Application Note

使用硬件加速器提高毫米波雷达性能

Allen Yin *China Central FAE*

摘要

TI 的毫米波雷达芯片通过集成 MCU、DSP 和 HWA（Hardware Accelerator 硬件加速器），形成了一个高度协同的计算架构。这种架构显著降低了 MCU 和 DSP 的计算负担，同时提高了整体系统的效率和灵活性。HWA 灵活使用和充分利用是提升系统效率的关键。本文将从系统设计、并行计算以及编程优化等方面，探讨如何高效地利用 MCU、DSP 和 HWA，以实现系统性能的全面提升。

修改记录

DATE	REVISION	NOTES
May 2025	*	Initial Release

内容

插图清单

表格清单

1 前言

德州仪器（TI）的毫米波雷达芯片系列凭借其高性能、高集成度和车规级可靠性，已成为智能驾驶感知领域的核心技术方案。其代表性产品如 AWR1843、AWR1642、AWR6843、AWR2944、AWRL1432 和 AWRL6432，覆盖了从基础 ADAS 到高阶自动驾驶的全场景需求。AWR1843 以 3Tx4Rx 配置和 FMCW 技术为核心，内置 Cortex-M4+DSP 和硬件加速器(HWA)，兼具高性价比与灵活扩展性。AWR2944 作为第二代旗舰芯片，采用 45nm RFCMOS 工艺，支持 4Tx 和 4Rx 通道与 5GHz 带宽，通过硬件安全模块（HSM）和锁步 Cortex-R5F+DSP+HWA 架构，满足 L3 级自动驾驶对前向雷达与角雷达的高精度要求。而 AWRL1432 针对短距感知场景优化，集成射频子系统与低功耗设计，在自动泊车、车门防撞等应用中展现快速响应优势。AWR6843 和 AWRL6432 作为 60GHz 的毫米波雷达芯片，在舱内的应用如儿童检测，安全带提醒以及防侵入等应用里得到广泛采用。

IWR1843、IWR6843、IWRL1432 和 IWRL6432 系列是面向工业和消费电子领域的单芯片毫米波雷达传感器。其中，IWR1843 工作在 76-81GHz 频段，集成 DSP、MCU 和雷达硬件加速器，适用于高精度工业雷达检测场景；IWR6843 覆盖 60-64GHz 频段，内置处理功能，支持智能应用如人员跟踪、生命体征监测及跌倒检测；IWRL1432 和 IWRL6432 均主打低功耗设计，前者工作于 76-81GHz，适合电动自行车安全系统等中短距离检测，后者频段为 57-64GHz，支持边缘计算和微型机器学习，可应用于楼宇自动化、医疗监测及低成本家电。全系列采用高集成度架构，兼顾分辨率、响应速度与能效，满足工业控制、消费电子及医疗健康等多元化需求。

TI 的毫米波雷达传感器通过硬件加速器（HWA）和 DSP，MCU 共同实现了信号处理效率与灵活性的双重突破。HWA 作为专用计算模块，通过固化关键算法、优化数据流架构及并行处理机制，显著降低 MCU 和 DSP 的计算负载，为雷达系统的高分辨率、低功耗和实时响应提供了核心支持。HWA 作为 TI 毫米波雷达芯片的专用信号处理引擎，旨在将传统由 DSP 或 MCU 执行的标准化算法（如 FFT、CFAR 检测、对数幅度计算等）硬件化，从而释放主处理器资源用于更高阶的数据处理（如目标分类、AI 融合等）。

