Radar Hardware Accelerator - Part 2

The radar hardware accelerator user's guide is presented in two parts: *Radar Hardware Accelerator User's Guide - Part 1* and *Radar Hardware Accelerator 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.

This user's guide is split into two parts:

- The first part (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 this document covers additional features like CFAR-CA, 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 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 detection feature.
- Section 3 covers other miscellaneous capabilities such as statistics computation are discussed.
- Section 4 covers the specific use-case of FFT stitching is described using an example.

Contents

List of Figures

List of Tables

Trademarks

1 Core Computational Unit – Pre-Processing

This section provides an overview of the pre-processing block inside the core computational unit. Specifically, the features of interference zero-ing out and complex multiplication.

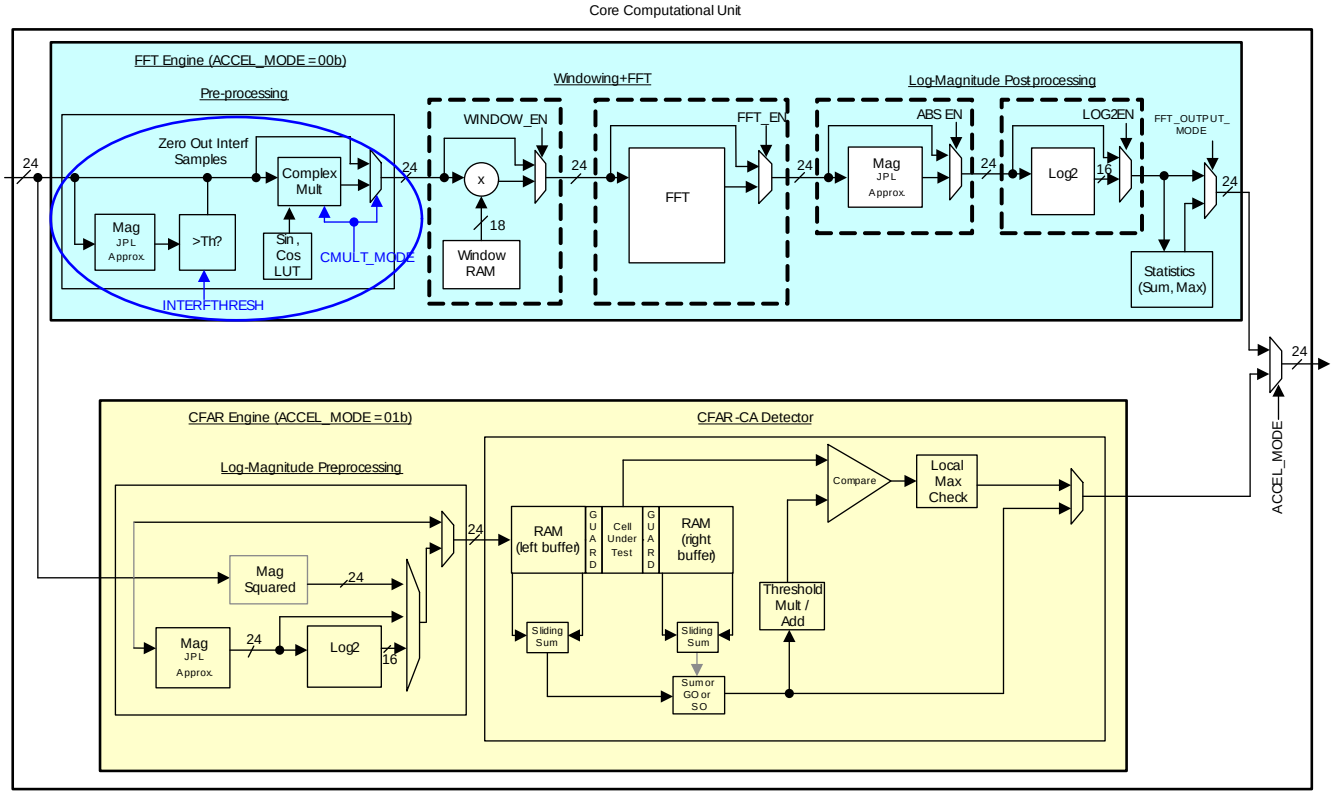

Figure 1. Core Computational Unit

1.1 Pre-Processing Block

The pre-processing block (see Figure 1) provides the ability to perform some simple pre-FFT manipulations of the samples. As shown in Figure 1, the pre-processing block, the windowing block, FFT block and log-magnitude post-processing block are connected to each other, so that the output from one block can be fed into the next block, if required. One or more of these blocks can be enabled in each parameter-set to perform the desired operations.

The pre-processing block comprises provision for interference zero-out and complex multiplication. These sub-blocks are described in the following sections.

1.1.1 Interference Zero-Out

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 loss of detection.

In order to mitigate the impact of interference, one common technique is to zero-out any large glitches in the time-domain samples. The pre-processing block provides capability to perform this operation. Specifically, 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 *Radar Hardware Accelerator User's Guide - Part 1*. Any sample whose magnitude exceeds a programmable threshold is zero-ed out, both in real and imaginary part. The programmable threshold is a 24-bit register named INTERFTHRESH. This threshold register is not part of the parameter-set and is a common register. Note that a value of 0xFFFFFF can be programmed to disable the zero-out feature, since the magnitude can never exceed this value anyway. Alternately, a separate INTERFTHRESH_EN register is provided as part of the parameter-set to control when the interference zero-ing out should be enabled.

Note the limitation that the threshold is a static configuration and there is no specific provision in the accelerator to automatically calculate the threshold based on the RMS of the current set of data samples. However, there are possible ways of accomplishing this indirectly, by performing a first-pass of the input samples to compute the signal statistics (with a bit-shift scaling on the statistic) and then using the main processor or DMA to copy this scaled statistic value to the interference threshold register, so that in a subsequent second pass, the interference zeroing out can be accomplished. On a separate note, if the user is interested just to know the indices of the time-domain spikes, then it is possible to use the CFAR engine path to look for spikes in the samples and record the indices of the spikes. The CFAR engine is covered in Section 2.

1.1.2 Complex Multiplication

In addition to interference zero-out, the pre-processing block contains a complex multiplication sub-block. The purpose of this sub-block 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 000b. 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 seven 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 and 2.

- Frequency shifter mode: If the register value is CMULT_MODE = 001b, 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). Another register (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). In effect, the input samples x(n) for n = 0 to SRCACNT-1 are multiplied by the sequence, exp(j*2*pi*TWIDINCR*(0:SRCACNT-1)/16384).

- Frequency shifter with auto-increment mode (a slow DFT mode): If the register value is CMULT_MODE = 010b, 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 200 MHz 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 = 011b, then the complex multiplier functions in FFT stitching mode. This mode is useful when large size FFTs (2K and 4K) are required. Since the FFT block natively supports only up to 1024 size, for 2048 and 4096 point FFT, an FFT Stitching procedure using two steps (two parameter-sets) can be used. As an example, when a 4K size FFT is needed, it is achieved in two steps as follows. In the first step, every 4th input sample is passed through a 1K size FFT (four 1K point FFTs are performed on decimated input samples). Then, in the next step, the resulting 4x1024 FFT outputs are sent through four-point "stitching" FFTs (1024 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 TWIDINCR register (2 LSB bits). If the LSB two bits of TWIDINCR register are 00b, then the twiddle factor pattern will correspond to what is required for 2K (2x1024) size FFT stitching. If the LSB two bits are 01b, then the twiddle factor pattern will correspond to what is required for 4K (4x1024) size FFT stitching. Values of 10b and 11b are reserved and should not be used. Also, the unused 12 MSB bits of TWIDINCR register must be zero in this mode of operation. The last section includes a more detailed explanation and configuration information for the FFT stitching example for 2K and 4K FFT, including the use of WINDOW_INTERP_FRACTION register for extending the window RAM using linear interpolation to more than 1024 coefficients.
- Magnitude squared mode: If the register value is CULT_MODE = 100b, then the complex multiplier functions in magnitude squared mode. In this case, the complex multiplier takes every complex input and produce 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: If the register value is CMULT_MODE = 101b, then the complex multiplier functions in 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 ICMULTSCALE and QCMULTSCALE registers (for I and Q value, each having 21 bits). The ICMULTSCALE and QCMULTSCALE registers are common registers and not part of parameter-set. Note that this feature cannot be used to multiply the input samples for different iterations (channels) with different complex scalars.

