

AWR1843 DMM 接口介绍和验证测试

Chris Meng

Central mmWave FAE

ABSTRACT

硬件在环（Hardware in loop/HIL）被较多的使用在汽车等领域，用户可以方便的通过仿真系统进行实现嵌入式系统的开发和测试。TI 的单芯片毫米芯片 AWR1843 已经广泛的应用在汽车雷达中，硬件在环功能的支持也显得越来越重要。AWR1843 芯片集成了 DMM 接口，通过这个接口用户就可以实现硬件在环的功能。本文对 AWR1843 DMM 接口以及接口的验证测试进行了详细描述，可以作为用户实现自己硬件在环的有力参考。

Contents

1	DMM 接口介绍.....	2
1.1	DMM 的硬件接口.....	2
1.2	DMM 的工作模式.....	4
1.2.1	跟踪模式.....	4
1.2.2	直接数据模式.....	6
2	DMM 接口验证测试.....	7
2.1	验证测试流程.....	7
2.2	验证测试中使用的相关模块和中断.....	9
2.2.1	ADC 缓冲器.....	9
2.2.2	EDMA.....	9
2.2.3	帧/chirp 中断.....	10
2.3	验证测试相关时序图和 CCS 显示.....	10
3	DMM 接口调试小贴士.....	12
3.1	使用 CCS.....	12
3.2	跟踪模式调试.....	12
3.3	直接数据模式调试.....	13
3.4	其他注意事项.....	13
4	结束语.....	13
5	参考文献.....	13
6	附录 - 测试中 MSS 相关代码.....	13
6.1	宏定义和全局变量.....	13
6.2	中断处理函数.....	14
6.3	主要配置和控制代码.....	14
6.4	EDMA 初始化函数.....	16

Figures

Figure 1.	AWR1843BOOST 上 DMM 相关引脚和接口^[2].....	4
------------------	--	----------

Figure 2.	跟踪模式数据格式 ^[3]	4
Figure 3.	跟踪模式时序示意图（传输数据大小为 32 位）	5
Figure 4.	跟踪模式具体实例时序	6
Figure 5.	直接数据模式数据格式 ^[3]	6
Figure 6.	直接数据模式时序示意图（传输数据大小为 32 位）	7
Figure 7.	DMM 验证测试流程图	8
Figure 8.	AWR1843 ADC 缓冲器框图 ^[3]	9
Figure 9.	FPGA 输出 DMM_CLK 时序图	10
Figure 10.	FPGA 输出 DMM_SYNC 时序图	11
Figure 11.	EDMA OPT 参数初始化设置（CCS 中显示）	11
Figure 12.	L3 存储器在运行结束后结果（CCS 中显示）	12

Tables

Table 1	DMM 接口引脚	3
Table 2	SIZE[1:0]表示的传输数据量	5
Table 3	跟踪模式具体实例的数据内容	5

1 DMM 接口介绍

DMM 是 Data Modification Module 的缩写。利用这个接口，外部处理器例如 FPGA 可以直接把数据写入 AWR1843 的内部存储器。对于 AWR1843，用户可以把 ADC 原始数据（通过 AWR1843 的 LVDS 获取），或者是其他雷达仿真数据，利用 DMM 接口输入到 AWR1843 内，仅仅使用 AWR1843 的 MSS/DSS/HWA 对数据进行处理，而不使用 AWR1843 上的射频数据，从而可以实现硬件在环的功能。用户通过 DMM 接口可以在 AWR1843 上反复处理同一组数据，进行 AWR1843 上处理软件的功能验证。并且可以和 PC 处理同样的数据，对比输出结果，方便用户的雷达算法开发。

1.1 DMM 的硬件接口

DMM 的接口引脚在 AWR1843 上总共有 11 个，包括 8 根数据线，一个同步信号，一个时钟信号和一个 DMM 选择信号（DMM_MUX_IN）。

AWR1843 的内部有两个 DMM 的模块，DMM1 和 DMM2。可以通过输入 DMM_MUX_IN 引脚的信号电平高低来选择使用哪个 DMM 模块。当输入 DMM_MUX_IN 的信号为 0，表示当前使用 DMM1，如果为 1，表示使用 DMM2。

DMM_CLK 信号可以一直有效，或者只在数据传输时候有效。用户需要设置 DMMGLBCTRL 寄存器里的 CONTCLK 来选择时钟信号的输出方式。外部 FPGA 输出的 DMM 时钟信号，需要和寄存器配置的方式一致，不然会导致 DMM 操作失败。AWR1843 会在 DMM_CLK 下降沿的时候获取数据，所以外部 FPGA 需要在 DMM_CLK 上升沿的时候输出数据。本文验证测试时选择使用 DMM_CLK 在数据传输时才有效（DMMGLBCTRL.CONTCLK=0）。

