Application Note

Improving mmWave Radar Performance Using Hardware Accelerator

Allen Yin *China Central FAE*

ABSTRACT

This document was translated from a simplified Chinese source. (ZHCAFE5)

TI's mmWave radar chip builds a highly collaborative computing architecture by integrating an MCU, DSP, and hardware accelerator (HWA). Such an architecture significantly reduces the computational burden on the MCU and DSP while increasing overall system efficiency and flexibility. Flexibility and full utilization of the HWA are key to improving system efficiency. This article will explore how to achieve comprehensive improvements in system performance by efficiently leveraging the MCU, DSP, and HWA in terms of system design, parallel computing, and programming optimization.

Revision History

DATE	REVISION	NOTES
May 2025	*	Initial Release

Table of Contents

Revision History 1
1 Introduction 2
2 Introduction to Hardware Accelerator (HWA) 3
2.1 HWA Functional Modules and Version Differences 4
2.2 Core Computational Unit 5
3 HWA Use Case: Matrix Multiplication 7
4 HWA Usage in mmWave Radar Link 11
4.1 RangeProcDDMA 11
4.2 DopplerProcDDMA 12
4.3 RangeCFARprocDDMA 13
5 Summary 13
6 References 13

List of Figures

Figure 1-1. TI's xWRL6432 mmWave Radar Chip 3
Figure 2-1. Hardware Accelerator System 4
Figure 2-2. HWA1.2 Core Computational Unit System 6
Figure 3-1. Complex Multiplication Functional Module 8
Figure 4-1. Range DDMA Flow 12
Figure 4-2. Doppler DDMA Flow 12

List of Tables

Table 2-1. Function Comparison by HWA Version 5
Table 3-1. xWRL6432 Matrix Multiplication Performance Comparison 11

1 Introduction

The mmWave radar chip family from Texas Instruments (TI) has become a core technology solution in the intelligent driving perception space due to its excellent performance, high integration, and automotive-grade reliability. Its representative products include AWR1843, AWR1642, AWR6843, AWR2944, AWRL1432, and AWRL6432, covering the full range of scenarios from basic advanced driver assistance systems (ADAS) to higher-level autonomous driving. The AWR1843 is built around a 3Tx4Rx configuration and FMCW technology, featuring an integrated Cortex-M4+DSP and a hardware accelerator (HWA), thus combining excellent cost-effectiveness with flexible scalability. The AWR2944, as a second-generation flagship chip, is fabricated using a 45nm RFCMOS process and supports 4Tx and 4Rx channels with a 5GHz bandwidth. With a hardware security module (HSM) and a lockstep Cortex-R5F+DSP+HWA architecture, it addresses the high-accuracy requirements of L3 autonomous driving for forward and corner radars. Optimized for short-range perception scenarios, the AWRL1432 integrates an RF subsystem with a low-power design, delivering rapid response in autonomous parking, door collision avoidance, and other applications. The AWR6843 and AWRL6432, as 60GHz mmWave radar chips, are widely used in in-cabin applications such as child presence detection, seat belt reminder, and anti-intrusion applications.

The IWR1843, IWR6843, IWRL1432, and IWRL6432 families are single-chip mmWave radar sensors targeted at the industrial and consumer electronics segments. Among them, the IWR1843 operates in the 76-81GHz band, integrates a DSP, MCU, and radar hardware accelerator, and is suitable for high-precision industrial radar detection scenarios. The IWR6843 covers the 60-64GHz band and integrates processing capabilities, enabling smart applications such as human movement tracking, vital sign monitoring, and fall detection. The IWRL1432 and IWRL6432 both feature a low-power design. The former operates at 76-81GHz, suitable for medium- to short-range detection in applications such as e-bike safety systems, and the latter operates in the 57-64GHz band, supporting edge computing and TinyML, thus being ideal for building automation, medical monitoring, and low-cost appliances. With highly integrated architectures, all these families combine resolution, responsiveness, and energy efficiency together to meet the diverse requirements of industrial control, consumer electronics, healthcare, and so forth.

TI’s mmWave radar sensors integrate an HWA, DSP, and MCU to achieve a breakthrough in both signal processing efficiency and flexibility. As a dedicated computing module, the HWA significantly reduces the computational load on the MCU and DSP by hardening key algorithms, optimizing the data flow architecture, and implementing parallel processing mechanisms. It provides core support for high resolution, low-power performance, and real-time response of radar systems. TI's mmWave radar chips also use an HWA as a dedicated signal processing engine, which is designed to implement standardized algorithms (such as FFT, CFAR detection, log magnitude computation, and so forth) in hardware that are traditionally executed by the DSP or MCU, thereby freeing up master processor resources for higher-level data processing (such as target classification, AI fusion, and so forth).

