Radar Hardware Accelerator 2.0 User Guide – Part 2

Version 0.97: 25 November 2020

Note: This is a preliminary draft version of the HWA2.0 user guide. This document is subject to changes.

Radar Hardware Accelerator 2.0 – Part 2

The radar hardware accelerator user's guide is presented in two parts: Radar Hardware Accelerator 2.0 User's Guide – Part 1 and Radar Hardware Accelerator 2.0 User's Guide – Part 2. It describes the radar hardware accelerator architecture, features, operation of various blocks and their register descriptions. The purpose is to enable the user to understand the capabilities offered by the radar hardware accelerator and to program it appropriately to achieve the desired functionality.

The user's guide is split into two parts:

- The first part of the user guide (a separate document) provides an overview of the overall architecture and features available in the radar hardware accelerator. The main features, such as, windowing, FFT and log- magnitude are covered in the first part.
- The second part of the user guide (this document) covers additional features like DC Estimation and correction, Interference localization and mitigation, CFAR detection, Local Maxima Engine, FFT stitching, complex multiplication and other advanced usage possibilities. This part of the user's guide assumes that the user has already read and understood Part 1 (see Radar Hardware Accelerator 2.0 User's Guide – Part 1.

This document is organized as follows:

- Section 1 covers some additional features of the core computational unit related to pre-FFT processing.
- Section 2 covers details of CFAR-CA & CFAR-OS detection features.
- Section 3 covers other miscellaneous capabilities such as statistics computation.
- Section 4 covers the Local Maxima engine.
- Section 5 covers context switching.
- Section 6 includes a link to Compression Engine

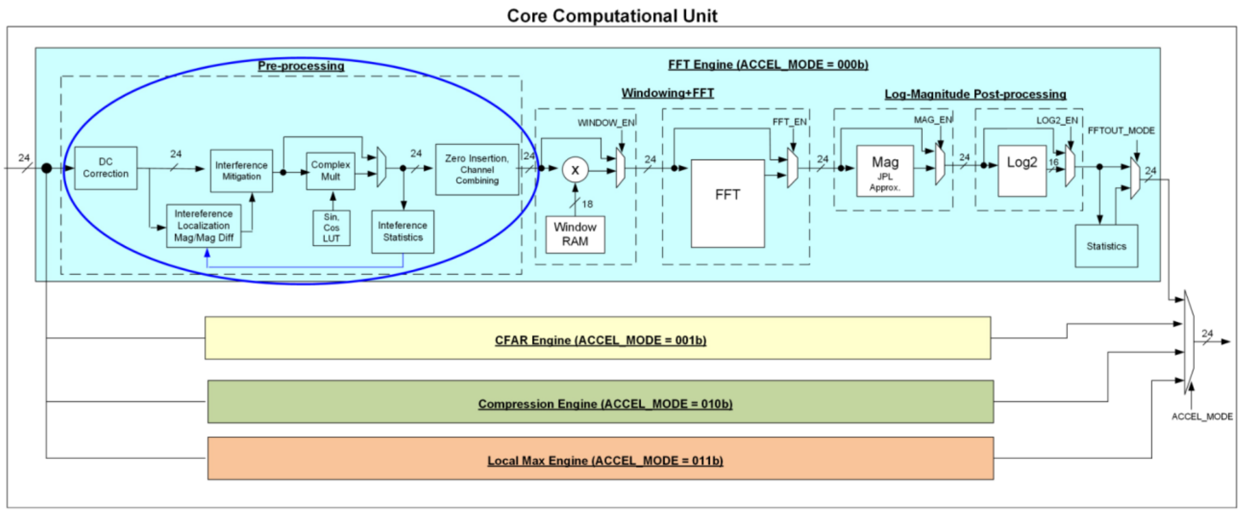
Contents

Figures

Tables

1 FFT Engine – Pre-Processing

As explained in Part 1 of the user guide, the FFT Engine comprises pre-processing, windowing, FFT and Log-magnitude subblocks and these are stitched together one after the other in series (refer Figure 1). This architecture allows multiple operations to be done in a streaming manner (for example, windowing and FFT can be done together), while at the same time, providing the user flexibility to choose one operation at a time. This section provides an overview of the pre-processing subblock inside the FFT engine of the core computational unit.



Core Computational Unit
FFT Engine (ACCEL_MODE = 000b)
Pre-processing
DC Correction
Interference Mitigation
Intereference Localization Mag/Mag Diff
Complex Mult
Sin, Cos LUT
Inteference Statistics
Zero Insertion, Channel Combining
Windowing+FFT
WINDOW_EN
Window RAM
18
FFT_EN
FFT
Log-Magnitude Post-processing
MAG_EN
Mag JPL Approx.
LOG2_EN
Log2
16
FFTOUT_MODE
Statistics
24
CFAR Engine (ACCEL_MODE = 001b)
Compression Engine (ACCEL_MODE = 010b)
Local Max Engine (ACCEL_MODE = 011b)
ACCEL_MODE

Figure 1: Core Computational Unit

The pre-processing subblock provides capability for DC estimation and correction, Interference localization and mitigation, complex multiplication, channel combining and zero-insertion.

1.1.1 DC Estimation

The DC estimation block estimates the time-domain average of the stream of samples along the A dimension. The stream can be one chirp, or set of chirps, i.e., frame. The DC is estimated on a per-iteration basis (i.e., along A dimension for each B iteration) for I & Q samples. Up-to 12 estimates corresponding to up to 12 iterations are available.

DC estimation is based on accumulation followed by a fine scaling and a programmable right shift. The fine scaling is configured as 1.8 value, via the 9-bit DCEST_SCALE register. The subsequent programmable right shift is configurable from 6 to 20 bits. Therefore, the DC estimation is well suited for cases where the number of samples per iteration is between 2^6 and 2^20. The fixed point details are captured in Figure 2. The internal accumulator reset supports several modes as shown in Table 1. For example, when DCEST_RESET_MODE = 2, the internal DC accumulators are reset at the beginning of the current parameter-set execution. Therefore, this mode estimates DC value for each set of SRCACNT samples along the A-dimension for up to 12 iterations along B-dimension within the current parameter-set. This mode is useful for per-chirp DC estimation. In this mode, the estimated DC values per iteration are latched at the end of current param-set and the accumulators are reset at the start. On the other hand, when DCEST_RESET_MODE = 3, the internal DC accumulators are reset only when the state machine executes the first loop of the parameter-set. As the state machine loops through various parameter-sets multiple times as programmed via NLOOPS register, the DC accumulators are not reset in between these loops. This mode is useful for per-frame DC estimation, where each loop corresponds to one chirp and the NLOOPS loops (chirps) correspond to a complete frame. The estimated DC values per iteration are latched at the end of last execution of the param-set.

The processor can read the DC estimates through the read-only registers – DCESTI_0VAL, .., DCESTI_11VAL & DCESTQ_0VAL, .., DCESTQ_11VAL. The DC estimates can also be used for DC subtraction described next.

Table 1: DC Estimation – Reset modes

DCEST_RESET_MODE	Comments
0	Hold the DC internal accumulators without updating (bypass DC estimation).
1	DC estimation enabled, but free-running without automatic reset (i.e., not reset at the start of this parameter-set). In this mode, the software can reset the DC accumulators by writing to DC_EST_RESET_SW register bit.
2	Reset the DC internal accumulators at the start of this parameter-set. This mode is applicable for per-chirp DC estimation.
3	Reset the DC internal accumulators at the start of this parameter-set only if the loop-counter is 0. This mode is applicable for per-frame DC estimation.

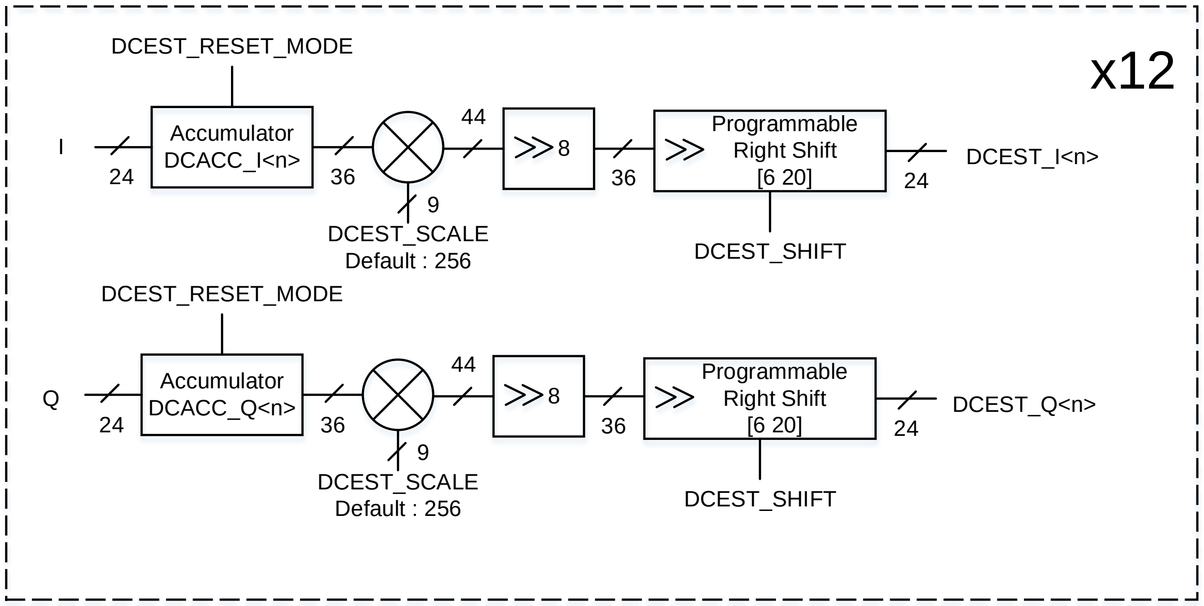


Figure 2: DC estimation

1.1.2 *DC Subtraction*

The DC subtraction feature is enabled if the register DCSUB_EN is set to 1.

DC subtraction (see Figure 3) can use the output from the built-in DC estimation accumulators, or a user-programmed value, based on the register bit, DCSUB_SELECT. If DCSUB_SELECT is 1, the DC estimation based on the internal accumulators is used. If DCSUB_SELECT is 0, the software override values are used (they are given by DC_SW_I_<n> and DC_SW_Q_<n> for the n^{th} iteration).

When using the built-in DC estimation accumulators, DC subtraction is performed on 12 individual streams corresponding to 12 RX on a per-iteration basis. Note that in a typical usage, for performing per-chirp DC estimation and DC subtraction, a two-pass approach is needed, where the first pass is configured for DC estimation via one parameter-set, and the second pass is configured for DC subtraction in the next parameter-set. Alternately, if a previous DC estimate (eg. From the previous chirp) is desired to be used for DC subtraction for the current chirp, then DC subtraction can be directly accomplished in one pass.

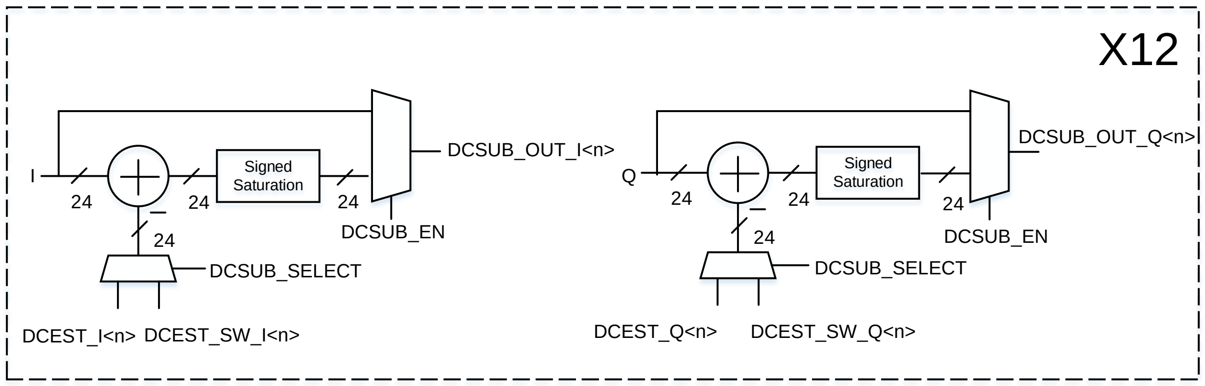


Figure 3: DC Subtraction

1.1.3 *Interference localization*

In an FMCW radar transceiver, interference from another radar typically manifests itself as a time-domain spike in a few samples. This spike corresponds to the time duration when the chirping frequency of both radars overlap with each other. Such a time-domain spike caused by interference can lead to degradation in the noise floor at the FFT output, causing degradation in detection performance.

In order to mitigate the impact of interference, the pre-processing block provides capability to perform interference localization to identify samples corrupted by interference, followed by interference mitigation to repair those samples.

The INTF_LOC_THRESH_EN register is provided as part of the parameter-set to control when the interference localization should be enabled. When enabled, the input samples are fed through a magnitude calculation (based on JPL approximation), which computes a 24-bit magnitude of the 24-bit input complex sample. For definition of this approximation, see Part 1 of this user guide. Similarly, magnitude of the backward difference between adjacent samples is also computed, which is another useful metric for interference (glitch) detection.

Any sample whose magnitude and/or magnitude of backward difference exceeds thresholds THRESH_MAG<n> and THRESH_MAGDIFF<n> is considered as affected by interference and is marked by a corresponding Interference indicator Bit (IIB). This is supported individually for up to 12 iterations. The register, INTF_LOC_THRESH_MODE determines the logic to set the IIB bit using the magnitude and/or magnitude of difference estimates. Based on this register, samples are marked with IIB if they exceed the THRESH_MAG<n>, or THRESH_MAGDIFF<n>, a logical AND of both, a logical OR of both as shown in Figure 4

This applies across all iterations

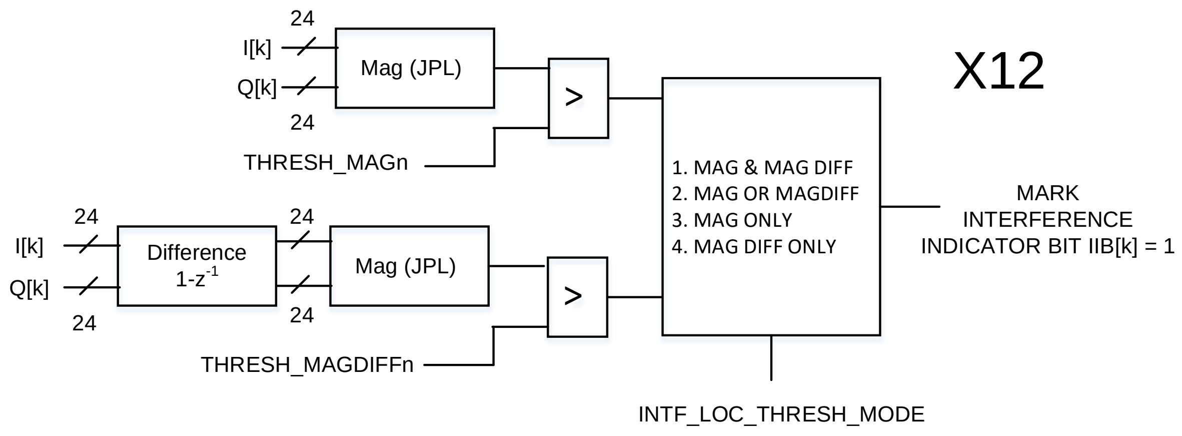


Figure 4: Interference Localization

The threshold values of THRESH_MAG<n> and THRESH_MAGDIFF<n> applied on a per-channel basis can derived from SW – INTF_LOC_THRESH_MAG<n>_SW, INTF_LOC_THRESH_MAGDIFF<n>_SW or from a built-in Interference statistics block – INTF_LOC_THRESH_MAG<n>, INTF_LOC_THRESH_MAGDIFF<n> as described in the next section. The user can also choose to sum the built-in interference statistics estimates across all channels to derive a common interference threshold across all iterations– INTF_STATS_SUM_MAG, INTF_STATS_SUM_MAGDIFF. The register, INTF_LOC_THRESH_SELECT is used to select these threshold options.

The number of samples marked with IIB across the iterations is recorded in the read-only registers, INTF_LOC_COUNT_ALL_CHIRP and INTF_LOC_COUNT_ALL_FRAME. This can be read after every chirp or after the completion of a frame (when the state machine completes all the programmed parameter-set loops and enters idle state).

1.1.4 *Interference statistics*

This block (Figure 5**)** provides the thresholds for interference localization. In order to obtain the interference statistics and derive the thresholds, the magnitude and magnitude of backward difference of the incoming samples are accumulated per iteration and up to 12 such independent accumulations are supported. These registers can be reset on a per-chirp or per-frame basis, and the behavior can be controlled using the register INTF_STATS_RESET_MODE. These reset modes are similar to DCEST_RESET_MODE previously explained (refer Table 1). The interference statistics accumulators can be reset by software via writing the INTF_STATS_RESET_SW register bit. This reset also clears the INTF_LOC_COUNTS.

The determination of interference threshold for interference localization is based on taking the above accumulator values and applying a programmable fine scaling, followed by a programmable right shift. The fine scaling is configured via 8-bit registers, INTF_STATS_MAG_SCALE and INTF_STATS_MAGDIFF_SCALE in 5.3 format. The fine scaling value is interpreted as an unsigned 8-bit number with 5 integer bits and 3 fractional bits giving a scale in range [0 to 31.875]. The default value of this register is 8, applying a scaling of 1.0. The programmable right-shift in the range of 6 to 12 is applied via the registers, INTF_STATS_MAG_SHIFT and INTF_STATS_MAGDIFF_SHIFT respectively. Note that if the sum mode of threshold selection is made, then the shift values have to include the extra division based on number of iterations being summed.

The resulting values INTF_LOC_THRESH_MAGn and INTF_LOC_THRESH_MAGDIFFn are used as thresholds in the interference localization block as described in the previous section.

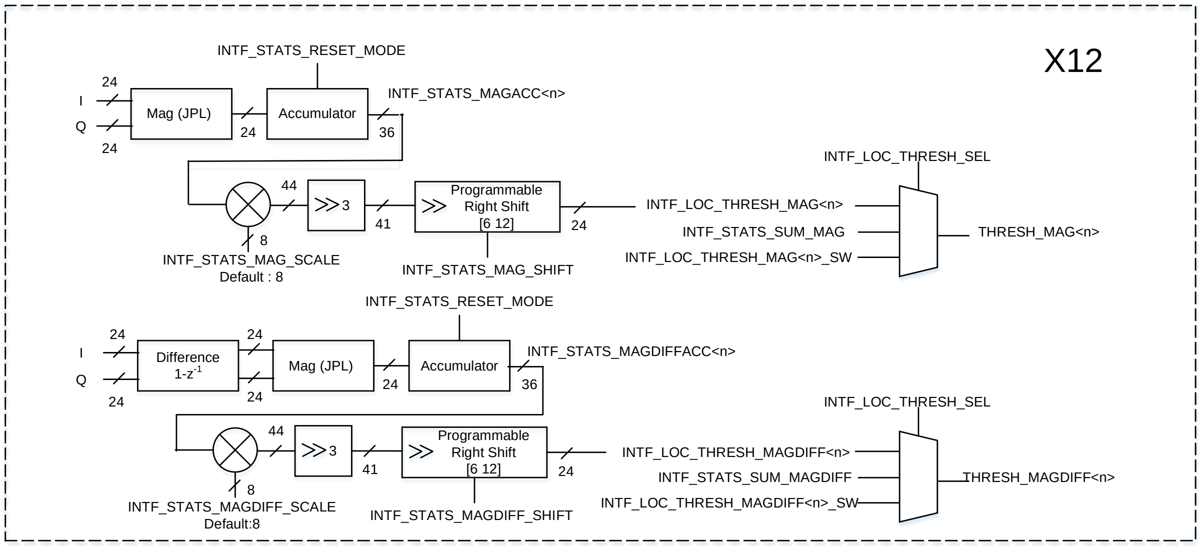


Figure 5: Interference Statistics

1.1.5 *Interference Mitigation*

Interference mitigation block (Figure 6) uses the results of Interference Localization block and mitigates the interference affecting the input samples which are marked as interference-corrupted through the Interference Indicator Bit. Interference mitigation is applicable for max. 12 iterations

The interference mitigation feature can be enabled by setting the INTF_MITG_EN bit in the parameter-set.

The first sub-module of the Interference Mitigation Block is a hysteresis module, which provides de-bouncing logic. For each incoming sample, its own IIB bit, as well as INTF_MITG_RIGHT_HYST_ORD number of right IIB bits and INTF_MITG_LEFT_HYST_ORD number of left IIB bits are considered in order to decide whether that particular sample is actually affected by interference. If the number of IIB bits in that interval is greater than or equal to INTF_MITG_CNTTHRESH, then that sample is assumed to be affected by interference. Thus, the hysteresis module outputs a filtered version of the IIB bit stream, which is then used for interference mitigation.

There are three different options (Figure 7) for interference mitigation and one of these can be selected using the INTF_MITG_PATH_SEL register. If INTF_MITG_PATH_SEL = 0, then the interference mitigation block simply zeros out samples that are marked with the IIB bit. This is a simple form of interference mitigation.