- Vector multiplication mode 1: If the register value is CMULT_MODE = 110b, 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 internal RAM inside the core computational unit. (This internal RAM is a RAM that is normally used by the FFT block when performing 1024-point FFT computation). This internal RAM can store 512-complex samples and hence the vector multiplication can support a maximum of 512 elements of multiplication. The Vector multiplication is not a highly parallelized operation, in the sense that only one complex multiplication is done per 200MHz clock cycle.

  It is important to note one important limitation: since the internal RAM is shared with the FFT, performing a 1024-point FFT destroys the pre-loaded contents of the RAM. Therefore, performing vector multiplication and 1024-point FFT back-to-back many times requires re-loading of the internal RAM each time and will be inefficient. However, note that this limitation of having to re-load the internal RAM does not apply when performing FFT of size 512 or less, which is often the case for second and third dimension FFTs.

  The operation of the vector multiplication mode 1 is as follows. The streaming set of samples from the Input Formatter block coming at 200 MHz is element-wise multiplied with successive samples from the internal 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 internal RAM is reset, so that for the next iteration, the samples are picked up from the start of the internal RAM.
- Vector multiplication mode 2: If the register value is CMULT_MODE = 111b, then the complex multiplier functions in vector multiplication mode 2, which is slightly different from the earlier mode. The only difference in this case is that at the end of every iteration, the addressing of the internal RAM is not reset, so that for the next iteration, the samples from the internal 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 internal RAM can be loaded with all the vectors, such that for successive iterations, the input samples are multiplied with successive vectors.

For loading the internal RAM used for the vector multiplication modes, the register bit STG1LUTSELWR is used. The internal RAM for vector multiplication mode is mapped to the same address space as the Window RAM. Therefore, this register bit is required to specify which of these two (Window RAM or internal RAM) need to be selected, when loading the co-efficients via DMA or main processor. If the register bit is 0, then the Window RAM is selected, else, the internal RAM for vector multiplication mode is selected. Note that the other registers such as WINDOW_START, which pertains to windowing, are always applicable only for the Window RAM. The STG1LUTSELWR register bit should in general be kept as 0 (Window RAM selected). This allows the main processor or DMA to have access to the Window RAM by default. Only when it is desired to load the internal RAM with coefficients for vector multiplication mode, this register bit should be temporarily set to 1. After loading the coefficients, the register bit should be made 0 again.

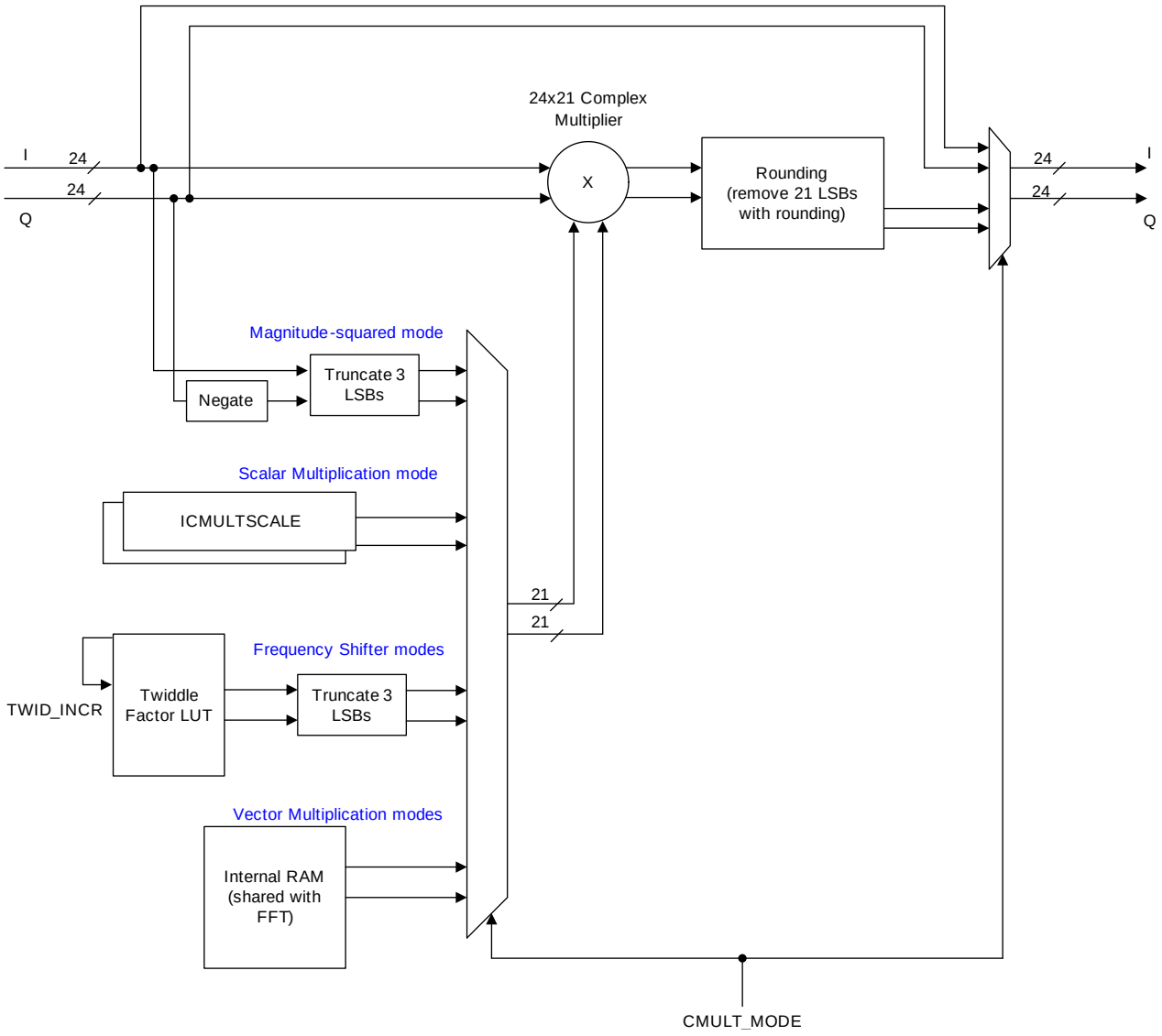

Figure 2. Complex Multiplication Capability in Pre-Processing Block

Note that in all the above seven modes of the complex multiplier, only one complex multiplication is performed every 200 MHz clock. So, the effective speed achieved for the multiply or multiply-accumulate operation is 200 MHz.

1.1.3 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 64). 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: 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 sequence present in the 64-bit BPMPATTERNLSB and BPMPATTERNMSB register. 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.

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 3.

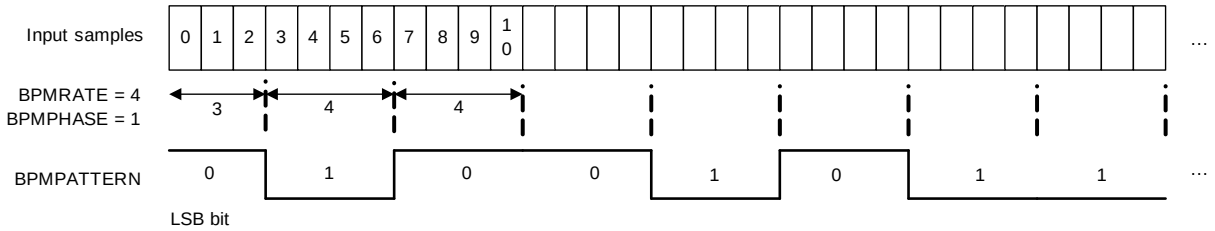

Figure 3. BPM Removal Capability

If multiple iterations (for example, four back-to-back FFTs in a single parameter-set using REG_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 64 bits long, hence, the maximum BPM sequence length that is supported is 64. For higher BPM sequence length, the alternate approach of pre-multiplying the window coefficients stored in the window RAM may be considered.

