TRGO signals	SEQ	N	-	-
SET_GPO	Set GPO signals	SEQ	N	-	-
SET_BITS	Forces a specific field to a value	SEQ	Y	DP	Y
CHK_BITS	Checks values of a bit field	SEQ	Y	DP	Y
WAIT_COND	Loops until condition is met	SEQ	Y	DP	Y
CRC32	Compute CRC	SEQ, ACC	-	DP	Y

5 Processing functions description

This section describes the command format to be used to program a processing function into a processing list or to execute directly a processing function.

Fields written in uppercase represent immediate values. Fields mixing the upper and lower cases generally represent pointers.

The following naming convention is adopted for data buffers: If Out, InA, InB represent data buffers, pOut, pInA, pInB represents pointers on data buffers Out, InA, InB.

In the tables describing the command format of each processing function, the row titled ‘Type’ gives an information on the parameter type:

- I8 stands for an 8-bit signed integer
- U(I)32 stands for a 32-bit unsigned (signed) integer
- U(I)32* stands for a pointer on 32-bit unsigned (signed) integer (scalar or vector)
- F32 stands for a 32-bit floating-point number
- F32* stands for a pointer on 32-bit floating-point (scalar or vector)
- X32 stands for a 32-bit number, float or integer
- X32* stands for a pointer on 32-bit number, float or integer
- ST* stands for a pointer on data structure, used when the data buffer contains data having different types

Those tables also give a row titled ‘Target’, defining the scope of each pointer:

- ABMP stands for a pointer on BRAM-AB memory
- IBP stands for pointer on HSP interface buffers (ADCIN or STREAM buffer)

Note: Each pointer must be translated into pointers usable by SPE. Refer to Section 2.2.4: Buffer pointer transmission on processing lists and Section 2.2.5: Buffer pointer transmission on direct commands for details.

The programing of processing functions into processing lists are using the HSP_PARAMRx registers and the MSGB block. Refer to Section 7.1: Start processing list recording for additional information.

The direct execution of processing functions are using HSP_PARAMRx registers and the DCMD block.

5.1 Command status

All the commands are returning a command status. This command status indicates if the commands have been properly executed or not.

Command status is always returned into HSP_H2CMSGDR and HSP_FWERR registers.

There are at least two methods to record the processing lists and check that commands are properly accepted by the HSP.

- After every command, check the status returned by the HSP. It shall always be equal to CMD_STATUS_OK.
- To simplify the operation, an alternative to the methods above is to clear the register HSP_FWERR using the command FWERR_CLR, then record all the processing lists. When the recording is complete, check the HSP_FWERR register. It contains the last error detected. If it is not equal to CMD_STATUS_OK an error occurred during processing list recording.

Table 6. Command status

Direct	Value	Status class
CMD_STATUS_OK	0x0000	-
User command completed with success		
CMD_STATUS_ERROR_CMDL_MEM_ALLOC	0x0001	Error
Failed to allocate memory for a processing list. This happens when the CRAM does not have enough room to record a new processing function into a processing list. The processing start address is removed from the TSKLUT. It is no longer possible to record any new processing list, but processing lists already recorded are functional.		
CMD_STATUS_ERROR_KERNEL_MEM_ALLOC	0x0002	Error

Direct	Value	Status class
Failed to allocate memory for a processing list. This happens when the DRAM does not have enough room to record a new processing function into a processing list. The processing start address is removed from the TSKLUT. It is no longer possible to record any new processing list, but processing lists already recorded are functional.		
CMD_STATUS_ERROR_SUPER_CMD	0x0003	Minor Error
The supervisor received a command different from PL_REST and PROCFG_START. This error is not blocking the HSP goes back to WFE.		
CMD_STATUS_ERROR_INI_CMD	0x0004	Fatal Error
The HSP was waiting for FW_INIT command and receives a bad command. After this error the HSP sets BSTAT to 9 and loops in an infinite loop. The application must reset the HSP via the reset bit of the RCC. The application must not use the reset request feature.		
CMD_STATUS_ERROR_KERNEL_ID	0x0005	Error
Bad processing function ID		
CMD_STATUS_ERROR_EMPTY_PL	0x0006	Error
Empty processing list scheduled		
CMD_STATUS_ERROR_FATAL	0x0008	Fatal error
Did not succeed to allocate a processing list: supervisor/direct/error		
CMD_STATUS_ERROR_DIR_CMD	0x000A	Error
Bad command in direct command		
CMD_STATUS_ERROR_TSKOFF	0x000B	Error
The address offset of the processing list exceeds the maximum allowed value of the product size.		
CMD_STATUS_ERROR_MAT_INV_SINGULAR	0x000C	Error
Mat inverse: input matrix is singular (non-invertible)		
CMD_STATUS_ERROR_PLUGIN_D_ALLOC	0x000D	Error
Failed to allocated memory in DRAM to add plugin		
CMD_STATUS_ERROR_IFE_CMD	0x0010	Error
Bad command IF_ELSE		
CMD_STATUS_ERROR_IFE_MAX_IF	0x0011	Error
Maximum number of nested IF_ELSE is reached		
CMD_STATUS_ERROR_IFE_UNCLOSED_IF	0x0012	Error
Unclosed IF_ELSE.		
CMD_STATUS_ERROR_MAX_LOOP	0x0013	Error
Maximum nested loop reached		
CMD_STATUS_ERROR_LOOP_UNCLOSED	0x0014	Error
Unclosed loop		
CMD_STATUS_ERROR_NO_LOOP	0x0015	Error
Not in /loop/ statement		
CMD_STATUS_ERROR_PLUGIN_INST	0x0016	Error
Bad instruction in plugin		
CMD_STATUS_ERROR_CNN_CONV2D	0x0017	Error
Bad CNN conv2d parameters		

5.2 Configuration functions

5.2.1 FW_INIT

This command must be used during the HSP initialization phase. After BOOTEN is set to 1, the HSP informs that it is waiting for the FW_INIT command by setting H2CSEM to 1.

The application must set H2CSEM to 0 and sent the FW_INIT command respecting the format below. Refer to Section 5.2.1: FW_INIT for additional information.

Table 7. FW_INIT configuration functions

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	FW_INIT	U32	-	See Section 8: Function ID list
HSP_PARAMR0	SOC_BRAMAB_START	U32	-	CPU base address of the BRAM-AB
HSP_PARAMR1	PERFEN	U32	-	Activation of the performance counter
HSP_PARAMR[15:2]	-	-	-	Not used

1. *Refer to Section 5: Processing functions description for details.*

SOC_BRAMAB_START	The HSP needs to have the BRAM-AB base address see by the CPU (in byte) for pointer translation. The BRAM-AB base address may differ from a product to another.
PERFEN	Activation of the cycle counter 0: does not count the number of cycles 1: counts the number of cycles

The cycle counter counts the execution duration of each processing list. It is updated after the completion of each processing list. The duration value is given in HSP core clock cycles (hsp_xclk_core_ck). The cycle counter value can be read in the DRAM, at the following address:

SOC_HSP_BASE_ADD + CRAM_SIZE * 1024 + 8

Note: The application cannot read this cycle counter if CDRLCK[1:0] is set to 3.

For command status, refer to Section 5.1: Command status for details.

5.2.2 PROCFG_START

This command must be used when the supervisor is activated in order to start the recording of a processing list. Each command contains a unique processing list ID (PLID). This PLID must be in line with event controller configuration.

Each processing list must be encapsulated by PROCFG_START and PROCFG_END commands.

Table 8. PROCFG_START processing list ID

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	PROCFG_START	U32	-	See Section 8: Function ID list
HSP_PARAMR0	PLID	U32	-	Processing list number
HSP_PARAMR[15:1]	-	-	-	-

1. *Refer to Section 5: Processing functions description for details.*

PLID	Processing list ID: The processing list number must be an integer between 0 and 36. The event number given by **tsk_active**[5:0] (from EVTC) activates the execution of the corresponding processing list.

Command status: see Section 5.1: Command status for details.

5.2.3 PROCFG_END

This command must be used to indicate the end of a processing list programming.

Each processing list must be encapsulated by PROCFG_START and PROCFG_END commands.

Table 9. PROCFG_END processing list ID

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	PROCFG_END	U32	-	See Section 8: Function ID list
HSP_PARAMR[15:0]	-	-	-	-

1. *Refer to Section 5: Processing functions description for details.*

Command status: see Section 5.1: Command status for details.

5.2.4 PL_RESET

This command must be used to erase all the programmed processing lists.

Table 10. PL_RESET processing list ID

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	PL_RESET	U32	-	See Section 8: Function ID list
HSP_PARAMR0	-	-	-	-
HSP_PARAMR1	PERFEN	U32	-	Activation of the performance counter
HSP_PARAMR[15:2]	-	-	-	-

1. *Refer to Section 5: Processing functions description for details.*

PERFEN	Activation of the cycle counter 0: does not count the number of cycles 1: counts the number of cycles See additional information on Section 5.2.1: FW_INIT.

Command status: see Section 5.1: Command status for details.

5.3 Filter functions

The HSP offers several filtering processing functions performing filtering on real data.

The function ID of each filter function is provided on Section 8: Function ID list.

The data structure of each filter function is similar. As shown in the next figure, each filter instance has its own filter state (FltState), its own filter state buffer (FltStateBuf), and its coefficient table (FltCoef).

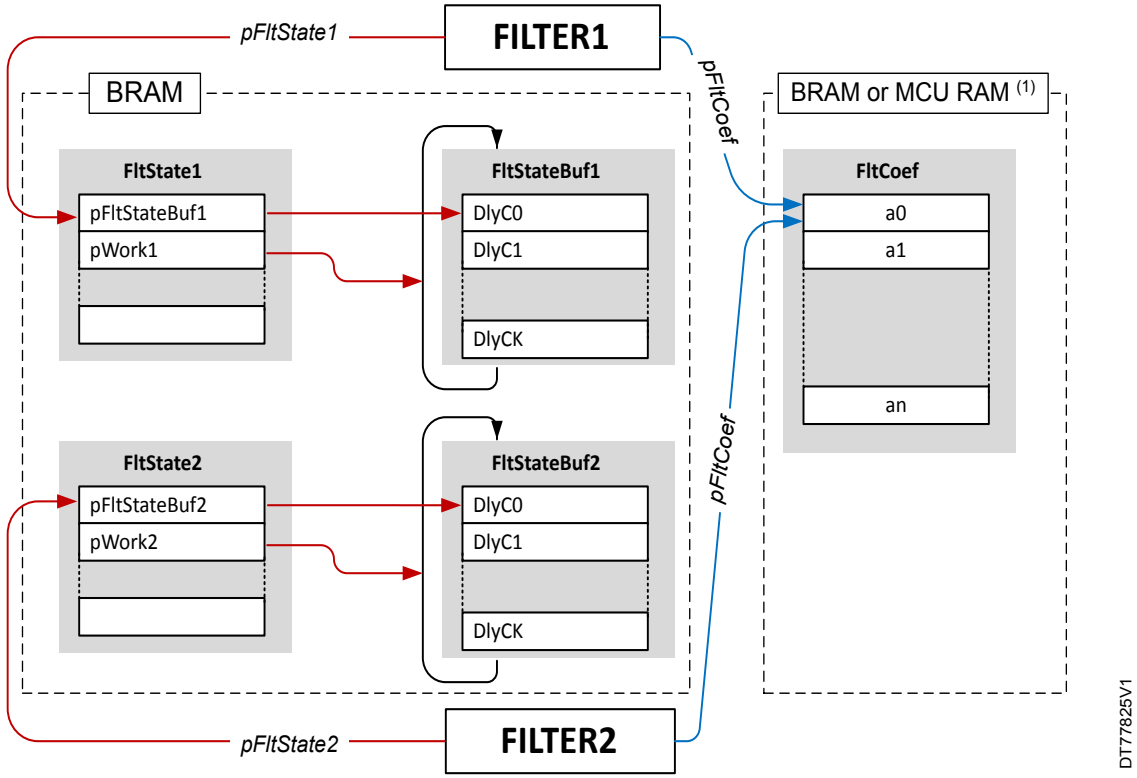
The filter state and filter state buffers must be located into the BRAM-AB. The filter coefficients can be located in the BRAM-AB (more efficient), but also on system RAM.

Filter performances are degraded when the coefficient table is not located into the BRAM-AB.

The FltState and FltStateBuf can be contiguous or not.

The example below shows a case where FILTER1 and FILTER2 are sharing the same coefficient table.

Figure 8. **Filters data structure**



1. *Using MCU RAM is only possible when the HSP embeds the HSPDMA and STI.*

5.3.1 Tap definition in digital filters

This document often refers to the filter taps. For filters based on FIR structure, each tap corresponds to a coefficient. For filters based on IIR structure, taps refer to the coefficients of both the feedforward (numerator) and feedback (denominator) parts of the filter's difference equation.

The table below shows the footprint of the coefficient buffer and filter states.

Table 11. **Filter state and coefficient footprint**

NTAPS represents the number of delay blocks of the filter.

Processing functions	Buffer size to be instantiated by the application (bytes)	
	Coefficient (SoC RAM or BRAM)	FltState + FtlStateBuf. (BRAM)
FIR_F32	4 * (NCOEF)	12 + 4 * (NCOEF-1)
FIRDEC_F32	4 * (NCOEF+1)	20 + 4 * NCOEF
FIRLMS_F32	4 * (NCOEF+1)	12 + 4 * NCOEF
BIQDF1_CASC_F32	20 * NBIQ	12 + 16 * NBIQ
BIQDF2T_CASC_F32	20 * NBIQ	12 + 8 * NBIQ
IIRDF1_F32	4 * (NTAPS+1)	12 + 4 * NTAPS
IIR_LATTICE_F32	8* NTAPS	16 * NTAPS
IIR_3P3Z_F32	48	12 + 24
DC_IIRDF1_2P2Z_F32	40	12 + 16

1. *The BRAM is a dual-bank RAM, so coefficients located into the BRAM are written into both banks. The CPU just sees one bank but the raw footprint is 2 times the footprint seen by the CPU.*

2. *The BRAM is a dual-bank RAM, so states are written into both banks. The CPU just sees one bank but the raw footprint is 2 times the footprint seen by the CPU.*

5.3.2 FIR_F32

The FIR_F32 function can be either executed as a direct command (accelerator mode) or programmed into a processing list.

The FIR_F32 performs an optimal signal filtering using finite impulse response filter structure.

For a FIR_F32 filter of order k, each value of the output sequence is a weighted sum of the most recent input values, as shown in the following expression:

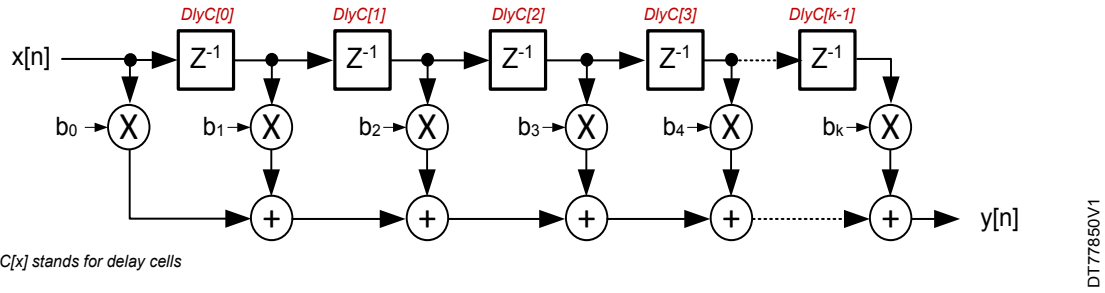
$$y[n] = b_0 \cdot x[n] + b_1 \cdot x[n-1] + \ldots + b_k \cdot x[n-k] = \sum_{i=0}^{k} b_i \cdot x[n-i]$$

Where:

- x[n] are the input samples
- y[n] are the output samples
- **k** is the filter order. A k^{th} FIR_F32 order has **k**+1 coefficients ($\mathbf{b_i}$)

The figure below shows a graphical representation of the FIR_F32 structure:

Figure 9. FIR_F32 structure



DlyC[x] stands for delay cells

For optimization reasons, the HSP embeds several FIR_F32 processing functions:

- FIR_LESS8C_F32 is used when the filter has less than eight coefficients.
- FIR_MOD4C_F32 is used when the number of coefficients is a multiple of 4.
- FIR_1S_F32 is used when only one sample is processed.
- FIR_F32 is used in all other cases.

When the application records a FIR_F32 filter into a processing list, the sequence builder embedded in the HSP selects the appropriate filter function. The sequence builder replaces the function ID of FIR_F32 with the optimal function.

When the application wants to directly execute a FIR function, it must select the appropriate FIR_F32 function as explained above. When using middleware functions or HSP-CMSIS-DSP-like functions, they automatically select the optimal filter.

In both situations, to get the best filter performance, it is strongly recommended to use FIR filters having an number of coefficients multiple of 4.

5.3.2.1 *Command format to record FIR_F32*

The table below gives the FIR_F32 command format when it is recorded into a processing list.

Table 12. FIR_F32 command format

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	FIR_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, IBP, POP	Pointer on input data buffer

Registers	Acronyms	Type	Target	Description
HSP_PARAMR1	pFltCoef	F32*	ABMP, POP	Pointer on filter coefficient buffer
HSP_PARAMR2	pOut	F32*	ABMP, IBP, POP	Pointer on output data buffer
HSP_PARAMR3	pFltState	ST*	ABMP	Pointer on filter state buffer
HSP_PARAMR4	N	U32	-	Number of elements to process
HSP_PARAMR[14:5]	-	-	-	-
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data buffer The input buffer must contain a real vector in float32. The input and output buffer pointers can be the same. The buffer size is limited to 4096 elements
pFltCoef	Pointer on filter coefficient buffer Must contain real values in float32. See format in Table 14. pFltCoef format for FIR_F32, FIRDEC and FIRLMS_F32.
pOut	Pointer on output data buffer Filtered samples. The input and output buffer pointers can be the same.
pFltState	Address of the filter state buffer See Table 15. FltState structure format for FIR_F32, FIRLMS_F32 for details.
N	Number of elements to process Can be any integer between 1 and 4096.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is an ABMP or IBP pointer depending on its value. pFltCoef target: If POP1 = 1, it is a pointer-on-pointer otherwise it is an ABMP pointer. pOut target: If POP2 = 1, it is a pointer-on-pointer otherwise it is an ABMP or IBP pointer depending on its value. Refer to Section 2.2.6: IOTYPE description for details

5.3.2.2 *Command format to directly execute FIR_F32*

The next table shows the command format when the FIR_F32 function is executed as a direct command.

Table 13. FIR_F32 function

Order	Registers	Acronyms	Type	Target	Description
1	HSP_DCMDIDR	FIR_F32 FIR_LESS8C_F32 FIR_MOD4C_F32 FIR_1S_F32	U32	-	See Section 8: Function ID list See also Section 5.3.2.5: Selection of the FIR function
2	HSP_PARAMR3	pFltState	ST*	ABMP	See description above
3	HSP_PARAMR4	N	U32	-	
4	HSP_DCMDPTR0	pIn	F32*	ABMP	
5	HSP_DCMDPTR1	pFltCoef	F32*	ABMP	
6	HSP_DCMDPTR2	pOut	F32*	ABMP	

1. *Parameters must be written in the specified order.*
2. *Refer to Section 5: Processing functions description for acronym details.*

Command status: see Section 5.1: Command status for details.

5.3.2.3 *FIR_F32 coefficient format*

Filter coefficients must be stored into the buffer pointed by pFltCoef, as shown below:

Table 14. pFltCoef format for FIR_F32, FIRDEC and FIRLMS_F32

CPU Address offset	Acronyms	Description	Data type
0	b[0]	filter coefficient	F32
4	b[1]	filter coefficient	F32
...	...	filter coefficient	F32
4*k	b[k]	filter coefficient	F32

1. *U8 stands for 8-bit unsigned integer, U16 stands for 16-bit unsigned integer, U32 stands for 32-bit unsigned integer, and F32 stands for 32-bit floating-point.*

5.3.2.4 *Filter states*

Each filter instance has its own filter state containing the filter taps and some variables giving the state of the filter. The application must instantiate a buffer in BRAM-AB containing the filter state for each filter instance.

Table 15. FltState structure format for FIR_F32, FIRLMS_F32

CPU Address offset	Field name	Type	Buff Type	Description
0	pFltStateBuf	F32*	ABMP	Filter State buffer start address
4	pWork	F32*	ABMP	Working pointer on next State buffer entry
8	NCOEF	U32	-	Number of filter coefficients

The next table shows the format of the filter state table.

Table 16. FltStateBuf for FIR_F32, FIRLMS_F32 and FIRDEC

CPU Address offset	Field name	Type	Buff Type	Description
0	DlyC[0]	F32	ABMP	Filter state data
4	DlyC[1]	F32	ABMP	Filter state data
...	...	...	...	...
4*(k-1)	DlyC[k-1]	F32	ABMP	Filter state data

Refer to Table 11. Filter state and coefficient footprint for details on memory footprint.

The created FltState structure must be initialized before a filter uses it.

- pStateBuf and pWork floating-pointers must point to the FltStateBuf[0] element. Those pointers are for SPE internal usage and must be translated into pointers directly usable by the SPE as explained on Section 2.2.4: Buffer pointer transmission on processing lists even if filter is used in a direct command.
- NCOEF must contain the number of coefficients.
- FltStateBuf[x] must be initialized to 0.

5.3.2.5 *Selection of the FIR function*

The HSP offers several FIR filter functions, each optimized according to the number of samples (N), and the number of coefficients (NCOEF). When the FIR is used as a direct command, the application must select the appropriate FIR function. The pseudocode below shows how to select the optimal FIR filter function:

```
if (N == 1)          {
    dirCmd = FIR_1S_F32;
}
else     {
    if (NCOEF < 7) {
        dirCmd = FIR_LESS8C_F32;
    }
    else {
        if (((NCOEF & 1) != 1) && ((NCOEF & 3) == 0) && ((N & 1) == 0) && (N > 4)) {
            dirCmd = FIR_MOD4C_F32;
        }
        else     {
            dirCmd = FIR_F32;
        }
    }
}
```

Note: When using middleware functions or HSP-CMSIS-DSP-like functions, they automatically select the optimal FIR_F32 filter.

5.3.3 FIRDEC_F32

The FIRDEC_F32 can be either executed as a direct command (accelerator mode) or programmed into a processing list.

This function performs a filtering based on a FIR filter, followed by a decimation. FIRDEC_F32 is used to reduce the sample rate of a signal without introducing aliasing distortion.

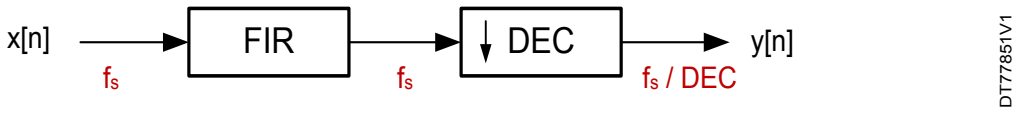
Refer to Section 5.3.2: FIR_F32 for information about FIR filters.

The FIRDEC_F32 function combines the FIR filter and the decimation in an efficient manner. Instead of calculating all of the FIR filter outputs and discarding DEC-1 samples output of every DEC, only the samples output by the decimation are computed. The function operates on blocks of N input samples, and generates N/DEC output samples.

N must be a multiple of DEC.

The figure below shows the FIRDEC_F32 function:

Figure 10. FIRDEC_F32 function



Note: The decimation ratio (DEC) is not part of the command, but is provided by the filter state buffer.

For optimization reasons, the HSP embeds several FIRDEC_F32 processing functions.

When the application is recording a FIRDEC_F32 filter into a processing list, the sequence builder selects the appropriate FIRDEC_F32 function. So the function ID of FIRDEC_F32 must be used; the sequence builder replaces it by the optimal function.

5.3.3.1 Command format to record FIRDEC_F32

Table 17. Command format to record FIRDEC_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	FIRDEC_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pIn	F32*	ABMP, IBP, POP	Pointer on input data buffer
HSP_PARAMR1	pFltCoef	F32*	ABMP, POP	Pointer on filter coefficient buffer
HSP_PARAMR2	pOut	F32*	ABMP, IBP, POP	Pointer on output data buffer
HSP_PARAMR3	pFltState	ST*	ABMP	Pointer on filter state buffer
HSP_PARAMR4	N	U32	-	Number of output samples

Registers	Acronyms	Type	Target	Description
HSP_PARAMR[14:5]	-	-	-	-
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pIn	Pointer on input data buffer The input buffer must contain a real vector in float32. The input and output buffer pointers can be the same. The buffer size must be at least DEC x (N-1) +1. DEC represents the decimation ratio.
pFltCoef	Pointer on filter coefficient buffer Must contain real values in float32. See format in Table 14. pFltCoef format for FIR_F32, FIRDEC and FIRLMS_F32.
pOut	Pointer on output data buffer Filtered samples. The input and output buffer pointers can be the same.
pFltState	Address of the filter state buffer See Table 19. FltState structure format for FIRDEC_F32 for details.
N	Number of elements to process Can be any integer between 1 and 4096.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is an ABMP or IBP pointer depending on its value. pFltCoef target: If POP1 = 1, it is a pointer-on-pointer otherwise it is an ABMP pointer. pOut target: If POP2 = 1, it is a pointer-on-pointer otherwise it is an ABMP or IBP pointer depending on its value. Refer to Section 2.2.6: IOTYPE description for details

5.3.3.2 *FIRDEC_F32 direct command format*

The next table shows the command format when FIRDEC_F32 is executed as a direct command.

Table 18. FIRDEC_F32 direct command format

Order	Registers	Acronyms	Type	Target	Description
1	HSP_DCMDIDR	FIRDEC_F32 FIRDEC_1C_F32 FIRDEC_SP_F32	U32	-	See Function ID list See Selection of FIRDEC function ID
2	HSP_PARAMR3	pFltState	ST*	ABMP	See description above
3	HSP_PARAMR4	N	U32	-	
4	HSP_DCMDPTR0	pIn	F32*	ABMP	
5	HSP_DCMDPTR1	pFltCoef	F32*	ABMP	
6	HSP_DCMDPTR2	pOut	F32*	ABMP	

1. *Parameters must be written in the specified order.*
2. *Refer to Section 5: Processing functions description for acronym details*

Command status: see Section 5.1: Command status for details.

5.3.3.3 *Filter states*

Each filter instance has its own filter state containing the filter taps and some variables giving the state of the filter. The application must instantiate a buffer in BRAM containing the filter state for each filter instance.

Table 19. FltState structure format for FIRDEC_F32

CPU Address offset	Field name	Type	Description
0	pFltStateBuf	F32*	Filter State buffer start address
4	pWork	F32*	Working pointer on next State buffer entry
8	NCOEF	U32	Number of filter coefficients
12	FirstLoop	U32	Internal variable
16	SecondLoop	U32	Internal variable

The filter state buffer (FltStateBuf) is described in Table 16. FltStateBuf for FIR_F32, FIRLMS_F32 and FIRDEC.

The created FltState structure must be initialized before a filter uses it.

- pStateBuf and pWork pointers must point to the StateBuf[0] element. Those pointers are for SPE internal usage and must be translated into pointers directly usable by the SPE as explained on Section 2.2.4: Buffer pointer transmission on processing lists even if the filter is used in a direct command.
- NCOEF must contain the number of coefficients.
- StateBuf[x] must be initialized to 0.
- FirstLoop and SecondLoop are deduced from the decimation ratio (DEC), and must be computed as explained below.

5.3.3.4 *Selection of FIRDEC function ID*

When an application directly executes the FIRDEC_F32 function, it must select the appropriate FIRDEC_F32 function as follows:

- If NCOEF = 1, then function FIRDEC_1C_F32 must be selected. FirstLoop and SecondLoop are not used.
- If NCOEF is greater than or equal to N, the function FIRDEC_SP_F32 must be selected. FirstLoop and SecondLoop are not used.
- If NCOEF is less than N, then FirstLoop and SecondLoop must be computed and according to the result, the application must select FIRDEC_SP_F32 or FIRDEC_F32.

See details below.

```
uint32_t q;
uint32_t r;
uint32_t FirstLoop;
uint32_t SecondLoop;
uint32_t CmdID;

if (NCOEFF == 1 {
    CmdID = FIRDEC_1C_F32;
}
else {
    if (NCOEFF >= N) {
        CmdID = FIRDEC_SP_F32;
    }
    else {
        q = NCOEF/DEC;
        r = NCOEF - (q*DEC);
        if (r > 1) {
            FirstLoop = q + 1;
        }
        else {
            FirstLoop = q;
        }
        SecondLoop = (N / DEC) - FirstLoop;
        if (SecondLoop < 1) {
            CmdID = FIRDEC_SP_F32;
        }
```

```
        else {
            CmdID = FIRDEC_F32;
        }
    }
}
```

Note: When using middleware functions or HSP-CMSIS-DSP-like functions, they automatically select the optimal FIRDEC_F32 filter.

5.3.4 FIRLMS_F32

The FIRLMS_F32 can be either executed as a direct command (accelerator mode) or programmed into a processing list.

This function performs the least mean square (LMS) filtering function.

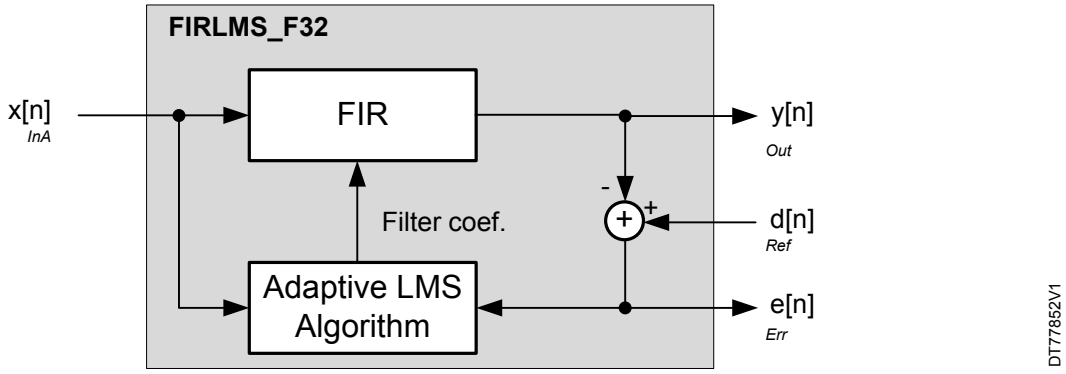
LMS filter self-adjusts the filter transfer function according to an adaptive algorithm.

LMS filters use a gradient descent method in which the filter coefficients are updated based on the instantaneous error signal. Adaptive filters are often used in communication systems, equalizers, and for noise removal.

An LMS filter consists of two components as shown below. The first component is a standard transversal or FIR filter. The second component is a coefficient update mechanism. The LMS filter has two input signals. The input (x[n]) feeds the FIR filter while the reference input (d[n]) corresponds to the desired output of the FIR filter. That is, the FIR filter coefficients are updated so that the output of the FIR filter matches the reference input. The filter coefficient update mechanism is based on the difference between the FIR filter output and the reference input. This error signal (e[n]) tends towards zero as the filter adapts. The LMS processing functions accept the input and reference input signals and generate the filter output and error signal.

The figure below shows the FIRLMS_F32 function:

Figure 11. FIRLMS_F32 function



Where:

- x[n] is the input signal
- y[n] is the corresponding filter output
- d[n] is the desired response
- e[n] is the error signal d[n] - y[n]

After each sample, the error signal is computed, the filter coefficients b[k] are updated on a sample-by-sample basis. This means that at each sample, b[k] are modified as follows:

b[k] = b[k] + e[n] * MU* x[n-k] for k=0, 1,..., NBTAPS-1

Where MU is the step size and controls the rate of coefficient convergence.

Each FIRLSM instance has an implicit internal buffer storing the state of the filter. The size of this buffer is NBTAPS+1 words of 32 bits.

5.3.4.1 *Command format to record FIRLMS_F32*

Table 20. Command format to record FIRLMS_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	FIRLMS_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pIn	F32*	ABMP, IBP, POP	Pointer on input data buffer
HSP_PARAMR1	pFltCoef	F32*	ABMP, POP	Pointer on filter coefficient buffer
HSP_PARAMR2	pOut	F32*	ABMP, IBP, POP	Pointer on output data buffer
HSP_PARAMR3	pFltState	ST*	ABMP	Pointer on filter state buffer
HSP_PARAMR4	pRef	F32*	ABMP	Pointer on reference input data buffer
HSP_PARAMR5	pError	F32*	ABMP	Pointer on error data buffer
HSP_PARAMR6	N	U32	-	Number of samples to precess
HSP_PARAMR7	MU	F32	-	Convergence rate
HSP_PARAMR[14:8]	-	-	-	-
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pIn	Pointer on input data buffer The input buffer must contain a real vector in float32. The input and output buffer pointers can be the same. The buffer size is limited to 4096 elements
pFltCoef	Pointer on filter coefficient buffer Must contain real values in float32. See format in Table 14. pFltCoef format for FIR_F32, FIRDEC and FIRLMS_F32.
pOut	Pointer on output data buffer Filtered samples. The input and output buffer pointers can be the same.
pFltState	Address of the filter state buffer See Table 16. FltStateBuf for FIR_F32, FIRLMS_F32 and FIRDEC for details.
pRef	Pointer on reference (desired) input data buffer Must contain a real vector in float32 representing the reference input samples.
pError	Pointer on error output buffer
N	Number of elements to process Can be any integer between 1 and 4096.
MU	Defines the rate of convergence It must be a float32
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is an ABMP or IBP pointer depending on its value. pFltCoef target: If POP1 = 1, it is a pointer-on-pointer otherwise it is an ABMP pointer. pOut target: If POP2 = 1, it is a pointer-on-pointer otherwise it is an ABMP or IBP pointer depending on its value. Refer to Section 2.2.6: IOTYPE description for details

5.3.4.2 *FIRLMS_F32 direct command format*

The next table shows the command format when FIRLMS_F32 is executed as a direct command.

Table 21. FIRLMS_F32 direct command format

Order	Registers	Acronyms	Type	Target	Description
1	HSP_DCMDIDR	FIRLMS_F32	U32	-	See Function ID list
2	HSP_PARAMR3	pFltState	F32*	ABMP	See description above
3	HSP_PARAMR4	pRef	F32*	ABMP	
4	HSP_PARAMR5	pError	F32*	ABMP	
5	HSP_PARAMR6	N	U32	-	
6	HSP_PARAMR7	MU	F32	-	
7	HSP_DCMDPTR0	pIn	F32*	ABMP	
8	HSP_DCMDPTR1	pFltCoef	ST*	ABMP	
9	HSP_DCMDPTR2	pOut	F32*	ABMP	

1. *Parameters must be written in the specified order.*
2. *Refer to Section 5: Processing functions description for acronym details*

Command status: see Section 5.1: Command status for details.

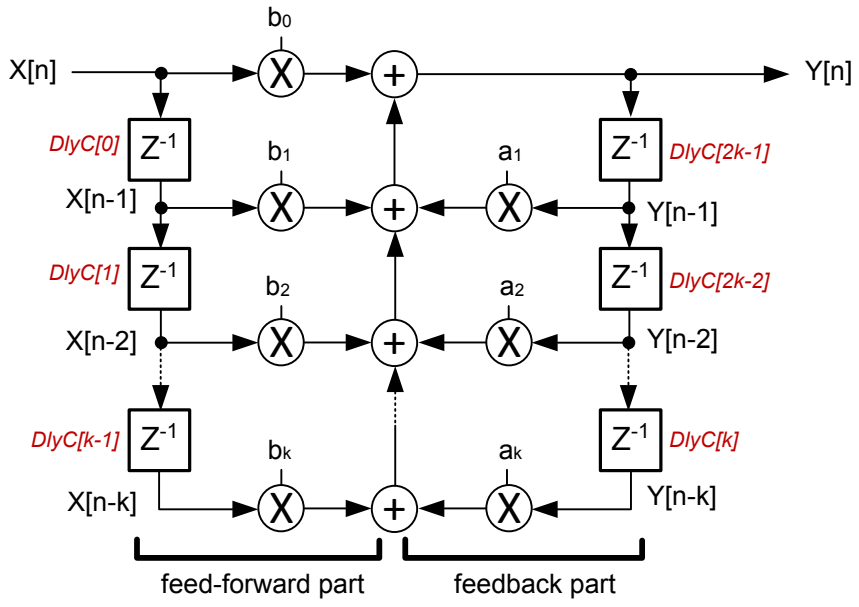
5.3.5 IIRDF1_F32

The IIRDF1_F32 can be either executed as a direct command (accelerator mode) or programmed into a processing list.

This function performs an optimal IIR filtering using the direct form 1.

The figure below shows the IIRDF1_F32 structure:

Figure 12. IIRDF1_F32 structure



DlyC[x] stands for delay cells

5.3.5.1 *Command format to record IIRDF1_F32*

Table 22. Command format to record IIRDF_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	IIRDF1_F32	U32	-	See Function ID list
HSP_PARAMR0	pIn	F32*	ABMP, IBP, POP	Pointer on input data buffer
HSP_PARAMR1	pFltCoef	F32*	ABMP, POP	Pointer on filter coefficient buffer
HSP_PARAMR2	pOut	F32*	ABMP, IBP, POP	Pointer on output data buffer
HSP_PARAMR3	pFltState	ST*	ABMP	Pointer on filter state buffer
HSP_PARAMR4	N	U32	-	Number of samples to precess
HSP_PARAMR[14:5]	-	-	-	-
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pIn	Pointer on input data buffer The input buffer must contain a real vector in float32. The input and output buffer pointers can be the same. The buffer size is limited to 4096 elements
pFltCoef	Pointer on filter coefficient buffer Must contain real values in float32. See format in Table 24. pFltCoef format for IIRDF1_F32.
pOut	Pointer on output data buffer Filtered samples. The input and output buffer pointers can be the same.
pFltState	Address of the filter state buffer See Table 25. FltState structure format for IIRDF1_F32 for details.
N	Number of elements to process Can be any integer between 1 and 4096.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is an ABMP or IBP pointer depending on its value. pFltCoef target: If POP1 = 1, it is a pointer-on-pointer otherwise it is an ABMP pointer. pOut target: If POP2 = 1, it is a pointer-on-pointer otherwise it is an ABMP or IBP pointer depending on its value. Refer to Section 2.2.6: IOTYPE description for details

5.3.5.2 *IIRDF1_F32 direct command format*

The next table shows the command format when IIRDF1_F32 is executed as a direct command.

Table 23. IIRDF_F32 direct command format

Order	Registers	Acronyms	Type	Target	Description
1	HSP_DCMDIDR	IIRDF1_F32	U32	-	See Section 8: Function ID list
2	HSP_PARAMR3	pFltState	ST*	ABMP	See description above
3	HSP_PARAMR4	N	U32	-	
4	HSP_DCMDPTR0	pIn	F32*	ABMP	

Order	Registers	Acronyms	Type	Target	Description
5	HSP_DCMDPTR1	pFltCoef	F32*	ABMP	See description above
6	HSP_DCMDPTR2	pOut	F32*	ABMP	

1. *Parameters must be written in the specified order.*
2. *Refer to Section 5: Processing functions description for acronym details*

Command status: see Section 5.1: Command status for details.

5.3.5.3 *IIRDF1_F32 coefficient format*

Filter coefficients must be stored into the buffer pointed by pFltCoef, as shown below:

Table 24. pFltCoef format for IIRDF1_F32

CPU Offset addr.	Acronyms	Description	Data type
0	b[k]	filter coefficient	F32
1	a[k]	filter coefficient	F32
2	b[k-1]	filter coefficient	F32
3	a[k-1]	filter coefficient	F32
...	...	filter coefficient	F32
2k-2	b[1]	filter coefficient	F32
2k-1	a[1]	filter coefficient	F32
2k	b[0]	filter coefficient	F32

1. *'k' represents the filter order, and is equal to NTAPS/2*

5.3.5.4 *Filter states*

Each filter instance has its own filter state containing the filter taps and some variables giving the state of the filter. The application must instantiate a buffer in BRAM containing the filter state for each filter instance.

Table 25. FltState structure format for IIRDF1_F32

CPU Address offset	Field name	Type	Description
0	pFltStateBuf	F32*	Filter State buffer start address
4	pWork	F32*	Working pointer on next State buffer entry
8	NDLYC	U32	Filter state size or number of delay cells. NDLYC is equal to 2 x filter order

The created FltState structure must be initialized before a filter uses it.

- pStateBuf and pWork pointers must point to the StateBuf[0] element. Those pointers are for SPE internal usage and must be translated into pointers directly usable by the SPE as explained on Section 2.2.4: Buffer pointer transmission on processing lists even if filter is used in a direct command.
- NDLYC must contain the filter order multiplied by 2.
- StateBuf[x] must be initialized to 0.

The next table shows the format of the filter state table.

Table 26. Filter state buffer for IIRDF1_F32

CPU Address offset	Field name	Type	Buff Type	Description
0	DlyC[0]	F32	ABMP	Filter state data
4	DlyC[1]	F32	ABMP	Filter state data

CPU Address offset	Field name	Type	Buff Type	Description
...	...	...	...	...
4*(NDLYC-1)	DlyC[NDLYC-1]	F32	ABMP	Filter state data

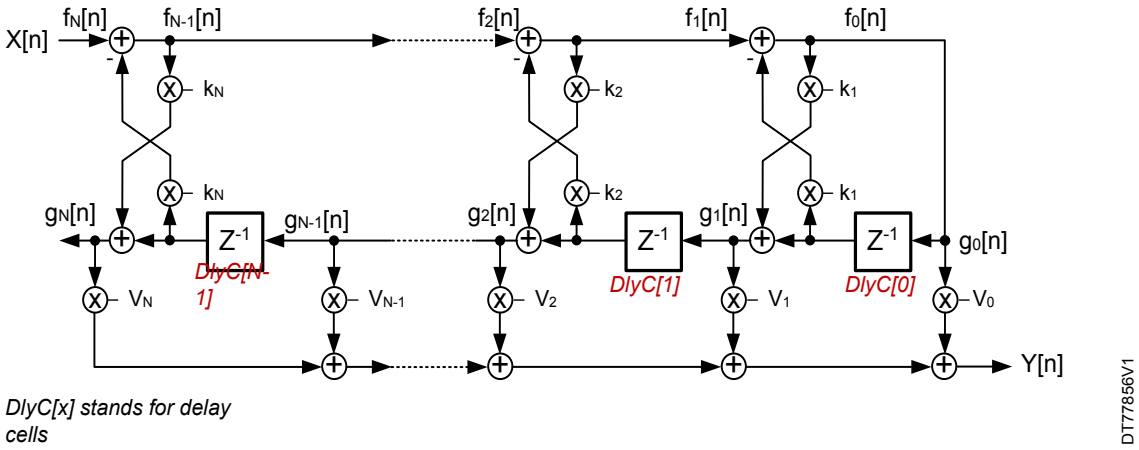
5.3.6 IIR_LATTICE_F32

The IIR_LATTICE_F32 can be either executed as a direct command (accelerator mode) or programmed into a processing list.

This function performs an IIR filtering using the Lattice structure.

The figure below shows the filter structure:

Figure 13. IIR_LATTICE_F32 structure



DlyC[x] stands for delay cells

5.3.6.1 *Command format to record IIR_LATTICE_F32*

Table 27. Command format to record IIR_LATTICE_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	IIR_LATTICE_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pIn	F32*	ABMP, IBP, POP	Pointer on input data buffer
HSP_PARAMR1	pFltCoefK	F32*	ABMP, POP	Pointer on reflexion filter coefficient buffer
HSP_PARAMR2	pOut	F32*	ABMP, IBP, POP	Pointer on output data buffer
HSP_PARAMR3	pFltState	ST*	ABMP	Pointer on filter state buffer
HSP_PARAMR4	N	U32	-	Number of samples to precess
HSP_PARAMR5	pFltCoefV (tbc)	F32*	ABMP	Pointer on ladder filter coefficient buffer
HSP_PARAMR[14:6]	-	-	-	-
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pIn	Pointer on input data The input buffer must contain a real scalar in float32. The input and output buffer pointers can be the same.
pFltCoefK	Pointer on reflexion filter coefficient buffer Must contain real values in float32. See format in Table 29. pFltCoef format for IIR_LATTICE_F32.

pOut	Pointer on output data Filtered samples. The input and output buffer pointers can be the same.
pFltState	Address of the filter state buffer See Table 31. Filter state buffer for IIR_LATTICE_F32 for details.
N	Number of elements to process Can be any integer between 1 and 4096.
pFltCoefV	Pointer on ladder filter coefficient buffer Must contain real values in float32. See format in Table 29. pFltCoef format for IIR_LATTICE_F32.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is an ABMP or IBP pointer depending on its value. pFltCoefK target: If POP1 = 1, it is a pointer-on-pointer otherwise it is an ABMP pointer. pOut target: If POP2 = 1, it is a pointer-on-pointer otherwise it is an ABMP or IBP pointer depending on its value. Refer to Section 2.2.6: IOTYPE description for details

5.3.6.2 *IIR_LATTICE_F32 direct command format*

The next table shows the command format when IIR_LATTICE_F32 is executed as a direct command.

Table 28. IIR_LATTICE_F32 direct command format

Order	Registers	Acronyms	Type	Target	Description
1	HSP_DCMDIDR	IIR_LATTICE_F32	U32	-	See Section 8: Function ID list
2	HSP_PARAMR3	pFltState	ST*	ABMP	See description above
3	HSP_PARAMR4	N	U32	-	
4	HSP_DCMDPTR0	pIn	F32*	ABMP	
5	HSP_DCMDPTR1	pFltCoef	F32*	ABMP	
6	HSP_DCMDPTR2	pOut	F32*	ABMP	

1. *Parameters must be written in the specified order.*
2. *Refer to Section 5: Processing functions description for acronym details*

Command status: see Section 5.1: Command status for details.

5.3.6.3 *IIR_LATTICE_F32 coefficient format*

The filter coefficients must be stored into two buffers, one pointed by pFltCoefk and the other by pFltCoefv as shown below:

Table 29. pFltCoef format for IIR_LATTICE_F32

CPU Address Offset	Acronyms	Description	Data type
0	v[n] or k[n]	filter coefficient	F32
4	v[n-1] or k[n-1]	filter coefficient	F32
...	...	filter coefficient	F32
4 x (n-1)	v[1] or k[1]	filter coefficient	F32
4 x n	v[0]	filter coefficient	F32

1. *'n' represents the filter order, and is equal to NDLYC*

5.3.6.4 *Filter states*

Each filter instance has its own filter state. The application must instantiate a buffer in BRAM-AB containing the filter state for each filter instance.

Table 30. FltState structure format for IIR_LATTICE_F32

CPU Address offset	Field name	Type	Description
0	pStateBuf	F32*	Filter state buffer start address
4	pWork	F32*	Working pointer on next State buffer entry
8	NDLYC (TBD)	U32	Filter state size or number of delay blocks. NDLYC is equal to filter order

The created FltState structure must be initialized before a filter uses it.

- pStateBuf and pWork pointers must point to the StateBuf[0] element. These pointers are for SPE internal usage and must be translated into pointers directly usable by the SPE as explained on Section 2.2.4: Buffer pointer transmission on processing lists even if filter is used in a direct command.
- NDLYC must contain the filter order.
- StateBuf[x] must be initialized to 0

The next table shows the format of the filter state table.

Table 31. Filter state buffer for IIR_LATTICE_F32

CPU Address offset	Field name	Type	Buff Type	Description
0	DlyC[0]	F32	ABMP	Filter state data
4	DlyC[1]	F32	ABMP	Filter state data
...	...	...	...	...
4*(NDLYC-1)	DlyC[NDLYC-1]	F32	ABMP	Filter state data

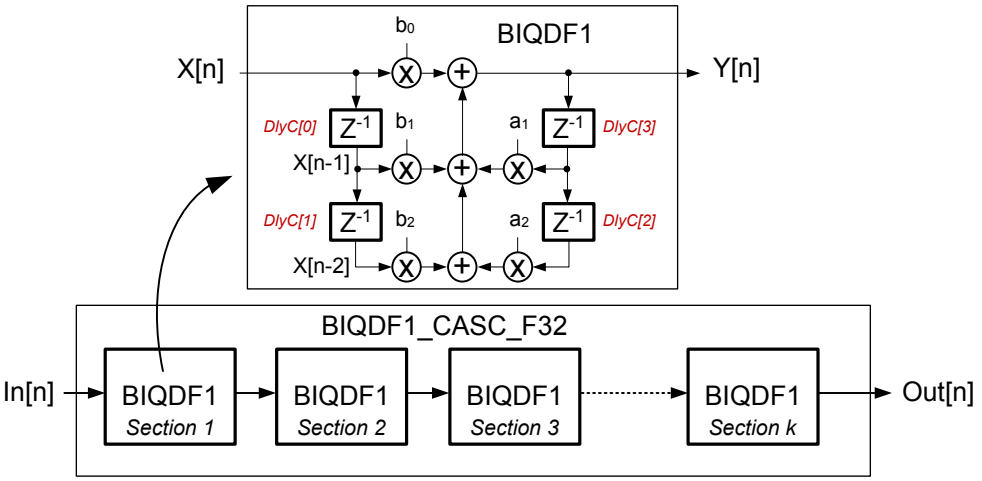
5.3.7 BIQDF1_CASC_F32

The BIQDF1_CASC_F32 can be either executed as a direct command (accelerator mode) or programmed into a processing list.