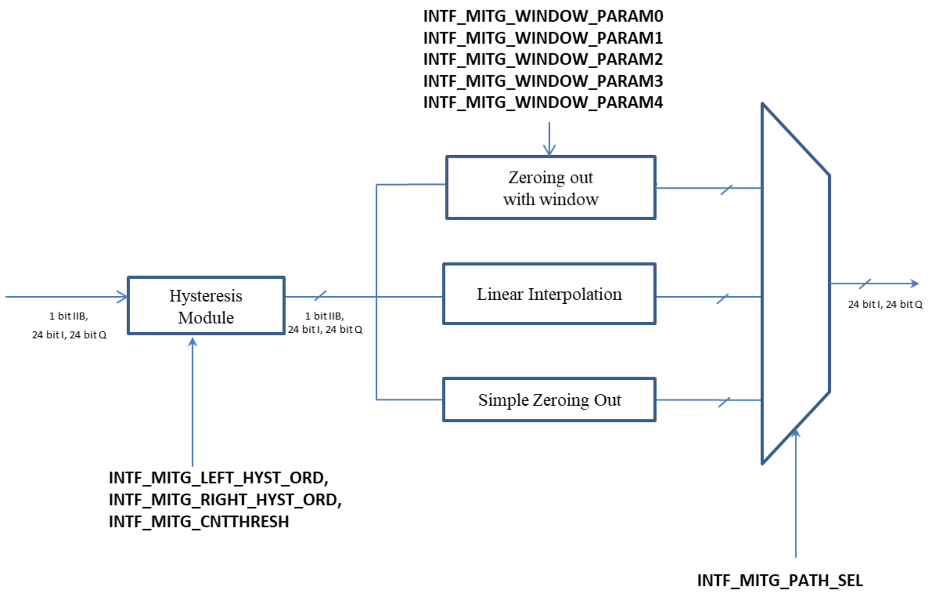


Figure 6: Interference Mitigation Block

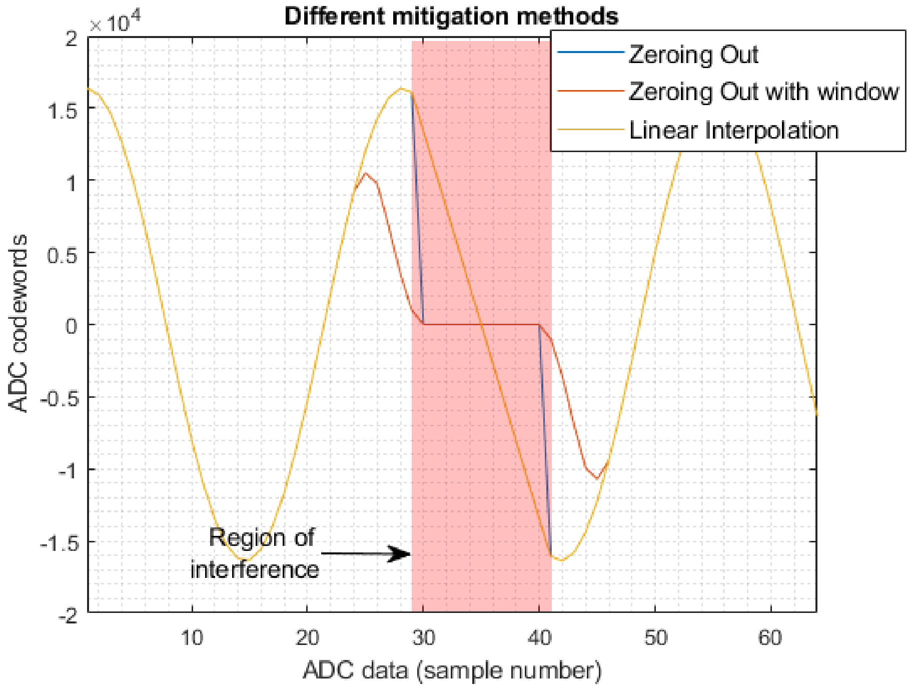


Figure 7: Interference Mitigation methods

If INTF_MITG_PATH_SEL = 1, then the interference mitigation block performs a windowed zero-out, where a smoothing window is applied to the edge samples of the interference-affected set of samples, in order to reduce the side-lobe increase that can happen with abrupt windowing. The windowing function that is applied is programmable via the INTF_MITG_WINDOW_PARAM <n> registers. The manner in which this smoothing window is applied is as follows.

Assume that the input sample array is: [x[0], x[1], x[2], x[3], x[4], x[5], x[6], x[7], x[8], x[9], x[10]]. Assume that the IIB array input is of the form [0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0]. (In other words, the 6th and 7th samples are corrupted by interference.) Then, the output of the windowed zero-out module would be:

[x[0]w[4], x[1]w[3], x[2]w[2], x[3]w[1], x[4]w[0], 0, 0, x[7]w[0], x[8]w[1], x[9]w[2], x[10]w[3]].

Where w[n] = INTF_MITG_WINDOW_PARAM<n>.

Note that the samples that are affected by interference are always zeroed out. The window is applied on the neighbours of these samples, so as to smoothen the transition to zero.

If INTF_MITG_PATH_SEL = 2, then the interference mitigation block performs a linear interpolation between the two good (i.e., non-interference-affected) samples at the start and end of the interference affected set of samples and this linear interpolation procedure is used to replace the interference affected samples themselves. The number of interference affected samples between these good samples can be any number. However, if it exceeds 32, then the 'last good sample' is pushed out in the place of affected samples until a new good sample arrives. Then linear interpolation will be performed across the 32 remaining affected samples.

1.1.6 Complex Multiplication

In addition to interference zero-out, the pre-processing block contains a complex multiplication sub-block. The purpose of this sub-block (Figure 8) is to enable several assorted capabilities that require complex multiplication of the input samples. The CMULT_MODE register is used to enable and configure the complex multiplication functionality. The complex multiplication sub-block can be disabled (bypassed) by the setting CMULT_MODE to 0b0000. Any other value of this register will enable the complex multiplication sub-block and configure it to perform specific operation as described in the next few paragraphs.

There are nine modes of the complex multiplier supported as follows. They are frequency shifter mode, frequency shifter with auto-increment mode (a slow DFT mode), FFT stitching mode, magnitude squared mode, scalar multiplication mode, vector multiplication modes-1 & 2, recursive windowing and LUT based frequency shifter modes.

In all the nine modes of the complex multiplier, one complex multiplication is performed every clock cycle.

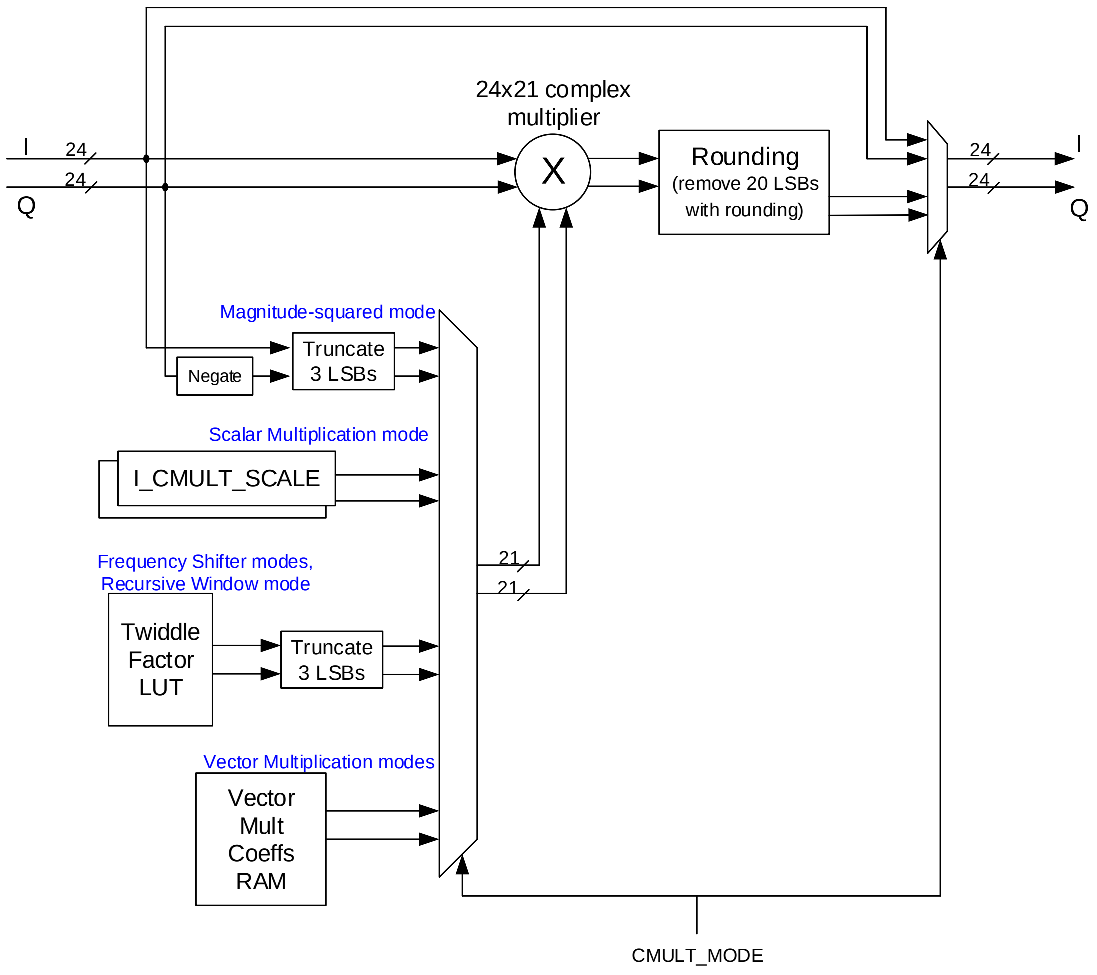
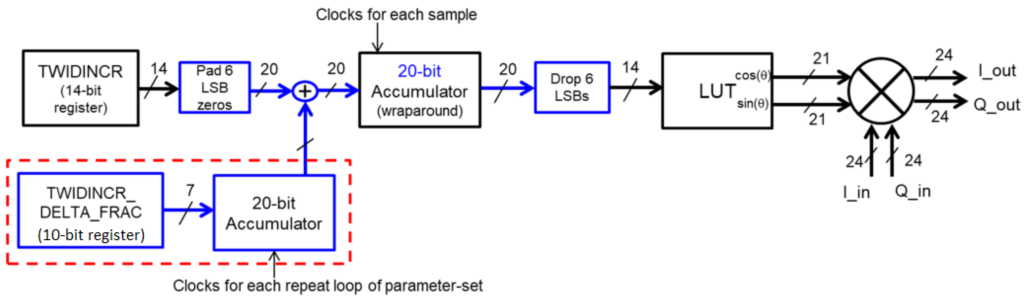


Figure 8: Complex Multiplication Capability in Pre-Processing Block

- **Frequency shifter mode**: If the register value is CMULT_MODE = 0001b, then the complex multiplier functions as a frequency shifter, which can be used to de-rotate the input samples by a certain frequency. This de-rotation is accomplished using cos, sin values from a twiddle factor look-up table (LUT). This LUT contains the (compressed) equivalent of the cos, sin values corresponding to the 16384 long sequence exp(–j*2*pi*(0:16383)/16384). TWIDINCR is used to specify the de-rotation frequency, by specifying how much the phase should change for each successive input sample (that register controls how much the LUT read index increments every sample) as shown in (Figure 9). The starting phase in this mode is always zero, since the 20-bit accumulator always starts at zero for each iteration.

  Note that although the figure shows another TWIDINCR_DELTA_FRAC register (portions shown in the red dotted box), that functionality is only applicable for CMULT_MODE = 1010b described later and it is not applicable in the present complex multiplier mode (0001b).



Clocks for each sample
TWIDINCR (14-bit register)
14
Pad 6 LSB zeros
20
20
20-bit Accumulator (wraparound)
20
Drop 6 LSBs
14
LUT
cos(θ)
sin(θ)
21
21
24
24
I_out
Q_out
24
24
I_in
Q_in
TWIDINCR_ DELTA_FRAC (10-bit register)
7
20-bit Accumulator
Clocks for each repeat loop of parameter-set

Figure 9: Frequency shifter mode

- **Frequency shifter with auto-increment mode** (a slow DFT mode): If the register value is CMULT_MODE = 0010b, then the complex multiplier functions in a mode which enables Discrete Fourier Transform (DFT) computation. In this case, the complex multiplier performs a function that is very similar to frequency shifter mode, except that, at the end of each iteration, the de-rotation frequency is automatically incremented for the next iteration. Note that DFT computation for a given set of input samples involves de-rotating the samples by one frequency at a time, and computing a sum of the de-rotated samples for each such frequency. To achieve DFT computation, the Input Formatter should be configured to send the same set of input samples to the complex multiplier for multiple iterations (as many as the number of DFT bins required) and the complex multiplier de-rotates the samples by one frequency at a time and auto-increments to the next frequency for the next iteration. Also, the statistics block (explained in a later section) is used to compute the sum of the de-rotated samples corresponding to each iteration, which then becomes the final DFT value.

  The DFT computation is 'slow' in the sense that in each clock cycle, only one complex multiplication is performed. For example, for a 512-point input sample set, it would take 512 clock cycles per DFT bin. However, since the DFT mode is typically only used for FFT peak interpolation (very few bins), it is acceptable. The starting frequency for the DFT computation is specified in the TWIDINCR register (similar to the frequency shifter mode). The increment value by which the frequency increments every iteration is obtained from FFTSIZE register – Note that the DFT mode cannot be used simultaneously with FFT enabled, hence the FFTSIZE register has been over-loaded for providing the increment value in this mode. The increment value is calculated as 2^(14 – FFTSIZE) and hence the DFT resolution is 16384/2^(14 – FFTSIZE) = 2^FFTSIZE. As an example, if FFTSIZE = 1011b, then the DFT resolution is 2048. This is equivalent to computing DFT points corresponding to 2K size FFT grid. The highest resolution for the DFT would be obtained when FFTSIZE = 1110b (max allowed value), in which case the DFT resolution is 16384 (corresponding to 16K size FFT grid). In effect, for the k^{th} iteration (with k starting from 0), the input samples x(n) for n = 0 to SRCACNT-1 are multiplied by the sequence, exp(–j*2*pi*(TWIDINCR+2^(14-FFTSIZE)*k)*(0:SRCACNT-1)/16384).

- **FFT Stitching mode**: If the register value is CMULT_MODE = 0011b, then the complex multiplier functions in FFT stitching mode. This mode is useful when large size FFTs (4K and 8K) are required. Since the FFT block natively supports only up to 2048 size, for 4096 and 8192 point FFT, an FFT Stitching procedure using two steps (two parameter-sets) can be used. As an example, when an 8K size FFT is needed, it is achieved in two steps as follows. In the first step, every 4th input sample is passed through a 2K size FFT (four 2K point FFTs are performed on decimated input samples). Then, in the next step, the resulting 4x2048 FFT outputs are sent through four-point "stitching" FFTs (2048 four- point FFTs), with an additional pre-multiplication by the complex multiplier block to achieve FFT stitching. This pre-multiplication uses the twiddle factor LUT in a specific pattern, for which additional configuration information is available in 2 LSBs of TWID_INCR register. Value '01' is for 4K (2x2048) size FFT stitching. If '10', then the twiddle factor pattern will correspond to what is required for 8K (4x2048) size FFT stitching. In the FFT stitching mode of operation, the 12 MSBs of TWID_INCR must be kept as 0.

The last section includes a more detailed explanation and configuration information for the FFT stitching example for 4K and 8K FFT, including the use of WINDOW_INTERP_FRACTION register for extending the window RAM using linear interpolation to more than 2048 coefficients. Note that Window Symmetry and Interpolation modes can't be used simultaneously.

- **Magnitude squared mode**: If the register value is CULT_MODE = 0100b, then the complex multiplier functions in magnitude squared mode. In this case, the complex multiplier takes a complex input and produces the magnitude squared as the output. This can be used together with the statistics block (explained in Section 3) to compute the mean squared sum of the input samples.

- **Scalar multiplication mode**: This mode is selected by setting CMULT_MODE = 0101b. It supports two options – multiplication by a complex scalar that remains constant across all iterations or by a complex scalar that changes per iteration.

  If the register bit CMULT_SCALE_EN = 0, then the complex multiplier functions in constant scalar multiplication mode. This feature is useful if the input samples need to be scaled by some constant factor. In this case, the complex multiplier will multiply each input sample with a 21-bit scalar complex number that is programmed in ICMULT_SCALE0 and QCMULT_SCALE0 registers (for I and Q value, each having 21 bits). The ICMULT_SCALE0 and QCMULT_SCALE0 registers are common registers and not part of parameter-set.

  To multiply the input samples for different iterations (channels) with different complex scalars, set CMULT_SCALE_EN = 1. In this mode, upto 12 different complex scalars are supported, viz. from ICMULT_SCALE0, QMULT_SCALE0 to ICMULT_SCALE11, QMULT_SCALE11 that are used for multiplication per iteration. TWID_INCR Register has no implication in this mode.

- **Vector multiplication mode 1**: If the register value is CMULT_MODE = 0110b, then the complex multiplier functions in vector multiplication mode 1. The purpose of this mode is to enable element-wise multiplication of two complex vectors, as well as dot-product capability (using statistics block to sum the element-wise multiplication output). The samples from the Input Formatter block constitute one of the two vectors, whereas the other vector is taken from a pre-loaded 'Vector Multiplication Coefficient RAM' inside the core computational unit. This Vector Multiplication Coefficient RAM can store 1024-complex samples and hence the vector multiplication can support a maximum of 1024 elements of multiplication. The Vector multiplication is not a highly parallelized operation, in the sense that only one complex multiplication is done per clock cycle.

  The operation of the vector multiplication mode 1 is as follows. The streaming set of samples from the Input Formatter block is element-wise multiplied with successive samples from the Vector Multiplication Coefficient RAM. The statistics block (described in a later section) can be used to compute the sum for every iteration, which enables a dot-product implementation if desired. At the end of every iteration, the addressing from the Vector Multiplication Coefficient RAM is reset, so that for the next iteration, the samples are picked up from the start index of the Vector Multiplication Coefficient RAM. It is possible to choose a non-zero start address for the Vector Multiplication Coefficient RAM, by programming the TWID_INCR register. The top 12 MSBs of the TWID_INCR register functions as a sample address offset for the RAM. The 2 LSBs must be kept zero. For example, if TWID_INCR = 20, then the vector multiplication happens starting from the 5th coefficient in the RAM (zero-based count).

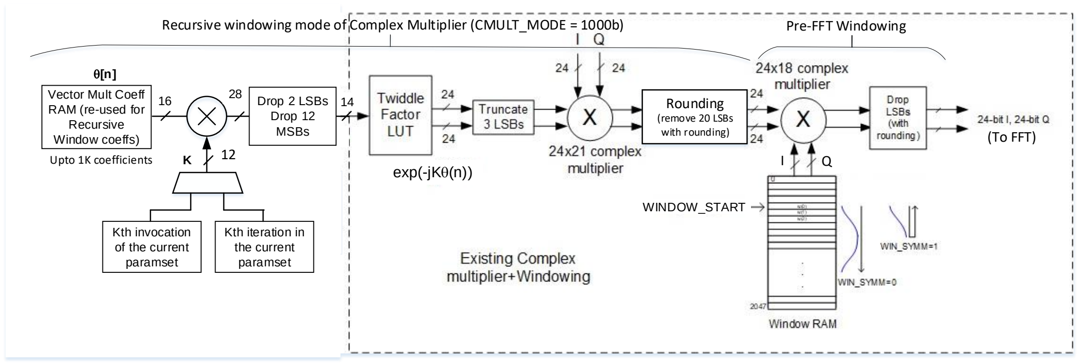
  When the size of vectors is small (<=12 coefficients), this mode can also be realized by setting the bit CMULT_SCALE_EN = 1 and programming ICMULT_SCALE0, QCMULT_SCALE0 to ICMULT_SCALE11, QCMULT_SCALE11 registers to store the successive coefficients. This mode can be used for example in RX channel gain and phase mismatch equalization. Because these ICMULT_SCALE<n> and QCMULT_SCALE<n> registers are independent of the Vector Multiplication Coefficients RAM, the user can use the RAM approach for some parameter-set which may need vector multiplication coefficients up to 1024, while another parameter-set of the processing chain uses the registers approach.

- **Vector multiplication mode 2**: If the register value is CMULT_MODE = 0111b, then the complex multiplier functions in vector multiplication mode 2, which is slightly different from the earlier Vector multiplication mode 1. The only difference in this case is that at the end of every iteration, the addressing of the Vector Multiplication Coefficient RAM is not reset, so that for the next iteration, the samples from the Vector Multiplication Coefficient RAM are picked up with an address that continues from where it left off at the end of the previous iteration. This mode can be used when a given set of input samples needs to be element-wise multiplied with multiple vectors. In this case, the input formatter block can be configured to repeat the same set of samples for multiple iterations, and the Vector Multiplication Coefficient RAM can be loaded with all the vectors, such that for successive iterations, the input samples are multiplied with successive vectors. As in previous mode, TWID_INCR functions as an address offset for the RAM.

- **Recursive windowing** mode (Figure 10) is supported using CMULT_MODE = 0b1000. A set of possibly random phase values $\theta(n)$ is stored in an internal RAM. This internal RAM is shared with the Vector Multiplication Coefficient RAM, in the sense that each of the $\theta(n)$ values takes up one coefficient location in the Vector Multiplication Coefficients RAM, and therefore use of Recursive Windowing mode comes at the expense of reduced number of coefficients storage for the other vector multiplication modes. The operation of Recursive Windowing mode is explained as follows.