1.1.4 Pre-Processing Block – Register Descriptions

Table 1 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 1.

Table 1. Pre-Processing Block Registers

Register	Width	Parameter-Set? (Y/N)	Description
INTERFTHRESH .	24	N	Interference threshold: This register is used to specify the threshold for zero-ing out samples affected by interference. Any sample whose magnitude exceeds this threshold is zero-ed out, if the feature is enabled using INTERFTHRESH_EN register bit
INTERFTHRESH_EN	1	Y	Interference zero-out Enable/Disable: This register bit controls the enable/disable for the interference zero-out feature. The feature is enabled if this register bit is set in any given parameter-set.
CMULT_MODE	3	Y	Complex multiplication mode: This register is used to configure the mode of the complex multiplication sub-block. A value of 000b disables/bypasses the complex multiplication. Any other value chooses one of 7 available modes of operation . Detailed description of the seven modes in the main description section.

Table 1. Pre-Processing Block Registers (continued)

Register	Width	Parameter-Set? (Y/N)	Description
TWIDINCR	14	Y	Twiddle factor configuration: When the complex multiplication sub-block is programmed in one of the frequency shifter modes (CMULT_MODE = 001b or 010b), this register is used to indicate the amount of frequency shift. Specifically, the input samples x(n) for n = 0 to SRCACNT-1 are multiplied by the sequence: exp(j*2*pi*TWIDINCR*(0:SRCACNT-1)/16384). When the complex multiplication sub-block is programmed in FFT stitching mode (CMULT_MODE = 011b), the last two bits of this register specify whether it is 2K or 4K FFT stitching. Specifically, if the last two bits are 00b, then it is 2K (2*1024-point) FFT stitching and if the last two bits are 01b, then it is 4K (4*1024-point) FFT stitching. Values of 10b and 11b are reserved. Also, the 12 MSB bits of this register must be zero. In all other modes of the complex multiplication sub-block, this register must be kept as 0.
FFTSIZE	4	Y	FFT size (DFT mode): For the register description during normal FFT operation, see *Radar Hardware Accelerator User's Guide - Part 1*. This register also has a second purpose in DFT mode as described below. When FFT is disabled, and complex multiplication mode is set to be frequency shifter with auto-increment (slow DFT mode), then this register is used to specify the DFT bin resolution. The DFT bin resolution is given by 2^FFTSIZE. For example, if FFTSIZE = 1011b, then the DFT resolution is 2048 (2K size).
STG1LUTSELWR	1	N	Select Window RAM or internal RAM: The internal RAM for vector multiplication mode is mapped to the same address space as the Window RAM. Hence, this register bit is required to specify which of these two needs to be selected when loading the co-efficients via DMA or R4F. If this register bit is 0, then the Window RAM is selected, else, the internal RAM for vector multiplication mode is selected. Keep this register bit as 0 always, except during the period when internal RAM needs to be loaded.
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.
BPMPATTERNLSB and BPMPATTERNMSB	64	N	BPM pattern: Specifies the BPM pattern to be used to multiply the input samples if BPM removal is enabled.
BPMRATE	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 in Section 1.1.3.

2 Core Computational Unit - CFAR Engine

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

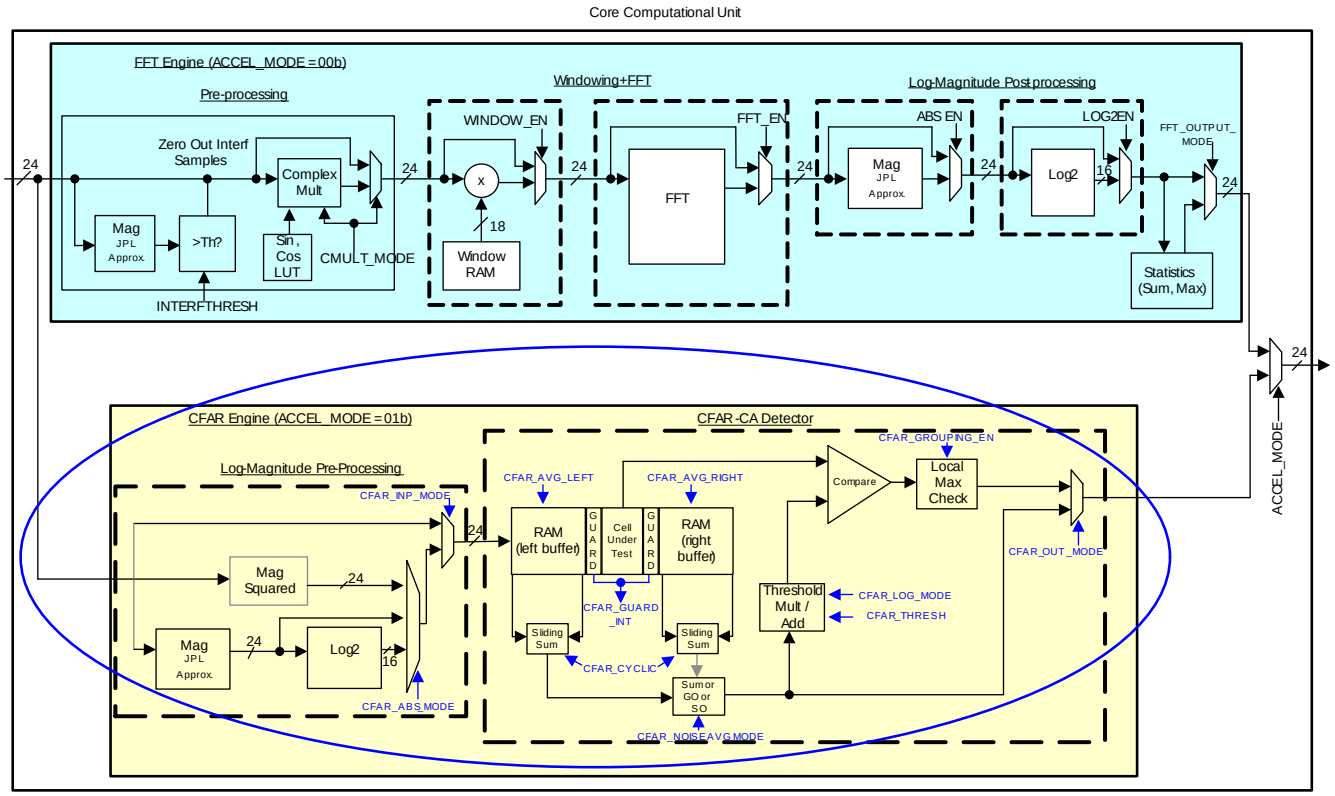

Figure 4. CFAR Engine

2.1 CFAR Engine

The CFAR engine (see Figure 4) is a module that enables detection of objects, by identifying peaks in the FFT output. Although there are several detection algorithms, the accelerator supports only the CFAR-CA algorithm and a few variants of it. CFAR-CA stands for constant false alarm rate – Cell Averaging. The other popular technique known as ordered-statistic CFAR, a.k.a CFAR-OS is not supported.

As shown in Figure 4, the CFAR engine path is selected by setting the accelerator mode ACCEL_MODE = 01b. 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-CA detection processing (both linear and logarithmic CFAR modes are available) and generate a peak list.

