使用低速串行总线的实时 ADC 原始数据采集方法

Wesley He *Central FAE*

摘要

TI 的 IWR6843 是业内第一款基于 RF-CMOS 工艺，将中射频电路，VCO，ADC，DSP 和硬件加速器集成在单颗芯片内的 60GHz 毫米波雷达 SoC，它具有集成度高，成本低，开发简单等优点，在工业及楼宇自动化里有广泛的应用。本文介绍了使用低速串行总线的实时 ADC 原始数据采集方法，目的是在开发过程中增加少量工作，即可使用低速串行总线接口采集雷达回波的 ADC 信号进行分析及算法验证。

雷达芯片的常规流程是，启动后实时采集 ADC 数据并存放在片上 RAM，DSP 会实时的对数据进行雷达信号算法处理，但是实时处理 ADC 数据存在由于环境变化带来的可重复性不够的问题，算法的数据验证受到局限。在数据验证的过程中，操作步骤可拆分为数据采集以及数据回填这两个步骤，本文主要关注 ADC 数据采集的实现方法。

对于数据采集的实现，现有方案可以采用 LVDS 接口，软件配置 EDMA 将 ADC 数据通过 LVDS 外设送出，硬件上使用 DCA1000 数据采集卡对 LVDS 接口上送出的 ADC 数据进行采集，这是一个目前在研发阶段使用广泛的解决方案。但是对于已量产的产品而言，多数情况并不会留出高速 LVDS 接口，只会保留低速接口送出检测数据，使用低速串行总线如 UART 等接口采集数据的方法，可在模组出现故障时方便问题的回溯和分析。

本应用手册着重介绍使用低速串行总线 UART 口进行实时 ADC 数据采集的快速实现方法，此方法无需预留高速 LVDS 接口，利用 UART 接口或者其他通信接口即可实现 ADC 数据的实时采集，并给出了实现方法及示例代码。

修改记录

Version	Date	Author	Notes
1.0	November 2021	Wesley He	First release

目录

图

表

1. 三种 ADC 数据采集方法的介绍及对比

1.1. 总览

IWR6843 是一个集成的单芯片调频连续波（FMCW）雷达传感器，能够在 60 至 64 GHz 频段内工作。该传感器采用 TI 的低功耗 45nm RFCMOS 工艺制造，并以极小的体积实现了前所未有的集成度。 IWR6843 是工业领域中集成低功耗，丰富内部自监控电路，超精确探测的雷达系统的理想解决方案，本应用手册基于 IWR6843 进行示例，方法也适用于其他型号器件。

对于数据获取而言，IWR6843 支持 LVDS 高速接口进行实时的数据采集，其他的对外接口 SPI/UART/CANFD 等常用接口均可用于外部通信，各个接口的速率支持如下：

- LVDS:
  - 900 Mbps (450 MHz DDR Clock)
  - 600 Mbps (300 MHz DDR Clock)
  - 450 Mbps (225 MHz DDR Clock)
  - 400 Mbps (200 MHz DDR Clock)
  - 300 Mbps (150 MHz DDR Clock)
  - 225 Mbps (112.5 MHz DDR Clock)
  - 150 Mbps (75 MHz DDR Clock)
- Hardware in Loop（DMM/HIL）: Up to 100 Mbit/s pin data rate
- SPI: support up to 40MHz clock
- UART: up to 3.125 Mbps
- CANFD: supports up to 10 Mbit/s data rate