Assuming that the Window RAM (which is separate) contains the window coefficients $w_0(n)$, the final window function is computed as: $w_K(n) = w_0(n) * \exp(-j*K*\theta(n))$. This computed window function $w_K(n)$ is used for the windowing prior to FFT operation. Here K is either the iteration count (zero-based count corresponding to B-dimension iterations) within the parameter-set, or the current execution count of the parameter-set where CMULT_MODE = 0b1000. This selection can be made using the register bit, RECWIN_MODE_SEL. A value of RECWIN_MODE_SEL = 0 indicates K is based on iteration count, and RECWIN_MODE_SEL = 1 indicates it is based on the said execution count. When REC_WIN_MODE_SEL = 0, the value of K resets to zero at the end of all iterations of the current parameter-set. On the other hand, when REC_WIN_MODE_SEL = 1, the value of K persists (and increments) across multiple loops of the parameter-set and it can be reset via software by writing to the register bit RECWIN_RESET_SW.

Figure 10: Recursive windowing using the complex multiplier



Recursive windowing mode of Complex Multiplier (CMULT_MODE = 1000b)
Pre-FFT Windowing
θ[n]
Vector Mult Coeff RAM (re-used for Recursive Window coeffs)
Upto 1K coefficients
16
28
Drop 2 LSBs Drop 12 MSBs
14
K
12
Kth invocation of the current paramset
Kth iteration in the current paramset
Twiddle Factor LUT
exp(-jKθ(n))
24
24
Truncate 3 LSBs
I Q
24 24
X
24x21 complex multiplier
Rounding (remove 20 LSBs with rounding)
24x18 complex multiplier
24
24
X
Drop LSBs (with rounding)
24-bit I, 24-bit Q (To FFT)
I Q
WINDOW_START
WIN_SYMM=1
WIN_SYMM=0
2047
Window RAM
Existing Complex multiplier+Windowing

- **LUT based frequency and phase de-rotation mode:** This mode is enabled using CMULT_MODE = 1001b. This mode is similar to Frequency shifter mode (previously explained, where CMULT_MODE = 0001b), except that it supports both frequency de-rotation and phase de-rotation based on a programmable RAM, TWID_ANGLE_RAM. Up to store 64 different frequency de-rotation and starting phase values can be programmed. Each entry in the RAM is 32 bits wide. The 16 MSBs contain starting phase word – TWID_PHA and in remaining 16 LSB, the top-most two bits are don't cares, the next 14 LSBs contain the de-rotation frequency word. The 6 LSBs of TWIDINCR register are used as a start address index to the TWID_ANGLE_RAM.

  There are two sub-modes of this feature, selectable by bit 13 of TWIDINCR register. If this bit set to 1 (Auto-increment mode), then the RAM index is incremented automatically after each iteration (i.e., B dimension), else (Non-increment mode), the RAM index is constant for all iterations, based on the 6 LSBs of TWIDINCR register. Further, bit 12 of TWIDINCR controls whether the RAM index saturates at 63 (behavior when the bit is set to 1) or wraps around (behavior when the bit is 0) in the Auto-increment mode.

- **Frequency shifter mode with fine frequency increment**: This mode is enabled using CMULT_MODE = 1010b. This mode is an extension of the Frequency shifter mode (previously explained, where CMULT_MODE = 0001b). In this mode, in addition to the previously explained frequency shifter functionality, there is another signed 10-bit offset TWIDINCR_DELTA_FRAC that can be added to the de-rotation frequency, such that the de-rotation frequency changes incrementally after every "execution count". The TWIDINCR_DELTA_FRAC value is automatically accumulated to the de-rotation frequency at the end of the current execution of the current parameter-set with CMULT_MODE = 1010b. The fixed-point design of this datapath is illustrated in Figure 9). In effect, the input samples x(n) for n = 0 to SRCACNT-1 are multiplied by the sequence: exp(–j * 2*pi * ((TWIDINCR + 2^(-6) * TWIDINCR_DELTA_FRAC * execution_count) * (0:SRCACNT-1) / 16384)). The execution_count here refers to the current execution count of the parameter-set with CMULT_MODE = 1010b. The TWIDINCR_DELTA_FRAC accumulator is reset only via software by writing to the TWID_INCR_DELTA_FRAC_RESET_SW register bit. Note that the TWIDINCR_DELTA_FRAC register is applicable only in this mode of the complex multiplier and is ignored in all other modes.

The memory layouts for different modes are illustrated in Figure 11

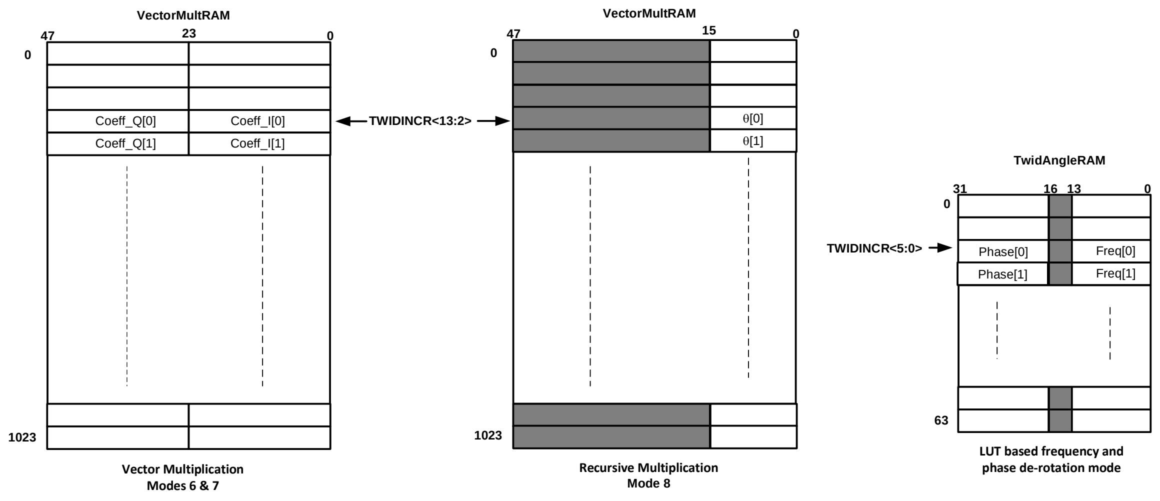


Figure 11: Memory layout for different multiplication modes

1.1.7 BPM Removal

Although not explicitly shown in Figure 1, it is possible to multiply the input samples going from the input formatter into the core computational unit with a +1/-1 programmable binary sequence (of length up to 256). This feature is enabled by setting the register bit BPM_EN in the parameter-set.

This feature may be useful when Binary Phase Modulation (BPM) is used during transmission of chirps. The BPM pattern is generally a pseudo-random sequence (chipping sequence) of 1's and -1's, which have already been applied to the radar transmit signal. Therefore, the radar signal processing of the resultant analog-to-digital converter (ADC) samples prior to FFT needs to undo the modulation. For instance, if each chirp is transmitted with a +1 or -1 polarity, then it is necessary to undo this sequence prior to the second dimension FFT processing across chirps. The BPM removal feature can be used to achieve this.

Note that an alternate way to achieve this is to pre-multiply the window coefficients, which are signed numbers, in the window RAM, so that the process of windowing prior to FFT takes care of undoing the BPM sequence.

When BPM removal is enabled, each input sample is multiplied by a +1 or -1, based on the bit sequence present in the eight 32-bit registers, BPMPATTERN0, BPMPATTERN1, … BPMPATTERN7. A bit value of 0 (1) multiplies the input sample with 1 (–1). The register BPMRATE is used to control for how many consecutive samples the same BPM bit is applied. For example, if BPMRATE = 4, then the same BPM bit is applied for 4 consecutive samples. Similarly, if BPMRATE = 1, then the BPM bit is changed for every sample.

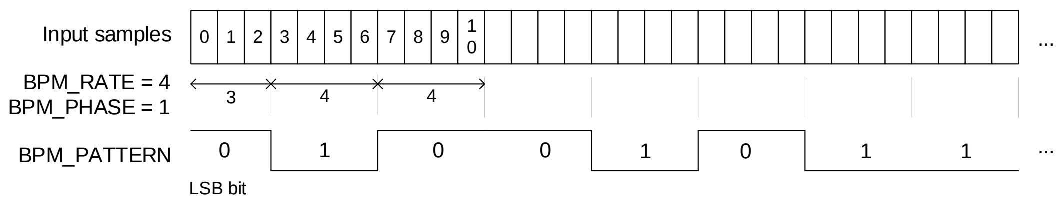


Figure 12: BPM Removal Capability

There is another register BPMPHASE that specifies the number of consecutive samples for which the first BPM bit is applied. Note that this is applicable only for the first BPM bit. If BPMPHASE = 0, then the first BPM bit is applied for BPMRATE number of samples. Otherwise, the first BPM bit is applied for BPMRATE – BPMPHASE number of samples. For example, if BPMPHASE = 1 and BPMRATE = 4, then the first BPM bit is applied for 4-1 = 3 samples, and then subsequent BPM bits are applied with periodicity of 4 samples for each bit. This is shown in Figure 12 . If multiple iterations (for example, four back-to-back FFTs in a single parameter-set using BCNT=3) are done, then the same BPM pattern gets applied to the input samples in each iteration.

Note the limitation that the BPM pattern register is 256 bits long, hence, the maximum BPM sequence length that is supported is 256. For higher BPM sequence length, the alternate approach of pre-multiplying the window coefficients stored in the window RAM may be considered.

1.1.8 Channel combining

The accelerator allows combining of data samples from multiple channels into one effective channel. This is done via an accumulator that sums a set of successive samples along the A dimension. The samples to combine are indicated by a 256-length bit-vector, CHAN_COMB_VEC. A string of '1' or a string of '0' sums the samples corresponding to the indices. A '10' or '01' transition demarcates the groups. This results in a variable rate output based on the length of 1s-string & 0s-string. The down-stream processing stalls during the grouping. The LSB of this bit-vector corresponds to the first input sample. An additional register, CHAN_COMB_SIZE needs to be programmed to indicate the number of samples after this combination in A dimension. The combination pattern is the same across all iterations. The channel combining is feature is enabled only if CHAN_COMB_EN is set to 1. The bit-vector CHAN_COMB_VEC is specified through eight 32-bit registers, CHAN_COMB_VEC0, CHAN_COMB_VEC1, … CHAN_COMB_VEC7.

Channel combination is illustrated with an example here: if CHAN_COMB_VEC0 = 0x008C (and CHAN_COMB_VEC1-7 are 0s), SRCACNT = 11 (i.e. 12 samples), and CHAN_COMB_SIZE = 5, then the first 2 samples are combined into one output sample, the next 2 samples are combined into one output sample, and the next 3 samples are combined into one output sample, the next sample is output as another output sample, and the last 4 samples are combined into one output sample. Note that the 1s and 0s in CHAN_COMB_VEC can be flipped to achieve the same effect.

1.1.9 Zero Insertion

In addition to traditional zero padding before FFT, the accelerator includes a Zero Insertion feature. This feature allows filling of zeros at arbitrary locations in the A dimension, prior to windowing and FFT. Zero insertion is typically meant for angle-dimension FFT to account for missing antenna positions. The Zero Insertion capability is applicable to FFT sizes up to 256.

ZERO_INSERT_MASK is a 256-bit vector (split in eight 32-bit registers) that holds the positions of zero-insertion. A bit value of ‘0’ inserts a zero and a value of ‘1’ means the input is passed through. Input streaming is stalled during the zero insertion. The register ZERO_INSERT_NUM is used indicate the number of zeros to be inserted. The register ZERO_INSERT_EN, if set to 1, enables this feature.

Zeros insert locations directly refer to FFT’s input sample indices. For example, if FFT size is 16, SRCANT is 11, ZERO_INSERT_MASK0 = 0xFF0F means that the first 4 samples are passed through, 4 zeros are inserted, and the next 8 samples are passed through. As with channel combination, the zero insertion pattern remains same across iterations.

This mode is not to be confused with zero padding mode where zeros are appended to an input stream. The number of zeros padded at the end depends on FFTsize, zero insertion and channel combination. It is recommended to avoid zero insertion for the last sample in the iteration (the last sample should come through input formatter).

1.1.10 Pre-Processing Block – Register Descriptions

Table 2 lists all the registers of the pre-processing block. As explained in *Radar Hardware Accelerator User’s Guide – Part 1*, some of the registers are common (common for all parameter-sets) registers, whereas, some others are “part of each parameter-set”. For each register, this distinction is captured as part of the register description in Table 2.

Table 2: Pre-Processing Registers

Register	**Width**	**Parameter-Set? (Y/N)**	**Description**
DCEST_RESET_MODE	2	Y	2-bit field that controls the reset behavior for all 12 DC accumulators 00 : Hold Accumulator state without updating 01 : Reserved 10 : Reset at start of param-set (i.e., per-chirp DC estimation). 11 : Reset at start of param-set only if loop counter is 0 (i.e., per-frame DC estimation)

DC_EST_SCALE	9	N	Programmable fine scaling for DC estimation: 9-bit scale applied to all 12 DC accumulators. This is followed by right shift and truncation. Multiplies the accumulator output by DCEST_SCALE/256. Default value is 256 giving a scale of 1.0. Setting it to 128, gives a scale of 0.5.
DC_EST_SHIFT	4	N	Programmable right shift for DC estimation: Right bit-shift applied to all 12 DC accumulator outputs. Cannot be bypassed. Accumulator outputs are scaled by 2^(8 + 6 + DCEST_SHIFT). Valid range for this register is 0 to 14 (i.e., scaling of 2^14 to 2^28). Note that DCEST_SHIFT = 15 is not supported.
DC_ACC_I_<n>_VAL_LSB DC_ACC_Q_<n>_VAL_LSB n = 0:11	32	N	These read-only registers provide the lower 32 bits of 36b DC estimation accumulator values –I&Q for 12 streams for processor read-out.
DC_ACC_I_<n>_VAL_MSB DC_ACC_Q_<n>_VAL_MSB n = 0:11	4	N	These read-only registers provide the upper 4 bits of 36b DC estimation accumulator values –I&Q for 12 streams for processor read-out.
DC_EST_RESET_SW	1	N	Software reset for DC accumulators: Setting this register bit to 1 resets all 12 DC estimation accumulators. This is a self-clearing reset bit.
DC_EST_I_<n>_VAL, DC_EST_Q_<n>_VAL n = 0:11	24	N	These read-only registers provide the DC estimates – I&Q for 12 streams – for the processor to read.
DC_ACC_CLIP_STATUS	12	N	Clip status indication (read-register) for the 12 DC accumulators (both I and Q combined). Value of 1 indicates a clipping event occurred.
DC_EST_CLIP_STATUS	12	N	Clip status indication (read-register) for the 12 DC estimates (both I and Q combined). Value of 1 indicates a clipping event occurred.
DCSUB_EN	1	Y	Enable or Disable DC subtraction. If this register bit is set to 1, DC subtraction is enabled. Else, it is disabled.
DCSUB_SELECT	1	Y	Source select for DC subtraction: 0 : Value comes from processor via DC_SW_I<n> & DC_SW_Q<n> 1: Value comes from built-in DC estimation hardware, i.e., DCEST_I<n> & DCEST_Q<n>
DC_I<n>_SW, DC_Q<n>_SW n = 0:11	24	N	User-programmed DC values used for DC subtraction. These registers are relevant only when DCSUB_SELECT is 0.
DC_SUB_CLIP	1	N	Clip status indication (read-register) for DC subtraction node (both I and Q combined). Value of 1 indicates a clipping event occurred.
INTF_LOC_THRESH_EN	1	Y	Enable/Disable for Interference localization (marking out): This register bit controls the enable/disable for the interference marking (setting Interference Indicator Bit) feature. The feature is enabled if this register bit is set to 1.
INTF_LOC_THRESH_MAG<n>_SW n = 0:11	24	N	Software Interference threshold for Magnitude These registers are used to specify the user-programmed threshold for marking out samples affected by interference in the Interference localization block. The magnitude of each incoming samples is compared with this threshold to decide whether it is corrupted by interference or not.

INTF_LOC_THRESH_MAGDIFF<n>_SW n = 0:11	24	N	Software Interference threshold for Magnitude of backward difference These registers are used to specify the user-programmed threshold for marking out samples affected by interference in the Interference localization block. The magnitude of backward difference of incoming samples is compared with this threshold to decide whether it is corrupted by interference or not.
INTF_LOC_THRESH_MODE	2	Y	Interference detection mode selection: This register is used to control the mode for interference detection in the Interference localization block. 00 : Magnitude OR Magnitude difference 01: Only Magnitude difference 10: Only Magnitude 11 : Magnitude AND Magnitude difference
INTF_LOC_THRESH_SEL	2	Y	Select the source of interference threshold 0 : User-defined threshold via INTERFTHRESH_MAG_SW and INTERFTHRESH_MAGDIFF_SW 1 : Single threshold based on built-in interference statistics outputs using sum value across collected interference statistics 2 : Threshold based on built-in interference statistics outputs, with each statistic being used for corresponding iteration (RX channel)
INTF_STATS_RESET_MODE	2	Y	Reset mode control for Interference statistics accumulators: Controls the reset behavior for all 12 magnitude and magdiff accumulators. 00 : Hold Accumulator state without updating 01 : Free-running accumulator mode 10 : Reset at start of parameter-set (i.e.,per-chirp accumulation). 11 : Reset at start of parameter-set only if loop counter is 0 (i.e., per-frame)
INTF_STATS_MAG_SCALE	8	N	Programmable fine scaling for Interference statistics Magnitude: Scaling applied to INTF_STATS_MAGACC<n> from interference statistics block.
INTF_STATS_MAG_SHIFT	3	N	Programmable right shift for Interference statistics Magnitude: Right bit-shift applied to the interference magnitude accumulator. Total right shift of the accumulator is 2^(3+6+INTF_STATS_MAG_SHIFT). Valid range for this register is 0 to 6 (i.e., the total right shift can't be more than 2^15).
INTF_STATS_MAGDIFF_SCALE	8	N	Programmable fine scaling for Interference statistics MagDiff: Scaling applied to INTF_STATS_MAGDIFFACC<n> from interference statistics block.
INTF_STATS_MAGDIFF_SHIFT	3	N	Programmable right shift for Interference statistics MagDiff: Right bit-shift applied to the interference magdiff accumulator. Total right shift of the accumulator is 2^(3+6+INTF_STATS_MAGDIFF_SHIFT). Valid range for this register is 0 to 6 (i.e., the right shift can't be more than 2^15).
INTF_STATS_MAG_ACC_<n>_LSB n = 0:11	32	N	These read-only registers provide the lower 32 bits of 36b magnitude accumulator values –I&Q 12 streams for processor read-out.
INTF_STATS_MAG_ACC_<n>_MSB n = 0:11	4	N	These read-only registers provide the upper 4 bits of 36b magnitude accumulator values –I&Q 12 streams for processor read-out.
INTF_STATS_MAGDIFF_ACC_<n>_LSB n = 0:11	32	N	These read-only registers provide the lower 32 bits of 36b magnitude difference accumulator values –I&Q 12 streams for processor read-out.
INTF_STATS_MAGDIFF_ACC_<n>_MSB n = 0:11	4	N	These read-only registers provide the upper 4 bits of 36b magnitude difference accumulator values –I&Q 12 streams for processor read-out.