In cell-averaging 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 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.

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 average' 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.

2.1.1 CFAR Engine – Operation

The CFAR engine receives 24-bit input samples from the Input Formatter block. Typically, these are 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 (see block diagram). The real unsigned result from this pre-processing operation is sent to CFAR-CA 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. 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.

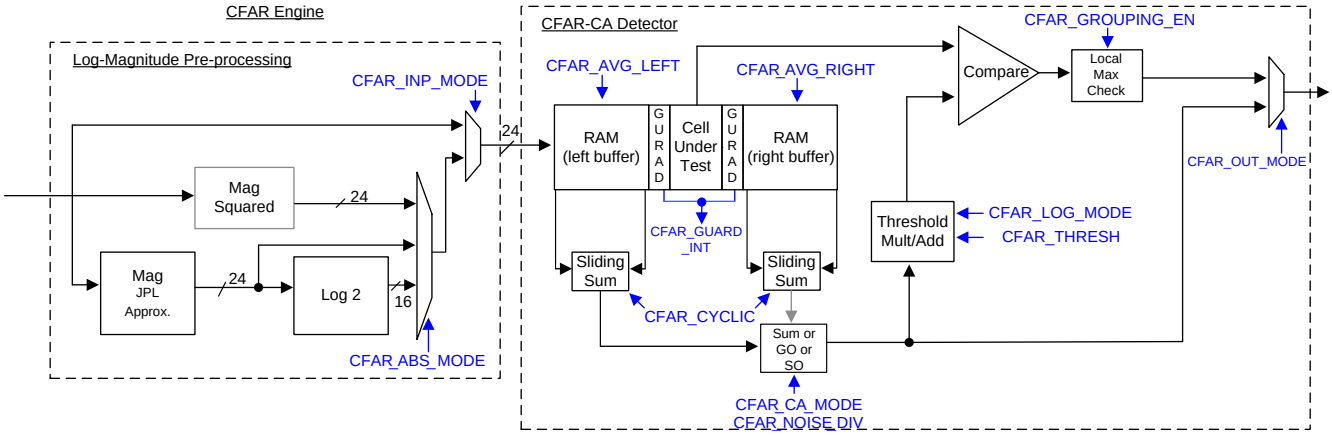

Figure 5. CFAR Engine Block Diagram

As described earlier, the CFAR-CA detection processing involves finding a “surrounding noise average” for each cell under test and then determining a threshold that is a function of the surrounding noise average. 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 average 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.

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 2 summarizes the register settings for the different CFAR modes of operation.

Table 2. 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-CA Algorithm	CFAR_CA_MODE Register Setting
CFAR-CA	00
CFAR-CAGO	01
CFAR-CASO	10

The surrounding noise average computation has multiple options – cell averaging (CFAR-CA), cell averaging with greater-of selection (CFAR-CAGO) and cell averaging with smaller-of selection (CFAR-CASO). The register CFAR_CA_MODE is used to select between CFAR-CA, CFAR-CAGO and CFAR-CASO 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 average value. 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

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. 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, 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 5.

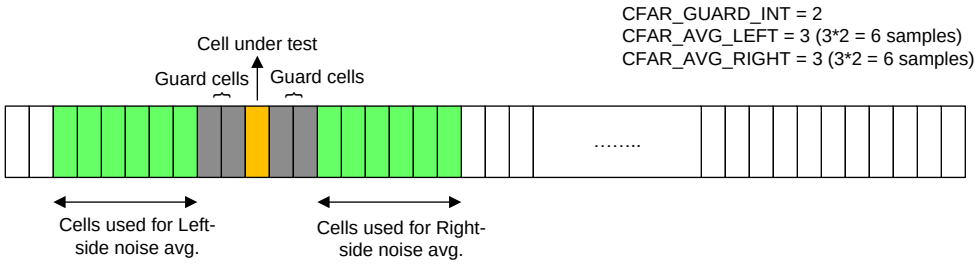

Figure 6. CFAR-CA: Cells Used for Surrounding Noise Average

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 average’ 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 average to determine the threshold, else it is added to the surrounding noise average. 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

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 7. 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 REG_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 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 average’ value for each cell. This is called ‘Raw output mode’.

Figure 7 and Table 3 show the different output formats available.

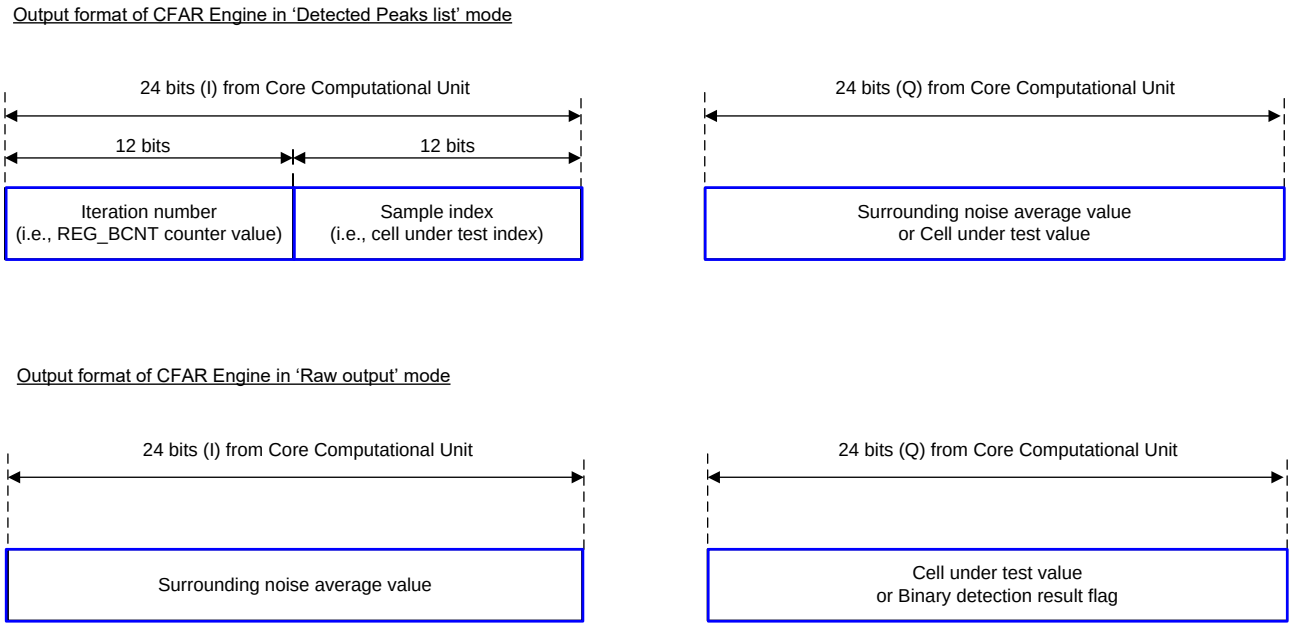

Figure 7. CFAR Engine Output Format

Table 3. CFAR Output Modes and Register Settings

CFAR Output Mode	I Channel Output	Q Channel Output	CFAR_OUT_MODE Register Setting
Detected peaks list mode (only detected peaks are output)	Peak index	Surrounding noise average value	10
	Peak index	Cell under test value	11
Raw output mode (all cells are output)	Surrounding noise average value	Cell under test value	00
	Surrounding noise average value	Binary detection result flag (0 or 1)	01

In detected peaks list mode, only the detected peaks are output to the destination memory. In this case, the read-only register FFTPEAKCNT indicates how many peaks have been totally detected, so that the main processor can read that many locations from the destination memory.

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).