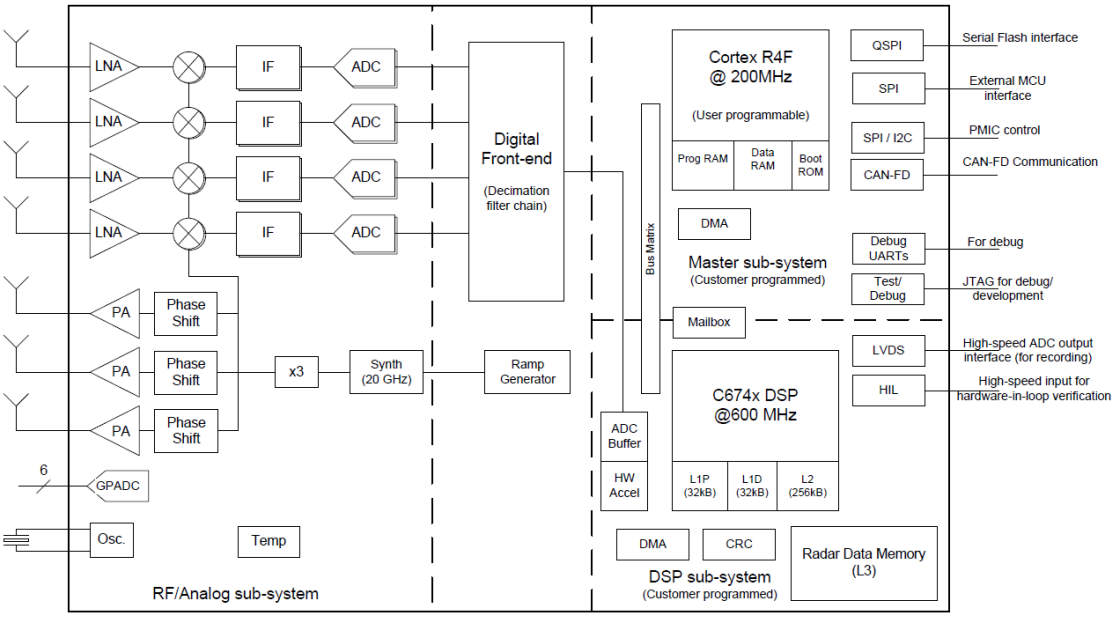

图 **1**. IWR6843 芯片框图

1.2. 使用 DCA1000+mmWave Studio 软件进行数据采集

DCA1000EVM 是一款数据采集卡，适配于德州仪器（TI）的 77GHz&60GHz 毫米波雷达传感器 EVM 高速 60PIN 接口，使用户能够通过 DCA1000 板卡的以太网获得从雷达的 LVDS 接口送出的 ADC 数据。该数据采集卡使用带有 DDR3L 的 Lattice FPGA（LFE5UM- 85F-8BG381I）进行 LVDS 接口到以太网的接口转换。数据采集卡和雷达传感器 EVM 之间的信号接口使用 60 针 Samtec 高密度连接器（型号 SAMTEC QTH-030--01-L-D-A）。使用数据采集卡 DCA1000EVM 可以实现以下功能：

- 原始数据模式：在此模式下，所有 LVDS 数据均按原数据格式捕获并通过以太网接口传输出来。

- 数据分离模式：在此模式下，用户可以将特定的报头 header 分配给不同的数据类型； FPGA 根据报头分离出不同的数据，并通过以太网接口对其进行传输。

mmWave Studio 是一款独立的 Windows GUI，它具有配置和控制 TI 毫米波传感器模块以及收集 ADC 数据以进行离线分析的功能。该 ADC 数据捕获旨在评估和表征射频性能，以及进行信号处理算法的 PC 开发。下图是使用 DCA1000+mmWave Studio 软件进行毫米波雷达数据采集的框图，PC 上使用 mmWave Studio 软件，通过 USB←→SPI 接口对毫米波雷达芯片进行工作模式配置并采集数据。其优势是可支持全部的毫米波雷达工作模式配置，包括 chirping mode、advance frame mode、continuous wave mode。并且 GUI 集成的数据分析功能可对采集回来的数据进行初步的分析。

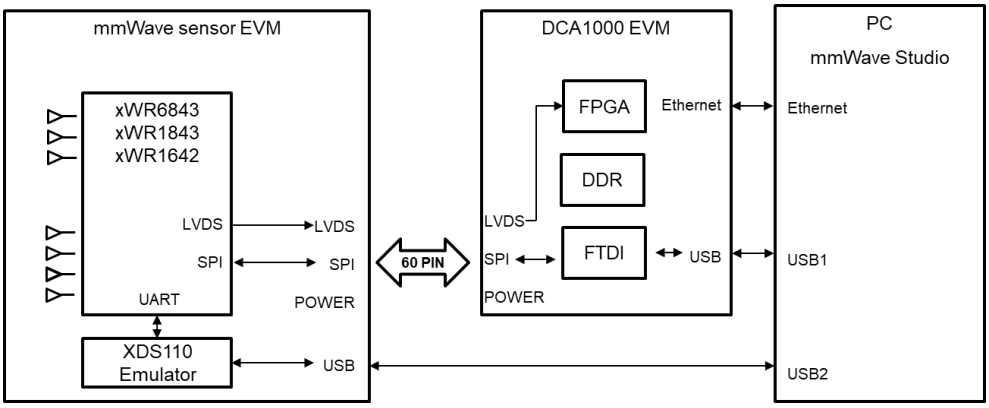

图 **2**. 使用 DCA1000+mmWave Studio 软件进行数据采集框图

小结，使用 DCA1000+mmWave Studio 软件进行数据采集的软硬件要求如下：

- 硬件：mmWave radar sensor EVM + DCA1000 + PC
- 软件：mmWave studio
- Radar 硬件所需预留接口：LVDS（ADC 数据传输） + SPI（RF 参数配置） + UART（固件加载）+ SOP（SOC 启动模式配置）

1.3. 使用 DCA1000+mmWave Studio CLI tools 软件进行数据采集

mmWave Studio CLI tools 是使用命令行界面 (CLI) 控制毫米波传感器的 GUI 工具，可以替换 mmWave Studio 的基本功能，对比于完整功能的 mmWave Studio，mmWave Studio CLI tools 是一个轻量化的工具，它使用与 OOB(SDK out-of-box demo) 相同的配置方式与命令，同时，在硬件连接的需求上，省了一个 SPI 接口，所以在外场测试过程中，能够简化硬件连接及操作流程。

以下是此工具提供的通过 UART 配置毫米波传感器设备的命令行界面的功能列表。

- 它使用来自文本配置文件的所有 CLI 输入，因此用户可以从该文件控制工具的功能。
- 它支持毫米波传感器配置参数的 CFG 或 JSON 输入文件格式。
- 配置 DCA1000EVM 并通过以太网将 ADC 数据从毫米波传感器捕获到 PC。
- 将毫米波传感器生成的监控报告捕获到 JSON 文件，然后绘制这些数据。
- 使用 Matlab 工具对捕获的 ADC 数据进行后处理。

此方法同样需要 DCA1000 数据采集卡的支持，于此同时，PC 通过串口加载配置命令，配置雷达芯片的工作模式及参数。下图是使用 DCA1000+mmWave Studio CLI tools 软件进行数据采集的框图。

小结，使用 DCA1000+mmWave Studio CLI tools 软件进行数据采集的软硬件要求如下：

- 硬件：mmWave radar sensor EVM + DCA1000 + PC
- 软件：mmWave studio CLI tools
- Radar 硬件所需预留接口：LVDS（ADC 数据传输） + UART（RF 参数配置及固件加载）+ SOP（SOC 启动模式配置，仅需配置一次）