This function filter is implemented as a cascade of second order Biquad sections. Direct form 1 Biquad sections are used. The amount of cascaded sections is programmable and limited to 32.

The figure below shows the filter structure:

Figure 14. **Cascaded Biquad direct form 1 structure**



DlyC[x] stands for delay cells

DT77847V1

5.3.7.1 *Command format to record BIQDF1_CASC_F32*

Table 32. **Command format to record BIQDF1_CASC_F32**

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	BIQDF1_CASC_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pIn	F32*	ABMP, IBP, POP	Pointer on input data buffer
HSP_PARAMR1	pFltCoef	F32*	ABMP, POP	Pointer on filter coefficient buffer
HSP_PARAMR2	pOut	F32*	ABMP, IBP, POP	Pointer on output data buffer
HSP_PARAMR3	pFltState	ST*	ABMP	Pointer on filter state buffer
HSP_PARAMR4	N	U32	-	Number of samples to precess
HSP_PARAMR[14:5]	-	-	-	-
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pIn	Pointer on input data The input buffer must contain a real scalar in float32. The input and output buffer pointers can be the same.
pFltCoef	Pointer on filter coefficient buffer Must contain real values in float32. See format in Table 34. pFltCoef format for BIQDF1_CASC_F32 and BIQDF2T_CASC_F32.
pOut	Pointer on output data Filtered samples. The input and output buffer pointers can be the same.
pFltState	Address of the filter state buffer See Section 5.3.7.4: BIQDF1_CASC_F32 filter states for details.
N	Number of elements to process Can be any integer between 1 and 4096.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target:

If POP0 = 1, it is a pointer-on-pointer otherwise it is an ABMP or IBP pointer depending on its value.

pFltCoef target:

If POP1 = 1, it is a pointer-on-pointer otherwise it is an ABMP pointer.

pOut target:

If POP2 = 1, it is a pointer-on-pointer otherwise it is an ABMP or IBP pointer depending on its value.

Refer to Section 2.2.6: IOTYPE description for details

5.3.7.2 *BIQDF1_CASC_F32 direct command format*

The next table shows the command format when BIQDF1_CASC_F32 is executed as a direct command.

Table 33. BIQDF1_CASC_F32 direct command format

Order	Registers	Acronyms	Type	Target	Description
1	HSP_DCMDIDR	BIQDF1_CASC_F32	U32	-	See Section 8: Function ID list
2	HSP_PARAMR3	pFltState	ST*	ABMP	See description above
3	HSP_PARAMR4	N	U32	-	
4	HSP_DCMDPTR0	pIn	F32*	ABMP	
5	HSP_DCMDPTR1	pFltCoef	F32*	ABMP	
6	HSP_DCMDPTR2	pOut	F32*	ABMP	

1. *Parameters must be written in the specified order.*
2. *Refer to Section 5: Processing functions description for acronym details*

Command status: see Section 5.1: Command status for details.

5.3.7.3 *BIQDF1_CASC_F32 and BIQDF2T_CASC_F32 table coefficient format*

IIR coefficients are stored interleaved and in reversed order, the if N-1 is the number of stages, coefficient buffer is as follows:

Table 34. pFltCoef format for BIQDF1_CASC_F32 and BIQDF2T_CASC_F32

CPU address offset	Acronyms	Description	Data type
0	b_{10}	Section 1 (first section)	F32
4	b_{11}		F32
8	b_{12}		F32
12	a_{11}		F32
16	a_{12}		F32
...	...	...	F32
20x(NBIQ-1)	b_{k0}	Section NBIQ (last section)	F32
20x(NBIQ-1) + 4	b_{k1}		F32
20x(NBIQ-1) + 8	b_{k2}		F32
20x(NBIQ-1) + 12	a_{k1}		F32
20x(NBIQ-1) + 16	a_{k2}		F32

5.3.7.4 *BIQDF1_CASC_F32 filter states*

Each filter instance has its own filter state containing the filter taps and some variables giving the state of the filter. The application must instantiate a buffer in BRAM containing the filter state for each filter instance.

Table 35. FltState structure format for BIQDF1_CASC_F32

CPU address offset	Field name	Type	Description
0	pStateBuf	F32*	Filter State buffer start address
4	pWork	F32*	Working pointer on next State buffer entry
8	NBIQ	U32	Number of biquad stages. Must be between 1 and 32.

The created FltState structure must be initialized before a filter uses it.

- pStateBuf and pWork pointers must point to the StateBuf[0] element. These pointers are for SPE internal usage and must be translated into pointers directly usable by the SPE as explained on Section 2.2.4: Buffer pointer transmission on processing lists even if filter is used in a direct command.
- NBIQ must contain the number of biquad stages.
- StateBuf[x] must be initialized to 0

The next table shows the format of the filter state table. The filter state buffer can be contiguous to FltState.

Table 36. Filter state buffer for BIQDF1_CASC_F32

CPU Address offset	Field name	Type	Buff Type	Description
0	StateBuf[0]	F32	ABMP	Filter state data
4	StateBuf[1]	F32	ABMP	Filter state data
...	...	...	...	...
12*NBIQ	StateBuf[(4xNBIQ)-1]	F32	ABMP	Filter state data

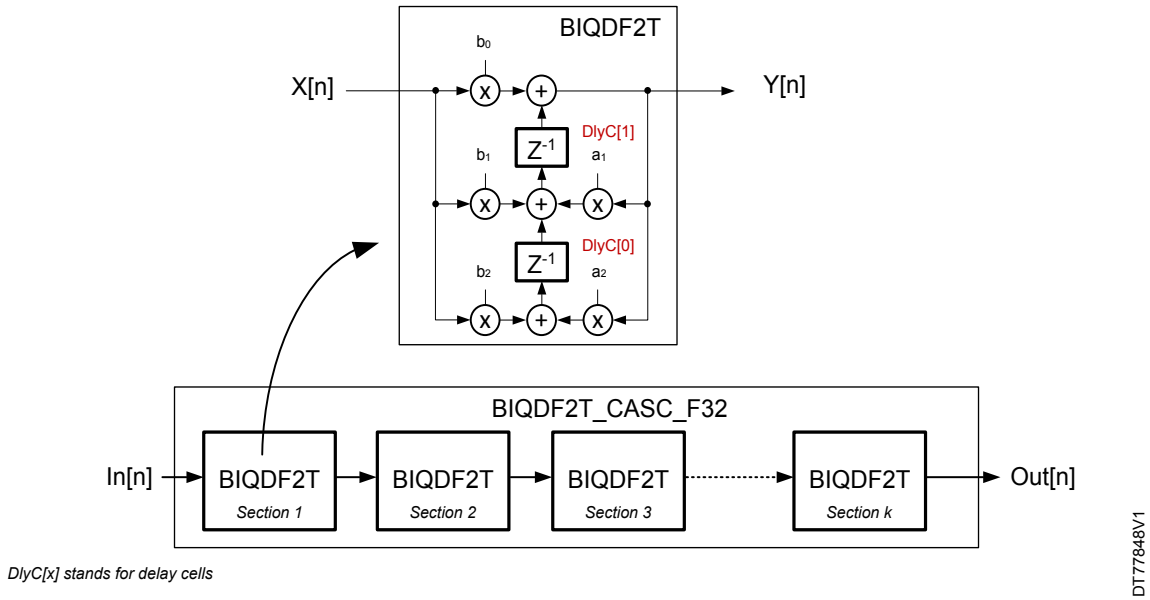
5.3.8 BIQDF2T_CASC_F32

The BIQDF2T_CASC_F32 can be either executed as a direct command (accelerator mode) or programmed into a processing list.

This function filter is implemented as a cascade of second order Biquad sections. Direct form 2 transposed Biquad sections are used. The amount of cascaded sections is programmable.

The figure below shows the filter structure:

Figure 15. Cascaded Biquad direct form 2 transposed filter structure



DlyC[x] stands for delay cells

Each filter instance has its own filter state. The filter state is stored into the HSP private RAM.

5.3.8.1 *Command format to record BIQDF2T_CASC_F32*

Table 37. Command format to record BIQDF2T_CASC_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	BIQDF2T_CASC_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pIn	F32*	ABMP, IBP, POP	Pointer on input data buffer
HSP_PARAMR1	pFltCoef	F32*	ABMP, POP	Pointer on filter coefficient buffer
HSP_PARAMR2	pOut	ST*	ABMP, IBP, POP	Pointer on output data buffer
HSP_PARAMR3	pFltState	F32*	ABMP	Pointer on filter state buffer
HSP_PARAMR4	N	U32	-	Number of samples to precess
HSP_PARAMR[14:5]	-	-	-	-
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pIn	Pointer on input data The input buffer must contain a real scalar in float32. The input and output buffer pointers can be the same.
pFltCoef	Pointer on filter coefficient buffer Must contain real values in float32. See format in Table 34. pFltCoef format for BIQDF1_CASC_F32 and BIQDF2T_CASC_F32.
pOut	Pointer on output data Filtered samples. The input and output buffer pointers can be the same.
pFltState	Address of the filter state buffer See Section 5.3.8.3: BIQDfilter states for details.
N	Number of elements to process Can be any integer between 1 and 4096.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is an ABMP or IBP pointer depending on its value. pFltCoef target: If POP1 = 1, it is a pointer-on-pointer otherwise it is an ABMP pointer. pOut target: If POP2 = 1, it is a pointer-on-pointer otherwise it is an ABMP or IBP pointer depending on its value. Refer to Section 2.2.6: IOTYPE description for details

5.3.8.2 *BIQDF2T_CASC_F32 direct command format*

The next table shows the command format when BIQDF2T_CASC_F32 is executed as a direct command.

Table 38. BIQDF2T_CASC_F32 direct command format

Order	Registers	Acronyms	Type	Target	Description
1	HSP_DCMDIDR	BIQDF2T_CASC_F32	U32	-	See Section 8: Function ID list
2	HSP_PARAMR3	pFltState	ST*	ABMP	See description above
3	HSP_PARAMR4	N	U32	-	
4	HSP_DCMDPTR0	pIn	F32*	ABMP	

Order	Registers	Acronyms	Type	Target	Description
5	HSP_DCMDPTR1	pFltCoef	F32*	ABMP	See description above
6	HSP_DCMDPTR2	pOut	F32*	ABMP	

1. *Parameters must be written in the specified order.*
2. *Refer to Section 5: Processing functions description for acronym details*

Command status: see Section 5.1: Command status for details.

5.3.8.3 *BIQDfilter states*

Each filter instance has its own filter state containing the filter taps and some variables giving the state of the filter. The application must instantiate a buffer in BRAM containing the filter state for each filter instance.

Table 39. FltState structure format for BIQDF2T_CASC_F32

CPU address offset	Field name	Type	Description
0	pStateBuf	F32*	Filter State buffer start address
4	pWork	F32*	Working pointer on next State buffer entry
8	NBIQ	U32	Number of biquad stages. Must be between 1 and 32

1. *U(I)32 stands for 32-bit unsigned (signed) integer, U(I)32* stands for pointer on 32-bit unsigned (signed) integer,F32 stands for 32-bit floating-point and F32* stands for pointer on 32-bit floating-point.*

The created FltState structure must be initialized before a filter uses it.

- pStateBuf and pWork floating-pointers must point on the StateBuf[0] element. These pointers are for SPE internal usage and must be translated into pointers directly usable by the SPE as explained on Section 2.2.4: Buffer pointer transmission on processing lists even if filter is used in a direct command.
- NBIQ must contain the number of biquad stages.
- StateBuf[x] must be initialized to 0

The next table shows the format of the filter state table. The filter state buffer can be contiguous to FltState.

Table 40. Filter state buffer for BIQDF2T_CASC_F32

CPU Address offset	Field name	Type	Buff Type	Description
0	DlyC_1[0]	F32	ABMP	Filter state data
4	DlyC_1[1]	F32	ABMP	Filter state data
8	DlyC_2[0]	F32	ABMP	Filter state data
12	DlyC_2[1]	F32	ABMP	Filter state data
...	...	...	...	...
8xNBIQ - 4	DlyC_NBIQ[1]	F32	ABMP	Filter state data

1. *U(I)32 stands for 32-bit unsigned (signed) integer, U(I)32* stands for pointer on 32-bit unsigned (signed) integer,F32 stands for 32-bit floating-point and F32* stands for pointer on 32-bit floating-point.*
2. *ABMP stands for pointer on BRAM-AB. Each floating-pointer must be translated into pointers usable by SPE. Refer to Section 2.2.5: Buffer pointer transmission on direct commands for details.*

5.3.9 FLTBANK_F32

The FLTBANK_F32 can be either executed as a direct command (accelerator mode) or programmed into a processing list.

5.3.9.1 *Overview*

The FLTBANK_F32 function performs the sum of the product of a bank of filters with the power spectrum of the input signal. Typically, it can be used to perform MEL filters.

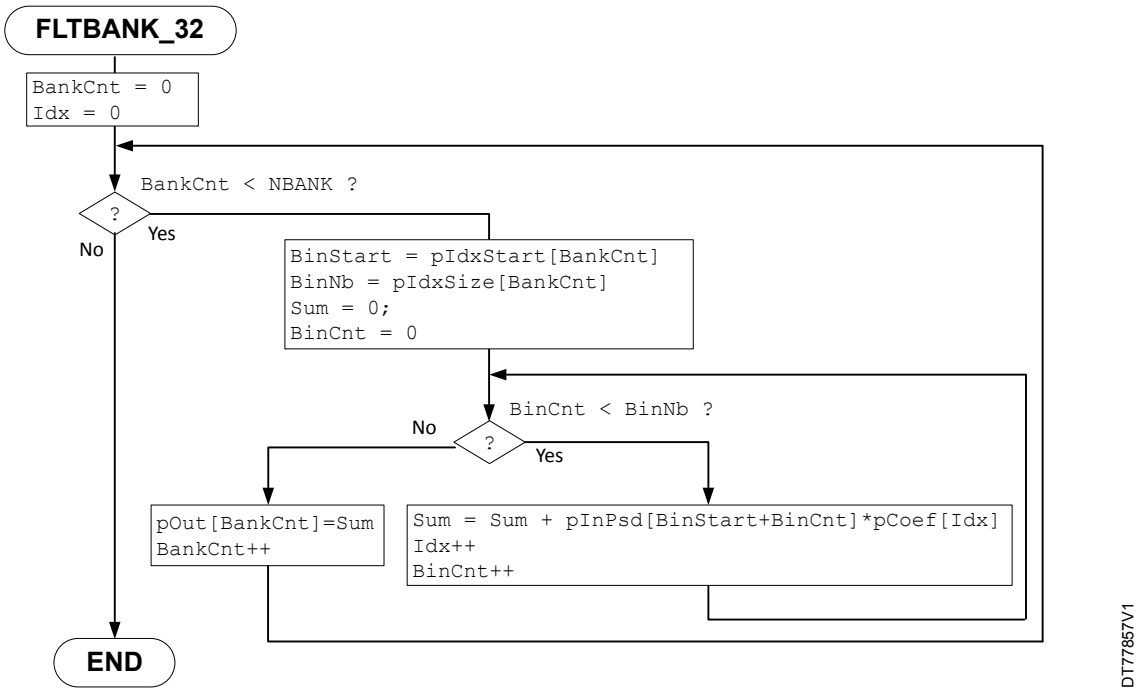
The filter bank buffer must respect the data format described in this section. ST provides a function generating the MEL filter shapes.

The FLTBANK_F32 function performs the following operations:

- Perform the sum of the products of each filter bank shapes contained into **pFltShape** with the power spectrum contained into the **pInPSD**.
- The result is provided into the **pOut** buffer.

The pseudocode is shown in the figure below.

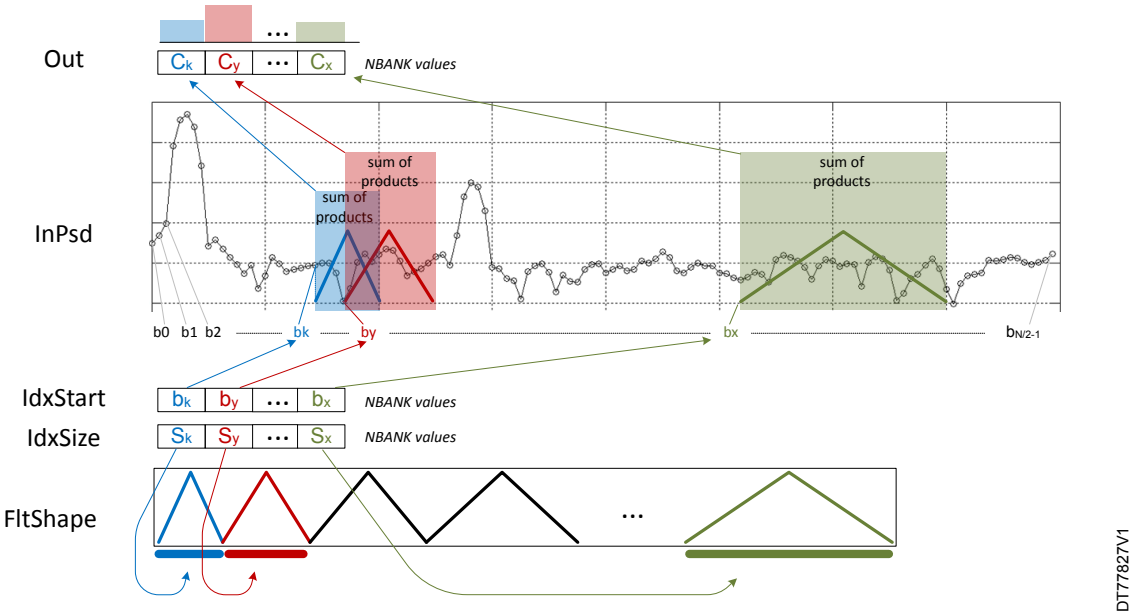
Figure 16. FLTBANK_F32 algorithm



The next figure shows the way the function FLTBANK_F32 is using the data parameters.

The InPsd buffer contains the power spectrum of the signal to be processed. The buffer IdxStart contains the first bin value of InPsd (b_n) to be used with the shapes stored into the FltShape buffer. The IdxSize gives the length of each shape.

Figure 17. **FLTBANK_F32 data organization**



Caution: *In order to work properly the application must follow the following rules*

- The size provided into IdxSize buffer must be aligned with the number of elements of each filter shape.
- $b_n + S_n$ must not exceed the InPsd size.
- The size of FltShape buffer must be at least the sum of all S_n values.

5.3.9.2 *Command format to record FLTBANK_F32*

Table 41. **Command format to record FLTBANK_F32**

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	FLTBANK_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInPsd	F32*	ABMP	Pointer on input PSD buffer
HSP_PARAMR1	pOut	F32*	ABMP	Pointer on output data buffer
HSP_PARAMR2	pIdxStart	U32	ABMP	Pointer on start index buffer
HSP_PARAMR3	pIdxSize	U32	ABMP	Pointer on filter size buffer
HSP_PARAMR4	pFltShape	F32*	ABMP	Pointer on filter shape buffer
HSP_PARAMR5	NBANK	U32	-	Number of samples to precess
HSP_PARAMR[14:6]	-	-	-	-
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInPsd	Pointer on input PSD buffer The input buffer must contain an array of real numbers in float32 representing typically the power spectrum of the signal to be processed through the banks of filters.
pOut	Pointer on output data Output buffer contains energies for each filter bank. The buffer size must be at least NBANK.

pIdxStart	Pointer on a buffer containing the start index of each filter shape This buffer must contain integer numbers representing the power spectrum bin number starting point where each filter bank must be applied. See Table 44. pIdxStart format of FLTBANK_F32 for details.
pIdxSize	Pointer on the vector containing the number bin to take for each start index. The vector must contain integer numbers representing the power spectrum bin size for each corresponding index where each filter bank must be applied. See Table 45. IdxSize format of FLTBANK_F32 for details.
pFltShape	Pointer on filter shapes buffer Must contain real numbers in float32 representing the filter shapes. This buffer must respect the format given in Table 43. pFltShape format of FLTBANK_F32 and FLTBANK_EXT_F32. The size of this buffer must be limited to 4096 points.
NBANK	Number of filter banks Can be any integer between 8 and 512.
IOTYPE	Input and output buffer types: Indicates the pointers type pInPsd target: it is an ABMP pointer pOut target: it is an ABMP pointer Refer to Section 2.2.6: IOTYPE description for details

5.3.9.3 *FLTBANK_F32 direct command format*

The next table shows the command format when FLTBANK_F32 is executed as a direct command.

Table 42. FLTBANK_F32 direct command format

Order	Registers	Acronyms	Type	Target	Description
1	HSP_DCMDIDR	FLTBANK_F32	U32	-	See Section 8: Function ID list
2	HSP_PARAMR2	pIdxStart	U32*	ABMP	See description above
3	HSP_PARAMR2	pIdxSize	U32*	ABMP	
4	HSP_PARAMR2	pFltShape	F32*	ABMP	
5	HSP_PARAMR4	NBANK	U32	-	
6	HSP_DCMDPTR0	pInPsd	F32*	ABMP	
7	HSP_DCMDPTR1	pOut	F32*	ABMP	

1. *Parameters must be written in the specified order.*
2. *Refer to Section 5: Processing functions description for acronym details*

Command status: see Section 5.1: Command status for details.

5.3.9.4 *pFltShape format for FLTBANK_F32 function*

This buffer contains all the filter shapes in the frequency domain. The frequency resolution shall be the same as the one of the power spectrum. This buffer must be used in conjunction with **pIdxStart**. The number of values describing each filter shape is variable.

pFltShape is generally a large buffer located into the global memory, but can also be placed into the BRAM to improve performance if there is enough room.

The filter shape buffer format is as follows:

Table 43. pFltShape format of FLTBANK_F32 and FLTBANK_EXT_F32

Offset addr.	Acronyms	Description	Data type
0	C_0	First filter shape computed with the frequency resolution of the power spectrum.	F32
1	C_1		F32

Offset addr.	Acronyms	Description	Data type
...	...	First filter shape computed with the frequency resolution of the power spectrum.	F32
k	C_k		F32
k+1	C_{k+1}	Second filter shape computed with the frequency resolution of the power spectrum.	F32
...	...		F32
k+j	C_{k+j}		F32
...	...	...	...
k+j+n-1	$C_{k+j+n-1}$	Last filter shape computed with the frequency resolution of the power spectrum.	F32
k+j+n	C_{k+j+n}		F32

1. *U8 stands for 8-bit unsigned integer, U16 stands for 16-bit unsigned integer, U32 stands for 32-bit unsigned integer, and F32 stands for 32-bit floating-point.*

5.3.9.5 *pIdxStart format for FLTBANK_F32 function*

This buffer contains the start index where each filter bank must be applied. This buffer must be used in conjunction with pFltShape and **pIdxSize**.

The index buffer format is as follows:

Table 44. pIdxStart format of FLTBANK_F32

Offset addr.	Acronyms	Description	Data type
0	BinStart[0]	First bin number where the first filter shape must be applied	U32
1	BinStart[1]	First bin number where the second filter shape must be applied	U32
...	...	...	...
x	BinStart[x]	First bin number where the last filter shape must be applied	U32

1. *U8 stands for 8-bit unsigned integer, U16 stands for 16-bit unsigned integer, U32 stands for 32-bit unsigned integer, and F32 stands for 32-bit floating-point.*

5.3.9.6 *pIdxSize format for FLTBANK_F32 function*

This buffer contains the number of data for each filter bank. This buffer must be used in conjunction with **pFltShape** and **pIdxStart**.

The index buffer format is as follows:

Table 45. IdxSize format of FLTBANK_F32

Offset addr.	Acronyms	Description	Data type
0	BinNb[0]	Number of elements of each filter shape	U32
1	BinNb[1]	Number of elements of each filter shape	U32
...	...	...	...
x	BinNb[x]	Number of elements of each filter shape	U32

1. *U8 stands for 8-bit unsigned integer, U16 stands for 16-bit unsigned integer, U32 stands for 32-bit unsigned integer, and F32 stands for 32-bit floating-point.*

Refer to Using FLTBANK_F32 to perform a Mel Filter for additional information.

5.3.10 CONV_F32

The CONV_F32 can be either executed as a direct command (accelerator mode) or programmed into a processing list.

This function performs a convolution between two vectors. Convolution is a mathematical operation frequently used in filtering and data analysis.

If two input vectors are considered, where a[n] has a length of La, the convolution of those two vectors gives an output vector y[n] having a length of La+Lb-1

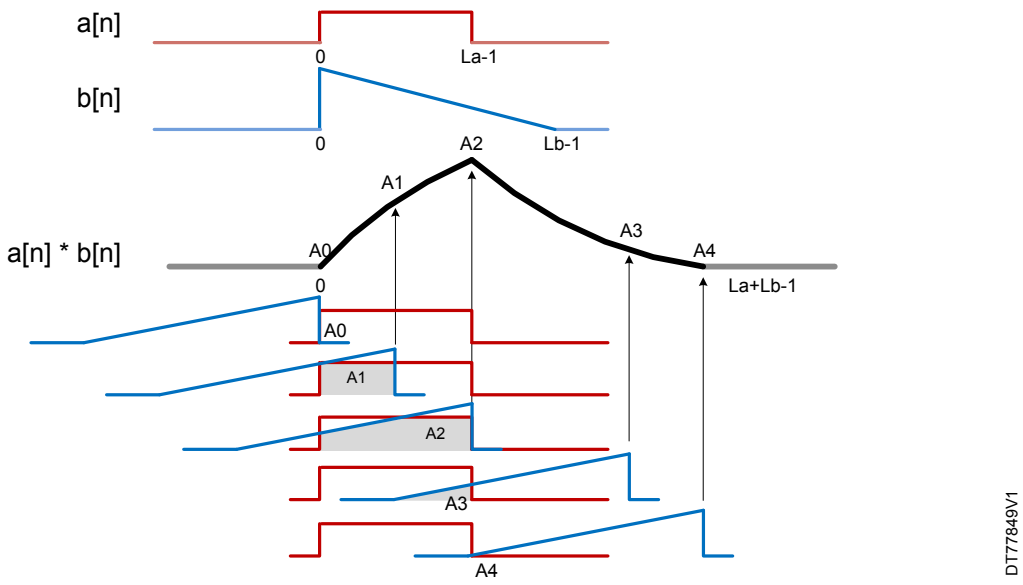
The convolution operation is as follows:

$$y[n] = a[n] \times b[n]$$

$$y[n] = \sum_{k=0}^{La} a[k] \cdot b[n-k]$$

When two signals a[n] and b[n] are convolved, the signal b[n] slides over a[n]. For each offset n, the overlapping portions of a[n] and b[n] are multiplied and summed together.

Figure 18. CONV_F32 operation



5.3.10.1 *Command format to record CONV_F32*

Table 46. Command format to record CONV_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	CONV_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP	Pointer on input data buffer A
HSP_PARAMR1	pInB	F32*	ABMP, POP	Pointer on input data buffer B
HSP_PARAMR2	pOut	F32*	ABMP, POP	Pointer on output data buffer
HSP_PARAMR3	LA	U32	-	Number of elements on buffer pInA
HSP_PARAMR4	LB	U32	-	Number of elements on buffer pInB
HSP_PARAMR[14:5]	-	-	-	-
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data buffer A Input buffer A is an array of real numbers in float32 representing input samples.

pInB	Pointer on input data buffer B Input buffer B is an array of real numbers in float32 representing input samples.
pOut	Pointer on output data buffer Contain the convolution result in float32 The buffer size must be bigger than or equal to LA+LB.
LA	Number of elements to consider on buffer A. Must be lower than or equal to buffer A size Can be any integer between 2 and 2048.
LB	Number of elements to consider on buffer B. Must be lower than or equal to buffer B size Can be any integer between 2 and 2048.
IOTYPE	Input and output buffer types: Indicates the floating-pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is an ABMP pointer depending on its value. pInB target: If POP1 = 1, it is a pointer-on-pointer otherwise it is an ABMP pointer depending on its value. pOut target: If POP2 = 1, it is a pointer-on-pointer otherwise it is an ABMP pointer depending on its value. Refer to Section 2.2.6: IOTYPE description for details

5.3.10.2 *CONV_F32 direct command format*

The next table shows the command format when CONV_F32 is executed as a direct command.

Table 47. CONV_F32 direct command format

Order	Registers	Acronyms	Type	Target	Description
1	HSP_DCMDIDR	CONV_F32	U32	-	See Section 8: Function ID list
2	HSP_PARAMR3	LA	U32	ABMP	See description above
3	HSP_PARAMR4	LB	U32	ABMP	
4	HSP_DCMDPTR0	pInA	F32*	ABMP	
5	HSP_DCMDPTR1	pInB	F32*	ABMP	
6	HSP_DCMDPTR2	pOut	F32*	ABMP	

1. *Parameters must be written in the specified order.*
2. *Refer to Section 5: Processing functions description for acronym details*

Command status: see Section 5.1: Command status for details.

5.3.11 CORR_F32

The CORR_F32 can be either executed as a direct command (accelerator mode) or programmed into a processing list.

This function performs a correlation between two vectors. The correlation is an operation similar to convolution, the main difference is that one of the input is flipped in convolution. The correlation is commonly used to measure the similarity between two signals.

If we consider two input vectors: a[n] having a length of La, and b[n] having a length of Lb, the correlation of those two vectors gives an output vector y[n] having a length of La+Lb-1

The correlation operation is as follows:

$$y[n] = a[n] * b[-n]$$

$$y[n] = \sum_{k=0}^{La} a[k] \cdot b[k-n]$$

5.3.11.1 Command format to record CORR_F32

Table 48. Command format to record CORR_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	CORR_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP	Pointer on input data buffer A
HSP_PARAMR1	pInB	F32*	ABMP, POP	Pointer on input data buffer B
HSP_PARAMR2	pOut	F32*	ABMP, POP	Pointer on output data buffer
HSP_PARAMR3	LA	U32	-	Number of elements on buffer pInA
HSP_PARAMR4	LB	U32	-	Number of elements on buffer pInB
HSP_PARAMR[14:5]	-	-	-	-
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data buffer A Input buffer A is an array of real numbers in float32 representing input samples.
pInB	Pointer on input data buffer B Input buffer B is an array of real numbers in float32 representing input samples.
pOut	Pointer on output data buffer Contain the correlation result in float32 The buffer size must be bigger than or equal to LA+LB.
LA	Number of elements to consider on buffer A. Must be lower than or equal to buffer A size Can be any integer between 2 and 2048.
LB	Number of elements to consider on buffer B. Must be lower than or equal to buffer B size Can be any integer between 2 and 2048.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is an ABMP pointer depending on its value. pInB target: If POP1 = 1, it is a pointer-on-pointer otherwise it is an ABMP pointer depending on its value. pOut target: If POP2 = 1, it is a pointer-on-pointer otherwise it is an ABMP pointer depending on its value. Refer to Section 2.2.6: IOTYPE description for details

5.3.11.2 CORR_F32 direct command format

The next table shows the command format when CORR_F32 is executed as a direct command.

Table 49. **CORR_F32 direct command format**

Order	Registers	Acronyms	Type	Target	Description
1	HSP_DCMDIDR	CORR_F32	U32	-	See Section 8: Function ID list
2	HSP_PARAMR3	LA	U32	ABMP	See description above
3	HSP_PARAMR4	LB	U32	ABMP	
4	HSP_DCMDPTR0	pInA	F32*	ABMP	
5	HSP_DCMDPTR1	pInB	F32*	ABMP	
6	HSP_DCMDPTR2	pOut	F32*	ABMP	

1. *Parameters must be written in the specified order.*
2. *Refer to Section 5: Processing functions description for acronym details*

5.3.12 GET_STATE

The GET_STATE can be only executed as a direct command.

This function allows the application to read the filter state of the filters generated by:

- FIR_F32
- FIRDEC_F32
- FIRLMS_F32
- DC_IIRDF1_3P3Z_F32
- DC_IIRDF1_2P2Z_F32
- IIR_LATTICE_F32

This function reorders and transfers the filter state contained into **pFltState** into **pOut**. Using this command insures that the HSP is not using this filter when the filter state is read.

5.3.12.1 GET_STATE direct command format

The next table shows the command format when GET_STATE is executed as a direct command.

Table 50. **GET_STATE direct command format**

Order	Registers	Acronyms	Type	Target	Description
1	HSP_DCMDIDR	GET_STATE	U32	-	See Section 8: Function ID list
2	HSP_PARAMR1	pFltState	ST*	ABMP	Pointer on the filter state
3	HSP_PARAMR2	SIZE	U32	ABMP	State size according filter type
4	HSP_DCMDPTR0	pOut (tbc)	F32*	ABMP	Pointer on output buffer

1. *Parameters must be written in the specified order.*
2. *Refer to Section 5: Processing functions description for acronym details*

pFltState	Pointer on the filter state. It must be the address seen by SPE.
SIZE	State size according filter type: - FIR_F32: nb taps (i.e. coefficients) - FIRDEC_F32: nb taps (i.e. coefficients) - FIRLMS_F32: nb taps (i.e. coefficients) - IIR_LATTICE_F32: nb delay cells - DC_IIRDF1_3P3Z_F32: 6 - DC_IIRDF1_2P2Z_F32: 4
pOut	Pointer on output buffer where the filter state must be copied The pointer address is the address seen on CPU side. See Table 51. pOut format for FIR, FIRDEC_F32, FIRLMS_F32 and IIR_LATTICE_F32 and Table 52. pOut format for DC_IIRDF1_3P3Z_F32 and DC_IIRDF1_2P2Z_F32.

Next tables show the format of the filter states buffers for different filter types.

Table 51. pOut format for FIR, FIRDEC_F32, FIRLMS_F32 and IIR_LATTICE_F32

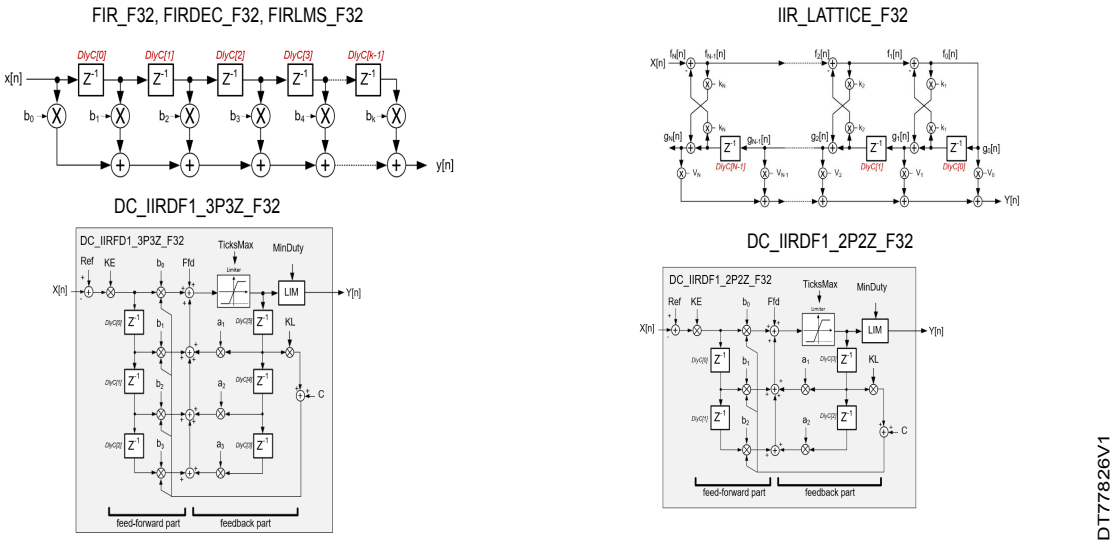
CPU address offset	FIR_F32, FIRDEC_F32, FIRLMS_F32	CPU address offset	IIR_LATTICE_F32
0	DlyC[k-1]	0	DlyC[N-1]
4	DlyC[k-2]	4	DlyC[N-2]
...	...	...	...
4(k-1)	DlyC[0]	4(N-1)	DlyC[0]

Table 52. pOut format for DC_IIRDF1_3P3Z_F32 and DC_IIRDF1_2P2Z_F32

CPU address offset	DC_IIRDF1_3P3Z_F32	DC_IIRDF1_2P2Z_F32
0	DlyC[2]	DlyC[1]
4	DlyC[3]	DlyC[2]
8	DlyC[1]	DlyC[0]
12	DlyC[4]	DlyC[3]
16	DlyC[0]	
20	DlyC[5]	

The next figure illustrates the position of each delay cell.

Figure 19. Filter delay cell identification



5.3.13 GET_STATE_IIRDF1

The GET_STATE_IIRDF1 can be only executed as a direct command (accelerator mode).

This function allows the application to read the filter state of IIRDF1_F32 filters.

This function reorders and transfers the filter state contained into **pFltState** into **pOut**. Using this command insures that the HSP is not using this filter when the filter state is read.

5.3.13.1 *GET_STATE_IIRDF1 direct command format*

Table 53. GET_STATE_IIRDF1 direct command format

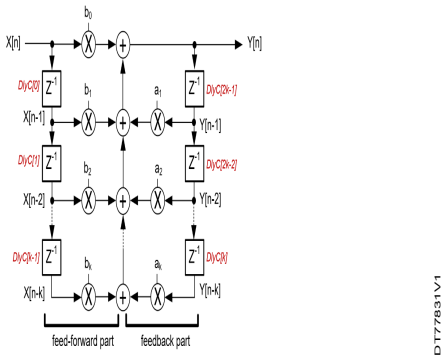
Order	Registers	Acronyms	Type	Target	Description
1	HSP_DCMDIDR	GET_STATE_IIRDF1	U32	-	See Section 8: Function ID list
2	HSP_PARAMR1	pFltState	ST*	ABMP	Pointer on the filter state
3	HSP_PARAMR2	SIZE	U32	ABMP	State size
4	HSP_DCMDPTR0	pOut (tbc)	F32*	ABMP	Pointer on output buffer

1. *Parameters must be written in the specified order.*
2. *Refer to Section 5: Processing functions description for acronym details*

pFltState	Pointer on the filter state. It must be the address seen by SPE.
SIZE	State size is the number of delay cells of IIRDF1_F32
pOut	Pointer on output buffer where the filter state must be copied. The pointer address is the address seen on CPU side. See Table 54. pOut format for IIRDF1_F32.

The next tables show the format of the filter state buffer.

Table 54. pOut format for IIRDF1_F32

CPU address offset	IIRDF1_F32	Delay cell identification
0	DlyC[k-1]	
4	DlyC[k]	
...	...	
4(2k-4)	DlyC[1]	
4(2k-3)	DlyC[2k-2]	
4(2k-2)	DlyC[0]	
4(2k-1)	DlyC[2k-1]	

5.3.14 GET_STATE_BIQ

The GET_STATE_BIQ can be only executed as a direct command (accelerator mode).

This function allows the application to read the filter state of the filters generated by:

- BIQDF1_CASC_F32
- BIQDF2T_CASC_F32

This function reorders and transfers the filter state contained into **pFltState** into **pOut**. Using this command insures that the HSP is not using this filter when the filter state is read.

5.3.14.1 *GET_STATE_BIQ direct command format*

The next table shows the command format when GET_STATE_BIQ is executed as a direct command.

Table 55. GET_STATE_BIQ direct command format

Order	Registers	Acronyms	Type	Target	Description
1	HSP_DCMDIDR	GET_STATE_BIQ	U32	-	See Section 8: Function ID list
2	HSP_PARAMR1	pFltState	ST*	ABMP	Pointer on the filter state

Order	Registers	Acronyms	Type	Target	Description
3	HSP_PARAMR2	SIZE	U32	ABMP	State size according filter type
4	HSP_DCMDPTR0	pOut (tbc)	F32*	ABMP	Pointer on output buffer

1. *Parameters must be written in the specified order.*
2. *Refer to Section 5: Processing functions description for acronym details*

pFltState	Pointer on the filter state. It must be the address seen by SPE.
SIZE	State size according filter type: - BIQDF1_CASC_F32: NBIQ x 4 - BIQDF2T_CASC_F32: NBIQ x 2
pOut	Pointer on output buffer where the filter state must be copied. The pointer address is the address seen on CPU side. See Table 56. pOut format for BIQDF1_CASC_F32 and BIQDF2T_CASC.

Table 56. pOut format for BIQDF1_CASC_F32 and BIQDF2T_CASC

CPU address offset	BIQDF1_CASC_F32	Stage nb		CPU address offset	BIQDF2T_CASC_F32	Stage nb
0	DlyC[1]	1		0	DlyC[0]	1
4	DlyC[2]			4	DlyC[1]	
8	DlyC[0]			8	DlyC[0]	2
12	DlyC[3]			12	DlyC[1]	
16	DlyC[1]	2		...	...	...
20	DlyC[2]					
24	DlyC[0]					
28	DlyC[3]					
...	...	...				

5.3.15 SET_STATE

The SET_STATE can be only executed as a direct command (accelerator mode).

This function allows the application to change the filter state of the filters generated by:

- FIR_F32
- FIRDEC_F32
- FIRLMS_F32
- DC_IIRDF1_3P3Z_F32
- DC_IIRDF1_2P2Z_F32
- IIR_LATTICE_F32

This function reorders and transfers the filter state contained into **pInA** into **pFltState**. Using this command insures that the HSP is not using this filter when the filter state is updated.

5.3.15.1 SET_STATE direct command format

The next table shows the command format when SET_STATE is executed as a direct command.

Table 57. SET_STATE direct command format

Order	Registers	Acronyms	Type	Target	Description
1	HSP_DCMDIDR	SET_STATE	U32	-	See Section 8: Function ID list
2	HSP_PARAMR1	pFltState	ST*	ABMP	Pointer on the filter state

Order	Registers	Acronyms	Type	Target	Description
3	HSP_PARAMR2	SIZE	U32	ABMP	State size according filter type
4	HSP_DCMDPTR0	pIn (tbc)	F32*	ABMP	Pointer on input buffer

1. *Parameters must be written in the specified order.*
2. *Refer to Section 5: Processing functions description for acronym details*

pFltState	Pointer on the filter state. It must be the address seen by SPE.
SIZE	State size according filter type: - FIR_F32: nb taps (i.e. coefficients) - FIRDEC_F32: nb taps (i.e. coefficients) - FIRLMS_F32: nb taps (i.e. coefficients) - IIR_LATTICE_F32: nb delay cells - IIRDF1_F32: nb delay cells - DC_IIRDF1_3P3Z_F32: 6 - DC_IIRDF1_2P2Z_F32: 4
pIn	Pointer on input buffer This buffer must contain the filter state to be copied into the buffer pointed by pFltState. The pointer address is the address seen on CPU side. The data format inside pIn must respect the format given on Table 51. pOut format for FIR, FIRDEC_F32, FIRLMS_F32 and IIR_LATTICE_F32 and Table 52. pOut format for DC_IIRDF1_3P3Z_F32 and DC_IIRDF1_2P2Z_F32.

5.3.16 SET_STATE_IIRDF1

The SET_STATE_IIRDF1 can be only executed as a direct command.

This function allows the application to change the filter state of the filters generated by:

- FIR_F32
- FIRDEC_F32
- FIRLMS_F32
- DC_IIRDF1_3P3Z_F32
- DC_IIRDF1_2P2Z_F32
- IIR_LATTICE_F32

This function reorders and transfers the filter state contained into **pInA** into **pFltState**. Using this command insures that the HSP is not using this filter when the filter state is updated.

5.3.16.1 SET_STATE_IIRDF1 direct command format

The next table shows the command format when SET_STATE_IIRDF1 is executed as a direct command.

Table 58. SET_STATE_IIRDF1 direct command format

Order	Registers	Acronyms	Type	Target	Description
1	HSP_DCMDIDR	SET_STATE_IIRDF1	U32	-	See Section 8: Function ID list
2	HSP_PARAMR1	pFltState	ST*	ABMP	Pointer on the filter state
3	HSP_PARAMR2	SIZE	U32	ABMP	State size according filter type
4	HSP_DCMDPTR0	pIn (tbc)	F32*	ABMP	Pointer on input buffer

1. *Parameters must be written in the specified order.*
2. *Refer to Section 5: Processing functions description for acronym details*

pFltState	Pointer on the filter state. It must be the address seen by SPE.
SIZE	State size is the number of IIRDF1_F32 delay cells.
pIn	Pointer on input buffer This buffer must contain the filter state to be copied into the buffer pointed by pFltState. The pointer address is the address seen on CPU side. The data format of the buffer pointer by pIn must respect the format given on Table 54. pOut format for IIRDF1_F32.

5.3.17 SET_STATE_BIQ

The SET_STATE_BIQ can be only executed as a direct command (accelerator mode).

This function allows the application to change the filter state of the filters generated by:

- BIQDF1_CASC_F32
- BIQDF2T_CASC_F32

This function reorders and transfers the filter state contained into **pInA** into **pFltState**. Using this command insures that the HSP is not using this filter when the filter state is read.

5.3.17.1 SET_STATE_BIQ direct command format

The next table shows the command format when SET_STATE_BIQ is executed as a direct command.

Table 59. SET_STATE_BIQ direct command forma

Order	Registers	Acronyms	Type	Target	Description
1	HSP_DCMDIDR	SET_STATE_BIQ	U32	-	See Section 8: Function ID list
2	HSP_PARAMR1	pFltState	ST*	ABMP	Pointer on the filter state
3	HSP_PARAMR2	SIZE	U32	ABMP	State size according filter type
4	HSP_DCMDPTR0	pIn (tbc)	F32*	ABMP	Pointer on input buffer

1. *Parameters must be written in the specified order.*
2. *Refer to Section 5: Processing functions description for acronym details*

pFltState	Pointer on the filter state. It must be the address seen by SPE.
SIZE	State size according filter type: - BIQDF1_CASC_F32: NBIQ x 4 - BIQDF2T_CASC_F32: NBIQ x 2
pIn	Pointer on input buffer This buffer must contain the filter state to be copied into the buffer pointed by pFltState. The pointer address is the address seen on CPU side. The data format inside pIn must respect the format given on Table 56. pOut format for BIQDF1_CASC_F32 and BIQDF2T_CASC.

5.3.18 RST_STATE

The RST_STATE can be only executed as a direct command (accelerator mode).

This function allows the application to reset the filter state pointed by **pFltState** to 0. Using this command insures that the HSP is not using this filter when the filter state is reset.

5.3.18.1 RST_STATE direct command format

The next table shows the command format when RST_STATE is executed as a direct command.

Table 60. RST_STATE direct command format

Order	Registers	Acronyms	Type	Target	Description
1	HSP_DCMDIDR	RST_STATE	U32	-	See Section 8: Function ID list
2	HSP_PARAMR1	pFltState	ST*	ABMP	Pointer on the filter state
3	HSP_PARAMR2	SIZE	U32	ABMP	State size according filter type
4	HSP_DCMDPTR0	START	F32*	-	To start execution

1. *Parameters must be written in the specified order.*
2. *Refer to Section 5: Processing functions description for acronym details*

pFltState	Pointer on the filter state to be reset It must be the address seen by SPE.
SIZE	State size according filter type: - FIR_F32: number of coefficients - FIRDEC_F32: number of coefficients - FIRLMS_F32: number of coefficients - IIR_LATTICE_F32: number of delay cells - IIRDF1_F32: number of delay cells - DC_IIRDF1_3P3Z_F32: 6 - DC_IIRDF1_2P2Z_F32: 4 - BIQDF1_CASC_F32: NBIQ x 4 - BIQDF2T_CASC_F32: NBIQ x 2
START	Start execution Any value written into HSP_DCMDPTR0 starts command execution.

5.4 Vector functions on real numbers

This section describes the command formats to record a processing function into a processing list or to execute it directly (accelerator mode).

For all vector functions, the vector size must be at least 2. For size equal to 1, use scalar functions (see Section 5.6: Scalar functions on real).

The function ID of each vector function is provided in Section 8: Function ID list.

All the vector functions described in this section can either be recorded into a processing list or executed as a direct command.