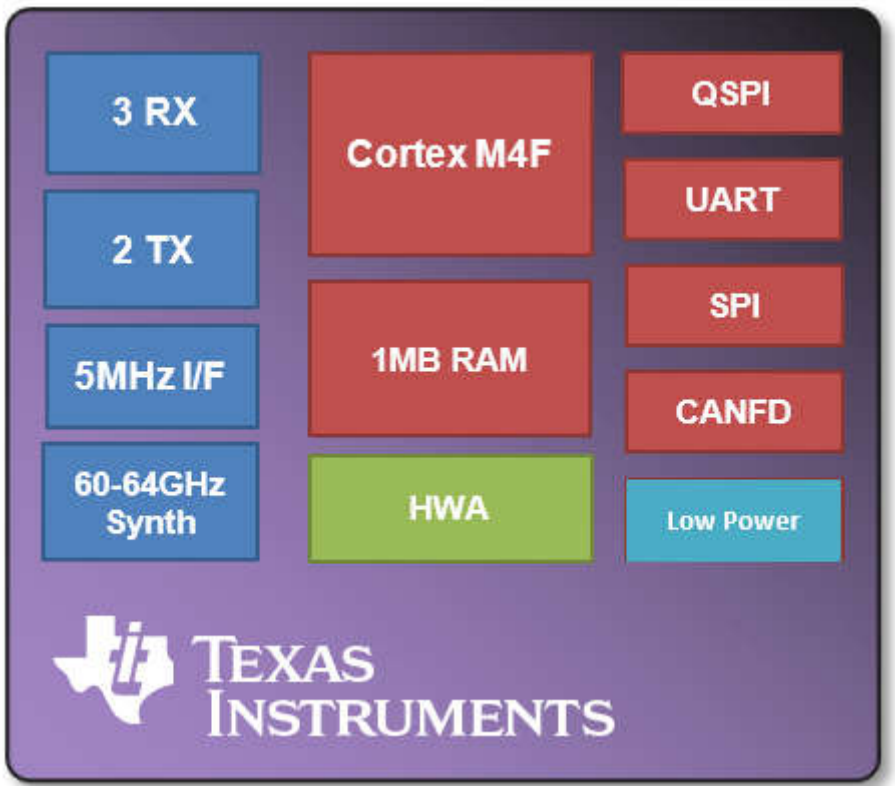


Figure 1-1. TI's xWRL6432 mmWave Radar Chip

The HWA features a modular hardware design with an integrated cache and computational units. The integrated cache supports ping-pong input/output, allowing DMA read and write to run in parallel with computation, thereby achieving zero-wait data transfers. It can also connect directly to the ADC buffer to acquire raw data instantly for FFT execution, which avoids bottlenecks caused by main memory access. The computational unit contains typical algorithms such as FFT and CFAR detection, integrates the magnitude/log magnitude computation module, and supports data compression and decompression to meet dynamic range compression needs. In terms of energy efficiency, for algorithms such as FFT/CFAR, the HWA is optimized to achieve an actual efficiency close to or even higher than that of a DSP implementation. Therefore, how to effectively use HWA resources in more computing scenarios is crucial for improving the cost-effectiveness and power efficiency of mmWave radars.

2 Introduction to Hardware Accelerator (HWA)

The HWA and the mmWave radar chip processor work in parallel, as shown in the block diagram below. The HWA connects to other systems via a 128-bit bus (64-bit bus for HWA version 1.2). HWA2.1 of the AWR2944, for example, contains an accelerator engine and eight 16KB memories, which are used to provide input and output data to the accelerator's computation engine. The HWA does not have direct access to memory units external to the accelerator. Data can only be transferred to or from the HWA memory via DMA or the processor to facilitate data exchange. The internal memory is divided into separate 16KB units, allowing simultaneous reads and writes via DMA or the processor, and uses a ping-pong buffering mechanism to process data. The HWA internal access bus is 128 bits wide, allowing a throughput of 128 bits per clock cycle.

It is important to note that any 16KB HWA local memory unit can serve as the source address for the accelerator engine's input data, and any of them can serve as the destination address for the output data, with one important limitation: The source and destination addresses cannot be the same 16KB unit. In other words, the same memory unit cannot be used for overwrite operations during a single computation. It is also important to note that the accelerator's local memories do not mandatorily require ping-pong mode. If the application scenario demands, two or four memory units can be combined into a larger 32KB or 64KB input and output unit. The HWA internal access to these memories starts at address 0x0000.

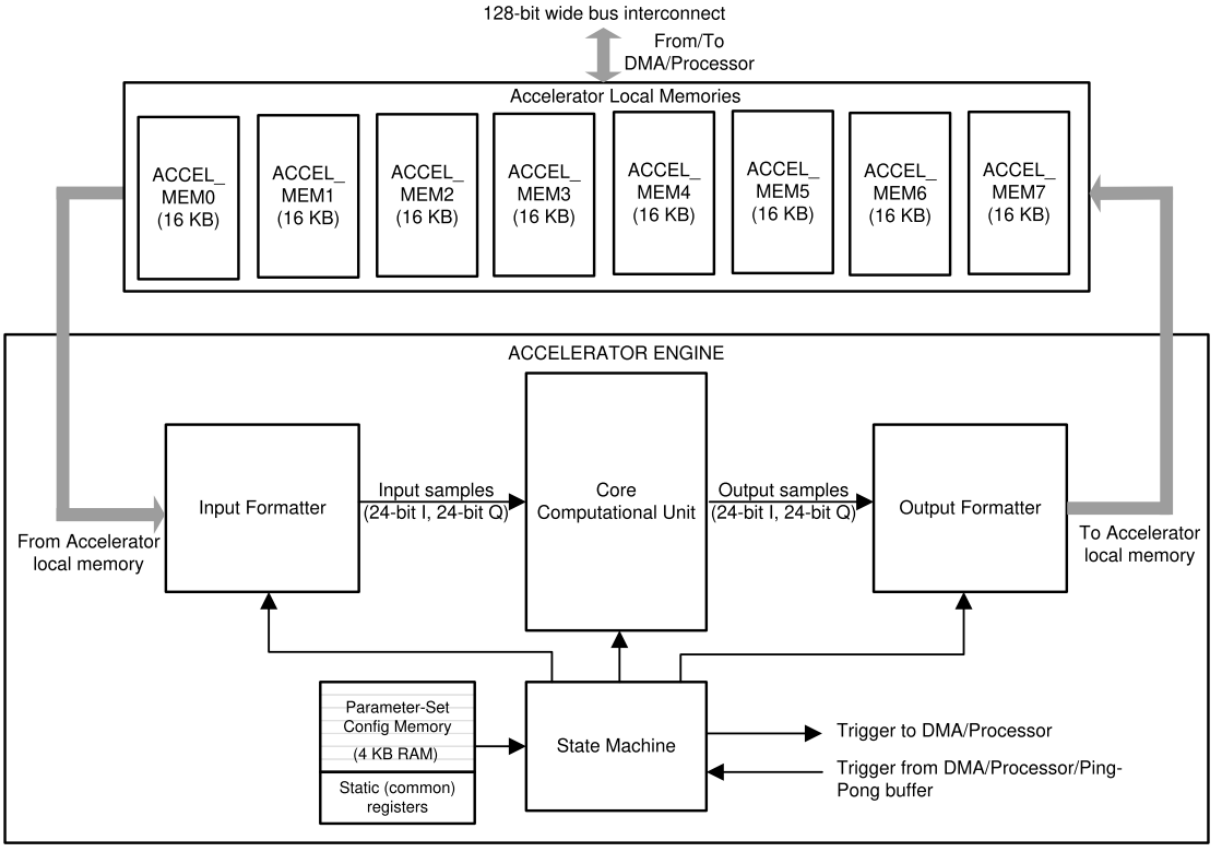


Figure 2-1. Hardware Accelerator System

2.1 HWA Functional Modules and Version Differences

The functional modules in the processing section of the HWA primarily include:

- State Machine:

  The State Machine controls the overall operation of the HWA, including initiation, looping, and stopping. It also controls the triggering and handshaking mechanisms among the HWA, DMA, and the master processor. In addition, it reads preset programs from the parameter set configuration memory and executes them in sequence. The State Machine can be configured to iterate through the parameter sets in a loop, executing a total of NUMLOOPS times (when the NUMLOOPS parameter is set to 0 or 0xFFF, the loop will not run or will run indefinitely, respectively). For example, to configure the state machine to run the first four parameter sets 64 times in a loop, the register should be set as follows: PARAM_START_IDX = 0, PARAM_END_IDX = 3, and NUMLOOPS = 64. The initiation of the State Machine can be triggered by software, hardware events, or DMA events. Similarly, the end of the State Machine can generate an interrupt event or a DMA event.
- Input Formatter:

  It reads data from the HWA local memory and feeds it into the core computational unit. During this process, the Input Formatter provides flexible access, including reads in 16-bit or 32-bit aligned data formats. Users can configure the number of bits to truncate or pad with zeros. This module also supports two-dimensional transposed data reads, along with flexible scaling and sign extension (converting the data to a 24-bit internal bit width). Finally, the module feeds the 24-bit complex samples into the core computational unit. Within this module, users can configure zero-padding for the input data. For example, when the length of the input data for FFT is 56 but a 64-point FFT is required, the Formatter will add zeros to the end of the original data before each FFT operation to meet the FFT data length requirements. Note that the configured input data length should not exceed the number of FFT points to be computed.
- Output Formatter:

This module writes data from the core computational unit to the local memory and provides various formatting options, such as writing in 16-bit or 32-bit aligned output data formats, two-dimensional transposed data writes, and scaling data from a 24-bit internal bit width to 16-bit or 32-bit. Users can configure the number of bits to truncate or pad with zeros, as well as sign extension, offering flexibility to fit the needs of different hardware interfaces.

When outputting data, users can also choose to skip a portion of the data at the beginning or the end. For example, after an FFT operation, if only the middle portion of the data is required, the registers can be configured to skip the data at the beginning and the end.

- Core Computational Unit:

  This unit performs computations such as windowing, FFT, magnitude calculation, logarithmic operation (log2), and constant false alarm rate (CFAR) detection. It receives data in a streaming input manner (one sample per clock cycle) and, after computation, delivers results via a streaming output.
- Parameter-Set Configuration Memory:

  It is used to pre-configure the parameter sets (register settings) for accelerator operations, allowing the State Machine to loop through a sequence of operations according to the program.

Different mmWave radar chips use different HWA versions, which differ in functionality. The mmWave radar chip and HWA pairs are as follows:

- xWR1443/1642/1843 – HWA 1.0
- xWR6843 – HWA 1.1
- xWR1432/6432 – HWA 1.2
- xWR2544 – HWA 1.5
- xWR2944/2E44 – HWA 2.1

These different versions' functions are listed in the following table:

Table 2-1. Function Comparison by HWA Version

	HWA1.0/1.1	HWA1.2	HWA1.5	HWA2.0/2.1
Frequency	200MHz	80MHz	300MHz	300MHz/400MHz
FFT	1024, 512, 256… (Radix-2)	1024,512,56… (Radix-2)	2048,1536,1024, 768, 512, 384… (Radix-2 & 3)	2048,1536,1024, 768, 512, 384… (Radix-2 & 3)
Parameter set	16	32	32	64
Complex multiplication	7-mode	7-mode	3-mode	10-mode
Interference mitigation	Interference mitigation using a fixed threshold	Interference mitigation using a fixed threshold	Interference mitigation or interpolation using multiple statistics	Interference mitigation or interpolation using multiple statistics
Logarithmic magnitude	Supported (0.3dB accuracy)	Supported (0.3dB accuracy)		Supported (0.02dB accuracy)
Data compression/ decompression	Not supported	Supported	Compression supported only	Supported
CFAR	CFARCA	CFARCA, CFAROS	Not supported	CFARCA, CFAROS
Statistics	1D array summation and maximum value search	1D array summation and maximum value search	Not supported	1D array summation and maximum value search, histogram, 2D maximum value search, CDF
Others	None	None	None	Channel merging, zero padding, context switching, and so forth

2.2 Core Computational Unit

The core computational unit (CCU) is the most important component of the HWA. In HWA1.2, the CCU plays three primary roles: the FFT engine, the CFAR engine, and the compression engine. Only one role is enabled at one time. With different parameter set configurations, the processing flow of different engines can be executed in a sequence, allowing multiple parameter sets to be executed in a sequence to complete the required series

of computations. The register ACCEL_MODE controls which processing engine is actually enabled for a specific parameter set.

The block diagram of the CCU system for HWA1.2 is shown below:

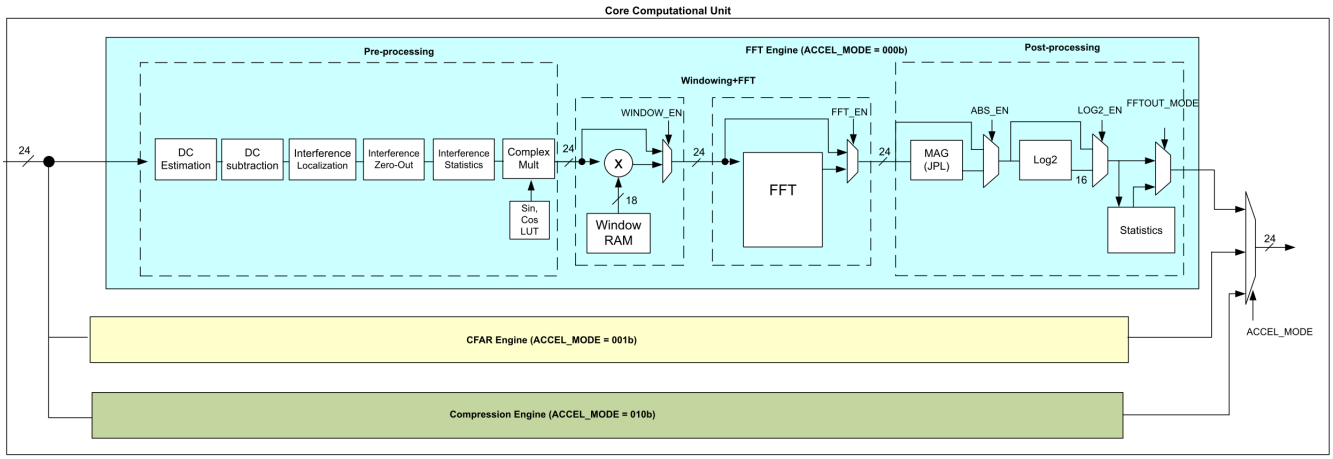


Figure 2-2. HWA1.2 Core Computational Unit System

When the CCU enables the FFT engine, the pre-FFT windowing operation can also be configured. Users can configure their own windowing coefficients through the Window RAM.

After the FFT, the CCU supports modulo (absolute value) and logarithmic operations for easier radar signal post-processing. Prior to the FFT, the CCU supports a number of pre-operations, including DC estimation and subtraction, interference localization and mitigation, interference statistics, complex multiplication, and so forth. Each module can be separately enabled for maximum flexibility.