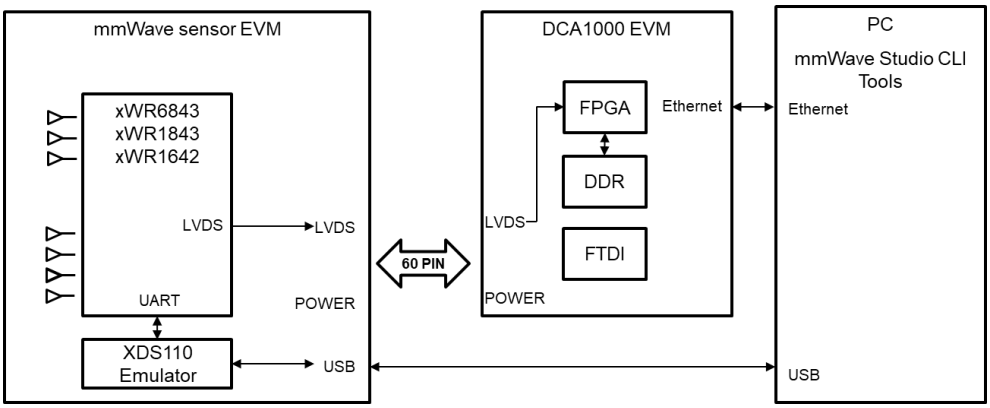

图 **3**. 使用 DCA1000+mmWave Studio CLI tools 软件进行数据采集框图

1.4. 使用低速串行总线 UART 口进行数据采集

在章节 1.1&1.2 介绍的数据采集方法中，都离不开高速接口 LVDS，其原因是 mmWave radar sensor 传输的数据率较高，需要使用高速接口实时的将数据送出。数据率举例如下：

128(ADC samples per chirp) * 128(Chirps per frame)*4(RX channel)*4(IQ data bytes)*25(Frames per second) = 50Mbps

在数据传输过程中，考虑数据传输的开销，需要>50Mbps 的高速接口才可实时采集 25Hz 刷新率的数据。

本文介绍的基于 UART 的 ADC 数据获取方法，无法实时的获取 25Hz 刷新率的 ADC 数据，但可获得单个 frame 的 ADC 原始数据，也就是 1Hz 刷新率，可用于验证单帧数据及信号处理算法的准确性，适用于未预留高速接口的板卡做功能调试，受限于 UART 口的最大传输速率 3.125Mbps，此方法并不适用于高刷新率的 ADC 数据获取，对于 1Hz 刷新率下的功能测试，串口传输的数据获取的传输时间能力如下：

(128(ADC samples per chirp) * 128(Chirps per frame)*4(RX channel)*4(IQ data bytes)*1(Frames per second) * 8(8bits)) / (3.125Mbps * 0.8(预估串口传输开销))= 0.83s

在使用 mmWave SDK 时，Frame 周期被限制为 300 us 到 1.342s，所以在使用 UART 进行数据获取时，需要注意控制数据总量，数据采集和传输时间不超过 RF Frame 的软件触发周期，即可实现连续的数据获取。若受到传输时间限制（如大于帧周期的限制），那么使用外部触发替代软件触发可解决问题，下图是使用 UART 串口进行数据采集的框图。

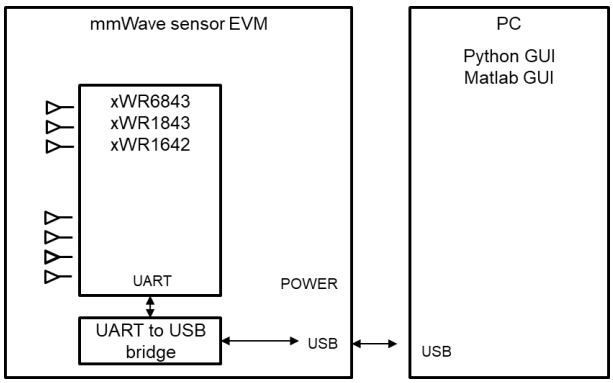

图 **4**. 使用 UART 串口进行数据采集框图

小结，使用 UART 串口进行数据采集的软硬件要求如下：

- 硬件：mmWave radar sensor EVM + PC
- 软件：需对 SDK 代码进行部分修改
- Radar 硬件所需预留接口：UART（ADC 数据获取、RF 参数配置及固件加载）

1.5. 本章小结

本章节描述的三种数据采集方法对比如下：

表 1. 三种数据采集方法对比表

	章节 1.2.使用 DCA1000+mmWave Studio 软件进行数据采集	章节 1.3.使用 DCA1000+mmWave Studio CLI tools 软件进行数据采集	章节 1.4.使用低速串行总线 UART 口进行数据采集
RF 配置工具	mmWave Studio 软件@ PC	mmWave Studio CLI tools 软件 @ PC 或 C 程序固化配置@器件	CLI 命令或者程序固化配置 @ 器件
ADC 数据输出接口	LVDS	LVDS	UART
所需硬件	DCA1000 数据采集卡	DCA1000 数据采集卡	USB←→UART 转换工具
所需软件	mmWave Studio 软件	mmWave Studio CLI tools 软件	没有要求
最大数据传输率	900Mbps	900Mbps	3.125Mbps
支持的刷新率	无限制	无限制	受限于数据传输率，一般为 1Hz