The command format for functions executed as direct command are generally as those shown in Table 61 or Table 62. For functions having a format not covered by these tables, the direct command format is given in the function description.

The format for recording functions into processing lists is given in the function description.

Table 61. Format for generic direct commands with three pointers

Order	Registers	Acronyms	Type	Target	Description
1	HSP_DCMDIDR	function ID	U32	-	See Section 8: Function ID list
2	HSP_PARAMR3	N	U32	-	Number of samples to process
3	HSP_DCMDPTR0	pInA	F32*	ABMP	Pointer on the input buffer A
4	HSP_DCMDPTR1	pInB	F32*	ABMP	Pointer on the input buffer B
5	HSP_DCMDPTR2	pOut	F32*	ABMP	Pointer on the output buffer

1. *Parameters must be written in the specified order.*
2. *Refer to Section 2.2: Data buffer handling for standard functions for details.*

3. *U(I)32 stands for 32-bit unsigned (signed) integer, U(I)32* stands for pointer on 32-bit unsigned (signed) integer, F32 stands for 32-bit floating-point and F32* stands for pointer on 32-bit floating-point.*
4. *ABMP stands for pointer on BRAM-AB memory. Each pointer must be translated into pointers usable by SPE. Refer to for details.*

Table 62. Format for generic direct commands with two pointers

Order	Registers	Acronyms	Type	Target	Description
1	HSP_DCMDIDR	function ID	U32	-	See Section 8: Function ID list
2	HSP_PARAMR2	N	U32	-	Number of samples to process
3	HSP_DCMDPTR0	pInA	F32*	ABMP	Pointer on the input buffer
4	HSP_DCMDPTR1	pOut	F32*	ABMP	Pointer on the output buffer

1. *Parameters must be written in the specified order.*
2. *Refer to Section 2.2: Data buffer handling for standard functions for details.*
3. *U(I)32 stands for 32-bit unsigned (signed) integer, U(I)32* stands for pointer on 32-bit unsigned (signed) integer,F32 stands for 32-bit floating-point and F32* stands for pointer on 32-bit floating-point.*
4. *ABMP stands for pointer on BRAM-AB memory. Each pointer must be translated into pointers usable by SPE. Refer to for details.*

5.4.1 VECT_ABS_F32

The VECT_ABS_F32 can be either executed as a direct command (accelerator mode) or programmed into a processing list.

This function computes the absolute value of a real vector:

$$pOut[n] = pInA[n]$$

with n = 0, 1, 2,..., N-1

5.4.1.1 Command format to record VECT_ABS_F32

Table 63. Command format to record VECT_ABS_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	VECT_ABS_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP	Pointer on input data buffer
HSP_PARAMR1	pOut	F32*	ABMP, POP	Pointer on output data buffer
HSP_PARAMR2	N	U32	-	Number of elements to process
HSP_PARAMR[14:3]	-	-	-	-
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data buffer Input buffer is an array of real numbers in float32 representing input samples. pIn can be the same as pOut.
pOut	Pointer on output data buffer The result is a real vector in float32. pInA can be the same as pOut.
N	Number of elements to process Can be any integer between 2 and 4096.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target:

	If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pOut target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. Refer to Section 2.2.6: IOTYPE description for details

See Table 62. Format for generic direct commands with two pointers when VECT_ABS_F32 is executed as a direct command.

5.4.2 VECT_ABSMAX_F32

The VECT_ABSMAX_F32 can be either executed as a direct command (accelerator mode) or programmed into a processing list.

VECT_ABSMAX_F32 provides the maximum of the absolute value of a real vector:

$$[MaxVal, \quad Pos] = max(pIn[n])$$

with n = 0, 1, 2,..., N-1

5.4.2.1 *Command format to record VECT_ABSMAX_F32*

Table 64. Command format to record VECT_ABSMAX_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	VECT_ABSMAX_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP	Pointer on input data buffer A
HSP_PARAMR1	pOutVal	F32*	ABMP, POP, IBP	Maximum absolute value
HSP_PARAMR2	pOutIdx	U32*	ABMP, POP, IBP	Position of the maximum absolute value
HSP_PARAMR3	N	U32	-	Number of elements to process
HSP_PARAMR[14:4]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data buffer A Input buffer A is an array of real numbers in float32 representing input samples.
pOutVal	Pointer on maximum absolute value Address of the variable containing the maximum absolute value
pOutIdx	Pointer on position of maximum absolute value Address of the variable containing the position of the maximum absolute value
N	Number of elements to process Can be any integer between 2 and 4096.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pOutVal target: If POP1 = 1, it is a pointer-on-pointer otherwise it is an ABMP or IBP pointer depending on its value. pOutIdx target: If POP2 = 1, it is a pointer-on-pointer otherwise it is an ABMP or IBP pointer depending on its value. Refer to Section 2.2.6: IOTYPE description for details

See Table 61. Format for generic direct commands with three pointers when VECT_ABSMAX_F32 is executed as a direct command.

5.4.3 VECT_ADD_F32

The VECT_ADD_F32 can be either executed as a direct command (accelerator mode) or programmed into a processing list.

VECT_ADD_F32 performs the addition of two real vectors:

$$pOut[n] = pInA[n] + pInB[n]$$

with n = 0, 1, 2,..., N-1

5.4.3.1 Command format to record VECT_ADD_F32

Table 65. Command format to record VECT_ADD_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	VECT_ADD_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP	Pointer on input data buffer A
HSP_PARAMR1	pInB	F32*	ABMP, POP	Pointer on input data buffer B
HSP_PARAMR2	pOut	F32*	ABMP, POP	Pointer on output data buffer
HSP_PARAMR3	N	U32	-	Number of elements to process
HSP_PARAMR[14:4]	-	-	-	-
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data buffer A Input buffer A is an array of real numbers in float32 representing input samples. pInA can be the same as pInB or pOut.
pInB	Pointer on input data buffer B Input buffer B is an array of real numbers in float32 representing input samples. pInB can be the same as pInA or pOut.
pOut	Pointer on output data buffer The result is a real vector in float32. pOut can be the same as pInA or pInB
N	Number of elements to process Can be any integer between 2 and 4096.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pInB target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pOut target: If POP2 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. Rrefer to Section 2.2.6: IOTYPE description for details

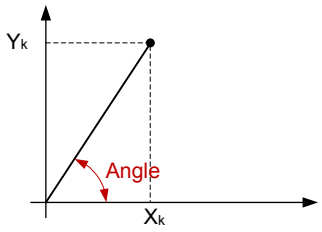
See Table 61. Format for generic direct commands with three pointers when VECT_ADD_F32 is executed as a direct command.

5.4.4 VECT_ATAN2_F32

The VECT_ATAN2_F32 can be either executed as a direct command (accelerator mode) or programmed into a processing list.

This function performs the calculation of arc tangent2 of an input vector containing X_k points and an input vector containing the Y_k points, and returns a vector containing the angles between the horizontal axis, and the vector defined by X_k, Y_k.

Figure 20. VECT_ATAN2_F32 overview



DT77861V1

5.4.4.1 *Command format to record VECT_ATAN2_F32*

Table 66. Command format to record VECT_ATAN2_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	VECT_ATAN2_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP	Pointer on input data buffer A (X_k)
HSP_PARAMR1	pInB	F32*	ABMP, POP	Pointer on input data buffer B (Y_k)
HSP_PARAMR2	pOut	F32*	ABMP, POP	Pointer on output data buffer
HSP_PARAMR3	N	U32	-	Number of elements to process
HSP_PARAMR[14:4]	-	-	-	-
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data buffer containing X_k Input buffer A is an array of real numbers in float32 representing input samples. pInA can be the same as pOut.
pInB	Pointer on input data buffer containing Y_k Input buffer B is an array of real numbers in float32 representing input samples. pInB can be the same as pOut.
pOut	The output buffer contains the angle in radian of each (X_k, Y_k) in float32. pOut can be the same as pInA or pInB
N	Number of elements to process Can be any integer between 2 and 4096.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pInB target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pOut target: If POP2 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer.

	Refer to Section 2.2.6: IOTYPE description for details

See Table 61. Format for generic direct commands with three pointers when VECT_ATAN2_F32 is executed as a direct command.

5.4.5 VECT_AVG_F32

The VECT_AVG_F32 can be either executed as a direct command (accelerator mode) or programmed into a processing list.

This function computes the average value of a real vector:

$$pOut = \frac{1}{N} \times \sum_{n=0}^{N-1} pInA[n]$$

5.4.5.1 *Command format to record VECT_AVG_F32*

Table 67. Command format to record VECT_AVG_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	VECT_AVG_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP	Pointer on input data buffer
HSP_PARAMR1	pOut	F32*	ABMP, POP, IBP	Pointer on output data buffer
HSP_PARAMR2	N	U32	-	Number of elements to process
HSP_PARAMR[14:3]	-	-	-	-
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data buffer Input buffer is an array of real numbers in float32 representing input samples. pIn can be the same as pOut.
pOut	Pointer on output data buffer The result is a real scalar in float32. pOut can be the same as pInA.
N	Number of elements to process Can be any integer between 2 and 4096.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pOut target: If POP1 = 1, it is a pointer-on-pointer otherwise it is an ABMP or IBP pointer depending on its value. Refer to Section 2.2.6: IOTYPE description for details

See Table 62. Format for generic direct commands with two pointers when VECT_AVG_F32 is executed as a direct command.

5.4.6 VECT_COPY

The VECT_COPY can be either executed as a direct command (accelerator mode) or programmed into a processing list.

This function performs a copy of the vector source pointed by pInA to the vector destination, pOut:

$$pOut[n] = pInA[n]$$

with n = 0, 1, 2,..., N-1

5.4.6.1 *Command format to record VECT_COPY*

Table 68. Command format to record VECT_COPY

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	VECT_COPY	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	X32*	ABMP, POP	Pointer on source data buffer A
HSP_PARAMR1	pOut	X32*	ABMP, POP	Pointer on destination data buffer
HSP_PARAMR2	N	U32	-	Number of elements to process
HSP_PARAMR[14:3]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on the input source data buffer A Must contain a real vector in float32 or (u)int32_t.
pOut	Pointer on the output destination data buffer The result is a real vector in float32 or (u)int32_t.
N	Number of elements to process Can be any integer between 2 and 4096.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pOut target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. Refer to Section 2.2.6: IOTYPE description for details

See Table 62. Format for generic direct commands with two pointers when VECT_COPY is executed as a direct command.

5.4.7 VECT_COS_F32

The VECT_COS_F32 can be either executed as a direct command (accelerator mode) or programmed into a processing list.

This function performs the cosine of each element of a vector.

The implementation is based on table lookup using 4096 values together with a linear interpolation. The overall precision is approximately 1e-6.

$$pOut[n] = \cos(pInA[n])$$

with n = 0, 1, 2,..., N-1

5.4.7.1 *Command format to record VECT_COS_F32*

Table 69. Command format to record VECT_COS_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	VECT_COS_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP	Pointer on input data buffer A

Registers	Acronyms	Type	Target	Description
HSP_PARAMR1	pOut	F32*	ABMP, POP	Pointer on output data buffer
HSP_PARAMR2	N	U32	-	Number of elements to process
HSP_PARAMR[14:3]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on the input data buffer A Must contain a real vector in float32 representing angles in radians. No range limitation. pInA can be the same as pOut.
pOut	Pointer on output data buffer The result is a real vector in float32. pInA can be the same as pOut.
N	Number of elements to process Can be any integer between 2 and 4096.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pOut target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. Rrefer to Section 2.2.6: IOTYPE description for details

See Table 62. Format for generic direct commands with two pointers when VECT_COS_F32 is executed as a direct command.

5.4.8 VECT_DIV_F32

The VECT_DIV_F32 can be either executed as a direct command (accelerator mode) or programmed into a processing list.

This function performs the element related division of two real vectors:

$$pOut[n] = \frac{pInA[n]}{pInB[n]}$$

with n = 0, 1, 2,..., N-1

Division by 0 sets the FPUERRF to 1. According to the HSP programming, the FPUERRF flag can generate an interrupt to the CPU, freeze the HSP execution, or generate a break signal for timers. A division by 0 operation returns ‘infinity’ or ‘-infinity’ number as described in IEEE-754.

5.4.8.1 Command format to record VECT_DIV_F32

Table 70. Command format to record VECT_DIV_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	VECT_DIV_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP	Pointer on input data buffer A (Num.)
HSP_PARAMR1	pInB	F32*	ABMP, POP	Pointer on input data buffer B (Den.)
HSP_PARAMR2	pOut	F32*	ABMP, POP	Pointer on output data buffer
HSP_PARAMR3	N	U32	-	Number of elements to process
HSP_PARAMR[14:4]	-	-	-	Not used

Registers	Acronyms	Type	Target	Description
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data buffer A (Numerator) Input buffer A is an array of real numbers in float32 representing input samples. pInA can be the same as pOut.
pInB	Pointer on input data buffer B (denominator) Input buffer A is an array of real numbers in float32 representing input samples. pInB can be the same as pOut.
pOut	Pointer on output data buffer The result is a real vector in float32. pOut can be the same as pInA or pInB
N	Number of elements to process Can be any integer between 2 and 4096.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pInB target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pOut target: If POP2 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. Refer to Section 2.2.6: IOTYPE description for details

See Table 61. Format for generic direct commands with three pointers when VECT_DIV_F32 is executed as a direct command.

5.4.9 VECT_DOTP_F32

The VECT_DOTP_F32 can be either executed as a direct command (accelerator mode) or programmed into a processing list.

This function performs the dot product of two real vectors:

$$pOut = \sum_{n=0}^{N-1} pInA[n] \times pInB[n]$$

5.4.9.1 Command format to record VECT_DOTP_F32

Table 71. Command format to record VECT_DOTP_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	VECT_DOTP_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP	Pointer on input data buffer A
HSP_PARAMR1	pInB	F32*	ABMP, POP	Pointer on input data buffer B
HSP_PARAMR2	pOut	F32*	ABMP, POP, IBP	Pointer on output data buffer
HSP_PARAMR3	N	U32	-	Number of elements to process
HSP_PARAMR[14:4]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data buffer A Input buffer A is an array of real numbers in float32 representing input samples. pInA can be the same as pInB.
pInB	Pointer on input data buffer B Input buffer A is an array of real numbers in float32 representing input samples. pInB can be the same as pInA.
pOut	Pointer on output data buffer The result is a real value in float32.
N	Number of elements to process Can be any integer between 2 and 4096.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pInB target: If POP1 = 1, it is a pointer-on-pointer otherwise it is an ABMP or IBP pointer depending on its value. pOut target: If POP2 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. Refer to Section 2.2.6: IOTYPE description for details

See Table 61. Format for generic direct commands with three pointers when VECT_DOTP_F32 is executed as a direct command.

5.4.10 VECT_DEC

The VECT_DEC can be either executed as a direct command (accelerator mode) or programmed into a processing list.

This function performs the decimation of a real vector:

$$pOut[n] = pInA[k]$$

with:

k = 0, DEC, 2.DEC, 3.DEC,...,(N-1).DEC

n = 0, 1, 2,..., N-1

5.4.10.1 Command format to record VECT_DEC

Table 72. Command format to record VECT_DEC

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	VECT_DEC	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	X32*	ABMP, POP	Pointer on input data buffer A
HSP_PARAMR1	DEC	U32	-	Decimation ratio
HSP_PARAMR2	pOut	X32*	ABMP, POP	Pointer on output data buffer
HSP_PARAMR3	N	U32	-	Number of output elements
HSP_PARAMR[14:4]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data buffer A Input buffer A is an array of real numbers in float32 representing input samples. pInA can be the same as pOut.
DEC	Decimation ratio The output buffer contains the following data: pOut[0] = pInA[0], pOut[1] = pInA[DEC],..., pOut[N-1] = pInA[(N-1)xDEC] DEC can be any integer between 2 and 255. The size of pInA must be at least equal to N x DEC.
pOut	Pointer on output data buffer The result is a real vector in float32. pOut can be the same as pInA or pInB
N	Number of output elements Can be any integer between 2 and 2048. N must be lower than or equal to the size of pOut.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pOut target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. Refer to Section 2.2.6: IOTYPE description for details

5.4.10.2 *VECT_DEC direct command format*

The next table shows the command format when VECT_DEC is executed as a direct command.

Table 73. VECT_DEC direct command format

Order	Registers	Acronyms	Type	Target	Description
1	HSP_DCMDIDR	VECT_DEC	U32	-	See Section 8: Function ID list
2	HSP_PARAMR1	DEC	U32	-	See description above
3	HSP_PARAMR3	N	U32	-	
6	HSP_DCMDPTR0	pInA	X32*	ABMP	
7	HSP_DCMDPTR1	pOut	X32*	ABMP	

1. *Parameters must be written in the specified order.*
2. *Refer to Section 5: Processing functions description for acronym details*

5.4.11 VECT_EXP_F32

The VECT_EXP_F32 can be either executed as a direct command (accelerator mode) or programmed into a processing list.

This function performs the calculation of exponential of each input element.

$$pOut[n] = exp^{(pInA[n])}$$

with n = 0, 1, 2,..., N-1

5.4.11.1 *Command format to record VECT_EXP_F32*

Table 74. Command format to record VECT_EXP_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	VECT_EXP_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP	Pointer on input data buffer A

Registers	Acronyms	Type	Target	Description
HSP_PARAMR1	pOut	F32*	ABMP, POP	Pointer on output data buffer
HSP_PARAMR2	N	U32	-	Number of elements to process
HSP_PARAMR[14:3]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data buffer A Input buffer A is an array of real numbers in float32 representing input samples. pInA can be the same as pOut.
pOut	Pointer on output data buffer The result is a real vector in float32. pInA can be the same as pOut.
N	Number of elements to process Can be any integer between 2 and 4096.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pOut target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. Refer to Section 2.2.6: IOTYPE description for details

See Table 62. Format for generic direct commands with two pointers when VECT_EXP_F32 is executed as a direct command.

5.4.12 VECT_EXP10_F32

The VECT_EXP10_F32 can be either executed as a direct command (accelerator mode) or programmed into a processing list.

This function performs the power of 10 of all elements of a vector.

$$pOut[n] = 10^{pInA[n]}$$

with n = 0, 1, 2,..., N-1

5.4.12.1 Command format to record VECT_EXP10_F32

Table 75. Command format to record VECT_EXP10_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	VECT_EXP10_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP	Pointer on input data buffer A
HSP_PARAMR1	pOut	F32*	ABMP, POP	Pointer on output data buffer
HSP_PARAMR2	N	U32	-	Number of elements to process
HSP_PARAMR[14:3]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data buffer A Input buffer A is an array of real numbers in float32 representing input samples. pInA can be the same as pOut.
pOut	Pointer on output data buffer The result is a real vector in float32. pInA can be the same as pOut.
N	Number of elements to process Can be any integer between 2 and 4096.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pOut target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. Refer to Section 2.2.6: IOTYPE description for details

See Table 62. Format for generic direct commands with two pointers when VECT_EXP10_F32 is executed as a direct command.

5.4.13 VECT_FTOI

The VECT_FTOI can be either executed as a direct command (accelerator mode) or programmed into a processing list.

VECT_FTOI performs the element related conversion of float32 into signed 32-bit integer.

The float32 number must be between -2^{31} and $2^{31} - 1$.

The result is rounded to the nearest integer value. It is up to the application to ensure that converted values are in the correct range.

If the float number is bigger than what is supported by the output type, the output value is set and an interrupt to the CPU can be generated. The OVFLOW flag is located into the HSP_ERRINFR register.

See also Section 5.4.14: VECT_FTOU, Section 5.4.15: VECT_ITOF, Section 5.4.36: VECT_UTOF, Section 5.4.16: VECT_I24TOF, Section 5.4.17: VECT_Q31TOF, Section 5.4.19: VECT_FTOQ31

5.4.13.1 Command format to record VECT_FTOI

Table 76. Command format to record VECT_FTOI

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	VECT_FTOI	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP	Pointer on input data buffer A
HSP_PARAMR1	pOut	I32*	ABMP, POP	Pointer on output data buffer
HSP_PARAMR2	N	U32	-	Number of elements to process
HSP_PARAMR[14:3]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data buffer A Input buffer A is an array of real numbers in float32 representing input samples. pInA can be the same as pOut.
pOut	Pointer on the output destination data buffer The result is an integer vector in int32_t. pOut can be the same as pInA.

N	Number of elements to process Can be any integer between 2 and 4096.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pOut target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. Refer to Section 2.2.6: IOTYPE description for details

See Table 62. Format for generic direct commands with two pointers when VECT_FTOI is executed as a direct command.

5.4.14 VECT_FTOU

The VECT_FTOU can be either executed as a direct command (accelerator mode) or programmed into a processing list.

This function performs the element related conversion of float32 into unsigned 32-bit.

Note: The float32 number must be between 0 and 2^{32} - 1.

The result is rounded to the nearest integer value. It is up to the application to insure that converted values are in the correct range.

If the float number is negative, the integer value is set to 0 and the INEXACT flag is set. This flag is located into the HSP_ERRINFR register.

If the float number is greater than what is supported by the output type, the output value is set and an interrupt to the CPU may be generated. The OVFLOW flag is located into the HSP_ERRINFR register.

See also Section 5.4.13: VECT_FTOI, Section 5.4.15: VECT_ITOF, Section 5.4.36: VECT_UTOF.

5.4.14.1 Command format to record VECT_FTOU

Table 77. Command format to record VECT_FTOU

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	VECT_FTOU	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP	Pointer on input data buffer A
HSP_PARAMR1	pOut	U32*	ABMP, POP	Pointer on output data buffer
HSP_PARAMR2	N	U32	-	Number of elements to process
HSP_PARAMR[14:3]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data buffer A Input buffer A is an array of real numbers in float32 representing input samples. pInA can be the same as pOut.
pOut	Pointer on the output destination data buffer The result is an unsigned integer vector in uint32_t. pOut can be the same as pInA.
N	Number of elements to process Can be any integer between 2 and 4096.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer.

	pOut target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. Refer to Section 2.2.6: IOTYPE description for details

See Table 62. Format for generic direct commands with two pointers when VECT_FTOU is executed as a direct command.

5.4.15 VECT_ITOF

The VECT_ITOF can be either executed as a direct command (accelerator mode) or programmed into a processing list.

VECT_ITOF performs the element related conversion of 32-bit integer to float32. The float32 has approximately 24-bit resolution (excluding sign), thus converting an int32_t into float32 leads to a loss of precision.

See also Section 5.4.13: VECT_FTOI, Section 5.4.14: VECT_FTOU, Section 5.4.36: VECT_UTOF.

5.4.15.1 Command format to record VECT_ITOF

Table 78. Command format to record VECT_ITOF

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	VECT_ITOF	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	I32*	ABMP, POP	Pointer on input data buffer A
HSP_PARAMR1	pOut	F32*	ABMP, POP	Pointer on output data buffer
HSP_PARAMR2	N	U32	-	Number of elements to process
HSP_PARAMR[14:3]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data buffer A Must contain a vector of integers in int32_t. pInA can be the same as pOut.
pOut	Pointer on the output destination data buffer The result is a real vector in float32. pOut can be the same as pInA.
N	Number of elements to process Can be any integer between 2 and 4096.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pOut target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. Refer to Section 2.2.6: IOTYPE description for details

See Table 62. Format for generic direct commands with two pointers when VECT_ITOF is executed as a direct command.

5.4.16 VECT_I24TOF

The VECT_I24TOF can be either executed as a direct command (accelerator mode) or programmed into a processing list.

VECT_I24TOF performs the element related conversion of 24-bit signed integers to float32.

It is expected to find the sign bit at position 23.

See also Section 5.4.13: VECT_FTOI, Section 5.4.14: VECT_FTOU, Section 5.4.36: VECT_UTOF.

5.4.16.1 *Command format to record VECT_I24TOF*

Table 79. Command format to record VECT_I24TOF

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	VECT_I24TOF	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	I32*	ABMP, POP	Pointer on input data buffer A
HSP_PARAMR1	pOut	F32*	ABMP, POP	Pointer on output data buffer
HSP_PARAMR2	N	U32	-	Number of elements to process
HSP_PARAMR[14:3]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data buffer A Must contain a vector of integers having the sign bit at position 23. pInA can be the same as pOut.
pOut	Pointer on the output destination data buffer The result is a real vector in float32. pOut can be the same as pInA.
N	Number of elements to process Can be any integer between 2 and 4096.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pOut target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. Refer to Section 2.2.6: IOTYPE description for details

See Table 62. Format for generic direct commands with two pointers when VECT_I24TOF is executed as a direct command.

5.4.17 VECT_Q31TOF

The VECT_Q31TOF can be either executed as a direct command (accelerator mode) or programmed into a processing list.

VECT_Q31TOF performs the element related conversion of Q31 integers to float32 in the following way:

- Convert the number to float32
- Multiply the result by 2^-31

The result is floating-point numbers in the range of +1 and -1.

The float32 has approximately 24-bit resolution (excluding sign), thus converting a Q31 into float32 leads to a loss of precision.

See also VECT_FTOI, VECT_FTOU, VECT_UTOF.

5.4.17.1 *Command format to record VECT_Q31TOF*

Table 80. Command format to record VECT_Q31TOF

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	VECT_Q31TOF	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	Q31*	ABMP, POP	Pointer on input data buffer A

Registers	Acronyms	Type	Target	Description
HSP_PARAMR1	pOut	F32*	ABMP, POP	Pointer on output data buffer
HSP_PARAMR2	N	U32	-	Number of elements to process
HSP_PARAMR[14:3]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

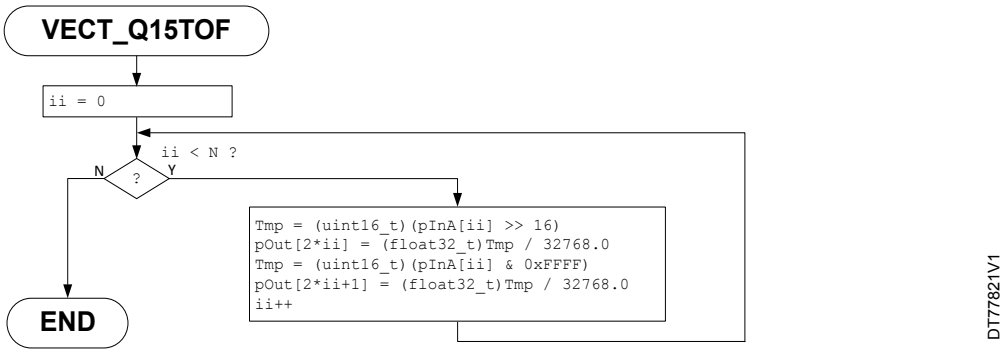
pInA	Pointer on input data buffer A Must contain a vector of Q31 numbers. pInA can be the same as pOut.
pOut	Pointer on the output destination data buffer The result is a real vector in float32. pOut can be the same as pInA.
N	Number of elements to process Can be any integer between 2 and 4096.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pOut target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. Refer to Section 2.2.6: IOTYPE description for details

See Table 62. Format for generic direct commands with two pointers when VECT_Q31TOF is executed as a direct command.

5.4.18 VECT_Q15TOF

The VECT_Q15TOF can be either executed as a direct command (accelerator mode) or programmed into a processing list. This processing function converts two Q15 numbers located into the same 32-bit location into two float32 contiguous numbers. VECT_Q15TOF performs the element related conversion of Q15 to float32 as detailed in Figure 21. VECT_Q15TOF process.

Figure 21. VECT_Q15TOF process



See also Section 5.4.13: VECT_FTOI, Section 5.4.14: VECT_FTOU, Section 5.4.36: VECT_UTOF.

5.4.18.1 *Command format to record CMPLX_Q15TOF*

Table 81. Command format to record CMPLX_Q15TOF

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	CMPLX_Q15TOF	U32	-	See Section 8: Function ID list

Registers	Acronyms	Type	Target	Description
HSP_PARAMR0	pInA	U32*	ABMP, POP	Pointer on input data buffer A
HSP_PARAMR1	pOut	F32*	ABMP, POP	Pointer on output data buffer
HSP_PARAMR2	N	U32	-	Number of elements to process
HSP_PARAMR[14:3]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data buffer A Each 32-bit element contains two Q15 numbers.
pOut	Pointer on the output destination data buffer Must contain a vector of float32 numbers
N	Number of complex inputs: Can be any number between 2 and 2048
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pOut target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. Refer to Section 2.2.6: IOTYPE description for details

See Table 62. Format for generic direct commands with two pointers when VECT_Q15TOF is executed as a direct command.

5.4.19 VECT_FTOQ31

The VECT_FTOQ31 can be either executed as a direct command (accelerator mode) or programmed into a processing list.

VECT_FTOQ31 performs the element related conversion of float32 to Q31 as follows:

$$pOut[n] = (Cast \quad to \quad Q31)pInA[n] \times 2^{31}$$

with n = 0, 1, 2,..., N-1

The saturation logic limits the output range to [0x80000000: 0x7FFFFFFF].

See also VECT_FTOI, VECT_FTOU, VECT_UTOF.

5.4.19.1 Command format to record VECT_FTOQ31

Table 82. Command format to record VECT_FTOQ31

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	VECT_FTOQ31	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP	Pointer on input data buffer A
HSP_PARAMR1	pOut	Q31*	ABMP, POP	Pointer on output data buffer
HSP_PARAMR2	N	U32	-	Number of elements to process
HSP_PARAMR[14:3]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data buffer A Must contain a vector of float32 numbers. pInA can be the same as pOut.
pOut	Pointer on the output destination data buffer The result is a real vector in Q31. pOut can be the same as pInA.
N	Number of elements to process Can be any integer between 2 and 4096.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pOut target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. Refer to Section 2.2.6: IOTYPE description for details

See Table 62. Format for generic direct commands with two pointers when VECT_FTOQ31 is executed as a direct command.

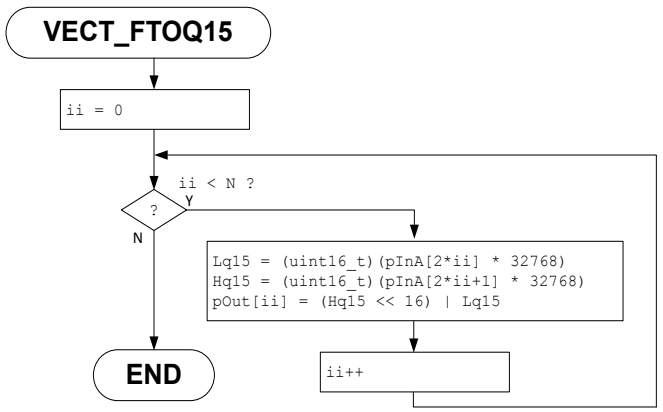
5.4.20 VECT_FTOQ15

The VECT_FTOQ15 can be either executed as a direct command (accelerator mode) or programmed into a processing list.

This processing function converts two float32 contiguous numbers into two Q15 numbers located into the same 32-bit location.

VECT_FTOQ15 performs the element related conversion of float32 to Q15 as detailed in Figure 22.

Figure 22. VECT_FTOQ15 process



There is no saturation logic. Numbers are simply truncated to 16 bits.

See also VECT_FTOI, VECT_FTOU, VECT_UTOF.

5.4.20.1 *Command format to record VECT_FTOQ15*

Table 83. Command format to record VECT_FTOQ15

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	VECT_FTOQ15	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP	Pointer on input data buffer A
HSP_PARAMR1	pOut	U32*	ABMP, POP	Pointer on output data buffer
HSP_PARAMR2	N	U32	-	Number of elements to process

Registers	Acronyms	Type	Target	Description
HSP_PARAMR[14:3]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data buffer A Must contain a vector of float32 complex numbers. pInA can be the same as pOut.
pOut	Pointer on the output destination data buffer The result is a complex vector in Q15. Each 32-bit element contains two Q15 numbers. pOut can be the same as pInA.
N	Number of elements to process Can be any even number between 2 and 4096.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pOut target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. Refer to Section 2.2.6: IOTYPE description for details

See Table 62. Format for generic direct commands with two pointers when VECT_FTOQ15 is executed as a direct command.

5.4.21 VECT_LN_F32

The VECT_LN_F32 can be either executed as a direct command (accelerator mode) or programmed into a processing list.

This function performs the calculation of natural logarithm of each input element. All input elements must be positive.

If a negative value is found, the functions output NaN (Not a Number), the flag FPUERRF is set and an interrupt can be generated if the FPUERRIE bit is set.

$$pOut[n] = In(pInA[n])$$

with n = 0, 1, 2,..., N-1

5.4.21.1 Command format to record VECT_LN_F32

Table 84. Command format to record VECT_LN_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	VECT_LN_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP	Pointer on input data buffer A
HSP_PARAMR1	pOut	F32*	ABMP, POP	Pointer on output data buffer
HSP_PARAMR2	N	U32	-	Number of elements to process
HSP_PARAMR[14:3]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data buffer A Must contain a vector of positive real in float32. pInA can be the same as pOut.
pOut	Pointer on output data buffer The result is a real vector in float32. pInA can be the same as pOut.
N	Number of elements to process Can be any integer between 2 and 4096.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pOut target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. Refer to Section 2.2.6: IOTYPE description for details

See Table 62. Format for generic direct commands with two pointers when VECT_LN_F32 is executed as a direct command.

5.4.22 VECT_LOG10_F32

The VECT_LOG10_F32 can be either executed as a direct command (accelerator mode) or programmed into a processing list.

This function performs the calculation of decimal logarithm (10-based logarithm) of each input element. All input elements must be positive.

If a negative value is found, the functions output NaN (Not a Number), the flag FPUERRF is set and an interrupt can be generated if the FPUERRIE bit is set.

$$pOut[n] = \log10(pInA[n])$$

with n = 0, 1, 2,..., N-1

5.4.22.1 Command format to record VECT_LOG10_F32

Table 85. Command format to record VECT_LOG10_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	VECT_LOG10_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP	Pointer on input data buffer A
HSP_PARAMR1	pOut	F32*	ABMP, POP	Pointer on output data buffer
HSP_PARAMR2	N	U32	-	Number of elements to process
HSP_PARAMR[14:3]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data buffer A Must contain a vector of positive real in float32. pInA can be the same as pOut.
pOut	Pointer on output data buffer The result is a real vector in float32. pInA can be the same as pOut.
N	Number of elements to process Can be any integer between 2 and 4096.
IOTYPE	Input and output buffer types: Indicates the pointers type

	pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pOut target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. Refer to Section 2.2.6: IOTYPE description for details

See Table 62. Format for generic direct commands with two pointers when VECT_LOG10_F32 is executed as a direct command.

5.4.23 VECT_MAX_F32

The VECT_MAX_F32 can be either executed as a direct command (accelerator mode) or programmed into a processing list.

This function searches for the value and position of the greatest element of a vector.

5.4.23.1 Command format to record VECT_MAX_F32

Table 86. Command format to record VECT_MAX_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	VECT_MAX_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP	Pointer on input data buffer A
HSP_PARAMR1	pOutVal	F32*	ABMP, POP, IBP	Pointer on maximum value
HSP_PARAMR2	pOutIdx	U32*	ABMP, POP, IBP	Pointer on position of maximum value
HSP_PARAMR3	N	U32	-	Number of elements to process
HSP_PARAMR[14:4]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data buffer A Input buffer A is an array of real numbers in float32 representing input samples.
pOutVal	Pointer on maximum value Must contain a real number in float32
pOutIdx	Pointer on position of maximum value The result is a number in uint32_t.
N	Number of elements to process Can be any integer between 2 and 4096.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pOutVal target: If POP1 = 1, it is a pointer-on-pointer otherwise it is an ABMP or IBP pointer depending on its value. pOutIdx target: If POP2 = 1, it is a pointer-on-pointer otherwise it is an ABMP or IBP pointer depending on its value. Refer to Section 2.2.6: IOTYPE description for details

5.4.23.2 VECT_MAX_F32 direct command format

The next table shows the command format when VECT_MAX_F32 is executed as a direct command.

Table 87. VECT_MAX_F32 direct command format

Order	Registers	Acronyms	Type	Target	Description
1	HSP_DCMDIDR	VECT_MAX_F32	U32	-	See Section 8: Function ID list
2	HSP_PARAMR3	N	U32	-	See description above
3	HSP_DCMDPTR0	pInA	F32*	ABMP	
4	HSP_DCMDPTR1	pOutVal	F32*	ABMP	
5	HSP_DCMDPTR2	pOutIdx	U32*	ABMP	

1. *Parameters must be written in the specified order.*
2. *Refer to Section 5: Processing functions description for acronym details*

5.4.24 VECT_MIN_F32

The VECT_MIN_F32 can be either executed as a direct command (accelerator mode) or programmed into a processing list.

This function searches for the value and position of the lowest element of a vector.

5.4.24.1 Command format to record VECT_MIN_F32

Table 88. Command format to record VECT_MIN_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	VECT_MIN_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP	Pointer to input data buffer A
HSP_PARAMR1	pOutVal	F32*	ABMP, POP, IBP	Pointer to minimum value
HSP_PARAMR2	pOutIdx	U32*	ABMP, POP, IBP	Pointer to position of minimum value
HSP_PARAMR3	N	U32	-	Number of elements to process
HSP_PARAMR[14:4]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer to input data buffer A Input buffer A is an array of real numbers in float32 representing input samples.
pOutVal	Pointer to minimum value Must contain a real number in float32
pOutIdx	Pointer to position of minimum value The result is a number in uint32_t.
N	Number of elements to process Can be any integer between 2 and 4096.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pOutVal target: If POP1 = 1, it is a pointer-on-pointer otherwise it is an ABMP or IBP pointer depending on its value. pOutIdx target: If POP2 = 1, it is a pointer-on-pointer otherwise it is an ABMP or IBP pointer depending on its value. Refer to Section 2.2.6: IOTYPE description for details

5.4.24.2 *VECT_MIN_F32 direct command format*

The next table shows the command format when VECT_MIN_F32 is executed as a direct command.

Table 89. VECT_MIN_F32 direct command format

Order	Registers	Acronyms	Type	Target	Description
1	HSP_DCMDIDR	VECT_MIN_F32	U32	-	See Section 8: Function ID list
2	HSP_PARAMR3	N	U32	-	See description above
3	HSP_DCMDPTR0	pInA	F32*	ABMP	
4	HSP_DCMDPTR1	pOutVal	F32*	ABMP	
5	HSP_DCMDPTR2	pOutIdx	U32*	ABMP	

1. *Parameters must be written in the specified order.*
2. *Refer to Section 5: Processing functions description for acronym details*

5.4.25 VECT_MUL_F32

The VECT_MUL_F32 can be either executed as a direct command (accelerator mode) or programmed into a processing list.

This function performs the element related product of two real vectors:

$$pOut[n] = pInA[n] \times pInB[n]$$

with n = 0, 1, 2,..., N-1

5.4.25.1 *Command format to record VECT_MUL_F32*

Table 90. Command format to record VECT_MUL_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	VECT_MUL_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP	Pointer on input data buffer A
HSP_PARAMR1	pInB	F32*	ABMP, POP	Pointer on input data buffer B
HSP_PARAMR2	pOut	F32*	ABMP, POP	Pointer on output data buffer
HSP_PARAMR3	N	U32	-	Number of elements to process
HSP_PARAMR[14:4]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data buffer A Input buffer A is an array of real numbers in float32 representing input samples. pInA can be the same as pInB or pOut.
pInB	Pointer on input data buffer B Input buffer A is an array of real numbers in float32 representing input samples. pInB can be the same as pInA or pOut.
pOut	Pointer on output data buffer The result is a real vector in float32. pOut can be the same as pInA or pInB
N	Number of elements to process Can be any integer between 2 and 4096.
IOTYPE	Input and output buffer types: Indicates the pointers type

pInA target:

If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer.

pInB target:

If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer.

pOut target:

If POP2 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer.

Refer to Section 2.2.6: IOTYPE description for details

See Table 61. Format for generic direct commands with three pointers when VECT_MUL_F32 is executed as a direct command.

5.4.26 VECT_MULCOS_F32

The VECT_MULCOS_F32 can be either executed as a direct command (accelerator mode) or programmed into a processing list.

This function performs the product of each element of a real vector and a cosine signal. This function takes benefit of the cosine function already implemented into the DROM.

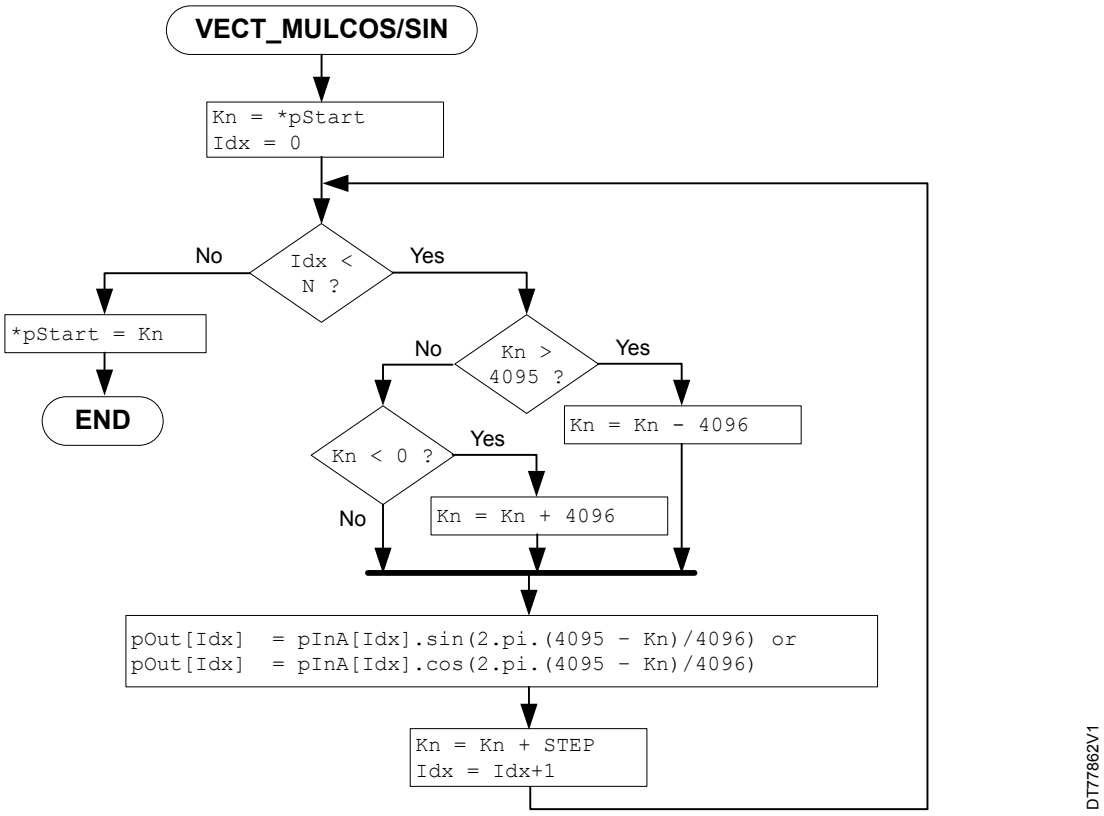
$$pOut[n] = pInA[n] \cdot \cos\left(\frac{2\pi(4095 - k_n)}{4096}\right)$$

With n = 0, 1, 2,..., N-1

The k_n values are computed as follows:

- k_0 = Start,
- $k_1 = (k_0 + \text{Step}) \bmod 4095$
- ...
- $k_n = (k_{n-1} + \text{Step}) \bmod 4095$

Figure 23. **VECT_MULCOS_F32/SIN overview**



As shown in Figure 23, the data ROM table of the HSP contains a sine and cosine reference signal of 4096 points. The horizontal axis represents the entry point of the data ROM.

The index 0 corresponds to the angle of 0 radian, and the index 4095 corresponds to the angle:

$$\frac{2 \times \pi \times 4095}{4096} radians$$

5.4.26.1 *Command format to record VECT_MULCOS_F32*

Table 91. **Command format to record VECT_MULCOS_F32**

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	VECT_MULCOS_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP	Pointer on input data buffer A
HSP_PARAMR1	pStart	I32*	ABMP, POP	Pointer on the start index variable
HSP_PARAMR2	pOut	F32*	ABMP, POP	Pointer on output data buffer
HSP_PARAMR3	N	U32	-	Number of elements to process
HSP_PARAMR4	STEP	I32	-	Increment
HSP_PARAMR[14:5]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data buffer A Input buffer A is an array of real numbers in float32 representing input samples. pInA can be the same as pInB or pOut.
pStart	Pointer on the start index variable It is a 32-bit integer containing the start index value of the cosine signal. This value is used to compute k_n values. At the end of the function processing, this variable is updated with the next k_n value.
pOut	Pointer on output data buffer The result is a real vector in float32. pOut can be the same as pInA or pInB
N	Number of elements to process Can be any integer between 2 and 4096.
STEP	Increment Used to compute k_n values, it can be a positive or negative integer value
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pStart target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pOut target: If POP2 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. Refer to Section 2.2.6: IOTYPE description for details

5.4.26.2 *VECT_MULCOS_F32 direct command format*

The next table shows the command format when VECT_MULCOS_F32 is executed as a direct command.

Table 92. VECT_MULCOS_F32 direct command format

Order	Registers	Acronyms	Type	Target	Description
1	HSP_DCMDIDR	VECT_MULCOS_F32	U32	-	See Section 8: Function ID list
2	HSP_PARAMR3	N	U32	-	See description above
3	HSP_PARAMR4	STEP	I32	-	
4	HSP_DCMDPTR0	pInA	F32*	ABMP	
5	HSP_DCMDPTR1	pStart	I32*	ABMP	
6	HSP_DCMDPTR2	pOut	F32*	ABMP	

1. *Parameters must be written in the specified order.*
2. *Refer to Section 5: Processing functions description for acronym details*

5.4.27 VECT_MULSIN_F32

The VECT_MULSIN_F32 can be either executed as a direct command (accelerator mode) or programmed into a processing list.

This function performs the product of each element of a real vector and a sinus signal. This function takes benefit of the sinus function already implemented into the DROM.

$$pOut[n] = pInA[n] \cdot \sin\left(\frac{2\pi(4095 - k_n)}{4096}\right)$$

With n = 0, 1, 2,..., N-1

The k_n values are computed as follows:

- k_0 = Start,
- $k_1 = (k_0+Step) \bmod 4095$
- ...
- $k_n = (k_{n-1} + Step) \bmod 4095$

See details on Section 5.4.26: VECT_MULCOS_F32.

5.4.27.1 *Command format to record VECT_MULSIN_F32*

Table 93. Command format to record VECT_MULSIN_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	VECT_MULSIN_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP	Pointer on input data buffer A
HSP_PARAMR1	pStart	I32*	ABMP, POP	Pointer on the start index variable
HSP_PARAMR2	pOut	F32*	ABMP, POP	Pointer on output data buffer
HSP_PARAMR3	N	U32	-	Number of elements to process
HSP_PARAMR4	STEP	I32	-	Increment
HSP_PARAMR[14:5]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data buffer A Input buffer A is an array of real numbers in float32 representing input samples. pInA can be the same as pInB or pOut.
pStart	Pointer on the start index variable It is a 32-bit integer containing the start index value of the cosine signal. This value is used to compute k_n values. At the end of the function processing, this variable is updated with the next k_n value.
pOut	Pointer on output data buffer The result is a real vector in float32. pOut can be the same as pInA or pInB
N	Number of elements to process Can be any integer between 2 and 4096.
STEP	Increment Used to compute k_n values, it can be a positive or negative integer value
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pStart target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pOut target: If POP2 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. Refer to Section 2.2.6: IOTYPE description for details

5.4.27.2 *VECT_MULSIN_F32 direct command format*

The next table shows the command format when VECT_MULSIN_F32 is executed as a direct command.

Table 94. VECT_MULSIN_F32 direct command format

Order	Registers	Acronyms	Type	Target	Description
1	HSP_DCMDIDR	VECT_MULSIN_F32	U32	-	See Section 8: Function ID list
2	HSP_PARAMR3	N	U32	-	See description above
3	HSP_PARAMR4	STEP	I32	-	
4	HSP_DCMDPTR0	pInA	F32*	ABMP	
5	HSP_DCMDPTR1	pStart	I32*	ABMP	
6	HSP_DCMDPTR2	pOut	F32*	ABMP	

1. *Parameters must be written in the specified order.*
2. *Refer to Section 5: Processing functions description for acronym details*

5.4.28 VECT_OFFSET_F32

The VECT_OFFSET_F32 can be either executed as a direct command (accelerator mode) or programmed into a processing list.

This function adds to each real vector element the offset value pointed by pOff.