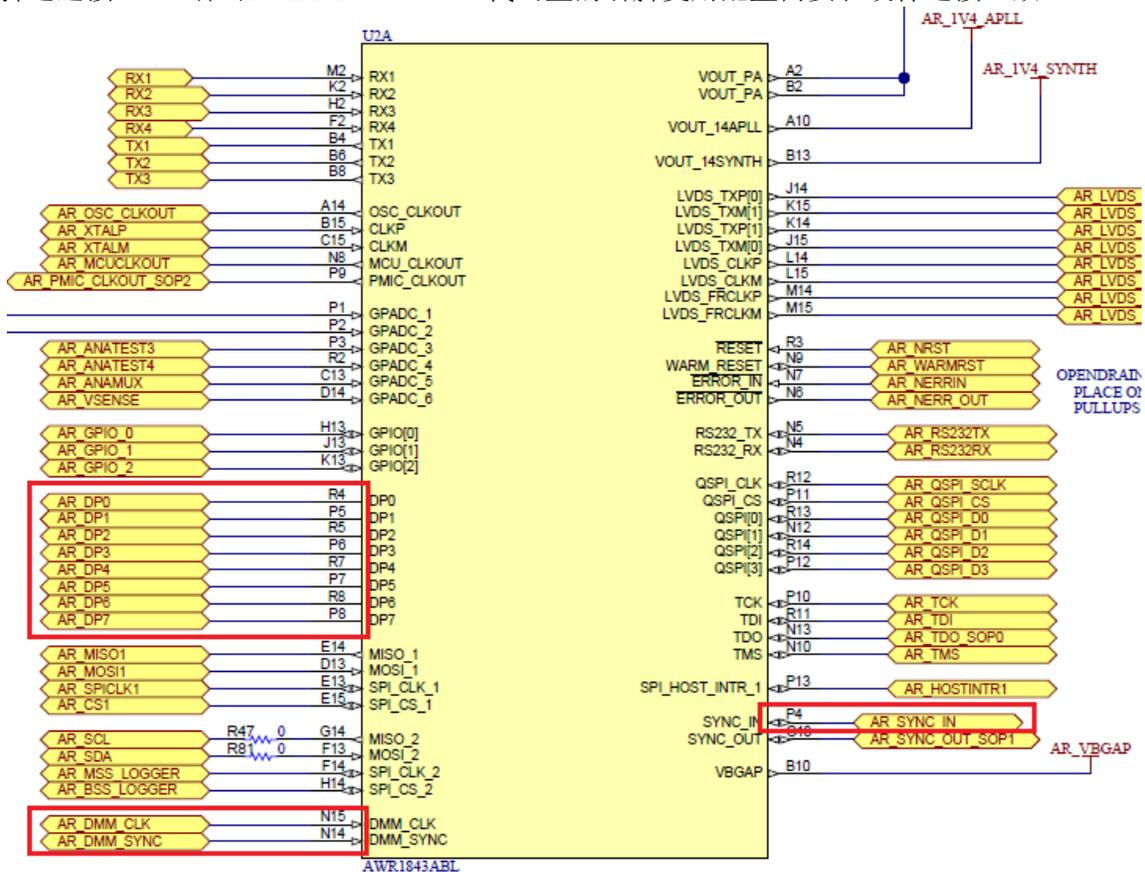
DMM_SYNC 的宽度只能是一个 DMM_CLK 周期。一次完整的数据传输过程中只能有一个同步信号。

信号名称	引脚类型	功能描述	引脚号
DMM_CLK	输入	时钟	N15

DMM_SYNC	输入	同步	N14
DMM0	输入	数据线	R4
DMM1	输入	数据线	P5
DMM2	输入	数据线	R5
DMM3	输入	数据线	P6
DMM4	输入	数据线	R7
DMM5	输入	数据线	P7
DMM6	输入	数据线	R8
DMM7	输入	数据线	P8
DMM_MUX_IN	输入	DMM1/DDM2 选择信号	G13, J13, P4

Table 1 DMM 接口引脚

在 AWR1843 的评估板 AWR1843BOOST 上, DMM_MUX_IN 使用的是 P4 引脚, 所有的 DMM 相关引脚通过接口 J1 引出。AWR1843 MSS 代码里的引脚复用配置需要和硬件连接一致。