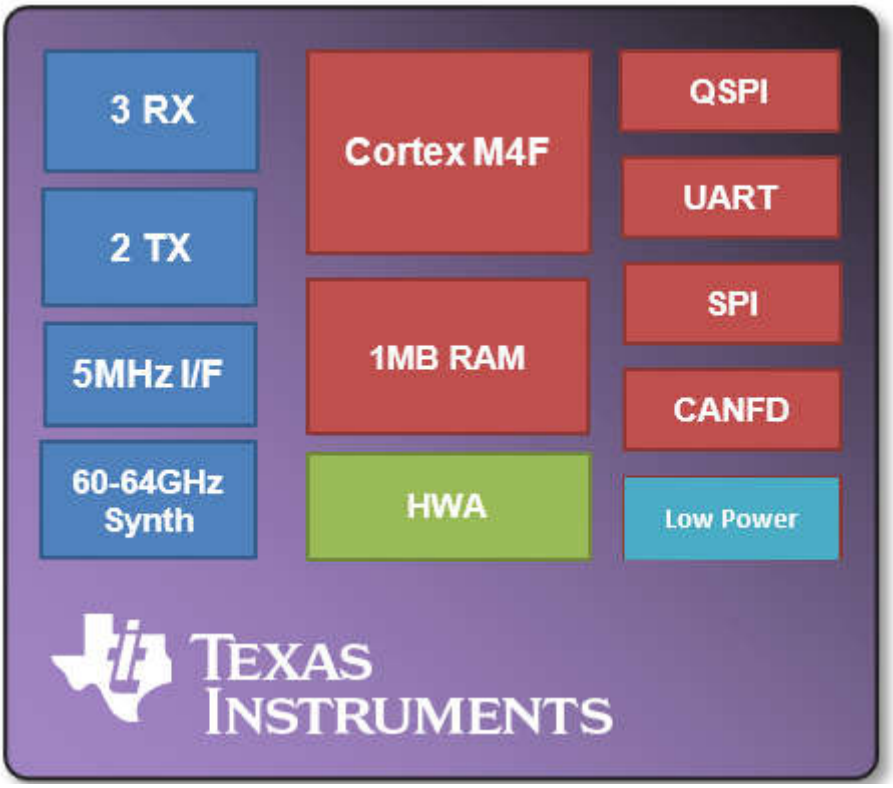


图 1-1. TI 毫米波雷达芯片 xWRL6432

HWA 采用模块化硬件设计，内置高速缓存以及计算单元，集成的高速缓存支持输入/输出乒乓操作，允许 DMA 读写与计算并行执行，实现零等待数据传输，并可以直接连接 ADC 缓冲区，即时获取原始数据完成 FFT，避免主内存访问瓶颈。计算单元包含 FFT 和 CFAR 检测等常见算法，集成幅度/对数幅度计算模块，并支持数据压缩与解压

缩，满足动态范围压缩需求。能效表现方面，对于 FFT/CFAR 等算法处理 HWA 通过优化，实际效率接近甚至超越 DSP 实现。因此，如何有效的利用 HWA 资源使其在更多的计算场景中发挥作用对于提高毫米波雷达性价比与功耗比变得尤其重要。

2 硬件加速器 HWA 简介

HWA 和毫米波雷达芯片的处理器为并行关系，框图如下，其通过一个 128 位总线与其他系统连接（HWA 1.2 版本为 64 位总线）。以 AWR2944 的 HWA2.1 为例，它包含一个加速引擎和八个 16KB 大小的内存，这些内存用于向加速器计算引擎提供输入数据和输出数据。HWA 无法直接访问加速器外部的存储单元，只能通过 DMA 或者处理器将外部数据搬移到 HWA 的内存或者搬出，实现部进行数据交互，内部内存划分为独立的 16 KB 的单元，允许通过 DMA 或者处理器同时进行读写操作，并使用乒乓缓冲机制处理数据。HWA 内部访问总线为 128 位宽，可实现每个时钟周期 128 位的吞吐量。

需要注意的是，HWA 本地内存中的任何一个 16KB 单元都可以作为加速器引擎输入数据的源地址，而其中的任何一个也都可以作为输出数据的目标地址，但有一个重要限制：源地址和目的地址不能是同一个 16KB 的单元，即在同一次运算中不能使用相同的存储单元进行覆盖操作。还需注意的是，加速器的本地单元并不强制要求使用乒乓模式，如果应用场景需要，可以将两个或四个存储体组合为更大的 32KB 或 64KB 输入和输出单元。HWA 内部访问这些内存的起始地址从 0x0000 开始。