The following operation is performed:

$$pOut[n] = pInA[n] + pOff$$

with n = 0, 1,...N-1

5.4.28.1 *Command format to record VECT_OFFSET_F32*

Table 95. Command format to record VECT_OFFSET_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	VECT_OFFSET_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP	Pointer on input data buffer A
HSP_PARAMR1	pOff	F32* F32	ABMP, POP, IBP, IMM	Pointer or immediate offset value
HSP_PARAMR2	pOut	F32*	ABMP, POP	Pointer on output data buffer
HSP_PARAMR3	N	U32	-	Number of elements to process
HSP_PARAMR[14:4]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data buffer A Input buffer A is an array of real numbers in float32 representing input samples. pInA can be the same as pInB or pOut.
pOff	Pointer on the offset value, or immediate offset value If IMM = 1, it must contain an immediate float32 value otherwise it must contain a pointer identifying a float32 variable.
pOut	Pointer on output data buffer The result is a real vector in float32. pOut can be the same as pInA or pInB
N	Number of elements to process Can be any integer between 2 and 4096.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target:

	If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pOff target: If POP1 = 1, it is a pointer-on-pointer if IMM = 1 it is an immediate value, otherwise it is an ABMP or IBP pointer depending on its value. pOut target: If POP2 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. Refer to Section 2.2.6: IOTYPE description for details

See Table 61. Format for generic direct commands with three pointers when VECT_OFFSET_F32 is executed as a direct command.

5.4.29 VECT_RMS_F32

The VECT_RMS_F32 can be either executed as a direct command (accelerator mode) or programmed into a processing list.

This function computes the root mean square (RMS) value of a real vector.

The following operation is performed:

$$pOut = \sqrt{\frac{1}{N} \cdot \left(pInA[0]^2 + pInA[1]^2 + \ldots + pInA[N-1]^2\right)}$$

5.4.29.1 Command format to record VECT_RMS_F32

Table 96. Command format to record VECT_RMS_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	VECT_RMS_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP	Pointer on input data buffer A
HSP_PARAMR1	pOut	F32*	ABMP, POP, IBP	Pointer on output data buffer
HSP_PARAMR2	N	U32	-	Number of elements to process
HSP_PARAMR[14:3]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data buffer A Must contain a vector of positive real in float32. pInA can be the same as pOut.
pOut	Pointer on the output data buffer The result is a real number in float32.
N	Number of elements to process Can be any integer between 2 and 4096.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is an ABMP pointer. pOut target: If POP1 = 1, it is a pointer-on-pointer otherwise it is an ABMP or IBP pointer depending on its value. Refer to Section 2.2.6: IOTYPE description for details

See Table 62. Format for generic direct commands with two pointers when VECT_RMS_F32 is executed as a direct command.

5.4.30 VECT_SIN_F32

The VECT_SIN_F32 can be either executed as a direct command (accelerator mode) or programmed into a processing list.

The implementation is based on table lookup using 4096 values together with a linear interpolation. The overall precision is approximately 1e-6.

This function computes the sine of each element of a vector.

$$pOut[n] = \sin(pInA[n])$$

with n = 0, 1, 2,..., N-1

5.4.30.1 *Command format to record VECT_SIN_F32*

Table 97. Command format to record VECT_SIN_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	VECT_SIN_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP	Pointer on input data buffer A
HSP_PARAMR1	pOut	F32*	ABMP, POP	Pointer on output data buffer
HSP_PARAMR2	N	U32	-	Number of elements to process
HSP_PARAMR[14:3]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on the input data buffer A Must contain a real vector in float32 representing angles in radians. No range limitation. pInA can be the same as pOut.
pOut	Pointer on output data buffer The result is a real vector in float32. pInA can be the same as pOut.
N	Number of elements to process Can be any integer between 2 and 4096.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is an ABMP pointer. pOut target: If POP1 = 1, it is a pointer-on-pointer otherwise it is an ABMP pointer. Refer to Section 2.2.6: IOTYPE description for details

See Table 62. Format for generic direct commands with two pointers when VECT_SIN_F32 is executed as a direct command.

5.4.31 VECT_SINCOS_F32

The VECT_SINCOS_F32 can be either executed as a direct command (accelerator mode) or programmed into a processing list.

This function computes the sine of each element of a vector.

The implementation is based on table lookup using 4096 values together with a linear interpolation. The overall precision is approximately 1e-6.

Tthe output buffer contains 2xN elements.

$$pOut[2n] = \cos(pInA[n])$$

$$pOut[2n+1] = \sin(pInA[n])$$

with n = 0, 1, 2,..., N-1

5.4.31.1 Command format to record VECT_SINCOS_F32

Table 98. Command format to record VECT_SINCOS_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	VECT_SINCOS_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP	Pointer on input data buffer A
HSP_PARAMR1	pOut	F32*	ABMP, POP	Pointer on output data buffer
HSP_PARAMR2	N	U32	-	Number of elements to process
HSP_PARAMR[14:3]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on the input data buffer A Must contain a real vector in float32 representing angles in radians. No range limitation. pInA can be the same as pOut.
pOut	Pointer on output data buffer The result is a real vector of 2N float32 elements. SeeSection 5.4.31.2: VECT_SINCOS_F32 pOut format.
N	Number of elements to process Can be any integer between 2 and 2048
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is an ABMP pointer. pOut target: If POP1 = 1, it is a pointer-on-pointer otherwise it is an ABMP pointer. Refer to Section 2.2.6: IOTYPE description for details

See Table 62. Format for generic direct commands with two pointers when VECT_SINCOS_F32 is executed as a direct command.
The format of this pOut is the following:

5.4.31.2 VECT_SINCOS_F32 pOut format

Table 99. VECT_SINCOS_F32 pOut format

Offset addr.	Acronyms	Description	Dir	Data type
0	cosine(pInA(0))	Computed cosine value	O	F32
1	sine(pInA(0))	Computed sine value	O	F32
2	cosine(pInA(1))	Computed cosine value	O	F32
3	sine(pInA(1))	Computed sine value	O	F32
...	...	...	O	F32
2N-2	cosine(pInA(N-1))	Computed cosine value	O	F32

Offset addr.	Acronyms	Description	Dir	Data type
2N-1	sine(pInA(N-1))	Computed sine value	O	F32

1. *I stands for input buffer, not modified by the processing function, O stands for output buffer, modified by the processing function, and IO stands for input/output buffer, modified by the processing function.*
2. *U8 stands for 8-bit unsigned integer, U16 stands for 16-bit unsigned integer, U32 stands for 32-bit unsigned integer, and F32 stands for 32-bit floating-point.*

5.4.32 VECT_SCALE_F32

The VECT_SCALE_F32 can be either executed as a direct command (accelerator mode) or programmed into a processing list.

This function multiplies each real vector element by the scale value pointed by pScale.

The following operation is performed:

$$pOut[n] = pInA[n] \cdot pScale$$

with n = 0, 1, 2,..., N-1

5.4.32.1 Command format to record VECT_SCALE_F32

Table 100. Command format to record VECT_SCALE_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	VECT_SCALE_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP	Pointer on input data buffer A
HSP_PARAMR1	pScale	F32*	ABMP, POP, IMM, IBP	Pointer or immediate scale value
HSP_PARAMR2	pOut	F32*	ABMP, POP	Pointer on output data buffer
HSP_PARAMR3	N	U32	-	Number of elements to process
HSP_PARAMR[14:4]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data buffer A Input buffer A is an array of real numbers in float32 representing input samples. pInA can be the same as pInB or pOut.
pScale	Pointer on the scale value or immediate value Scale factor to be multiplied with the N first elements of pInA. If IMM = 1, it must contain an immediate float32 value otherwise it must contain a pointer identifying a float32 variable.
pOut	Pointer on output data buffer The result is a real vector in float32. pOut can be the same as pInA or pInB
N	Number of elements to process Can be any integer between 2 and 4096.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pScale target: If POP1 = 1, it is a pointer-on-pointer if IMM = 1 it is an immediate value, otherwise it is an ABMP or IBP pointer depending on its value. pOut target:

	If POP2 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. Refer to Section 2.2.6: IOTYPE description for details

See Table 61. Format for generic direct commands with three pointers when VECT_SCALE_F32 is executed as a direct command.

5.4.33 VECT_SET

The VECT_SET can be either executed as a direct command (accelerator mode) or programmed into a processing list.

This function sets each element of a real vector to the value contained into pSetVal.

The following operation is performed:

$$pOut[n] = pSetVal$$

with n = 0, 1, 2,..., N-1

5.4.33.1 Command format to record VECT_SET

Table 101. Command format to record VECT_SET

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	VECT_SET	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pSetVal	X32* X32	ABMP, POP, IMM IBP,	Pointer or immediate on input set value
HSP_PARAMR1	pOut	F32*	ABMP, POP	Pointer on output data buffer
HSP_PARAMR2	N	U32	-	Number of elements to process
HSP_PARAMR[14:3]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pSetVal	Pointer or immediate set value Value to set. If IMM = 1, it must contain an immediate float32 value otherwise it must contain a pointer identifying a float32 variable.
pOut	Pointer on output data buffer The result is a real vector in float32.
N	Number of elements to process Can be any integer between 2 and 4096.
IOTYPE	Input and output buffer types: Indicates the pointers type pSetVal target: If POP0 = 1, it is a pointer-on-pointer if IMM = 1 it is an immediate value, otherwise it is an ABMP or IBP pointer depending on its value. pOut target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. Refer to Section 2.2.6: IOTYPE description for details

See Table 62. Format for generic direct commands with two pointers when VECT_SET is executed as a direct command.

5.4.34 VECT_SQRT_F32

The VECT_SQRT_F32 can be either executed as a direct command (accelerator mode) or programmed into a processing list.

This function performs the element related square-root of a real vector.

Doing the square root of a negative number sets the FPUERRF to 1. According to the HSP programming, the FPUERRF flag can generate an interrupt to the CPU, freeze the HSP execution, or generate a break signal for timers. The square root of a negative number returns NaN (Not a Number) value.

The following operation is performed:

$$pOut[n] = \sqrt{pInA[n]}$$

with n = 0, 1, 2,..., N-1

5.4.34.1 *Command format to record VECT_SQRT_F32*

Table 102. Command format to record VECT_SQRT_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	VECT_SQRT_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP	Pointer on input data buffer A
HSP_PARAMR1	pOut	F32*	ABMP, POP	Pointer on output data buffer
HSP_PARAMR2	N	U32	-	Number of elements to process
HSP_PARAMR[14:3]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data buffer A Must contain a float32 real vector. All elements must be positive or equal to 0. pInA can be the same as pOut.
pOut	Pointer on output data buffer The result is a real vector in float32. pOut can be the same as pInA.
N	Number of elements to process Can be any integer between 2 and 4096.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pOut target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. Refer to Section 2.2.6: IOTYPE description for details

See Table 62. Format for generic direct commands with two pointers when VECT_SQRT_F32 is executed as a direct command.

5.4.35 VECT_SUB_F32

The VECT_SUB_F32 can be either executed as a direct command (accelerator mode) or programmed into a processing list.

This function performs the element related subtraction of two real vectors.

The following operation is performed:

$$pOut[n] = pInA[n] - pInB[n]$$

with n = 0, 1,...N-1

5.4.35.1 *Command format to record VECT_SUB_F32*

Table 103. Command format to record VECT_SUB_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	VECT_SUB_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP	Pointer on input data buffer A
HSP_PARAMR1	pInB	F32*	ABMP, POP	Pointer on input data buffer B
HSP_PARAMR2	pOut	F32*	ABMP, POP	Pointer on output data buffer
HSP_PARAMR3	N	U32	-	Number of elements to process
HSP_PARAMR[14:4]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data buffer A Input buffer A is an array of real numbers in float32 representing input samples. pInA can be the same as pInB or pOut.
pInB	Pointer on input data buffer B Input buffer A is an array of real numbers in float32 representing input samples. pInB can be the same as pInA or pOut.
pOut	Pointer on output data buffer The result is a real vector in float32. pOut can be the same as pInA or pInB
N	Number of elements to process Can be any integer between 2 and 4096.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pInB target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pOut target: If POP2 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. Refer to Section 2.2.6: IOTYPE description for details

See Table 61. Format for generic direct commands with three pointers when VECT_MUL_F32 is executed as a direct command.

5.4.36 VECT_UTOF

The VECT_UTOF can be either executed as a direct command (accelerator mode) or programmed into a processing list.

VECT_UTOF performs the element related conversion of 32-bit unsigned integer into float32.

The float32 has approximately 24-bit resolution (excluding sign), thus converting an uint32_t into float32 leads to a loss of precision.

See also Section 5.4.13: VECT_FTOI, Section 5.4.14: VECT_FTOU, Section 5.4.15: VECT_ITOF.

5.4.36.1 *Command format to record VECT_UTOF*

Table 104. Command format to record VECT_UTOF

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	VECT_UTOF	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	U32*	ABMP, POP	Pointer on input data buffer A
HSP_PARAMR1	pOut	F32*	ABMP, POP	Pointer on output data buffer
HSP_PARAMR2	N	U32	-	Number of elements to process
HSP_PARAMR[14:3]	-	-	-	Not used

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data buffer A Must contain a vector of integers in uint32_t. pInA can be the same as pOut.
pOut	Pointer on the output destination data buffer The result is an vector of real in float32. pOut can be the same as pInA.
N	Number of elements to process Can be any integer between 2 and 4096.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pOut target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. Refer to Section 2.2.6: IOTYPE description for details

See Table 62. Format for generic direct commands with two pointers when VECT_UTOF is executed as a direct command.

5.4.37 VECT_ZINS

The VECT_ZINS can be either executed as a direct command (accelerator mode) or programmed into a processing list.

This function performs zero insertion of a real vector.

Note: It is up to the application to insure that the output buffer does not exceed 4096 elements.

5.4.37.1 *Command format to record VECT_ZINS*

Table 105. Command format to record VECT_ZINS

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	VECT_ZINS	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	X32*	ABMP, POP	Pointer on input data buffer A
HSP_PARAMR1	ZNB	U32	-	Number of inserted zeros
HSP_PARAMR2	pOut	X32*	ABMP, POP	Pointer on output data buffer
HSP_PARAMR3	N	U32	-	Number of input elements
HSP_PARAMR[14:4]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data buffer A Must contain a vector having element size of 32 bits.
ZNB	Number of inserted zeros ZNB zeros are inserted between each element of the input buffer Can be any integer between 1 and 255.
pOut	Pointer on output data buffer The result is a vector.
N	Number of input elements Can be any integer between 2 and 4096/ZNB.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pOut target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. Refer to Section 2.2.6: IOTYPE description for details

5.4.37.2 *VECT_ZINS direct command format*

The next table shows the command format when VECT_ZINS is executed as a direct command.

Table 106. VECT_ZINS direct command format

Order	Registers	Acronyms	Type	Target	Description
1	HSP_DCMDIDR	VECT_ZINS	U32	-	See Section 8: Function ID list
2	HSP_PARAMR1	ZNB	U32	-	See description above
3	HSP_PARAMR3	N	U32	-	
6	HSP_DCMDPTR0	pInA	F32*	ABMP	
7	HSP_DCMDPTR1	pOut	F32*	ABMP	

1. *Parameters must be written in the specified order.*
2. *Refer to Section 5: Processing functions description for acronym details*

5.5 Functions operating on complex vectors

The functions operating on complex vectors support complex scalar vectors and vectors having the real and imaginary part interleaved, refer to Section 2.2.1: Vector, matrix operations, and formats for details.

The function ID of each function is provided on Section 8: Function ID list.

5.5.1 CMPLX_CONJ_F32

The CMPLX_CONJ_F32 can be either executed as a direct command (accelerator mode) or programmed into a processing list.

This function performs the conjugate of a scalar or of each element of a complex vector.

$$pOut[n] = Conj(pInA[n])$$

with n = 0, 1,...N-1

5.5.1.1 *Command format to record CMPLX_CONJ_F32*

Table 107. Command format to record CMPLX_CONJ_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	CMPLX_CONJ_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP	Pointer on complex input data buffer A
HSP_PARAMR1	pOut	F32*	ABMP, POP	Pointer on output data buffer
HSP_PARAMR2	NWRD	U32	-	Number of words
HSP_PARAMR[14:3]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on the input data buffer A Must contain a float32 complex vector or a complex number. pInA can be the same as pOut.
pOut	Pointer on the output data buffer Must contain a float32 complex vector or a complex number. pOut can be the same as pInA.
NWRD	Number of 32-bit words Defines the number of input words to process. A complex number represents 2 words. Can be any integer between 2 and 8192.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pOut target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. Refer to Section 2.2.6: IOTYPE description for details

See Table 62. Format for generic direct commands with two pointers when CMPLX_CONJ_F32 is executed as a direct command.

5.5.2 CMPLX_DOTP_F32

The CMPLX_DOTP_F32 can be either executed as a direct command (accelerator mode) or programmed into a processing list.

This function performs the sum of the product of two scalars or of each element of two complex vectors.

The following operation is performed:

$$pOut = \sum_{n=0}^{N-1} pInA[n] \times pInB[n]$$

with n = 0, 1,...N-1, and pInA, pInB and pOut complex.

5.5.2.1 *Command format to record CMPLX_DOTP_F32*

Table 108. Command format to record CMPLX_DOTP_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	CMPLX_DOTP_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP	Pointer on complex input data buffer A
HSP_PARAMR1	pInB	F32*	ABMP, POP	Pointer on complex input data buffer B
HSP_PARAMR2	pOut	F32*	ABMP, POP	Pointer on output data buffer
HSP_PARAMR3	NWRD	U32	-	Number of words
HSP_PARAMR[14:4]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on the input data buffer A Must contain a float32 complex vector or a complex number. pInA can be the same as pInB.
pInB	Pointer on the input data buffer B Must contain a float32 complex vector or a complex number. pInB can be the same as pInA
pOut	Pointer on the output data buffer The result is a complex number in float32.
NWRD	Number of 32-bit words Defines the number of input words to process. A complex number represents 2 words. Can be any integer between 2 and 8192.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pInB target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pOut target: If POP2 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. Refer to Section 2.2.6: IOTYPE description for details

See Table 61. Format for generic direct commands with three pointers when CMPLX_DOTP_F32 is executed as a direct command.

5.5.3 CMPLX_MAG_F32

The CMPLX_MAG_F32 can be either executed as a direct command (accelerator mode) or programmed into a processing list.

Computes the magnitude of a scalar or of each element of the input vector.

The following operation is performed:

$$pOut = \sqrt{pInA[2n]^2 + pInA[2n+1]^2}$$

with n = 0, 1,...N-1, and pOut is real and pInA is complex.

5.5.3.1 *Command format to record CMPLX_MAG_F32*

Table 109. Command format to record CMPLX_MAG_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	CMPLX_MAG_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP	Pointer on complex input data buffer A
HSP_PARAMR1	pOut	F32*	ABMP, POP	Pointer on output data buffer
HSP_PARAMR2	NWRD	U32	-	Number of words
HSP_PARAMR[14:3]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on the input data buffer A Must contain a float32 complex vector or a complex number. pInA can be the same as pOut.
pOut	Pointer on the output data buffer Must contain a float32 real vector or a real number. pOut can be the same as pInA.
NWRD	Number of 32-bit words Defines the number of input words to process. A complex number represents 2 words. Can be any integer between 2 and 8192.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pOut target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. Refer to Section 2.2.6: IOTYPE description for details

See Table 62. Format for generic direct commands with two pointers when CMPLX_MAG_F32 is executed as a direct command.

Note: The output buffer (Out) is two times smaller than the size of the input buffer (InA).

Example:

With:

$$InA = [(a_0 + jb_0), \quad (a_1 + jb_1)]$$

then the operation:

pOut = **CMPLX_MAG_F32**(pInA[], 4)

Gives:

$$Out = \left[\sqrt{a_0^2 + b_0^2}, \quad \sqrt{a_1^2 + b_1^2}\right]$$

5.5.4 CMPLX_MAGSQR_F32

The CMPLX_MAGSQR_F32 can be either executed as a direct command (accelerator mode) or programmed into a processing list.

This function computes the magnitude squared of a scalar or of each element of the input vector.

The following operation is performed:

$$pOut[n] = InA[2n]^2 + InA[2n+1]^2$$

with n = 0, 1,...N-1, and Out real and InA complex.

5.5.4.1 *Command format to record CMPLX_MAGSQR_F32*

Table 110. Command format to record CMPLX_MAGSQR_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	CMPLX_MAGSQR_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP	Pointer on complex input data buffer A
HSP_PARAMR1	pOut	F32*	ABMP, POP	Pointer on output data buffer
HSP_PARAMR2	NWRD	U32	-	Number of input words
HSP_PARAMR[14:3]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on the input data buffer A Must contain a float32 complex vector or a complex number. pInA can be the same as pOut.
pOut	Pointer on the output data buffer Must contain a float32 real vector or a real number. pOut can be the same as pInA.
NWRD	Number of input words Defines the number of input words to process. A complex number represents 2 words of 32 bits. Can be any integer between 2 and 8192.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pOut target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. Refer to Section 2.2.6: IOTYPE description for details

See Table 62. Format for generic direct commands with two pointers when CMPLX_MAGSQR_F32 is executed as a direct command.

Note: The output buffer (Out) is two times smaller than the size of the input buffer (InA).

Example:

With:

$$InA = [(a_0 + jb_0), \quad (a_1 + jb_1)]$$

then the operation:

pOut = **CMPLX_MAGSQR_F32**(pInA[], 4)

Gives:

$$Out = \left[\left(a_0^2 + b_0^2\right), \quad \left(a_1^2 + b_1^2\right)\right]$$

5.5.5 CMPLX_MUL_F32

The CMPLX_MUL_F32 can be either executed as a direct command (accelerator mode) or programmed into a processing list.

This function computes the multiplication of two complex scalars or vector elements.

The following operation is performed:

$$pOut[n] = pInA[n] \times pInB[n]$$

with n = 0, 1,...N-1, and Out, InA and InB are complex.

5.5.5.1 *Command format to record CMPLX_MUL_F32*

Table 111. Command format to record CMPLX_MUL_F3

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	CMPLX_MUL_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP	Pointer on complex input data buffer A
HSP_PARAMR1	pInB	F32*	ABMP, POP	Pointer on complex input data buffer B
HSP_PARAMR2	pOut	F32*	ABMP, POP	Pointer on output data buffer
HSP_PARAMR3	NWRD	U32	-	Number of words
HSP_PARAMR[14:4]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on the input data buffer A Must contain a float32 complex vector or a complex number. pInA can be the same as pOut or pInB.
pInB	Pointer on the input data buffer B Must contain a float32 complex vector or a complex number. pInB can be the same as pOut or pInA
pOut	Pointer on the output data buffer Must contain a float32 real vector or a real number. pOut can be the same as pInA or pInB
NWRD	Number of 32-bit words Defines the number of input words to process. A complex number represents 2 words. Can be any integer between 2 and 8192.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pInB target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pOut target: If POP2 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. Refer to Section 2.2.6: IOTYPE description for details

See Table 61. Format for generic direct commands with three pointers when CMPLX_MUL_F32 is executed as a direct command.

5.5.6 CMPLX_RMUL_F32

The CMPLX_RMUL_F32 can be either executed as a direct command (accelerator mode) or programmed into a processing list.

This function computes the multiplication of a complex scalar or vector by a real scalar or vector.

The following operation is performed:

$$pOut[2n] = pInA[2n] \times pInB[n]$$

$$pOut[2n+1] = pInA[2n+1] \times pInB[n]$$

with n = 0, 1,...N-1, and Out, InA are complex and InB is real.

5.5.6.1 *Command format to record CMPLX_RMUL_F32*

Table 112. Command format to record CMPLX_RMUL_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	CMPLX_RMUL_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP	Pointer on complex input data buffer A
HSP_PARAMR1	pInB	F32*	ABMP, POP	Pointer on real input data buffer B
HSP_PARAMR2	pOut	F32*	ABMP, POP	Pointer on output data buffer
HSP_PARAMR3	NWRD	U32	-	Number of words
HSP_PARAMR[14:4]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on the input data buffer A Must contain a float32 complex vector or a complex number. pInA can be the same as pOut.
pInB	Pointer on input data buffer B Must contain a real vector or number in float32. pInB can be the same as pOut.
pOut	Pointer on the output data buffer Must contain a float32 real vector or a real number. pOut can be the same as pInA or pInB
NWRD	Number of 32-bit words Defines the number of input words to process. A complex number represents 2 words. Can be any integer between 2 and 8192.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pInB target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pOut target: If POP2 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. Refer to Section 2.2.6: IOTYPE description for details

See Table 61. Format for generic direct commands with three pointers when CMPLX_MUL_F32 is executed as a direct command.

5.5.7 CMPLX_MULEXP_F32

The command format to immediately run this processing function or program it into a processing list is similar. The only difference is the function ID written into HSP_C2HMSGDR register.

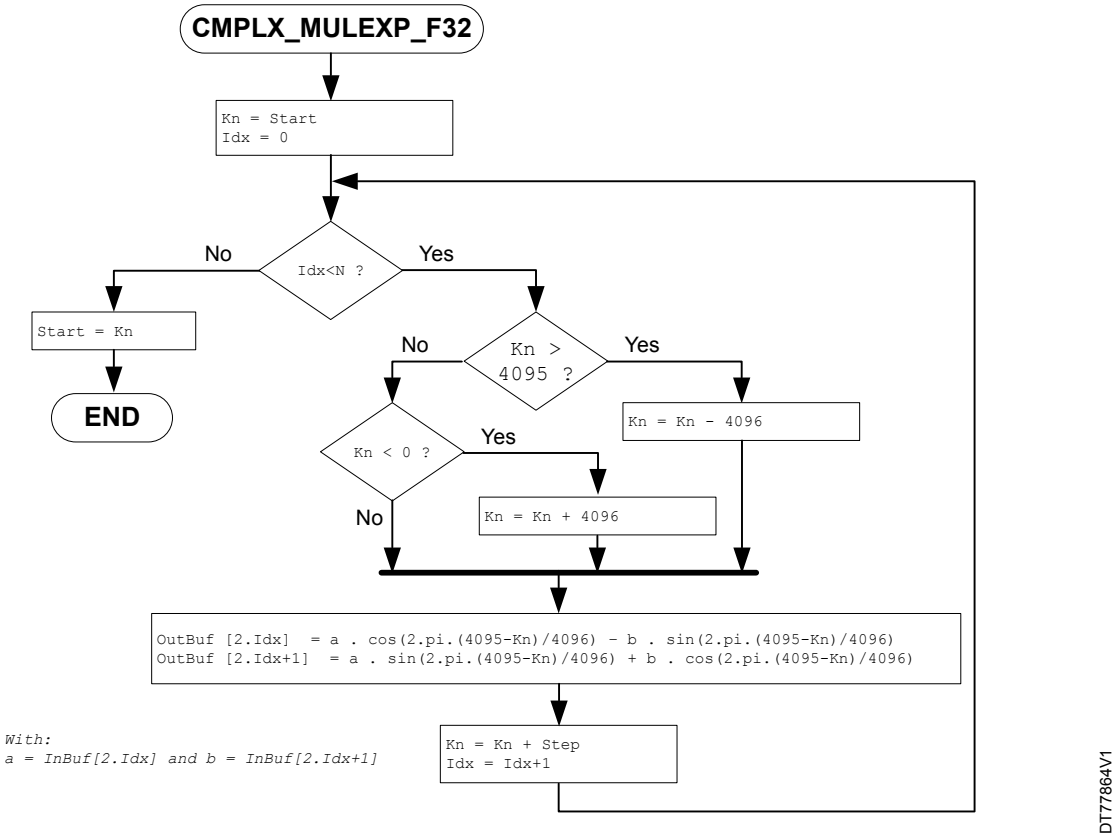
The CMPLX_MULEXP_F32 function performs the complex product of a scalar or of each element of a complex vector and an exponential complex signal. This function takes benefit of the exponential function already implemented into the DROM.

$$Out[n] = InA[n] \cdot e^{\frac{\pm j2\pi k_n}{4096}}$$

The k_n values are computed as follows:

- k_0 = Start,
- k_1 = (k_0+Step) mod 4095
- ...
- k_n = (k_{n-1} + Step) mod 4095

Figure 24. CMPLX_MULEXP_F32 overview



Refer to Section 5.4.26: VECT_MULCOS_F32 to have information about the reference signal located into the DROM.

5.5.7.1 *Command format to record CMPLX_MULEXP_F32*

Table 113. Command format to record CMPLX_MULEXP_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	CMPLX_MULEXP_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP	Pointer on complex input data buffer A
HSP_PARAMR1	pStart	I32*	ABMP, POP	Pointer on the start index variable
HSP_PARAMR2	pOut	F32*	ABMP, POP	Pointer on output data buffer
HSP_PARAMR3	NWRD	U32	-	Number of words to process
HSP_PARAMR4	STEP	I32	-	Increment
HSP_PARAMR[14:5]	-	-	-	Not used

Registers	Acronyms	Type	Target	Description
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data buffer A Input buffer A is an array of real numbers in float32 representing input samples. pInA can be the same as pOut.
pStart	Pointer on the start index variable It is a 32-bit integer containing the start index value of the complex exponential signal. This value is used to compute the k_n values. At the end of the function processing, this buffer is updated with the next k_n value.
pOut	Pointer on output data buffer The result is a complex vector in float32. pOut can be the same as pInA
NWRD	Number of 32-bit words Defines the number of input words to process. A complex number represents 2 words. Can be any integer between 2 and 8192.
STEP	Increment Used to compute k_n values, it can be a positive or negative integer value
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pStart target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pOut target: If POP2 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. Refer to Section 2.2.6: IOTYPE description for details

5.5.7.2 *CMPLX_MULEXP_F32 direct command format*

The next table shows the command format when CMPLX_MULEXP_F32 is executed as a direct command.

Table 114. CMPLX_MULEXP_F32 direct command format

Order	Registers	Acronyms	Type	Target	Description
1	HSP_DCMDIDR	CMPLX_MULEXP_F32	U32	-	See Section 8: Function ID list
2	HSP_PARAMR3	N	U32	-	See description above
3	HSP_PARAMR4	STEP	I32	-	
4	HSP_DCMDPTR0	pInA	F32*	ABMP	
5	HSP_DCMDPTR1	pStart	I32*	ABMP	
6	HSP_DCMDPTR2	pOut	F32*	ABMP	

1. *Parameters must be written in the specified order.*
2. *Refer to Section 5: Processing functions description for acronym details*

5.6 Scalar functions on real

The function ID of each function is provided in Section 8: Function ID list.

5.6.1 __SCA_ABS_F32

This in-line processing function can only be included into a processing list.

This function computes the absolute value of a real scalar.

$$pOut = |pInA|$$

5.6.1.1 *Command format to record __SCA_ABS_F32*

Table 115. Command format to record __SCA_ABS_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	__SCA_ABS_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	U32*	ABMP, IBP, POP	Pointer on input data A
HSP_PARAMR1	pOut	F32*	ABMP, IBP, POP	Pointer on output data
HSP_PARAMR[14:2]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data A The input data is a float32 real number. pInA can be the same as pOut.
pOut	Pointer on the output data The output is a float32 real number. pOut can be the same as pInA.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer, otherwise it is a ABMP or IBP pointer. pOut target: If POP1 = 1, it is a pointer-on-pointer, otherwise it is a ABMP or IBP pointer. Refer to Section 2.2.6: IOTYPE description for details

5.6.2 __SCA_ADD_F32

This in-line processing function can only be included into a processing list.

This function performs the addition of two real scalars.

$$pOut = pInA + pInB$$

5.6.2.1 *Command format to record __SCA_ADD_F32*

Table 116. Command format to record __SCA_ADD_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	__SCA_ADD_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, IBP, POP	Pointer on input data A
HSP_PARAMR1	pInB	F32* F32	ABMP, IBP, IMM, POP	Immediate or pointer on input data B
HSP_PARAMR2	pOut	F32*	ABMP, IBP, POP	Pointer on output data
HSP_PARAMR[14:3]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

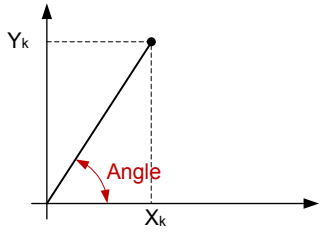
pInA	Pointer on input data A The input data is a float32 real number. pInA can be the same as pOut or pInB.
pInB	Immediate or pointer on input data B The input data is a float32 real number. pInB can be the same as pOut or pInA. If IMM = 1, it must contain an immediate float32 value otherwise it must contain a pointer identifying a float32 variable.
pOut	Pointer on the output data The result is a float32 real number. pOut can be the same as pInA or pInB.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer, otherwise it is a ABMP or IBP pointer. pInB target: If POP1 = 1, it is a pointer-on-pointer, if IMM = 1, it is an immediate value, otherwise it is a ABMP or IBP pointer. pOut target: If POP2 = 1, it is a pointer-on-pointer, otherwise it is a ABMP or IBP pointer. Refer to Section 2.2.6: IOTYPE description for details

5.6.3 SCA_ATAN2_F32

This processing function can only be included into a processing list.

This function performs the calculation of ATAN2 of input point (X, Y), and returns the angle between the horizontal axis, and the vector defined by X, Y. The overall precision is 1e-6.

Figure 25. ATAN2 overview



DT77861V1

5.6.3.1 *Command format to record SCA_ATAN2_F32*

Table 117. Command format to record SCA_ATAN2_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	SCA_ATAN2_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInX	F32*	ABMP, IBP	Pointer on input data X
HSP_PARAMR1	pInY	F32*	ABMP, IBP	Pointer on input data Y
HSP_PARAMR2	pOut	F32*	ABMP, IBP	Pointer on output data
HSP_PARAMR[14:3]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInX	Pointer on input data X The input data must be a float32 real number. pInX can be the same as pOut.
pInY	Pointer on input data Y The input data must be a float32 real number. pInY can be the same as pOut.

pOut	Pointer on the output data The result is the angle in radian of (X, Y) in float32. pOut can be the same as pInX or pInY.
IOTYPE	Input and output buffer types: Indicates the pointers type pInX target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP or IBP pointer. pInY target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP or IBP pointer. pOut target: If POP2 = 1, it is a pointer-on-pointer otherwise it is a ABMP or IBP pointer. Refer to Section 2.2.6: IOTYPE description for details

5.6.4 SCA_COS_F32

This processing function can only be included into a processing list.

This function performs the cosine of a float32 number.

The implementation is based on table lookup using 4096 values together with a linear interpolation. The overall precision is approximately 1e-6

5.6.4.1 Command format to record SCA_COS_F32

Table 118. Command format to record SCA_COS_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	SCA_COS_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP, IBP	Pointer on input data A
HSP_PARAMR1	pOut	F32*	ABMP, POP, IBP	Pointer on output data
HSP_PARAMR[14:2]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data A Must contain a float32 number representing an angle in radians. The input and output buffer pointers can be the same.
pOut	Pointer on output data The result is the cosine of the input angle. The input and output buffer pointers can be the same.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP or IBP pointer. pOut target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP or IBP pointer. Refer to Section 2.2.6: IOTYPE description for details

5.6.5 __SCA_DIV_F32

This in-line processing function can only be included into a processing list.

This function performs the division of two real scalars. A division by 0 sets the FPUERRF to 1. According to the HSP programming, the FPUERRF flag can generate an interrupt to the CPU, freeze the HSP execution, or generate a break signal for timers.

A division by 0 operation returns ‘infinity’ or ‘-infinity’ number as described in IEEE-754.

5.6.5.1 *Command format to record __SCA_DIV_F32*

Table 119. Command format to record __SCA_DIV_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	__SCA_DIV_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, IBP, POP	Pointer on input data A (Num.)
HSP_PARAMR1	pInB	F32* F32	ABMP, IBP, IMM, POP	Immediate or pointer on input data B (Den.)
HSP_PARAMR2	pOut	F32*	ABMP, IBP, POP	Pointer on output data
HSP_PARAMR[14:3]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data A (Numerator) Must contain a real float32 number. The inputs and output buffer pointers can be the same.
pInB	Immediate value or pointer on input data B (Denominator) Must contain a real float32 number. The inputs and output buffer pointers can be the same. If IMM = 1, it must contain an immediate float32 value otherwise it must contain a pointer identifying a float32 variable.
pOut	Pointer on output data buffer The result is a real number of float32. The inputs and output buffer pointers can be the same.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer, otherwise it is a ABMP or IBP pointer. pInB target: If POP1 = 1, it is a pointer-on-pointer, if IMM = 1, it is an immediate value, otherwise it is a ABMP or IBP pointer. pOut target: If POP2 = 1, it is a pointer-on-pointer, otherwise it is a ABMP or IBP pointer. Refer to Section 2.2.6: IOTYPE description for details

5.6.6 SCA_EXP_F32

This processing function can only be included into a processing list.

This function performs the calculation of the exponential of the input element.

$$pOut = exp^{(pInA)}$$

5.6.6.1 *Command format to record SCA_EXP_F32*

Table 120. Command format to record SCA_EXP_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	SCA_EXP_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP, IBP	Pointer on input data A

Registers	Acronyms	Type	Target	Description
HSP_PARAMR1	pOut	F32*	ABMP, POP, IBP	Pointer on output data
HSP_PARAMR[14:2]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data buffer A Must contain a real float32 number. The inputs and output buffer pointers can be the same.
pOut	Pointer on output data buffer The result is a real float32 number. The inputs and output buffer pointers can be the same.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP or IBP pointer. pOut target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP or IBP pointer. Refer to Section 2.2.6: IOTYPE description for details

5.6.7 SCA_EXP10_F32

This processing function can only be included into a processing list.
Perform the power of 10 of an input element.

$$pOut = 10^{pInA}$$

5.6.7.1 *Command format to record SCA_EXP10_F32*

Table 121. Command format to record SCA_EXP10_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	SCA_EXP10_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP, IBP	Pointer on input data A
HSP_PARAMR1	pOut	F32*	ABMP, POP, IBP	Pointer on output data
HSP_PARAMR[14:2]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data buffer A Must contain a real float32 number. The inputs and output buffer pointers can be the same.
pOut	Pointer on output data buffer The result is a real float32 number. The inputs and output buffer pointers can be the same.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP or IBP pointer.

	pOut target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP or IBP pointer. Refer to Section 2.2.6: IOTYPE description for details

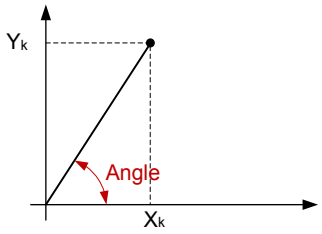
5.6.8 SCA_FATAN2_F32

This processing function can only be included into a processing list.

This function performs the calculation of Fast ATAN2 of input point (X, Y), and returns the angle between the horizontal axis, and the vector defined by X, Y.

SCA_FATAN2_F32 performs an efficient computation of atan2 with a precision of approximately 1e-3.

Figure 26. SCA_FATAN2_F32 overview



DT77861V1

X, Y must be in the range (TBD).

5.6.8.1 *Command format to record SCA_FATAN2_F32*

Table 122. Command format to record SCA_FATAN2_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	SCA_FATAN2_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInX	F32*	ABMP, POP, IBP	Pointer on input data X
HSP_PARAMR1	pInY	F32*	ABMP, POP, IBP	Pointer on input data Y
HSP_PARAMR2	pOut	F32*	ABMP, POP, IBP	Pointer on output data
HSP_PARAMR[14:3]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInX	Pointer on input data X The input data must be a float32 real number. pInX can be the same as pOut.
pInY	Pointer on input data Y The input data must be a float32 real number. pInY can be the same as pOut.
pOut	Pointer on the output data The result is the angle in radian of (X, Y) in float32. pOut can be the same as pInX or pInY.
IOTYPE	Input and output buffer types: Indicates the pointers type pInX target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP or IBP pointer. pInY target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP or IBP pointer. pOut target: If POP2 = 1, it is a pointer-on-pointer otherwise it is a ABMP or IBP pointer.

	Refer to Section 2.2.6: IOTYPE description for details

5.6.9 SCA_FCOS_F32

This processing function can only be included into a processing list.
It performs the cosine of a float32 number.
The implementation is based on table lookup using 4096 values. The overall precision is approximately 1e-3.

5.6.9.1 Command format to record SCA_FCOS_F32

Table 123. Command format to record SCA_FCOS_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	SCA_FCOS_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP, IBP	Pointer on input data A
HSP_PARAMR1	pOut	F32*	ABMP, POP, IBP	Pointer on output data
HSP_PARAMR[14:2]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data A Must contain a float32 number representing an angle in radians. The input and output buffer pointers can be the same.
pOut	Pointer on output data The result is the cosine of the input angle. The input and output buffer pointers can be the same.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP or IBP pointer. pOut target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP or IBP pointer. Refer to Section 2.2.6: IOTYPE description for details

5.6.10 SCA_FSIN_F32

This processing function can only be included into a processing list.
It performs the sine of a float32 number. The overall precision is approximately 1e-3.
The implementation is based on lookup table using 4096 values.

$$OutBuf = \sin(InBuf)$$

5.6.10.1 Command format to record SCA_FSIN_F32

Table 124. Command format to record SCA_FSIN_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	SCA_FSIN_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP, IBP	Pointer on input data A
HSP_PARAMR1	pOut	F32*	ABMP, POP, IBP	Pointer on output data
HSP_PARAMR[14:2]	-	-	-	Not used

Registers	Acronyms	Type	Target	Description
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data A Must contain a float32 number representing an angle in radians. The input and output buffer pointers can be the same.
pOut	Pointer on output data The result is the sine of the input angle. The input and output buffer pointers can be the same.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP or IBP pointer. pOut target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP or IBP pointer. Refer to Section 2.2.6: IOTYPE description for details

5.6.11 __SCA_FTOI

This in-line processing function can only be included into a processing list.

It converts a float32 into a signed 32-bit integer.

The float32 number must be between -2^{31} and $2^{31} - 1$.

It is up to the application to insure that converted value is in the correct range.

The result is rounded to the nearest integer value.

If the float number is greater than what is supported by the output type, the output value is set to the max or min value supported by the output data, and the OVERFLOW flag is set. This flag is located into the HSP_ERRINFR register. This error can also generate an interrupt.

See also Section 5.6.12: __SCA_FTOU, Section 5.6.14: __SCA_ITOF, Section 5.6.13: __SCA_UTOF.

5.6.11.1 Command format to record __SCA_FTOI

Table 125. Command format to record __SCA_FTOI

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	__SCA_FTOI	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP, IBP	Pointer on input data A
HSP_PARAMR1	pOut	I32*	ABMP, POP, IBP	Pointer on output data
HSP_PARAMR[14:2]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data A Must contain a float32 number. The inputs and output buffer pointers can be the same.
pOut	Pointer on output data The result is a number in int32_t format. The inputs and output buffer pointers can be the same.

IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer, otherwise it is a ABMP or IBP pointer. pOut target: If POP1 = 1, it is a pointer-on-pointer, otherwise it is a ABMP or IBP pointer. Refer to Section 2.2.6: IOTYPE description for details

5.6.12 __SCA_FTOU

This in-line processing function can only be included into a processing list.

It converts a float32 into an unsigned 32-bit integer.

The float32 number must be between 0 and 2^{32} - 1.

It is up to the application to insure that converted value is in the correct range.

The result is rounded to the nearest integer value.

If the float number is bigger than what is supported by the output type, the output value is set to the max or min value supported by the output data, and the OVERFLOW flag is set. This flag is located into the HSP_ERRINFR register. This float can also generate an interrupt.

See also Section 5.6.11: __SCA_FTOI, Section 5.6.14: __SCA_ITOF, Section 5.6.13: __SCA_UTOF.

5.6.12.1 Command format to record __SCA_FTOU

Table 126. Command format to record __SCA_FTOU

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	__SCA_FTOU	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP, IBP	Pointer on input data A
HSP_PARAMR1	pOut	U32*	ABMP, POP, IBP	Pointer on output data
HSP_PARAMR[14:2]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data A Must contain a float32 number. The inputs and output buffer pointers can be the same.
pOut	Pointer on output data The result is a number in uint32_t format. The inputs and output buffer pointers can be the same.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer, otherwise it is a ABMP or IBP pointer. pOut target: If POP1 = 1, it is a pointer-on-pointer, otherwise it is a ABMP or IBP pointer. Refer to Section 2.2.6: IOTYPE description for details

5.6.13 __SCA_UTOF

This processing function can only be included into a processing list.

It converts an unsigned 32-bit integer into float32.

The float32 has approximately 24-bit resolution (excluding sign), thus converting an uint32_t into float32 leads to a loss of precision.

See also Section 5.6.11: __SCA_FTOI, Section 5.6.12: __SCA_FTOU, Section 5.6.14: __SCA_ITOF.

5.6.13.1 *Command format to record __SCA_UTOF*

Table 127. Command format to record __SCA_UTOF

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	__SCA_UTOF	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	U32*	ABMP, POP, IBP	Pointer on input data A
HSP_PARAMR1	pOut	F32*	ABMP, POP, IBP	Pointer on output data
HSP_PARAMR[14:2]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data A Must contain a uint32_t number. The inputs and output buffer pointers can be the same.
pOut	Pointer on output data The result is a number in float32_t format. The inputs and output buffer pointers can be the same.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer, otherwise it is a ABMP or IBP pointer. pOut target: If POP1 = 1, it is a pointer-on-pointer, otherwise it is a ABMP or IBP pointer. Rfer to Section 2.2.6: IOTYPE description for details

5.6.14 __SCA_ITOF

This in-line processing function can only be included into a processing list.

It converts a signed 32-bit integer into float32.

The float32 has approximately 24-bit resolution (excluding sign), thus when converting an int32_t into float32 leads to a loss of precision.

See also Section 5.6.11: __SCA_FTOI, Section 5.6.12: __SCA_FTOU, Section 5.6.13: __SCA_UTOF.

5.6.14.1 *Command format to record __SCA_ITOF*

Table 128. Command format to record __SCA_ITOF

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	__SCA_ITOF	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	I32*	ABMP, POP, IBP	Pointer on input data A
HSP_PARAMR1	pOut	F32*	ABMP, POP, IBP	Pointer on output data
HSP_PARAMR[14:2]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data A Must contain a int32_t number. The inputs and output buffer pointers can be the same.
pOut	Pointer on output data The result is a number in float32_t format. The inputs and output buffer pointers can be the same.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer, otherwise it is a ABMP or IBP pointer. pOut target: If POP1 = 1, it is a pointer-on-pointer, otherwise it is a ABMP or IBP pointer. Refer to Section 2.2.6: IOTYPE description for details

5.6.15 __SCA_I24TOF

This in-line processing function can only be included into a processing list.
__SCA_I24TOF performs the element related conversion of 24-bit signed integer to float32.
It is expected to find the sign bit at position 23.
See also Section 5.4.13: VECT_FTOI, Section 5.4.14: VECT_FTOU, Section 5.4.36: VECT_UTOF.

5.6.15.1 Command format to record SCA_I24TOF

Table 129. Command format to record SCA_I24TOF

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	SCA_I24TOF	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	I32*	ABMP, POP, IBP	Pointer on input data A
HSP_PARAMR1	pOut	F32*	ABMP, POP, IBP	Pointer on output data
HSP_PARAMR[14:2]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data A Must contain an integer having the sign bit at position 23. The inputs and output buffer pointers can be the same.
pOut	Pointer on output data The result is a number in float32_t format. The inputs and output buffer pointers can be the same.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer, otherwise it is a ABMP or IBP pointer. pOut target: If POP1 = 1, it is a pointer-on-pointer, otherwise it is a ABMP or IBP pointer. Refer to Section 2.2.6: IOTYPE description for details

5.6.16 SCA_Q31TOF

The SCA_Q31TOF can be either executed as a direct command (accelerator mode) or programmed into a processing list.

SCA_Q31TOF performs the element related conversion of Q31 integers to float32. The results are floating numbers in the range of +1 and -1.

The float32 has approximately 24-bit resolution (excluding sign), thus converting a Q31 into float32 leads to a loss of precision.

See also Section 5.4.13: VECT_FTOI, Section 5.4.14: VECT_FTOU, Section 5.4.36: VECT_UTOF.

5.6.16.1 *Command format to record SCA_Q31TOF*

Table 130. Command format to record SCA_Q31TOF

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	SCA_Q31TOF	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	Q31*	ABMP, POP, IBP	Pointer on input data A
HSP_PARAMR1	pOut	F32*	ABMP, POP, IBP	Pointer on output data
HSP_PARAMR[14:2]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data A Must contain a Q31 real number. The inputs and output buffer pointers can be the same.
pOut	Pointer on output data The result is a real number in float32. The inputs and output buffer pointers can be the same.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP or IBP pointer. pOut target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP or IBP pointer. Refer to Section 2.2.6: IOTYPE description for details

5.6.17 SCA_FTOQ31

The SCA_FTOQ31 can be either executed as a direct command (accelerator mode) or programmed into a processing list.