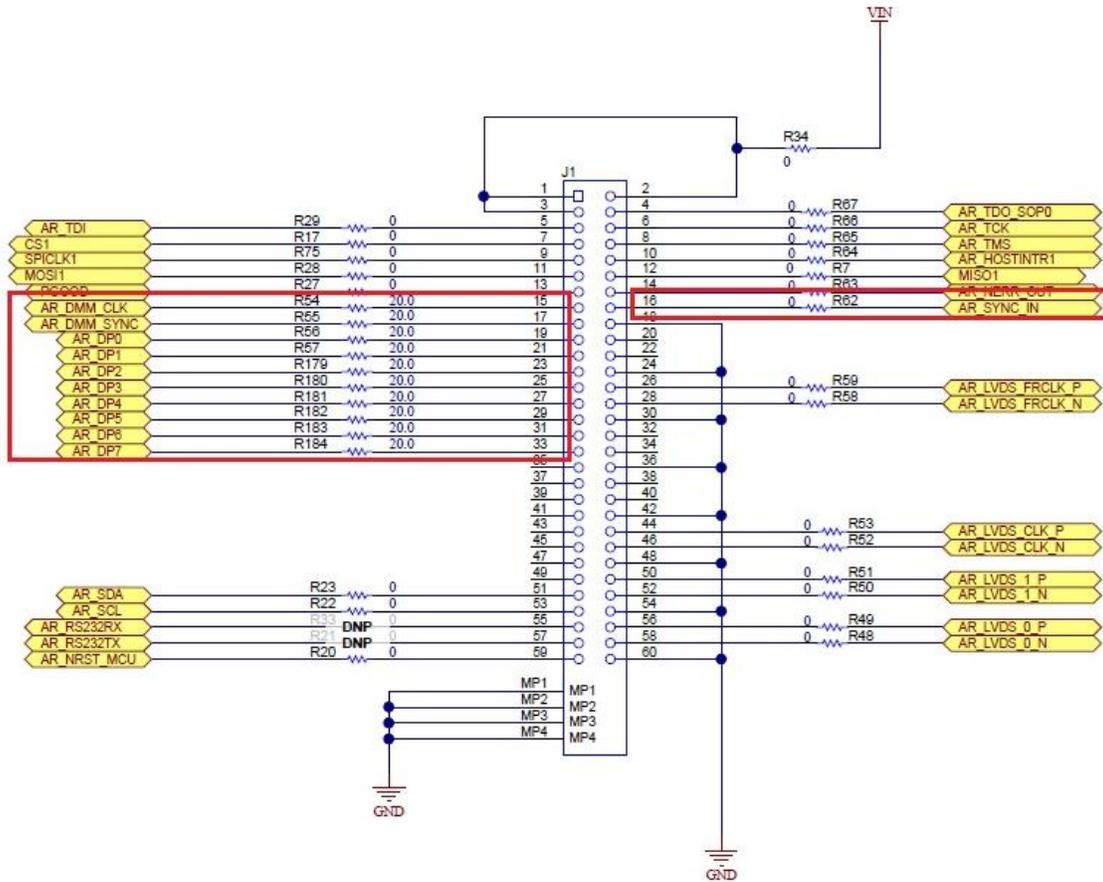


Figure 1. AWR1843BOOST 上 DMM 相关引脚和接口^[2]

1.2 DMM 的工作模式

DMM 可以工作在两种模式，跟踪模式（Trace mode）和直接数据模式（Direct Data Mode）。在跟踪模式下，DMM 接口把接收到的数据写入基于基地址偏移的地址里。通常这种模式用于少量数据写入，例如寄存器值的修改。在直接数据模式下，DMM 接口将接收到的数据写入到一个最大范围为 4G 字节的地址空间里。写入地址的初始地址，在 DMM 的寄存器 DMMDDMDEST 里定义，地址范围在寄存器 DMMDDMBL 里设定。所以大量数据写入的时候，会选择使用直接数据模式。在直接数据模式下内部地址会自动增加，DMM 数据线上只需要输入相关数据信息。

1.2.1 跟踪模式

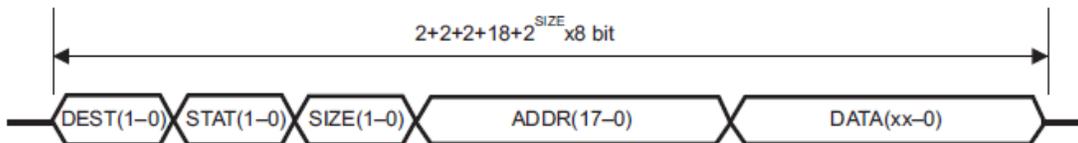


Figure 2. 跟踪模式数据格式^[3]

DEST[1:0] 表示选择使用的 DMM 跟踪模式下的基地址的寄存器。AWR1843 上总共有 4 个基地址寄存器可以使用。STAT[1:0]通常设置为 00。可以用于当外部设备发现异常后，输出 11，这样 AWR1843 会在 DMM Interrupt Flag Register (DMMINTFLG)的 SRC_OVF 位会置位。SIZE[1:0]表示传输数据的数据量，具体含义请参考表 2。ADDR[17:0]表示基于基地址的 18 位偏移地址。DATA[xx:0]是需要传输的具体数据内容。