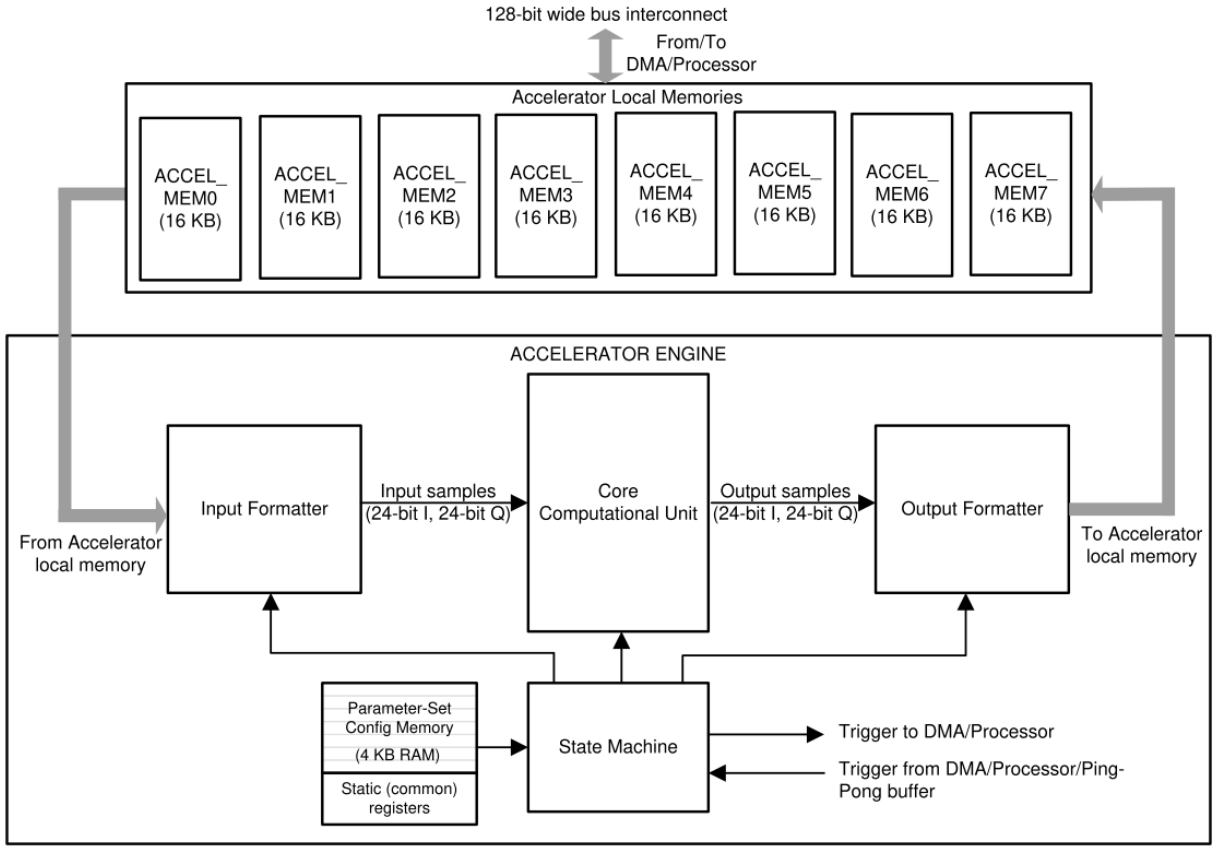


图 2-1. 硬件加速器系统

2.1 HWA 功能模块及版本区别

HWA 中处理部分的功能模块主要有

- 状态机（State Machine）：

  状态机负责控制 HWA 的整体运行，具体包括启动、循环、停止操作，以及 HWA、DMA 和主处理器之间的触发与握手机制, 它还负责从参数集配置内存中读取预设的程序进行顺序执行，状态机可以配置为循环遍历参数集，总共执行 NUMLOOPS 次（NUMLOOPS 参数被设置为 0 或 0xFFF，此时循环将不运行或无限次运行）。

例如，若需将状态机配置为循环运行前四个参数集 64 次，则寄存器应按如下方式设置：PARAM_START_IDX = 0， PARAM_END_IDX = 3，NUMLOOPS = 64。状态机的开始运行可以通过软件，硬件事件或者 DMA 事件进行触发。同样，状态机的结束也可以产生中断事件或者 DMA 事件。

- 输入格式化器（Input Formatter）：

  负责从 HWA 本地内存读取数据，并将其送入核心计算单元。在此过程中，输入格式化器支持灵活的访问方式，包括按 16 位或 32 位对齐的数据格式读取，用户可以配置截断或补 0 的位数， 二维数据转置读取，灵活的缩放和符号扩展（转换成 24 位内部位宽），最终，该模块将 24 位复数样本输入核心计算单元。在该模块内用户可以配置输入数据补 0 的操作，比如当输入 FFT 的数据长度为 56，需要做 64 点的 FFT, 则 Formatter 会在每次 FFT 的操作前在原始数据的尾端进行补 0 满足 FFT 对于数据长度的要求，要注意配置输入数据的长度不应超过要进行 FFT 的点数。
- 输出格式化器（Output Formatter）：

  该模块负责将核心计算单元的数据写入本地内存，并提供多种格式化方式，例如：16 位或 32 位对齐的输出数据格式写入，二维数据转置写入，从 24 位内部位宽数据缩放到 16 位或 32 位，用户可以配置截断或补 0 的位数，符号扩展，这种灵活性可适配不同硬件接口的需求。

  在进行数据输出时，用户也可以选择跳过开头或者尾部的一部分数据，比如在进行 FFT 操作以后，只需要中间一部分的数据，则可以通过寄存器配置跳过开头以及尾部的数据。
- 核心计算单元（Core Computational Unit）：

  该单元执行各类计算，例如加窗、FFT、幅度计算、对数运算（log2）和 CFAR（恒虚警）检测。它以流式输入方式接收数据（每时钟周期一个样本），经过计算后以流式输出传递结果。
- 参数集配置内存（Parameter-Set Configuration Memory）：

  用于预配置加速器操作的参数集（寄存器设置），支持状态机按程序循环执行一系列操作。

不同的毫米波雷达芯片内置的 HWA 版本不相同，其功能上有所区别。毫米波雷达芯片及其对应的 HWA 版本为：

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

不同的版本功能对应如下表

表 2-1. HWA 版本功能区别

	HWA1.0/1.1	HWA1.2	HWA1.5	HWA2.0/2.1
频率	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)
参数组	16	32	32	64
复数乘法	7 模式	7 模式	3 模式	10 模式
干扰消除	使用固定门限，干扰归 0	使用固定门限，干扰归 0	使用多种统计信息干扰归 0 或内插	使用多种统计信息干扰归 0 或内插
对数幅值	支持(0.3dB 精度)	支持(0.3dB 精度)		支持(0.02dB 精度)
数据压缩/解压缩	不支持	支持	仅支持压缩	支持
CFAR	CFARCA	CFARCA, CFAROS	不支持	CFARCA, CFAROS
统计信息	1 维数组求和及最大值搜索	1 维数组求和及最大值搜索	不支持	1 维数组求和及最大值， 直方图， 2 维最大值， CDF
其它	无	无	无	通道合并，插零，上下文切换等

2.2 核心计算单元

核心计算单元 CCU 是 HWA 最重要组成，在 HWA1.2 中 CCU 包含三项主要功能——即 FFT 引擎、CFAR 引擎和压缩引擎。在一个时刻这三个功能中只有一个能够使能。通过不同的参数集配置，可以依次执行不同引擎的处理流程，使得多个参数集按序执行完成所需的系列计算操作。寄存器 ACCEL_MODE 控制特定参数集中实际启用的处理引擎。