2.1.3 CFAR Engine – Cyclic vs. Non-Cyclic

The register CFAR_CYCLIC specifies whether the CFAR-CA detector needs to work in cyclic mode or in non-cyclic mode. These two modes are different in the way the samples at the edges are handled. In non-cyclic mode, the left side noise average is unavailable for the first several cells under test, therefore, only the right side noise average is used for those initial cells. Similarly, the right side noise average is unavailable for the last several cells under test, and only the left side noise average is used for those cells. For all other cells in the middle, both left and right side noise averaging is used as per the number of samples programmed in CFAR_AVG_LEFT and CFAR_AVG_RIGHT registers.

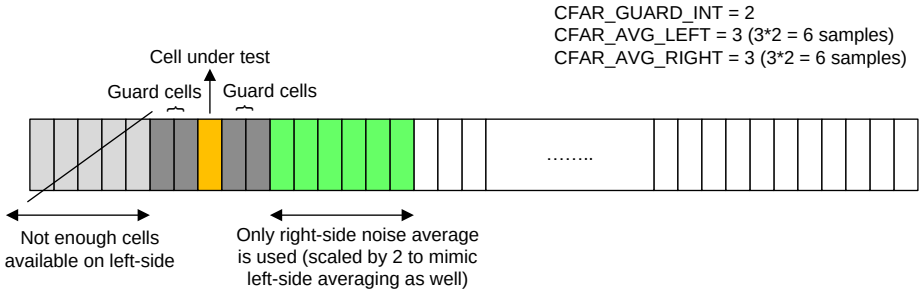

Figure 8. Handling of Samples Near the Edge in Non-Cyclic Mode

On the other hand, in cyclic mode, the CFAR-CA detector needs to wrap around the edges in a circular manner. In other words, for a cell under test that is near the left extreme, the left side noise average computation needs to use some samples from the right edge (circular wrap around the edge). Similarly, for a cell under test that is near the right extreme, the right side noise average computation needs to use some samples from the left edge (again, circular wrap around.

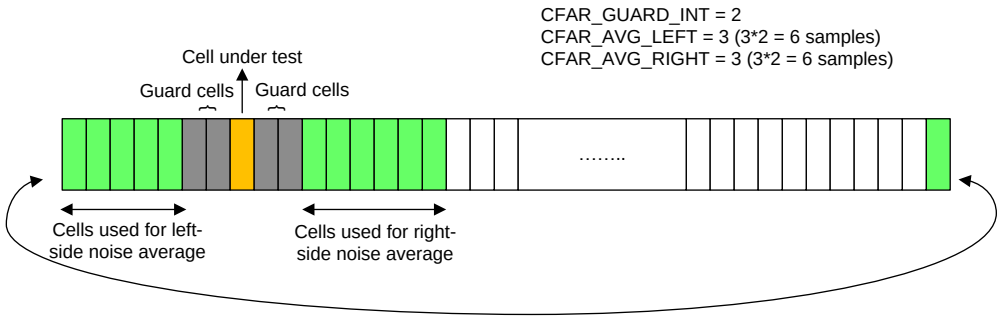

Figure 9. Handling of Samples Near the Edge in Cyclic Mode

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 circular shift (CIRCIRSHIFT) and circular wrap-around (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, 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 4

Table 4. 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
	CIRCIRSHIFT = 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
	CIRCSHIFTWRAP = 8	The circular wrap-around happens when SRCACNT counter value reaches 2^CIRCSHIFTWRAPP = 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

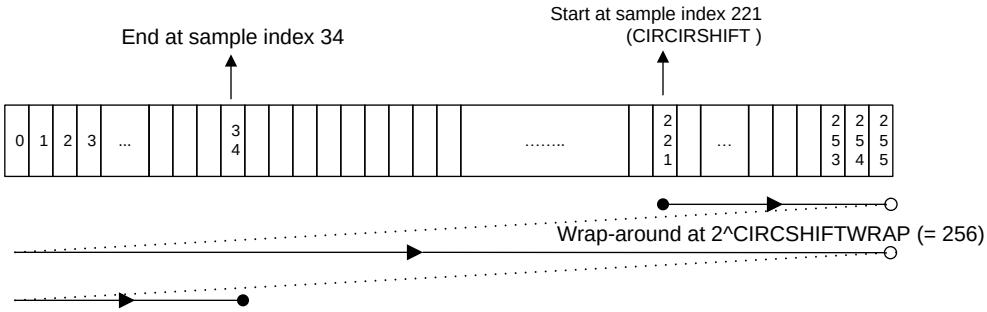

Figure 10. Input Formatter Sample Streaming for the Cyclic CFAR Example

2.1.4 CFAR Engine – Register Descriptions

Table 5 lists all the registers of the CFAR engine block. A few registers belonging to input formatter block that are related to cyclic CFAR mode of operation are also listed here.

Table 5. CFAR Engine Registers

Register	Width	Parameter-Set? (Y/N)	Description
CFAR_AVG_LEFT	6	Y	Number of samples for left-side noise averaging: This register is used to specify the number of samples used for noise averaging to the left of the cell under test. The number of samples used for noise averaging 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 averaging 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 averaging. A value of 1 is not supported (valid values are 0, 2, 3, … 63).
CFAR_AVG_RIGHT	6	Y	Number of samples for right-side noise averaging: 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.
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 average.

Table 5. CFAR Engine Registers (continued)

Register	Width	Parameter-Set? (Y/N)	Description
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 average' 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 average to determine the threshold, else it is added to the surrounding noise average. 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_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 average 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 average 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 approx.) 11b – Log2-Magnitude (using LUT approx.)
CFAR_OUT_MODE	2	Y	CFAR engine output mode: This register is used to select the output mode of the CFAR engine. 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. Refer main description section for details.
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 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:

Table 5. CFAR Engine Registers (continued)

Register	Width	Parameter-Set? (Y/N)	Description
CFAR_CYCLIC	1	Y	This register configures the noise averaging mode in the CFAR detector from one of three options – CFAR-CA, CFAR-CAGO, CFAR-CASO. 00b – CFAR-CA 01b – CFAR-CAGO 10b – CFAR-CASO 11b – Not valid CFAR cyclic vs. non-cyclic mode: This register bit specifies whether the CFAR-CA 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.
FFTPEAKCNT (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.
CIRCIRSHIFT	12	Y	This register is part of input formatter block. Source Circular Shift: This register specifies the circular shift (offset in samples) that should be applied on the sequence of input samples before feeding them to the core computational unit. This register, together with CIRCSHIFTWRAP register, is useful when CFAR detection needs to be done in cyclic mode. Refer main description section for details.
CIRCSHIFTWRAP	4	Y	This register is part of Input Formatter block Source Circular Shift Wraparound: This register indicates at what number (power-of-2) the sample counter value should wraparound, when circular shift is needed. The counter wraps around at 2^ CIRCSHIFTWRAP. The sample counter starts counting from the programmed circular shift value (CIRCIRSHIFT) and when the counter crosses (2^CIRCSHIFTWRAP-1), it wraps back to zero to continue the count.