SIZE[1:0]	传输数据量
00	8 位
01	16 位
10	32 位
11	64 位

Table 2 SIZE[1:0]表示的传输数据量

AWR1843 支持 8 根 DMM 数据线，如果在跟踪模式设置传输数据大小为 32 位，使用 DMM1 时相关信号的时序图如图 3 所示。

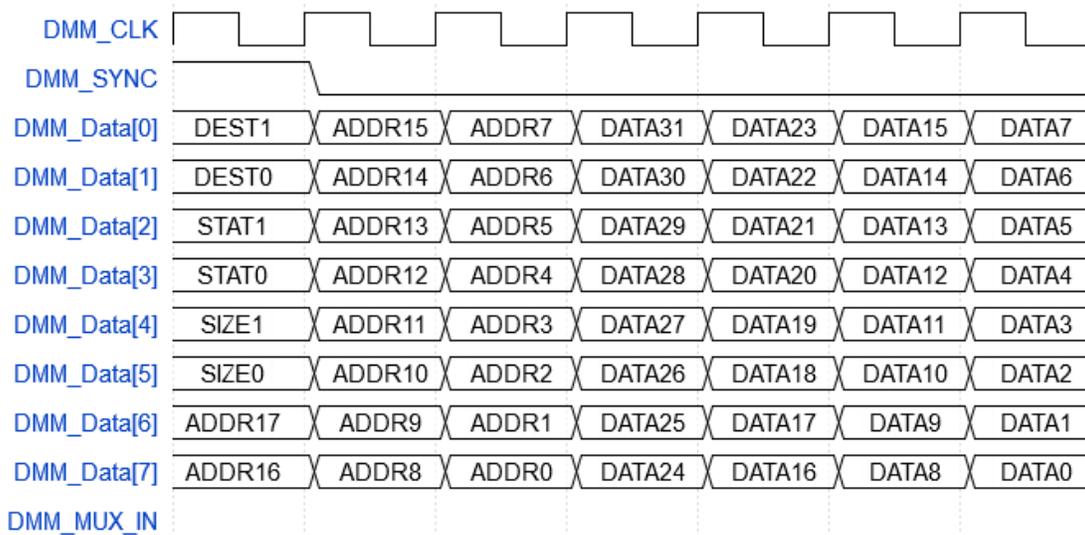


Figure 3. 跟踪模式时序示意图（传输数据大小为 32 位）

下面是一个使用 DMM1 跟踪模式的具体的例子，可以对跟踪模式的数据结构和对应 DMM 信号输出有一个更直观的了解。本例子中相关地址时基于 MSS 的内存分布视角，各个参数内容如表 3 所示。当 DMM1 的 DMMDEST2REG1 设置为 0xFFFFC000，DMM1 的 DMMDEST2BL1 设置为 0x9，然后在 DMM 的接口上输入图 4 所示的信号，AWR1843 MSS 地址 0xFFFF954（基地址：0xFFFFC000，偏移地址 0x3F954）的寄存器会被写入 0x880 的值。

跟踪模式具体实例	设置内容
DEST[1:0]	10b
STAT[1:0]	00b
SIZE[1:0]	10b
ADDR[17:0]	0x3F954
DATA[31:0]	0x00000880

Table 3 跟踪模式具体实例的数据内容

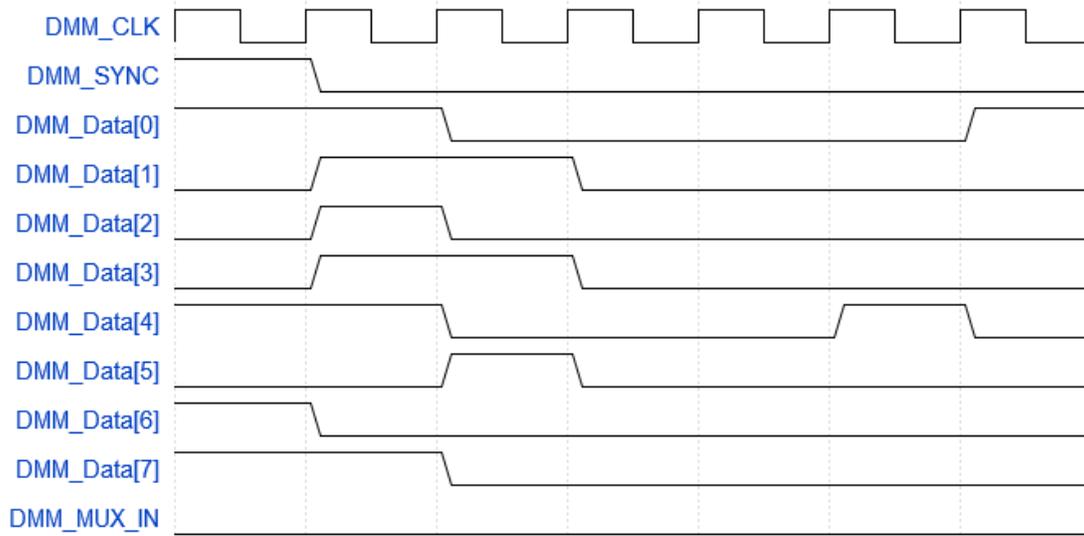


Figure 4. 跟踪模式具体实例时序

1.2.2 直接数据模式

使用直接数据模式，需要通过寄存器 DMMDDMDEST 配置直接数据模式需要访问地址的首地址，以及配置访问地址范围寄存器 DMMDDMBL。直接访问模式每次传输的数据位数也需要事先在 DMMGLBCTRL 的 DDM_WIDTH 中设置。直接访问模式数据格式如图 5 所示。在直接数据模式下也是需要通过 DMM_MUX_IN 来选择使用的 DMM。图 6 显示了选择使用 DMM2 时候通过直接数据模式在 DMM 接口上一次传输的 32 字节的时序。

当用户使用直接数据模式进行数据传输后，如果想从新的地址传输数据，而不是上次传输地址的后面，就需要复位 DMM 并重新配置 DMMDDMDEST 和 DMMDDMBL 寄存器。即使要回到寄存器设置的初始地址，也必须复位 DMM。