HWA1.2 的 CCU 系统的框图如下

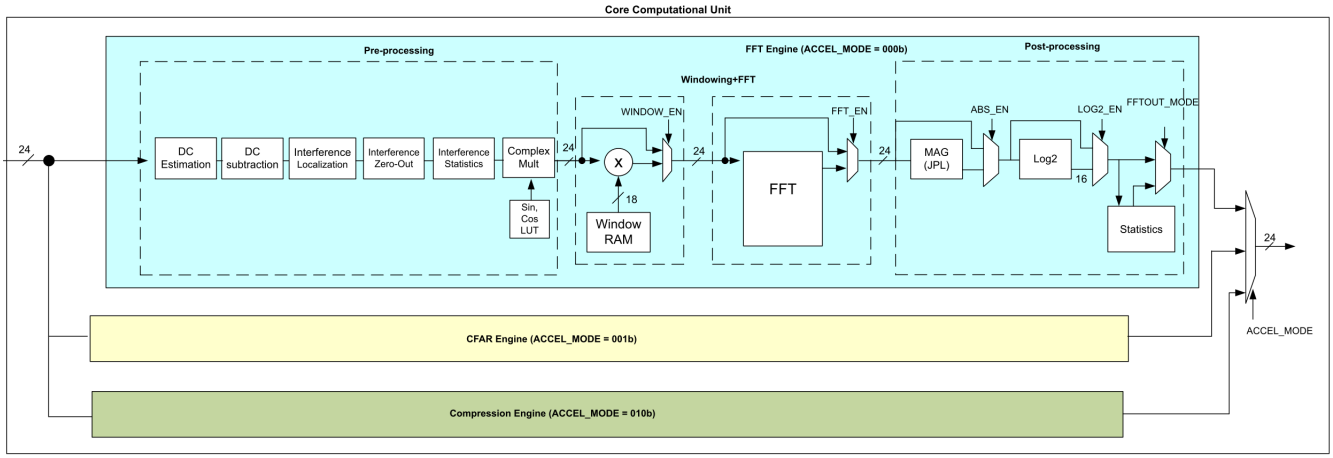


图 2-2. HWA1.2 核心 计算单元系统

CCU 使能 FFT 引擎时，同时可以配置 FFT 前的加窗操作，用户可以通过 Window RAM 配置自己的加窗系数。

在 FFT 之后，CCU 支持取模（绝对值）和取对数的操作，对于雷达信号后续处理更加便利。在 FFT 前，CCU 支持一些前置操作，其中包括直流估计和去除，干扰定位与消除，干扰统计信息，复数乘法等，每一个模块都可以单独使能以达到最大的灵活度。

FFT 引擎的复数乘法模块不仅可以与 FFT 配合在 FFT 前对数据进行预处理（如旋转，乘以标量，向量等操作），同样也可以单独作为一个功能模块运行，即在 CCU 里完成复数向量乘或求和等功能（此时 FFT 处于不使能状态），复乘模块的功能简要介绍如下，如果需要了解更多细节与操作的方法，请参考文献 3 和文献 4

- 移频模式，即通过复乘将输入数据移动一定的频率；
- DFT 模式，通过移频和自增功能完成 DFT 计算；
- FFT 拼接，可支持更多点数的 FFT，如 2048，4096，8192 等；
- 幅度平方，求输入数据的幅度的平方值，并可与最后的统计模块一起工作用于计算幅度平方和；
- 标量乘法，可以在输入数据上乘以不同的标量值；
- 向量乘法 1，计算两个向量的乘法，并可与统计模块求和功能共同完成点乘操作；
- 向量乘法 2，与向量乘法 1 的微小不同是每次计算迭代后，模式 1 会从内部 RAM 的 0 位置重新开始计数。

下面三种复乘模式仅在 HWA 2.0 上实现

- 递归窗口模式，基于加窗的操作叠加移频，根据如下公式给输入数据加窗，Wk(n) = W0(n) * exp(－j*K* θ (n)*2*pi/16384)，K 值可以根据用户设置在每一次迭代归 0 或者延续上次；
- 基于查找表的频率与相位去旋转模式，功能类似于移频模式，但增加了基于可编程 RAM 的频率去旋转和相位去旋转能力；
- 具有精细频率增量的移频模式：此模式是移频模式的扩展模式，除原有的频率移位功能外，还新增了一个带符号的 10bit 微调值，该微调量将被叠加到解旋转频率上，可以使解旋转频率在每次循环迭代后产生增量变化。

另外，CCU 还可以支持在 FFT 引擎内进行 BPM 解调，通道合并，内插 0（注意与 FFT 的补 0 区别）等操作。

在 FFT 引擎处理的最后，CCU 内置了一个统计（Statistics）模块可用于输出数据的统计信息处理，其中支持的功能有获取输入数据的最大值或累加。HWA2.0 以上的版本可以支持在二维数据上查找最大值，直方图/CDF 等更多的统计信息。

CCU 使能 CFAR 引擎时，可以支持 CFAR-CA 与 CFAR-OS，并可以支持线性模式和对数模式的 CFAR 检测处理，最终生成峰值列表。CFAR 引擎可以对 CFAR 算法的各个参数进行配置，具体参数可以参考 TRM。

为了最大化利用系统内存，CCU 可以配置为压缩/解压缩引擎用于雷达 FFT 数据压缩，支持 BFP （Block Floating Point）和 EGE （Exponential Golomb Encoder） 两种压缩方式。一般而言，BFP 适用于针对距离维上连续点的压缩，速度更快而且占用内存小，而 EGE 需要先对压缩数据进行分析再进行压缩，适用于将多根天线和多个距离维组成数据块进行要锁，速度较 BFP 会慢，但理论上 EGE 压缩/解压缩后能保留的信号细节会更多。