3 Core Computational Unit – Statistics

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

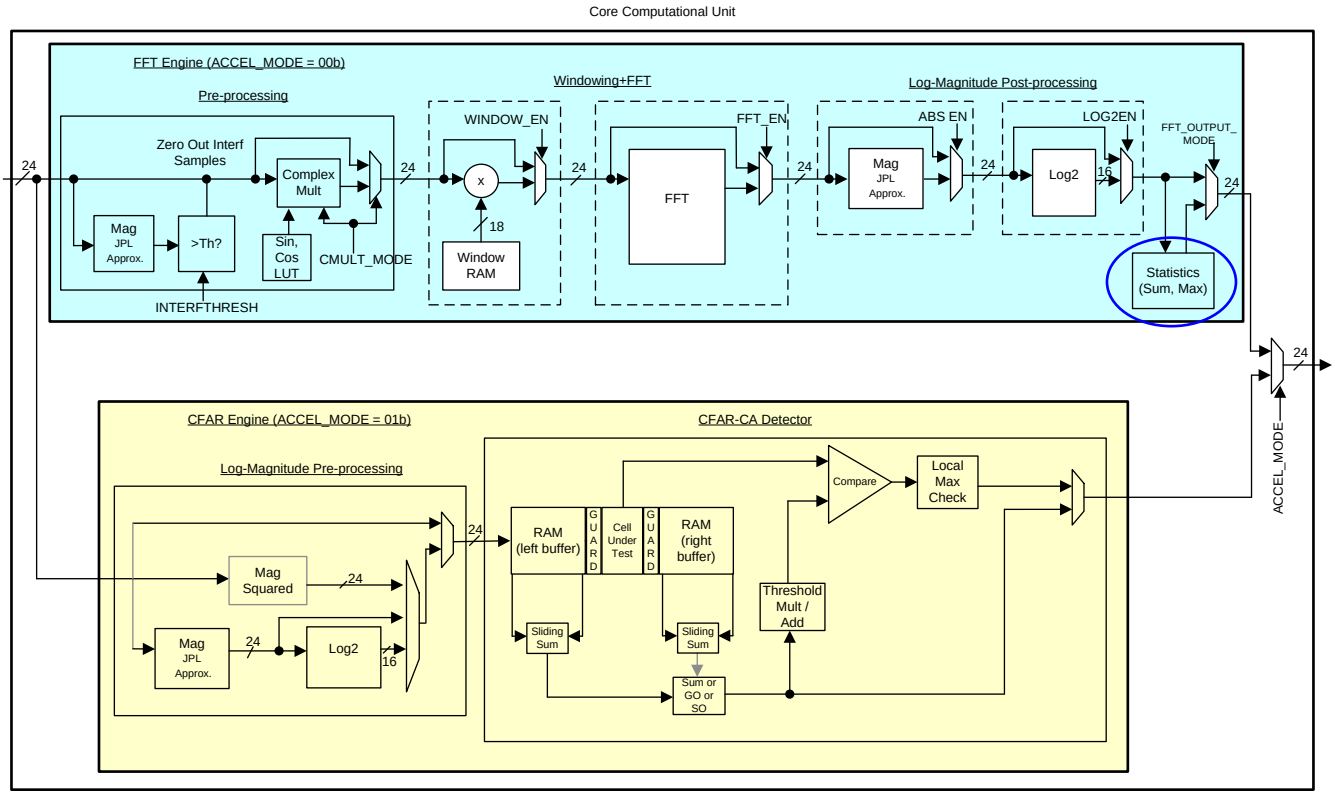

Figure 11. Statistics Block

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 11. This block can be used to compute a few statistics, such as sum and max of the samples output by the core computational unit.

3.1.1 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 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 6. 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 its maximum value of 4095, 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 7 lists all the registers of the statistics block.

Table 7. Statistics Block Registers

Register	Width	Parameter-Set? (Y/N)	Description
MAX1VALUE MAX2VALUE MAX3VALUE MAX4VALUE	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).
MAX1INDEX MAX2INDEX MAX3INDEX MAX4INDEX	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.
ISUM1LSB and ISUM1MSB ISUM2LSB, ISUM2MSB ISUM3LSB, ISUM3MSB ISUM4LSB, ISUM4MSB			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.

4 FFT Stitching Use-Case Example

This section presents examples that illustrate how to configure and use the radar hardware accelerator for the special use-case of FFT stitching.

4.1 FFT Stitching Use-Case

As described earlier, the radar hardware accelerator natively supports FFT sizes of up to 1024. However, FFT of size 2048 and 4096 can also be accomplished using a two-step FFT stitching process. This involves the use of two parameter-sets as shown in the example below.

Consider an industrial level-sensing use-case with 1 TX, 1 RX and 4096 complex samples per chirp. During active chirp transmission, the Digital Front-end (DFE) writes ADC samples to the ADC buffer in ping-pong manner. This example assumes that the ADC data is complex (the RF/analog is configured as complex baseband (instead of real-only) chain). Since 4096 samples requires 4096*4 = 16384 bytes, the DFE fills up the entire ping or pong ADC buffers (ACCEL_MEM0 or ACCEL_MEM1) for every chirp. The DFE configuration details are outside the scope of this user's guide.

The FFT processing using the radar hardware accelerator can be done inline (as and when the ADC data is available) by setting FFT1DEN = 1, such that the ADC buffer is shared with the accelerator input memories and the DFE output is directly available to the accelerator for processing at the end of every chirp (ping-pong switch). Alternately, it is possible to not use the radar hardware accelerator during ADC data collection, and instead, simply transferring the ADC data into L3 memory using DMA at the end of every ping-pong switch. In such a case, after all the chirps are collected, the radar hardware accelerator can be used to perform FFT during inter-frame time. This is accomplished by setting FFT1DEN register bit to 0, such that the ADC buffer is not shared with the accelerator input memories and therefore, the accelerator input memories are directly accessible for DMA transfer to provide input data for the accelerator. The use of inline FFT processing and inter-frame FFT processing is covered in the *Radar Hardware Accelerator - Use Case Example* section of the *Radar Hardware Accelerator User's Guide - Part 1*.

In Table 8 and Table 9, the parameter-set configurations for performing one 4096-point FFT using FFT stitching is shown. The input data is assumed to be in ACCEL_MEM0 as shown in the layout below. The FFT stitching is achieved in two steps as follows.

1. The first step involves computing four 1024-point FFTs of input samples. For these four 1024-FFT computation, input samples are sent in following order:
   a. Iteration #0: x[0], x[4], x[8], x[12], …, x[4092]
   b. Iteration #1: x[1], x[5], x[9], x[13], …, x[4093]
   c. Iteration #2: x[2], x[6], x[10], x[14], …, x[4094]
   d. Iteration #3: x[3], x[7], x[11], x[15], …, x[4095]

   Where x[0], x[1], x[2], …., x[4095] are the original 4096-point input samples which are stored in ACCEL_MEM0 in consecutive locations.

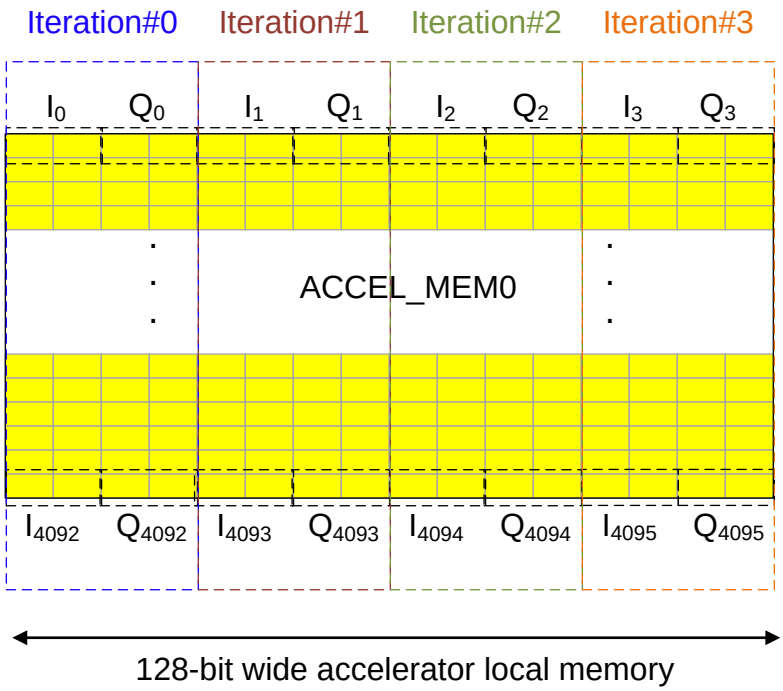

Figure 12. Layout of Samples in Source Memory for 1TX, 1RX, 4K Complex FFT

Further, before computing the 1024-point FFT in each iteration, apply windowing to the incoming input samples. Note that the window RAM can hold a maximum of only 1024 window coefficients. When larger FFT (2K and 4K) is needed via stitching of multiple smaller-size FFTs, a down-sampled set of window coefficients is stored in window RAM (1024 coefficients of original 4096 point window are stored by picking every 4th coefficient) and then, the accelerator is configured to use a linearly interpolation between these 1024 window samples to get intermediate window coefficients. The WINDOW_INTERP_FRACTION register, which is part of the parameter set, is used to configure interpolation mode for the window coefficients. When this register is 01b, then the window coefficients are applied as is from the window RAM for the first iteration, and then linearly interpolated between successive coefficients with an interpolation fraction of 0.25, 0.5, 0.75 for the second, third and fourth iterations respectively. This corresponds to the linear interpolation of window coefficients as needed for a 4K size FFT. (When performing 2K size FFT, the WINDOW_INTERP_FRACTION register should be programmed to 10b, in which case, the first iteration uses the window RAM coefficients as is, and the second iteration linearly interpolates between consecutive coefficients with interpolation fraction of 0.5). Note that when linear interpolation for the window coefficients is used, the symmetric window mode (WINSYMM = 1) cannot be used.