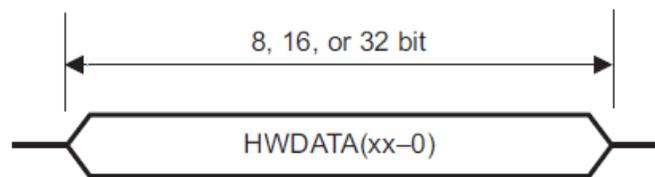


Figure 5. 直接数据模式数据格式^[3]

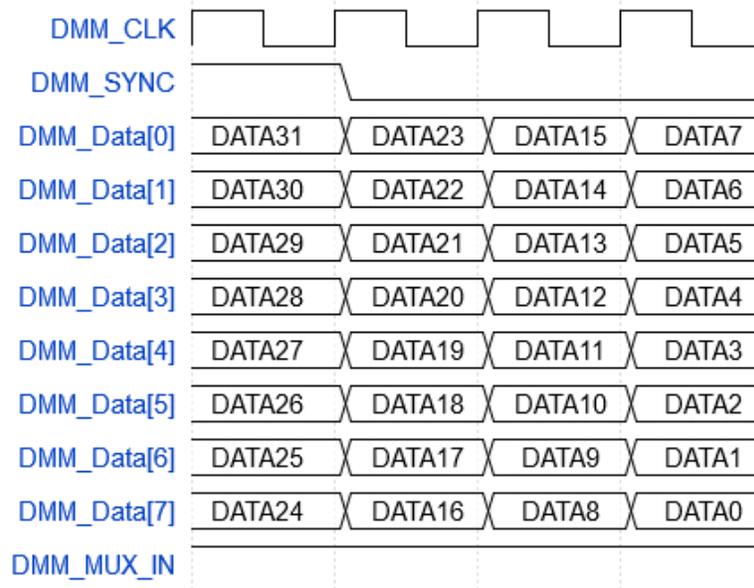


Figure 6. 直接数据模式时序示意图（传输数据大小为 32 位）

2 DMM 接口验证测试

本章节详细介绍了通过 DMM 接口进行 ADC 缓冲器数据获取的验证测试。

2.1 验证测试流程

为了验证 DMM 的功能，本文做了一个简化的 ADC buffer 数据获取测试。测试通过外部 FPGA 向 AWR1843 的 DMM 接口输入相应的数据到 AWR1843 的 ADC 缓冲器里。在 AWR1843 上完成通过 frame 中断/chirp 中断的中断函数，实现由 EDMA 将 ADC 缓冲器里的数据搬移到 L3 的过程。

测试假设外部输入 2 个 frame 的雷达数据，每一个 frame 有 2 个 chirp，每个 chirp 采样 128 个点，每个采样点采用 Complex 1x 采样，包含 16 位的实部和 16 位的虚部。

测试中 AWR1843 通过 GPIO2 和 FPGA 通信，通知 FPGA 是否 DMM 模块已经准备就绪。测试的流程图如图所示。流程图 7 (a) 显示的是 FPGA 的数据输出流程。流程图 7 (b) 显示的是 AWR1843 上 MSS 的处理流程。由于硬件的限制，流程图 7 中的 GPIO 读写在实际代码里没有实现。

在本次测试中，AWR1843 的操作都是在 MSS 侧完成。在实际的应用场景下，用户可以根据需要把相关代码移植到 DSS 侧。测试使用的 MSS 侧代码，请参考附录。测试中部分 DMM 相关寄存器的设置可改为外部 FPGA 通过 DMM 来完成，用户可以根据自己的需求做调整。