在 HWA 2.0/2.1 版本，CCU 可以支持局部最大值引擎（Local Maxima engine），通过比较检测单元幅度是否大于或等于其所有相邻样本的幅度来确定是否为局部峰值，相邻样本的选择可以通过寄存器进行配置；用户也可以定义行和列的门限值来筛选需要进行检测的单元。

3 利用 HWA 的矩阵乘法示例

本章以一个简单的实例来具体说明如何灵活使用 HWA。

在毫米波雷达信号处理系统中，复数矩阵乘法经常作为核心算法，其计算效率直接影响系统实时性指标。TI 的毫米波雷达处理器架构通常以 DSP 核为主导进行复数矩阵乘法运算，到第二代产品（如 xWRL1432/6432、AWR2944LC 等）考虑到功耗和成本因素，采取硬件加速器（HWA）承担主要信号处理任务，同时采用低功耗 MCU 核（ARM Cortex-M4/R5）进行系统控制与数据管理。这种架构降低了功耗和成本，但同时也带来新的技术挑战——当处理器未配置专用 DSP 核时，传统采用 MCU 和 CMSIS 库的矩阵运算方案存在显著性能瓶颈，其单线程串行执行模式导致计算时延增加，且占用 MCU 资源影响系统多任务并行处理能力。

在此背景下，HWA 的闲置资源利用成为优化突破口。HWA 作为专用硬件加速模块，可以实现复数矢量乘法计算。但要进行复数矩阵乘法，因为需要不断输入行列数据进行计算，常见的方法需要 MCU 干预进行数据管理和控制输入输出。本章节采用 EDMA 配合 HWA 的方式可实现无需 MCU 干预的矩阵乘法运算，同时也给用户示例如何灵活的配合使用 EDMA 和 HWA 的各个模块之间的数据拷贝，计算和触发，最后给出在 xWRL6432 芯片上测试的性能。

下面以 4×4 矩阵乘法为例说明在 xWRL6432 上的具体实现。矩阵 A（4 行×4 列）与矩阵 B（4 行×4 列）相乘时，结果矩阵 C 的维度为 4 行×4 列，C 的第 i 行第 j 列元素的计算公式如下，即 A 的第 i 行与 B 的第 j 列对应元素乘积之和

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

例如计算 C 的第一个元素

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

其它矩阵元素均按此方式计算，可知 C 的每一个元素都需要调用 HWA 进行一次复数点乘计算，4X4 矩阵一共需要 16 次点乘运算。

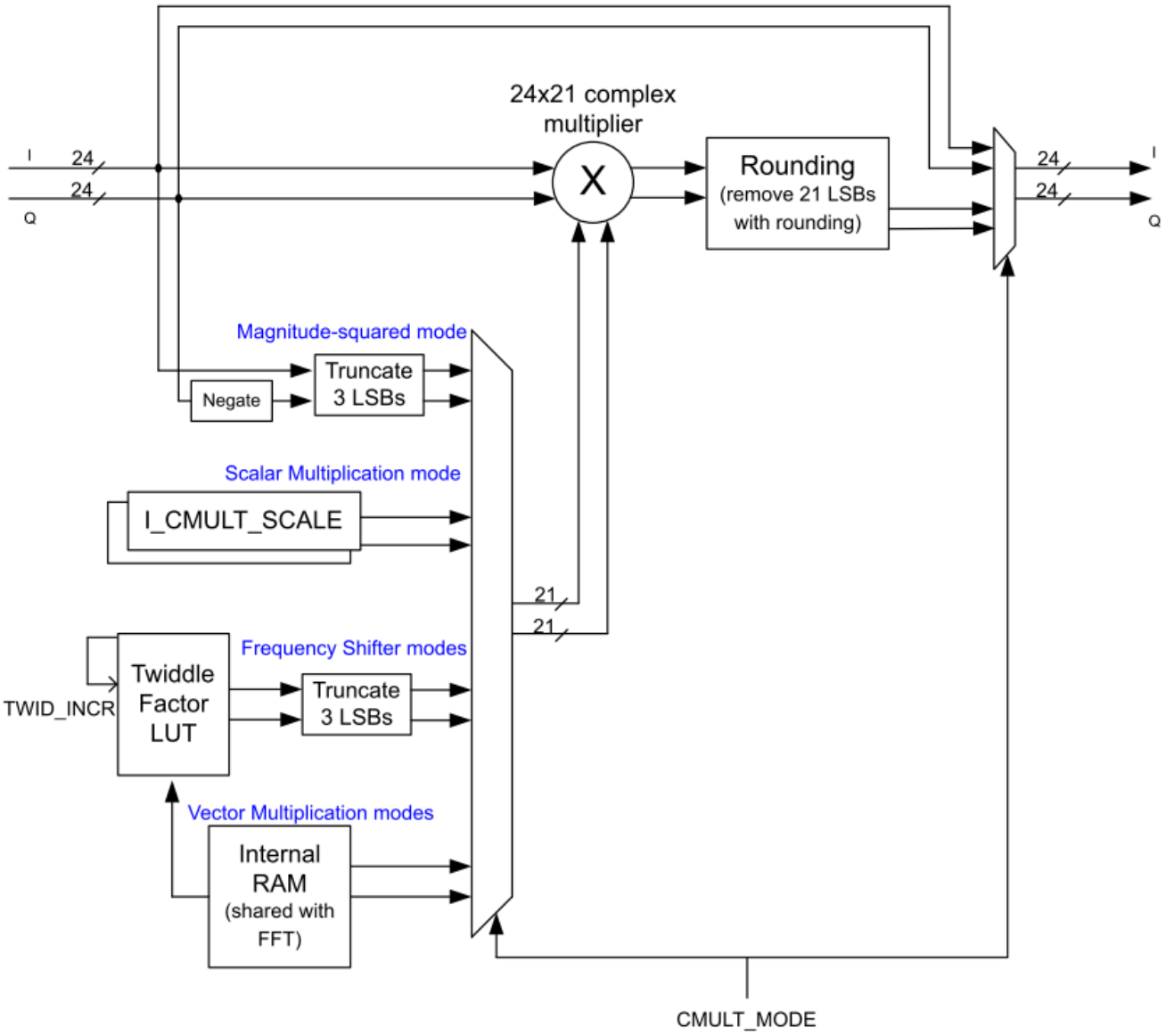
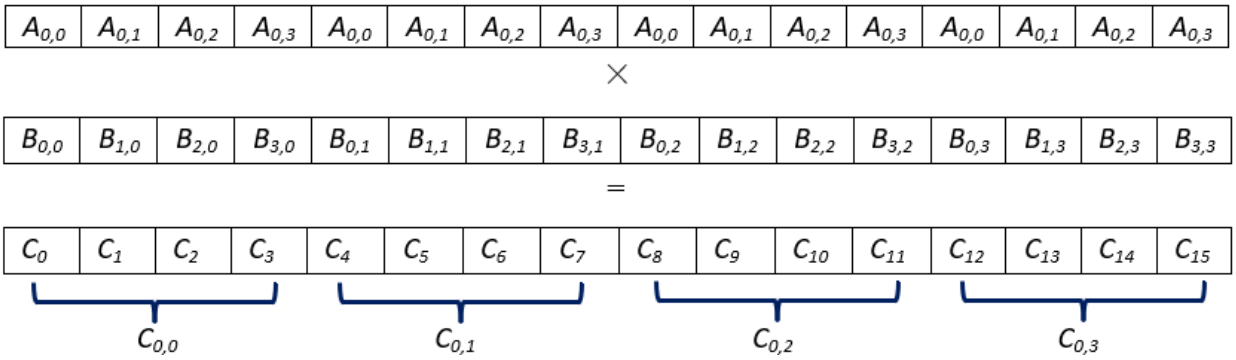