SCA_FTOQ31 performs the element related conversion of float32 to Q31 as follows:

$$pOut[n] = (Cast \quad to \quad Q31)pInA[n] \times 2^{31}$$

with n = 0, 1, 2,..., N-1

The saturation logic limits the output range to [0x80000000: 0x7FFFFFFF].

See also Section 5.4.13: VECT_FTOI, Section 5.4.14: VECT_FTOU, Section 5.4.36: VECT_UTOF.

5.6.17.1 *Command format to record SCA_FTOQ31*

Table 131. Command format to record SCA_FTOQ31

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	SCA_FTOQ31	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP, IBP	Pointer on input data A

Registers	Acronyms	Type	Target	Description
HSP_PARAMR1	pOut	Q31*	ABMP, POP, IBP	Pointer on output data
HSP_PARAMR[14:2]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data A Must contain a float32 number The inputs and output buffer pointers can be the same.
pOut	Pointer on output data The result is a real number in Q31. The inputs and output buffer pointers can be the same.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP or IBP pointer. pOut target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP or IBP pointer. Refer to Section 2.2.6: IOTYPE description for details

5.6.18 SCA_LN_F32

This processing function can only be included into a processing list.

This function performs the calculation of natural logarithm of the input element.

The input element must be a positive number. Doing the natural logarithm of a negative number sets the FPUERRF to 1. According to the HSP programming, the FPUERRF flag can generate an interrupt to the CPU, freeze the HSP execution, or generate a break signal for timers.

$$pOut = In(pInA)$$

5.6.18.1 Command format to record SCA_LN_F32

Table 132. Command format to record SCA_LN_F3

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	SCA_LN_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP, IBP	Pointer on input data A
HSP_PARAMR1	pOut	F32*	ABMP, POP, IBP	Pointer on output data
HSP_PARAMR[14:2]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data A Must contain a positive float32 number. The inputs and output buffer pointers can be the same.
pOut	Pointer on output data The result is a number in float32 format. The inputs and output buffer pointers can be the same.
IOTYPE	Input and output buffer types: Indicates the pointers type

	pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP or IBP pointer. pOut target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP or IBP pointer. Refer to Section 2.2.6: IOTYPE description for details

5.6.19 SCA_LOG10_F32

This processing function can only be included into a processing list.

This function performs the calculation of decimal logarithm (10 based logarithm) of input element.

The input element must be a positive number. Doing the decimal logarithm of a negative number sets the FPUERRF to 1. According to the HSP programming, the FPUERRF flag can generate an interrupt to the CPU, freeze the HSP execution, or generate a break signal for timers.

$$pOut = Log10(pInA)$$

5.6.19.1 Command format to record SCA_LOG10_F32

Table 133. Command format to record SCA_LOG10_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	SCA_LOG10_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP, IBP	Pointer on input data A
HSP_PARAMR1	pOut	F32*	ABMP, POP, IBP	Pointer on output data
HSP_PARAMR[14:2]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data A Must contain a positive float32 number. The inputs and output buffer pointers can be the same.
pOut	Pointer on output data The result is a number in float32 format. The inputs and output buffer pointers can be the same.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP or IBP pointer. pOut target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP or IBP pointer. Refer to Section 2.2.6: IOTYPE description for details

5.6.20 __SCA_MAC_F32

This in-line processing function can only be included into a processing list.

It performs the multiplication-accumulation of float32 numbers.

$$pOut = pInA \times pInB + pOut$$

5.6.20.1 *Command format to record __SCA_MAC_F32*

Table 134. Command format to record __SCA_MAC_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	__SCA_MAC_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, IBP, POP	Pointer on input data A
HSP_PARAMR1	pInB	F32* F32	ABMP, IBP, IMM, POP	Pointer or immediate on input data B
HSP_PARAMR2	pOut	F32*	ABMP, IBP, POP	Pointer on output data
HSP_PARAMR[14:3]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data A Must contain a real float32 number. The inputs and output buffer pointers can be the same.
pInB	Immediate value or pointer on input data B Must contain a real float32 number. The inputs and output buffer pointers can be the same. If IMM = 1, it must contain an immediate float32 value otherwise it must contain a pointer identifying a float32 variable.
pOut	Pointer on output data The result is a number in float32 format. The inputs and output buffer pointers can be the same.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer, otherwise it is a ABMP or IBP pointer. pInB target: If POP1 = 1, it is a pointer-on-pointer, if IMM = 1, it is an immediate value, otherwise it is a ABMP or IBP pointer. pOut target: If POP2 = 1, it is a pointer-on-pointer, otherwise it is a ABMP or IBP pointer. Refer to Section 2.2.6: IOTYPE description for details

5.6.21 __SCA_MUL_F32

This in-line processing function can only be included into a processing list.
This function performs the multiplication of two real scalars.

$$pOut = pInA \times pInB$$

5.6.21.1 *Command format to record __SCA_MUL_F32*

Table 135. Command format to record __SCA_MUL_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	__SCA_MUL_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, IBP, POP	Pointer on input data A
HSP_PARAMR1	pInB	F32*	ABMP, IBP, IMM, POP	Pointer or immediate on input data B

Registers	Acronyms	Type	Target	Description
		F32		
HSP_PARAMR2	pOut	F32*	ABMP, IBP, POP	Pointer on output data
HSP_PARAMR[14:3]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data A Must contain a real float32 number. The inputs and output buffer pointers can be the same.
pInB	Immediate value or pointer on input data B Must contain a real float32 number. The inputs and output buffer pointers can be the same. If IMM = 1, it must contain an immediate float32 value otherwise it must contain a pointer identifying a float32 variable.
pOut	Pointer on output data The result is a number in float32 format. The inputs and output buffer pointers can be the same.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer, otherwise it is a ABMP or IBP pointer. pInB target: If POP1 = 1, it is a pointer-on-pointer, if IMM = 1, it is an immediate value, otherwise it is a ABMP or IBP pointer. pOut target: If POP2 = 1, it is a pointer-on-pointer, otherwise it is a ABMP or IBP pointer. Refer to Section 2.2.6: IOTYPE description for details

5.6.22 __SCA_NEG_F32

This in-line processing function can only be included into a processing list.
This function negates the input number:

$$pOut = -pInA$$

5.6.22.1 *Command format to record __SCA_NEG_F32*

Table 136. Command format to record __SCA_NEG_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	__SCA_NEG_F32	U32	-	See Section 8: Function ID list
-	-	-	-	-
HSP_PARAMR1	pInA	F32*	ABMP, IBP, POP	Pointer on input data
HSP_PARAMR2	pOut	F32*	ABMP, IBP, POP	Pointer on output data
HSP_PARAMR[14:3]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data A Must contain a real float32 number. The inputs and output buffer pointers can be the same.
pOut	Pointer on output data The result is a number in float32 format. The inputs and output buffer pointers can be the same.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer, otherwise it is a ABMP or IBP pointer. pOut target: If POP1 = 1, it is a pointer-on-pointer, otherwise it is a ABMP or IBP pointer. Refer to Section 2.2.6: IOTYPE description for details

5.6.23 __SCA_SAT_F32

This in-line processing function can only be included into a processing list.

This function performs a saturation of a float32 number using two programmable thresholds.

When a saturation is performed the flag FPUSATF of HSP_EVT_ISR register is set to 1. The FPUSATF flag stores the saturation status (**spe_fpu_sat**) returned by the function __SCA_SAT_F32. The flag FPUSATF is cleared by the application when the bit FPUSATC of the HSP_EVT_ICR register is set to 1.

The saturation status (**spe_fpu_sat**) is cleared every time a processing function starts to be executed but not when an in-line processing function starts to be executed.

The function __CLR_SATF must be used to clear the saturation status, before running the __SCA_SAT_F32 function if there was no opportunity to clear it.

*Note: __GET_SATF can be used to copy the value **spe_fpu_sat** into a variable.*

5.6.23.1 *Command format to record __SCA_SAT_F32*

Table 137. Command format to record __SCA_SAT_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	__SCA_SAT_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP, IBP	Pointer on input data A
HSP_PARAMR1	pThr	F32*	ABMP, POP	Pointer on thresholds (LoLim, HiLim)
HSP_PARAMR2	pOut	F32*	ABMP, POP, IBP	Pointer on output data
HSP_PARAMR[14:3]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data A Must contain a float32 number. The input and output buffer pointers can be the same.
pThr	Pointer on threshold values Must contain a float32 pointer on threshold buffer (see Table 137. Command format to record __SCA_SAT_F32 for data format).
pOut	Pointer on output data The result is the sine of the input angle. The input and output buffer pointers can be the same.

IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer, otherwise it is a ABMP or IBP pointer. pThr target: If POP1 = 1, it is a pointer-on-pointer, otherwise it is a ABMP pointer. pOut target: If POP2 = 1, it is a pointer-on-pointer, otherwise it is a ABMP or IBP pointer. Refer to Section 2.2.6: IOTYPE description for details

5.6.24 SCA_SIN_F32

This processing function can only be included into a processing list.
it performs the sine of a float32 number. The overall precision is approximately 1e-6.
The implementation is based on lookup table using 4096 values together with linear interpolation.

$$OutBuf = \sin(InBuf)$$

5.6.24.1 Command format to record SCA_SIN_F32

Table 138. Command format to record SCA_SIN_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	SCA_SIN_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP, IBP	Pointer on input data A
HSP_PARAMR1	pOut	F32*	ABMP, POP, IBP	Pointer on output data
HSP_PARAMR[14:2]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data A Must contain a float32 number representing an angle in radians. The input and output buffer pointers can be the same.
pOut	Pointer on output data The result is the sine of the input angle. The input and output buffer pointers can be the same.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP or IBP pointer. pOut target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP or IBP pointer. Refer to Section 2.2.6: IOTYPE description for details

5.6.25 SCA_SINCOS_F32

This processing function can only be included into a processing list.
It performs the sine and cosine of a float32 number. Overall precision is 1e-6.
The implementation is based on lookup table using 4096 values together with linear interpolation.

$$pOut = \sin \quad \cos(pInA)$$

5.6.25.1 *Command format to record SCA_SINCOS_F32*

Table 139. Command format to record SCA_SINCOS_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	SCA_SINCOS_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP, IBP	Pointer on input data A
HSP_PARAMR1	pOut	F32*	ABMP, POP, IBP	Pointer on output data
HSP_PARAMR[14:2]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data A Must contain a float32 number representing an angle in radians. The input and output buffer pointers can be the same.
pOut	Pointer on output data Contain the sine and the cosine of the angle given by the input buffer in float32. Refer to Section 5.6.25.2: OutBuf format for SCA_SINCOS_F32 for details.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP or IBP pointer. pOut target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP or IBP pointer. Refer to Section 2.2.6: IOTYPE description for details

5.6.25.2 *OutBuf format for SCA_SINCOS_F32*

Table 140. OutBuf format for SCA_SINCOS_F3

Offset addr.	Acronyms	Description	Data type
0	cosine	Computed cosine value	F32
4	sine	Computed sine value	F32

5.6.26 __SCA_SET

This in-line processing function can only be included into a processing list.
This function initializes a 32-bit data with a constant value.

5.6.26.1 *Command format to record __SCA_SET*

Table 141. Command format to record __SCA_SET

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	__SCA_SET	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pSetVal	X32* X32	ABMP, POP, IMM, IBP	Pointer or immediate on source data
HSP_PARAMR1	pOut	X32*	ABMP, POP, IBP	Pointer on destination data
HSP_PARAMR[14:2]	-	-	-	Not used

Registers	Acronyms	Type	Target	Description
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pSetVal	Immediate value or pointer on input data Must contain a 32-bit number or pointer on a 32-bit number according IMM value. It represents the value to write into the content of pOut.
pOut	Pointer on output data buffer The result is a real float32 number.
IOTYPE	Input and output buffer types: Indicates the pointers type pSetVal target: If POP0 = 1, it is a pointer-on-pointer if IMM = 1, it is an immediate value, otherwise it is a ABMP or IBP pointer. pOut target: If POP1 = 1, it is a pointer-on-pointer, otherwise it is a ABMP or IBP pointer. Refer to Section 2.2.6: IOTYPE description for details

5.6.27 __SCA_SQRT_F32

This in-line processing function can only be included into a processing list.

This function performs the square root of a real float32 scalar. Doing the square root of a negative number sets the FPUERRF to 1. According to the HSP programming, the FPUERRF flag can generate an interrupt to the CPU, freeze the HSP execution, or generate a break signal for timers.

The following operation is performed:

$$pOut = \sqrt{pInA}$$

5.6.27.1 *Command format to record __SCA_SQRT_F32*

Table 142. Command format to record __SCA_SQRT_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	__SCA_SQRT_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, IBP, POP	Pointer on input data A
HSP_PARAMR1	pOut	F32*	ABMP, IBP, POP	Pointer on output data
HSP_PARAMR[14:2]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data A Must contain a float32 real number. Must be positive or equal to 0. The input and output buffer pointers can be the same.
pOut	Pointer on output data buffer The result is a float32 real number.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer, otherwise it is a ABMP or IBP pointer. pOut target:

	If POP1 = 1, it is a pointer-on-pointer, otherwise it is a ABMP or IBP pointer. Refer to Section 2.2.6: IOTYPE description for details

5.6.28 __SCA_SUB_F32

This in-line processing function can only be included into a processing list.

It performs the subtraction of two real scalars.

$$pOut = pInA - pInB$$

5.6.28.1 *Command format to record __SCA_SUB_F32*

Table 143. Command format to record __SCA_SUB_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	__SCA_SUB_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, IBP, POP	Pointer on input data A
HSP_PARAMR1	pInB	F32* F32	ABMP, IBP, IMM, POP	Pointer or immediate on input data B
HSP_PARAMR2	pOut	F32*	ABMP, IBP, POP	Pointer on output data
HSP_PARAMR[14:4]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data A Must contain a float32 real number. Must be positive or equal to 0. The input and output buffer pointers can be the same.
pInB	Immediate value or pointer on input data B Must contain a float32 real number. Must be positive or equal to 0. The input and output buffer pointers can be the same. If IMM = 1, it must contain an immediate float32 value otherwise it must contain a pointer identifying a float32 variable.
pOut	Pointer on output data buffer The result is a float32 real number.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer, otherwise it is a ABMP or IBP pointer. pInB target: If POP1 = 1, it is a pointer-on-pointer, if IMM = 1, it is an immediate value, otherwise it is a ABMP or IBP pointer. pOut target: If POP2 = 1, it is a pointer-on-pointer, otherwise it is a ABMP or IBP pointer. Refer to Section 2.2.6: IOTYPE description for details

5.6.29 SCA_TOVECT

This processing function can only be programmed into a processing list.

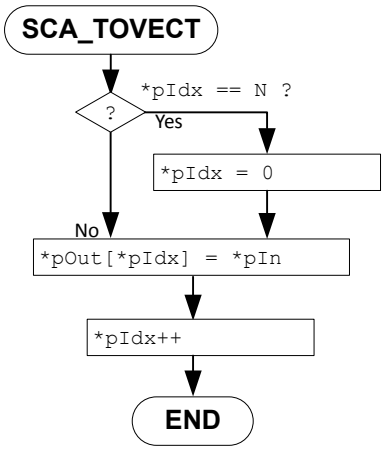
The SCA_TOVECT writes scalar values into a vector to build up a vector. Every time this function is executed, the scalar value is copied at the position pIdx inside the vector pointed by pOut and pIdx is incremented. When pIdx reaches its maximum value, given by N, it wraps-around on the next call of the function.

This function can be used, for example, if the application wants to perform the average of N consecutive coming from an ADC. The application first builds a vector from consecutive scalar values, then execute VECT_AVG_F32.

The figure below gives some details on the processing performed.

It is important to note that each SCA_TOVECT function uses the pIdx variable as the current pointer to write data into the output vector. It is up to the application to guarantee valid index values.

Figure 27. SCA_TOVECT timing example



5.6.29.1 *Command format to record SCA_TOVECT*

Table 144. Command format to record SCA_TOVECT

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	SCA_TOVECT	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pIn	X32*	ABMP, IBP, POP	Pointer on input data
HSP_PARAMR1	pIdx	U32*	ABMP, IBP, POP	Pointer on vector index
HSP_PARAMR2	pOut	X32*	ABMP, IBP, POP	Pointer on output vector
HSP_PARAMR3	N	U32	-	Number of vector elements
HSP_PARAMR[14:4]	-	-	-	-
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pIn	Pointer on input data The input buffer contains a 32-bit value, integer or float32.
pIdx	Pointer on vector index This variable contains the current vector index. Must be initialized the first time this function is executed.
pOut	Pointer on output vector The result is a vector containing 32-bit values, integer or float32.
N	Number of elements to be loaded into the buffer pointed by pOut.
IOTYPE	Input and output buffer types: Indicates the pointers type pIn target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP or IBP pointer. pIdx target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP or IBP pointer. pOut target:

If POP2 = 1, it is a pointer-on-pointer otherwise it is a ABMP or IBP pointer.

Refer to Section 2.2.6: IOTYPE description for details

5.7 Digital control functions

The function ID of each function is provided on Section 8: Function ID list.

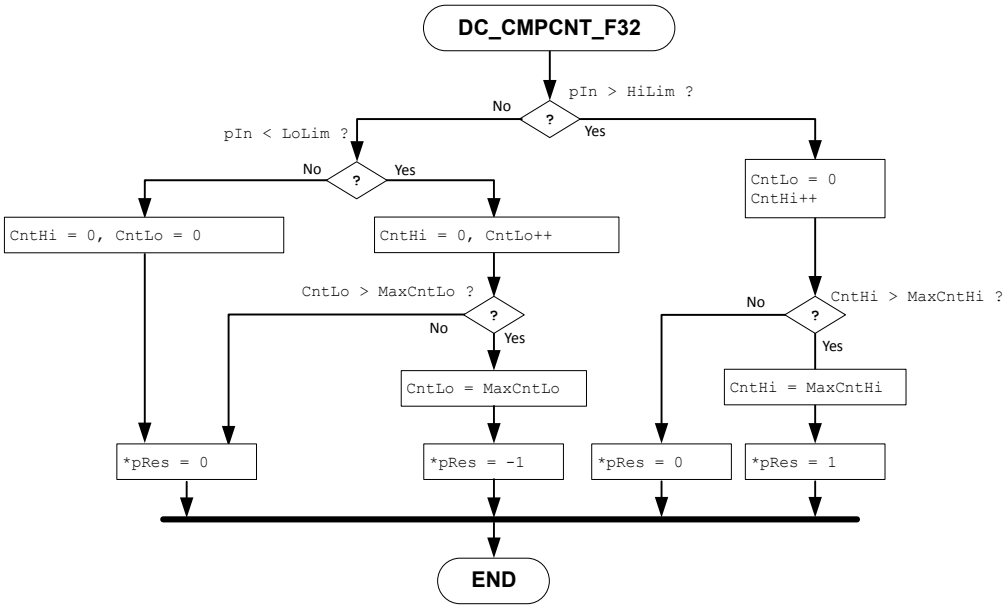
5.7.1 DC_CMPCNT_F32

This processing function can only be programmed into a processing list.

The DC_CMPCNT_F32 function performs a comparison of an input number to two thresholds (**LoLim**, **HiLim**):

- If a number of consecutive elements (given by **MaxCntHi**) are greater than **HiLim**, pRes is set to +1.
- If a number of consecutive elements (given by **MaxCntLo**) are less than **LoLim**, pRes is set to -1.
- If none of the previous conditions are met, pRes is set to 0.

Figure 28. DC_CMPCNT_F32 operation



5.7.1.1 *Initialization*

The DC_CMPCNT_F32 function is working with two main buffers pointers:

- A state buffer pointed by **pState**, containing the internal DC_CMPCNT_F32 state.
- A parameter buffer pointed by **pPar**, containing the DC_CMPCNT_F32 dynamic parameters.

The format of the buffers is given below:

5.7.1.2 *DC_CMPCNT_F32 pState format*

Table 145. DC_CMPCNT_F32 pState format

Offset addr.	Acronyms	Description	Data type
0	CntLo	Contains the current LO counter value.	U32
4	CntHi	Contains the current Hi counter value.	U32

1. *U8 stands for 8-bit unsigned integer, U16 stands for 16-bit unsigned integer, U32 stands for 32-bit unsigned integer, and F32 stands for 32-bit floating-point.*

5.7.1.3 DC_CMPCNT_F32 pLim format

Table 146. DC_CMPCNT_F32 pLim format

Offset addr.	Acronyms	Description	Data type
0	LoLim	Contains the current Low threshold value.	F32
4	HiLim	Contains the current High threshold value.	F32

1. *U8 stands for 8-bit unsigned integer, U16 stands for 16-bit unsigned integer, U32 stands for 32-bit unsigned integer, and F32 stands for 32-bit floating-point.*

5.7.1.4 DC_CMPCNT_F32 pCntLim format

Table 147. DC_CMPCNT_F32 pCntLim format

Offset addr.	Acronyms	Description	Data type
0	MaxCntLo	Contains the current LO counter value.	U32
4	MaxCntHi	Contains the current Hi counter value.	U32

1. *U8 stands for 8-bit unsigned integer, U16 stands for 16-bit unsigned integer, U32 stands for 32-bit unsigned integer, and F32 stands for 32-bit floating-point.*

Before running this processing function, the buffers pointed by **pState** and **pPar** must be instantiated, and initialized.

5.7.1.5 Command format to record DC_CMPCNT_F32

Table 148. Command format to record DC_CMPCNT_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	DC_CMPCNT_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pIn	F32*	ABMP, IBP, POP	Pointer on input data
HSP_PARAMR1	pRes	I32*	ABMP, POP	Comparison result
HSP_PARAMR2	pLim	F32*	ABMP, POP	Pointer on threshold limits
HSP_PARAMR3	pCntLim	U32*	ABMP	Pointer on counter limits
HSP_PARAMR4	pState	U32*	ABMP	Pointer on function state buffer
HSP_PARAMR[14:5]	-	-	-	-
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pIn	Pointer on input data buffer The input buffer must contain a real number in float32.
pRes	Pointer on the comparison result This function returns the following status: -1: Low limit reached +1: High limit reached 0: No limit reached
pLim	Pointer on the threshold limits buffer This buffer contains HiLim and LoLim values in float32. See Section 5.7.1.3: DC_CMPCNT_F32 pLim format for details.

pCntLim	Pointer on the counter limit buffer This buffer contains MaxCntLo, MaxCntHi values in uint32_t. See Section 5.7.1.4: DC_CMPCNT_F32 pCntLim format for details.
pState	Pointer on function state buffer This buffer contains CntLo and CntHi counters in float32. See Section 5.7.1.2: DC_CMPCNT_F32 pState format for details.
IOTYPE	Input and output buffer types: Indicates the pointers type pIn target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP or IBP pointer. pRes target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pLim target: If POP2 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. Refer to Section 2.2.6: IOTYPE description for details

5.7.2 DC_CLARKE1_2P_F32

Note: This feature is not available for STM32U3B5/3C5 or STM32U3A6/396 devices.

This processing function can only be included into a processing list.

The Clarke transform converts the instantaneous stator phases into a two-coordinate stationary vector. Generally the Clarke transform uses three-phase currents Ia, Ib and Ic to calculate currents in the two-phase orthogonal stator axis I_{alpha} and I_{beta}.

When I_{alpha} is superposed with Ia, and Ia + Ib + Ic = 0, I_{alpha} and I_{beta} can be calculated using only Ia and Ib.

The function returns I_{alpha} and I_{beta} with:

$$I_{alpha} = I_a$$

$$I_{beta} = \frac{1}{\sqrt{3}} \times (I_a + 2I_b)$$

5.7.2.1 *Command format to record DC_CLARKE1_2P_F32*

Table 149. Command format to record DC_CLARKE1_2P_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	DC_CLARKE1_2P_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP	Pointer on input data: Ia, Ib
HSP_PARAMR1	pOut	F32*	ABMP, POP	Pointer on output data: I_{alpha}, I_{beta}
HSP_PARAMR[14:2]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data buffer, containing Ia and Ib pInA must be a pointer to a buffer detailed in Section 5.7.2.2: IabBuf format. Input and output buffer pointers can be the same.
pOut	Pointer on output data buffer, containing I_{alpha} and I_{beta} pOut must be a pointer to a buffer detailed in Section 5.7.2.3: IalbeBuf format. The input and output buffer pointers can be the same.
IOTYPE	Input and output buffer types: Indicates the pointers type

pInA target:

If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer.

pOut target:

If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer.

Refer to Section 2.2.6: IOTYPE description for details

5.7.2.2 *IabBuf format*

Table 150. abBuf format

Offset addr.	Acronyms	Description	Data type
0	Ia	Contains the value of Ia current	F32
4	Ib	Contains the value of Ib current	F32

5.7.2.3 *IalbeBuf format*

Table 151. IalbeBuf format

Offset addr.	Acronyms	Description	Data type
0	I_{alpha}	Contains the value of Ia current	F32
4	I_{beta}	Contains the value of Ib current	F32

5.7.3 DC_ICLARKE1_2P_F32

Note: This feature is not available for STM32U3B5/3C5 or STM32U3A6/396 devices.

This processing function can only be included into a processing list.

The inverse Clarke transform converts the two-coordinate time invariant vector into instantaneous stator phases.

The function returns Ia and Ib with:

$$I_a = I_{alpha}$$

$$I_b = \frac{1}{2} \times \left(-I_{alpha} + \sqrt{3} \cdot I_{beta}\right)$$

where Ia and Ib are the instantaneous stator phases and Ia and Ib are the two coordinates of the stationary vector.

5.7.3.1 *Command format to record DC_ICLARKE1_2P_F32*

Table 152. Command format to record DC_ICLARKE1_2P_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	DC_ICLARKE1_2P_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP	Pointer on input data: I_{alpha}, I_{beta}
HSP_PARAMR1	pOut	F32*	ABMP, POP	Pointer on output data: Ia, Ib
HSP_PARAMR[14:2]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data structure, containing I_{alpha} and I_{beta} pInA must be a pointer to a structure detailed in Section 5.7.2.3: IalbeBuf format. Input and output buffer pointers can be the same.
pOut	Pointer on output data structure, containing Ia and Ib pOut must be a pointer to a structure detailed in Section 5.7.2.2: IabBuf format. The input and output buffer pointers can be the same.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pOut target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. Refer to Section 2.2.6: IOTYPE description for details

5.7.4 DC_PARK1_F32

Note: This feature is not available for STM32U3B5/3C5 or STM32U3A6/396 devices.

This processing function can only be included into a processing list.

The Park transform can be used to realize the transformation of the Ialpha and the Ibeta currents from the stationary to the moving reference frame and control the spatial relationship between the stator vector current and rotor flux vector.

The function returns Id and Iq with:

$$I_d = I_{alpha} \cdot \cos(\theta) + I_{beta} \cdot \sin(\theta)$$

$$I_q = -I_{alpha} \cdot \sin(\theta) + I_{beta} \cdot \cos(\theta)$$

5.7.4.1 *Command format to record DC_PARK1_F32*

Table 153. Command format to record DC_PARK1_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	DC_PARK1_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pAngle	F32*	ABMP, POP, IBP	Pointer on input data: angle
HSP_PARAMR1	pInA	F32*	ABMP, POP	Pointer on input data: I_{alpha}, I_{beta}
HSP_PARAMR2	pOut	F32*	ABMP, POP	Pointer on output data: Id, Iq
HSP_PARAMR[14:3]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pAngle	Pointer on buffer containing the angle (theta) pAngle points to a real number in float32, defining the angle Theta in radians.
pInA	Pointer on buffer containing I_{alpha} and I_{beta} pInA must be a pointer to a structure detailed in Section 5.7.2.3: IalbeBuf format. The input buffer can be the same as the output buffer (TBC).
pOut	Pointer on output data structure, containing Id and Iq pOut must be a pointer to a structure detailed in Section 5.7.4.2: IdqBuf format. The output buffer can be the same as the input buffer (TBC).

IOTYPE	Input and output buffer types: Indicates the pointers type pAngle target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP or IBP pointer. pInA target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pOut target: If POP2 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. Refer to Section 2.2.6: IOTYPE description for details

5.7.4.2 *IdqBuf format*

Table 154. dqBuf format

Offset addr.	Acronyms	Description	Data type
0	Id	Contains the value of Id current	F32
4	Iq	Contains the value of Iq current	F32

5.7.5 DC_IPARK1_F32

Note: This feature is not available for STM32U3B5/3C5 or STM32U3A6/396 devices.

This processing function can only be included into a processing list.

The inverse Park transform can be used to convert rotating d and q components back to stationary a and b components.

The function returns Id and Iq with:

$$I_{alpha} = I_d \cdot \cos(\theta) - I_q \cdot \sin(\theta)$$

$$I_{beta} = I_d \cdot \sin(\theta) + I_q \cdot \cos(\theta)$$

5.7.5.1 *Command format to record DC_IPARK1_F32*

Table 155. Command format to record DC_IPARK1_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	DCA_IPARK1_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pAngle	F32*	ABMP, POP, IBP	Pointer on input data: angle
HSP_PARAMR1	pInA	F32*	ABMP, POP	Pointer on input data: Id, Iq
HSP_PARAMR2	pOut	F32*	ABMP, POP	Pointer on output data: I_{alpha}, I_{beta}
HSP_PARAMR[14:3]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pAngle	Pointer on buffer containing the angle (theta) pAngle points to a real number in float32, defining the angle Theta in radians.
pInA	Pointer on input data structure, containing Id and Iq pInA must be a pointer to a structure detailed in Section 5.7.4.2: IdqBuf format. The input buffer can be the same as the output buffer (TBC).

pOut	Pointer on buffer containing I_{alpha} and I_{beta} pOut must be a pointer to a structure detailed in Section 5.7.2.3: IalbeBuf format. The output buffer can be the same as the input buffer (TBC).
IOTYPE	Input and output buffer types: Indicates the pointers type pAngle target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP or IBP pointer. pInA target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pOut target: If POP2 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. Refer to Section 2.2.6: IOTYPE description for details

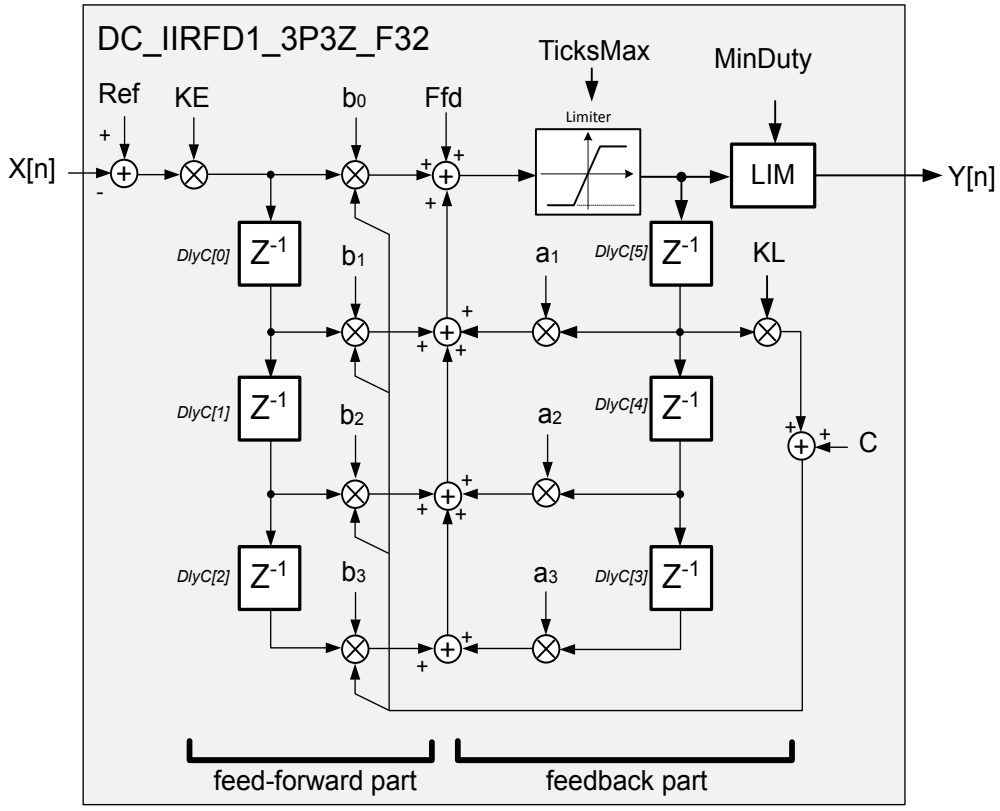
5.7.6 DC_IIRDF1_3P3Z_F32

The DC_IIRDF1_3P3Z_F32 supports the basic and enhanced form of a 3-poles, 3-zeros direct form 1 IIR filter. This filter is dedicated to digital control applications and is optimized to process sample-by-sample. It can only be executed into a processing.

Compared to a standard IIR filter, DC_IIRDF1_3P3Z_F32 offers a saturation block in the feedback path (anti-windup), the generation of the error signal (Ref-X[n]), a limiter at filter output to control the data range provided to the PWM modulator. The output result is in float32 format.

The block SAT sets the flag FPUSATF of the HSP_EVT_ISR register if a saturation is performed.

Figure 29. DC_IIRDF1_3P3Z_F32 structure



DT80673V1

DlyC[x] stands for delay cells

The difference equation of the filter is calculated as follows:

$$e[n] = (Ref - x[n]) \cdot KE$$

$$GL = (KL \cdot y[n-1]) + C$$

$$y[n] = GL \cdot (b_0 \cdot e[n] + b_1 \cdot e[n-1] + b_2 \cdot e[n-2] + b_3 \cdot e[n-3]) + a_1 \cdot y(n-1) + a_2 \cdot y[n-2] + a_3 \cdot y[n-3] + Ffd$$

Note: To implement a basic version of the DC_IIRDF1_3P3Z_F32, KL must be set to 0, and C to 1, so that GL = 1.

5.7.6.1 Command format to record DC_IIRDF1_3P3Z_F32

Table 156. Command format to record DC_IIRDF1_3P3Z_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	DC_IIRDF1_3P3Z_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pIn	F32*	ABMP, IBP, POP	Pointer on input data buffer
HSP_PARAMR1	pFltCoef	F32*	ABMP, POP	Pointer on filter coefficient buffer
HSP_PARAMR2	pOut	F32*	ABMP, IBP, POP	Pointer on output data buffer
HSP_PARAMR3	pFltState	ST*	ABMP	Pointer on filter state buffer
HSP_PARAMR[14:4]	-	-	-	-
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

Table 157. Specifications for acronyms of command formats

pIn	Pointer on input data The input buffer must contain a real scalar in float32. The input and output buffer pointers can be the same.
pFltCoef	Pointer on filter coefficient buffer Must contain real values in float32. See format in Table 158. pFltCoef format for DC_IIRDF1_3P3Z_F32.
pOut	Pointer on output data Filtered samples. The input and output buffer pointers can be the same.
pFltState	Address of the filter state buffer See Section 5.7.6.3: DC_IIRDF1_3P3Z_F32 filter states for details.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is an ABMP or IBP pointer depending on its value. pFltCoef target: If POP1 = 1, it is a pointer-on-pointer otherwise it is an ABMP pointer. pOut target: If POP2 = 1, it is a pointer-on-pointer otherwise it is an ABMP or IBP pointer depending on its value. Refer to Section 2.2.6: IOTYPE description for details

Command status: see Section 5.1: Command status for details.

5.7.6.2 DC_IIRDF1_3P3Z_F32 coefficient format

Filter coefficients must be stored into the buffer pointed by pFltCoef, as shown below:

Table 158. **pFltCoef format for DC_IIRDF1_3P3Z_F32**

CPU Offset addr.	Acronyms	Description	Data type
0	KL	Gain value	F32
4	C	-	F32
8	b3	-	F32
12	a3	-	F32
16	b2	-	F32
20	a2	-	F32
24	b1	-	F32
28	a1	-	F32
32	Ref	Input reference	F32
36	KE	Input gain	F32
40	b0	Filter coefficient	F32
44	Ffd	Feed Forward input	F32
48	TicksMax	Max signal value for feedback path	F32
52	MinDuty	Min signal value	F32

1. *U8 stands for 8-bit unsigned integer, U16 stands for 16-bit unsigned integer, U32 stands for 32-bit unsigned integer, and F32 stands for 32-bit floating-point.*

5.7.6.3 *DC_IIRDF1_3P3Z_F32 filter states*

Each filter instance has its own filter state. The application must instantiate a buffer in BRAM containing the filter state for each filter instance.

Table 159. **FltState structure format for DC_IIRDF1_3P3Z_F32**

CPU Address offset	Field name	Type	Description
0	pStateBuf	F32*	Filter state buffer start address
4	pWork	F32*	Working pointer on next state buffer entry
8	NDLYC	U32	Filter state size or number of delay cells, fixed to 6

The created FltState structure must be initialized before a filter uses it.

- pStateBuf and pWork pointers must point to the StateBuf[0] element. These pointers are for SPE internal usage and must be translated into pointers directly usable by the SPE as explained on Section 2.2.4: Buffer pointer transmission on processing lists even if filter is used in a direct command.
- NDLYC must be set to 6.
- StateBuf[x] must be initialized to 0

The next table shows the format of the filter state table.

Table 160. **Filter state buffer for DC_IIRDF1_3P3Z_F32**

CPU Address offset	Field name	Type	Buff Type	Description
0	DlyC[2]	F32	ABMP	Filter state data
4	DlyC[3]	F32	ABMP	Filter state data
8	DlyC[1]	F32	ABMP	Filter state data
12	DlyC[4]	F32	ABMP	Filter state data
16	DlyC[0]	F32	ABMP	Filter state data
20	DlyC[5]	F32	ABMP	Filter state data

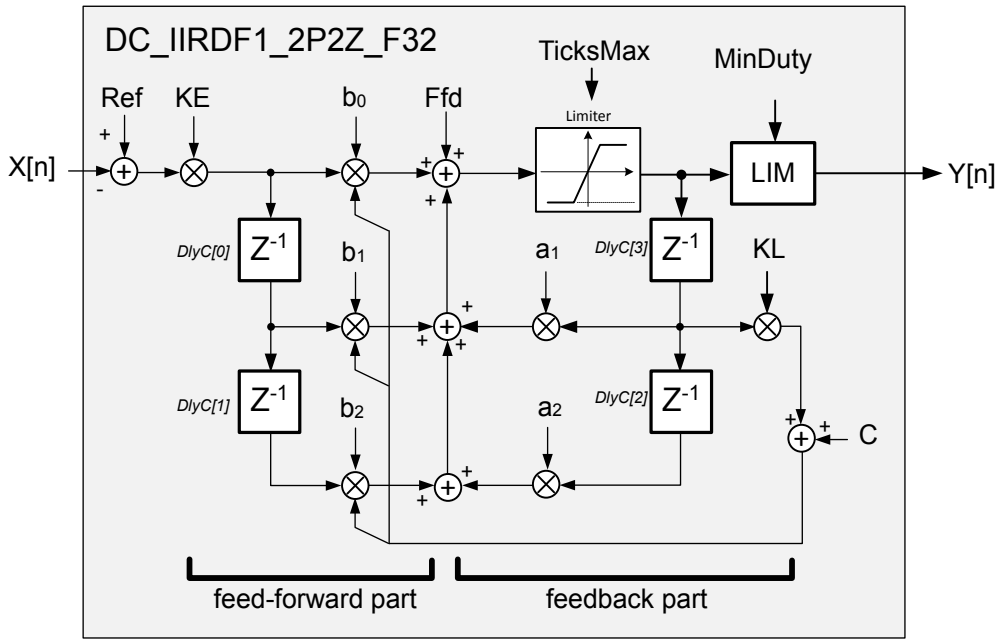
5.7.7 DC_IIRDF1_2P2Z_F32

The DC_IIRDF1_2P2Z_F32 supports the basic and enhanced form of a 2-poles, 2-zeros direct form 1 IIR filter. This filter is dedicated to digital control applications and optimized to process sample-by-sample. It can only be executed into a processing list.

Compared to a standard IIR filter, DC_IIRDF1_2P2Z_F32 offers a saturation block in the feedback path (anti-windup), the generation of the error signal (Ref-X[n]), a limiter at filter output to control the data range provided to the PWM modulator. The output result is in float32 format.

The block SAT sets the flag FPUSATF of the HSP_EVT_ISR register if a saturation is performed.

Figure 30. DC_IIRDF1_2P2Z_F32 structure



DlyC[x] stands for delay cells

DT80672V1

Below, the difference equation of the filter:

$$e[n] = (Ref - x[n]) \cdot KE$$

$$GL = (KL \cdot y[n-1]) + C$$

$$y[n] = GL \cdot (b_0 \cdot e[n] + b_1 \cdot e[n-1] + b_2 \cdot e[n-2]) + a_1 \cdot y(n-1) + a_2 \cdot y[n-2] + Ffd$$

Note: To implement a basic version of the DC_IIRDF1_2P2Z_F32, KL must be set to 0, and C to 1, so that GL = 1.

5.7.7.1 *Command format to record DC_IIRDF1_2P2Z_F32*

Table 161. Command format to record DC_IIRDF1_2P2Z_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	DC_IIRDF1_2P2Z_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pIn	F32*	ABMP, IBP, POP	Pointer on input data buffer
HSP_PARAMR1	pFltCoef	F32*	ABMP, POP	Pointer on filter coefficient buffer
HSP_PARAMR2	pOut	F32*	ABMP, IBP, POP	Pointer on output data buffer
HSP_PARAMR3	pFltState	ST*	ABMP	Pointer on filter state buffer

Registers	Acronyms	Type	Target	Description
HSP_PARAMR[14:4]	-	-	-	-
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pIn	Pointer on input data The input buffer must contain a real scalar in float32. The input and output buffer pointers can be the same.
pFltCoef	Pointer on filter coefficient buffer Must contain real values in float32. See format in Table 162. pFltCoef format for DC_IIRDF1_2P2Z_F32.
pOut	Pointer on output data Filtered samples. The input and output buffer pointers can be the same.
pFltState	Address of the filter state buffer See Section 5.7.6.3: DC_IIRDF1_3P3Z_F32 filter states for details.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is an ABMP or IBP pointer depending on its value. pFltCoef target: If POP1 = 1, it is a pointer-on-pointer otherwise it is an ABMP pointer. pOut target: If POP2 = 1, it is a pointer-on-pointer otherwise it is an ABMP or IBP pointer depending on its value. Refer to Section 2.2.6: IOTYPE description for details

Command status: see Section 5.1: Command status for details.

5.7.7.2 *DC_IIRDF1_2P2Z_F32 coefficient format*

Filter coefficients must be stored in the buffer pointed by pFltCoef, as shown below:

Table 162. pFltCoef format for DC_IIRDF1_2P2Z_F32

CPU Offset addr.	Acronyms	Description	Data type
0	KL	Gain value	F32
4	C	-	F32
8	b2	-	F32
12	a2	-	F32
16	b1	-	F32
20	a1	-	F32
24	Ref	Input reference	F32
28	KE	Input gain	F32
32	b0	Filter coefficient	F32
36	Ffd	Feed forward input	F32
40	TicksMax	Max signal value for feedback path	F32
44	MinDuty	Min signal value	F32

1. *U8 stands for 8-bit unsigned integer, U16 stands for 16-bit unsigned integer, U32 stands for 32-bit unsigned integer, and F32 stands for 32-bit floating-point.*

5.7.7.3 *DC_IIRDF1_2P2Z_F32 filter states*

Each filter instance has its own filter state. The application must instantiate a buffer in BRAM containing the filter state for each filter instance.

Table 163. FltState structure format for DC_IIRDF1_2P2Z_F32

CPU Address offset	Field name	Type	Description
0	pStateBuf	F32*	Filter State buffer start address
4	pWork	F32*	Working pointer on next State buffer entry
8	NDLYC	U32	Filter state size or number of delay cells, fixed to 4

The created FltState structure must be initialized before a filter uses it.

- pStateBuf and pWork pointers must point to the StateBuf[0] element. These pointers are for SPE internal usage and must be translated into pointers directly usable by the SPE as explained on Section 2.2.4: Buffer pointer transmission on processing lists even if filter is used in a direct command.
- NDLYC must be set to 4.
- StateBuf[x] must be initialized to 0

The next table shows the format of the filter state table.

Table 164. Filter state buffer for DC_BIIR_2P2Z_F32 and DC_EIIR_2P2Z_F32

CPU Address offset	Field name	Type	Buff Type	Description
0	DlyC[1]	F32	ABMP	Filter state data
4	DlyC[2]	F32	ABMP	Filter state data
8	DlyC[0]	F32	ABMP	Filter state data
12	DlyC[3]	F32	ABMP	Filter state data

5.8 Scalar functions working on 32-bit integer

The function ID of each function is provided on Section 8: Function ID list.

5.8.1 __SCA_ADD_I32

This in-line processing function can only be included into a processing list.

The value returned into pOut is a 32-bit integer. If the addition result is greater than 32 bits, only the first 32 bits LSB are returned. It is up to the application to insure that the result is lower than 32 bits.

It performs the addition of two 32-bit signed integers.

$$pOut = pInA + pInB$$

5.8.1.1 *Command format to record __SCA_ADD_I32*

Table 165. Command format to record __SCA_ADD_I32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	__SCA_ADD_I32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	I32*	ABMP, POP, IBP	Pointer on input data A
HSP_PARAMR1	pInB	I32* I32	ABMP, POP, IMM, IBP	Immediate or pointer on input data B
HSP_PARAMR2	pOut	I32*	ABMP, POP, IBP	Pointer on output data
HSP_PARAMR[14:4]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data A Must contain integer number. The input and output buffer pointers can be the same.
pInB	Immediate or pointer on input data B Must be a pointer to an integer number if IMM = 0, otherwise it must represent an integer number. The input and output buffer pointers can be the same.
pOut	Pointer on output data buffer The result is an integer number.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer, otherwise it is a ABMP or IBP pointer. pInB target: If POP1 = 1, it is a pointer-on-pointer, if IMM = 1, it is an immediate value, otherwise it is a ABMP or IBP pointer. pOut target: If POP2 = 1, it is a pointer-on-pointer, otherwise it is a ABMP or IBP pointer. Refer to Section 2.2.6: IOTYPE description for details

5.8.2 __SCA_INC_U32

This in-line processing function can only be included into a processing list.
It performs the increment of an unsigned 32-bit integer.

5.8.2.1 Command format to record __SCA_INC_U32

Table 166. Command format to record __SCA_INC_U32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	__SCA_INC_U32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	U32*	ABMP, POP, IBP	Pointer on input data A
HSP_PARAMR1	pOut	U32*	ABMP, POP, IBP	Pointer on output data
HSP_PARAMR[14:2]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data A Must contain unsigned integer number. The input and output buffer pointers can be the same.
pOut	Pointer on output data buffer Must contain unsigned integer number.. The input and output buffer pointers can be the same.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer, otherwise it is a ABMP or IBP pointer. pOut target: If POP2 = 1, it is a pointer-on-pointer, otherwise it is a ABMP or IBP pointer.

	Refer to Section 2.2.6: IOTYPE description for details

5.8.3 __SCA_MUL_I32

This in-line processing function can only be included into a processing list.

It performs the multiplication of two 32-bit signed integers.

The value returned into pOut is a 32-bit integer. If the multiplication result is bigger than 32 bits, only the least significant 32 bits are returned. It is up to the application to insure that the result is lower than 32 bits.

5.8.3.1 *Command format to record __SCA_MUL_I32*

Table 167. Command format to record __SCA_MUL_I32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	__SCA_MUL_I32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	I32*	ABMP, POP, IBP	Pointer on input data A
HSP_PARAMR1	pInB	I32* I32	ABMP, POP, IMM, IBP	Immediate or pointer on input data B
HSP_PARAMR2	pOut	I32*	ABMP, POP, IBP	Pointer on output data
HSP_PARAMR[14:4]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data A Must contain integer number. The input and output buffer pointers can be the same.
pInB	Immediate or pointer on input data B Must be a pointer to an integer number if IMM = 0, otherwise it must represent an integer number. The input and output buffer pointers can be the same.
pOut	Pointer on output data buffer The result is an integer number.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer, otherwise it is a ABMP or IBP pointer. pInB target: If POP1 = 1, it is a pointer-on-pointer, if IMM = 1, it is an immediate value, otherwise it is a ABMP or IBP pointer. pOut target: If POP2 = 1, it is a pointer-on-pointer, otherwise it is a ABMP or IBP pointer. Refer to Section 2.2.6: IOTYPE description for details

5.8.4 __SCA_SHIFT_I32

This in-line processing function can only be included into a processing list.

It performs the shift of an integer. If pShift is positive, then the integer is shifted to the left. If pShift is negative then the integer is shifted to the right.