2. 使用 UART 串口进行数据采集的实现

2.1. 硬件框图

本文使用 IWR6843ISK EVM+MMWAVEICBOOST EVM 实现 ADC 数据的采集的硬件架构，其中，IWR6843 串口波特率最大为 3.125Mbps，为满足最大的串口波特率，需要保证串口<->USB 转换芯片也必须支持此数据率，IWR6843ISK 板卡上的 CP2105 转换芯片最大支持串口波特率为 2Mbps，所以需要使用 MMWAVEICBOOST 进行接口转换，使用板载 XDS110 仿真器的自带串口进行数据获取，XDS110 的串口波特率最大支持到 6Mbps，可以满足数据传输的速率需求。其中 MMWAVEICBOOST 转接板卡并非必须，可用使用其他的满足速率要求的 USB 转 UART 串口的转接板替代。

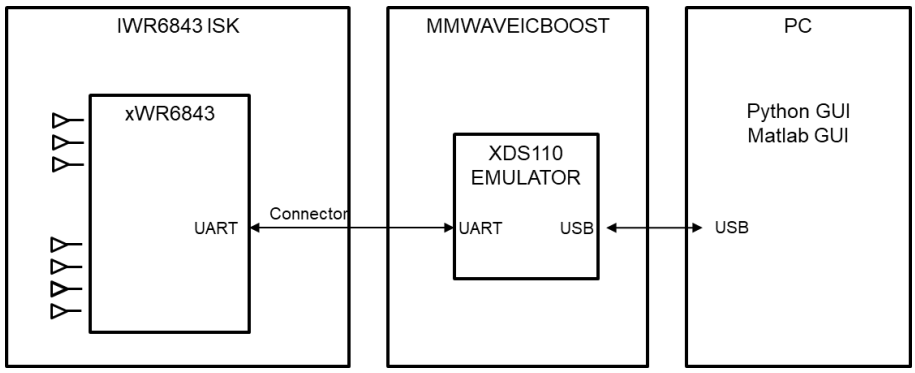

图 5. 使用 UART 串口进行数据的 TI EVM 硬件连接图

2.2. 软件框图

软件处理流程图如下图所示，考虑到 L3 中的 ADC 数据可以一直保留到下一帧 chirping 开始，所以保证把 L3 的 ADC 数据需要在这个时间内发出即可，具体的软件时序图如下图所示。帧周期的选择，需要考虑预留足够的 frame idle time 给到数据传输，受限于低速串行总线的速率及传输数据量，往往这个传输都需要数百毫秒量级，所以用户需要注意控制 frame 周期的设置。

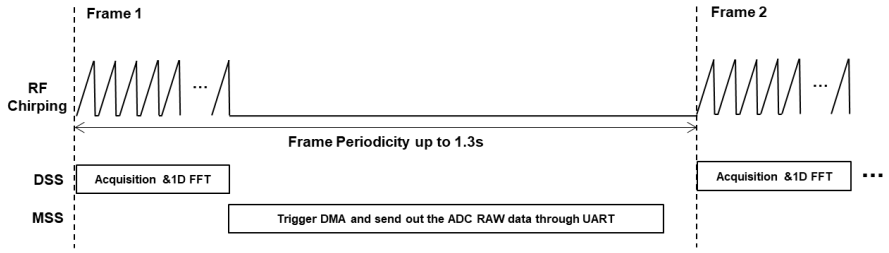

图 **6**. 软件处理时序图

2.3. 串口的工作模式选择

为满足数据传输速率的要求，串口的工作速率需要配置为 3125000bps（串口最大支持速率既是 3125000bps，参考 TRM），工作模式可以配置成 SDK 默认的普通模式，也可以打开 DMA，使用 DMA 进行数据搬移，将 R4F 的资源释放出来。

串口的普通模式配置代码如下：

```
/* Setup the default UART Parameters */
UART_Params_init(&uartParams);
uartParams.writeDataMode = UART_DATA_BINARY;
uartParams.readDataMode = UART_DATA_BINARY;
uartParams.clockFrequency = gMmwMssMCB.cfg.platformCfg.sysClockFrequency;
uartParams.baudRate       = gMmwMssMCB.cfg.platformCfg.loggingBaudRate;
uartParams.isPinMuxDone   = 1U;
/* Open the Logging UART Instance: */
gMmwMssMCB.loggingUartHandle = UART_open(1, &uartParams);
if (gMmwMssMCB.loggingUartHandle == NULL)
{
    System_printf("Error: Unable to open the Logging UART Instance\n");
    MmwDemo_debugAssert (0);
    return;
}
```

串口的 DMA 模式配置代码如下：

```
#define UART_DMA_TX_CHANNEL   1
#define UART_DMA_RX_CHANNEL   2
/* Open the DMA Instance */
DMA_Params_init(&dmaParams);
dmaHandle = DMA_open(0, &dmaParams, &errCode);
if (dmaHandle == NULL)
{
    printf ("Error: Unable to open the DMA Instance [Error code %d]\n", errCode);
    return;
}
/* Setup the default UART Parameters */
UART_Params_init(&uartParams);
uartParams.writeDataMode = UART_DATA_BINARY;
uartParams.readDataMode = UART_DATA_BINARY;
uartParams.clockFrequency = gMmwMssMCB.cfg.platformCfg.sysClockFrequency;
uartParams.baudRate       = gMmwMssMCB.cfg.platformCfg.loggingBaudRate;
uartParams.isPinMuxDone   = 1U;
uartParams.dmaHandle      = dmaHandle;
uartParams.txDMAChannel   = UART_DMA_TX_CHANNEL;
uartParams.rxDMAChannel   = UART_DMA_RX_CHANNEL;

/* Open the Logging UART Instance: */
gMmwMssMCB.loggingUartHandle = UART_open(1, &uartParams);
if (gMmwMssMCB.loggingUartHandle == NULL)
{
    System_printf("Error: Unable to open the Logging UART Instance\n");
    MmwDemo_debugAssert (0);
    return;
}
```