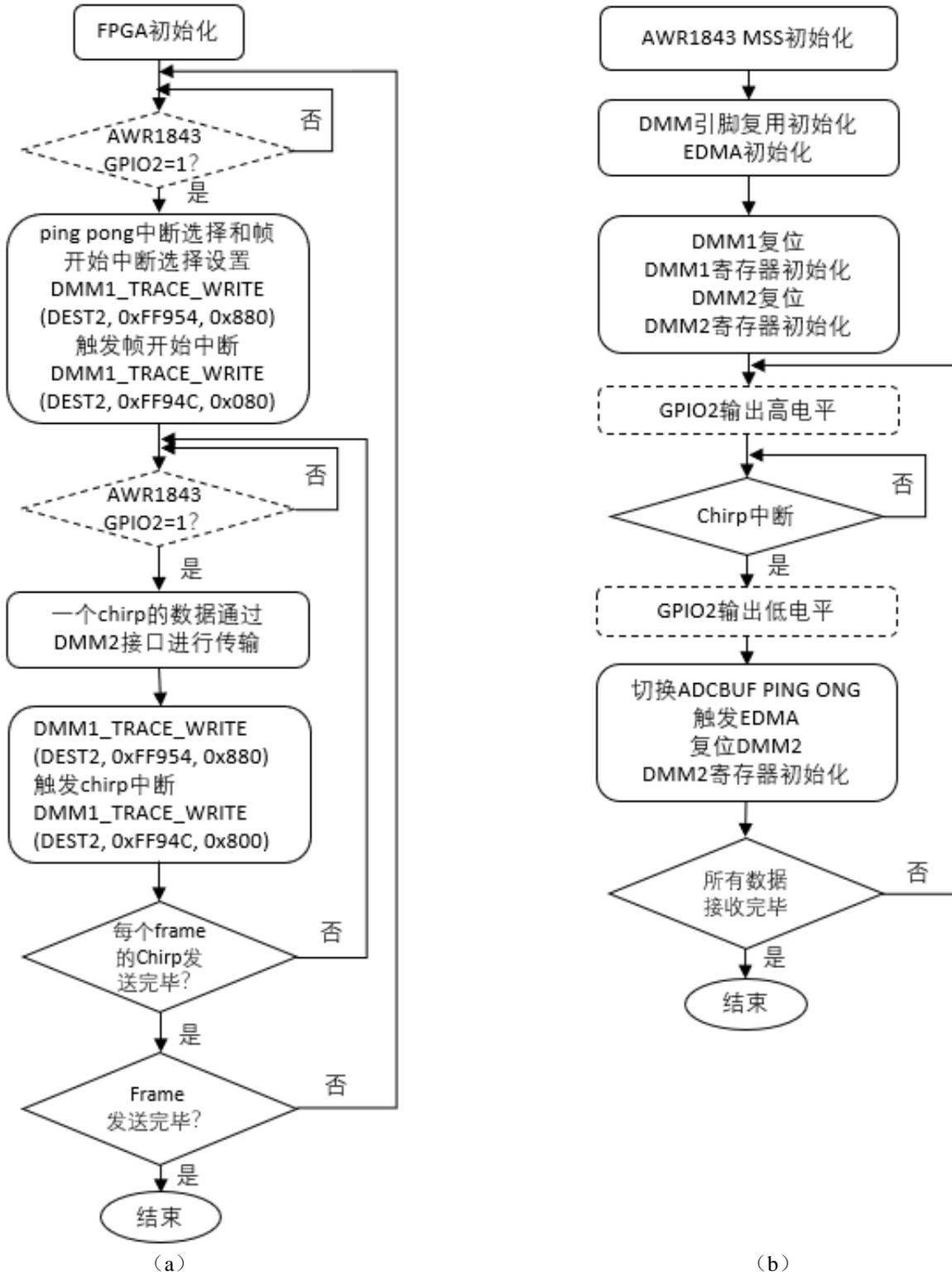


Figure 7. DMM 验证测试流程图

2.2 验证测试中使用的相关模块和中断

2.2.1 ADC 缓冲器

ADC 缓冲器 (ADC buffer) 是 AWR1843 芯片上用于存储毫米波中频原始数据的内部存储器，它支持 PING/PONG 工作模式。举例来说，当 PING 缓冲器在进行数据缓存的时候，PONG 缓冲区可以作为 DSP/HWA/EDMA 的数据输入，反之亦然。也就是说，在 AWR1843 上采集原始数据和处理可以并行进行。

从图 8 可知，ADC 缓冲器的输入可以是数字前端输出 DFE (通过芯片射频获得)，或者是芯片内部测试序列产生器，或者是 DMM 的写入接口输入的数据。用户需要通过寄存器 `DSS_REG.DMMSWINT1.DMMADCBUFWREN` (第 17 位) 进行设置来决定 ADC 缓冲器的输入源。当该寄存器位设置为 0，表示 ADC 缓冲器的输入是 DFE/测试序列，PING/PONG 缓冲器的选择由 HW 序列器来决定。而当该寄存器设置为位 1，表示 ADC 缓冲器的输入是 DMM，并且 PING/PONG 缓冲器的选择由 `DSS_REG` 里的 `DMMSWINT1.DMMADCBUFPINPONSEL` (第 16 位) 来决定。当 `DSS_REG.DMMSWINT1.DMMADCBUFPINPONSEL` 设置位为 0，表示使用 PING 缓冲器，设置为 1，表示使用 PONG 缓冲器。

下面是选择 DMM 输入为 PING 缓冲器的参考代码：

```
WR_MEM_32(DSS_REG_DMMSWINT1,0x420000);
```

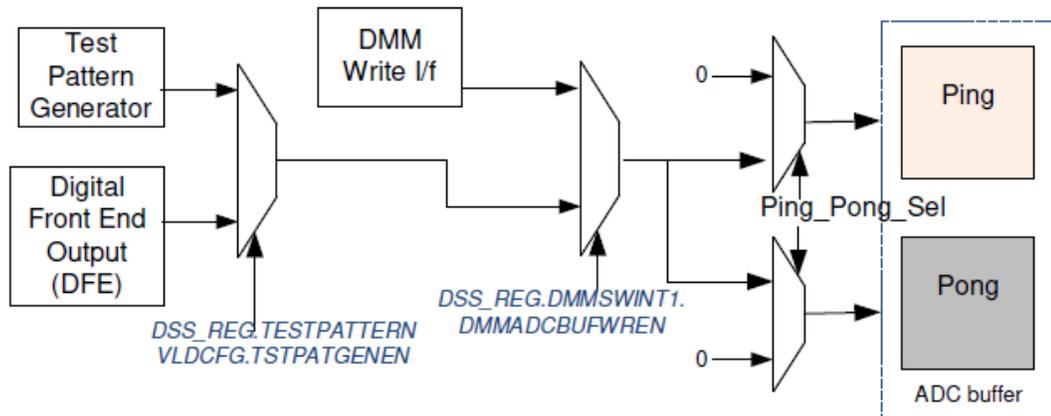


Figure 8. AWR1843 ADC 缓冲器框图^[3]