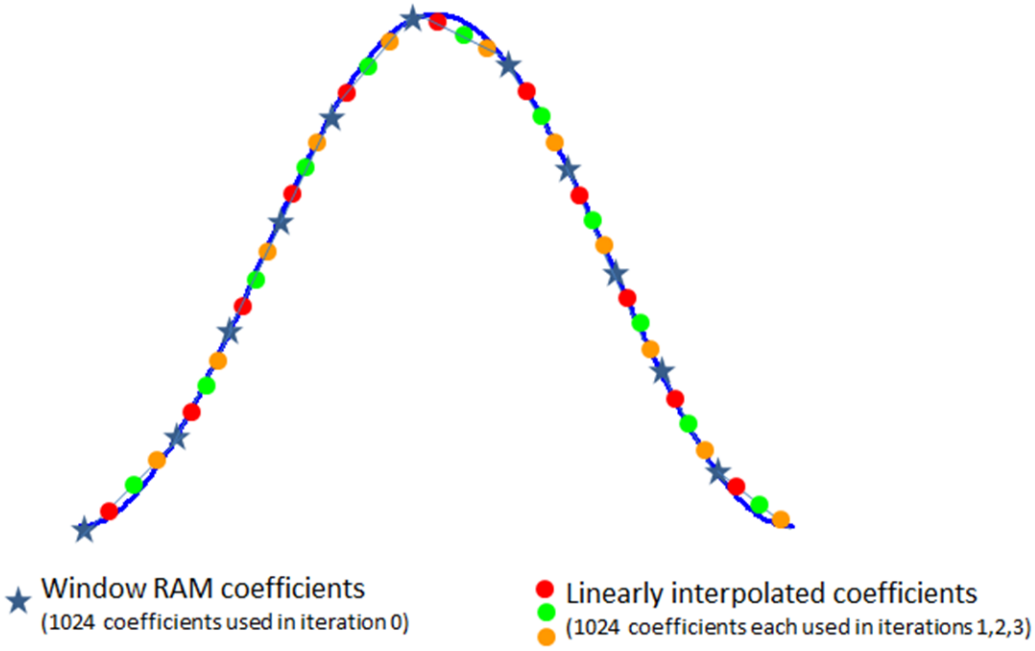

Figure 13. Linear Interpolation of Window RAM Coefficients for 4K FFT

Windowing output is fed to FFT engine for 1024-point FFT computation (see Step 1). At the end of this step, the complex FFT output is stored in ACCEL_MEM2 in the same fashion as they are picked from source memory as illustrated in Figure 14. Note that $I_{i,j}$ represent real part of i^{th} bin FFT output for j^{th} iteration. Similarly, $Q_{i,j}$ represent imaginary part of i^{th} bin FFT output for j^{th} iteration.

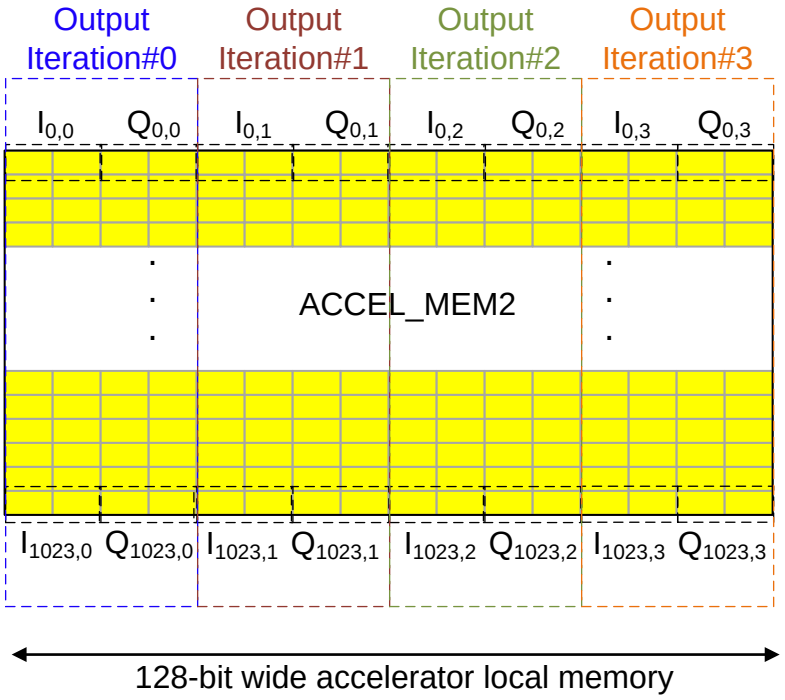

Figure 14. Layout of Samples in Destination Memory (after Step 1)

2. In the second step, 1024 4-point FFTs in stitching mode are computed and the results written back to ACCEL_MEM0, thus, over-writing the original input data. For 4-point FFT computation, the input samples are sent through the FFT engine in a transpose manner as illustrated in Figure 15. In this case, the complex multiplier in the pre-processing block needs to be configured to enable 4K FFT stitching. After each 4-point FFT, the output samples correspond to FFT bins spaced apart by 1024. For example, the first iteration produces outputs corresponding to bins 0, 1024, 2048 and 3072. Similarly, the second iteration produces outputs corresponding to bins 1, 1025, 2049 and 3073. By using appropriate DSTAINDX and DSTBINDX settings, the output samples can be arranged in the correct bin order as desired.

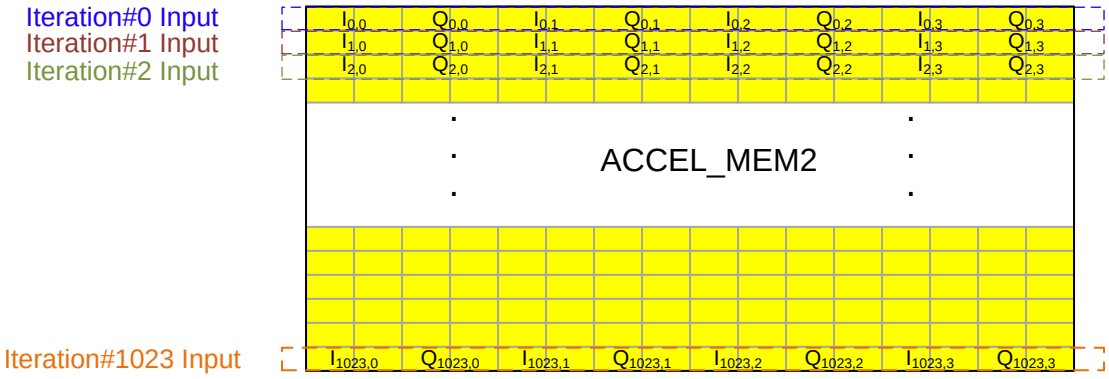

Figure 15. 1024 4-Point FFTs in Step 2 (FFT Stitching)

The key register configurations to perform 4K size complex FFT using FFT stitching are tabulated in Table 8.