The complex multiplication module of the FFT engine can not only pre-process data (such as rotation, multiplication by scalars, vectors, and so forth) in conjunction with the FFT engine before the FFT, but can also run as a separate functional module, performing complex vector multiplication or summation functions within the CCU (with the FFT engine disabled). The function of this module is briefly described below. For more details and methods of operation, please refer to References 3 and 4

- Frequency Shifter Mode, where input data is shifted by a certain frequency through complex multiplication;
- DFT Mode, where DFT computations are performed through the frequency shifter and auto-increment functions;
- FFT Stitching Mode, which supports the FFT of more dots, such as 2048, 4096, 8192, and so forth;
- Magnitude-Squared Mode, which calculates the squared magnitude of the input data and works with the final Statistics module to calculate the sum of the squared magnitude;
- Scalar Multiplication Mode, which multiplies the input data by different scalar values;
- Vector Multiplication Mode 1, which calculates the product of two vectors and can perform dot-product operations in conjunction with the Statistics module's summation function;
- Vector Multiplication Mode 2, which differs slightly from Vector Multiplication Mode 1 in that Mode 1 restarts counting from Location 0 in Internal RAM after every iteration of computation.

The following three complex multiplication modes are only implemented on HWA 2.0:

- Recursive Windowing Mode, where the windowing operation is superimposed with frequency shifting. The input data is windowed using the following equation: $Wk(n) = W0(n) * \exp(-j*K*\theta(n)*2*pi/16384)$, where the K value can be either reset to 0 or carried over from the last one on each iteration according to user configurations;
- Look-up Table-Based Frequency and Phase Derotation Mode, which is similar in function to the Frequency Shifter Mode, but adds the capabilities of programmable RAM-based frequency derotation and phase derotation;
- Frequency Shifter Mode With Fine Frequency Increment, which is an extension of the Frequency Shifter Mode and adds a signed 10-bit trim value in addition to the existing frequency shifting capability. The trim

value is superimposed on the derotation frequency, making it possible for the derotation frequency to vary incrementally after each loop iteration.

In addition, the CCU supports operations such as BPM demodulation, channel merging, and zero interpolation (note the difference with FFT's zero padding) within the FFT engine.

At the end of the FFT engine processing, the CCU allows statistical processing of the output data with its integrated Statistics module, which supports functions such as obtaining the maximum value of the input data or summation. HWA2.0 and the later support maximum value search on two-dimensional data, as well as providing histograms, CDF, and more statistical information.

If enabled by the CCU, the CFAR engine supports CFAR-CA and CFAR-OS, and supports both linear and logarithmic CFAR detection processing, ultimately generating peak lists. The CFAR engine configures various parameters of the CFAR algorithm. See also the TRM for detailed parameters.

To maximize system memory utilization, the CCU can be configured as a compression/decompression engine for radar FFT data compression, which supports Block Floating Point (BFP) and Exponential Golomb Encoder (EGE) compression modes. Generally speaking, the BFP mode is suitable for compressing consecutive points in the range dimension, which is faster and consumes less memory. In contrast, the EGE mode requires analysis of the data to be compressed before compression and is suitable for compressing data blocks composed of multiple antennas and multiple range dimensions. Although slower than the BFP mode, theoretically, the EGE mode retains more signal details after compression/decompression.

In the HWA versions 2.0/2.1, the CCU provides the Local Maxima engine, which determines that a detection bin is a local maxima by determining that its magnitude is greater than or equal to the magnitude of all its adjacent samples. The selection of adjacent samples is configured via registers. Users can also define thresholds for rows and columns to filter the bins to be detected.

3 HWA Use Case: Matrix Multiplication

This chapter provides a simple example of how to use the hardware accelerator (HWA) flexibly.

In mmWave radar signal processing systems, complex matrix multiplication is often used as the core algorithm, and its computational efficiency directly affects the real-time performance of the system. TI's mmWave radar processor architecture typically relies on a DSP core for complex matrix multiplication. However, in the second generation (xWRL1432/6432, AWR2944LC, and so forth), power consumption and cost considerations have driven the use of HWAs to undertake the main signal processing tasks, while a low-power MCU core (ARM Cortex-M4/R5) is adopted for system control and data management. This architecture reduces both power consumption and cost, but it also introduces new technical challenges: Traditional matrix operation schemes using MCUs and CMSIS libraries present significant performance bottlenecks when the processor is not configured with a dedicated DSP core. Their single-threaded serial execution mode results in increased compute latency, and the MCU resource occupation affects the system's parallel multitasking capability.

In this context, the utilization of idle HWA resources becomes an entry point for optimization. As a dedicated hardware accelerator module, the HWA enables complex vector product computations. However, to perform complex matrix multiplication, row and column data must be constantly fed for computation. Therefore, common approaches require MCU intervention for data management and controlled input and output. This section describes the implementation of matrix multiplication without MCU intervention using the EDMA in conjunction with the HWA. It also provides examples demonstrating how to flexibly coordinate data copying, computation, and triggering between the EDMA and various HWA modules. Finally, it presents the performance tested on the xWRL6432 chip.

The following provides a 4×4 matrix multiplication example to illustrate the implementation on the xWRL6432. When Matrix A (4 rows × 4 columns) is multiplied by Matrix B (4 rows × 4 columns), the resulting Matrix C has dimensions of 4 rows × 4 columns. The element at Row i and Column j of Matrix C is calculated as follows, or as the sum of the products of the corresponding elements from Row i of Matrix A and Column j of Matrix B

$$C_{\{i,j\}} = \sum_{k=0}^{3} A_{\{i,k\}} \times B_{\{k,j\}} \quad (1)$$

For example, the first element of Matrix C is calculated as follows:

$$C_{0,0} = A_{0,0} \times B_{0,0} + A_{0,1} \times B_{1,0} + A_{0,2} \times B_{2,0} + A_{0,3} \times B_{3,0} \quad (2)$$

Other matrix elements are also calculated in this way, and each element of Matrix C requires one complex dot product computation via the HWA. A 4×4 matrix thus requires a total of 16 dot product operations.