图 3-1. 复乘功能模块

根据 HWA 进行复数点乘的方式输入，复数向量一个通过 Input Formatter 正常输入，另一个复数向量写入上图中的 Internal RAM，需要注意通过 Input Formatter 的向量取 24 bit 精度，而写入 Internal RAM 的向量取高位 21 bit 精度，两个数相乘后去掉最低的 21 bit 最终得到 24 bit 的复数，因此在进行计算时要先对输入数据进行必要的缩放移位操作防止最终结果过大或过小造成溢出。

接下来考虑如何将二维的矩阵计算转换为 HWA 能够支持的一维向量计算，当计算 C 矩阵的第一行结果时，需要将 A 矩阵的第一行分别和 B 矩阵的 1-4 列进行点乘，通过将 A 矩阵的第一行复制 4 份，形成 1 个 16 个元素的一维向量，同时将 B 矩阵的 4 列在列方向上进行级联拼接，也形成 1 个 16 个元素的一维向量；将两个向量相乘得到 16 个复数结果，然后再分别将 4 个元素相加得到 C 矩阵第一行结果。



余下的三行同理。

根据 HWA 的向量乘法的实现，此处将 B 矩阵按列写入 Internal RAM，一共写入 16 个复数值，为了能完全脱离 MCU 运行，在流程控制上分为三步，使用两个 HWA 的参数组配置

1. 使用 EDMA 对 A 矩阵每行复制 4 份同时拷贝到 HWA 的输入内存单元中，结束后用 Link 方式执行下一步，
2. 使用 EDMA 的 Link 方式，通过 HWA 的 DMA2ACC 触发方式启动 HWA 进行向量乘法操作，每一次向量乘法计算出 16 个元素，共计算 4 次向量乘法得到 64 个复数元素，
3. 使用 HWA 的统计模块的求和功能，将每 4 个复数元素进行相加，得到最终的矩阵结果。

EDMA 的配置如下，第一组参数用于矩阵 A 复制到 HWA 的内存，并通过 EDMA AB sync 方式及 Chain 事件自动触发完成矩阵复制和传输，并通过 Link 和完成事件触发第二组参数的运行，第二组参数复制 HWA 的 Channel 1 的触发字到 DMA2ACC 的触发寄存器地址完成对 HWA 的触发启动操作。

```
/* 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);
```

HWA 的参数配置如下，第一组参数使用 DMA 触发方式，通过上述 EDMA 事件触发，然后调用复乘模块计算复数乘法结果，完成后立刻触发第二组参数，第二组参数使用累加模块对每 4 个复数值进行累加得到最后的结果。

```
/* 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);
```

在整个流程中， EDMA 参数组和 HWA 参数组可以预先写入节省每一次操作的时间。在后续的每一帧计算中，MCU 核只需要完成矩阵 B 写入 Internal RAM 的操作（需将矩阵 B 转置写入以开展计算），然后启动 EDMA，等待 EDMA 自动完成触发 HWA 开展计算，最后 HWA 完成中断通知 MCU 进行后续操作。