2.4. 数据通信格式的统一

与 SDK DEMO 代码通信协议兼容，新增加的 ADC 原始数据获取代码遵循与 SDK DEMO 一致的数据通信格式，以 TLV 的方式将 ADC 原始数据发出，在“MmwDemo_transmitProcessedOutput” TASK 中增加如下代码即可实现数据的外送，L3 的地址在 MSS 端既是 0x5100_0000 开始的，在 SDK DEMO 中，Radar ADC cube 数据既是存储在 L3 起始地址，若客制化代码中修改了 radar cube 的地址，那么此段代码也需要根据实际地址进行修改，SDK DEMO 中的 TLV type 在 MmwDemo_output_message_type 中定义。

表 2. TLV Type 定义说明

TLV TYPE	Description
1	MMWDEMO_OUTPUT_MSG_DETECTED_POINTS
2	MMWDEMO_OUTPUT_MSG_RANGE_PROFILE
3	MMWDEMO_OUTPUT_MSG_NOISE_PROFILE
4	MMWDEMO_OUTPUT_MSG_AZIMUT_STATIC_HEAT_MAP
5	MMWDEMO_OUTPUT_MSG_RANGE_DOPPLER_HEAT_MAP
6	MMWDEMO_OUTPUT_MSG_STATS
7	MMWDEMO_OUTPUT_MSG_DETECTED_POINTS_SIDE_INFO
8	MMWDEMO_OUTPUT_MSG_AZIMUT_ELEVATION_STATIC_HEAT_MAP
9	MMWDEMO_OUTPUT_MSG_TEMPERATURE_STATS
10(New Adding)	MMWDEMO_OUTPUT_MSG_L3_DATA
11	MMWDEMO_OUTPUT_MSG_MAX

新增 TLV 代码如下：

```
tl[tlvIdx].type = MMWDEMO_OUTPUT_MSG_L3_DATA;
tl[tlvIdx].length = subFrameCfg->numRangeBins * subFrameCfg->numDopplerBins *  subFrameCfg-
>numVirtualAntennas * 4 + 2;
subFrameCfg->numVirtualAntennas * 4 + 2;
packetLen += sizeof(MmwDemo_output_message_tl) + tl[tlvIdx].length;
tlvIdx++;
```

新增数据发送代码如下，使用 CRC16 作为校验方式，以保证数据传输的可靠性。

```
uint32_t *outdata;
outdata = 0x51000000;
UART_writePolling (uartHandle,(uint8_t*)&tl[tlvIdx],sizeof(MmwDemo_output_message_tl));
CRC16Bit = crc16_ccitt((uint8_t*)(&outdata[0]), tl[tlvIdx].length-2);
UART_writePolling(uartHandle, (uint8_t*)(&outdata[0]), tl[tlvIdx].length-2);
UART_writePolling(uartHandle, (uint8_t*)&CRC16Bit, 2);
```

2.5. ADC RAW DATA 数据格式

如章节 2.4 所描述的方法，即可将 Radar Cube 的数据送出，需要注意的是，在 frame processing 时送出的 radar cube 数据，是经过了 1D FFT 运算的数据，是 1D FFT 的结果，1D FFT 的运算发生在 chirp processing 阶段，在 IWR6843 的信号处理链路中，是使用 HWA 进行 1D FFT 的运算的，所以，如需要获得 1D FFT 运算之前的 ADC 数据，需要将 FFTEN 及 windowEN 设置修改为 0，需要修改 range processing 的 DPC 代码，示意如下，注意修改完成后需要重新编译 .\datapath\dpu\rangeproc 的库。

```
C:\ti\mmwave_sdk_03_05_00_04\packages\ti\datapath\dpu\rangeproc\src\rangeprochwa.c
static int32_t rangeProcHWA_ConfigHWA(...)
{
...
hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.fftEn = 0;//SDK default 1
hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.fftSize = mathUtils_ceilLog2(pDPParams->numRangeBins);
hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.butterflyScaling =
                                (1 << pDPParams->numLastButterflyStagesToScale) - 1U;
hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.interfZeroOutEn = 0;
hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.windowEn = 0;//SDK default 1
hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.windowStart = rangeProcObj->hwaCfg.hwaWinRamOffset;
hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.winSymm = rangeProcObj->hwaCfg.hwaWinSym;
hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.winInterpolateMode = 0;
hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.magLogEn = HWA_FFT_MODE_MAGNITUDE_LOG2_DISABLED;
hwaParamCfg[paramsetIdx].accelModeArgs.fftMode.fftOutMode = HWA_FFT_MODE_OUTPUT_DEFAULT;
```

```
hwaParamCfg[paramsetIdx].complexMultiply.mode = HWA_COMPLEX_MULTIPLY_MODE_DISABLE;
...
}
```