INTF_STATS_RESET_SW	1	N	Software reset bit for all the interference statistics accumulators. This is a self-clearing reset bit.
INTF_LOC_THRESH_MAG<n>_VAL, n=0:11	24	N	Read-only thresholds – scaled and shifted INTF_STATS_MAGACC<n> of interference statistics block
INTF_STATS_SUM_MAG_VAL	24	N	Sum of INTF_LOC_THRESH_MAG<n>_VAL, based on number of iterations. Useful as single magnitude threshold value across all iterations
INTF_LOC_THRESH_MAGDIFF<n>_VAL, n=0:11	24	N	Read-only thresholds – scaled and shifted INTF_STATS_MAGACCDIFF<n> of interference statistics block
INTF_STATS_SUM_MAGDIFF_VAL	24	N	Sum of INTF_LOC_THRESH_MAGDIFF<n>_VAL, based on number of iterations. Useful as single magnitude difference threshold value across all iterations
INTF_STATS_MAG_ACCUMULATOR_CLIP_STATUS	12	N	Read-only clip status indication register for 12 interference statistics magnitude accumulators INTF_STATS_MAGACC<n>. Value of 1 indicates a clipping event occurred in atleast one accumulator
INTF_STATS_MAGDIFF_ACCUMULATOR_CLIP_STATUS	12	N	Read-only clip status indication register for 12 interference statistics magnitude-difference accumulators INTF_STATS_MAGACCDIFF<n>. Value of 1 indicates a clipping event occurred in atleast one accumulator
INTF_STATS_THRESH_MAG_CLIP_STATUS	12	N	Read-only clip status indication register for 12 magnitude based interference threshold. INTF_LOC_THRESH_MAG<n>_VAL. Value of 1 indicates a clipping event occurred in atleast one estimate
INTF_STATS_THRESH_MAGDIFF_CLIP_STATUS	12	N	Read-only clip status indication register for magnitude-difference based interference threshold. INTF_LOC_THRESH_MAGDIFF<n>_VAL. Value of 1 indicates a clipping event occurred in atleast one estimate
INTF_STATS_SUM_MAG_VAL_CLIP_STATUS	1	N	Read-only clip status for sum of all magnitude thresholds computed by the statistics block. Value of 1 indicates that the sum clipped.
INTF_STATS_SUM_MAGDIFF_VAL_CLIP_STATUS	1	N	Read-only clip status for sum of all magnitude difference thresholds computed by statistics block. Value of 1 indicates that the sum clipped.
INTF_LOC_COUNT_ALL_CHIRP	12	N	Read-only register indicating the number of samples that exceeded the threshold in a given param-set. The count is saturated to 2^12 – 1.
INTF_LOC_COUNT_ALL_FRAME	20	N	Read-only register indicating the number of samples that exceeded the threshold across multiple executions of same param-set. The count is saturated to 2^20 – 1.
INTF_MITG_EN	1	Y	If this bit is set, the interference mitigation path is activated, else it is bypassed.
INTF_MITG_PATH_SEL	2	Y	Based on the value of this register, one of the three paths is activated. 00b: Simple Zeroing out 01b: Windowed Zeroing out 10b: Linear Interpolation 11b: Reserved

INTF_MITG_WINDOW_PARAM0 INTF_MITG_WINDOW_PARAM1 INTF_MITG_WINDOW_PARAM2 INTF_MITG_WINDOW_PARAM3 INTF_MITG_WINDOW_PARAM4	5 (each)	N	This is a programmable array of window parameters. Each window parameter is an unsigned 5 bit integer. The length of the array is 5. The parameters of the window are assumed to be monotonically ascending. For example : val = floor(hanning(14)*32) INTF_MITG_WINDOW_PARAM = val(2:6); If a shorter window (of length less than 5) is desired, some of the earlier window parameters can be set to 31. This sets the window parameter to 31/32 (or ~1).
INTF_MITG_CNTTHRESH	5	Y	The (total) number of non-zero IIB within the 'Hysteresis window' should exceed this threshold for the sample-under-test to be considered to be affected by interference.
INTF_MITG_RIGHT_HYST_ORD	4	Y	The length of the IIB array considered on the right side of (i.e. after) the sample under test. Range : 0 to 15.
INTF_MITG_LEFT_HYST_ORD	4	Y	The length of the IIB array considered on the left side of (i.e. before) the sample under test. Range : 0 to 15.
CMULT_MODE	4	Y	Complex multiplication mode selection: This register is used to configure the mode of the complex multiplication sub-block. A value of 0000b disables/bypasses the complex multiplication. Any other value chooses one of nine available modes of operation . Detailed description of the nine modes in the main description section.
CMULT_SCALE_EN	1	Y	Complex multiplier iteration enable : This register bit is applicable in certain modes of the complex multiplication pre-processing to enable per-iteration change of the complex scalar coefficient. When using scalar multiplication mode of the complex multiplier (CMULT_MODE = 0101b), if CMULT_SCALE_EN is set to 1, then the input samples are multiplied by a different complex scalar (i.e., ICMULTSCALE0, QMULTSCALE0 to ICMULTSCALE11, QMULTSCALE11) for each iteration. Else, a constant complex scalar ICMULTSCALE0 and QCMULTSCALE0 is applied to all samples across all iterations. When using vector multiplication mode (CMULT_MODE = 0110b), if CMULT_SCALE_EN is set to 1, then instead of pulling the coefficients for vector multiplication from the Vector Multiplication Coefficients RAM, the input samples are multiplied successively by ICMULTSCALE0, QMULTSCALE0 to ICMULTSCALE11, QMULTSCALE11 registers.
ICMULT_SCALE<n> QCMULT_SCALE<n> n = 0 to 11	21	N	Coefficients for Complex multiplication: Refer the description for CMULT_SCALE_EN register.
VEC_MULT_RAM[2048]	48	N	Vector multiplication RAM : Stores the complex vector multiplication coefficients used in modes 6, 7 and 8. Layout shown in Fig. 11

TWIDINCR	14	Y	Frequency shifter configuration: When the complex multiplication sub-block is programmed in one of the frequency shifter modes (CMULT_MODE = 0001b or 0010b), this register is used to indicate the amount of frequency shift. When the complex multiplication sub-block is programmed in FFT stitching mode (CMULT_MODE = 0011b), the last two bits of this register specify whether it is 4K or 8K FFT stitching. Specifically, if the last two bits are 01b, then it is 4K FFT stitching and if the last two bits are 10b, then it is 8K FFT stitching. Values of 00b and 11b are reserved. Also, the 12 MSB bits of this register must be kept zero in the FFT stitching mode. In all other modes of the complex multiplication sub-block, this 14-bit register must be kept as 0. When the complex multiplication sub-block is programmed with CMULT_MODE = 0110b, 0111b or 1000b, then the 12 MSBs of this register are used as an address offset for the Vector Multiplication Coefficients RAM (the 2 LSBs must be kept 0). When the complex multiplication sub-block is programmed with CMULT_MODE = 1001b, then the 6 LSBs of this register are used an address offset for TWID_ANGLE_RAM, with bit 13 enabling auto-address increment over iterations and bit 12 enabling address saturation or address roll-over after 63.
TWIDINCR_DELTA_FRAC	10	N	Fractional frequency increment per execution of the parameter-set: Frequency shift value to be accumulated at the end of current parameter-set. Refer main description for more details.
TWID_ANGLE_RAM[64]	32	N	Frequency and Phase values for generic frequency shifter mode: This is only applicable when CMULT_MODE = 1001b. In this mode, these 64 registers represent the starting phase and frequency values used. Layout is shown in Fig. 11
RECWIN_MODE	1	Y	Recursive window mode select bit.(0) –the K value increments with iteration. (1) –the K value increments with paramset execution count. K always starts from 0
RECWIN_RESET_SW	1	N	Software reset bit for recursive window K value. This is a self-clearing reset bit.
RECWIN_INIT_KVAL	12	N	RESERVED. This is a reserved register and should be always kept 0.
TWID_INCR_DELTA_FRAC_RESET_SW	1	N	Software reset bit for fine frequency increment accumulator (in CMULT_MODE = 1010b). This is a self-clearing reset bit.
TWID_INCR_DELTA_FRAC_CLIP_STATUS	1	N	Read-only register bit that indicates clip status for the fine-frequency increment accumulator (in CMULT_MODE = 1010b).
BPM_EN	1	Y	Enable/Disable BPM removal: This register bit specifies whether the BPM removal needs to be enabled or not. If this register is set, then BPM removal is enabled prior to feeding samples from the input formatter into the core computational unit.

BPM_PATTERN_0 BPM_PATTERN_1 : BPM_PATTERN_7	256	N	BPM pattern: Specifies the BPM pattern to be used to multiply the input samples if BPM removal is enabled. The 256-bit word is split into 8 32-bit words as [BPM_PATTERN_7, BPM_PATTERN_6, BPM_PATTERN_5, BPM_PATTERN_4, BPM_PATTERN_3, BPM_PATTERN_2, BPM_PATTERN_1, BPM_PATTERN_0]
BPM_RATE	10	N	BPM rate: Specifies the number of input samples corresponding to each BPM bit. Minimum valid value for this register is 1.
BPMPHASE	4	Y	BPM starting phase: Specifies the starting phase of the BPM pattern periodicity. For more information, see the detailed description.
CHAN_COMB_EN	1	Y	Enable/Disable channel combining: If this register bit is set to 1, then the channel combining feature is enabled.
CHAN_COMB_VEC_0, CHAN_COMB_VEC_1, . . CHAN_COMB_VEC_7	256	N	Sample index indicator for Channel combining : This register indicates the sample indices that need to be combined in the Channel combiner block. A ‘01’ or ‘10’ transition demarcates the groups. The 256-bit word is split into 8 32-bit words as [CHAN_COMB_VEC_7,CHAN_COMB_VEC_6, CHAN_COMB_VEC_5, CHAN_COMB_VEC_4, CHAN_COMB_VEC_3, CHAN_COMB_VEC_2 CHAN_COMB_VEC_1, CHAN_COMB_VEC_0]
CHAN_COMB_SIZE	8	N	This register indicates the number of samples after channel combination in each iteration.
CHANNEL_COMB_CLIP_STATUS	1	N	Clip status indication (read-register) during channel combining. Value of 1 indicates a clipping event occurred.
ZERO_INSERT_EN	1	Y	Enable/Disable zero-insertion: If this register bit is set to 1, then the zero-insertion feature is enabled.
ZERO_INSERT_NUM	8	N	This register indicates the number of zeros to be inserted in each iteration.
ZERO_INSERT_MASK_0, ZERO_INSERT_MASK_1, . . ZERO_INSERT_MASK_7	256	N	Sample index indicator for Zero Insertion: This is a 256-bit register that holds the positions of zero-insertion. A bit value of ‘0’ inserts a zero at the corresponding index location. A bit value of ‘1’ means the input is passed through. The input sample stream from Input formatter is stalled during the zero insertion. The 256-bit word is split into 8 32-bit words as [ZERO_INSERT_MASK_7, ZERO_INSERT_MASK_6 ZERO_INSERT_MASK_5, ZERO_INSERT_MASK_4 ZERO_INSERT_MASK_3, ZERO_INSERT_MASK_2 ZERO_INSERT_MASK_1, ZERO_INSERT_MASK_0]

2 Core Computational Unit – CFAR Engine

This section describes the CFAR engine block present in the core computational unit.

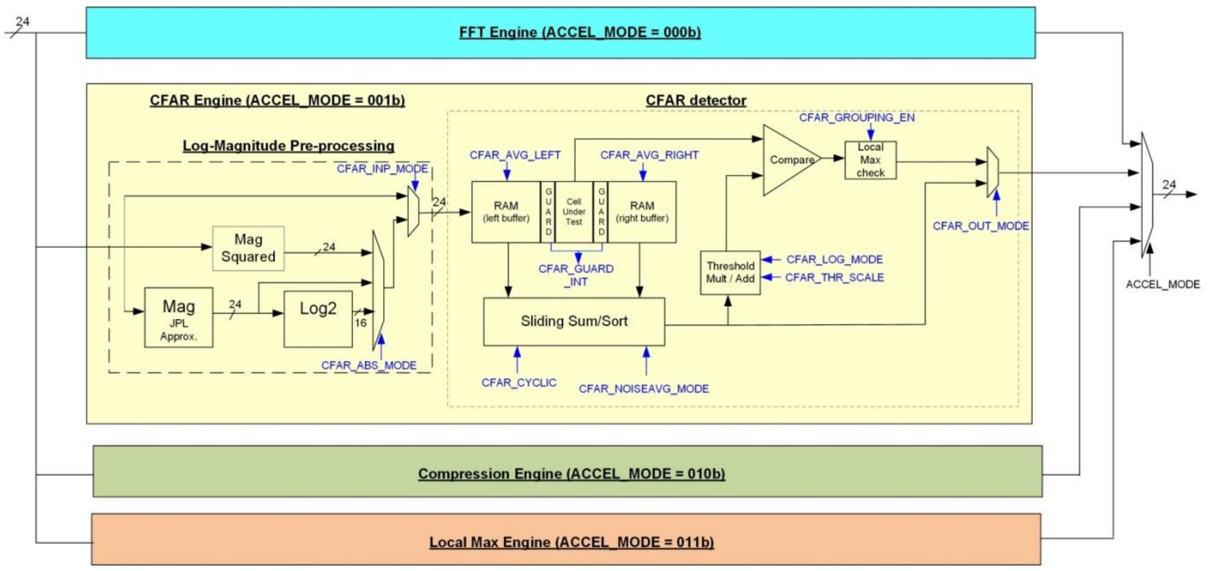


Figure 13: CFAR Engine

2.1 *CFAR Engine*

The CFAR engine (Figure 13) is a module that enables detection of objects, by identifying peaks in the FFT output. Although there are several detection algorithms, the accelerator supports CFAR-CA and CFAR-OS algorithms. CFAR-CA stands for constant false alarm rate – Cell Averaging. CFAR-OS stands for constant false alarm rate – Ordered Statistic.

As shown in figure, the CFAR engine path is selected by setting the accelerator mode ACCEL_MODE = 001b. In this mode, the FFT path is not usable simultaneously and the input 24-bit samples from the input formatter block will be routed into the CFAR engine. The CFAR engine has capability to perform CFAR- detection processing (both linear and logarithmic CFAR modes are available) and generate a peak list.

In CFAR, the processing steps involve computing a threshold for each sample under test (cell under test) and deciding whether a peak is detected or not based on whether the cell under test crosses that threshold. Additionally, peak grouping may be done, where a peak is declared only if the cell under test is greater than or equal to its most immediate neighboring cells to its left and right. One thing to note here is that for peak grouping, the left and right neighboring cells themselves are not required to be CFAR qualified.

In CFAR-CA case, for each cell under test, the computation of threshold is done by averaging the magnitude (or magnitude- squared or log-magnitude) of a specified number of noise samples to the left and right of the cell under test to determine a 'surrounding noise level' and then applying a scale factor (or addition factor in case log-magnitude is used) on that surrounding noise average to determine the threshold. Thus, the CFAR-CA detector takes one cell at a time, computes the threshold and decides whether a valid peak is present at that cell. In the case of CFAR-OS, for each cell under test, the computation of threshold is done by sorting the magnitude (or magnitude-squared or log-magnitude) of a specified number of noise samples to the left and right of the cell under test and selecting a specific "K-th" lowest value from the sorted list as representative of the surrounding noise level, and then applying a scale factor on that value to determine the threshold.

2.1.1 CFAR Engine – Operation

The CFAR engine receives 24-bit input samples from the Input Formatter block. Typically, these are unsigned real samples, representing the magnitude or magnitude-squared or log-magnitude of the FFT output. However, the input to CFAR engine can instead be complex samples, in which case, either magnitude or magnitude- squared or log-magnitude of the complex samples can be computed inside the CFAR engine itself. This is done by the log-magnitude pre-processing sub-block inside the CFAR engine (

Figure 14). The real unsigned result from this pre-processing operation is sent to CFAR detection processing. The registers CFAR_INP_MODE and CFAR_ABS_MODE are used to configure real vs. complex input, as well as the nature of pre-processing required (refer register description table). The log-magnitude computation uses the same JPL approximation for magnitude calculation and the same look-up table (LUT) approximation for log2 computation as described in Part 1 of the user guide for FFT engine post-processing. Note that for the case of real input (i.e., CFAR_INP_MODE = 1), the input samples must be unsigned. In this case, CFAR_ABS_MODE register has no effect.

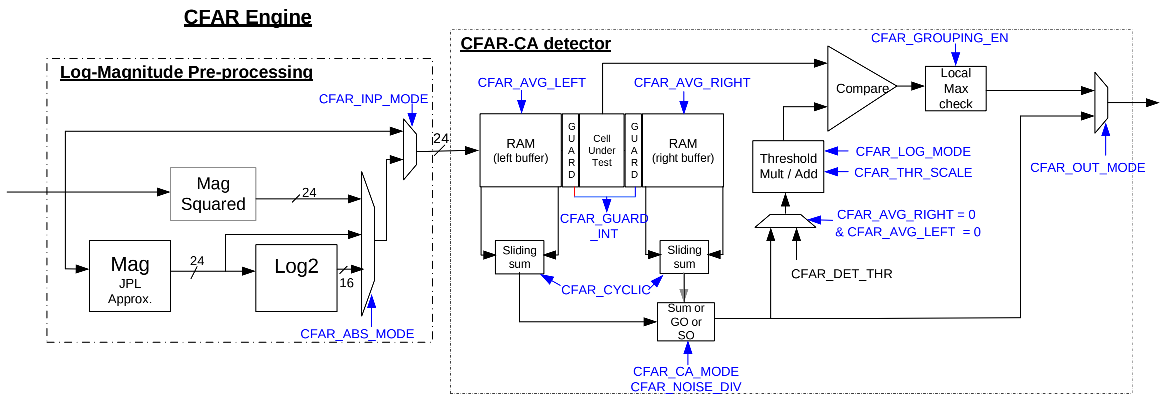


Figure 14: CFAR-CA Engine Block Diagram

As described earlier, the CFAR- detection processing involves finding a “surrounding noise level" for each cell under test and then determining a threshold that is a function of the surrounding noise level. The cell under test is compared against this threshold to decide whether a peak is present or not in that cell. To calculate the threshold, the surrounding noise level is multiplied with (or added to) a threshold scaling factor specified in CFAR_THRESH register.

There are two modes in which the CFAR detector can be used – in non-logarithmic mode (a.k.a linear CFAR), the threshold scale factor is multiplied, and in logarithmic mode (a.k.a logarithmic CFAR), the threshold scale factor is added. This is decided based on CFAR_LOG_MODE register.

Note: The linear and logarithmic modes are available for CFAR-CA and its variants. Their detection cores are built with 24-bit datapath width. Only the logarithmic mode is available for CFAR-OS. The CFAR-OS detection core is built with 16-bit datapath width, which is sufficient in logarithmic mode.

The final detection threshold that is so obtained is used to compare against the cell under test to determine whether a peak is detected in that cell.

Table 3 summarizes the register settings for the different CFAR modes of operation.

Table 3 : CFAR modes and Register settings

Desired CFAR Mode	Input Real or Complex	Desired Pre- Processing	Register Values to Use		
			CFAR_INP_MODE	CFAR_ABS_MODE	CFAR_LOG_MODE
Linear CFAR	Real	N/A	1	00	0
	Complex	Magnitude	0	10	0
		Mag-squared	0	00	0
		Log2-Mag	0	11	0
Log CFAR	Real	N/A	1	00	1
	Complex	Log2-Mag	0	11	1

Desired CFAR Algorithm	CFAR_CA_MODE Register Setting
CFAR-CA	00
CFAR-CAGO	01
CFAR-CASO	10
CFAR-OS	11

The surrounding noise level computation has multiple options – cell averaging (CFAR-CA), cell averaging with greater-of selection (CFAR-CAGO), cell averaging with smaller-of selection (CFAR- CASO) and ordered statistic (CFAR-OS). The register CFAR_CA_MODE is used to select one among CFAR-CA, CFAR-CAGO, CFAR- CASO and CFAR-OS modes. In CFAR-CA, the noise samples on the left side and right side of the cell under test (after ignoring some guard cells on either side) are simply averaged to determine the surrounding noise level. In CFAR-CAGO, the noise samples on the left side and right side are averaged independently and the greater of the two is used to determine the threshold. In CFAR-CASO, the lesser of the two is used. In CFAR-OS, the noise samples on the left side and right side of the cell under test (after ignoring some guard cells on either side) are sorted and the “K-th” lowest value from the sorted list is selected as the surrounding noise level. The selection of “K-th value” from the sorted list is based on the register CFAR_OS_KVALUE (see table of registers for more details on this register).

The number of samples on the left side and right side used for computing the noise average is configured using CFAR_AVG_LEFT and CFAR_AVG_RIGHT registers and the number of guard cells is configured using CFAR_GUARD_INT register (Figure 15). The number of samples used for left side noise averaging is given by 2*CFAR_AVG_LEFT. The number of samples used for right side noise averaging is given by 2*CFAR_AVG_RIGHT. The number of guard cells that are ignored on each side of the cell under test is given by CFAR_GUARD_INT. For example as shown in Figure 15, if CFAR_AVG_LEFT = CFAR_AVG_RIGHT = 16, and CFAR_GUARD_INT = 3, then it means that the most immediate three samples each to the left and right of the cell under test are skipped and then, 32 samples on the left and 32 samples on the right side are used for noise averaging. Note that even though the term noise averaging is used here, the actual implementation simply adds the noise samples first and the “averaging” is done as a divide by a power-of- 2 as specified in a separate register, CFAR_NOISE_DIV. These registers are described in Table 7.

Note: The CFAR engine also supports a special “constant threshold mode” of CFAR detection. In this special mode, the detection threshold value to compare with each cell-under-test is based on a user configurable constant – CFAR_DET_THR. This detection threshold value is independent of “surrounding noise level”, and the detection comparison depends only CFAR_DET_THR, CFAR_THR_SCALE, and CFAR_LOG_MODE. This mode of operation can be achieved by setting the engine in CFAR-CA mode and additionally setting CFAR_AVG_LEFT = CFAR_AVG_RIGHT = 0. In this constant threshold mode, CFAR_THRESH scale factor is multiplied with CFAR_DET_THR in the linear mode, and in the logarithmic mode the threshold scale factor is added.

In case of CFAR-CA, the valid values for CFAR_AVG_LEFT and CFAR_AVG_RIGHT is any number between 0 and 63 (except 1), which means that the number of samples each on the left side and right side used for noise averaging can be one of 0, 4, 6, 8, 10, 12, 14, … 124, 126. The values of CFAR_AVG_LEFT and CFAR_AVG_RIGHT can be different in cyclic mode of CFAR and need to be equal in non-cyclic mode (both are described in a later section).

However, in the case of CFAR-OS, the valid values for CFAR_AVG_LEFT and CFAR_AVG_RIGHT are highly restricted. They need to be equal (i.e., same window size on left and right sides) and further, the only values supported for these registers in CFAR-OS mode are: 0, 4, 6, 8, 12, 16, 24 and 32 (which corresponds to number of samples being 0, 8, 12, 16, 24, 32, 48 and 64 on either side). Note that the register CFAR_NOISE_DIV, which is used in CFAR-CA for noise “averaging”, is not applicable in case of CFAR-OS.

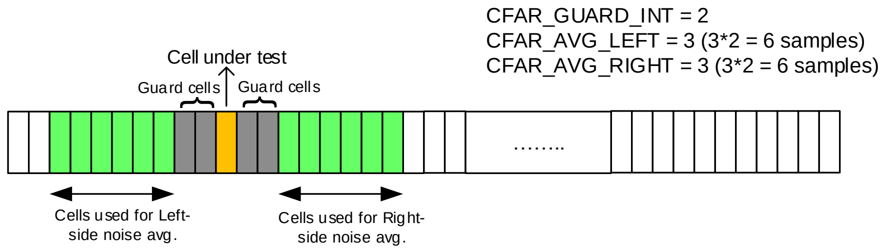

CFAR_GUARD_INT = 2
CFAR_AVG_LEFT = 3 (3*2 = 6 samples)
CFAR_AVG_RIGHT = 3 (3*2 = 6 samples)
Cell under test
Guard cells
Guard cells
……..
Cells used for Left-side noise avg.
Cells used for Right-side noise avg.