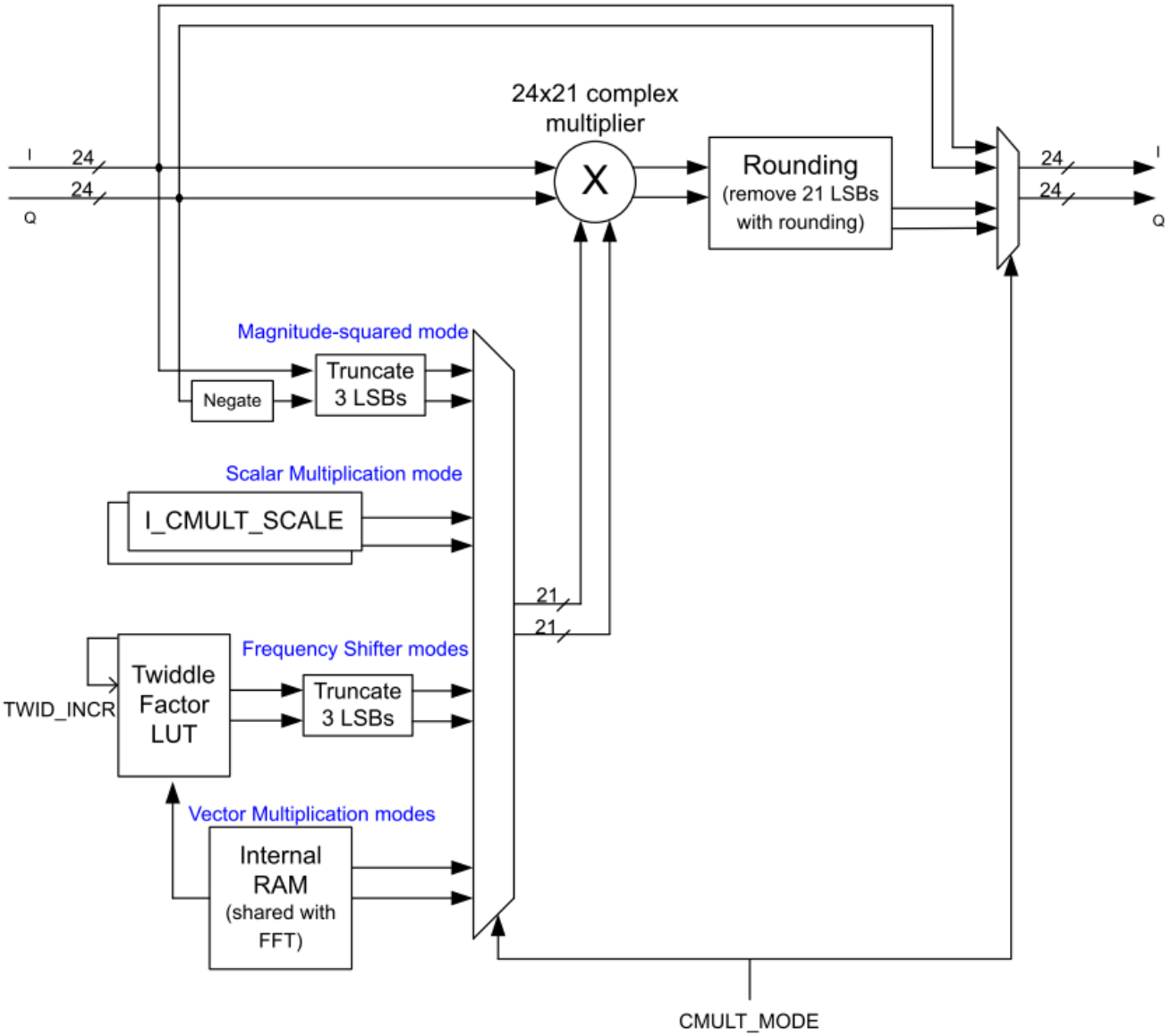
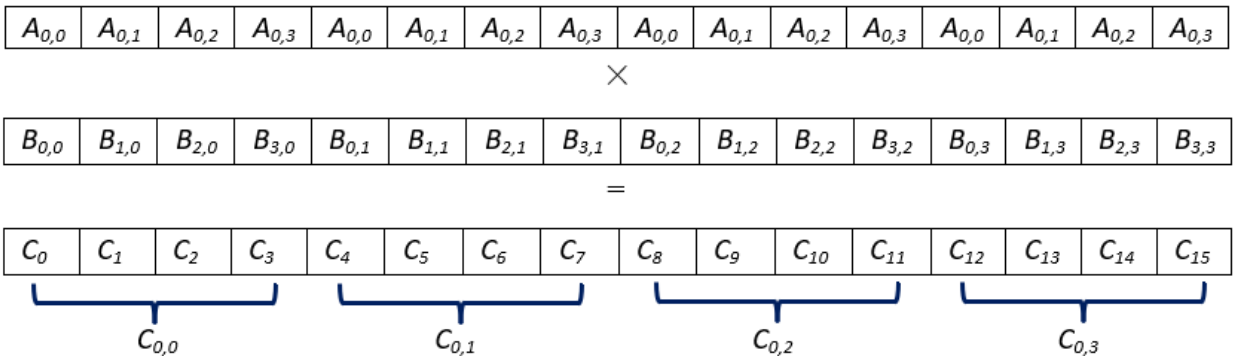


Figure 3-1. Complex Multiplication Functional Module

According to the HWA input method for complex dot product computation, one complex vector is input normally through the Input Formatter, while the other complex vector is written to the Internal RAM, as shown in the figure above. It is important to note that the vector input through the Input Formatter uses 24-bit accuracy, whereas the vector written to the Internal RAM uses 21-MSB precision, where the 21 LSBs are discarded after the multiplication of the two numbers, resulting in a 24-bit complex number. Therefore, necessary scaling and shifting operations must be performed on the input data to prevent overflow caused by the final result being too large or too small.

Next, consider how to convert the two-dimensional matrix computation into a one-dimensional vector computation supported by the HWA. When computing the first row of Matrix C, perform dot products of the first row of Matrix A with Columns 1 to 4 of Matrix B. To do this, copy the first row of Matrix A four times to form a one-dimensional vector of 16 elements. Meanwhile, concatenate the four columns of Matrix B in the column

direction to form another one-dimensional vector of 16 elements. Multiply these two vectors to obtain 16 complex results, which are then added up in groups of four elements to produce the first row of Matrix C.



The remaining three rows are computed in the same way.

Based on the HWA's implementation for vector multiplication, Matrix B is written column-wise into the Internal RAM, with a total of 16 complex values written. To make the operation completely independent of the MCU, the process is divided into three steps using two HWA parameter set configurations

1. Use the EDMA to copy each row of Matrix A four times and transfer them to the input memory unit of the HWA. Upon completion, use the Link method to proceed to the next step;
2. Using the EDMA Link method, initiate the HWA in DMA2ACC trigger mode to perform vector multiplications. Each vector multiplication results in 16 elements, and a total of four vector multiplications result in 64 complex elements;
3. Use the summation function of the HWA's Statistics module to add every four complex elements, producing the final matrix result.