Radar cube 里的 ADC 原始数据的排布格式与 radarCubeFormat 参数设置直接相关，几种 data format 的区别可参考文档： “C:/ti/mmwave_sdk_<ver>/packages/ti/datapath/dpu/rangeproc/docs/doxygen/html/dpu_rangehwa.html” 本文不再赘述，代码中配置位于如下位置：

```
mss_main.c
static int32_t MmwDemo_dataPathConfig (void)
{
...
    /* objectdetection DSP DPC needs radacube in format DPIF_RADARCUBE_FORMAT_1 */
    staticCfg->radarCubeFormat = DPIF_RADARCUBE_FORMAT_1;
...
}
```

SDK demo 里使用的是 data format 1，其数据格式如下图所示，在 PC 接收到数据之后，可以采用对应的格式进行解码及二次处理。

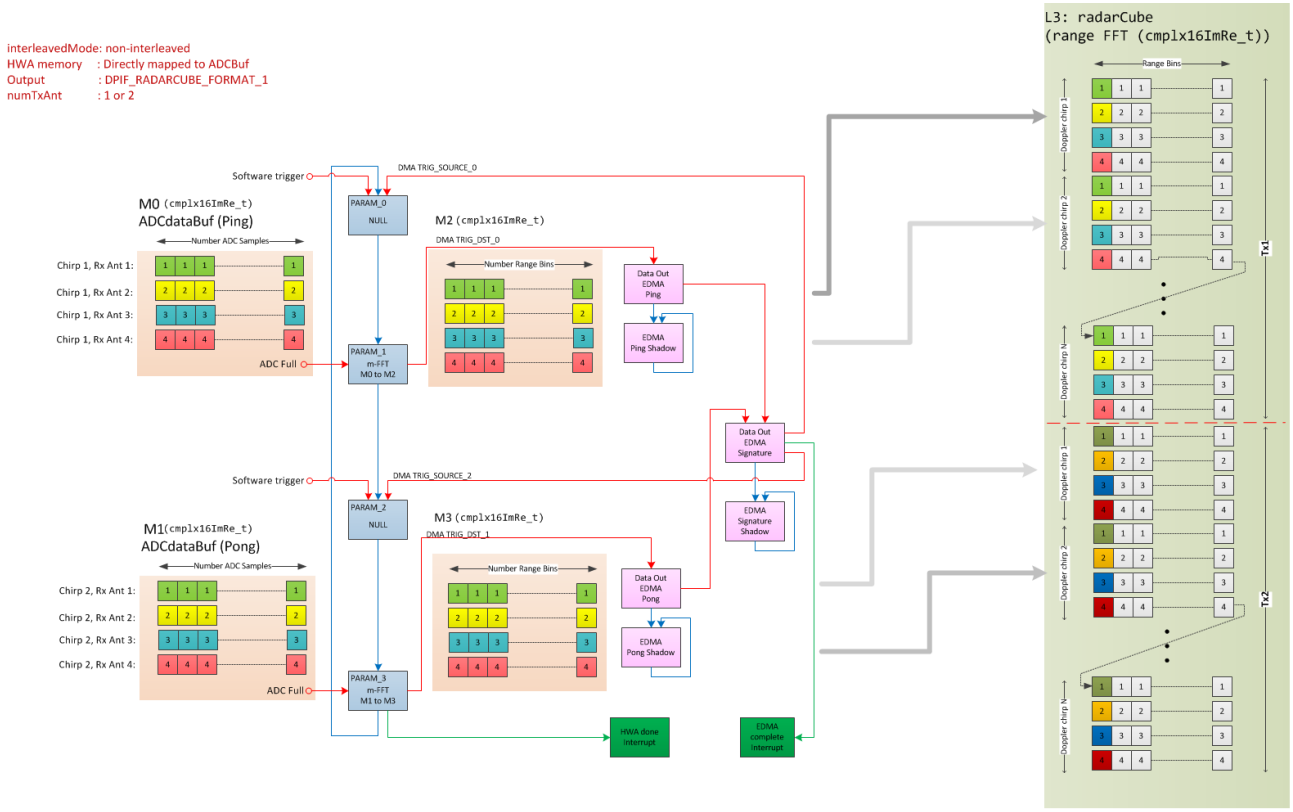

图 7. L3 radar cube 数据格式示意图

2.6. 上位机获取数据

上位机获取从雷达送出的数据，可以使用 TI 现有的 GUI 进行二次修改得到，亦可重新开发一款客制化的 GUI 以满足进一步的需求，本文将以 TI 现有的 GUI 进行示例，在现有 GUI 中增加 TLV 解码部分，将 ADC 数据提取出来，示意代码如下：

```
base on "C:\ti\mmwave_industrial_toolbox_4_8_0\labs\people_counting\visualizer\oob_parser.py"

def WriteFile(self, data):
    filepath=self.now_time + '.bin'
    objStruct = '6144B'
    objSize = struct.calcsize(objStruct)
```

```
        binfile = open(filepath, 'ab+')
        binfile.write(bytes(data))
        binfile.close()
    def parseL3Data(self, dataIn, profile):
            samples = profile['samples']
            numLoops = profile['numLoops']
            numTx = profile['numTx']
            objStruct = "%dh1H" % (numTx*numLoops*samples*4*2)
            objStruct2 = "%dB1H" % (numTx*numLoops*samples*4*4)
            objStructLength = struct.calcsize(objStruct)
            temp = np.zeros(objStructLength+1)
            temp2 = np.zeros((objStructLength+1)*2)
            try:
                temp = struct.unpack(objStruct,dataIn[:objStructLength])
                temp2 = struct.unpack(objStruct2,dataIn[:objStructLength])
                self.CRC16BIT = temp[int(numTx*numLoops*samples*2*4)]
                if (hex(self.CRC16BIT)) == (hex(getcrc(temp2[0:int(numTx*numLoops*samples*4*4)], list))):
                    for i in range(int(numTx*numLoops*samples)):
                        self.ADCRAW[i] = temp[2*i]*temp[2*i]+temp[2*i+1]*temp[2*i+1]
                    self.WriteFile(temp2[0:int(numTx*numLoops*samples*2*4)])
                    print('ADC RAW Succeed\r\n')
                else:
                    print('ADC RAW CRC ERROR!\r\n 6843-CRC:')
                    print(hex(self.CRC16BIT))
                    print('\r\n GUI-CRC:')
                    print (hex(getcrc(temp2[0:int(numTx*numLoops*samples*4*4)], list)))
                    temp2 = np.zeros(objStructLength*2)
                    self.WriteFile(temp2[0:int(numTx*numLoops*samples*2*4)])
            except Exception as e:
                print(e.args)
                print('\r\nADC RAW error!')
                print('dataInLength = ',len(dataIn))
                print('\r\n')
    #parsing for SDK 3.x Point Cloud
        def sdk3xTLVHeader(self, dataIn):
            #reset point buffers
            self.pcBufPing = np.zeros((5,self.maxPoints))
            headerStruct = 'Q8I'
            headerLength = struct.calcsize(headerStruct)
            tlvHeaderLength = 8
            #search until we find magic word
            while(1):
                try:
                    magic, version, totalPacketLen, platform, self.frameNum, timeCPUCycles,
    self.numDetectedObj, numTLVs, subFrameNum = struct.unpack(headerStruct, dataIn[:headerLength])
                except:
                    #bad data, return
                    self.fail = 1
                    return dataIn
                if (magic != self.magicWord):
                    #wrong magic word, increment pointer by 1 and try again
                    dataIn = dataIn[1:]
                else:
                    #we have correct magic word, proceed to parse rest of data
                    break
            dataIn = dataIn[headerLength:]