Figure 15: CFAR-CA: Cells used for Surrounding Noise Average

As mentioned earlier, the CFAR_THRESH register specifies the threshold scaling factor. This is an 18-bit register whose value is used to either multiply or add to the ‘surrounding noise level’ to determine the threshold used for detection of the present cell under test. If logarithmic mode is disabled (in magnitude or magnitude-squared mode), then the register value is multiplied with the surrounding noise level to determine the threshold, else it is added to the surrounding noise level. In the former case, this 18-bit register is interpreted as a 14.4 value and supports a range of values from 1/16 to 2^14-1. In the latter case (logarithmic mode), the 18-bit register is interpreted as a 7.11 value.

The CFAR engine supports a few output formats that are described next.

2.1.2 CFAR Engine – Output Formats

As part of CFAR detection, the cells that exceed the threshold are noted and this 'Detected Peaks list' is sent to the destination memory. Since the output format of the core computational unit is 24-bits I and 24-bits Q, the detected peaks list is formatted into 'I' and 'Q' channels as shown in Figure 16 below. The 24-bit I channel contains the index at which the peak is detected, with the MSB 12 bits containing the iteration number (corresponding to BCNT counter value) and the LSB 12 bits containing the sample index number (corresponding to SRCACNT counter value). The 24-bit Q channel contains the surrounding noise level value or the cell under test value of that detected peak. This is chosen based on CFAR_OUT_MODE register setting. Instead of 'Detected Peaks list', it is also possible for the CFAR engine to send out the raw 'surrounding noise level' value for each cell. This is called 'Raw output mode'.

Figure 16, Figure 17 and Table 4 show the different output formats available.

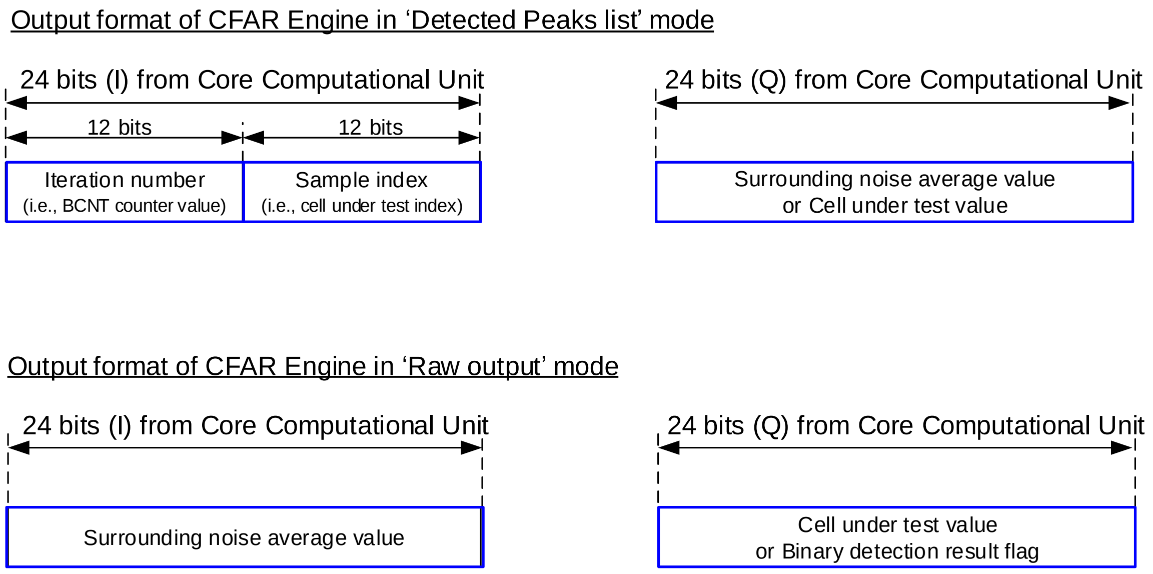

Output format of CFAR Engine in 'Detected Peaks list' mode
24 bits (I) from Core Computational Unit
12 bits
12 bits
Iteration number
(i.e., BCNT counter value)
Sample index
(i.e., cell under test index)
24 bits (Q) from Core Computational Unit
Surrounding noise average value
or Cell under test value
Output format of CFAR Engine in 'Raw output' mode
24 bits (I) from Core Computational Unit
Surrounding noise average value
24 bits (Q) from Core Computational Unit
Cell under test value
or Binary detection result flag

Figure 16: CFAR Engine Output Format

In detected peaks list mode, only the detected peaks are output to the destination memory. In this case, the read-only register CFARPEAKCNT indicates how many peaks have been totally detected, so that the main processor can read that many locations from the destination memory. In this mode, the number of peaks stored in the destination memory is limited to a maximum of 4095, or DSTACNT, whichever is smaller. If more peaks are detected beyond this number, they wrap around and circularly overwrite the same locations in the destination memory. Also, in this mode, the register DSTBINDX is not applicable and is ignored.

While detecting peaks, if 'peak grouping' is required, then it can be enabled using CFAR_GROUPING_EN register. In this case, a peak is declared as detected only if it the cell under test exceeds the threshold, as well as, if the cell under test exceeds the two neighboring cells to its immediate left and right (the peak is a local maximum).

Further, there is a special mode of the CFAR Engine called "Dominant peaks mode". This mode can be used to create a modified copy of the range FFT output, where the range FFT bins corresponding to large objects are kept as is, and the range FFT bins which do not correspond to large objects are masked (zeroed out). In this special mode of the CFAR engine, the CFAR engine takes complex (I & Q) input samples and the CFAR engine outputs values that are simply equal to its input values at the sample indices corresponding to the detected peaks, and outputs zeros at all other sample indices. This thereby gives an output array which is the same as the input array AND'ed with the binary CFAR peak detection flags. Note that this special mode is only meaningful when the CFAR input is complex , i.e., when CFAR_INP_MODE = 0. The purpose of this "Dominant Peaks mode" is to re-construct interference affected samples, where this mode can be used to extract the FFT output bins corresponding to large peaks and later, doing an IFFT on this output to re-construct the time domain signal corresponding only to dominant peaks.

Output format of CFAR Engine in the special 'Dominant peaks' mode

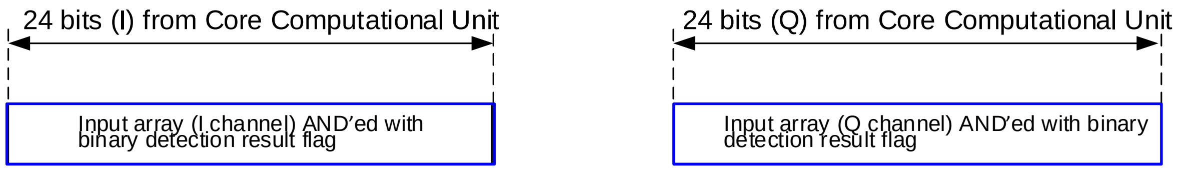

24 bits (I) from Core Computational Unit
24 bits (Q) from Core Computational Unit
Input array (I channel) AND'ed with binary detection result flag
Input array (Q channel) AND'ed with binary detection result flag

Figure 17: CFAR Engine Output Format in Dominant Peaks mode

Table 4: CFAR Output modes and Register Settings

CFAR Output Mode	**I Channel Output**	**Q Channel Output**	**Register Settings (CFAR_ADV_OUT_MODE, CFAR_OUT_MODE)**
Raw output mode (all cells are output)	Surrounding noise level	Cell under test value	(0,00)
	Surrounding noise level	Binary detection result flag (0 or 1)	(0,01)
Detected peaks list mode (only detected peaks are output)	Peak index	Surrounding noise level value	(0,10)
	Peak index	Cell under test value	(0,11)
Dominant peaks mode	Reserved	Reserved	(1,00)
	Input array (I channel) AND'ed with binary detection result flag	Input array (Q channel) AND'ed with binary detection result flag	(1,01)

2.1.3 *CFAR Engine – Cyclic vs. Non-Cyclic*

The register CFAR_CYCLIC specifies whether the CFAR detector needs to work in cyclic mode or in non-cyclic mode. In general, the programmed number of samples for noise level computation (specified by CFAR_AVG_RIGHT and CFAR_AVG_LEFT) are available fully only for the cells under test which are in the middle of the input array (Figure 18). For first several cells under test, the available number of samples to the left is lesser than the programmed number. Similarly, for the last several cells under test, the available number of samples to the right is lesser than programmed.

In cyclic mode (Figure 19), this is handled by wrapping around the edges in a circular manner. For a cell under test near the left edge, some samples from the right edge (circular wrap around the edge) are fetched to collect the programmed CFAR_AVG_LEFT number of left side samples for noise level computation. Similarly, for a cell under test near the right edge, an appropriate number of samples from the left edge are used (again, circular wrap around the edge).

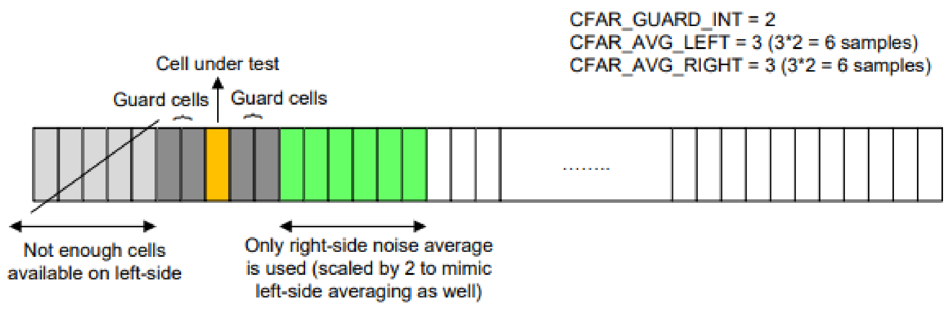

CFAR_GUARD_INT = 2
CFAR_AVG_LEFT = 3 (3*2 = 6 samples)
CFAR_AVG_RIGHT = 3 (3*2 = 6 samples)
Cell under test
Guard cells
Guard cells
....
Not enough cells available on left-side
Only right-side noise average is used (scaled by 2 to mimic left-side averaging as well)

Figure 18: Handling of samples near edges in non-cyclic mode

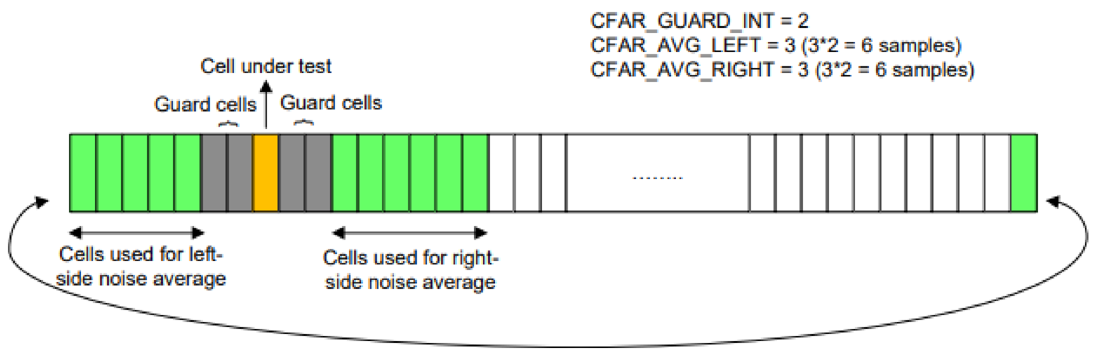

CFAR_GUARD_INT = 2
CFAR_AVG_LEFT = 3 (3*2 = 6 samples)
CFAR_AVG_RIGHT = 3 (3*2 = 6 samples)
Cell under test
Guard cells
Guard cells
....
Cells used for left-side noise average
Cells used for right-side noise average

Figure 19: Handling of samples near edges in cyclic mode

This cyclic CFAR implementation is accomplished through a combination of a few register settings within the CFAR engine, as well as in the input and output formatter blocks. Specifically, the input formatter is configured to send additional samples (repeat samples) in a circular manner wrapping around the left and right edges. This is achieved by using the A-dimension circular shift (SRCA_CIRCSHIFT) and wrap-around (SRCA_CIRCSHIFTWRAP) registers in the input formatter, such that the required number of extra samples at both edges are streamed into the CFAR engine. The cyclic CFAR mode only works when the number of cells under test is a power of 2.

For example (Figure 20), if the number of cells under test is 256, the average number of left and right noise samples is 32 each and the number of guard cells is 3 on either side. Then, the registers need to be programmed as shown in Table 5.

Table 5: Configuration example for CFAR Cyclic mode

Module	Register Setting	Comments
CFAR Engine	CFAR_GUARD_INT = 3	3 guard cells on either side

	CFAR_AVG_LEFT = 16
	CFAR_AVG_RIGHT = 16
		32 samples on left side and 32 samples on right side for noise averaging

Input Formatter	SRCACNT = 325	255 + (32+3) + (32+3), where 255 is the usually configured value of SRCACNT for a 256 sample vector, plus 32+3 additional samples for circular repeat at either end
	SRCA_CIRCSHIFT = 221	256 – (32+3), which is the starting offset for the circular shift, so that samples are streamed into CFAR engine start from this point

	SRCA_CIRCSHIFTWRAP = 8
	SRC_CIRCSHIFTWRAP3X = 0
		The circular wrap-around happens when SRCACNT counter value reaches 2^SRCA_CIRCSHIFTWRAP = 256

Output Formatter	REG_DST_SKIP_INIT = 0	No need to skip any samples at Output Formatter even though extra samples are fed into CFAR engine, because CFAR engine automatically strips out the extra samples
	DSTACNT = 255	256 outputs corresponding to 256 cells

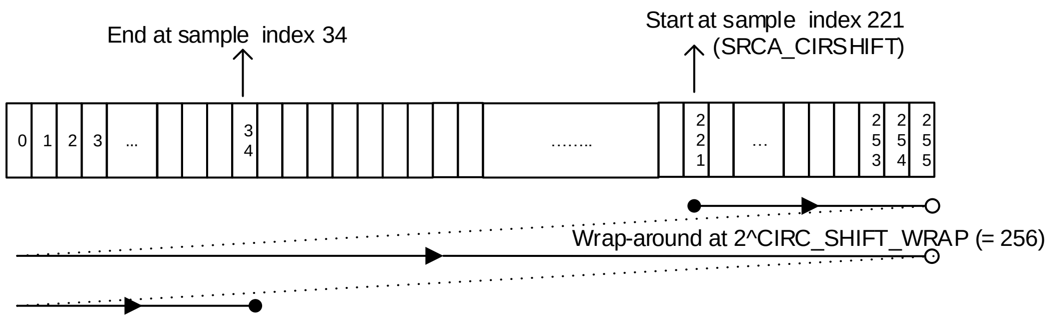

End at sample index 34
Start at sample index 221
(SRCA_CIRSHIFT)
0
1
2
3
…
34
……..
221
…
253
254
255
Wrap-around at 2^CIRC_SHIFT_WRAP (= 256)

Figure 20: Input Formatter Streaming for Cyclic CFAR example

However, the handling of edge samples in non-cyclic mode of CFAR is different. It is explained below – first for CFAR-CA and then for CFAR-OS versions.

In non-cyclic mode of CFAR-CA, if the number of available samples on the left for any cell under test is lesser than CFAR_AVG_LEFT, then the noise average is computed solely from the right side. This is done by calculating the noise sum as twice the right side noise sum. Similarly, if the number of available samples on the right is lesser than CFAR_AVG_RIGHT, then the noise average is computed solely from the left side. This is done by calculating the noise sum as twice the left side noise sum. It is required that the CFAR_AVG_LEFT and CFAR_AVG_RIGHT be programmed equally in non-cyclic mode – otherwise, the noise computation for the edge samples is not ideal.

In non-cyclic mode of CFAR-OS, the edge samples are handled as follows. For the cells under test that are near the edges, the number of available surrounding samples for sorting is lesser than programmed (CFAR_AVG_LEFT or CFAR_AVG_RIGHT). These available samples are first sorted and the Kth lowest value is selected as the noise level. It should be noted that this Kth lowest sample in the available samples may result in a sub-optimal noise level for edge samples than non-edge samples because the number of available samples is lesser for edge samples than for non-edge samples. A minor variant for better handling this edge sample case can be enabled by setting the register CFAR_OS_NONCYC_VARIANT_EN to 1. In that case, available samples are first sorted. But instead of using the programmed K value directly for noise sample selection from the sorted array, a proportionally scaled down value is used based on the number of available surrounding samples for each cell under test. This is illustrated in Table 6 below, where, L represents the programmed CFAR_AVG_LEFT (same as CFAR_AVG_RIGHT) and K represents the programmed CFAR_OS_KVAL. It is required that the CFAR_AVG_LEFT and CFAR_AVG_RIGHT be programmed equally in non-cyclic mode.

Table 6: Internal K value used in CFAR-OS non-cyclic mode

No. of available samples on one side (excluding guard)	No. of available samples on the other side (excluding guard)	Internal K value used for noise sample selection (CFAR_OS_NON_CYC_ VARIANT_EN = 0)	Internal K value used for noise sample selection (CFAR_OS_NON_CYC_ VARIANT_EN = 1)
L	0 to floor(L/4)-1	K	floor (4K/8)
L	floor(L/4) to floor(2L/4)-1	K	floor (5K/8)
L	floor(2L/4) to floor(3L/4)-1	K	floor (6K/8)
L	floor(3L/4) to floor(4L/4)-1	K	floor (7K/8)
L	L	K	K

In general, it is expected that CFAR Engine will be used for arrays much larger than the configured left and right window and guard lengths. Specifically, the ACNT should exceed the sum of configured left and right window and guard lengths.

2.2 CFAR Engine – Register Descriptions

Table 7 lists all the registers of the CFAR engine block.

Table 7: CFAR Engine Registers

Register	Width	Parameter-Set? (Y/N)	Description
CFAR_AVG_LEFT	6	Y	Number of left-side samples for noise level computation: This register is used to specify the number of samples used for noise level computation to the left of the cell under test. The number of samples used for noise level computation is equal to the value of this register multiplied by 2. For example, if this register value is 15, then the number of left- side samples used for averaging is 30. The maximum number that is possible is 126. A value of zero in this register means that the noise samples on the left side are not used for noise level computation. The valid values for this register are different for CFAR-CA and CFAR-OS modes: In CFAR-CA (and its variants CFAR-CAGO and CFAR-CASO), valid values for this register are 0, 2, 3, 4, …63 (Note that a value of 1 is not supported). This corresponds to number of samples equal to 0, 4, 6, 8, 10, 12, 14, … 124, or 126. In CFAR-OS mode, valid values for this register are restricted to 0, 4, 6, 8, 12, 16, 24, 32 only (which corresponds to number of samples equal to 0, 8, 12, 16, 24, 32, 48 or 64).

CFAR_AVG_RIGHT	6	Y	Number of right-side samples for noise level computation: This register is very similar to the above, except that this register specifies the averaging to the right of the cell under test. In most cases, it is expected that CFAR_AVG_RIGHT has the same value as CFAR_AVG_LEFT. In non-cyclic modes of CFAR, CFAR_AVG_RIGHT must be programmed equal to CFAR_AVG_LEFT.
CFAR_GUARD_INT	3	Y	Number of guard cells: This register specifies the number of guard cells to ignore on either side of the cell under test. If this register value is 3, then three guard cells on the left side and three guard cells on the right side are ignored. Only the noise samples beyond this guard region are used for calculating the surrounding noise level.
CFAR_OS_KVALUE	7	Y	K-th value for ordered statistic: This register is useful only in CFAR-OS mode, where it indicates the parameter K. From the sorted list of left and right noise samples, the K'th lowest value is used as the noise sample. This is a zero-based count – for instance, if this register value is 27, then the 28th lowest element in the sorted array is selected. Note that since CFAR-OS supports a maximum of 64 samples each on left and right side, the maximum size of the vector to sort is 128, and hence the maximum valid value of CFAR_OS_KVALUE register is 127.
CFAR_OS_NON_CYC_VARIANT_EN	1	Y	Enable scaling of K value for edge samples in non-cyclic CFAR-OS: This is useful only in CFAR-OS in non-cyclic mode. Setting this to 1 enables a variant where the K value used for noise sample selection for edge samples is scaled down proportional to the number of available neighboring samples at edges.
CFAR_THRESH	18	N	Threshold scale factor: This register is used to specify the threshold scale factor. This value is used to either multiply or add to the 'surrounding noise level' to determine the threshold used for detection of the present cell under test. If logarithmic CFAR mode is disabled (in magnitude or magnitude-squared mode), then the register value is multiplied with the surrounding noise level to determine the threshold, else it is added to the surrounding noise level. In the former case, this 18-bit register is interpreted as a 14.4 value. In the latter case (logarithmic mode), the 18-bit register is interpreted as a 7.11 value.
CFAR_DET_THR	24	N	Constant detection threshold value in constant threshold mode: This register is applicable only in constant threshold mode of CFAR (i.e. only in CFAR-CA mode and only if CFAR_AVG_LEFT = CFAR_AVG_RIGHT = 0). In this special mode, this register specifies the detection threshold value used to compare with cell under test. The detection threshold value is held constant, and scaled by CFAR_THRESH linearly or logarithmically
CFAR_LOG_MODE	1	Y	CFAR linear or logarithmic mode: This register is one of the registers used to specify whether the CFAR detector operates in linear or logarithmic mode. If this register bit is set, then the CFAR detector operates in logarithmic mode, which means that the threshold scale factor is added to (instead of multiplied with) the surrounding noise level value to determine the threshold. Note that this mode is meaningful only when the input samples to the CFAR detector are log- magnitude samples (see CFAR_INP_MODE as well). If this register bit is 0, then the logarithmic mode is disabled, in which case, the threshold scale factor is multiplied with (instead of added to) the surrounding noise level to determine the threshold. This mode is meaningful when magnitude or magnitude-squared samples are fed to the CFAR detector.
CFAR_INP_MODE	1	Y	CFAR engine input mode: This register bit specifies whether the inputs to the CFAR engine are complex samples or real values (the real values are already magnitude, magnitude-squared or log-magnitude numbers that can be directly sent to CFAR detection process). If this register bit is 1, then the input samples are real values and are directly sent to CFAR detection. If this register bit is 0, then the inputs are complex samples and hence either magnitude or magnitude- squared or log-magnitude computation is required prior to CFAR detection. Which of the three, viz., magnitude or magnitude-squared or log-magnitude is done, is selected by CFAR_ABS_MODE register described below.