The EDMA is configured as follows: The first set of parameters is used for copying Matrix A to the HWA memory, automatically triggering and completing the matrix copy and transfer via EDMA AB sync mode and Chain events. It then uses the Link method and completion events to trigger the execution of the second parameter set, which copies the trigger word from Channel 1 of the HWA to the trigger register address of the DMA2ACC to initiate the HWA.

```
/* Initiate EDMA to HWA */
rdEdmaPrms->srcAddr = (uint32_t) (src); //Matrix A source address
rdEdmaPrms->destAddr = (uint32_t) (rdMemBankAddr); //HWA memory bank
rdEdmaPrms->aCnt = (uint16_t) (COLUMN *  sizeof  (cmplx32ImRe_t)); //One row data size
rdEdmaPrms->bCnt = (uint16_t) COLUMN; // Copy one row for Column times
rdEdmaPrms->cCnt = (uint16_t) ROW; // Load all ROWs
rdEdmaPrms->srcBIdx = (int16_t) 0;
rdEdmaPrms->destBIdx =(int16_t) (COLUMN *  sizeof  (cmplx32ImRe_t));
rdEdmaPrms->srcCIdx = (int16_t) (COLUMN *  sizeof  (cmplx32ImRe_t));
rdEdmaPrms->destCIdx = (int16_t) (ROW*COLUMN*  sizeof  (cmplx32ImRe_t));
rdEdmaPrms->opt = EDMA_OPT_SYNCDIM_MASK
| EDMA_OPT_TCCHEN_MASK
| EDMA_OPT_ITCCHEN_MASK
| ((((uint32_t)resObj->rdTcc) << EDMA_OPT_TCC_SHIFT) & EDMA_OPT_TCC_MASK); //AB sync
 EDMAChainChannel  (baseAddr, 1, 1, EDMA_OPT_ITCCHEN_MASK);
hwaEdmaPrms->srcAddr = (uint32_t) (0x550100A0);//hwa signal done channel 1 code address
hwaEdmaPrms->destAddr = (uint32_t) (0x55010004); //DMA2ACC trigger address
hwaEdmaPrms->aCnt = (uint16_t) (4);
hwaEdmaPrms->bCnt = (uint16_t) 1;
hwaEdmaPrms->cCnt = (uint16_t) 1;
hwaEdmaPrms->srcBIdx = (int16_t) 0;
hwaEdmaPrms->destBIdx = (int16_t) 0;
hwaEdmaPrms->opt = (((uint32_t)resObj->rdTcc) << EDMA_OPT_TCC_SHIFT) &EDMA_OPT_TCC_MASK;
 EDMASetPaRAM  (baseAddr, resObj->rdParamId, rdEdmaPrms);
 EDMASetPaRAM  (baseAddr, hwaParamId, hwaEdmaPrms);
 EDMALinkChannel  (baseAddr, resObj->rdParamId, hwaParamId);
```

The HWA parameters are configured as follows: The first set of parameters uses the DMA trigger mode and is triggered by the EDMA event mentioned above. It then invokes the complex multiplication module to compute the results. Upon completion, it immediately triggers the second set of parameters, which uses the summation module to add up every four complex values to produce the final result.

```
/* Init param set */
paramCfg = &resObj->paramCfg;
 memset  (paramCfg , 0,  sizeof  (*paramCfg));
paramCfg->triggerMode = HWA_TRIG_MODE_DMA;
paramCfg->dmaTriggerSrc = 0;
paramCfg->accelMode = HWA_ACCELMODE_FFT;
paramCfg->source.srcAddr = HWADRV_ADDR_TRANSLATE_CPU_TO_HWA(resObj->rdMemBankAddr);
paramCfg->source.srcSign = HWA_SAMPLES_SIGNED;
paramCfg->source.srcAcnt = ROW*COLUMN - 1U;
paramCfg->source.srcAIdx =  sizeof  (cmplx32ImRe_t);
paramCfg->source.srcBcnt = COLUMN-1;
paramCfg->source.srcBIdx = ROW*COLUMN*  sizeof  (cmplx32ImRe_t);
paramCfg->source.srcWidth = HWA_SAMPLES_WIDTH_32BIT;
paramCfg->source.srcScale = 0x0;
paramCfg->source.srcRealComplex = HWA_SAMPLES_FORMAT_COMPLEX;
paramCfg->source.srcConjugate = HWA_FEATURE_BIT_DISABLE;
paramCfg->source.bpmEnable = HWA_FEATURE_BIT_DISABLE;
paramCfg->dest.dstAddr = HWADRV_ADDR_TRANSLATE_CPU_TO_HWA(resObj->wrMemBankAddr);
paramCfg->dest.dstSkipInit = 0U;
paramCfg->dest.dstAcnt = ROW*COLUMN - 1U;
paramCfg->dest.dstAIdx =  sizeof  (cmplx32ImRe_t);
paramCfg->dest.dstBIdx = ROW*COLUMN*  sizeof  (cmplx32ImRe_t);
paramCfg->dest.dstRealComplex = HWA_SAMPLES_FORMAT_COMPLEX;
paramCfg->dest.dstWidth = HWA_SAMPLES_WIDTH_32BIT;
paramCfg->dest.dstSign = HWA_SAMPLES_SIGNED;
paramCfg->dest.dstScale = 8U;
paramCfg->dest.dstConjugate = HWA_FEATURE_BIT_DISABLE;
paramCfg->accelModeArgs.fftMode.fftEn = HWA_FEATURE_BIT_DISABLE;
paramCfg->accelModeArgs.fftMode.windowEn = HWA_FEATURE_BIT_DISABLE;
paramCfg->accelModeArgs.fftMode.windowStart = 0U;
paramCfg->accelModeArgs.fftMode.winSymm = HWA_FFT_WINDOW_NONSYMMETRIC;
paramCfg->accelModeArgs.fftMode.fftSize = HWAFFT_log2Approx(numSamples);
paramCfg->accelModeArgs.fftMode.butterflyScaling = 0x0;
paramCfg->complexMultiply.mode = HWA_COMPLEX_MULTIPLY_MODE_VECTOR_MULT;
paramCfg->accelModeArgs.fftMode.magLogEn = HWA_FFT_MODE_MAGNITUDE_LOG2_DISABLED;
paramCfg->accelModeArgs.fftMode.fftOutMode = HWA_FFT_MODE_OUTPUT_DEFAULT;
 HWA_configParamSet  (fftObj->hwaHandle, resObj->paramIdx, paramCfg, NULL);
 memset  (paramCfg , 0,  sizeof  (*paramCfg));
paramCfg->triggerMode = HWA_TRIG_MODE_IMMEDIATE;
paramCfg->dmaTriggerSrc = 0;
paramCfg->accelMode = HWA_ACCELMODE_FFT;
paramCfg->source.srcAddr = HWADRV_ADDR_TRANSLATE_CPU_TO_HWA(resObj->wrMemBankAddr);
paramCfg->source.srcSign = HWA_SAMPLES_SIGNED;
paramCfg->source.srcAcnt = ROW - 1U;
paramCfg->source.srcAIdx =  sizeof  (cmplx32ImRe_t);
paramCfg->source.srcBcnt = ROW*COLUMN-1;
paramCfg->source.srcBIdx = ROW*  sizeof  (cmplx32ImRe_t);
paramCfg->source.srcWidth = HWA_SAMPLES_WIDTH_32BIT;
paramCfg->source.srcScale = 0x0;
paramCfg->source.srcRealComplex = HWA_SAMPLES_FORMAT_COMPLEX;
paramCfg->source.srcConjugate = HWA_FEATURE_BIT_DISABLE;
paramCfg->source.bpmEnable = HWA_FEATURE_BIT_DISABLE;
paramCfg->dest.dstAddr = HWADRV_ADDR_TRANSLATE_CPU_TO_HWA(hwa_fft_dst);
paramCfg->dest.dstSkipInit = 0U;
paramCfg->dest.dstAcnt = 1U;
paramCfg->dest.dstAIdx =  sizeof  (cmplx32ImRe_t);
paramCfg->dest.dstBIdx =  sizeof  (cmplx32ImRe_t);
paramCfg->dest.dstRealComplex = HWA_SAMPLES_FORMAT_COMPLEX;
paramCfg->dest.dstWidth = HWA_SAMPLES_WIDTH_32BIT;
paramCfg->dest.dstSign = HWA_SAMPLES_SIGNED;
paramCfg->dest.dstScale = 8U;
paramCfg->dest.dstConjugate = HWA_FEATURE_BIT_DISABLE;
paramCfg->accelModeArgs.fftMode.fftEn = HWA_FEATURE_BIT_DISABLE;
paramCfg->accelModeArgs.fftMode.windowEn = HWA_FEATURE_BIT_DISABLE;
paramCfg->accelModeArgs.fftMode.windowStart = 0U;
paramCfg->accelModeArgs.fftMode.winSymm = HWA_FFT_WINDOW_NONSYMMETRIC;
paramCfg->accelModeArgs.fftMode.fftSize = HWAFFT_log2Approx(ROW);
paramCfg->accelModeArgs.fftMode.butterflyScaling = 0x0;
paramCfg->complexMultiply.mode = HWA_COMPLEX_MULTIPLY_MODE_DISABLE;
paramCfg->accelModeArgs.fftMode.magLogEn = HWA_FFT_MODE_MAGNITUDE_LOG2_DISABLED;
```

```
paramCfg->accelModeArgs.fftMode.fftOutMode = HWA_FFT_MODE_OUTPUT_SUM_STATS;
 HWA_configParamSet  (fftObj->hwaHandle, resObj->paramIdx+1, paramCfg, NULL);
```