            #check to ensure we have all of the data
            while(1):
                remainingData = totalPacketLen - len(dataIn) - headerLength
                # print('remainingData '+str(int(remainingData)))
                if (remainingData > 0):
                    # print(remainingData)
                    newData = self.dataCom.read(remainingData)
                    dataIn += newData
                    self.oldData += newData
                else:
                    break
```

```
            count = 0
            #check to ensure we have all of the data
            while (remainingData > 0 and count < 3):
                newData = self.dataCom.read(remainingData)
                remainingData = totalPacketLen - len(dataIn) - len(newData)
                dataIn += newData
                count += 1
                if (self.saveBinary):
                    self.oldData += newData
            #now check TLVs
            for i in range(numTLVs):
                try:
                    tlvType, tlvLength = self.tlvHeaderDecode(dataIn[:tlvHeaderLength])
                except Exception as e:
                    print(e)
                    print ('failed to read OOB SDK3.x TLV')
                dataIn = dataIn[tlvHeaderLength:]
                if (tlvType == 1):
                    self.parseSDK3xPoints(dataIn[:tlvLength], self.numDetectedObj)
                elif (tlvType == 7):
                    self.parseSDK3xSideInfo(dataIn[:tlvLength], self.numDetectedObj)
                elif (tlvType == 10):
                    self.parseL3Data(dataIn[:tlvLength], self.profile)
                    # print('tlvType == 10,tlvLength=%d',tlvLength)
                dataIn = dataIn[tlvLength:]
            return dataIn

    def connectComPorts(self, uartCom, dataCom):
        self.uartCom = serial.Serial(uartCom,
    115200,parity=serial.PARITY_NONE,stopbits=serial.STOPBITS_ONE,timeout=0.3)
        if (self.capon3D == 1 and self.aop == 0):
            self.dataCom = serial.Serial(dataCom,
    921600*1,parity=serial.PARITY_NONE,stopbits=serial.STOPBITS_ONE,timeout=0.025)
        else:
            self.dataCom = serial.Serial(dataCom,
    3125000,parity=serial.PARITY_NONE,stopbits=serial.STOPBITS_ONE,timeout=0.01)
        self.dataCom.reset_output_buffer()
        print('Connected')
```

2.7. MATLAB 数据解析示例

获取原始数据 BIN 文件后，可以通过 MATLAB 解析并进行数据分析及验证，针对当前 radar cube data format 1 的情况，MATLAB 解析代码示意如下：

```
close all;
clear all;
file_path = '20211123-1524.bin';
numFramesRead = 5;
number_of_Channels = 8;
samplesPerChirp = 128*2;
chirpsPerFrame = 32;
fid = fopen(file_path, 'rb');
%%%%%%%%%%  Read the Binary Data File %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
readBufferSize = numFramesRead*number_of_Channels*samplesPerChirp*chirpsPerFrame;
CollectBuffer = fread(fid,readBufferSize, 'int16',0,'l') ;
radar_data = reshape(CollectBuffer, samplesPerChirp, number_of_Channels, chirpsPerFrame, numFramesRead
);%DPIF_RADARCUBE_FORMAT_1
% |cmplx16ImRe_t x[numTXPatterns][numDopplerChirps][numRX][numRangeBins] |1D Range FFT output

%%%%%%%%%%  Re-Format the DATA %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
radar_data_I_Channel = radar_data(2:2:samplesPerChirp, :, :, :);
radar_data_Q_Channel = radar_data(1:2:samplesPerChirp, :, :, :);
radar_data = radar_data_I_Channel + sqrt(-1)*radar_data_Q_Channel;
radar_data_cube = permute(radar_data,[1 3 2 4]);

%%%%%%%%%%  Plot %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%figure(1),mesh(abs(radar_data_cube(:,:,1,1)));%1DFFT data
figure(1),mesh(real(radar_data_cube(:,:,1,1)));%ADC RAW
%plot(imag(radar_data_cube(:,:,1,1)),'b');
legend('Real');
figure(2),plot(real(radar_data_cube(:,1,1,1)));
hold on,plot(imag(radar_data_cube(:,1,1,1)));
```