CFAR_ABS_MODE	2	Y	CFAR magnitude, mag-squared or log-mag mode: This register is used to specify which of the three computations, namely Magnitude, Mag- squared or Log-Magnitude, is enabled inside the CFAR engine prior to CFAR detection. This register is only relevant when CFAR_INP_MODE is 0 (complex samples are fed to CFAR engine). 00b – Magnitude-squared 01b – Not valid 10b – Magnitude (using JPL approximation) 11b – Log2-Magnitude (using LUT approximation)
CFAR_OUT_MODE	2	Y	CFAR engine output mode: The MSB bit of this register selects whether the CFAR Engine outputs all the noise average values for all the cells ('Raw output' mode), or whether the CFAR Engine outputs only the detected peaks ('Detected Peaks List' mode). The LSB bit specifies the content of the 24-bit 'I' and 'Q' channel outputs logged in destination memory. If CFAR_ADV_OUT_MODE is set to 1 (special mode called "Dominant peaks"), this register should be set to 1. Refer main description section for details.
CFAR_ADV_OUT_MODE	1	Y	CFAR engine special output mode (Dominant peaks mode): This register bit enables the special "Dominant peaks" mode of the CFAR engine. In this mode, the CFAR engine outputs the input array (I and Q) samples corresponding detected peak locations as is and suppresses the non-detected peak locations (sends zeros).
CFAR_GROUPING_EN	1	Y	CFAR peak grouping enable: This register bit specifies whether peak grouping should be enabled. When this register bit is 0, peak grouping is disabled, which means that a peak is declared as detected as long as the cell under test exceeds the threshold. On the other hand, if this register bit is 1, then a peak is declared as detected only if it the cell under test exceeds the threshold, as well as, if the cell under test exceeds the two neighboring cells to its immediate left and right (local maximum).
CFAR_NOISE_DIV	4	Y	CFAR noise average division factor: This parameter is applicable only in CFAR-CA modes and it is not applicable in CFAR-OS mode. This register specifies the division factor with which the noise sum calculated from the left and right noise windows are divided, in order to get the final surrounding noise average value. The division factor is equal to 2^ CFAR_NOISE_DIV. Therefore, only powers-of-2 division are possible, even though the number of samples specified in CFAR_AVG_LEFT and CFAR_AVG_RIGHT are not restricted to powers of 2. The surrounding noise average value obtained after the division is multiplied or added with CFAR_THRESH to determine the final threshold used to compare the cell under test for detection. The maximum allowed value for this register is 8, which gives a division factor of 256.
CFAR_CA_MODE	2	Y	CFAR noise averaging mode: This register configures the noise averaging mode in the CFAR detector from one of these options – CFAR-CA, CFAR-CAGO, CFAR-CASO, CFAR-OS. 00b – CFAR-CA 01b – CFAR-CAGO 10b – CFAR-CASO 11b – CFAR-OS
CFAR_CYCLIC	1	Y	CFAR cyclic vs. non-cyclic mode: This register bit specifies whether the CFAR detector needs to work in cyclic mode or in non-cyclic mode. When this register bit is 0, the CFAR detector works in non-cyclic mode and when it is 1, it works in cyclic mode. Refer main description section for details on how to configure and use cyclic mode.

CFAR_PEAKCNT (read-only)	12	N	CFAR detected peak count: This is a read-only register that contains the number of detected peaks that are logged in the destination memory, when CFAR Engine is configured in 'Detected Peaks List' mode. In the Detected Peaks List mode, since only the detected peaks are logged in the destination memory, this read-only register provides the number of detected peaks that are logged to the main processor, so that the main processor can determine how many entries to read from the destination memory.

3 Core Computational Unit – Statistics

This section describes the statistics block present in the core computational unit.

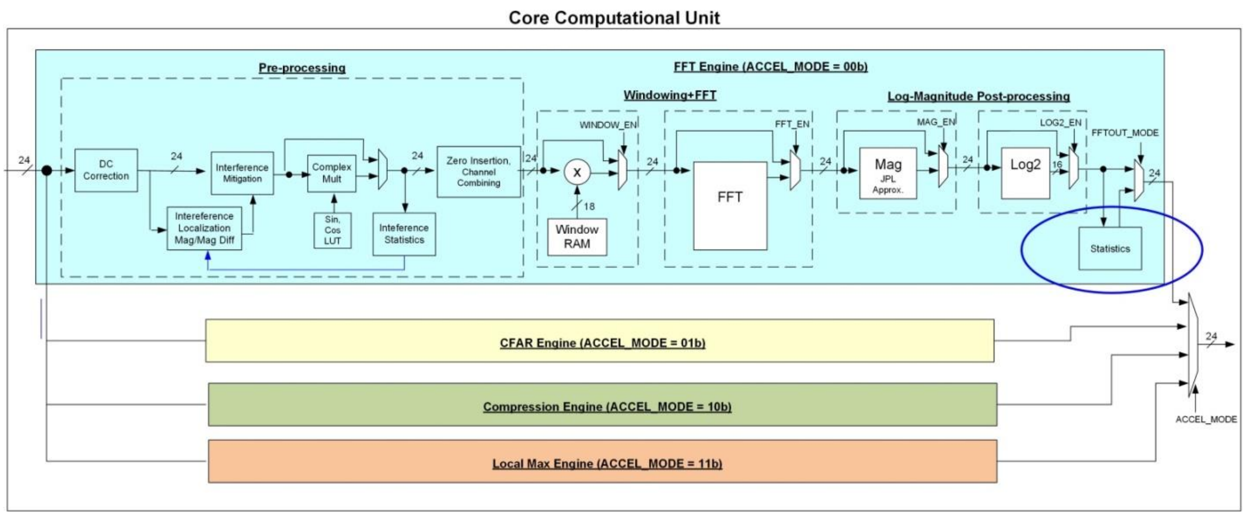

Core Computational Unit
Pre-processing
FFT Engine (ACCEL_MODE = 00b)
Windowing+FFT
Log-Magnitude Post-processing
WINDOW_EN
FFT_EN
MAG_EN
LOG2_EN
FFTOUT_MODE
24
DC Correction
Interference Mitigation
Complex Mult
Zero Insertion, Channel Combining
Intereference Localization Mag/Mag Diff
Sin, Cos LUT
Inteference Statistics
Window RAM
18
FFT
Mag JPL Approx.
Log2
16
Statistics
CFAR Engine (ACCEL_MODE = 01b)
Compression Engine (ACCEL_MODE = 10b)
Local Max Engine (ACCEL_MODE = 11b)
ACCEL_MODE

Figure 21: Statistics Block

3.1 Statistics Block

The core computational unit has a statistics computation block at the end of the FFT Engine path as shown in Figure 21. It can be used to compute a few simple statistics of the samples output by the core computational unit. It supports computing statistics on 1- and 2-dimensional array inputs.

With 1-dimensional array input, it can compute sum and maximum of samples. It also supports a few advanced features, with 2-dimensional array inputs. It can compute 2 arrays of maxima (one in each dimension) and multiple histograms or cumulative distribution functions (CDFs) of the input samples (one histogram for each sample index).

The simple basic mode of operation is described first. The advanced features are described in separate later sub-sections.

3.1.1 Basic Statistics Block – Operation

The 24-bit I and 24-bit Q output of the core computational unit goes to a statistics computation block. The purpose of this block is to find the maximum and sum (average) of the output samples

The sum and max statistics are computed on a 'per-iteration' basis (the sum and max values are logged at the end of each iteration) and the computation is reset for the next iteration. The sum and max values are logged in register-sets (see MAXn_VALUE, MAXn_INDEX, ISUMn, QSUMn register-sets), which can be read by the main processor. However, only four such registers are provided for each statistic and therefore, the sum and max values can be logged in these registers only for up to a maximum of four iterations.

The max statistics register-set comprises four read-only registers of 24 bits each, named MAXn_VALUE, for recording max values, and four read-only registers each 12 bits unsigned, named MAXn_INDEX, for recording the max indices. The sum statistics register-set contains four registers of 36 bits each, named ISUMn, for I-sum statistics, and 4 registers of 36 bits each, named QSUMn, for Q-sum statistics.

For larger number (>4) of iterations, either the sum or the max value can be sent to the destination memory for each iteration, which allows the statistic to be available even for cases with more than four iterations. The logging of the statistic into the destination memory is enabled using FFT_OUTPUT_MODE register described below.

The MSB bit of the FFT_OUTPUT_MODE (Table 8) register selects whether the default (main) output of the core computational unit goes to the destination memory, or the statistics block output. If the MSB of this 2-bit register is 0, then it selects the default mode of operation, where the main output (FFT or Log-Mag result) is sent to the destination memory. If the MSB is 1, then it selects the statistics output mode, where either the sum or max statistic is sent to the destination memory (one value per iteration). Whether the sum or max is sent to memory is dependent on the LSB bit. If the LSB bit is 0, then the statistic value that is sent is the max value (useful in conjunction with Log-Mag enabled to find the biggest peak and peak index per iteration). Here, the I output is the maximum value itself and the Q output is the index (location) of the maximum value. If the LSB bit is 1, then the statistic value that is sent is the sum value (useful for DFT mode, as well as for mean squared or mean of absolute values computation).

Table 8: Statistics Output Modes

FFT_OUTPUT_MODE Register	I Channel Output	Q Channel Output
00b – Default output mode	Main output of core computational unit	
10b – Max statistics output (One output per iteration)	Max Value	Max Index
11b – Sum statistics output (One output per iteration)	Sum of I values	Sum of Q values

The max statistic records the maximum value (and its index) of the magnitude or log-magnitude samples corresponding to every iteration. The sum statistic records the sum of the magnitude or log-magnitude or the complex output samples corresponding to each iteration. If the main output of the core computational unit is the complex FFT output (ABS EN=0 and LOG2EN=0), then the sum statistics is the complex sum.

The complex sum statistics mode is useful when used in conjunction with the complex multiplier block in DFT mode or vector multiplication mode. For example, the sum statistic computed here, together with the DFT mode of the complex multiplier block, enable DFT computation for the desired number of bins (iterations). When the desired number of bins is more than 4, the sum statistic can be sent to destination memory (instead of the main data output that is normally sent to the destination memory).

Note that when the sum statistics is logged into the destination memory, it goes through the Output Formatter block as only 24-bits each for I and Q (same bit-width as the primary FFT outputs). Hence, the computed sum statistics value of 36-bits width, needs to be scaled down by right-shifting the appropriate number of LSBs (using FFTSUMDIV register) before sending to output formatter. Thus, when logging the statistics in destination memory, the sum statistics is to be used as an “average” value, rather than a “sum” value itself.

The FFTSUMDIV register specifies the number of bits to right-shift the sum statistic before it is written to destination memory. The internal sum statistic register is 36-bits wide (allowing 12 bits of MSB growth of the 24-bit data path), but this statistics value needs to be scaled down to 24 bits to match the data path width going to the Output Formatter. This register specifies how many LSBs to drop to convert the sum statistics to 24-bit value. Note that only signed saturation is implemented (irrespective of whether magnitude values are being summed or complex FFT output values are being summed). Therefore, it is recommended that this register is configured to drop an appropriate number of LSBs such that incorrect saturation in case of magnitude sum is avoided.

Note that in statistics output mode, the registers DSTACNT, DSTAINDX, DSTBINDX, DST16b32b and DSTREAL are not meant to be used, since it is known that there is only one value to be written to destination memory for every iteration in a specific format. It is recommended that in this mode, DSTACNT be programmed to a value of 0, DSTAINDX and DSTBINDX are both programmed to a value of 8 bytes, DST16b32b is set to 1 and DSTREAL is reset to 0. The statistics is then always logged in the destination memory as consecutive 32-bit I and Q samples, irrespective of whether sum statistic or max statistic is being logged.

3.1.2 Statistics block – Register Descriptions

Table 9 lists all the registers of the statistics block.

Table 9: Statistics Block Registers

Register	Width	Parameter-Set? (Y/N)	Description
MAX1_VALUE MAX2_VALUE MAX3_VALUE MAX4_VALUE	24	N	Max value: These registers contain the max value on a per-iteration basis. These registers are meaningful only when Magnitude or Log-Magnitude is enabled. Only the max values for up to four iterations are recorded in these registers. For larger number of iterations, use statistics output mode (FFT_OUTPUT_MODE below).
MAX1_INDEX MAX2_INDEX MAX3_INDEX MAX4_INDEX	12	N	Max index: These registers contain the max index on a per-iteration basis, corresponding to each max value in the MAXn_VALUE registers.

I_SUM1_LSB, I_SUM1_MSB I_SUM2_LSB, I_SUM2_MSB I_SUM3_LSB, I_SUM3_MSB I_SUM4_LSB, I_SUM4_MSB Q_SUM1_LSB, Q_SUM1_MSB Q_SUM2_LSB, Q_SUM2_MSB Q_SUM3_LSB, Q_SUM3_MSB Q_SUM4_LSB, Q_SUM4_MSB			Sum statistics: These registers contain the sum of the I outputs and Q outputs on a per-iteration basis. Only the statistics for up to four iterations are recorded in these registers. For larger number of iterations, use statistics output mode (FFT_OUTPUT_MODE below).
FFT_OUTPUT_ MODE	2	Y	FFT Path output mode: This register specifies the output mode of the FFT path. Instead of the default mode where the main output of the core computational unit is sent to the destination memory, this register can be configured such that either the max or sum statistics can be sent to the destination memory. 00b – Default mode (main output) 10b – Max statistics output mode 11b – Sum statistics output mode
FFTSUMDIV	5	N	Right-shifting for Sum statistic: This register specifies the number of bits to right-shift the sum statistic before it is written to destination memory. The internal sum statistic register is 36-bits wide (allowing 12 bits of MSB growth of the 24-bit data path), but this statistics value needs to be scaled down to 24 bits to match the data path width going to the Output Formatter. This register specifies how many LSBs to drop to convert the sum statistics to 24-bit value.

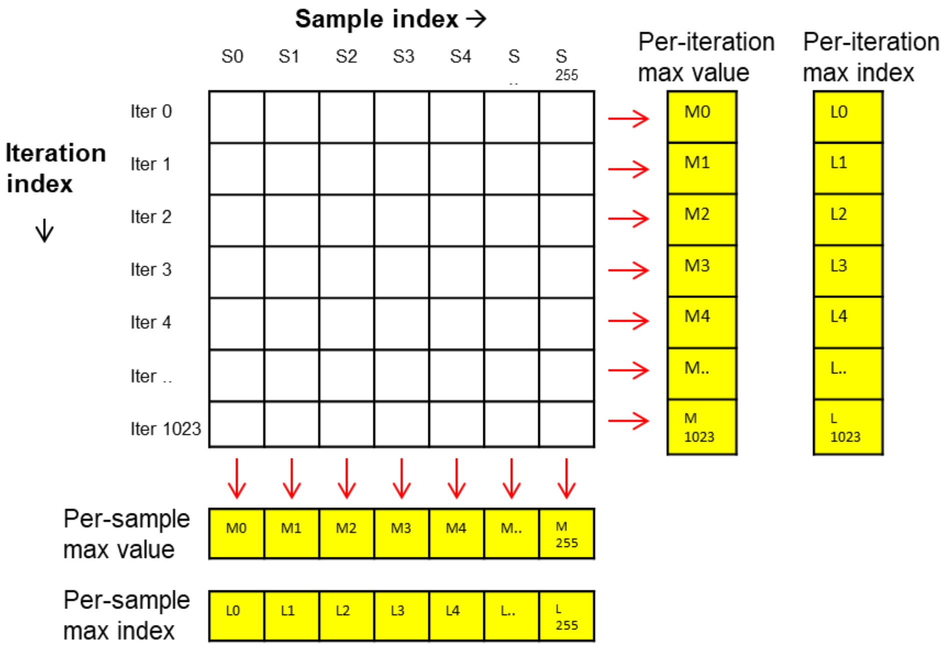
3.1.3 Advanced Statistics – 2-Dimensional Maxima

The Statistics block can be used to compute 2 arrays (one in each dimension) of maxima on the input samples. This feature can be enabled by setting the register, MAX2D_EN to 1.

For each iteration, the maximum value observed and the corresponding sample index are stored in per-iteration RAMs. Similarly, for each sample index, the maximum value observed over all the iterations and the corresponding iteration index are stored in per-sample RAMs. Thus, as shown in Figure 22 the results include (value, location) pairs stored on a per-iteration basis and per-sample index basis. These RAMs are a part of the Advanced Statisitics Memories. This feature supports SRCACNT (samples) in the range of 1 to 255 and BCNT (iteration) in the range of 0 to 1023.

The processor as well as the local maxima computation engine can access the Advanced Statistics Memories. This allows the 2D maxima to be used as thresholds in the local maxima computation steps.

Local Maxima peak detector can be configured to use the max value RAMs as arrays of thresholds. The max value and max index RAMs can also be read and modified by the CPU/DSP. This allows flexible and application specific control on the peak detection thresholds. The data read/write format for the 2Dmax output RAMs are described in the table



of Advanced Statistics Memories (Table 13).

Figure 22: 2D Max Advanced Statistics Computation

3.1.4 *Advanced Statistics – Histograms, CDF and CDF Count Threshold*

The Statistics block can compute the histograms or CDFs of the input samples. The statistics block can also be used to calculate a detection threshold based on the CDF.

The histogram function operates on a 2-D matrix of input samples. The ACNT (or 2^FFT_SIZE) number of samples received in a single iteration constitute the "sample" dimension. BCNT+1 iterations of the sample dimension data are received, which constitutes the "iteration" dimension. One histogram is computed for each sample index using values received across all iterations corresponding to that sample index as shown in figure below.

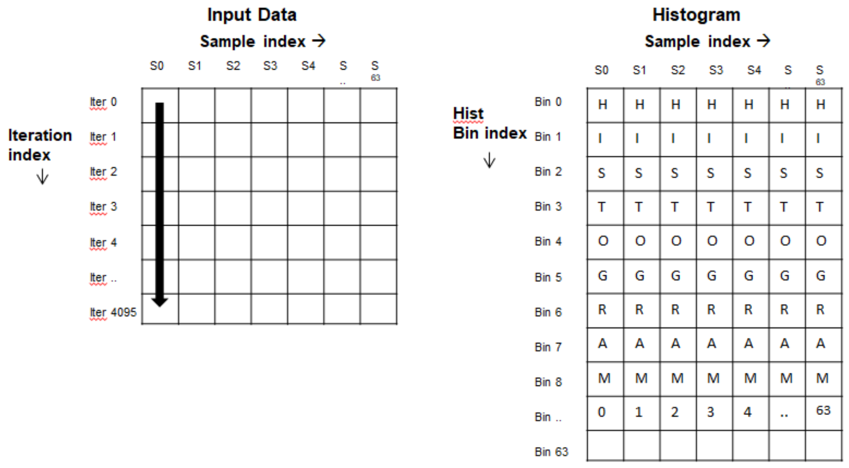


Figure 23: Advanced statistics histogram computation

The results are stored in corresponding RAMs which are part of the Advanced Statistics Memories. The register HIST_MODE can be used to enable this feature and select the exact operation mode as mentioned in the table below. MEM_INIT_START bit 13 & 14 should be set to start the Histogram Memory initialization. MEM_INIT_DONE bit 13 & 14 indicates the histogram Memory initialization completion

Table 10: Histogram operating modes

HIST_MODE	Histogram Statistics Computation
00	Disabled: In this mode, the histogram computation is totally bypassed and the Histogram RAM is retained in its existing state.
01	Histogram computation mode: In this mode, histograms are computed, one histogram per sample index. The histograms are stored in the Histogram RAM.
10	CDF count computation mode: In this mode, CDF counts are computed, one CDF per sample index. The CDF count range (0, number of iterations programmed) maps to the CDF range (0, 1). The CDFs are stored in the Histogram RAM. The CDF count computation needs additional computation time over and above the histogram computation mode. The number of histogram bins times the number of histograms is the number of additional cycles..
11	CDF threshold computation mode: In this mode, histograms are computed similar to the histogram computation mode. Additionally, for the purpose of computing a detection threshold, information about the histogram bin at which the corresponding CDF would just exceed the programmed register, CDF_CNT_THRESH, is calculated and reported through CDF_CNT_BINNUM. The corresponding CDF and histogram values at that bin are reported through CDF_CNT_CDFVAL and CDF_CNT_HISTVAL.