Table 8. Parameter-Set #0: Used for Step #1

Register	Value	Comments
FFT_EN	1	Enable FFT computation
FFTSIZE	10	FFT size = 2^10 = 1024
WINDOW_EN	1	Enable windowing
WINDOW_INTERP_ FRACTION	1	Enable windowing interpolation for 4096-point FFT stitching
SRCACNT	1023	1024 valid samples
SRCAINDX	16	Adjacent samples spaced 16 bytes apart
REG_BCNT	3	Four back-to-back 1024-point FFT
SRCBINDX	4	Samples are 4 bytes apart for successive iterations
SRCADDR	0	Start at beginning of ACCEL_MEM0
DSTADDR	32KB	Destination memory is ACCEL_MEM2
DSTAINDX	16	
DSTBINDX	4	
DSTACNT	1023	
SRC16b32b	0	FFT input samples are 16-bit word aligned
DST16b32b	0	FFT output samples are 16-bit word aligned

Table 9. Parameter-Set #1: Used for Step #2

Register	Value	Comments
FFT_EN	1	Enable FFT computation
FFTSIZE	2	FFT size = 2^2 = 4
CMULT_MODE	3	Enables FFT Stitching Mode of complex multiplier
TWIDINCR	1	4096-Point FFT Stitching
SRCACNT	3	Four valid samples (zero-based count)
SRCAINDX	4	Adjacent samples spaced 4 bytes apart
REG_BCNT	1023	Need to compute 4-point FFT 1024 times
SRCBINDX	16	Each set for 4–point FFT are spaced 16 bytes apart
SRCADDR	32KB	Input samples for this step are stored in ACCEL_MEM2
DSTADDR	0KB	Output of this step stored back in ACCEL_MEM0
DSTAINDX	1024*4	Adjacent output samples of each iteration are 4KB apart, so that at the end, 4096-point FFT is output in linear bin order
DSTBINDX	4	First output sample of each iteration is 4 bytes apart to ensure final 4096-point FFT output is in linear bin order
DSTACNT	3	4-point FFT output
SRC16b32b	0	FFT input samples are 16-bit word aligned
DST16b32b	0	FFT output samples are 16-bit word aligned

At the end of two parameter sets, the result of the 4096-point FFT is available in ACCEL_MEM0 in correct bin order and can be transferred back to L3 memory via DMA. Alternately, log-magnitude and/or CFAR detection processing can be done in the radar hardware accelerator itself.

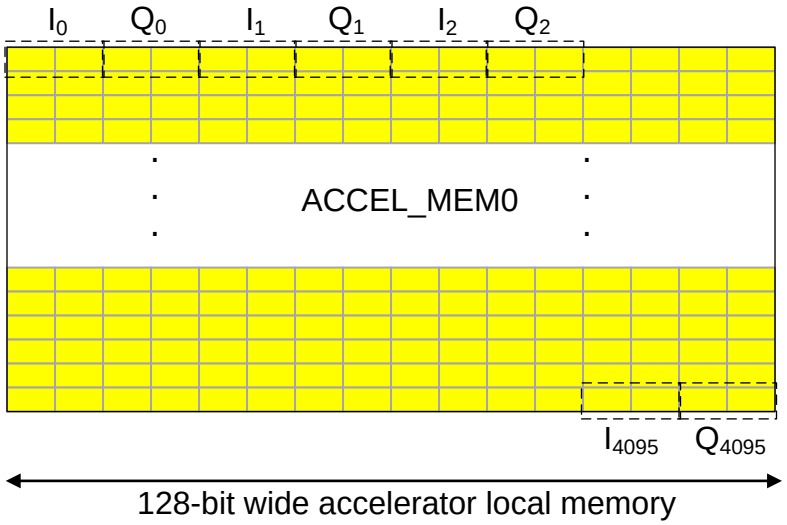

Figure 16. Layout of Final FFT Output in Correct Bin Order

5 References

Radar Hardware Accelerator User's Guide - Part 1

Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from Original (May 2017) to A Revision	Page
• Update was made in Section 2.1.	9

IMPORTANT NOTICE FOR TI DESIGN INFORMATION AND RESOURCES

Texas Instruments Incorporated ('TI") technical, application or other design advice, services or information, including, but not limited to, reference designs and materials relating to evaluation modules, (collectively, "TI Resources") are intended to assist designers who are developing applications that incorporate TI products; by downloading, accessing or using any particular TI Resource in any way, you (individually or, if you are acting on behalf of a company, your company) agree to use it solely for this purpose and subject to the terms of this Notice.

TI's provision of TI Resources does not expand or otherwise alter TI's applicable published warranties or warranty disclaimers for TI products, and no additional obligations or liabilities arise from TI providing such TI Resources. TI reserves the right to make corrections, enhancements, improvements and other changes to its TI Resources.

You understand and agree that you remain responsible for using your independent analysis, evaluation and judgment in designing your applications and that you have full and exclusive responsibility to assure the safety of your applications and compliance of your applications (and of all TI products used in or for your applications) with all applicable regulations, laws and other applicable requirements. You represent that, with respect to your applications, you have all the necessary expertise to create and implement safeguards that (1) anticipate dangerous consequences of failures, (2) monitor failures and their consequences, and (3) lessen the likelihood of failures that might cause harm and take appropriate actions. You agree that prior to using or distributing any applications that include TI products, you will thoroughly test such applications and the functionality of such TI products as used in such applications. TI has not conducted any testing other than that specifically described in the published documentation for a particular TI Resource.

You are authorized to use, copy and modify any individual TI Resource only in connection with the development of applications that include the TI product(s) identified in such TI Resource. NO OTHER LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE TO ANY OTHER TI INTELLECTUAL PROPERTY RIGHT, AND NO LICENSE TO ANY TECHNOLOGY OR INTELLECTUAL PROPERTY RIGHT OF TI OR ANY THIRD PARTY IS GRANTED HEREIN, including but not limited to any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information regarding or referencing third-party products or services does not constitute a license to use such products or services, or a warranty or endorsement thereof. Use of TI Resources may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

TI RESOURCES ARE PROVIDED "AS IS" AND WITH ALL FAULTS. TI DISCLAIMS ALL OTHER WARRANTIES OR REPRESENTATIONS, EXPRESS OR IMPLIED, REGARDING TI RESOURCES OR USE THEREOF, INCLUDING BUT NOT LIMITED TO ACCURACY OR COMPLETENESS, TITLE, ANY EPIDEMIC FAILURE WARRANTY AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

TI SHALL NOT BE LIABLE FOR AND SHALL NOT DEFEND OR INDEMNIFY YOU AGAINST ANY CLAIM, INCLUDING BUT NOT LIMITED TO ANY INFRINGEMENT CLAIM THAT RELATES TO OR IS BASED ON ANY COMBINATION OF PRODUCTS EVEN IF DESCRIBED IN TI RESOURCES OR OTHERWISE. IN NO EVENT SHALL TI BE LIABLE FOR ANY ACTUAL, DIRECT, SPECIAL, COLLATERAL, INDIRECT, PUNITIVE, INCIDENTAL, CONSEQUENTIAL OR EXEMPLARY DAMAGES IN CONNECTION WITH OR ARISING OUT OF TI RESOURCES OR USE THEREOF, AND REGARDLESS OF WHETHER TI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

You agree to fully indemnify TI and its representatives against any damages, costs, losses, and/or liabilities arising out of your non-compliance with the terms and provisions of this Notice.

This Notice applies to TI Resources. Additional terms apply to the use and purchase of certain types of materials, TI products and services. These include; without limitation, TI's standard terms for semiconductor products http://www.ti.com/sc/docs/stdterms.htm), evaluation modules, and samples (http://www.ti.com/sc/docs/sampterms.htm).

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265