3. 测试及结果

3.1. 数据采集时的总线负载分析

如章节 1.4 所描述，传输一帧数据所需的时间预估为 0.83s，实际的总线占用率如下图所示，实际传输使用了 0.84s，符合预期，在帧周期设置为 1s 情况下，总线占用率为 84%，可以稳定的将 1Hz 刷新率的 ADC 原始数据送出，方便后续算法进行静态分析，可用于算法准确率的评估。

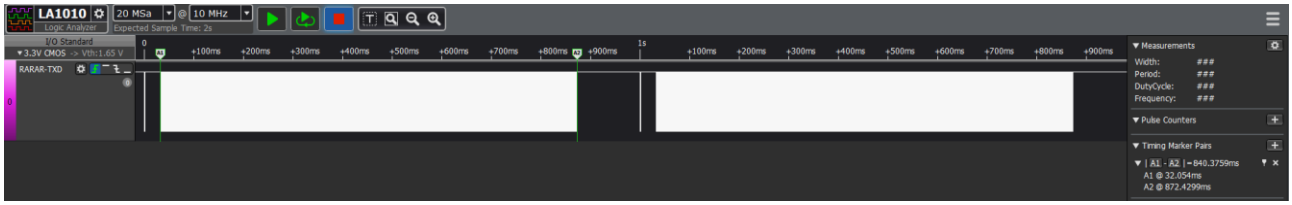

图 **8**. ADC 数据采集时的总线占用率实测图

3.2. 数据解析结果

MATLAB 解析的结果如下，可根据 range proc DPC 的设置，选择使用 1DFFT 运算后的结果，或者使用原始的 ADC 数据，可以成功获取数据并达到设计目标。

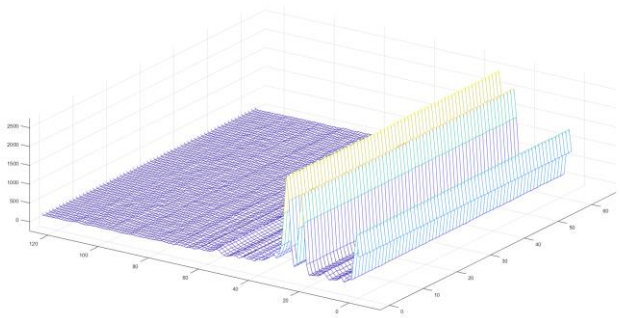

图 **9**. 获取 1D FFT 后的运算结果(abs)图

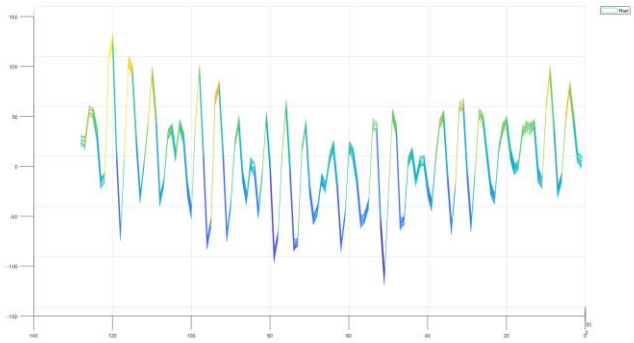

图 **10**. 获取 ADC 原始数据的实数时域波形图

4. 总结

本应用手册详细描述了三种 ADC 数据采集方式的对比，介绍了使用低速串行总线接口 UART 获取 ADC 原始数据的具体实现方法并提供了实现代码，可帮助用户快速实现 ADC 数据的获取及解析。

受限于 UART 口的最大传输速率 3125000bps，此方法在使用时需限制刷新率为 1Hz，可应用于静态场景的算法分析，对于动态目标的算法分析，还是需要预留高速 LVDS 接口，进行高刷新率的 ADC 数据获取，使用 DCA1000 数据采集卡进行数据采集。

对于没有预留高速 LVDS 接口的产品，本应用手册介绍的实现方法将可以实现较低成本下，并且非破坏性的对现有模块进行 ADC 数据采集，帮助对硬件、RF、及算法问题的快速定位及分析。

5. 参考文献

1. IWR6843 Datasheet
2. mmWave Software Development Kit Ver 3.5.0.4
3. IWR14xx/16xx/18xx/68xx/64xx Industrial Radar Family Technical Reference Manual
4. DCA1000EVM Data Capture Card User's Guide (Rev. A)
5. 60GHz mmWave Sensor EVMs (Rev. D)
6. XDS110 Debug Probe User's Guide

重要声明和免责声明

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、某特定用途方面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功能安全、信息安全、监管或其他要求。

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 产品的应用。严禁对这些资源进行其他复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。您应全额赔偿因在这些资源的使用中对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，TI 对此概不负责。

TI 提供的产品受 TI 的销售条款或 ti.com 上其他适用条款/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。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

邮寄地址：Texas Instruments, Post Office Box 655303, Dallas, Texas 75265