The histograms are computed on a per-sample-index basis. Up to 64 histograms can be computed. The valid range of SRCACNT for this feature is 1 to 63. The histogram size (number of bins) is programmable (from 8 to 64, in powers of 2) using the register HIST_SIZE_SEL. The bins are uniformly spaced. It should be noted that histogram resolution is optimal if input is in log-magnitude mode rather than linear-magnitude. Since the log-magnitude mode gives outputs with high precision (in 5.11u format, i.e. 5 integer bits, 11 fractional bits, unsigned), they can be scaled down through a programmable right shift before histogram computation, using the register HIST_SCALE_SEL. Valid values for HIST_SCALE_SEL are 7, 8, 9, 10, 11, 12 and 13. The HIST_SIZE_SEL and the HIST_SCALE_SEL registers are explained in the Figure 24 below

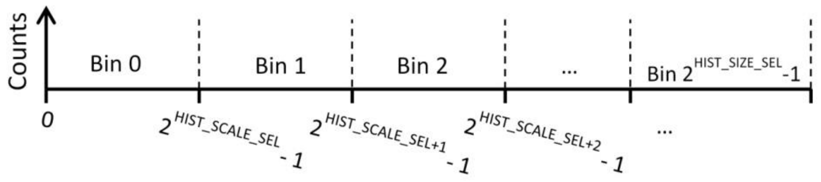

Counts
Bin 0
Bin 1
Bin 2
...
Bin $2^{HIST\_SIZE\_SEL}-1$
0
$2^{HIST\_SCALE\_SEL} - 1$
$2^{HIST\_SCALE\_SEL+1} - 1$
$2^{HIST\_SCALE\_SEL+2} - 1$
...

Figure 24: HIST_SCALE_SEL and HIST_SCALE_VAL configuration

It should be noted that the input samples that exceed the highest bin are counted in the histogram's highest bin index. Also, if any bin's count (i.e. the number of occurrences) exceeds the bit-width allocated in the histogram output memory, then the count is clipped to the maximum level.

As an illustration, if SRCACNT = 31, BCNT = 767, HIST_SIZE_SEL = 4, HIST_SCALE_SEL = 10, and HIST_MODE = 0b01, then 32 histograms, each with 16 bins are computed and stored in the Histogram RAM. The bins correspond to the input ranges, [0 to 1023], [1024 to 2047], …, [14*1024 to 15*1024 -1], [>/= 15*1024]. In each of the 32 histograms, the sum of all bin values will match 768. This example assumes that the pre-processing block's channel combining and zero insertion features are not enabled. If they are enabled, then the number of histograms will match the resultant number of samples in A-dimension after the channel combining and zero insertion operations.

The contents of the Histogram RAM can be read by the CPU/DSP and may be utilized to update the threshold arrays for Local Maxima peak detection. The Histogram RAM can be read with the format explained in table of the Advanced Statistics Memories (Table 13).

In HIST_MODE = 0b11, the CDF_CNT_BINNUM (bin number where CDF just exceeds the programmed CDF_CNT_THRESH) for each sample is stored along with the corresponding CDF_CNT_CDFVAL and CDF_CNT_HISTVAL. These values are stored in the CDF Threshold RAM which can be read with the format as explained in the Advanced Statistics Memories (Table 13).

3.1.5 Advanced Statistics block – Register Descriptions

Table 11 lists all the registers of the statistics block.

Table 11: Advanced Statistics – Control Registers

Register	Width	Parameter Set	Description
MAX2D_EN	1	Yes	2-Dimensional Maxima Computation Enable: If this bit is set to 1, one maxima (and corresponding index) array for A dimension and another for B dimension are computed and stored in the per-sample and per-iteration RAMs.
HIST_MODE	2	Yes	Histogram Computation Mode: 00: Disable 01: Histogram computation mode 10: CDF computation mode 11: CDF threshold computation mode Refer to the earlier section for details.
HIST_SCALE_SEL	4	Yes	Histogram Input Scale Select: This register is used to select the input scaling before histogram computation. It provides a way to control the resolution of the histogram bins by right-shifting the input sample values before histogram computation. If this register is set to X (and Y=2^X), then the histogram bin ranges are [0, Y-1], [Y, 2Y–1] and so on. Valid values for this register are X=7, 8, 9, 10, 11, 12, and 13.
HIST_SIZE_SEL	4	Yes	Histogram Size Select: This register is used to select the histogram size (number of bins). The size is configurable in powers of 2. The number of histogram bins is 2^HIST_SIZE_SEL. Valid values of this register are 3, 4, 5, and 6. They correspond to histogram sizes of 8, 16, 32, and 64.
CDF_CNT_THRESH	12	No	This register is applicable in CDF count threshold mode of operation. CDF is computed over the histogram till the value of the CDF just exceeds the CDF_CNT_THRESH specified by the user. This register can take values from 0 to BCNT. CDF count threshold value is same for all samples indices (histograms).

MEM_INIT_START	32	No	Writing 1'b1 to bit locations – 13 & 14 would start the memory initialization for the Histogram Memory. These are self-clearing bits Bit 13 : MEM_INIT_START_HIST_ODD_RAM Bit 14 : MEM_INIT_START_HIST_EVEN_RAM
MEM_INIT_DONE	32	No	When the memory initialization is complete, then bit locations 13 & 14 shall be 1b'1 Bit 13 : MEM_INIT_DONE_HIST_ODD_RAM Bit 14 : MEM_INIT_DONE_HIST_EVEN_RAM

Table 12 lists the contents of the Advanced Statistics Memory.

Table 12: Contents of Advanced Statistics Memory output

RAM content	Width	Description
MAXVAL_ARRAY_DIM1 [1024]	24 bits each	2-D Maxima Array – maximum values of each iteration: The maximum value across samples in each iteration is recorded here (one value per iteration, first address corresponding to first iteration).
MAXLOC_ARRAY_DIM1 [1024]	10 bits each	2-D Maxima Array – maximum locations corresponding to each iteration: The sample index at which the maximum value occurred in each iteration is recorded here (one value per iteration, first address corresponding to first iteration).
MAXVAL_ARRAY_DIM2 [256]	24 bits each	2-D Maxima Array – maximum values corresponding to each sample index: The maximum value across the iterations for each sample index is recorded here (one value per sample index, first address corresponding to first sample index).
MAXLOC_ARRAY_DIM2 [256]	10 bits each	2-D Maxima Array – maximum locations corresponding to each sample:: The iteration count at which the maximum value occurred corresponding to each sample is recorded here (one value per sample index, first address corresponding to first sample index).
HIST_OUT_ARRAY [64][64]	12 bits each	Histogram Output Corresponding to each Sample Index: The number of occurrences in each histogram bin is recorded here for different sample indices.
CDF_CNT_BINNUM [64]	6 bits each	If CDF count threshold mode is enabled: the bin number at which the specified count (CDF_CNT_THRESH) is just exceeded is stored in this register for all the sample indices.
CDF_CNT_CDFVAL [64]	12 bits each	Valid when CDF count threshold mode is enabled. This stores the CDF count at CDF_CNT_BINNUM bin for all the sample indices.
CDF_CNT_HISTVAL [64]	12 bits each	Valid when CDF count threshold mode is enabled. This stores the Histogram count at CDF_CNT_BINNUM bin for all the sample indices.

Table 13 lists the read/write format for Advanced Statistics Memories

Table 13: Advanced Statistics Memories Format

RAM	Reg no.	Data Format	Description
Per-iteration Max Value	0	0:23 MAX_VAL_ITER_0 24:31 NU	The maximum value across samples in each iteration is recorded here
	:	:	
	1023	0:23 MAX_VAL_ITER_1023 24:31 NU	
Per-sample Max Value	0	0:23 MAX_VAL_SAMPLE_0 24:31 NU	The maximum value across the iterations for each sample index is recorded here
	:	:	
	255	0:23 MAX_VAL_SAMPLE_255 24:31 NU	
Per-iteration Max Index	0	0:9 MAX_INDEX_ITER_0 10:15 NU 16:25 MAX_INDEX_ITER_1 26:31 NU	The sample index at which the maximum value occurred in each iteration is recorded here
	:	:	
	511	0:9 MAX_INDEX_ITER_1022 10:15 NU 16:25 MAX_INDEX_ITER_1023 26:31 NU	
Per-sample Max Index	0	0:9 MAX_INDEX_SAMPLE_0 10:15 NU 16:25 MAX_INDEX_SAMPLE_1 26:31 NU	The iteration count at which the maximum value occurred corresponding to each sample is recorded here
	:	:	
	127	0:9 MAX_INDEX_SAMPLE_254 10:15 NU 16:25 MAX_INDEX_SAMPLE_256 26:31 NU	
Histogram	[0][0]	0:11 HIST_BIN_0_SAMPLE_0 12:15 NU 16:27 HIST_BIN_1_SAMPLE_0 28:31 NU	Histogram output is stored when the HIST_MODE = 0b01 or 0b11. CDF output is stored when the HIST_MODE = 0b10. All the bins of the first sample are stored first. Followed by all the bins of the subsequent samples.
	:	:	
	[32][0]	0:11 HIST_BIN_62_SAMPLE_0 12:15 NU 16:27 HIST_BIN_63_SAMPLE_0 28:31 NU	
	[0][1]	0:11 HIST_BIN_0_SAMPLE_1 12:15 NU 16:27 HIST_BIN_1_SAMPLE_1 28:31 NU	
	:	:	
	:	:	
	:	:	
	:	:	
	[32][63]	0:11 HIST_BIN_62_SAMPLE_63 12:15 NU 16:27 HIST_BIN_63_SAMPLE_63 28:31 NU	
CDF Threshold	0	0:11 CDF_CNT_HISTVAL_SAMPLE_0 12:23 CDF_CNT_CDFVAL_SAMPLE_0 24:29 CDF_CNT_BINNUM_SAMPLE_0 30:31 NU	When HIST_MODE = 0b11, the bin number where the CDF_CNT_THRESH is crossed is stored as CDF_CNT_BINNUM. It is stored in the same register combined with the
	:	:	

	63	0:11 CDF_CNT_HISTVAL_SAMPLE_63 12:23 CDF_CNT_CDFVAL_SAMPLE_63 24:29 CDF_CNT_BINNUM_SAMPLE_63 30:31 NU	CDF_CNT_CDFVAL and CDF_CNT_HISTVAL which store the cdf count and the histogram count in that bin respectively. These registers are stored for all the sample separately.

4 Core Computational Unit – Local Maxima Engine

The Core Computation Unit includes a Local Maxima Engine, which can be enabled by setting ACCEL_MODE = 011b.

In context of Radar signal processing, Local Maxima Engine is useful in a peak detection step where each sample/bin (Cell Under Test (CUT)) is compared against detection threshold(s), and also compared against the neighboring samples and the CUT is declared as a valid local peak if the sample amplitude exceeds the detection threshold and is "more than or equal to" the neighboring cells. Local maxima computations are done on a 2D matrix. Typically, local maxima are computed after the Angle FFT, on the Doppler-Beam/angle "2D" plane (2D Local Maxima).

The output of the local maxima computations is stored into the destination memory as a bit pattern, where each bit indicates whether the specific sample/CUT was detected as a valid local peak or not. More details on this are given in the later section of the document. The detection thresholds can either be configured into the configuration registers or in the Advanced Statistics Memory (borrowed from the Statistics module), in case there are individual thresholds for the different rows and columns of the 2D matrix.

4.1 Local Maxima Engine – Operation

Figure 25 below shows an example of 2D plane (eg. Dopplers-Beams plane) used for Local maxima computations. A 3x3 matrix (CUT and all its neighboring samples) is needed to compute the local maxima. To keep the scheme simple and efficient from memory access perspective (as the input samples for local maxima computations are fetched from the memory), a 3x4 matrix (shown as red box in Figure) is fetched.

Every cycle, four consecutive samples (16-bit x 4 = 64bits) are picked from the memory and this is repeated three times across three row vectors (doppler bins) to fetch the 3x4 matrix data. This box keeps on moving to the right (computation wise) till the last group of four samples are fetched. By the end of this process, we get a bit-pattern of size (bits) equivalent to the number of columns (i.e., beams), with each bit indicating whether the specific bin was a local peak or not. This scheme is repeated for each doppler vector to get the bit-pattern output for the complete 2D plane.

Samples are fetched using the three-dimension addressing scheme built-in to the Input Formatter (the dimensions denoted by A, B and C) by configuring the configuration registers accordingly. For Local Maxima operation, there are restrictions on the maximum sizes of these dimensions. The number of columns (achieved through dimension B) has to be one of these supported values: 8, 12, 16, 24, 32, 48, 64, 96, 128, 192, 256. The number of rows (achieved through dimension C) can be any value between 3 and 1024.

Typically, for CUT processing at the edges, wrap-around around these dimensions would be required (cyclic mode of operation). This is achieved using the configuration registers SRCB_CIRCSHIFTWRAP, SRC_CIRCSHIFTWRAP3X and WRAP_COMB. If non-cyclic mode of operation is desired (i.e., for the edge samples, the neighboring cells should not be wrapped around), then LM_DIMB_NONCYCLIC and LM_DIMC_NONCYCLIC register bits can be set. In this case, the unavailable neighbouring samples will be considered as masked out.

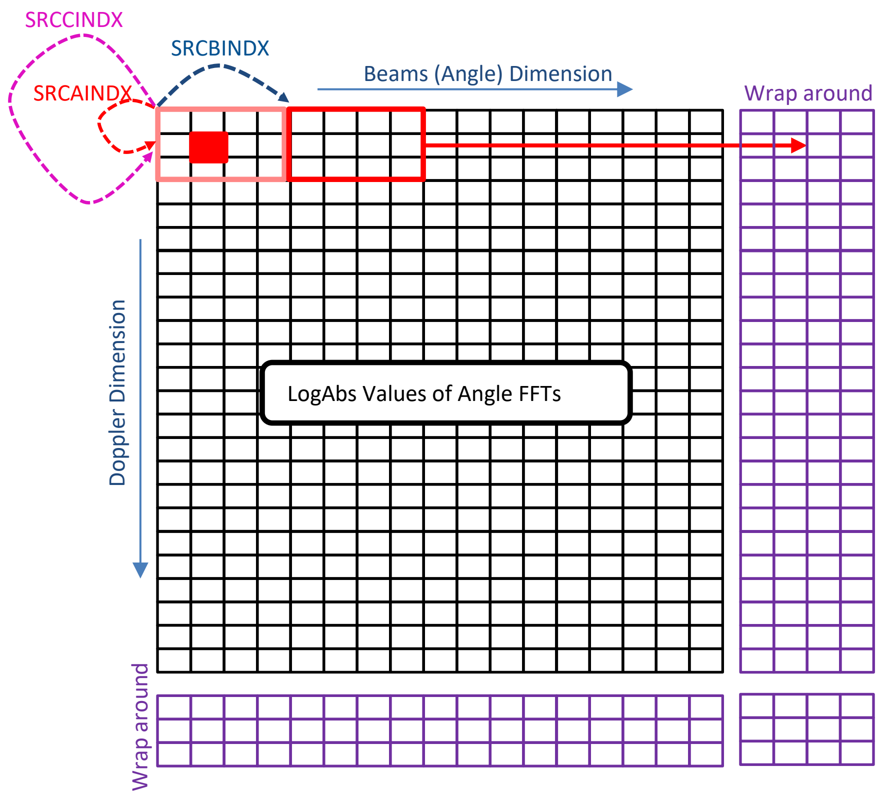
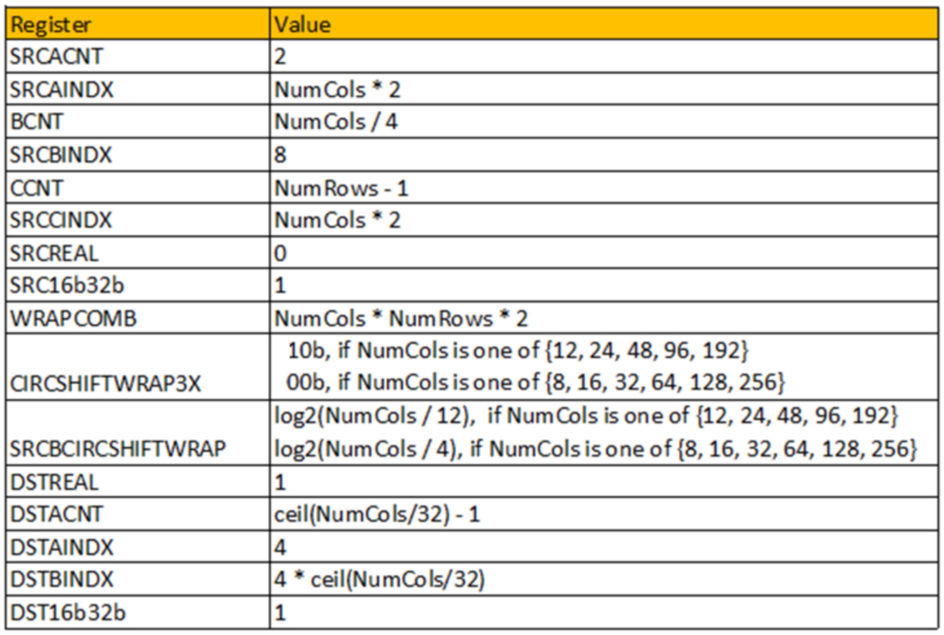


Figure 25: Local Maxima Engine A, B and C-dimension usage

Below is the example of the configuration values of each of these registers accessing the samples from the source memory for Local maxima computations for a 256x12 matrix. Note that though SRCACNT and SRCBINDX values are part of configuration registers, for the local maxima functionality, the values shall always be fixed to the values of 2 and 8 respectively, as shown below in Figure 26. The other registers are calculated easily as a function of the number of beams (number of columns) and number of Doppler (number of rows) as shown in the figure. Note that these configurations are irrespective of whether cyclic or non-cyclic mode of operation. It is important that in Local Maxima Engine mode of operation, the registers of Input Formatter and Output Formatter listed in figure below are programmed exactly as per the calculations shown.

Figure 26: Local Maxima Engine Configuration



Register	Value
SRCACNT	2
SRCAINDX	NumCols * 2
BCNT	NumCols / 4
SRCBINDX	8
CCNT	NumRows - 1
SRCCINDX	NumCols * 2
SRCREAL	0
SRC16b32b	1
WRAPCOMB	NumCols * NumRows * 2
CIRCSHIFTWRAP3X	10b, if NumCols is one of {12, 24, 48, 96, 192} 00b, if NumCols is one of {8, 16, 32, 64, 128, 256}
SRCBCIRCSHIFTWRAP	log2(NumCols / 12), if NumCols is one of {12, 24, 48, 96, 192} log2(NumCols / 4), if NumCols is one of {8, 16, 32, 64, 128, 256}
DSTREAL	1
DSTACNT	ceil(NumCols/32) - 1
DSTAINDX	4
DSTBINDX	4 * ceil(NumCols/32)
DST16b32b	1

4.2 *Local Maxima Engine – Operating mode Configurations*

Local maxima computations involve comparing each CUT with several values. The values that can be compared with are the 8 neighboring (adjacent right/left/top/bottom/diagonal) samples in the 2D matrix, row-threshold and a column-threshold. Each of these comparisons can be enabled or disabled through the configuration registers LM_NEIGH_BITMASK (8-bit) and LM_THRESH_BITMASK (2-bit).

The row- and column-thresholds can be selected from software configurable registers DIMB_THRESH_VALUE and DIMC_THRESH_VALUE or Maxima Arrays from Advanced Statistics RAM. This selection can be made through the register LM_THRESH_MODE. Further, if the Maxima Arrays from Advanced Statistics RAM are selected for determining thresholds, then the Local Maxima engine has provision for adding 24-bit signed offsets (one in each dimension). This can be useful in log-mode of operation, to convert the maxima arrays to row- and column-thresholds for peak detection. The offsets are programmable through the registers, MAX2D_OFFSET_DIM1 and MAX2D_OFFSET_DIM2. As these offsets are *added* to the Maxima Arrays from Advanced Statistics RAM to derive the detection thresholds, typically, these registers are expected to be set to negative values.

Figure 27 and Figure 28 illustrate the thresholding and comparisons in Local Maxima. The result of the Local Maxima comparisons is a 2D bit pattern. Each bit in it indicates whether the corresponding (row, column) CUT either exceeded *or equaled* each of the values it was compared against. A value of 0 in the result bit indicates that at least one of these values exceeded the CUT and 1 indicates otherwise.