5.8.4.1 *Command format to record __SCA_SHIFT_I32*

Table 168. Command format to record __SCA_SHIFT_I32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	__SCA_SHIFT_I32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	I32*	ABMP, POP, IBP	Pointer on input data A
HSP_PARAMR1	pShift	I32* I32	ABMP, POP, IMM, IBP	Immediate or pointer on shift value
HSP_PARAMR2	pOut	I32*	ABMP, POP, IBP	Pointer on output data
HSP_PARAMR[14:4]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data A Must contain integer number. The input and output buffer pointers can be the same.
pShift	Immediate or pointer on shift value Must be a pointer to an integer number if IMM = 0, otherwise it must represent an integer number.
pOut	Pointer on output data buffer The result is an integer number. The input and output buffer pointers can be the same.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer, otherwise it is a ABMP or IBP pointer. pInB target: If POP1 = 1, it is a pointer-on-pointer, if IMM = 1, it is an immediate value, otherwise it is a ABMP or IBP pointer. pOut target: If POP2 = 1, it is a pointer-on-pointer, otherwise it is a ABMP or IBP pointer. Refer to Section 2.2.6: IOTYPE description for details

5.8.5 __SCA_AND_U32

This in-line processing function can only be included into a processing list.
It performs the logical bitwise AND between two 32-bit unsigned integers.

5.8.5.1 *Command format to record __SCA_AND_U32*

Table 169. Command format to record __SCA_AND_U32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	__SCA_AND_U32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	U32*	ABMP, POP, IBP	Pointer on input data A
HSP_PARAMR1	pInB	U32* U32	ABMP, POP, IMM, IBP	Immediate or pointer on input data B
HSP_PARAMR2	pOut	U32*	ABMP, POP, IBP	Pointer on output data
HSP_PARAMR[14:4]	-	-	-	Not used

Registers	Acronyms	Type	Target	Description
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data A Must contain integer number. The input and output buffer pointers can be the same.
pInB	Immediate or pointer on input data B Must be a pointer to an integer number if IMM = 0, otherwise it must represent an integer number. The input and output buffer pointers can be the same.
pOut	Pointer on output data buffer The result is an integer number.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer, otherwise it is a ABMP or IBP pointer. pInB target: If POP1 = 1, it is a pointer-on-pointer, if IMM = 1, it is an immediate value, otherwise it is a ABMP or IBP pointer. pOut target: If POP2 = 1, it is a pointer-on-pointer, otherwise it is a ABMP or IBP pointer. Refer to Section 2.2.6: IOTYPE description for details

5.8.6 __SCA_OR_U32

This in-line processing function can only be included into a processing list.
This function performs the logical bitwise OR between two 32-bit unsigned integers.

5.8.6.1 Command format to record __SCA_OR_U32

Table 170. Command format to record __SCA_OR_U32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	__SCA_OR_U32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	U32*	ABMP, POP, IBP	Pointer on input data A
HSP_PARAMR1	pInB	U32* U32	ABMP, POP, IMM, IBP	Immediate or pointer on input data B
HSP_PARAMR2	pOut	U32*	ABMP, POP, IBP	Pointer on output data
HSP_PARAMR[14:4]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data A Must contain integer number. The input and output buffer pointers can be the same.
pInB	Immediate or pointer on input data B Must be a pointer to an integer number if IMM = 0, otherwise it must represent an integer number. The input and output buffer pointers can be the same.

pOut	Pointer on output data buffer The result is an integer number.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer, otherwise it is a ABMP or IBP pointer. pInB target: If POP1 = 1, it is a pointer-on-pointer, if IMM = 1, it is an immediate value, otherwise it is a ABMP or IBP pointer. pOut target: If POP2 = 1, it is a pointer-on-pointer, otherwise it is a ABMP or IBP pointer. Refer to Section 2.2.6: IOTYPE description for details

5.8.7 __SCA_XOR_U32

This in-line processing function can only be included into a processing list.
It performs the logical bitwise exclusive OR between two 32-bit unsigned integers.

5.8.7.1 Command format to record __SCA_XOR_U32

Table 171. Command format to record __SCA_XOR_U32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	__SCA_XOR_U32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	U32*	ABMP, POP, IBP	Pointer on input data A
HSP_PARAMR1	pInB	U32* U32	ABMP, POP, IMM, IBP	Immediate or pointer on input data B
HSP_PARAMR2	pOut	U32*	ABMP, POP, IBP	Pointer on output data
HSP_PARAMR[14:4]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data A Must contain integer number. The input and output buffer pointers can be the same.
pInB	Immediate or pointer on input data B Must be a pointer to an integer number if IMM = 0, otherwise it must represent an integer number. The input and output buffer pointers can be the same.
pOut	Pointer on output data buffer The result is an integer number.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer, otherwise it is a ABMP or IBP pointer. pInB target: If POP1 = 1, it is a pointer-on-pointer, if IMM = 1, it is an immediate value, otherwise it is a ABMP or IBP pointer. pOut target: If POP2 = 1, it is a pointer-on-pointer, otherwise it is a ABMP or IBP pointer. Refer to Section 2.2.6: IOTYPE description for details

5.8.8 __SCA_NOT_U32

This in-line processing function can only be included into a processing list.
It performs the logical bitwise NOT of a 32-bit unsigned integer.

5.8.8.1 *Command format to record __SCA_NOT_U32*

Table 172. Command format to record __SCA_NOT_U32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	__SCA_NOT_U32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	U32*	ABMP, POP, IBP	Pointer on input data A
HSP_PARAMR1	-	-	-	-
HSP_PARAMR2	pOut	U32*	ABMP, POP, IBP	Pointer on output data
HSP_PARAMR[14:4]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data A Must contain integer number. The input and output buffer pointers can be the same.
pOut	Pointer on output data buffer The result is an integer number. The input and output buffer pointers can be the same.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer, otherwise it is a ABMP or IBP pointer. pOut target: If POP1 = 1, it is a pointer-on-pointer, otherwise it is a ABMP or IBP pointer. Refer to Section 2.2.6: IOTYPE description for details

5.8.9 SCA_MADD_I32

This processing function can only be included into a processing list.
It performs the addition of two 32-bit signed integers with modulus.
The function performs the following operation:
*InA = *InA + INC

5.8.9.1 *Command format to record SCA_MADD_I32*

Table 173. Command format to record SCA_MADD_I32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	SCA_MADD_I32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	I32*	ABMP, POP, IBP	Pointer on input data A
HSP_PARAMR1	INC	U32	-	Increment
HSP_PARAMR2	BASE	U32	-	Base value
HSP_PARAMR3	SIZE	U32	-	Size value
HSP_PARAMR[14:4]	-	-	-	Not used

Registers	Acronyms	Type	Target	Description
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data A Must contain integer number.
INC	Increment value Value added to *pInA
BASE	Base value If *pInA is bigger than BASE+SIZE, it is set to BASE
SIZE	Size value
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP or IBP pointer. Refer to Section 2.2.6: IOTYPE description for details

5.9 Matrix functions on real

The function ID of each function is provided on Section 8: Function ID list.

5.9.1 MAT_ABS_F32

The MAT_ABS_F32 can be either executed as a direct command (accelerator mode) or programmed into a processing list.

This function computes the absolute value of each element of a matrix. This function is similar to Section 5.4.1: VECT_ABS_F32.

Refer to Section 2.2.1: Vector, matrix operations, and formats for data buffer formats.

5.9.1.1 Command format to record MAT_ABS_F32

Table 174. Command format to record MAT_ABS_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	MAT_ABS_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP	Pointer on input matrix
HSP_PARAMR1	pOut	F32*	ABMP, POP	Pointer on output matrix
HSP_PARAMR2	N	U32	-	Number of elements
HSP_PARAMR[14:3]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input matrix Must contain a real matrix in float32. pIn can be the same as pOut.
pOut	Pointer on output matrix The result is a real matrix in float32. pOut can be the same as pIn.
N	Number of elements Defines the number of elements to be processed. N must be equal to COL x ROW. Can be any integer between 2 and 4096.

IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP or IBP pointer. pOut target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP or IBP pointer. Refer to Section 2.2.6: IOTYPE description for details

See Table 62. Format for generic direct commands with two pointers when VECT_ABS_F32 is executed as a direct command.

5.9.2 MAT_ADD_F32

The MAT_ADD_F32 can be either executed as a direct command (accelerator mode) or programmed into a processing list.

This function adds each element of two matrices. Both input matrices have the same dimension.

This function is similar to Section 5.4.3: VECT_ADD_F32.

Refer to Section 2.2.1: Vector, matrix operations, and formats for data buffer formats.

The following operation is performed:

$$OutBuf[r][c] = InABuf[r][c] + InBBuf[r][c]$$

with N = r x c and r, c integer

5.9.2.1 *Command format to record MAT_ADD_F32*

Table 175. Command format to record MAT_ADD_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	MAT_ADD_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP,	Pointer on input matrix A
HSP_PARAMR1	pInB	F32*	ABMP, POP,	Pointer on input matrix B
HSP_PARAMR2	pOut	F32*	ABMP, POP,	Pointer on output matrix
HSP_PARAMR3	N	U32	-	Number of matrix elements
HSP_PARAMR[14:4]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input matrix A Must contain a real ROW-by-COL matrix in float32. pInA can be the same as pInB or pOut.
pInB	Pointer on input matrix B Must contain a real ROW-by-COL matrix in float32. pInB can be the same as pInA or pOut.
pOut	Pointer on output matrix The result is a real ROW-by-COL matrix in float32. pOut can be the same as pInA or pInB.
N	Number of matrix elements Number of elements to be processed. N must be equal to COL x ROW. Can be any integer between 2 and 4096.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer.

	pInB target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pOut target: If POP2 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. Refer to Section 2.2.6: IOTYPE description for details

See Table 61. Format for generic direct commands with three pointers when MAT_ADD_F32 is executed as a direct command.

5.9.3 MAT_INV_F32

The matrix inversion can be either executed as a direct command (accelerator mode) or programmed into a processing list.

Considering a square matrix A, the matrix B is inverse of the matrix A if

$AB = BA = I_n$

Where I_n is the identity matrix.

B is also noted A^{-1}

The Gauss-Jordan method is used to find the inverse. The algorithm performs a sequence of elementary row-operations until it reduces the input matrix to an identity matrix. Applying the same sequence of elementary row-operations to an identity matrix yields the inverse matrix. If the input matrix is singular, then the algorithm terminates and returns error status.

Refer to Section 2.2.1: Vector, matrix operations, and formats for data buffer formats.

5.9.3.1 *Command format to record MAT_INV_F32*

Table 176. Command format to record MAT_INV_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	MAT_INV_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP,	Pointer on input matrix
HSP_PARAMR1	pOut	F32*	ABMP, POP,	Pointer on output matrix
HSP_PARAMR2	-	-	-	-
HSP_PARAMR3	ROW	U32	-	Number of matrix rows
HSP_PARAMR4	COL	U32	-	Number of matrix columns
HSP_PARAMR[14:5]	-	-	-	-
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input matrix Must contain a real ROW-by-COL matrix in float32.
pOut	Pointer on output matrix The result is a real ROW-by-COL matrix in float32.
ROW	Number of rows of the input matrix COL x ROW must be lower than or equal to 4096.
COL	Number of columns of the input matrix COL x ROW must be lower than or equal to 4096.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer.

	pOut target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. Refer to Section 2.2.6: IOTYPE description for details

5.9.3.2 *MAT_INV_F32 direct command format*

The next table shows the command format when MAT_INV_F32 is executed as a direct command.

Table 177. MAT_INV_F32 direct command forma

Order	Registers	Acronyms	Type	Target	Description
1	HSP_DCMDIDR	MAT_INV_F32	U32	-	See Section 8: Function ID list
2	HSP_PARAMR3	ROW	U32	-	See description above
3	HSP_PARAMR4	COL	U32	-	
4	HSP_DCMDPTR0	pInA	F32*	ABMP	
5	HSP_DCMDPTR1	pOut	F32*	ABMP	

1. *Parameters must be written in the specified order.*
2. *Refer to Section 5: Processing functions description for acronym details*

5.9.4 MAT_OFFSET_F32

The matrix offset can be either executed as a direct command (accelerator mode) or programmed into a processing list.

This function adds to each real matrix element an offset value. This function is similar to Section 5.4.28: VECT_OFFSET_F32.

The following operation is performed:

$$OutBuf[r][c] = InBuf[r][c] + OffBuf$$

with N = r x c and r, c integer

5.9.4.1 *Command format to record MAT_OFFSET_F32*

Table 178. Command format to record MAT_OFFSET_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	MAT_OFFSET_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP,	Pointer on input matrix
HSP_PARAMR1	pOff	F32* F32	ABMP, IMM, POP,	Immediate or pointer on the offset value
HSP_PARAMR2	pOut	F32*	ABMP, POP,	Pointer on output matrix
HSP_PARAMR3	N	U32	-	Number of elements to process
HSP_PARAMR[14:4]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input matrix Must contain a real ROW -by-COL matrix in float32. pIn can be the same as pOut.
pOff	Immediate value or pointer on the offset value If IMM = 1, it must contain an immediate float32 value otherwise it must contain a pointer identifying a float32 variable.

pOut	Pointer on output matrix The result is a real ROW-by-COL matrix in float32. pOut can be the same as pIn.
N	Number of elements to process Can be any integer between 2 and 4096.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pOff target: If POP1 = 1, it is a pointer-on-pointer if IMM = 1 it is an immediate value, otherwise it is an ABMP pointer depending on its value. pOut target: If POP2 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. Refer to Section 2.2.6: IOTYPE description for details

See Table 61. Format for generic direct commands with three pointers when MAT_OFFSET_F32 is executed as a direct command.

5.9.5 MAT_SCALE_F32

The matrix scaling can be either executed as a direct command (accelerator mode) or programmed into a processing list.

This function multiplies each real matrix element by a scale value. This function is similar to Section 5.4.32: VECT_SCALE_F32.

The following operation is performed:

$$Out[r][c] = In[r][c] \cdot Scale$$

with N = r x c and r, c integers

5.9.5.1 Command format to record MAT_SCALE_F32

Table 179. Command format to record MAT_SCALE_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	MAT_SCALE_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP,	Pointer on input matrix
HSP_PARAMR1	pScale	F32* F32	ABMP, POP, IMM,	Pointer or immediate on the scale value
HSP_PARAMR2	pOut	F32*	ABMP, POP,	Pointer on output matrix
HSP_PARAMR3	N	U32	-	Number of elements to process
HSP_PARAMR[14:4]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input matrix Must contain a real ROW-by-COL matrix in float32. pIn can be the same as pOut.
pScale	Immediate value or pointer on the scale value If IMM = 1, it must contain an immediate float32 value otherwise it must contain a pointer identifying a float32 variable.

pOut	Pointer on output matrix The result is a real ROW-by-COL matrix in float32. pOut can be the same as pIn.
N	Number of elements to process Can be any integer between 2 and 4096.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pScale target: If POP1 = 1, it is a pointer-on-pointer if IMM = 1 it is an immediate value, otherwise it is an ABMP pointer depending on its value. pOut target: If POP2 = 1, it is a pointer-on-pointer otherwise it is a ABMP IP pointer. Refer to Section 2.2.6: IOTYPE description for details

See Table 61. Format for generic direct commands with three pointers when MAT_SCALE_F32 is executed as a direct command.

5.9.6 MAT_SUB_F32

The matrix subtraction can be either executed as a direct command (accelerator mode) or programmed into a processing list.

It is element related subtraction of two real matrix. Both input matrix have the same dimension.

This function is similar to Section 5.4.35: VECT_SUB_F32.

The following operation is performed:

$$OutBuf[r][c] = InABuf[r][c] - InBBuf[r][c]$$

with N = r x c and r, c integer

5.9.6.1 Command format to record MAT_SUB_F32

Table 180. Command format to record MAT_SUB_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	MAT_SUB_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP,	Pointer on input matrix A
HSP_PARAMR1	pInB	F32*	ABMP, POP,	Pointer on input matrix B
HSP_PARAMR2	pOut	F32*	ABMP, POP,	Pointer on output matrix
HSP_PARAMR3	N	U32	-	Number of matrix elements
HSP_PARAMR[14:4]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input matrix A Must contain a real ROW-by-COL matrix in float32. pInA can be the same as pInB or pOut.
pInB	Pointer on input matrix B Must contain a real ROW-by-COL matrix in float32. pInB can be the same as pInA or pOut.
pOut	Pointer on output matrix The result is a real ROW-by-COL matrix in float32. pOut can be the same as pInA or pInB.

N	Number of matrix elements Number of elements to be processed. N must be equal to COL x ROW. Can be any integer between 2 and 4096.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pInB target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pOut target: If POP2 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. Refer to Section 2.2.6: IOTYPE description for details

See Table 61. Format for generic direct commands with three pointers when MAT_SUB_F32 is executed as a direct command.

5.9.7 MAT_MUL_F32

The matrix multiplication can be either executed as a direct command (accelerator mode) or programmed into a processing list.

Matrix multiplication is only defined if the number of columns of the matrix A equals the number of rows of the matrix B. Multiplying an M-by-N matrix with an N-by-P matrix results in an M-by-P matrix.

Refer to Section 2.2.1: Vector, matrix operations, and formats for data buffer formats.

5.9.7.1 Command format to record MAT_MUL_F32

Table 181. Command format to record MAT_MUL_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	MAT_MUL_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP,	Pointer on input matrix A
HSP_PARAMR1	pInB	F32*	ABMP, POP,	Pointer on input matrix B
HSP_PARAMR2	pOut	F32*	ABMP, POP,	Pointer on output matrix
HSP_PARAMR3	ROWA	U32	-	Number of matrix rows
HSP_PARAMR4	COLA	U32	-	Number of matrix columns
HSP_PARAMR5	ROWB	U32	-	Number of matrix rows
HSP_PARAMR6	COLB	U32	-	Number of matrix columns
HSP_PARAMR[14:7]	-	-	-	-
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input matrix A Must contain a real ROWA-by-COLA matrix in float32.
pInB	Pointer on input matrix B Must contain a real ROWB-by-COLB matrix in float32.
pOut	Pointer on output matrix The result is a real ROWA-by-COLB matrix in float32.
ROWA	Number of rows of the input matrix A
COLA	Number of columns of the input matrix A, must be equal to ROWB

ROWB	Number of rows of the input matrix B, must be equal to COLA
COLB	Number of columns of the input matrix B
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pInB target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pOut target: If POP2 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. Refer to Section 2.2.6: IOTYPE description for details

5.9.7.2 *MAT_MUL_F32 direct command format*

The next table shows the command format when MAT_MUL_F32 is executed as a direct command.

Table 182. MAT_MUL_F32 direct command format

Order	Registers	Acronyms	Type	Target	Description
1	HSP_DCMDIDR	MAT_MUL_F32	U32	-	See Section 8: Function ID list
2	HSP_PARAMR0	ROWA	U32	-	See description above
3	HSP_PARAMR1	COLA	U32	-	
4	HSP_PARAMR3	ROWB	U32	-	
5	HSP_PARAMR4	COLB	U32	-	
6	HSP_PARAMR6	ROWOUT	U32	-	Number of row on output matrix
7	HSP_PARAMR7	COLOUT	U32	-	Number of column on output matrix
8	HSP_DCMDPTR0	pInA	F32*	ABMP	See description above
9	HSP_DCMDPTR1	pInB	F32*	ABMP	
10	HSP_DCMDPTR1	pOut	F32*	ABMP	

1. *Parameters must be written in the specified order.*
2. *Refer to Section 5: Processing functions description for acronym details*

5.9.8 MAT_TRANS_F32

The matrix transpose can be either executed as a direct command (accelerator mode) or programmed into a processing list.

Transposing an M-by-N matrix gives an N-by-M matrix.

Refer to Section 2.2.1: Vector, matrix operations, and formats for data buffer formats.

5.9.8.1 *Command format to record MAT_TRANS_F32*

Table 183. Command format to record MAT_TRANS_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	MAT_TRANS_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP,	Pointer on input matrix A
HSP_PARAMR1	pOut	F32*	ABMP, POP,	Pointer on output matrix
HSP_PARAMR2	ROW	U32	-	Number of matrix rows
HSP_PARAMR3	COL	U32	-	Number of matrix columns
HSP_PARAMR[14:4]	-	-	-	-

Registers	Acronyms	Type	Target	Description
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input matrix Must contain a real ROW-by-COL matrix in float32.
pOut	Pointer on output matrix The result is a real NROW -by-NCOL matrix in float32.
ROW	Number of rows of the input matrix COL x ROW must be lower than or equal to 4096.
COL	Number of columns of the input matrix A, NCOL x NROW must be lower than or equal to 4096.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pOut target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. Refer to Section 2.2.6: IOTYPE description for details

5.9.8.2 *MAT_TRANS_F32 direct command format*

The next table shows the command format when MAT_TRANS_F32 is executed as a direct command.

Table 184. MAT_TRANS_F32 direct command format

Order	Registers	Acronyms	Type	Target	Description
1	HSP_DCMDIDR	MAT_TRANS_F32	U32	-	See Section 8: Function ID list
2	HSP_PARAMR2	ROW	U32	-	See description above
3	HSP_PARAMR3	COL	U32	-	
4	HSP_DCMDPTR0	pIn	F32*	ABMP	
5	HSP_DCMDPTR1	pOut	F32*	ABMP	

1. *Parameters must be written in the specified order.*
2. *Refer to Section 5: Processing functions description for acronym details*

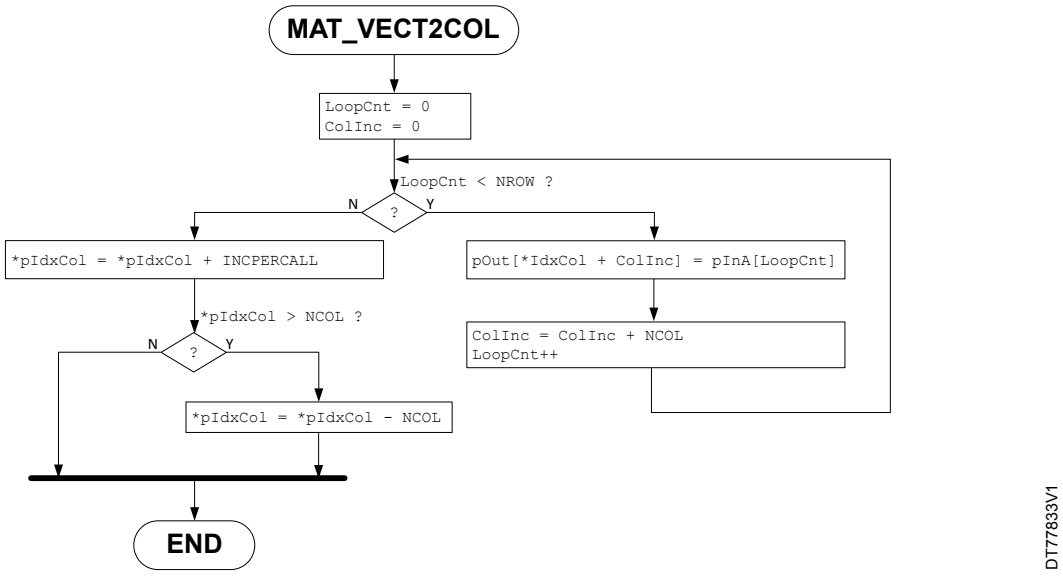
5.9.9 MAT_VECT2COL

This function can be either executed as a direct command (accelerator mode) or programmed into a processing list (TBC).

This function copies NROW elements of a vector into a column of a NROW-by-NCOL matrix. The column number is given by the content of IdxCol. Then the function increments the content of IdxCol by INCPERCALL. Figure 31 shows a simplified view of the process performed by this function.

Refer to Section 2.2.1: Vector, matrix operations, and formats for data buffer formats.

Figure 31. **MAT_VECT2COL process**



5.9.9.1 *Command format to record MAT_VECT2COL*

Table 185. **Command format to record MAT_VECT2COL**

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	MAT_VECT2COL	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	X32*	ABMP, POP	Pointer on vector to read
HSP_PARAMR1	pIdxCol	U32*	ABMP, POP	Pointer on next column index
HSP_PARAMR2	pOut	X32*	ABMP, POP	Pointer on output matrix
HSP_PARAMR3	ROW	U32	-	Number of matrix rows
HSP_PARAMR4	COL	U32	-	Number of matrix columns
HSP_PARAMR5	INCPERCALL	U32	-	Increment value per function call
HSP_PARAMR[14:6]	-	-	-	-
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on the vector to read Must contain at least real ROW elements in float32 or uint32_t, or int32_t.
pIdxCol	Pointer on variable containing the next column index. Must be an integer value. Must be initialized by the application for the first iteration.
pOut	Pointer on the output matrix to write The matrix must be ROW-by-COL in float32 or uint32_t, or int32_t.
ROW	Number of rows of the input matrix COL x ROW must be lower than or equal to 4096.
COL	Number of columns of the input matrix A, NCOL x NROW must be lower than or equal to 4096.

INCPERCALL	pOut increment between two calls of MAT_VECT2COL function Must be a positive integer
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pIdxCol target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pOut target: If POP2 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. Refer to Section 2.2.6: IOTYPE description for details

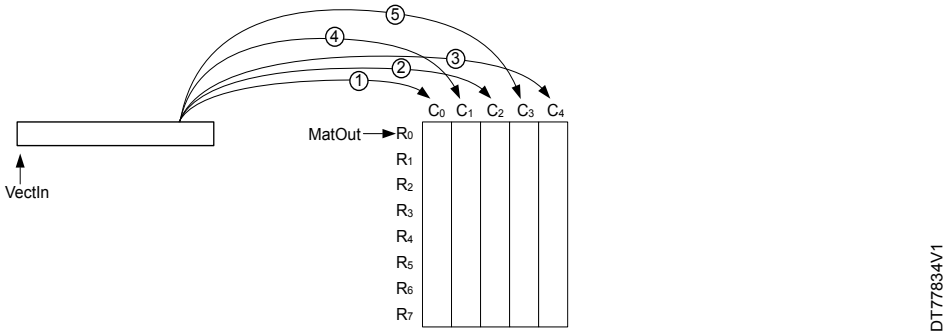
5.9.9.2 *Example*

When the MAT_VECT2COL function is programmed with the parameters listed below, the result is the following:

1. During the first call, the content of VectIn is copied into the first column.
2. During the second call, the content of VectIn is copied into the third column.
3. During the third call, the content of VectIn is copied into the fifth column.
4. During the fourth call, the content of VectIn is copied into the second column.
5. During the fifth call, the content of VectIn is copied into the fourth column.
6. During the sixth call, the content of VectIn is copied into the first column.
7. ...

HSP_PARAM0 = &VectIn; // Address of input vector
HSP_PARAM1 = &IdxCol; // Address of column index
HSP_PARAM2 = &MatOut; // Address of matrix output
HSP_PARAM3 = 8; // NROW
HSP_PARAM4 = 5; // NCOL
HSP_PARAM5 = 2; // INCPERCALL
HSP_PARAM615 = 0; // All addresses are pointing into BRAM

Figure 32. MAT_VECT2COL process



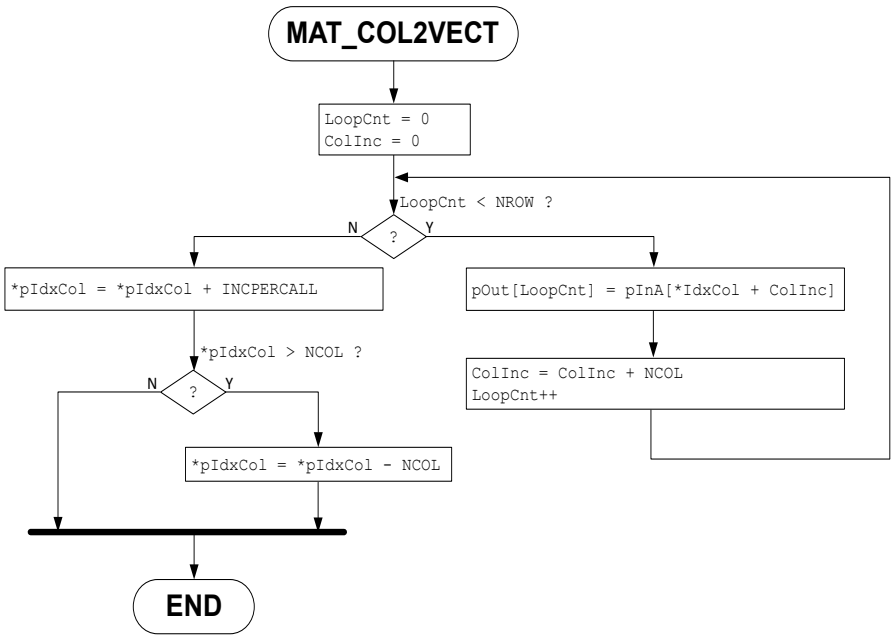
5.9.10 MAT_COL2VECT

This function can be either executed as a direct command (accelerator mode) or programmed into a processing list (TBC).

This function copies a column of a NROW-by-NCOL matrix into a vector. The column number is given by the content of IdxCol. Then the function increments the content of IdxCol by INCPERCALL. The Figure 31 shows a simplified view of the process performed by this function.

Refer to Section 2.2.1: Vector, matrix operations, and formats for data buffer formats.

Figure 33. **MAT_COL2VECT process**



5.9.10.1 *Command format to record MAT_COL2VECT*

Table 186. **Command format to record MAT_COL2VECT**

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	MAT_COL2VECT	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pIn	X32*	ABMP, POP	Pointer on matrix to read
HSP_PARAMR1	pIdxCol	U32*	ABMP, POP	Pointer on next column index
HSP_PARAMR2	pOut	X32*	ABMP, POP	Pointer on output vector
HSP_PARAMR3	ROW	U32	-	Number of matrix rows
HSP_PARAMR4	COL	U32	-	Number of matrix columns
HSP_PARAMR5	INCPERCALL	U32	-	Increment value per function call
HSP_PARAMR[14:6]	-	-	-	-
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pIn	Pointer on the matrix to read The matrix must be NROW-by-NCOL in float32 or uint32_t, or int32_t.
pIdxCol	Pointer on variable containing the next column index. Must be an integer value. Must be initialized by the application for the first iteration.
pOut	Pointer on output vector The vector size must be at least ROW elements in float32 or uint32_t, or int32_t.
ROW	Number of rows of the input matrix COL x ROW must be lower than or equal to 4096.
COL	Number of columns of the input matrix A, NCOL x NROW must be lower than or equal to 4096.

INCPERCALL	pOut increment between two calls of MAT_VECT2COL function Must be a positive integer, lower than COL
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pIdxCol target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pOut target: If POP2 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. Refer to Section 2.2.6: IOTYPE description for details

5.10 Transform functions

The function ID of each function is provided in Section 8: Function ID list.

5.10.1 FFT_F32

The FFT_F32 function can either be executed as a direct command (accelerator mode) or programmed into a processing list.

It performs an in-place FFT transform of complex or real vector, according to command arguments. Input and output buffers are the same. The real FFT takes advantage of the symmetry properties of the DFT on real numbers, and has speed and memory footprint advantages compared to complex algorithms.

The FFT_F32 performed on a real vector of N elements returns N/2 +1 complex values representing the spectrum of the input signal from 0 to fs/2.

The FFT_F32 function offers the following options:

- Performing the bit reverse of not
- Performing the IFFT or not
- Four different output formats

The FFT_F32 uses a radix-4 stage or radix-2 stage according to the need.

The function uses the standard FFT definition and output values grow by a factor of FFTSIZE when computing the forward transform. The inverse transform includes a scale of 1/FFTSIZE as part of the calculation.

The supported FFT sizes are 32, 64, 128, 256, 512, 1024, 2048, and 4096 points. Depending on the amount of BRAM, only a subset of those values can be used. The twiddle factor table is included into the HSP, no need to precalculate it.

5.10.1.1 Command format to record FFT_F32

Table 187. Command format to record FFT_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	FFT_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInOut	F32*	ABMP	Pointer on FFT data buffer
HSP_PARAMR1	IFFT	U32	-	Inverse FFT
HSP_PARAMR2	FFTSIZE	U32	-	FFT size
HSP_PARAMR3	FMT	U32	-	Complex/real FFT
HSP_PARAMR4	BITREV	U32	-	Bit reverse
HSP_PARAMR[15:5]	-	-	-	-

1. *Refer to Section 5: Processing functions description for acronym details*

pInOut	Pointer on FFT data buffer This buffer must contain the real or complex vector (in float32) where the FFT must be performed. The Fourier transform of the input vector is provided into this buffer. The input values are lost. When the operation is completed, this buffer contains the FFT result. Must be located into the BRAM-AB. See FFT input data format.
IFFT	Inverse FFT 0: Perform FFT 1: Perform inverse FFT
FFTSIZE	FFT size This parameter defines the FFT size. The accepted values are: 32, 64, 128, 256, 512, 1024, 2048 and 4096
FMT	Complex/real FFT 0: CFFT (Input vector is complex) 1: RFFT: outputs a data buffer containing N/2 complex values. 2: RFFT: outputs a data buffer of N/2 complex values, but the imaginary part of the first value contains the real part of the sample N/2. 3: RFFT: outputs a data buffer of N/2+1 complex values. See Table 191. FFT and RFFT output data format for details.
BITREV	Bit reverse 0: Bit reverse not performed 1: Bit reverse performed

5.10.2 CFFT_F32

The next table shows the command format to execute complex FFT (CFFT) as a direct command.

Table 188. CFFT_F32

Order	Registers	Acronyms	Type	Target	Description
1	HSP_DCMDIDR	CFFT_F32	U32	-	See Section 8: Function ID list
2	HSP_PARAMR2	FFTATTR	U32	-	FFT attribute
3	HSP_PARAMR3	FFTSIZE	U32	-	FFT size
4	HSP_PARAMR4	LOG2NBP	U32	-	Log2 FFT size
5	HSP_PARAMR5	IFFT	U32	-	Inverse FFT
6	HSP_DCMDPTR0	pInOut	F32*	ABMP	

1. *Parameters must be written in the specified order.*
2. *Refer to Section 5: Processing functions description for acronym details*

pInOut	Pointer on FFT data buffer This buffer must contain the complex vector (in float32) where the FFT must be performed. This buffer is also used as working buffer for the FFT. The Fourier transform of the input vector is provided into this buffer. The input values are lost. Must be located into the BRAM-AB.
FFTATTR	FFT attribute: This value is split into 2 fields: FFTATTR[5] and FFTATTR[4:0]. FFTATTR[5]: Direct or Inverse FFT 0: Direct FFT 1: Inverse FFT FFTATTR[4:0]: Bit reverse control

	0: No bit reverse Log2(FFTSIZE): bit reverse
FFTSIZE	FFT size This parameter defines the number of points. The accepted values are: 32, 64, 128, 256, 512, 1024, 2048 and 4096
LOG2NBP	LOG2NBP must be: 4: for 32-point FFT 5: for 64-point FFT 6: for 128-point FFT 7: for 256-point FFT 8: for 512-point FFT 9: for 1024-point FFT 10: for 2048-point FFT 11: for 4096-point FFT
IFFT	Inverse FFT 0: Perform FFT 1: Perform inverse FFT

5.10.3 RFFT_F32

The next table shows the command format when real FFT (RFFT) is executed as a direct command.

Table 189. RFFT_F32

Order	Registers	Acronyms	Type	Target	Description
1	HSP_DCMDIDR	RFFT_F32	U32	-	See Section 8: Function ID list
2	HSP_PARAMR2	RFFTATTR	U32	-	FFT attribute
3	HSP_PARAMR3	RFFTSIZE	U32	-	FFT size / 2
4	HSP_PARAMR4	LOG2NBP	U32	-	Log2 of FFT size
5	HSP_PARAMR5	IRFFT	U32	-	Inverse FFT
6	HSP_PARAMR6	FMT	U32	-	RFFT format
7	HSP_DCMDPTR0	pInOut	F32*	ABMP	Pointer on RFFT data buffer

1. *Parameters must be written in the specified order.*
2. *Refer to Section 5: Processing functions description for acronym details*

pInOut	Pointer on RFFT data buffer This buffer must contain the real or complex vector (in float32) where the FFT must be performed. This buffer is also used as working buffer for the FFT. The Fourier transform of the input vector is provided into this buffer. The input vector is lost. Must be located into the BRAM-AB. See Section 5.10.1: FFT_F32.
RFFTATTR	RFFT attribute: This value is split into 2 fields: FFTATTR[5] and FFTATTR[4:0]. FFTATTR[5]: Direct or Inverse FFT 0: Direct FFT 1: Inverse FFT FFTATTR[4:0]: Bit reverse control 0: No bit reverse Log2(FFTSIZE) - 1: bit reverse

RFFTSIZE	RFFT size This parameter must give the number of points divided by 2. The accepted values are: 16: for 32-point RFFT 32: for 64-point RFFT 64: for 128-point RFFT 128: for 256-point RFFT 256: for 512-point RFFT 512: for 1024-point RFFT 1024: for 2048-point RFFT 2048: for 4096-point RFFT
LOG2NBP	LOG2NBP must be: 3: for 32-point RFFT 4: for 64-point RFFT 5: for 128-point RFFT 6: for 256-point RFFT 7: for 512-point RFFT 8: for 1024-point RFFT 9: for 2048-point RFFT 10: for 4096-point RFFT
IRFFT	Inverse RFFT 0: RFFT 1: Inverse RFFT
FMT	RFFT format 1: RFFT: outputs a data buffer containing N/2 complex values. 2: RFFT: outputs a data buffer of N/2 complex values, but the imaginary part of the first value contains the real part of the sample N/2. 3: RFFT: outputs a data buffer of N/2+1 complex values. See Table 191. FFT and RFFT output data format for details.

The table below gives the minimum size that the buffer pointed by pInOut must have versus FFTSIZE.

Table 190. BRAM-AB size versus FFTSIZE

Footprint	FMT	FFTSIZE							
		32	64	128	256	512	1024	2048	4096
InOut buffer size (Bytes)	0	256	512	1024	2048	4096	8192	16384	32768
Requested BRAM size (Bytes)		512	1024	2048	4096	8192	16384	32768	65536
InOut buffer size (Bytes)	1 or 2	128	256	512	1024	2048	4096	8192	16384
Requested BRAM size (Bytes)		256	512	1024	2048	4096	8192	16384	32768
InOut buffer size (Bytes)	3	136	264	520	1032	2056	4104	8200	16392
Requested BRAM size (Bytes)		272	528	1040	2064	4112	8208	16400	32784

1. *FMT=0 for CFFT, FMT=1,2 for RFFT with N/2 points, FMT=3 for RFFT with N/2+1 points*

The next table shows the data format provided by the RFFT according to the selected format (FMT). Complex data are represented by $x = a_k + jb_k$ where a_k is the real part and b_k the imaginary part.

Table 191. FFT and RFFT output data format

Buffer index	Output format				Comments
	FMT = 0	FMT = 1	FMT = 2	FMT = 3	
0	a_0	a_0	a_0	a_0	DC Offset
1	b_0	$b_0 = 0$	$b_0 = a_{N/2}$	$b_0 = 0$	-
2	a_1	a_1	a_1	a_1	Real value of frequency component at DF
3	b_1	b_1	b_1	b_1	Imaginary value of frequency component at DF
...	...	...	...	...	...
N-1	$b_{N/2-1}$	$b_{N/2-1}$	$b_{N/2-1}$	$b_{N/2-1}$	Imaginary value of frequency component at (N/2-1)*DF
N	$a_{N/2}$	-	-	$a_{N/2}$	-
N+1	$b_{N/2}$	-	-	$b_{N/2} = 0$	-
...	...	-	-	-	...
2N-2	a_{N-1}	-	-	-	-
2N-1	b_{N-1}	-	-	-	-

1. *With BITREV = 1*

5.10.4 DCT_F32

The DCT_F32 function can be either executed as a direct command (accelerator mode) or programmed into a processing list.

The function performs the discrete cosine transform of a real vector. It is possible to perform a DCT type II or DCT type III for the inverse DCT. Input and output buffers are the same.

The supported DCT size are 64, 128, 256, 512, 1024, 2048, and 4096 points. Depending on the amount of BRAM, only a subset of those values can be used. The twiddle factor table is included into the HSP.

The bit-reverse operation is always performed.

DCT type II

The DCT-II is probably the most commonly used form.

The expression of the DCT-II is the following:

$$X_k = \sum_{n=0}^{N-1} x_n \cdot \cos\left[\frac{\pi}{N} \cdot \left(n + \frac{1}{2}\right) \cdot k\right]$$

with k = 0,..., N-1

DCT type III

The DCT-III is also referred to as the reverse DCT-II.

The expression of the DCT-III is the following:

$$X_k = \frac{x_0}{2} + \sum_{n=1}^{N-1} x_n \cdot \cos\left[\frac{\pi}{N} \cdot \left(k + \frac{1}{2}\right) \cdot n\right]$$

with k = 0,..., N-1

5.10.4.1 *Command format to record DCT_F32*

Table 192. Command format to record DCT_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	DCT_F32	U32	-	See Section 8: Function ID list

Registers	Acronyms	Type	Target	Description
HSP_PARAMR0	pInOut	F32*	ABMP	Pointer on DCT data buffer
HSP_PARAMR1	-	-	-	-
HSP_PARAMR2	DCTSIZE	U32	-	DCT size
HSP_PARAMR[15:3]	-	-	-	-

1. *Refer to Section 5: Processing functions description for acronym details*

pInOut	Pointer on DCT data buffer The buffer must contain the real vector in float32. This buffer is also used as working buffer for the DCT. The result is also provided into this buffer. The input vector is lost. This buffer must be located into the internal BRAM. The size of this buffer must be 2 x DCT size
DCTSIZE	DCT size The accepted values are: 32: for 32-point RFFT 64: for 64-point RFFT 128: for 128-point RFFT 256: for 256-point RFFT 512: for 512-point RFFT 1024: for 1024-point RFFT 2048: for 2048-point RFFT 4096: for 4096-point RFFT

5.10.4.2 *DCT direct command format*

The next table shows the command format when DCT is executed as a direct command.

Table 193. DCT direct command format

Order	Registers	Acronyms	Type	Target	Description
1	HSP_DCMDIDR	DCT_F32	U32	-	See Section 8: Function ID list
2	HSP_PARAMR2	DCTATTR	U32	-	DCT attribute
3	HSP_PARAMR3	DCTSIZE	U32	-	DCT Size
4	HSP_PARAMR4	LOG2NBP	U32	-	DCT log2 size
5	HSP_PARAMR5	IDCT	U32	-	Reverse DCT
7	HSP_DCMDPTR0	pInOut	F32*	ABMP	-

1. *Parameters must be written in the specified order.*
2. *Refer to Section 5: Processing functions description for acronym details*

pInOut	Pointer on DCT data buffer The buffer must contain the real vector in float32. This buffer is also used as working buffer for the DCT. The result is also provided into this buffer. The input vector is lost. This buffer must be located into the internal BRAM. The size of this buffer must be 2 x DCTSIZE
DCTATTR	DCT attribute This filed must have the same value as LOG2NBP

DCTSIZE	DCT size The accepted values are: 64, 128, 256, 512, 1024, 2048 and 4096
LOG2NBP	DCT size 6 = DCT 64 points 7 = DCT 128 points 8 = DCT 256 points 9 = DCT 512 points 10 = DCT 1024 points 11 = DCT 2048 points 12 = DCT 4096 points
IDCT	Inverse DCT Must be set to 0.

5.10.5 IDCT_F32

The IDCT_F32 function can be only recorded into a processing list.

It performs the inverse discrete cosine transform of a real vector. Input and output buffers are the same.

The supported IDCT sizes are 64, 128, 256, 512, 1024, 2048, and 4096 points. Depending on the amount of BRAM, only a subset of those values can be used. The twiddle factor table is included into the HSP.

The bit-reverse operation is always performed.

The inverse DCT-II is also referred to as DCT-III.

The expression of the DCT-III is the following:

$$X_k = \frac{x_0}{2} + \sum_{n=1}^{N-1} x_n \cdot \cos\left[\frac{\pi}{N} \cdot \left(k + \frac{1}{2}\right) \cdot n\right]$$

with k = 0,..., N-1

5.10.5.1 Command format to record IDCT_F32

Table 194. Command format to record IDCT_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	IDCT_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInOut	F32*	ABMP	Pointer on IDCT data buffer
-	-	-	-	-
HSP_PARAMR2	DCTSIZE	U32	-	DCT size
HSP_PARAMR[15:3]	-	-	-	-

1. *Refer to Section 5: Processing functions description for acronym details*

pInOut	Pointer on IDCT data buffer The buffer must contain the real vector in float32. This buffer is also used as working buffer for the DCT. The result is also provided into this buffer. The input vector is lost. This buffer must be located into the internal BRAM. The size of this buffer must be 2 x IDCT size
DCTSIZE	IDCT size The accepted values are: 32: for 32-point RFFT

	64: for 64-point RFFT 128: for 128-point RFFT 256: for 256-point RFFT 512: for 512-point RFFT 1024: for 1024-point RFFT 2048: for 2048-point RFFT 4096: for 4096-point RFFT

5.10.5.2 *IDCT_F32 direct command format*

Table 195. IDCT_F32 direct command format

Order	Registers	Acronyms	Type	Target	Description
1	HSP_DCMDIDR	IDCT_F32	U32	-	See Section 8: Function ID list
2	HSP_PARAMR2	DCTATTR	U32	-	IDCT attribute
3	HSP_PARAMR3	DCTSIZE	U32	-	IDCT Size
4	HSP_PARAMR4	LOG2NBP	U32	-	IDCT log2 size
5	HSP_PARAMR5	IDCT	U32	-	Inverse DCT
7	HSP_DCMDPTR0	pInOut	F32*	ABMP	-

1. *Parameters must be written in the specified order.*
2. *Refer to Section 5: Processing functions description for acronym details*

pInOut	Pointer on DCT data buffer The buffer must contain the real vector in float32. This buffer is also used as working buffer for the DCT. The result is also provided into this buffer. The input vector is lost. This buffer must be located into the internal BRAM. The size of this buffer must be 2 x DCTSIZE
DCTATTR	IDCT attribute This filed must have the same value as LOG2NBP | 0x20
DCTSIZE	IDCT size The accepted values are: 64, 128, 256, 512, 1024, 2048 and 4096
LOG2NBP	IDCT size 6 = IDCT 64 points 7 = IDCT 128 points 8 = IDCT 256 points 9 = IDCT 512 points 10 = IDCT 1024 points 11 = IDCT 2048 points 12 = IDCT 4096 points
IDCT	Inverse DCT Must be set to 1.

5.11 Conditional functions

Conditional functions are a group of special functions that can be used into processing lists, to perform conditional execution of processing functions. Those functions are directly in-lined into the processing lists.

The function ID of each function is provided on Section 8: Function ID list.

5.11.1 __IF_ELSE

This in-line conditional function can only be included into a processing list.

__IF_ELSE(IF) checks the defined condition between the variable InA and Val or between InA and a constant value.

- If the comparison result is true, then the processing functions after the __IF_ELSE(IF) are executed until the comparison functions __IF_ELSE(ELSE) or __IF_ELSE(ENDIF) is met.
- If the comparison result is false, then the processing functions after the __IF_ELSE (IF) till the next __IF_ELSE(ELSE) or __IF_ELSE(ENDIF) are skipped.

Up to 15 nested __IF_ELSE(IF) are supported.

The same processing function is used to insert a IF or ELSE or ENDIF into processing lists.

5.11.1.1 Command format to record __IF_ELSE

Table 196. Command format to record __IF_ELSE

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	__IF_ELSE	U32	-	See Section 8: Function ID list
HSP_PARAMR0	COMPTYPE	U32	-	Comparison type
HSP_PARAMR1	pIn	X32*	ABMP, IBP	Pointer on data to compare
HSP_PARAMR2	pVal	X32* X32	ABMP, IBP	Constant or variable address for comparison.
HSP_PARAMR3	VALTYPE	U32	-	Indicate pVal type
HSP_PARAMR[15:4]	-	-	-	-

1. *Refer to Section 5: Processing functions description for acronym details*

COMPTYPE	Type of comparison: 0x0000: IF equal for float32 (IFEQ_F32) 0x0001: IF equal for int32 (IFEQ_I32) 0x0002: IF not equal float32 (IFNE_F32) 0x0003: IF not equal int32 (IFNE_I32) 0x0004: IF greater than for float32 (IFGT_F32) 0x0005: IF greater than for int32 (IFGT_I32) 0x0006: IF greater than for unsigned int32 (IFGT_U32) 0x0007: IF less than for float32 (IFLT_F32) 0x0008: IF less than for int32 (IFLT_I32) 0x0009: IF less than for unsigned int32 (IFLT_U32) 0x000A: IF greater or equal for float32 (IFGE_F32) 0x000B: IF greater or equal for int32 (IFGE_I32) 0x000C: IF greater or equal for unsigned int32 (IFGE_U32) 0x000D: IF less or equal for float32 (IFLE_F32) 0x000E: IF less or equal for int32 (IFLE_I32) 0x000F: IF less or equal for unsigned int32 (IFLE_U32) 0x0010: IF value is odd (IFODD_I32) 0x0020: IF value is even (IFEVEN_I32) 0x0100: ELSE 0x0200: ENDIF

pIn	Pointer on data to compare First variable of the comparison. Not significant when COMPTYPE = 0x100 or 0x200.
pVal	Constant value or pointer according to VALTYPE Second member of the comparison. Not significant when COMPTYPE = 0x100 or 0x200.
VALTYPE	Indicates the type of pVal 0: pVal is a pointer 1: pVal is a constant Not significant when COMPTYPE = 0x100 or 0x200.