在 xWRL6432 芯片上，MCU 核 M4F 运行于 160Mhz 主频而 HWA 1.2 运行于 80Mhz 主频，在 MCU 上使用 CMSIS 库及 HWA 计算定点复数矩阵乘法的周期统计时间如下所示，为了方便比较，表中数值均采用 M4F 的时间统计周期（Cycle）数值。

表 3-1. xWRL6432 矩阵乘法性能对比

		矩阵大小				
		4x4	6x6	8x8	12x12	16x16
		周期数				
M4F	CMSIS 库	704	1856	3776	10960	23984
HWA	加载 Internal RAM	160	272	432	912	1600
	计算矩阵乘法	784	1920	3904	12272	28064

因为 xWRL6432 HWA 频率只有 M4F 的一半，因此 HWA 矩阵计算的时间比 CMSIS 库略长，但是在 HWA 计算过程中 M4F 可以并行进行其它任务，有效的利用 HWA 释放了 MCU 计算资源达到并行计算目的 。在 AWR2x44 系列芯片上，HWA 的主频与 MCU 相同，使用 HWA 进行矩阵计算的 Cycle 数将降低到上表中的 50%左右，此时使用 HWA 进行矩阵计算可以降低处理时延。

4 毫米波雷达链路中的 HWA 使用

本章介绍 TI 的毫米波雷达 SDK 中数据处理单元 DPU（Data Process Unit）使用 HWA 对数据进行处理的方式，用户可以通过本章内容更加了解 HWA 是如何提高毫米波雷达性能。因为用户使用场景的差异，如 TDMA 和 DDMA, 或芯片本身的差异，如有 DSP 和没有 DSP 核，这些 DPU 内使用 HWA 的方法和处理流程稍有不同，本章以常见的 AWR2944 系列上 DDMA 的流程为例进行说明。

4.1 RangeProcDDMA

一般 Range FFT 处理由 HWA 硬件完成。根据射频参数，RangeProcDDMA DPU 通过配置 FFT 引擎实现运算，同时设置 EDMA 通道控制数据输入输出，采用乒乓模式方便与原始数据采集进行并行处理，乒乓模式两边各采用三组或五组 HWA 参数组。了提升最大测速范围，在有些时候用户可以在该 DPU 里开启快速处理模式，其原理是使用前一个 chirp 的直流（DC）统计数据和干扰阈值估计值，而非实时计算这些参数，这一机制节省了两个参数集（DC 估计、DC 去除及干扰统计信息）的执行时间。

具体流程如下：

1. 数据输入

   首先通过 EDMA 将 ADC 缓冲区的采样数据搬运至 HWA 内存。此处采用一组没有配置的参数组，不进行任何数据处理，仅配置触发流程的方式（DMA 或直接由用户软件触发），后一种用户软件触发的方式主要用于快速处理模式。
2. 预处理及 FFT 阶段

   对于非快速处理模式，需要三组参数组顺序执行，A 参数组执行直流分量估计（DC Estimation），随后 B 参数组进行直流分量消除（DC Subtraction）还有干扰估计（Interference statistics estimation），然后调用 C 参数组进行干扰消除（Interference mitigation）以及 FFT 计算。快速处理模式，通过前一个 Chirp 得到的直流及干扰统计信息直接手动更新门限值在同一组参数组进行去直流，干扰消除及 FFT 计算。
3. 数据压缩