在 TRM (参考文献[3]) 里用户可以在 MSS 的内存分布 (memory map) 里找到 ADC 缓冲器的起始地址为 `0x52000000`，这是一个只读的地址。DMM 需要时使用 `0x52090000` 这个 ADC 缓冲器可写入的地址。当图 8 中选择了 PING 缓冲器，也就是 DMM 会通过地址 `0x52090000` 写入 PING 缓冲器，那用户在 MSS 的 `0x52000000` 起始的地址里看到的内容是 PONG 缓冲器的内容。

2.2.2 EDMA

EDMA 是增强型直接存储器访问的缩写。在本测试例程里，使用 EDMA 将 ADC 缓冲器中的数据搬运到 AWR1843 的 L3 存储器里。

根据本次测试假设的输入的原始数据（两帧/每帧有 2 个 chirp/每个 chirp 采样 128 个点/每个采样点 4 个字节），EDMA 的 ACOUNT 应为每个采样点的数据位数*每个 chirp 采样点数，配置为 4*128（字节），BCOUNT 是总共的 chirp 数，配置为 2*2，COUNT 配置为 1。由于 ADC 缓冲器使用的是 PING/PONG 机制，所以源地址是不变的，配置里设置源的 BINDEX 为 0。而目的地址的起始地址是 L3 存储器区域，目的 BINDEX 为一个 chirp 的数据量 4*128（字节）。EDMA 的同步模式配置为 ASYNC 模式。

2.2.3 帧/chirp 中断

通过图 7（a）的 FPGA 流程图可以了解到，在本测试中 AWR1843 在每个帧开始和每个 chirp 结束都会收到相关的中断信号。本测试利用 chirp 中断设置标志，在任务里通过判断这个标志来手动触发 EDMA 的数据搬移。每一个 chirp 数据触发一次 EDMA，这也是 EDMA 设置为 ASYNC 模式的原因。