See example in Section 7.7: __IF_ELSE example.

5.11.2 __IF_LOOP

This in-line conditional function can only be included into a processing list, between __LOOP(LOOP) and __LOOP(LOOP_END).

This function checks the defined condition between the LOOP variable and pIn or a constant value.

- If the comparison result is true, then the processing functions after the __IF_LOOP(IF) function are executed until __IF_LOOP(ELSE) or __IF_LOOP(ENDIF) is met.
- If the comparison result is false, then the processing functions after the __IF_LOOP(ELSE) (if it exists) till __IF_LOOP(ENDIF) are executed.

Up to 15 nested __IF_LOOP(IF) are supported.

The same processing function is used to insert a IF or ELSE or ENDIF into processing lists.

5.11.2.1 Command format to record __IF_LOOP

Table 197. Command format to record __IF_LOOP

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	__IF_LOOP	U32	-	See Section 8: Function ID list
HSP_PARAMR0	COMPTYPE	U32	-	Comparison type
HSP_PARAMR1	-	-	-	-
HSP_PARAMR2	pVal	X32* X32	ABMP, IBP	Constant or pointer for comparison
HSP_PARAMR3	VALTYPE	U32	-	Indicates pVal type
HSP_PARAMR[15:4]	-	-	-	-

1. *Refer to Section 5: Processing functions description for acronym details*

COMPTYPE	Type of comparison: 0: IF loop counter is equal (IFEQ) 1: IF loop counter is not equal (IFNE) 2: IF loop counter is greater than (IFGT) 3: IF loop counter is less than (IFLT) 4: IF loop counter is greater or equal (IFGE) 5: IF loop counter is less or equal (IFLE) 6: ELSE 7: ENDIF 8: IF loop counter is odd (IFODD)

	9: IF loop counter is even (IFEVEN)
pVal	Constant value, or pointer, according to VALTYPE Second member of the comparison. This parameter can be omitted if COMPTYPE = 6 or 7.
VALTYPE	Indicates the type of pVal 1: pVal is a pointer on a uint32_t variable 2: pVal is a pointer on a int32_t variable 3: pVal is a pointer on a float32_t variable 5: pVal is a uint32_t constant 6: pVal is an int32_t constant 7: pVal is a float32_t constant This parameter can be omitted if COMPTYPE = 6 or 7.

See example in Section 7.8: __LOOP example.

5.11.3 __LOOP

This in-line conditional function can only be included into a processing list.

__LOOP allows the repetition of the group of processing functions located between __LOOP(__LOOP) and __LOOP(LOOP_END).

Up to 15 nested __LOOP are supported.

The same processing function is used to insert a LOOP or LOOP_END into processing lists.

5.11.3.1 Command format to record __LOOP

Table 198. Command format to record __LOOP

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	__LOOP	U32	-	See Section 8: Function ID list
HSP_PARAMR0	TYPE	U32	-	LOOP command type
HSP_PARAMR1	COUNT	U32	-	Number of iteration to execute
HSP_PARAMR[15:2]	-	-	-	-

1. *Refer to Section 5: Processing functions description for acronym details*

TYPE	LOOP command type: 0: LOOP_START 1: LOOP_END
COUNT	Number of iteration to execute COUNT must be between 1 and 65535 This parameter can be omitted if TYPE = 1.

5.11.4 COUNT

This function can only be included into a processing list.

COUNT allows the application to use up to 15 global counters located into the DRAM. This command works with IF_COUNT and SET_COUNT. When HSP is started all counters initial values are set to 0. SET_COUNT command allows the application to change the counters value.

The command COUNT does the following operation:

- if CMD = 1 then counter[IDX] = counter[IDX] + VAL
- if CMD = 0, then counter[IDX] = VAL

5.11.4.1 *Command format to record COUNT*

Table 199. Command format to record COUNT

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	COUNT	U32	-	See Section 8: Function ID list
HSP_PARAMR0	IDX	U32	-	Counter instance
HSP_PARAMR1	VAL	U32	-	Value to set or add
HSP_PARAMR2	CMD	U32	-	Counter command
HSP_PARAMR[15:3]	-	-	-	-

1. *Refer to Section 5: Processing functions description for acronym details*

IDX	Counter instance: Must be an integer value between 0 and 14
VAL	Value to set or add VAL must be a positive integer number. If CMD = 0, counter[IDX] is set to VAL If CMD = 1, VAL is added to counter[IDX]
CMD	Counter command If CMD = 0, counter[IDX] is set to VAL If CMD = 1, VAL is added to counter[IDX]

5.11.5 __IF_COUNT

This in-line function can only be included into a processing list.

__IF_COUNT(IF) checks the defined condition between the variable counter[IDX] and COMPVAL.

- If the comparison result is true, then the processing functions after the IF_COUNT(IF) are executed until the functions __IF_COUNT(ELSE) or __IF_COUNT(ENDIF) is met.
- If the comparison result is false, then the processing functions after the __IF_COUNT (IF) till the next __IF_COUNT(ELSE) or __IF_COUNT(ENDIF) are skipped.

Up to 15 nested __IF_COUNT(IF) are supported.

The same processing function is used to insert a IF or ELSE or ENDIF into processing lists.

5.11.5.1 *Command format to record __IF_COUNT*

Table 200. Command format to record __IF_COUNT

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	__IF_COUNT	U32	-	See Section 8: Function ID list
HSP_PARAMR0	COMPTYPE	U32	-	Type of comparison
HSP_PARAMR1	COMPVAL	U32	-	Comparison value
HSP_PARAMR2	-	-	-	-
HSP_PARAMR3	IDX	U32	-	Counter instance
HSP_PARAMR[15:4]	-	-	-	-

1. *Refer to Section 5: Processing functions description for acronym details*

COMPTYPE	Type of comparison: 0: IF counter[IDX] is equal (IFEQ)

	1: IF counter[IDX] is not equal (IFNE) 2: IF counter[IDX] is greater than (IFGT) 3: IF counter[IDX] is less than (IFLT) 4: IF counter[IDX] is greater or equal (IFGE) 5: IF counter[IDX] is less or equal (IFLE) 6: ELSE 7: ENDIF 8: IF counter[IDX] is odd (IFODD) 9: IF counter[IDX] is even (IFEVEN)
COMPVAL	Constant value to be compared to counter[IDX] This parameter can be omitted if COMPTYPE = 6 or 7.
IDX	Counter instance: Must be an integer value between 0 and 14

5.11.6 SET_COUNT

This function can only be used as a direct command.

When HSP is started all counters initial values are set to 0. SET_COUNT command allows the application to change the counters value.

5.11.6.1 SET_COUNT direct command format

Table 201. SET_COUNT direct command form

Order	Registers	Acronyms	Type	Target	Description
1	HSP_DCMDIDR	SET_COUNT	U32	-	See Section 8: Function ID list
2	HSP_PARAMR0	IDX	U32	-	Counter instance
3	HSP_PARAMR1	VAL	U32	-	Counter value to set
4	HSP_DCMDPTR0	START	-	ABMP	To start execution

1. *Parameters must be written in the specified order.*
2. *Refer to Section 5: Processing functions description for acronym details*

IDX	Counter instance: Must be an integer value between 0 and 14
VAL	Counter value to set Must be a positive integer value
START	Start execution Any value written into HSP_DCMDPTR0 starts command execution.

5.11.7 __CLR_SATF

This in-line function can only be included into a processing list.

When a saturation is performed the flag FPUSATF of HSP_EVT_ISR register is set to 1. The FPUSATF flag stores the saturation status (**spe_fpu_sat**) returned by the function __SCA_SAT_F32. The flag FPUSATF is cleared by the application when the bit FPUSATC of the HSP_EVT_ICR register is set to 1.

The saturation status (**spe_fpu_sat**) is cleared every time a processing function starts to be executed but not when an in-line processing function starts to be executed. This function clears the saturation status.

5.11.7.1 *Command format to record __CLR_SATF*

Table 202. Command format to record __CLR_SATF

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	__CLR_SATF	U32	-	See Section 8: Function ID list
HSP_PARAMR[15:2]	-	-	-	Not used

1. *Refer to Section 5: Processing functions description for acronym details*

5.11.8 __GET_SATF

This in-line function can only be included into a processing list.

This function writes the value 0x40 into pStat if the saturation status is activated; otherwise, it writes 0.

5.11.8.1 *Command format to record __GET_SATF*

Table 203. Command format to record __GET_SATF

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	__GET_SATF	U32	-	See Section 8: Function ID list
HSP_PARAMR0	-	-	-	-
HSP_PARAMR1	pStat	U32*	ABMP, POP, IBP	Pointer on saturation status variable
HSP_PARAMR[14:2]	-	-	-	Not used
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pStat	Variable containing the saturation status: 0: Saturation status is not active 0x40: Saturation status is active
IOTYPE	Input and output buffer types: Indicates the pointers type pInStat target: If POP0 = 1, it is a pointer-on-pointer, if GMP0 = 1, it is a global memory pointer, otherwise it is an ABMP or IBP pointer . Refer to Section 2.2.6: IOTYPE description for details

5.12 CNN functions

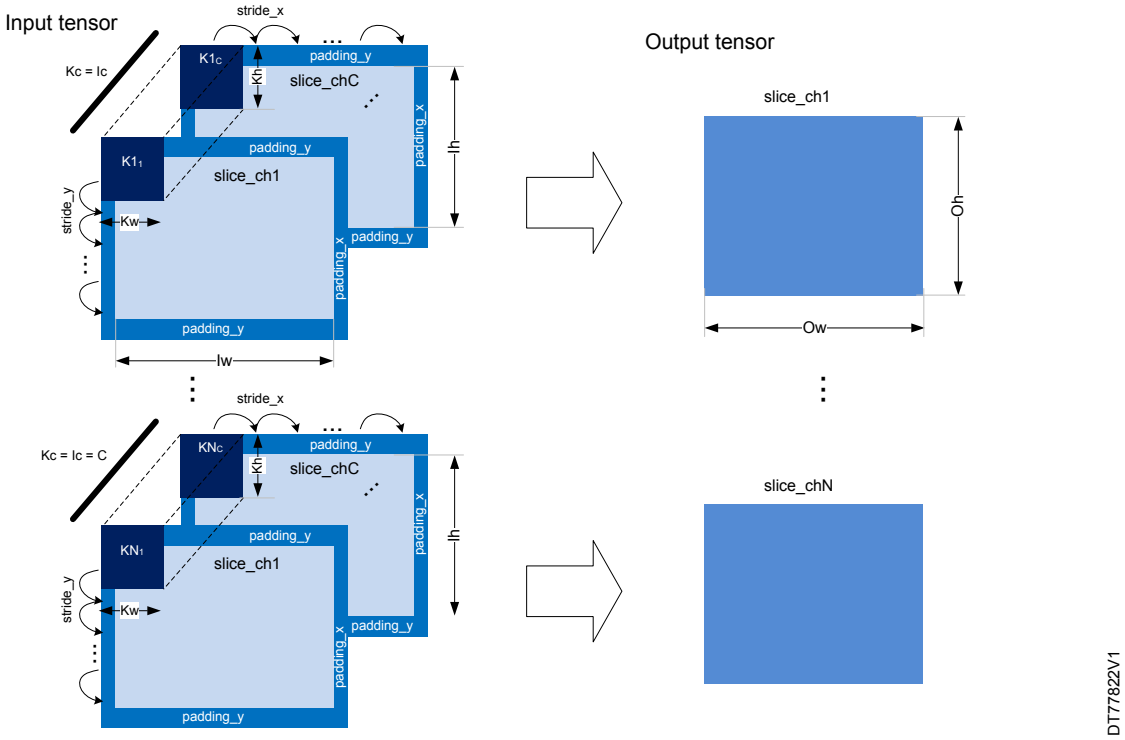
The function ID of each function is provided on Section 8: Function ID list.

The figure below shows a simplified view of the operations performed by this function. In the figure:

- The input tensor format is: 1 x Ih x Iw x C
- The kernel tensor format is: N x Kh x Kw x C
- The output tensor format is: 1 x Oh x Ow x N

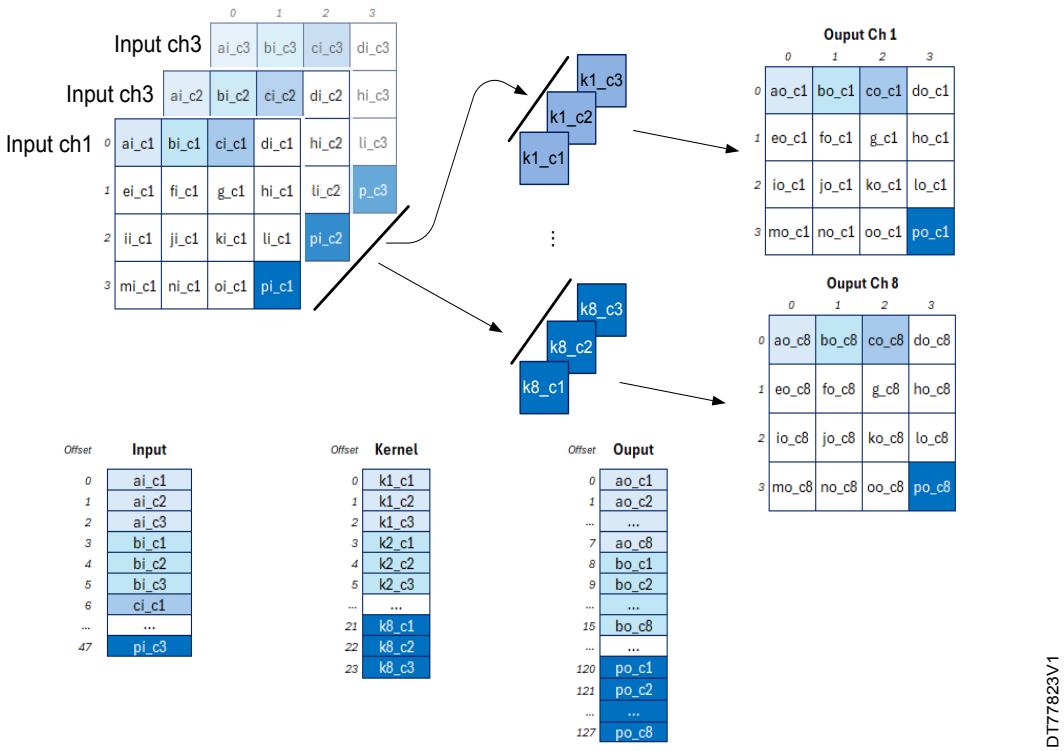
The padding is not handled by the processing function and must be handled externally.

Figure 34. Convolution 2D simplified process



The next figure shows the data arrangement into memory using a simple convolution case of an input tensor having three channels, a kernel filter 8x1x1x3.

Figure 35. Data arrangement



5.12.1 CNN_CONV2D_I8

The CNN_CONV2D_I8 can only be executed into as direct command.
This function performs a 2D convolution on signed 8-bit data tensor.

5.12.1.1 CNN_CONV2D_I8 direct command format

Table 204. CNN_CONV2D_I8 direct command format

Order	Registers	Acronyms	Type	Target	Description
1	HSP_DCMDIDR	CNN_CONV2D_I8	U32	-	See Section 8: Function ID list
2	HSP_PARAMR0	IN_W	U32	-	Input tensor width
3	HSP_PARAMR1	IN_H	U32	-	Input tensor height
4	HSP_PARAMR2	IN_C	U32	-	Input tensor channel number
5	HSP_PARAMR3	K_W	U32	-	Kernel tensor width
6	HSP_PARAMR4	K_H	U32	-	Kernel tensor height
7	HSP_PARAMR5	OUT_W	U32	-	Output tensor width
8	HSP_PARAMR6	OUT_H	U32	-	Output tensor height
9	HSP_PARAMR7	OUT_C	U32	-	Output tensor channel number
10	HSP_PARAMR8	STRIDE_X	U32	-	Stride x
11	HSP_PARAMR9	STRIDE_Y	U32	-	Stride y
12	HSP_PARAMR10	OFF_IN	U32	-	Input offset
13	HSP_PARAMR11	OFF_OUT	U32	-	Output offset
14	HSP_PARAMR12	SAT_MIN	U32	-	Saturation min
15	HSP_PARAMR13	SAT_MAX	U32	-	Saturation max
16	HSP_PARAMR14	pBQ	U32	ABMP	Pointer on interleaved data bias, Q-mult, Q-shift (internal buffer)
17	HSP_PARAMR15	CFG	U32	-	Convolution mode
18	HSP_DCMDPTR0	pIn	I8*	ABMP	Pointer on input buffer
19	HSP_DCMDPTR1	pKer	I8*	ABMP	Pointer on kernel buffer
20	HSP_DCMDPTR2	pOut	I8*	ABMP	Pointer on output buffer

1. *Parameters must be written in the specified order.*
2. *Refer to Section 5: Processing functions description for acronym details*

IN_W	Input tensor width
IN_H	Input tensor height
IN_C	Input tensor channel number
K_W	Kernel tensor width
K_H	Kernel tensor height
OUT_W	Output tensor width
OUT_H	Output tensor height
OUT_C	Output tensor channel number
STRIDE_X	Stride x
STRIDE_Y	Stride y
OFF_IN	Input offset
OFF_OUT	Output offset

SAT_MIN	Saturation min
SAT_MAX	Saturation max
pBQ	Pointer on interleaved data bias, Q-mult and Q-shift
CFG	Convolution mode CGF[3:0]: Defines the number of steps, can be 0, 1, 2 or 3 steps CFG[17:4]: Number of lines of the input buffer. A line is IN_W x IN_C bytes. CFG[18:31]: Number of lines on output buffer. A line is OUT_W x OUT_C bytes.
pIn	Pointer on the input data buffer
pKer	Pointer on kernel data buffer
pOut	Pointer on output data buffer

5.12.2 CNN_CONVDW_I8

The CNN_CONVDW_I8 can only be executed into as direct command.
This function performs a depth-wise convolution on signed 8-bit data tensor.

5.12.2.1 CNN_CONVDW_I8 direct command format

Table 205. CNN_CONVDW_I8 direct command forma

Order	Registers	Acronyms	Type	Target	Description
1	HSP_DCMDIDR	CNN_CONVDW_I8	U32	-	See Section 8: Function ID list
2	HSP_PARAMR0	IN_W	U32	-	Input tensor width
3	HSP_PARAMR1	IN_H	U32	-	Input tensor height
4	HSP_PARAMR2	IN_C	U32	-	Input tensor channel number
5	HSP_PARAMR3	K_W	U32	-	Kernel tensor width
6	HSP_PARAMR4	K_H	U32	-	Kernel tensor height
7	HSP_PARAMR5	OUT_W	U32	-	Output tensor width
8	HSP_PARAMR6	OUT_H	U32	-	Output tensor height
9	HSP_PARAMR7	OUT_C	U32	-	Output tensor channel number
10	HSP_PARAMR8	STRIDE_XY	U32	-	Stride x, y
11	HSP_PARAMR9	OFF_IN	U32	-	Input offset
12	HSP_PARAMR10	OFF_OUT	U32	-	Output offset
13	HSP_PARAMR11	SAT_MIN	U32	-	Saturation min
14	HSP_PARAMR12	SAT_MAX	U32	-	Saturation max
15	HSP_PARAMR13	pBais	U32	ABMP	Pointer on bias data
16	HSP_PARAMR14	pQuant	U32	ABMP	Pointer on Interleaved Q-mult Q-shift
17	HSP_PARAMR15	CFG	U32	-	Convolution mode
18	HSP_DCMDPTR0	pIn	I8*	ABMP	Pointer on input buffer
19	HSP_DCMDPTR1	pKer	I8*	ABMP	Pointer on kernel buffer
20	HSP_DCMDPTR2	pOut	I8*	ABMP	Pointer on output buffer

1. *Parameters must be written in the specified order.*
2. *Refer to Section 5: Processing functions description for acronym details*

IN_W	Input tensor width
IN_H	Input tensor height

IN_C	Input tensor channel number
K_W	Kernel tensor width
K_H	Kernel tensor height
OUT_W	Output tensor width
OUT_H	Output tensor height
OUT_C	Output tensor channel number
STRIDE_XY	Stride x, y STRIDE_XY[15:0]: Contains stride x STRIDE_XY[31:16]: Contains stride y
OFF_IN	Input offset
OFF_OUT	Output offset
SAT_MIN	Saturation min
SAT_MAX	Saturation max
pBias	Pointer on data bias
pQuant	Pointer interleaved Q-mult and Q-shift
CFG	Convolution mode CGF[3:0]: Defines the number of steps, can be 0, 1, 2 or 3 steps CFG[17:4]: Number of lines of the input buffer. A line is IN_W x IN_C bytes. CFG[18:31]: Number of lines on output buffer. A line is OUT_W x OUT_C bytes.
pIn	Pointer on the input data buffer
pKer	Pointer on kernel data buffer
pOut	Pointer on output data buffer

5.12.3 CNN_CONVPW_I8

The CNN_CONVPW_I8 can only be executed into as direct command.
This function performs a point-wise convolution on signed 8-bit data tensor.

5.12.3.1 CNN_CONVPW_I8 direct command format

Table 206. CNN_CONVPW_I8 direct command format

Order	Registers	Acronyms	Type	Target	Description
1	HSP_DCMDIDR	CNN_CONVPW_I8	U32	-	See Section 8: Function ID list
2	HSP_PARAMR0	IN_W	U32	-	Input tensor width
3	HSP_PARAMR1	IN_H	U32	-	Input tensor height
4	HSP_PARAMR2	IN_C	U32	-	Input tensor channel number
5	HSP_PARAMR3	OUT_W	U32	-	Output tensor width
6	HSP_PARAMR4	OUT_H	U32	-	Output tensor height
7	HSP_PARAMR5	OUT_C	U32	-	Output tensor channel number
8	HSP_PARAMR6	K_C	U32	-	Kernel tensor channel number
9	HSP_PARAMR7	STRIDE_X	U32	-	Stride x
10	HSP_PARAMR8	STRIDE_Y	U32	-	Stride y
11	HSP_PARAMR9	OFF_IN	U32	-	Input offset
12	HSP_PARAMR10	OFF_OUT	U32	-	Output offset

Order	Registers	Acronyms	Type	Target	Description
13	HSP_PARAMR11	SAT_MIN	U32	-	Saturation min
14	HSP_PARAMR12	SAT_MAX	U32	-	Saturation max
15	HSP_PARAMR13	pBais	U32	ABMP	Pointer on bias data
16	HSP_PARAMR14	pQuant	U32	ABMP	Pointer on Interleaved Q-mult Q-shift
17	HSP_PARAMR15	CFG	U32	-	Convolution mode
18	HSP_DCMDPTR0	pIn	I8*	ABMP	Pointer on input buffer
19	HSP_DCMDPTR1	pKer	I8*	ABMP	Pointer on kernel buffer
20	HSP_DCMDPTR2	pOut	I8*	ABMP	Pointer on output buffer

1. *Parameters must be written in the specified order.*
2. *Refer to Section 5: Processing functions description for acronym details*

IN_W	Input tensor width
IN_H	Input tensor height
IN_C	Input tensor channel number
OUT_W	Output tensor width
OUT_H	Output tensor height
OUT_C	Output tensor channel number
K_C	Kernel tensor channel number
STRIDE_X	Stride x
STRIDE_Y	Stride y
OFF_IN	Input offset
OFF_OUT	Output offset
SAT_MIN	Saturation min
SAT_MAX	Saturation max
pBias	Pointer on data bias
pQuant	Pointer interleaved Q-mult and Q-shift
CFG	Convolution mode CGF[3:0]: Defines the number of steps, can be 0, 1, 2 or 3 steps CFG[17:4]: Number of lines of the input buffer. A line is IN_W x IN_C bytes. CFG[18:31]: Number of lines on output buffer. A line is OUT_W x OUT_C bytes.
pIn	Pointer on the input data buffer
pKer	Pointer on kernel data buffer
pOut	Pointer on output data buffer

5.12.4 CNN_FC_I8

The CNN_FC_I8 can only be executed into as direct command.

This function performs a fully connected operation on signed 8-bit data tensor.

5.12.4.1 *CNN_FC_I8 direct command format*

Table 207. CNN_FC_I8 direct command format

Order	Registers	Acronyms	Type	Target	Description
1	HSP_DCMDIDR	CNN_FC_I8	U32	-	See Section 8: Function ID list
-	HSP_PARAMR[1:0]	-	-	-	-
2	HSP_PARAMR2	IN_S	U32	-	Input vector dimension
-	HSP_PARAMR[6:3]	-	-	-	-
7	HSP_PARAMR7	OUT_S	U32	-	Output vector dimension
8	HSP_PARAMR8	QMULT	U32	-	Quantization multiplication value
9	HSP_PARAMR9	QSHIFT	U32	-	Quantization offset value
10	HSP_PARAMR10	OFF_IN	U32	-	Input offset
11	HSP_PARAMR11	OFF_OUT	U32	-	Output offset
12	HSP_PARAMR12	SAT_MIN	U32	-	Saturation min
13	HSP_PARAMR13	SAT_MAX	U32	-	Saturation max
14	HSP_PARAMR14	pBais	U32	ABMP	Pointer on bias data
15	HSP_PARAMR15	CFG	U32	-	Fully connected mode
12	HSP_DCMDPTR0	pIn	I8*	ABMP	Pointer on input buffer
13	HSP_DCMDPTR1	pKer	I8*	ABMP	Pointer on kernel buffer
14	HSP_DCMDPTR2	pOut	I8*	ABMP	Pointer on output buffer

1. *Parameters must be written in the specified order.*
2. *Refer to Section 5: Processing functions description for acronym details*

IN_S	Input vector dimension
OUT_S	Output vector dimension
QMULT	Quantization multiplication value
QSHIFT	Quantization offset value
OFF_IN	Input offset
OFF_OUT	Output offset
SAT_MIN	Saturation min
SAT_MAX	Saturation max
pBias	Pointer on data bias
CFG	Fully connected mode CGF[3:0] = 0, full buffer CGF[3:0] = 1, circular buffer
pIn	Pointer on the input data buffer
pKer	Pointer on kernel data buffer
pOut	Pointer on output data buffer

5.12.5 CNN_POOL_I8

The CNN_POOL_I8 can only be executed into as direct command.

This function performs a pooling operation on signed 8-bit data tensor.

5.12.5.1 *CNN_POOL_I8 direct command format*

Table 208. CNN_POOL_I8 direct command format

Order	Registers	Acronyms	Type	Target	Description
1	HSP_DCMDIDR	CNN_POOL_I8	U32	-	See Section 8: Function ID list
2	HSP_PARAMR0	IN_W	U32	-	Input tensor width
3	HSP_PARAMR1	IN_H	U32	-	Input tensor height
4	HSP_PARAMR2	IN_C	U32	-	Input tensor channel number
5	HSP_PARAMR3	K_W	U32	-	Kernel tensor width
6	HSP_PARAMR4	K_H	U32	-	Kernel tensor height
7	HSP_PARAMR5	OUT_W	U32	-	Output tensor width
9	HSP_PARAMR6	OUT_H	U32	-	Output tensor height
10	HSP_PARAMR7	OUT_C	U32	-	Output tensor channel number
11	HSP_PARAMR8	STRIDE_X	U32	-	Stride x
12	HSP_PARAMR9	STRIDE_Y	U32	-	Stride y
-	HSP_PARAMR[11:10]	-	-	-	-
13	HSP_PARAMR12	SAT_MIN	U32	-	Saturation min
14	HSP_PARAMR13	SAT_MAX	U32	-	Saturation max
15	HSP_PARAMR14	POOLTYPE	U32	ABMP	Pooling type
16	HSP_PARAMR15	CFG	U32	-	Pooling mode
17	HSP_DCMDPTR0	pIn	I8*	ABMP	Pointer on input buffer
-	HSP_DCMDPTR1	-	-	-	-
19	HSP_DCMDPTR2	pOut	I8*	ABMP	Pointer on output buffer

1. *Parameters must be written in the specified order.*
2. *Refer to Section 5: Processing functions description for acronym details*

IN_W	Input tensor width
IN_H	Input tensor height
IN_C	Input tensor channel number
K_W	Kernel tensor width
K_H	Kernel tensor height
OUT_W	Output tensor width
OUT_H	Output tensor height
OUT_C	Output tensor channel number
STRIDE_X	Stride x
STRIDE_Y	Stride y
SAT_MIN	Saturation min
SAT_MAX	Saturation max
POOLTYPE	Pooling type 0: Average pooling 1: Max pooling

CFG	Convolution mode CGF[3:0]: 0 = full buffer, 1 = circular buffer CFG[17:4]: Number of lines of the input buffer. A line is IN_W x IN_C bytes. CFG[18:31]: Number of lines on output buffer. A line is OUT_W x OUT_C bytes.
pIn	Pointer on the input data buffer
pOut	Pointer on output data buffer

5.13 Specific functions

The function ID of each function is provided on Section 8: Function ID list.

5.13.1 MULWIN_F32

The MULWIN_F32 can only be executed into a processing list.

This function performs an element related product between a vector containing data and a periodic window template.

The goal of this function is to save memory space when the window shape must be hosted into the BRAM. Before using MULWIN_F32 the application must build the window shape. The size of the window shape is WinLen/2 + 1.

The code below shows how to generate the most popular periodic cosine windows shapes.

```
/*=============================================================================
    - Generate the window reference table for PERIODIC windows.
    The reference table size is WinLen/2 + 1 in worst case
=============================================================================*/
void HAL_HSP_WinPer_Init( uint32_t WinLen, uint32_t WinType, float *WinShape )
{
uint32_t Size, nn;
float N;

    Size = WinLen/2 + 1;
    N = (float)WinLen;
    switch (WinType) {
        case HANNING:
            for (nn = 0 ; nn < Size ; nn++) {
                WinShape[nn] = 0.5 * (1.0 - cos( (2*PI*nn) / N  ));
            }
            break;
        case HAMMING:
            for (nn = 0 ; nn < Size ; nn++) {
                WinShape[nn] = 0.54  - 0.46 * cos( (2*PI*nn) / N );
            }
            break;
        case BLACKMAN:
            for (nn = 0; nn < Size ; nn++) {
                WinShape[nn] = 0.42-0.5 * cos((2*PI*nn)/N) + 0.08 * cos((4*PI*nn)/N);
            }
            break;
    }
}
```

5.13.1.1 *Command format to record MULWIN_F32*

Table 209. Command format to record MULWIN_F32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	MULWIN_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP	Pointer on input data buffer
HSP_PARAMR1	pWinShape	F32*	ABMP, POP	Pointer to the periodic window template
HSP_PARAMR2	pOut	U32*	ABMP, POP	Pointer on compare result
HSP_PARAMR3	WINLEN	U32	-	Window length

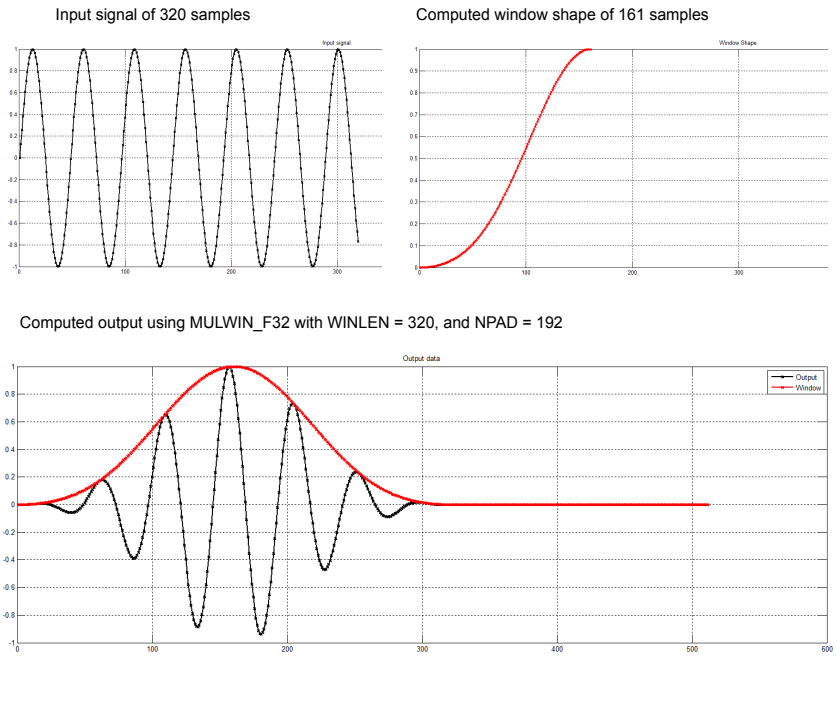
Registers	Acronyms	Type	Target	Description
HSP_PARAMR4	NPAD	U32	-	Number of zeros padding
HSP_PARAMR[14:5]	-	-	-	-
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffers type

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data A Must contain a real vector in float32. pInA can be the same as pOut.
pWinShape	Pointer on window shape This buffer contains WINLEN+1 float32 data representing the window shape to apply to the input buffer pointed by pInA. The window shape must be computed before running this function.
pOut	Pointer on ouput data Contains the element related multiplication of the window defined by pWinShape and the input vector pointed by pInA. A zero padding operation can be performed, typically to round the output buffer size to the next power-of-two. The size of the output buffer is WINLEN + NPAD.
WINLEN	Window length Window length can be any number between 32 and 4096. Typically it is equal to the input vector size.
NPAD	Number of padding data Number of zero-padding data, added after the windowing operation
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pWinShape target: If POP1 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pOut target: If POP2 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointerdepending on its value. Refer to Section 2.2.6: IOTYPE description for details

The figure below gives an example using a Blackman window shape on a sine input signal, with zero-padding at the end of the buffer.

Figure 36. **MULWIN_F32 example**



5.13.2 COMP_F32

The COMP_F32 can only be executed into a processing list.

This function performs a comparison of an input vector to a threshold. Several comparison options are possible:

- Check if at least one element of the vector is greater than the threshold
- Check if at least one element of the vector is greater or equal to the threshold
- Check if at least one element of the vector is lower than the threshold
- Check if at least one element of the vector is lower or equal to the threshold
- Check if at least one element of the vector is equal to the threshold

This function updates a variable providing the comparison result.

If there is no matching, the variable is set to 0xFFFF FFFF, otherwise this variable returns the position of the first element matching the comparison.

5.13.2.1 Command format to record COMP_F32

Table 210. **Command format to record COMP_F32**

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	COMP_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, POP	Pointer on input data buffer
HSP_PARAMR1	pThr	F32* F32	ABMP, POP, IBP, IMM	Threshold
HSP_PARAMR2	pRes	U32*	ABMP, POP	Pointer on compare result
HSP_PARAMR3	N	U32	-	Number of samples to compare
HSP_PARAMR4	CMP_TYPE	U32	-	Comparison options
HSP_PARAMR[14:5]	-	-	-	-
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffers type

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data A Must contain a real vector in float32.
pThr	Immediate value or pointer on threshold data This buffer contains either a pointer to the data containing the threshold value, or an immediate threshold value. IOTYPE shall give this information.
pRes	Pointer on comparison result data Content of the result data: 0xFFFF FFFF: No match found others: Position where the first match is found
N	Number of samples to compare Can be any integer between 2 and 4096. It is up to the application to insure that the number of elements contained into buffer pointed by pInA is bigger or equal to N.
CMP_TYPE	Comparison options Several comparison options are possible 1: Check if at least one element of the vector is greater than the threshold defined by pThr 2: Check if at least one element of the vector is greater or equal to the threshold defined by pThr 3: Check if at least one element of the vector is lower than the threshold defined by pThr 4: Check if at least one element of the vector is lower or equal to the threshold defined by pThr. 5: Check if at least one element of the vector is equal to the threshold defined by pThr.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointer. pThr target: If POP1 = 1, it is a pointer-on-pointer, if IMM = 1, it is an immediate value, otherwise it is a ABMP or IBP pointer. pRes target: If POP2 = 1, it is a pointer-on-pointer otherwise it is a ABMP pointerdepending on its value. Refer to Section 2.2.6: IOTYPE description for details

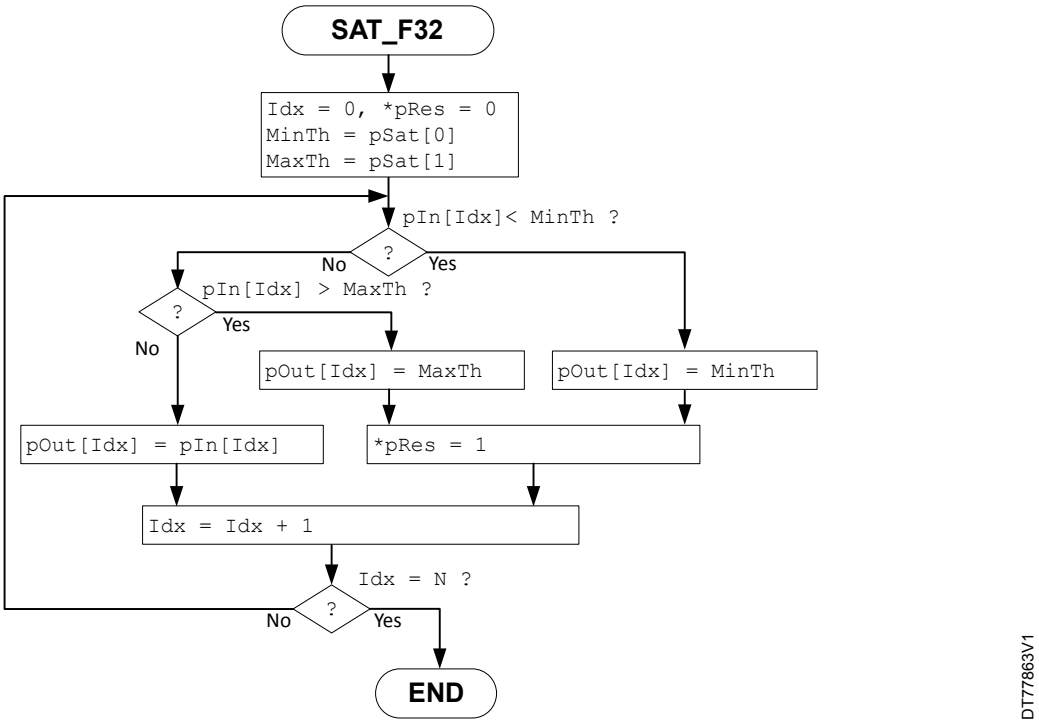
5.13.3 SAT_F32

The SAT_F32 can only be programmed into a processing list.

This function performs a saturation of a real vector or scalar using two programmable thresholds.

When a saturation is performed the flag FPUSATF of HSP_EVT_ISR register is set to 1, and a variable (pRes) is updated.

Figure 37. **SAT_F32 algorithm**



5.13.3.1 *Command format to record SAT_F32*

Table 211. **Command format to record SAT_F32**

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	SAT_F32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pInA	F32*	ABMP, IBP, POP	Pointer on input data buffer
HSP_PARAMR1	pThr	F32*	ABMP, POP	Pointer on threshold buffer (LoLim, HiLim)
HSP_PARAMR2	pOut	F32*	ABMP, IBP, POP	Pointer on output data buffer
HSP_PARAMR3	N	U32	-	Number of samples to compare
HSP_PARAMR4	pRes	U32	ABMP	Saturation status
HSP_PARAMR[14:5]	-	-	-	-
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

pInA	Pointer on input data A The input buffer must contain a real vector or a scalar in float32. The input and output buffer pointers can be the same.
pThr	Pointer on saturation threshold buffer This input buffer contains 2 real numbers in float32. The first element gives the minimum threshold, the second buffer the maximum threshold. See Table 211. Command format to record SAT_F32 for details.
pOut	Pointer on output data buffer The result is a real vector or a scalar in float32 having values between **MinTh** and **MaxTh**. The input and output buffer pointers can be the same.

N	Number of samples to process Can be any integer between 1 and 4096. It is up to the application to insure that the number of elements contained into buffer pointed by pInA is bigger or equal to N.
pRes	Pointer on data containing the saturation status After execution the SAT_F32 returns the following status: 0: if no saturation has been detected 1: if at least a saturation has been performed
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-Pointer otherwise it is a ABMP, IBP pointer depending on its value. pThr target: If POP1 = 1, it is a pointer-on-Pointer, otherwise it is a ABMP pointer. pOut target: If POP2 = 1, it is a pointer-on-Pointer otherwise it is a ABMP, IBP pointer depending on its value. Refer to Section 2.2.6: IOTYPE description for details

The format of this **pThr** is the following:

Table 212. pThr format

Offset addr.	Acronyms	Description	Data type
0	LoLim	Low limit threshold value. Must be lower than **HiLim**	F32
4	HiLim	High limit threshold value. Must be higher than **LoLim**.	F32

5.13.4 SEND_EVENT

The SEND_EVENT can only be programmed into a processing list.

This function generates an event to the event controller (EVTC), either using the HSP dedicated event generator (HDEG), or using the HSP shared event generator (HSEG).

5.13.4.1 Command format to record SEND_EVENT

Table 213. Command format to record SEND_EVENT

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	SEND_EVENT	U32	-	See Section 8: Function ID list
HSP_PARAMR0	EVTNUM	U32	-	Event number to generate
HSP_PARAMR1	ITF	U32	-	Event interface
HSP_PARAMR[15:2]	-	-	-	-

1. *Refer to Section 5: Processing functions description for acronym details*

EVTNUM	Event number to generate Integer value indicating which event number must be generated. For events generated via HDEG (ITF = 0), values from 0 to 36 are allowed. For events generated via HSEG (ITF = 1), values from 1 to 22 are allowed.
ITF	Event interface 0: Event is generated via the HDEG 1: Event is generated via the HSEG

The events generated with HDEG always have the highest priority (that is, 0), it means that at the end of the processing list executing the SEND_EVENT, the processing list defined by EVTNUM is immediately executed even if other events where pending.

If the application wants to generate events with a lower priority, HSEG must be used. The priority level is given by EVTNUM value. The event number is also the priority level.

5.13.5 SET_FLAG

The SET_FLAG can only be programmed into a processing list.

This function sets a PFCTF[x] flag into the HSP_PFCTEVT_ISR register in order to generate an interrupt to the CPU if the corresponding interrupt enable is set to 1.

5.13.5.1 Command format to record SET_FLAG

Table 214. Command format to record SET_FLAG

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	SET_FLAG	U32	-	See Section 8: Function ID list
HSP_PARAMR0	FLGPOS	U32	-	Flag position
HSP_PARAMR[15:1]	-	-	-	-

1. *Refer to Section 5: Processing functions description for acronym details*

FLGPOS	Flag position Integer value indicating which flag number must be generated. Must be a value between 0 and 31

5.13.6 CLR_FWERR

The CLR_FWERR function can only be used as a direct command.

This function reads the register HSP_FWERR writes its content into HSP_PARAMR0, and clear it.

5.13.6.1 CLR_FWERR direct command format

Table 215. CLR_FWERR direct command format

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	CLR_FWERR	U32	-	See Section 8: Function ID list
HSP_PARAMR0	FWERRN	U32	-	Firmware error value
HSP_PARAMR[15:1]	-	-	-	-

1. *Refer to Section 5: Processing functions description for acronym details*

FWERRN	Firmware error code HSP_PARAM0 contains the firmware error code. See Table 6. Command status for details.

5.13.7 SET_TRGO

The SET_TRGO can only be programmed into a processing list.

This function can generate a trigger pulse on one or several hsp_trgo[3:0] signals.

The application must set the TRGOEN bit to 1 before executing this processing function, otherwise, hsp_trgo[3:0] are not be affected. TRGOEN bit is located into HSP_ITFENR register.

5.13.7.1 *Command format to record SET_TRGO*

Table 216. Command format to record SET_TRGO

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	SET_TRGO	U32	-	See Section 8: Function ID list
HSP_PARAMR0	TRGVAL	U32	-	Trigger value
HSP_PARAMR[15:1]	-	-	-	-

1. *Refer to Section 5: Processing functions description for acronym details*

TRGVAL	Trigger value Integer value indicating which trigger output must be generated. Must be a value between 1 and 15. 0b0001: hsp_trgo[0] is activated 0b0010: hsp_trgo[1] is activated 0b0011: hsp_trgo[1:0] are activated 0b0100: hsp_trgo[2] is activated 0b0101: hsp_trgo[0] and **hsp_trg_out**[2] are activated ... 0b0111: hsp_trgo[2:0] are activated 0b1000: hsp_trgo[3] is activated ... 0b1111: hsp_trgo[3:0] are activated

5.13.8 SET_GPO

The SET_GPO can only be programmed into a processing list.

This function controls the level of signals hsp_gpo[7:0] signals.

The application must set the TRGOEN bit to 1 before executing this processing function, otherwise, hsp_gpo[7:0] is not affected. TRGOEN bit is located into HSP_ITFENR register.

5.13.8.1 *Command format to record SET_GPO*

Table 217. Command format to record SET_GPO

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	SET_GPO	U32	-	See Section 8: Function ID list
HSP_PARAMR0	-	-	-	-
HSP_PARAMR1	FIELDMASK	U32	-	Mask indicating which GPO to set
HSP_PARAMR2	FIELDVAL	U32	-	GPO values
HSP_PARAMR[15:3]	-	-	-	-

1. *Refer to Section 5: Processing functions description for acronym details*

FIELDMASK	Mask indicating which bits must be set Only bits 16 to 23 are considered. FIELDMASK[16] is the mask for hsp_gpo[0] FIELDMASK[17] is the mask for hsp_gpo[1] ...

	FIELDMASK[23] is the mask for hsp_gpo[7] When FIELDMASK[x] = 1 the corresponding GPO is forced to the corresponding bit value of FIELDVAL[x]. When FIELDMASK[x] = 0 the corresponding GPO is unchanged.
FIELDVAL	Field value Only bits with the corresponding bit in the FIELDMASK set to 1 are significant. Only bits 16 to 23 are considered. FIELDVAL[16] is the value for hsp_gpo[0] FIELDVAL[17] is the value for hsp_gpo[1] ... FIELDVAL[23] is the value for hsp_gpo[7]

5.13.9 SET_BITS

This processing function can only be programmed into a processing list.

The SET_BITS function modifies the data pointed by pInA by settings the bitfield specified by FIEKDMASK to the value given by FIELDVAL.

- Copy the content of pInA into a temporary buffer (tmp).
- Clear all the bits at position indicated by a 1 in the FIELDMASK:
  tmp = tmp & ~FIELDMASK.
- Set the bits filtered by FIELDMASK to the value specified the FIELDVAL
  tmp = tmp | (FIELDVAL & FIELDMASK).
- Copy tmp into pInA.

5.13.9.1 Command format to record SET_BITS

Table 218. Command format to record SET_BITS

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	SET_BITS	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pIn	U32*	ABMP, IBP	Pointer on input data
HSP_PARAMR1	FIELDMASK	U32	-	Mask indicating which bits to set
HSP_PARAMR2	FIELDVAL	U32	-	Field Value
HSP_PARAMR[15:3]	-	-	-	-

1. *Refer to Section 5: Processing functions description for acronym details*

pIn	Pointer on input data Pointer on the bit-field to be set.The input buffer contains a 32-bit value (integer or float32).
FIELDMASK	Mask indicating which bits must be set Each bit set to 1 indicates that the corresponding bit must be forced to the corresponding bit value of FIELDVAL. Each bit set to 0 indicates that the corresponding bit must not be changed.
FIELDVAL	Field value Only bits with the corresponding bit in the FIELDMASK set to 1 are significant.

5.13.10 CHK_BITS

This processing function can only be programmed into a processing list.

The CHK_BITS function checks the content of a bitfield against a reference value and updates a variable (pStat) according to the result.