Throughout the flow, the EDMA parameter sets and the HWA parameter sets can be pre-written to save time for each operation. In each subsequent frame of computation, the MCU core only needs to complete the operation of writing Matrix B into the Internal RAM (Matrix B must be written in transposed form for computation) and then initiate the EDMA. After the EDMA automatically triggers the HWA to perform the computation, the HWA finally generates an interrupt to notify the MCU for further actions.

On the xWRL6432 chip, the MCU core M4F runs at 160MHz, and the HWA 1.2 runs at 80MHz. The cycle counts for fixed-dot complex matrix multiplication using the CMSIS library on the MCU versus the HWA are shown below. For ease of comparison, the values in the table are based on the cycle counts of the M4F.

Table 3-1. xWRL6432 Matrix Multiplication Performance Comparison

		Matrix size				
		4x4	6x6	8x8	12x12	16x16
		Cycle count				
M4F	CMSIS library	704	1856	3776	10960	23984
HWA	Loading Internal RAM	160	272	432	912	1600
	Performing matrix multiplication	784	1920	3904	12272	28064

Since the xWRL6432 HWA frequency is half that of the M4F, the HWA takes slightly longer than the CMSIS library for matrix computation. However, the M4F can perform other tasks in parallel with the HWA computation, effectively utilizing the HWA to free up MCU compute resources for parallel processing. On AWR2x44 series chips, the HWA has the same frequency as the MCU, and the cycle count for matrix computation using the HWA will be reduced to around 50% of the number shown in the table above, at which point the HWA reduces the processing latency for matrix computation.

4 HWA Usage in mmWave Radar Link

This chapter describes how the data processing unit (DPU) in TI's mmWave Radar SDK uses the HWA to process data, which allows users to learn more about how the HWA improves mmWave radar performance. The methods and processes for using the HWA within a DPU vary slightly with user scenarios, such as TDMA and DDMA, or chip variants, such as the presence or absence of a DSP core. Therefore, this chapter uses the DDMA flow on the common AWR2944 family as an example for illustration.

4.1 RangeProcDDMA

Generally, Range FFT processing is done by the HWA hardware. Based on the RF parameters, the RangeProcDDMA DPU performs operations by configuring the FFT engine while setting up EDMA channels to control data input and output. A ping-pong mode is adopted to facilitate parallel processing with raw data acquisition, with three or five sets of HWA parameters used on each side of the ping-pong buffer. To increase the maximum velocity measurement range, users can sometimes enable fast processing mode in this DPU. This mode uses the direct-current (DC) statistics and interference threshold estimates from the previous Chirp instead of calculating these parameters in real time. This mechanism saves the execution time for the two parameter sets (DC estimation/DC subtraction and interference statistics).

The flow is as follows:

1. Data input

   The sample data is first transferred from the ADC buffer to the HWA memory via the EDMA. In this example, an non-configured parameter set is used, with no data processed but only the triggering method configured (DMA or direct trigger by user software). The latter, direct trigger by user software, is primarily used for fast processing mode.
2. Pre-processing and FFT stages

   For non-fast processing modes, three parameter sets are executed sequentially. Parameter set A performs DC estimation, followed by Parameter set B, which performs DC subtraction and interference statistics

estimation. Parameter set C is finally invoked for interference mitigation and FFT computation. In fast processing mode, the DC and interference statistics obtained from the previous Chirp are used to manually update the threshold values, and DC subtraction, interference mitigation, and FFT computation are performed within the same parameter set.

3. Data compression

   The compression engine is configured to the BFP or EGE mode for data compression, and the EDMA write memory (radar cube) is initiated by an HWA completion event.