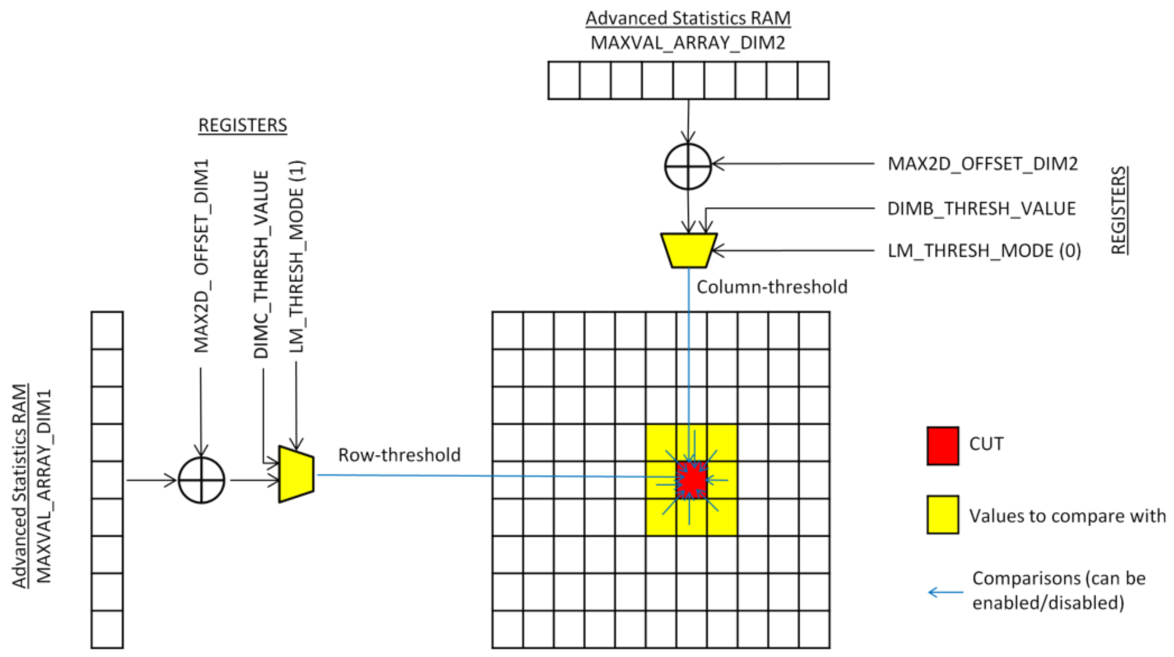


Figure 27: Local Maxima – Example Operating Mode Configurations

LM_NEIGH_BITMASK (0)	LM_NEIGH_BITMASK (1)	LM_NEIGH_BITMASK (2)
LM_NEIGH_BITMASK (7)	CUT	LM_NEIGH_BITMASK (3)
LM_NEIGH_BITMASK (6)	LM_NEIGH_BITMASK (5)	LM_NEIGH_BITMASK (4)

Figure 28: Local Maxima – Bitmask configuration

4.1 Local Maxima Engine – Output Write Pattern

The output of the local maxima computations is first shifted to a local shift register of 256 bits. The output is pushed on to the local memory in HWA only once the computations worth one full row is completed. So, 256 puts a restriction on the maximum number of bins can be used in a particular row.

The output bit pattern is stored in the destination memory packed as 32-bit words, with the LSB bit corresponding to B-dimension (column) count of 0. In case when the output of a vector is not a multiple of 32 bits, the remaining bits in the last 32-bit word would be irrelevant bits and shall be ignored (set to ‘0’). If the number of bits per Doppler row is less than 32, for example 16, then 16 LSB of a 32b word are populated with bitmask and the remaining 16b are set to ‘0’.

This mechanism is described in the below diagrams Figure 29 and Figure 30.

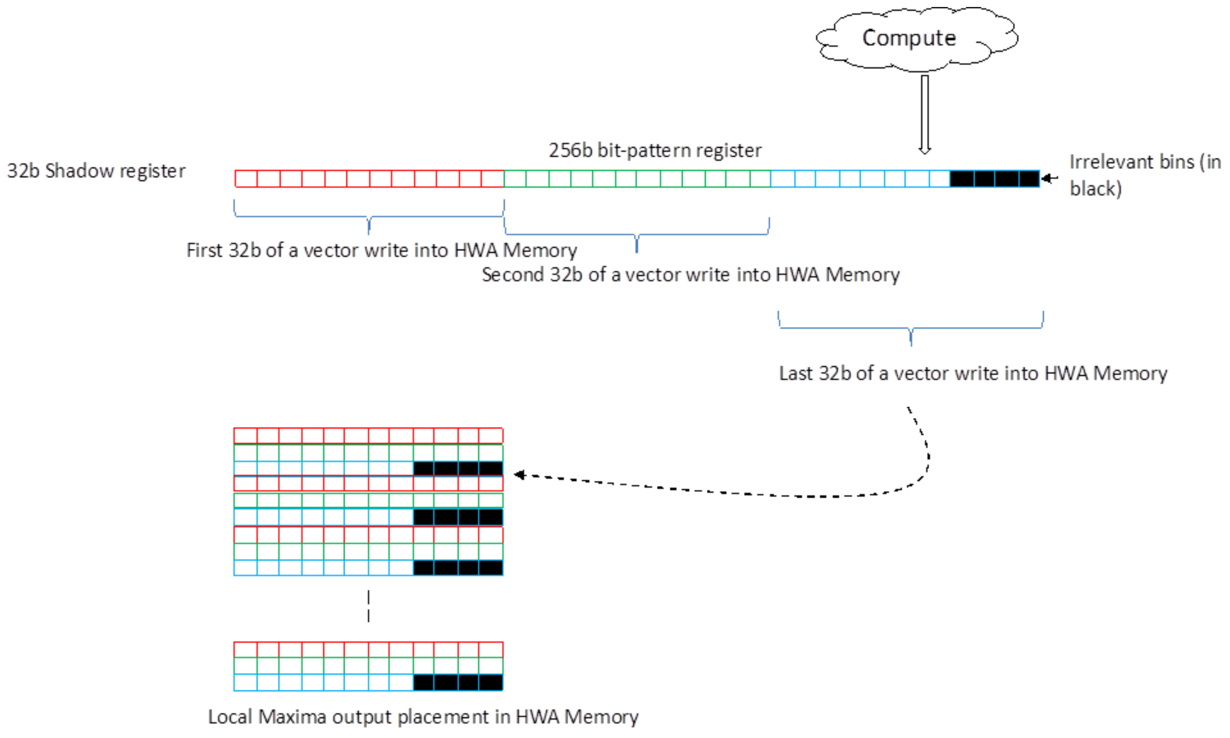

Compute
256b bit-pattern register
32b Shadow register
Irrelevant bins (in black)
First 32b of a vector write into HWA Memory
Second 32b of a vector write into HWA Memory
Last 32b of a vector write into HWA Memory
Local Maxima output placement in HWA Memory

Figure 29: Local Maxima Output Write Pattern

(a) for 16 angles, the lower half-word is relevant, the upper is all zeros

Row i	16b'0	16b
Row i+1	16b'0	16b

(b) for 48 angles, 3 half-words are relevant, the 4th is all zeros

Row i	16b	16b
	16b'0	16b
Row i+1	16b	16b
	16b'0	16b

Figure 30: Local Maxima Output Write Pattern for (a) 16 Angles, (b) 48 Angles.

4.2 Local Maxima Engine – Register Descriptions

Table 14: Local Maxima Engine – Registers

Register	Width	Parameter Set	Description
LM_NEIGH_BITMASK	8	Y	Neighbour bitmask for CUT comparison: Bit mask for the 8 neighbouring cells (adjacent left/right/top/left/diagonal) to enable or disable the comparison with cell under test. Starting from left top corner and moving in a clock-wise direction. If a bit is set to 1, then the corresponding comparison is disabled.
LM_THRESH_BITMASK	2	Y	Enable/Disable for detection threshold comparison: 00b: Enable CUT comparison with detection threshold in both dimensions 01b: Enable CUT comparison with detection threshold in column dimension only (B-
LM_THRESH_MODE	2	Y	Threshold source selection: This 2-bit register is used to indicate whether the detection threshold is provided by a software register, or derived from the built-in Advanced Statistics RAM. The LSB of this register corresponds to column dimension (B-dimension), and the MSB corresponds to row dimension (C-dimension). If the register bit is 0, then the threshold
DIMB_THRESH_VALUE	16	N	SW configurable column threshold register: Refer description of LM_THRESH_MODE for more details.
DIMC_THRESH_VALUE	16	N	SW configurable row threshold register: Refer description of LM_THRESH_MODE for more details.
MAX2D_ OFFSET_DIM1	24	No	Offset for row thresholds from Advanced Statistics RAM: Offset to be added to dimension 1 Maxima results (from Advanced Statistics) to derive row thresholds for Local Maxima computation.
MAX2D_ OFFSET_DIM2	24	No	Offset for column thresholds from Advanced Statistics RAM: Offset to be added to dimension 2 Maxima results (from Advanced Statistics) to derive column thresholds for Local Maxima computation.
LM_DIMB_NONCYCLIC	1	Y	Dimension B Non Cyclic: 1: Non-cyclic mode of Local Maxima detection (i.e., for the edge samples, the neighboring cells will not be wrapped around) 0: Cyclic mode of Local Maxima detection (i.e., for the edge samples, the neighboring cells will be wrapped around)

LM_DIMC_NONCYCLIC	1	Y	Dimension C Non Cyclic: 1: Non-cyclic mode of Local Maxima detection (i.e., for the edge samples, the neighboring cells will not be wrapped around) 0: Cyclic mode of Local Maxima detection (i.e., for the edge samples, the neighboring cells will be wrapped around)

5 Context Switching

The state machine supports an advanced feature called ”Context Switching” that allows a sequence of operations running in the hardware accelerator to be interrupted, in order to run a different (higher priority) sequence of operations, before returning back to resume execution of the original sequence. This state machine feature is useful when Doppler FFT, detection and angle FFT processing of a radar frame is still running, while the chirps for the subsequent frame have already started. In such a case, it is possible for the hardware accelerator to perform the Doppler FFT, detection and angle FFT processing in one context (a.k.a the background context), while permitting “context switching” to an alternate context (a.k.a the high priority context) to perform inline range FFT processing as and when new chirp ADC data samples are received.

5.1 Context Switching – Operation

The context switching feature of the state machine can be enabled by setting the CS_ENABLE register bit. This register is a common (static) register that controls the overall enabling/disabling of the feature.

The background context execution is programmed and configured using PARAM_START_IDX, PARAM_END_IDX, and NUMLOOPS registers as before. The high priority context is programmed using additional parameter-sets that are configured using PARAM_START_IDX_ALT, PARAM_END_IDX_ALT and ALT_NUMLOOPS registers (refer Table 16). Normally, the state machine executes the parameter-sets in the background (low-priority) context by looping through the parameter-sets between PARAM_START_IDX and PARAM_END_IDX, until a context switch is triggered. Whenever a context switch from background context to high priority context happens, the state machine jumps to PARAM_START_IDX_ALT and executes the parameter-sets from PARAM_START_IDX_ALT to PARAM_END_IDX_ALT for ALT_NUMLOOPS number of times and returns back to resume execution of the background context from where it left off.

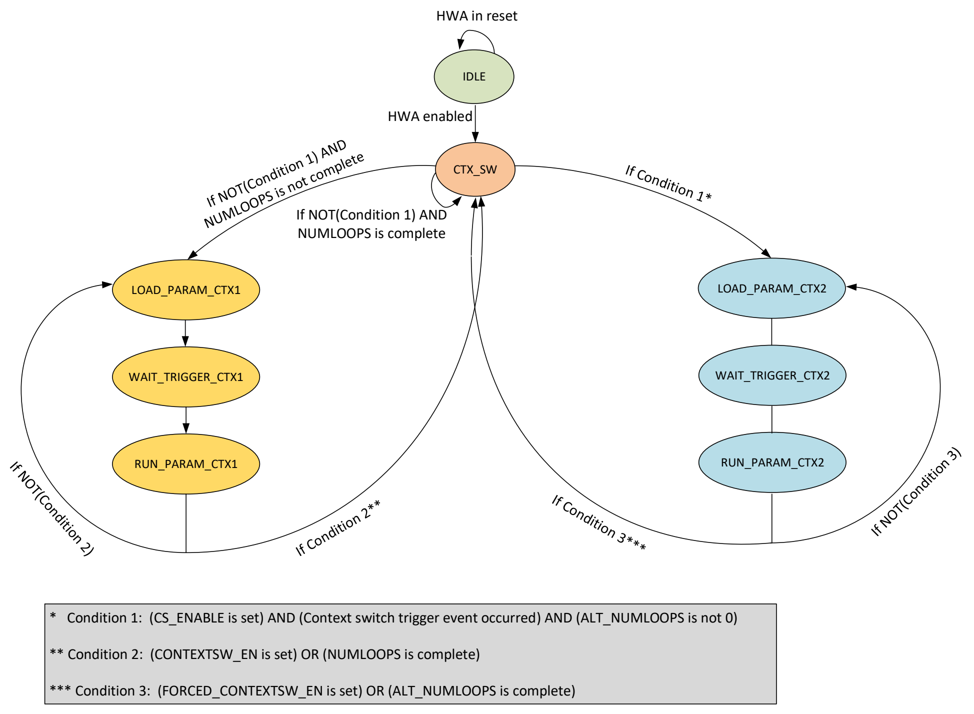
The context switch from background context to high priority context is triggered based on the settings of CS_TRIGMODE and CS_TRIGSRC registers. Specifically, CS_TRIGMODE can be configured to trigger context switch based on a DMA completion, or based on a CSI2 line/frame end event. Refer the register description section for details of these registers.

It is important to note that the context switch only happens at the end of execution of a parameter-set and never in the middle of execution of a parameter-set. Also, the context switch is allowed to happen at the end of a parameter-set only if CONTEXTSW_EN register bit in the parameter-set is set. If this register is not set, then context switch will not happen at the end of that parameter-set. CONTEXTSW_EN should only be set in the parameter-sets corresponding to background context.

Context switch from the high priority context back to the background context typically occurs after the completion of all parameter-sets in the high priority context. But if an early exit is needed, i.e. after the execution of a certain high priority parameter-set, the register FORCED_CONTEXTSW_EN in that parameter-set can be set. This forces a context switch to the background context, without the need for any explicit context switch trigger. In such a case, when the next context switch trigger occurs, the high priority context execution resumes from where it left off. FORCED_CONTEXTSW_EN should be set only in the parameter-sets corresponding to the high priority context, and can be used for debug purposes. It can also be used when a large time gap is expected between execution of one high priority parameter-set and the next (e.g. due to the timing of the associated trigger sources), and it is desired that this time gap be used for some background context execution.

Generally, for most computations, there is no ‘state information’ that carries forward across parameter-sets – i.e., each parameter-set is independent of the other. However, there are four specific items – namely, DC estimation accumulators, Interference statistics accumulators, TWID_INCR_DELTA_FRAC execution count and Recursive window execution count, which have state information in the form of accumulator value or execution count value that is applicable across parameter-sets. Therefore, while using the context switching feature, appropriate care has to be taken by the user to ensure that these features do not conflict when used in both the background context and high-priority context at the same time. The solution to this is to use these features only in one context at a time (eg. DC estimation and compensation can be used only in range FFT context), or to avoid jumping context between certain back-to-back parameter-sets (eg. Avoid jumping context between Interference statistics estimation and Interference mitigation steps), or to use the processor to save and restore state information (where possible). Also, care should be taken by the user to ensure that the usage of source and destination (ACCEL_MEMx) memories is mutually exclusive between the two contexts, so that there is no unintentional overwriting of the memory bank that is still under use by the background context by some computations of the high-priority context.

The operation of the state machine featuring the context switch capability is shown in Figure 31 below. In the figure, Context1 and Thread1 indicate the background context (states shown in left side of the figure) and Context2 and Thread2 indicate the high-priority thread (states shown in right side of the figure).



* Condition 1: (CS_ENABLE is set) AND (Context switch trigger event occurred) AND (ALT_NUMLOOPS is not 0)

** Condition 2: (CONTEXTSW_EN is set) OR (NUMLOOPS is complete)

*** Condition 3: (FORCED_CONTEXTSW_EN is set) OR (ALT_NUMLOOPS is complete)

Figure 31: Context switching state machine

Whenever the NUMLOOPS of the background context completes, the state machine raises a loop complete interrupt (DSS_HWA_THREAD1_LOOP_INT). For every time the ALT_NUMLOOPS of the high-priority context completes, the state machine raises a different loop complete interrupt (DSS_HWA_THREAD2_LOOP_INT).

Note that when the background context completes (i.e., NUMLOOPS is done), the state machine does not automatically move to IDLE state and remains in the CTX_SW state. This state means that the high-priority context switch can still happen even after all the operations for the background context are complete. The only way to make the state machine go back to IDLE state is by disabling the hardware accelerator completely. In typical frame-by-frame processing applications, the host may disable the accelerator after the background context is completed and the necessary number of calls of the high priority context have been completed for one frame (counted by the host), and if necessary reconfigure the accelerator and trigger it again to restart the process.

Note that the accelerator's execution always begins with the background thread and proceeds to the high priority context upon receiving context switch trigger. If the application needs the operations to begin directly with the high priority context, then the user can define a "No Operation" parameter set in the background context and set CONTEXT_SW_EN in it, to indirectly achieve this.

5.2 State Machine – Context Switching – Register Descriptions

Table 15: Context Switching – Registers

Register	Width	Parameter Set	Description
CS_ENABLE	1	N	Context Switching Enable: 0: Disable context switching feature. 1: Enable context switching feature.
PARAM_START_IDX_ALT	10	N	Parameter-set start index for high-priority context: When context switch from background context to high-priority context happens, the state machine starts execution of high-priority context at the parameter-set indicated by this register. Valid range: 0-63
PARAM_END_IDX_ALT	10	N	Parameter-set stop index for high-priority context: When running the high-priority context, this register indicates the last parameter-set in the sequence, before looping back to PARAM_START_IDX_ALT. Valid range: 0-63
ALT_NUMLOOPS	12	N	Number of loops for high-priority context: This register specifies how many times (loops) for which the parameter-sets corresponding to the high-priority context are executed before returning back to resume the background context.
CS_TRIGMODE	4	N	Trigger mode selection for context switch: 0000b: Reserved 0001b: Reserved 0010b: Reserved 0011b: DMA-based trigger (used in conjunction with CS_TRIGSRC) 0100b: Hardware trigger (used in conjunction with CS_TRIGSRC) 0101b: Software trigger (used in conjunction with CS_FW2ACC_TRIG) Other values : Reserved

CS_TRIGSRC	4	N	Trigger source for context switch: When CS_TRIGMODE = 0011b (i.e., in case of DMA-based trigger mode), this register specifies which DMA channel (i.e., which bit in DMA2CS_TRIG register) to wait for being set in order to perform context switch to high-priority context. When CS_TRIGMODE = 0100b (i.e., in case of hardware trigger mode), this register specifies which CSI2 trigger signal (out of the 20 possible trigger signals) to wait for in order to switch to high-priority context. The 20 signals are listed below, with CS_TRIGSRC = 0 corresponding to the last signal in the list: {CSI2A_FRAME_START[1:0], CSI2A_LINE_END[7:0], CSI2B_FRAME_START[1:0], CSI2B_LINE_END[7:0]}
CONTEXTSW_EN	1	Y	Context switch enable register: This register is used in the background context. If this register is set to 1 in a parameter-set, then the state machine checks for a context switch trigger at the end of that parameter-set execution. If a trigger is found, the higher priority parameter-sets start executing. On returning from the higher priority context the next background parameter-set is run.
FORCED_CONTEXTSW_EN	1	Y	Forced context switch enable register: This register can be used in the higher priority context. If this register is set to 1 in a parameter-set, then the higher priority execution is interrupted after this parameter-set. And the execution switches to the next background parameter-set. Later when a context switch trigger is received the higher priority thread resumes from the parameter-set after the parameter-set having FORCED_CONTEXTSW_EN set.
FW2HWA_TRIGGER_CS	1	N	Software context switch trigger: When CS_TRIGMODE = 0101b, this register bit can be set by software to trigger a context switch.

6 Compression Engine

The accelerator includes a Compression Engine, which can compress or uncompress data in order to reduce storage RAM size requirements. For example, after range dimension FFT of the received RX data, the FFT results can be input to the Compression Engine and its output can be stored in a relatively smaller RAM in the device (e.g. L3 RAM). And before performing any Doppler FFT processing, the data can be retrieved from the RAM, input to the Compression Engine for un compression, and then fed to any further Doppler FFT processing steps.

The features and configurability of the Compression Engine are described in a separate document.

7 References

[1] *Radar Hardware Accelerator User's Guide – Part 1*

[2] *TPR 12 Technical Reference Manual*

[3] *Radar Hardware Acceletor Compression Decompression Manual*

Revision History (covers both Part 1 and Part 2)

0.8	29-Aug-2019	First external draft of HWA2.0 user guide, explaining most of the features – including the HWA2.0 new features like DC estimation and subtraction, Interference localization, Local Max Engine, FFT improvements, etc.
0.9	9-Nov-2019	Added description of Shuffled addressing in Input Formatter. Various changes to register names. Changes to some complex multiplier modes like recursive window mode.
0.91	25-Nov-2019	Additional information about interference mitigation. Added section on context switching feature. Made some edits to the descriptions in local max engine and complex multiplier modes. Fixed error in Figure showing output formatter scaling from 24-bit to 32-bit word (DSTSCAL register).
0.92	20-Dec-2019	Minor edits and clarifications added based on review feedback.
0.921	03-Feb-2020	Minor incremental editions based on review feedback. Added parameter set register layout to the user guide itself.
0.93	19-Feb-2020	Incremental editions based on review feedback. Including corrections and editions in local maxima and CFAR sections.
0.94	27-Mar-2020	Added Interference Localization figure, edited DC and Intf Stats figures. Updates to Register descriptions
0.95	25 April 2020	Merged Adv Stats, Local Max and CS edits from SJ. Cleaned Figure Captions and cross-reference. Vector Mult RAM layout figure added
0.96	28 April 2020	Merged KS 2D Window section. Increased font-size of captions
0.96	11 May 2020	Took in KR sections and incorporated all feedback
0.97	1 Sep 2020	Added list of Contents, Figures and Tables
0.97	25 Nov 2020	Minor fix to figure on Vector Mult RAM