   使用压缩引擎配置 BFP 或者 EGE 模式进行数据压缩，通过 HWA 完成事件启动 EDMA 写入内存（Radar cube）。

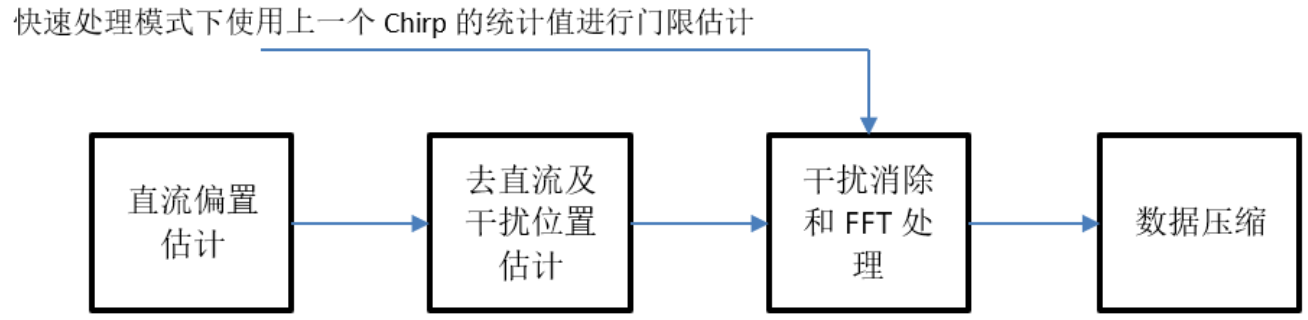


图 4-1. Range DDMA 流程

4.2 DopplerProcDDMA

在 DopplerProcDDMA 的处理流程中，需要 HWA 和 DSP/MCU 配合进行数据处理，分成四个主要工作阶段，其流程图及详细说明如下。

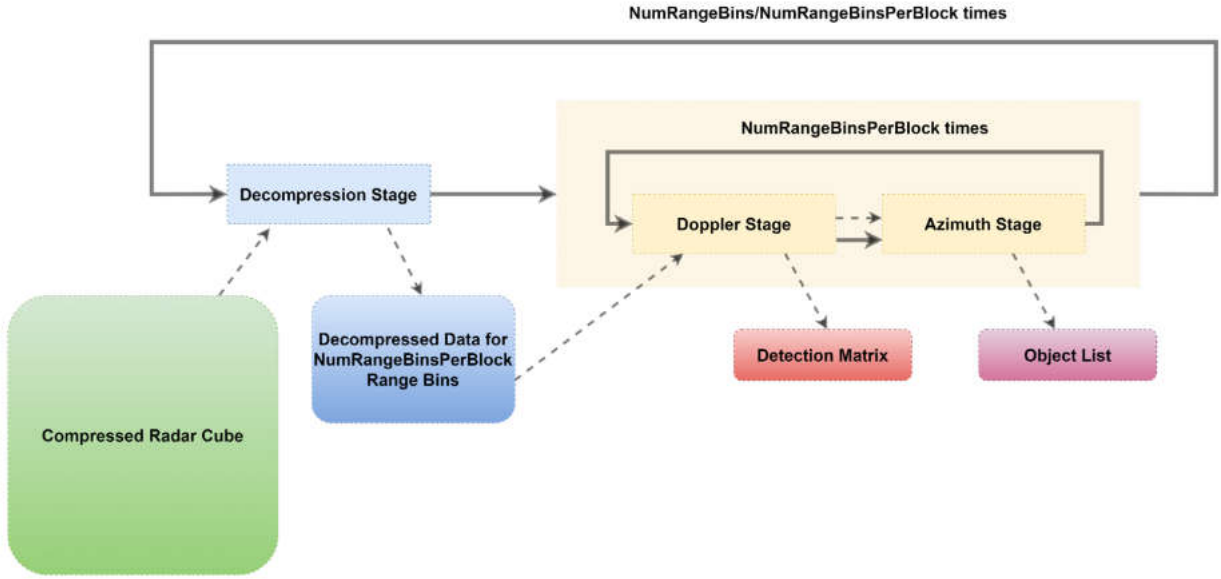


图 4-2. Doppler DDMA 流程

1. 数据解压缩

   在 DPU 的开始阶段，如果 Range DPU 采用了数据压缩，则需要使用 HWA 进行解压缩，为了提升 HWA 的执行效率，缩短参数组重配时间，一般 Doppler DPU 会一次性读入所有 Chirp 的 8 个距离单元数据进行数据解压缩，例如例如，假设 chirp 为 96，每个块的 rangeBinsPerBlock 为 2，则在 EGE 模式下，单个读入块的大小为 96*(sizeOfCompressedBlock)；而在 BFP 压缩中，因为是单独对每根接收天线的数据进行压缩，则解压缩环节的数据量需乘以接收天线数量，即 96*(sizeOfCompressedBlock)*RxAnt。
2. FFT 及 DDMA 解调

   解压缩完成后，Doppler DPU 调用 HWA 对解压缩后的数据进行加窗和 Doppler FFT 及 DDMA 解调。在没有 DSP 的雷达处理器上，HWA 在完成 FFT 后则继续使用 HWA 的统计模块中累加和最大值功能进行 DDMA 解调。在有 DSP 的处理器上，则并行的使用 DSP 核进行 DDMA 解调，HWA 继续进行下一个距离单元上的 Doppler FFT。
3. 水平角度计算

   完成 Doppler FFT 和 DDMA 解调后，继续使用 HWA 进行水平角度（Azimuth）的计算。
4. 速度角度图上的目标检测

使用 HWA 的 CFAR 引擎在 Doppler 维度上进行 CFAROS 检测，然后使用局部最大值引擎进行速度水平角度二维图上的检测，将两步检测的结果进行合并。因为该检测是从近距离到远距离逐步进行，为了防止近处点过多造成雷达探测距离受到内存的限制，可以通过 LIMIT_DETECTED_OBJS_PER_RANGEBIN 及 MAX_NUM_OBJ_PER_RANGE_BIN 宏来限制每一个距离上检测到的点的个数。

4.3 RangeCFARprocDDMA

在 Doppler DPU 得到的雷达距离速度二维图的基础上，调用 HWA 的 CFAR 引擎进行检测并得到距离维度上的目标点，将距离维度的目标点与节 4.2 中速度角度维度上的目标点进行合并得到最终的目标检测点，并计算目标点的垂直角度信息（使用 HWA 的 FFT 模块）。

5 总结

TI 毫米波雷达芯片中的 HWA 通过其高效的硬件加速能力，显著提升了系统的整体性能。然而，要充分发挥 HWA 的潜力，需要在系统设计、任务分配、编程优化等方面进行深入研究和实践。通过合理分配任务、利用并行计算能力和优化编程逻辑，可以实现毫米波雷达系统效率的最大化，这不仅有助于推动毫米波雷达技术的发展，也为未来的智能驾驶和自动驾驶系统提供了坚实的技术基础。

6 参考文献

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)*

重要通知和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、与某特定用途的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他安全、安保法规或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的相关应用。严禁以其他方式对这些资源进行复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。对于因您对这些资源的使用而对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，您将全额赔偿，TI 对此概不负责。

TI 提供的产品受 TI 销售条款)、TI 通用质量指南 或 ti.com 上其他适用条款或 TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。 除非德州仪器 (TI) 明确将某产品指定为定制产品或客户特定产品，否则其产品均为按确定价格收入目录的标准通用器件。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

最后更新日期：2025 年 10 月