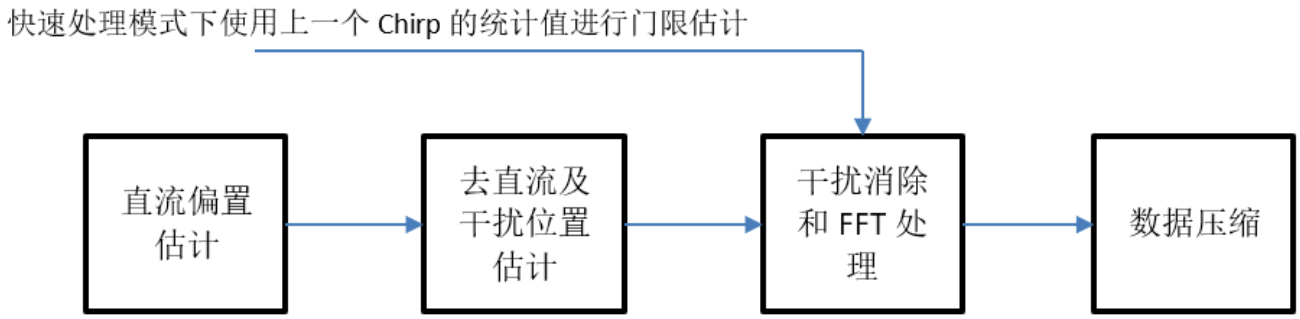


Figure 4-1. Range DDMA Flow

4.2 DopplerProcDDMA

In the DopplerProcDDMA processing flow, the HWA and DSP/MCU must work together for data processing, which is divided into four primary stages. The flow chart and detailed description are as follows.

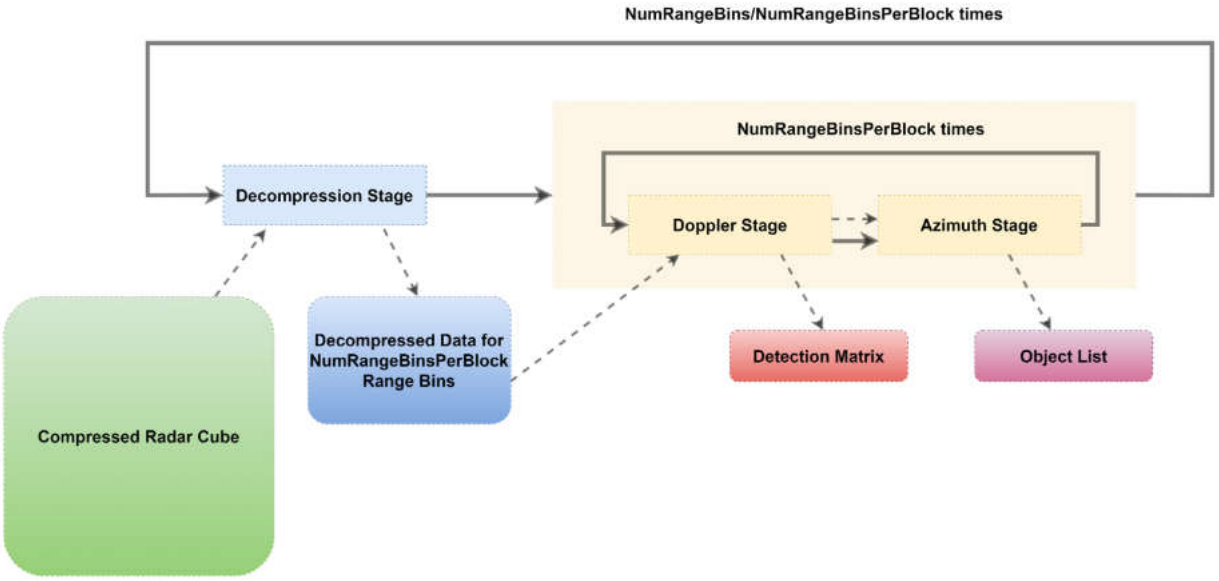


Figure 4-2. Doppler DDMA Flow

1. Data decompression

   When the DPU is initiated, if the Range DPU applies data compression, the HWA is required for decompression. To improve HWA execution efficiency and reduce parameter set reconfiguration time, the Doppler DPU typically reads in all eight range bins across all Chirps at once for data decompression. For example, assuming there are 96 Chirps and each block has a rangeBinsPerBlock of 2, in EGE mode, the size of a single read-in block is 96 * (sizeOfCompressedBlock). In BFP mode, because the data of each receive antenna is compressed individually, the data volume in the decompression stage must be multiplied by the number of receive antennas, i.e., 96 * (sizeOfCompressedBlock) * RxAnt.

2. FFT and DDMA demodulation

   After decompression, the Doppler DPU invokes the HWA to perform windowing, Doppler FFT, and DDMA demodulation on the decompressed data. On radar processors without a DSP, the HWA continues to perform DDMA demodulation using the summation and maximum value functions in its Statistics module after completing the FFT. On processors with a DSP, while DDMA demodulation is performed by the DSP core, the HWA continues with the Doppler FFT on the next range bin.
3. Horizontal angle calculation

   After the Doppler FFT and DDMA demodulation, the HWA continues with the horizontal angle (Azimuth) calculation.
4. Target detection on the speed-angle plot

   The CFAR engine of the HWA performs CFAROS detection in the Doppler dimension, and then the Local Maxima engine performs detection on the two-dimensional speed-azimuth plot. The results of the two detections are then combined. Because the detection proceeds from near range to far range gradually, in order to prevent an excessive number of near-range objects from restricting the radar detection range due to memory constraint, the number of detection objects per range bin is limited by the macros LIMIT_DETECTED_OBJS_PER_RANGEBIN and MAX_NUM_OBJ_PER_RANGE_BIN.

4.3 RangeCFARprocDDMA

Based on the two-dimensional radar range-velocity plot obtained by the Doppler DPU, the CFAR engine of the HWA is invoked to detect target objects in the range dimension. These range-dimension target objects are then merged with the target objects in the velocity-angle dimension in Section 4.2 to produce the final target objects, whose elevation angle is then calculated using the FFT module of the HWA.

5 Summary

The HWA in TI's mmWave radar chips significantly improves the overall system performance through its efficient hardware acceleration. However, to realize the full potential of the HWA, in-depth study and practice are required in system design, task assignment, and programming optimization. Maximizing the efficiency of mmWave radar systems by rationalizing task assignment, leveraging parallel computing power, and optimizing programming logic not only helps advance the development of mmWave radar technology but also provides a solid technical foundation for the future of intelligent driving and autonomous driving systems.

6 References

1. *swrs273a,* *AWR2943/44 Single-Chip 76- and 81-GHz FMCW Radar Sensor datasheet (Rev. A)*
2. *swrs296b,* *AWRL1432 Single-Chip 76- to 81-GHz Automotive Radar Sensor datasheet (Rev. B)*
3. *spruiv5,* *AWR294x Technical Reference Manual (Rev. C)*
4. *spru599b,* *AWRL6432, IWRL6432, AWRL1432, IWRL1432 Technical Reference Manual (Rev. B)*

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you fully indemnify TI and its representatives against any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale, TI’s General Quality Guidelines, or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products. Unless TI explicitly designates a product as custom or customer-specified, TI products are standard, catalog, general purpose devices.

TI objects to and rejects any additional or different terms you may propose.

Last updated 10/2025