Here are the details of CHK_BITS function:

- Copy the content of **pIn** into a temporary buffer (tmp).
- Clear all the bits at position indicated by a 0 in the FIELDMASK:
  tmp = tmp & FIELDMASK.
- Compare tmp to FIELDVAL
  - If FIELDVAL is equal to tmp., then pStat = 1
  - If FIELDVAL is different from tmp., then pStat = 0

5.13.10.1 *Command format to record CHK_BITS*

Table 219. Command format to record CHK_BITS

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	CHK_BITS	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pIn	U32*	ABMP, POP, IBP	Pointer on input data buffer
HSP_PARAMR1	pStat	U32*	ABMP	Pointer on input data buffer
HSP_PARAMR2	FIELDMASK	U32	-	Mask indicating which bits to check
HSP_PARAMR[8:3]	-	-	-	-
HSP_PARAMR9	FIELDVAL	U32	-	Field Value
HSP_PARAMR15	IOTYPE	U32	-	Input and output buffer types

1. *Refer to Section 5: Processing functions description for acronym details*

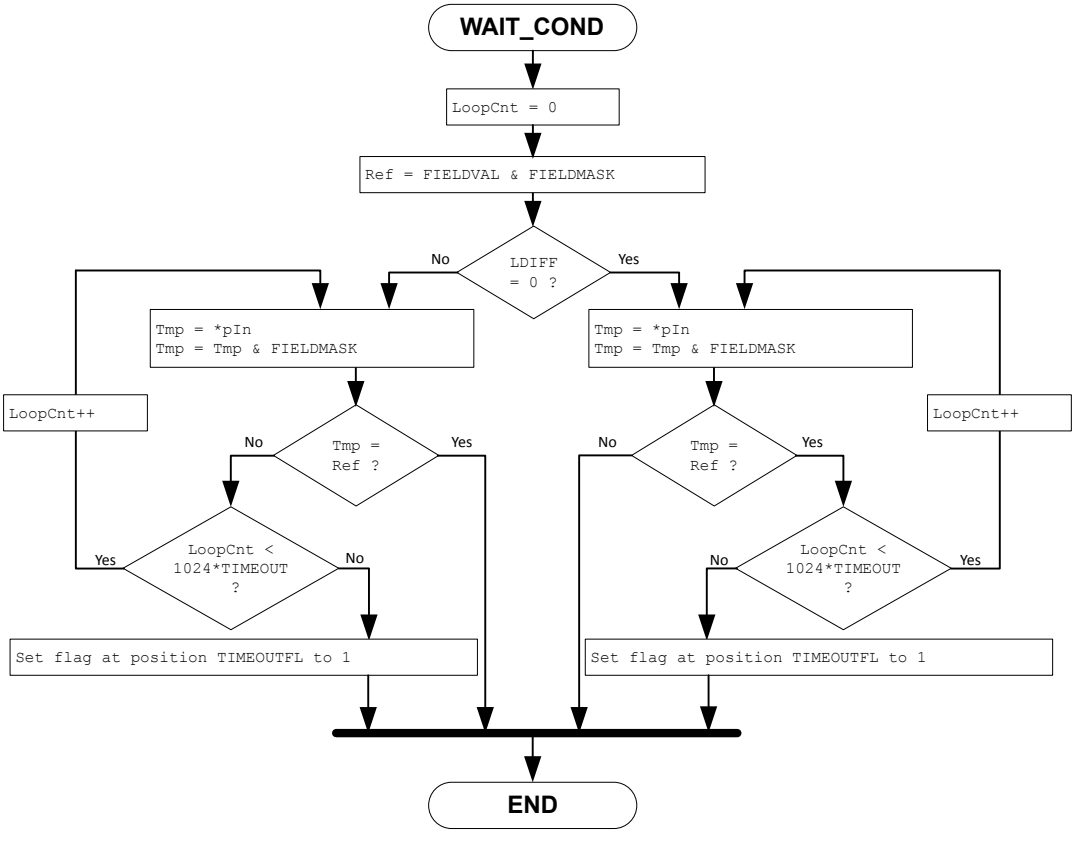
pIn	Pointer on input data buffer Pointer on the bitfield to be checked.The input buffer contains a 32-bit value (integer or float32).
FIELDMASK	Mask indicating which bits must be checked Each bit set to 1 indicates that the corresponding bit of pIn must be checked. Each bit set at 0 indicates that the corresponding bit must not be checked.
FIELDVAL	Field value This value is compared to content of pIn filtered by FIELDMASK.
IOTYPE	Input and output buffer types: Indicates the pointers type pInA target: If POP0 = 1, it is a pointer-on-pointer otherwise it is a ABMP, IBP pointer depending on its value. Refer to Section 2.2.6: IOTYPE description for details

5.13.11 WAIT_COND

This processing function can only be programmed into a processing list.

The WAIT_COND function checks the content of a bitfield against a reference value and waits until the condition is false or true. A timeout value forces the function to exit the loop.

Figure 38. **WAIT_COND function**



5.13.11.1 *Command format to record WAIT_COND*

Table 220. **Command format to record WAIT_COND**

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	WAIT_COND	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pIn	U32*	ABMP, IBP	Pointer on input data
HSP_PARAMR1	FIELDMASK	U32	-	Mask indicating which bits to check
HSP_PARAMR2	FIELDVAL	U32	-	Field Value
HSP_PARAMR3	TIMEOUTFL	U32	-	Field Value
HSP_PARAMR4	WCOND	U32	-	Wait condition
HSP_PARAMR5	TIMEOUT	U32	-	Timeout value
HSP_PARAMR[15:6]	-	-	-	-

1. *Refer to Section 5: Processing functions description for acronym details*

pIn	Pointer on input data Pointer on the bit-field to be set.The input buffer contains a 32-bit value (integer or float32).
FIELDMASK	Mask indicating which bits must be checked Each bit set to 1 indicates that the corresponding bit of pIn must be checked. Each bit set to 0 indicates that the corresponding bit must not be checked.

FIELDVAL	Field value This value is compared to content of pIn filtered by FIELDMASK
TIMEOUTFL	Timeout flag. Defines which flag of the HSP_KEREVT_ISR register must be set to 1, in case of timeout. Allowed values: integer from 0 to 31.
WCOND	Defines which condition to wait 0: means that the function loops as long as comparison result is equal 1: means that the function loops as long as comparison result is different
TIMEOUT	Timeout value Define the amount of time this function is waiting for condition

5.13.12 CRC32

The CRC32 can be either executed as a direct command (accelerator mode) or programmed into a processing list.

The CRC32 processing function offers the possibility to check the integrity of CROM and DROM. The ROM integrity is checked with a CRC-32 algorithm, using the polynomial 0xEDB88320.

The CRC32 function does not check the CRC, but computes the CRC on a given data block.

The pState structure contains the data to allow this function to process consecutive blocks without requesting parameter updates from CPU.

If TARGET is 1, then the function computes a new CRC value from CROM. The CRC is computed on BKSIZE words, starting at the CROM offset given by pState Offset, using pState.Crc as initial CRC value.

- If the end of the CROM is reached before processing BKSIZE words, the function stops the CRC computation, and:
  - Updates pOut with final CRC value
  - Sets pState.Offset to 0
  - Sets pState.Crc to 0 and
  - Sets flag FGEOR to 1
- When the function has processed BKSIZE words, the function stops the CRC computation, and:
  - Update pOut with the last CRC value
  - Update pState.Offset to select the next data to process
  - Update pState.Crc with the last CRC value
  - Sets flag FGEOB to 1

CRC32 behaves similarly for TARGET = 2.

5.13.12.1 Command format to record CRC32

Table 221. Command format to record CRC32

Registers	Acronyms	Type	Target	Description
HSP_C2HMSGDR	CRC32	U32	-	See Section 8: Function ID list
HSP_PARAMR0	pState	ST*	ABMP	Pointer on CRC32 data structure
HSP_PARAMR1	pOut	U32*	ABMP	Pointer on result
HSP_PARAMR2	BKSIZE	U32	-	Block size
HSP_PARAMR3	TARGET	U32	-	Memory target
HSP_PARAMR4	FGEOR	U32	-	End of ROM flag
HSP_PARAMR5	FGEOB	U32	-	End of block flag
HSP_PARAMR[15:6]	-	-	-	-

1. *Refer to Section 5: Processing functions description for acronym details*

pState	Pointer on CRC32 data structure. See Table 223. pState format for CRC32
pOut	Pointer on output data. Contains the CRC computed during last call.
BKSIZE	Block size Block size (in words) on which the CRC must be computed. There is no limitation on the block size.
TARGET	Defines the test target 1: CRC test on CROM 2: CRC test on DROM
FGEOR	End of ROM flag Position of the flag indicating when the CRC32 function reaches the end of the ROM.
FGEOB	End of block flag Position of the flag indicating when the CRC32 function reaches the end of the data block.

5.13.12.2 *CRC32 direct command format*

The next table shows the command format when CRC32 is executed as a direct command.

Table 222. CRC32 direct command format

Order	Registers	Acronyms	Type	Target	Description
1	HSP_DCMDIDR	CRC32	U32	-	See Section 8: Function ID list
2	HSP_PARAMR2	BKSIZE	U32	-	See description above
3	HSP_PARAMR3	TARGET	U32	-	
4	HSP_PARAMR4	FGEOR	U32	-	
5	HSP_PARAMR5	FGEOB	U32	-	
6	HSP_DCMDPTR0	pState	F32*	ABMP	
7	HSP_DCMDPTR1	pOut	U32*	ABMP	

1. *Parameters must be written in the specified order.*
2. *Refer to Section 5: Processing functions description for acronym details*

Table 223. pState format for CRC32

CPU Address offset	Field name	Type	Description
0	Crc	U32	Contains the CRC value of the ROM. It is updated by the CRC32 function at the end of each block. Can be initialized by the CPU before running CRC32.
4	Offset	F32*	Offset (in words) with respect to the ROM base address, where the CRC32 function starts the next CRC computation. It is updated by the CRC32 function at the end of each block. Can be initialized by the CPU before running CRC32.

6 Parameter updates

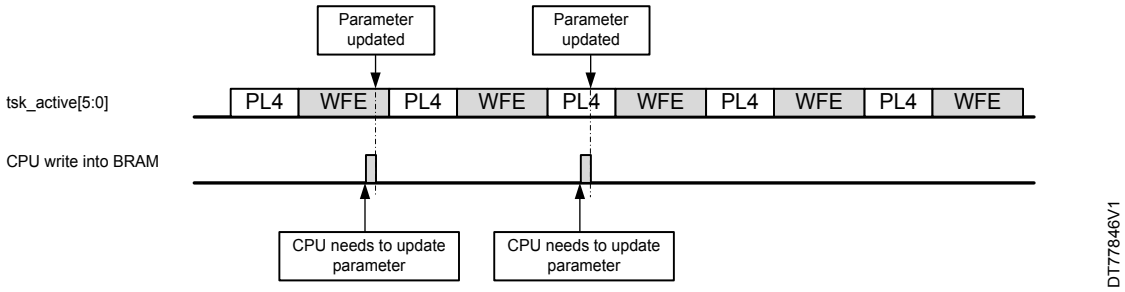
Most of the proposed functions have several parameters or data input and output.

Some of them may need to be updated from time to time during the application execution.

The dynamic parameters can be updated in various ways:

- The application can simply write new values at any time. The HSP may read at the same time the CPU updates the parameters. There is no data corruption, and in some applications, there is no issue if a new parameter is taken into account immediately or the next time the HSP processing is activated. As shown in the figure, a parameter update can occur at any moment.

Figure 39. Asynchronous parameter updates



In some cases this way of working is not acceptable.

A way to synchronize the CPU application with the HSP processing is to insert in the processing list, a command generating an interrupt (SET_FLAG).

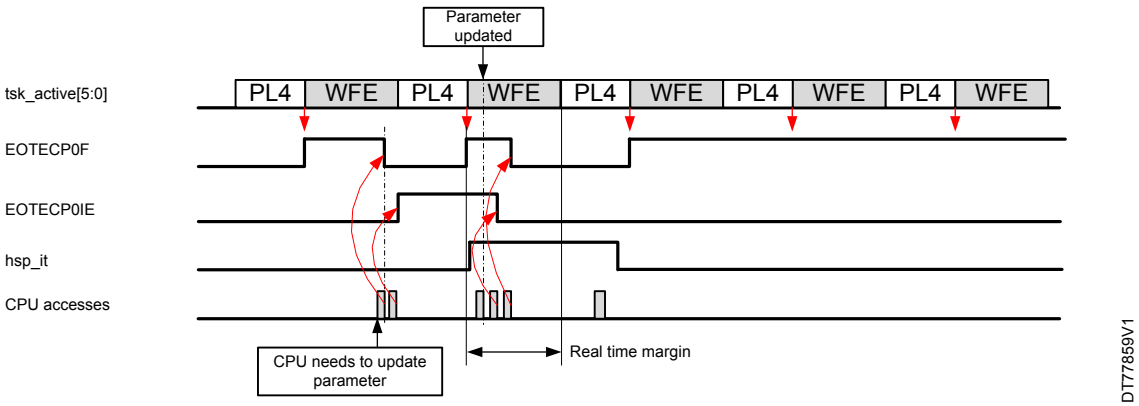
It is also possible to use a task comparator. Task comparators can generate an interrupt when the selected task starts to be executed, or when the execution is completed. With this method, the CPU knows when the processing allows an update, but it may also cause real-time constraint for some applications.

As shown in the next figure, the task comparator 0 is programmed to generate an interrupt at the end of the processing list PL4. The CPU may keep the interrupt enable control bit to 0 as long as there is no need to update parameters used by PL4. When an update must be done, the CPU does the following:

- Clears the end of task flag of comparator 0 (EOTECP0F)
- Enables the interrupt for the end-of-task comparator 0 (EOTECP0IE)
- At the end of the next completion of PL4, an interrupt occurs
- In the ISR, updates the parameters, disables the interrupt and optionally clears the flag

The sequence is repeated every-time parameters need to be updated.

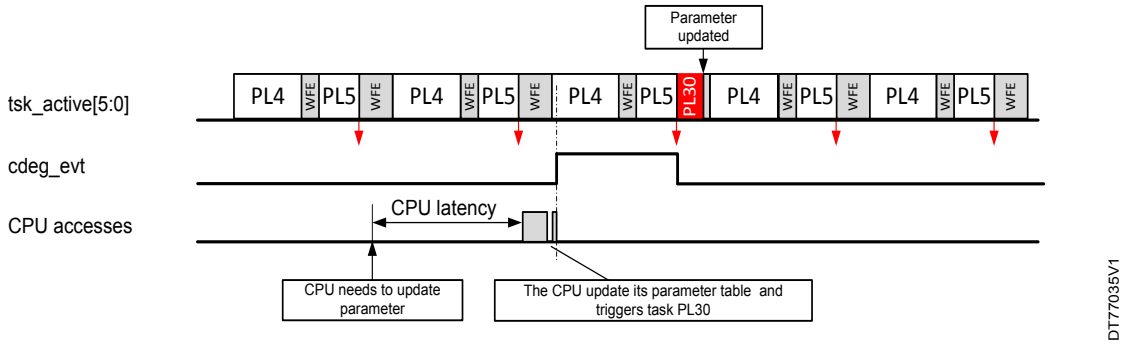
Figure 40. Asynchronous parameter updates



- If the application must ensure that several parameters are all updated before running a task, or if the CPU requires relaxing its real-time constraints, it is possible to create a processing list which copies the new parameters. In this implementation the parameters to be refreshed are duplicated.

To update the parameters in a coherent way, the CPU has to update the set of selected parameters and trigger the execution of a processing list, performing the transfer of those parameters at the location specified into the processing functions. Using the synchronization function, it is possible to force the execution of PL30, just after PL5 because there is enough room to insert PL30 process, without disturbing the process scheduling. This implementation ensures that the processing list using those parameters do not run until all parameters are updated.

Figure 41. Synchronous parameter updates with another processing list



7 Application examples

This section shows basic examples. To make the examples simpler, a pseudocode is used to show the recording of a processing list. The pseudocode uses the function mnemonics as show below:

```
PROCFG_START(ProcListNumber);    // To start the recording of ProcListNumber
PROCFG_END();                    // To stop the recording
```

A specific keyword is used to mention if a buffer is located into BRAM-AB, or in any other system memory. The line below defines a section into the BRAM-AB.

```
#define INBRAMAB __attribute__((section (".BRAMAB")))
```

The attribute INBRAMAB specifies that the buffers are located inside the BRAM-AB (that is, dual-port section of the BRAM). In the example below, InA is located into the BRAM-AB, and InB is in a system RAM.

```
float32_t InA[8]      INBRAMAB;
float32_t InB[200];
```

7.1 Start processing list recording

The example below gives the sequence to activate the supervisor and starts the recording of a processing list.

1. Enable the CDEG interface if not yet enabled by setting the bit CDEGEN of HSP_ITFENR register to 1
2. Send the PROCFG_START command as follows:
   a. Check that CDEG interface is free, and activate the supervisor by writing 37 into HSP_CDEGR register
   b. Check that C2HSEM = 0
   c. Write the processing list number into HSP_PARAMR0 register
   d. Write the PROCFG_START function ID into HSP_C2HMSGDR register
   e. Set C2HSEM to 1 in the HSP_C2HSEMR register
   f. Wait for H2CSEM = 1, read the command status, set H2CSEM to 0

7.2 Stop processing list recording

The example below gives the sequence to stop the recording of a processing list.

Write the PROCFG_END command to exit the supervisor as follows:

- Check that C2HSEM = 0
- Write the PROCFG_END function ID into HSP_C2HMSGDR register
- Set C2HSEM to 1 in the HSP_C2HSEMR register
- Wait for H2CSEM = 1, read the command status, set H2CSEM to 0

Note: It is a assumed that the supervisor is activated.

The start and the end of recording of a processing list can be written using the following pseudocode.

PROCFG_START(ProcListNumber) // To start the recording of ProcListNumber

PROCFG_END() // To stop the recording

7.3 Recording a simple processing

At any time, if the HSP is available, the application can add a new processing list by activating the supervisor:

1. Send the PROCFG_START command as explained in the Section 7.1: Start processing list recording.
2. Send a command to record the first processing function as follows:
   a. Check that C2HSEM = 0
   b. Write the processing function parameters into the HSP_PARAMRx registers
   c. Write the processing function ID into the HSP_C2HMSGDR register
   d. Set C2HSEM to 1 in the HSP_C2HSEMR register
   e. Wait for H2CSEM = 1, read the command status, set H2CSEM to 0
3. Record the wanted sequence by sending commands as explained in step 2.
4. Write the PROCFG_END command to exit the supervisor, as explained in the Section 7.2: Stop processing list recording.

For each command, the HSP firmware sends back a status via HSP_H2CMSGDR.

7.4 Programming a simple processing list using ABMP pointers

Shown below is a programming example based on the processing function VECT_DOTP_F32. It is assumed that pInA, pInB and pOut are pointers on BRAM-AB buffers.

The programming sequence below builds a processing list with a single processing function (VECT_DOTP_F32).

Activate the supervisor and send PROCFG_START command. See Section 7.1: Start processing list recording

Record VECT_DOTP_F32 processing function as follows:

- Check that C2HSEM = 0
- HSP_PARAMR0 = &InA[0] - SOC_BRAMAB_START + SPE_BRAMAB_START
- HSP_PARAMR1 = &InB[0] - SOC_BRAMAB_START + SPE_BRAMAB_START
- HSP_PARAMR2 = &Out - SOC_BRAMAB_START + SPE_BRAMAB_START
- HSP_PARAMR3 = Number of samples to process
- HSP_PARAMR15 = 0, IOTYPE = 0 because all buffers are ABMP
- Write into HSP_C2HMSGDR the function ID of VECT_DOTP_F32
- Write 1 into HSP_C2HSEMR in order to set C2HSEM to 1
- Wait for H2CSEM = 1, read the status into HSP_H2CMSGDR, set H2CSEM to 0

Send the PROCFG_END command. SeeSection 7.2: Stop processing list recording

Using the pseudocode, the sequence above can the translated as follows:

```
#define INBRAMAB __attribute__((section (".BRAMAB")))
#define NBDAT               4
#define PROCLIST_NB         18

// Variables located into BRAM-AB
float32_t InA[NBDAT] = {-0.543, 5.25, 3.12, -0.025}     INBRAMAB;
float32_t InB[NBDAT] = {1.1, 2.4, 6.0 3.14}             INBRAMAB;
float32_t Out                                           INBRAMAB;

PROCFG_START(PROCLIST_NB)                              // Start recording
    VECT_DOTP_F32(&InA[0], &InB[0], &Out, NBDAT, 0);
PROCFG_END()                                           // Stop recording
```

For simplification, assume that the pointer translation is done inside VECT_DOTP_F32.

7.5 Programming a simple processing list using IBP pointers

Shown below is a programming example based on the processing function __SCA_ADD_F32.

Assuming that:

- pInA is a pointer on BUFF0
- pInB is a pointer on BRAMAB
- pOut is a pointer on BUFF1

The programming sequence below builds a processing list with a single processing function (__SCA_ADD_F32).

Activate the supervisor and send PROCFG_START command. See Section 7.1: Start processing list recording.

Record __SCA_ADD_F32 processing function

- Check that C2HSEM = 0
- HSP_PARAMR0 = HSP_REG_SPE_BUFF0DR
- HSP_PARAMR1 = &pInB[0] - SOC_BRAMAB_START + SPE_BRAMAB_START
- HSP_PARAMR2 = HSP_REG_SPE_BUFF1DR
- HSP_PARAMR15 = IOTYPE = 0 because pointers are ABMP or IBP
- Write into HSP_C2HMSGDR the function ID of __SCA_ADD_F32
- Write 1 into HSP_C2HSEMR in order to set C2HSEM to 1
- Wait for H2CSEM = 1, read the status into HSP_H2CMSGDR, set H2CSEM to 0

Send the PROCFG_END command. See Section 7.2: Stop processing list recording.

Note: The application must program the BUFITF before running the processing list. HSP_REG_SPE_BUFF0DR = 0x002010F8 and HSP_REG_SPE_BUFF1DR = 0x002010FC.

```
#define INBRAMAB __attribute__((section (".BRAMAB")))
#define PROCLIST_NB        18

// Variables located into BRAM-AB
float32_t InB = 1.254              INBRAMAB;

PROCFG_START(PROCLIST_NB)                  // Start recording
    __SCA_ADD_F32(HSP_REG_SPE_BUFF0DR, &InB, HSP_REG_SPE_BUFF1DR, 0);
PROCFG_END()                               // Stop recording
```

For simplification, it is assumed that the pointer translation of &InB is done inside __SCA_ADD_F32.

7.6 Programming a processing list

This example shows a simple use-case of how to program the HSP in order to record a processing list.

The BRAM-AB contains a ping-pong buffer of 480 + 480 samples of 32 bits. This ping-pong buffer receives data from an external peripheral, and a system DMA is used to perform the data transfer from the peripheral into the BRAM-AB. Every time a ping or pong buffer is ready, the system DMA generates a transfer-complete event (TC).

This transfer-complete signal is connected to a hsp_trgi[x] input. In the example the block TRGIN0 is used, it is programmed to generate an event on the rising edge of the hsp_trgi[x] input receiving the system DMA transfer-complete. The EVTC block is programmed to trigger the processing list PL_PROC_DATA.

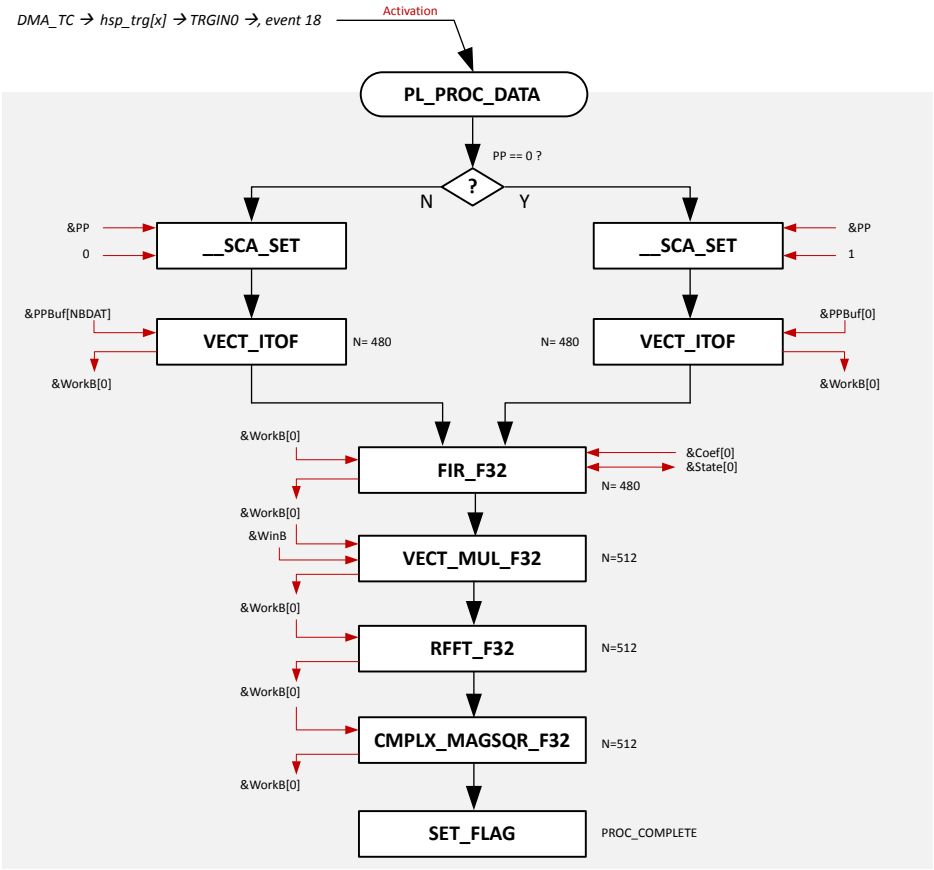
The processing list processes the data from ping or pong and informs the CPU when completed.

The processing consists of doing the following operations:

- Convert into float32 and copying the content of ping or pong buffer into a working buffer
- Filter the 480 samples
- Window and zero-padding to round the data buffer to 512 samples
- Perform a real FFT
- Compute the power spectrum

Figure 42 shows a graphical view of the processing list sequence.

Figure 42. **Processing list programming example**




```
#define INBRAMAB __attribute__((section (".BRAMAB")))
#define PL_PROC_DATA            18
#define NB_COEF                 32
#define FFTSIZE                 512
#define NBDAT                   480
#define PROC_COMPLETE           8

// Variables located into BRAM-AB
static int32_t PPBuf[NBDAT * 2]             INBRAMAB;     // Ping-pong buffer
static float32_t WorkB[FFTSIZE]             INBRAMAB;     // Working buffer
static uint32_t State[NB_COEF + 3]          INBRAMAB;     // Filter state
static float32_t PP                         INBRAMAB;     // Ping-pong variable

// Filter coefficients
static float32_t Coef[NB_TAPS] = { -0.432564811, -1.665584378,...}  INBRAMAB;
// Window shape
static float32_t WinB[FFTSIZE] = { 0.0, 0.002342, ...}              INBRAMAB;

// Initialize Filter state buffer
State[0] = &State[3] - SOC_BRAMAB_START + SPE_BRAMAB_START; // pointer conversion
State[1] = State[0];
State[2] = NB_COEF;

// Clear Filter state to 0
for (ii = 3 ;ii < (NB_COEF + 3) ; ii++) {
 State[ii] = 0;
}

PP=0;
PROCFG_START(PL_PROC_DATA);                     // Start recording
 __IF_ELSE(CMP_IFEQ, &PP, 0);                   // If PP == 0 --> Ping
  __SCA_SET(1.0, &PP,0);                        // PP = 1
```

```
  VECT_ITOF(&PPBuf[0], WorkB, NBDAT,0);            // Convert ping to float
 __IF_ELSE(CMP_ELSE);                              // else --> Pong
  __SCA_SET(0.0, &PP,0);                           // PP = 0
  VECT_ITOF(&PPBuf[NBDAT], WorkB, NBDAT,0);        // Convert ping to float
 __IF_ELSE(CMP_ENDIF);                             // endif
 FIR_F32(&WorkB[0], Coef, WorkB, State, NBDAT,0);
 VECT_MUL_F32(WorkB, WinB, WorkB, FFTSIZE,0);
 FFT_F32(WorkB,FFT,FFTSIZE,RFFT_TYPE1,BITREV,0);   // Performs RFFT
 CMPLX_MAGSQR_F32(WorkB, WorkB, FFTSIZE,0);        // Computes power spectrum
 SET_FLAG(PROC_COMPLETE);
PROCFG_END();                                      // Stop recording
```

It is assumed that the pointer translation is done inside the pseudocode functions.

7.6.1 Allocating the filter state and filter state buffer

The size of the FIR_F32 filter state structure (FltState) is 3 words of 32 bits (see Table 15. FltState structure format for FIR_F32, FIRLMS_F32).

The size of the FIR_F32 filter state buffer (FltStateBuf) is NB_COEF words of 32 bits (see Table 16. FltStateBuf for FIR_F32, FIRLMS_F32 and FIRDEC).

A single buffer of NB_COEF+3 words must be allocated in the BRAM-AB. It is also possible to allocate separately a structure or table of 3 elements, and a buffer of NB_COEF elements.

Assuming that a single buffer of NB_COEF+3 words named **State** is allocated, then **State** must be initialized as follows:

- State[0] = &StateTab[3] - SOC_BRAMAB_START + SPE_BRAMAB_START;
- State[1] = State[0]
- State[2] = NB_COEF
- State[NB_COEF+2:3] = 0

SOC_BRAM_START is the start address of the BRAM ‘seen’ by the CPU.

SPE_BRAM_START is the start address of the BRAM ‘seen’ by the SPE. SPE_BRAM_START is a constant fixed to 0x0010 0000.

7.7 __IF_ELSE example

In the example below, a signal is multiplied by a gain (Gain). If the RMS value of the signal is lower than 1.5, then the gain is increased. If the RMS value of the signal is higher than 1.5, then the gain is decreased.

A protection is added to limit the gain range between 20 and 0.1.

```
#define INBRAMAB __attribute__((section (".BRAMAB")))
#define PL_GAIN_CONTROL   18
#define NBDAT             256
#define U32_CTE           5
#define F32_VAR           3

// Variables located into BRAM-AB
static int32_t Sin[NBDAT]               INBRAMAB;     // Input data
float32_t Gain, RmsVal, StepUp, StepDw  INBRAMAB;
float32_t MaxGain, MinGain              INBRAMAB;

Gain = 1.0;
StepUp = 1.1;
StepDw = 0.95;
MaxGain = 20.0;
MinGain = 0.1;
PROCFG_START(PL_GAIN_CONTROL)          ;              // Start recording
 VECT_SCALE_F32(&Sin[0], &Gain, &Sin[0], NBDAT,0);    // Apply gain to Sin
 VECT_RMS_F32(&Sin[0], &RmsVal, NBDAT,0);             // RMS value of Sin
 __IF_ELSE(IFLT, &RmsVal, 1.5, U32_CTE);              // If (RmsVal < 1.5 rms)
  __SCA_MUL_F32(&Gain, &StepUp, &Gain,0);             // Gain = Gain * StepUp
  __IF_ELSE(IFGT, &Gain, &MaxGain, F32_VAR);          // If (Gain > MaxGain)
   __SCA_SET(&MaxGain, &Gain,0);                      // Gain = MaxGain
  __IF_ELSE(ENDIF);                                   // EndIf
 __IF_ELSE(ELSE);                                     // Else
  __SCA_MUL_F32(&Gain, &StepDw, &Gain,0);             // Gain = Gain * StepDw
  __IF_ELSE(IFLT, &Gain, MinGain, F32_VAR,0);         // If (Gain < MinGain)
```

```
   __SCA_SET(&MinGain, &Gain,0);                    // Gain = MinGain
  __IF_ELSE(ENDIF)    ;                            // EndIf
 __IF_ELSE(ENDIF);                                 // EndIf
PROCFG_END();                                      // Stop recording
```

7.8 __LOOP example

The example below shows a processing list using conditional functions.

A sequence of commands is repeated twice. During the first loop iteration, data InB[0], InB[2], InB[4],... are processed. During the second iteration, data InB[1], InB[3], InB[5],... are processed.

```
#define INBRAMAB __attribute__((section (".BRAMAB")))
#define PL_PROC_DATA        18
#define NB_COEF             32
#define FFTSIZE             512
#define NBDAT               480
#define PROC_COMPLETE       8

// Variables located into BRAM-AB
static int32_t InB[NBDAT * 2]               INBRAMAB;     // Input buffer
static float32_t WorkB[NBDAT * 2]           INBRAMAB;     // Working buffer
static uint32_t State[NB_COEF + 3]          INBRAMAB;     // Filter state

// Filter coefficients
static float32_t Coef[NB_TAPS] = { -0.432564811, -1.665584378,...}                      INB
RAMAB;

// Initialize filter state buffer
State[0] = &State[3] - SOC_BRAMAB_START + SPE_BRAMAB_START; // pointer conversion
State[1] = State[0];
State[2] = NB_COEF;

// Clear Filter state to 0
for (ii = 3 ;ii < (NB_COEF + 3) ; ii++) {
 State[ii] = 0;
}
PROCFG_START(PL_PROC_DATA);                    // Start recording
 __LOOP(LOOP, 2);                              // Repeat 2 times
  __IF_LOOP(IFEQ, 1, CTE_U32);                 // IF LOOP count == 1
   VECT_DEC(InB, 2, WorkB,0);                  // Copy even samples
   FIR_F32(WorkB, Coef, WorkB, State, 256,0);
  __IF_LOOP(ELSE);                             // ELSE
   VECT_DEC(&InB[1], 2, &WorkB[256],0);        // Copy odd samples
   FIR_F32(&WorkB[256], Coef, &WorkB[256], State, 256,0);
  __IF_LOOP(ENDIF)
 __LOOP(LOOP_END);
PROCFG_END();                                  // Stop recording
```

8 Function ID list

Table 224. List of control functions function IDs gives the function ID of the commands used for initialization and creation of processing lists.

Table 224. List of control functions function IDs

Mnemonics	Description	Function ID
Configuration functions		
FW_INIT	Firmware initialization command	0x84000000
PROCFG_START	Starts the creation of a processing list	0x81000000
PROCFG_END	Ends the creation of a processing list	0x82000000
PL_RESET	Processing list reset	0x83000000

The table below gives the function ID for each processing function. The function ID is given for recording functions into processing lists (Function ID for sequencer), and for executing directly the function in accelerator mode (Function ID for accelerator).

Table 225. List of processing function IDs

Mnemonics	Description	Function ID for Sequencer	Function ID for Accelerator
Filter functions			
BIQDF1_CASC_F32	Cascaded biquad direct form 1 filtering	0x00080608	0x00000A40
BIQDF2T_CASC_F32	Cascaded biquad direct form 2 transposed filtering	0x00080609	0x00000A48
FIR_F32	Optimal FIR filtering	0x00080600	0x00000A00
FIR_LESS8C_F32	Optimal FIR when less than 8 coefficients	-	0x00000A08
FIR_MOD4C_F32	Optimal FIR when coefficients are multiple of 4	-	0x00000A10
FIR_1S_F32	Optimal FIR when only 1 sample is processed	-	0x00000A18
FIRDEC_F32	FIR filtering with decimation	0x00000604	0x00000A20
FIRDEC_1COEF_F32	FIR filtering with decimation one coefficient	-	0x00000A28
FIRDEC_SPEC_F32	FIR filtering with decimation special	-	0x00000A30
FIRLMS_F32	FIR filtering with adaptive LMS algorithm	0x00000607	0x00000A38
IIRDF1_F32	Optimal IIR filtering using the direct form 1	0x0000060A	0x00000A50
IIR_LATTICE_F32	Performs an IIR filtering using the Lattice form	0x0000060D	0x00000A68
DC_IIRDF1_3P3Z_F32	3rd order IIR filter with saturation and gain adjust	0x0000060B	-
DC_IIRDF1_2P2Z_F32	2nd order IIR filter with saturation and gain adjust	0x0000060C	-
FLTBANK_F32	Filtering using internal filter banks	0x0007065C	0x00000CE0
CONV_F32	Performs the convolution between two real vectors	0x0000040E	0x00000A70
CORR_F32	Performs the correlation between two real vectors	0x0000040F	0x00000A78
RST_STATE	Reset the filter state	-	0x00000D48
GET_STATE	Gets the filter state for single stage filters	-	0x00000D68
GET_STATE_BIQ	Gets the filter state for biquads	-	0x00000D78
GET_STATE_IIRDF1	Gets the filter state for IIRDF1_F32	-	0x00000D70
SET_STATE	Sets the filter state for single stage filters	-	0x00000D50

Mnemonics	Description	Function ID for Sequencer	Function ID for Accelerator
SET_STATE_BIQ	Sets the filter state for biquads	-	0x00000D60
SET_STATE_IIRDF1	Sets the filter state for IIRDF1_F32	-	0x00000D58
Vector functions on real			
VECT_ABS_F32	element related absolute value	0x000C0116	0x00000AB0
VECT_ABSMAX_F32	Maximum absolute value of a vector	0x0008031C	0x00000AE0
VECT_ADD_F32	element related addition	0x00080311	0x00000A88
VECT_ATAN2_F32	element related arc tangent 2	0x0008032B	0x00000B58
VECT_AVG_F32	Average of a vector	0x000C0117	0x00000AB8
VECT_COPY	Vector copy	0x000C011E	0x00000AF0
VECT_COS_F32	element related cosine	0x000C0121	0x00000B08
VECT_DEC	Vector decimation	0x00020329	0x00000B48
VECT_DIV_F32	element related division	0x00080314	0x00000AA0
VECT_DOTP_F32	Dot product	0x00080320	0x00000B00
VECT_EXP_F32	element related exponential	0x000C012E	0x00000B70
VECT_EXP10_F32	element related 10^x	0x000C012F	0x00000B78
VECT_FTOI	Conversion of float32 to 32-bit signed integers	0x000C0124	0x00000B20
VECT_FTOU	Conversion of float32 to 32-bit unsigned integers	0x000C0127	0x00000B38
VECT_ITOF	Conversion of signed 32-bit integers to float32	0x000C0125	0x00000B28
VECT_I24TOF	Conversion of signed 24-bit integers to float32	0x000C0126	0x00000B30
VECT_Q31TOF	Conversion of signed Q31 to float32	0x000C0136	0x00000BB0
VECT_FTOQ31	Conversion of float32 to signed Q31	0x000C0135	0x00000BA8
VECT_FTOQ15	Convert float32 to Q15 format	0x000C0137	0x00000BB8
VECT_Q15TOF	Convert Q15 to float32 format	0x000C0138	0x00000BC0
VECT_LN_F32	element related natural logarithm	0x000C012C	0x00000B60
VECT_LOG10_F32	element related logarithm base 10	0x000C012D	0x00000B68
VECT_MAX_F32	Search the biggest element	0x0008031B	0x00000AD8
VECT_MIN_F32	Search the smallest element	0x0008031D	0x00000AE8
VECT_MUL_F32	element related product	0x00080313	0x00000A98
VECT_MULCOS_F32	element related product with a cosine	0x00080430	0x00000B80
VECT_MULSIN_F32	element related product with a sine	0x00080431	0x00000B88
VECT_OFFSET_F32	Add an offset to each element	0x0008031A	0x00000AD0
VECT_RMS_F32	Root mean square value of a vector	0x000C0118	0x00000AC0
VECT_SIN_F32	element related sinus	0x000C0122	0x00000B10
VECT_SINCOS_F32	element related sinus and cosine	0x000C0123	0x00000B18
VECT_SCALE_F32	Multiply each element by a value	0x00080319	0x00000AC8
VECT_SET	Sets a value to each element	0x0004011F	0x00000AF8
VECT_SQRT_F32	element related square-root	0x000C0115	0x00000AA8
VECT_SUB_F32	element related subtraction	0x00080312	0x00000A90
VECT_UTOF	Conversion of unsigned 32-bit integer to float32	0x000C0128	0x00000B40
VECT_ZINS	Performs the interpolation by inserting zeros	0x000A032A	0x00000B50

Mnemonics	Description	Function ID for Sequencer	Function ID for Accelerator
Functions operating on complex			
CMPLX_CONJ_F32	element related conjugate	0x000C0154	0x00000CA0
CMPLX_DOTP_F32	Complex dot product	0x00080355	0x00000CA8
CMPLX_MAG_F32	Complex Magnitude	0x000C0156	0x00000CB0
CMPLX_MAGSQR_F32	Complex Magnitude squared	0x000C0157	0x00000CB8
CMPLX_MUL_F32	Complex element related product	0x00080358	0x00000CC0
CMPLX_MULEXP_F32	element related product with complex exponential	0x0008045A	0x00000CD0
CMPLX_RMUL_F32	element related product of a complex with a real vector	0x00080359	0x00000CC8
Functions operating on real matrix			
MAT_ABS_F32	Compute the absolute value of each element	0x000C0116	0x00000AB0
MAT_ADD_F32	Performs the addition of two matrix	0x00080311	0x00000A88
MAT_INV_F32	Compute the inverse of a matrix	0x0000084D	0x00000C68
MAT_SUB_F32	Performs the subtraction of two matrix	0x00080312	0x00000A90
MAT_MUL_F32	Performs the product of two matrix	0x0000084B	0x00000C58
MAT_OFFSET_F32	Add an offset to each element of a matrix	0x0008031A	0x00000AD0
MAT_SCALE_F32	Multiply each element of a matrix by a scalar	0x00080319	0x00000AC8
MAT_TRANS_F32	Performs the transpose of a matrix	0x0000084C	0x00000C60
MAT_VECT2COL	Copy a vector to a matrix column	0x0008054F	-
MAT_COL2VECT	Copy a matrix column into a vector	0x0000054E	-
Scalar functions on real			
__SCA_ABS_F32	In-line absolute value	0x00040E8A	-
__SCA_ADD_F32	In-line addition	0x00000F7B	-
SCA_ATAN2_F32	Arc tangent 2	0x00000239	-
SCA_COS_F32	Cosine	0x0004003A	-
__SCA_DIV_F32	In-line division	0x00000F7E	-
SCA_EXP_F32	Exponential	0x0004003B	-
SCA_EXP10_F32	10^x	0x0004003C	-
SCA_FATAN2_F32	Low-precision Arc tangent 2	0x0000024A	-
SCA_FCOS_F32	Low-precision cosine	0x00040048	-
SCA_FSIN_F32	Low-precision sinus	0x00040049	-
__SCA_FTOI	In-line float32 to signed 32-bit integer conversion	0x00040E8C	-
SCA_FTOQ31	Float32 to Q31 conversion	0x0004003D	-
__SCA_FTOU	In-line float32 to unsigned 32-bit integer conversion	0x00040E8E	-
__SCA_I24TOF	In-line signed 24-bit integer to float32 conversion	0x00040E90	-
__SCA_ITOF	In-line signed 32-bit integer to float32 conversion	0x00040E8D	-
SCA_Q31TOF	Q31 to float32 conversion	0x0004003E	-
SCA_LN_F32	Natural logarithm	0x0004003F	-
SCA_LOG10_F32	Logarithm base 10	0x00040040	-
__SCA_MAC_F32	In-line multiply-accumulate	0x00000F80	-
__SCA_MUL_F32	In-line product	0x00000F7D	-

Mnemonics	Description	Function ID for Sequencer	Function ID for Accelerator
__SCA_NEG_F32	In-line negate	0x00010F7F	-
SCA_SIN_F32	Sinus	0x00040041	-
SCA_SINCOS_F32	Sinus and cosine	0x00040042	-
__SCA_SET	In-line sets a value	0x00040E89	-
__SCA_SQRT_F32	In-line square-root	0x00040E8B	-
__SCA_SUB_F32	In-line subtraction	0x00000F7C	-
SCA_TOVECT	Builds a vector with consecutive scalar values	0x00000363	-
__SCA_UTOF	In-line unsigned 32-bit integer into float32 conversion	0x00040E8F	-
Digital control functions			
DC_CMPCNT_F32	Comparison to a threshold with a counter	0x00000461	-
DC_CLARKE1_2P_F32	Clarke transform	0x00040043	-
DC_ICLARKE1_2P_F32	Inverse Clarke transform	0x00040045	-
DC_IPARK1_F32	Inverse Park transform	0x00000246	-
DC_PARK1_F32	Park transform	0x00000244	-
Transform functions			
FFT_F32	Real or Complex FFT transform	0x00070750	-
RFFT_F32	Real FFT transform	-	0x00000C88
CFFT_F32	Complex FFT transform	-	0x00000C80
DCT_F32	Discrete Cosine Transform type II	0x00070752	0x00000C90
IDCT_F32	Discrete Cosine Transform type III	0x00070753	0x00000C98
Conditional functions			
__IF_ELSE	In-line IF-THEN-ELSE function	0x00070A97	-
__IF_LOOP	In-line IF for LOOP counters	0x00070A96	-
__LOOP	In-line LOOP function	0x00070B98	-
__IF_COUNT	In-line IF for COUNT counters	0x00070A95	-
COUNT	General purpose counter	0x00000070	-
SET_COUNT	Set counters to a specific value	-	0x00000D80
__CLR_SATF	In-line function clearing the saturation status	0x00001094	-
__GET_SATF	In-line function reading the saturation status	0x00000E93	-
CNN functions			
CNN_CONV2D_I8	2D convolution	-	0x00000D98
CNN_CONVDW_I8	Depth-wise convolution	-	0x00000DA0
CNN_CONVPW_I8	Point-wise convolution	-	0x00000DA8
CNN_FC_I8	Fully connected layer	-	0x00000DB0
CNN_POOL_I8	Pooling layer	-	0x00000DB8
Specific functions			
MULWIN_F32	element related product with window shape	0x00000410	-
COMP_F32	Compare vector or scalar with branch	0x00000460	-
SAT_F32	Saturation using programmable thresholds	0x00000464	-
SEND_EVT	Triggers an event to the SPE	0x0000095D	-

Mnemonics	Description	Function ID for Sequencer	Function ID for Accelerator
SET_FLAG	Set flags to generate interrupt to CPU	0x0000095E	-
CLR_FWERR	Read and clear firmware error status	-	0x00000D90
SET_TRGO	Set TRGO signals	0x0000095F	-
SET_GPO	Set GPO signals	0x00070167	-
SET_BITS	Forces a specific field to a value	0x00020166	-
CHK_BITS	Checks values of a bit field	0x00071165	-
WAIT_COND	Loops until condition is met	0x00020C68	-
CRC32	Compute a CRC check	0x00000571	0x00000D88
Functions on integers			
__SCA_ADD_I32	In-line addition	0x00000F82	-
__SCA_INC_U32	In-line increment	0x00000E91	-
__SCA_SUB_I32	In-line subtraction	0x00000F83	-
__SCA_MUL_I32	In-line multiplication	0x00000F84	-
__SCA_NOT_U32	In-line bitwise NOT operation	0x00000E92	-
__SCA_SHIFT_I32	In-line left or right SHIFT operation	0x00000F85	-
__SCA_AND_U32	In-line bitwise AND operation	0x00000F86	-
__SCA_OR_U32	In-line bitwise OR operation	0x00000F87	-
__SCA_XOR_U32	In-line bitwise XOR operation	0x00000F88	-
SCA_MADD_U32	Addition with modulus	0x00020379	-

9 Glossary table

Table 226. **Glossary table**

Term	Definition
API	Application programming interface
CNN	Convolutional Neural Network
CRAM	Code RAM Contains the main HSP code, the plug-in if any, and the processing lists.
CROM	Code ROM Contains most of the processing function codes.
EVTC	HSP Event Controller The event controller is an hardware block included inside the HSP. It handles the generation of events.
DCMD	Direct Command The DCMD is a hardware block inside the HSP dedicated to the decoding of the direct commands. Direct commands are used mainly in accelerator mode.
DRAM	Data RAM Contains SPE private data, and processing functions parameters when included into processing lists.
DROM	Data ROM: Mainly contain twiddles for transform computation
HSP	Hardware signal processor
IBP	Interface buffer pointer This acronym is used to define a pointer data register of a HSP interface. HSP interfaces can be STREAM and ADCIN (if implemented).
PL	Stands for processing lists. Processing lists are series of processing functions recorded into the HSP RAM. They are executed when the event selected for the processing occurs.
SPE	Signal Processing Engine Processor embedded into the HSP
ABMP	BRAM-AB memory This acronym is used to define a pointer using data located into BRAM-AB.

Revision history

Table 227. **Document revision history**

Date	Version	Changes
02-Mar-2026	1	Initial release.

Contents

List of tables

List of figures