2.3 验证测试相关时序图和 CCS 显示

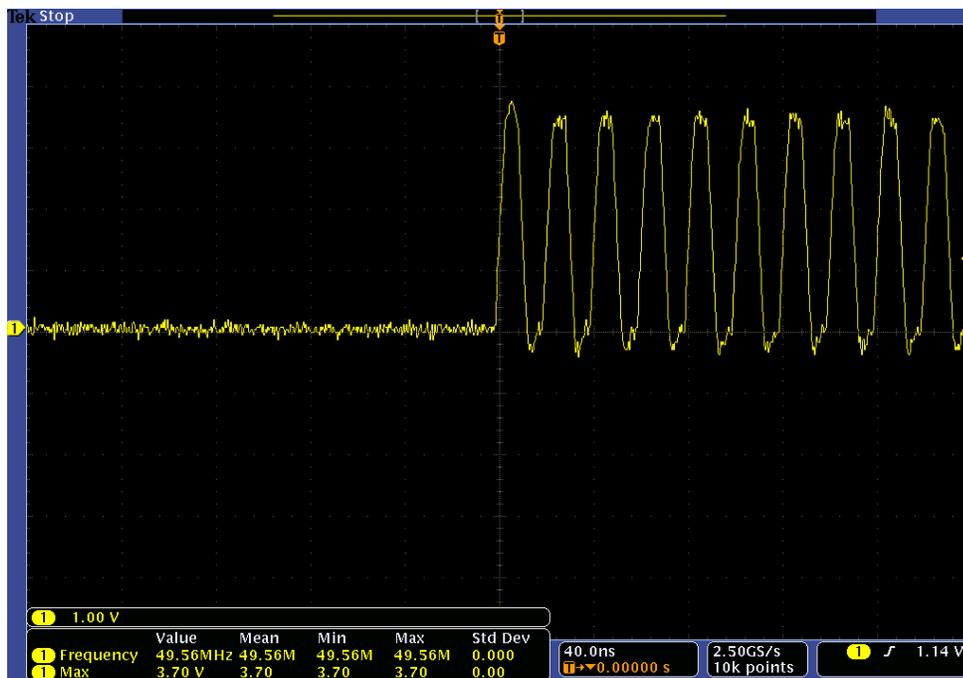


Figure 9. FPGA 输出 DMM_CLK 时序图

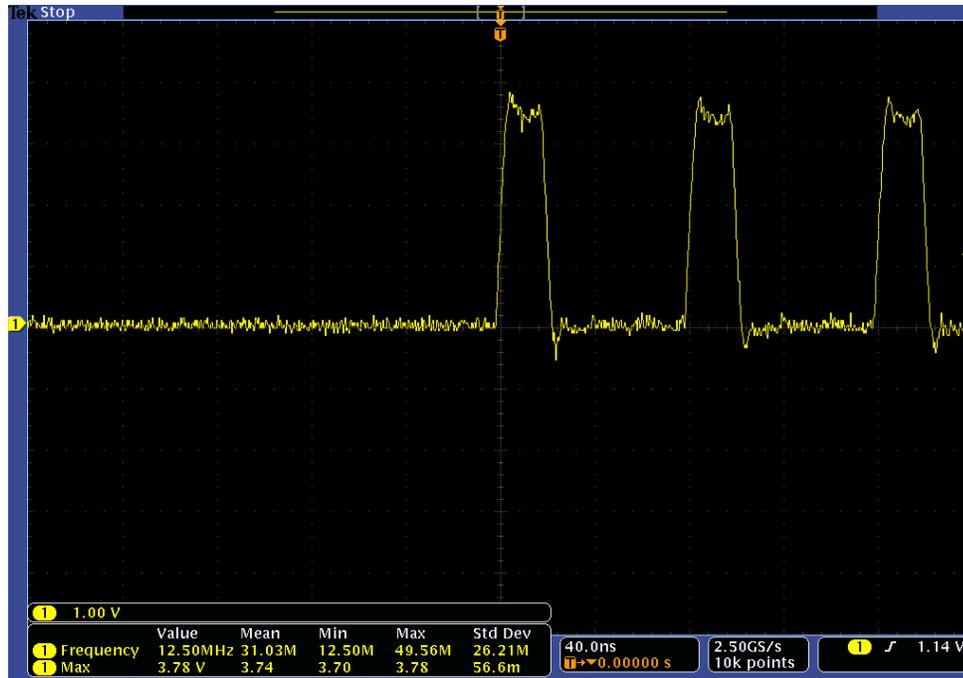


Figure 10. FPGA 输出 DMM_SYNC 时序图

```

0x500140E0 TPCC0_PARAMSET_7_OPT
0x500140E0 80307000
0x500140E4 TPCC0_PARAMSET_7_SRC
0x500140E4 21000000
0x500140E8 TPCC0_PARAMSET_7_A_B_CNT
0x500140E8 00040200
0x500140EC TPCC0_PARAMSET_7_DST
0x500140EC 20000000
0x500140F0 TPCC0_PARAMSET_7_SRC_DST_BIDX
0x500140F0 02000000
0x500140F4 TPCC0_PARAMSET_7_LINK_BCNTRLD
0x500140F4 0000FFFF
0x500140F8 TPCC0_PARAMSET_7_SRC_DST_CIDX
0x500140F8 00000000
0x500140FC TPCC0_PARAMSET_7_CCNT
0x500140FC 00000001
    
```

Figure 11. EDMA OPT 参数初始化设置 (CCS 中显示)

3.3 直接数据模式调试

建议先尝试使用直接模式修改 AWR1843 的可写内存。同样建议通过 MSS 代码，或者是 GEL 文件直接配置好 DMM 相关寄存器。FPGA 输出内容为特殊数字。用户在 CCS 里，通过访问配置好的直接数据模式的写入地址，观察数据是否被正确写入。如果写入地址是 ADC 缓冲器，需要注意 ADC 缓冲器工作在 PING/PONG 模式下。当通过 DMM 接口写入数据到 PING，用户在 MSS 的 0x52000000 地址是看不到写入的数据的，这个时候看到的是 ADC 缓冲器 PONG。用户需要手动切换 PING/PONG，才能在 0x52000000 看到正确的数据。

由于在中频原始数据传输中需要复位 DMM，用户需要保证对 DMM 的寄存器操作是在 DMM 复位完成后。所以在 AWR1843 和 FPGA 之间需要通过额外的 GPIO 口，或者其他通信方式来做同步。

3.4 其他注意事项

通常采用 DFE 获取 ADC 数据的时候，接收天线会有多根。用户可以通过 DSS_REG 里的 ADCBUFCFG2/3 寄存器来设定每个接收天线接收到的数据之前的偏移。这两个寄存器会在 ADC 缓冲器配置的时候进行配置。在 MRR 的例程里，这个偏移设置的是 8K，如果客户想使用 MMR 例程为基础，利用 DMM 输入 ADC 数据，那就需要注意这个偏移，在不同的偏移地址上通过 DMM 输入不同接收天线的的数据。

本测试里是一个 chirp 产生一个 chirp 中断，如果用户设置 CHIRPTHRESHOLD 值（DSS_REG 里的 ADCBUFCFG4. ADCBUFCFG4. ADCBUFCFG4. ADCBUFCFG4 寄存器）不是 1，而是其他值，那在 FPGA 代码里的数据传输和中断产生需要做相应的修改。

4 结束语

本文提供了利用 DMM 进行数据输入的测试例程。如果需要了解更多 DMM 相关的信息，包括不同命令间的延时要求，建议用户阅读参考文献[4]、[5]里的相关资料。

本文提供的 DMM 相关信息，也适用于 TI 其他毫米波芯片，但需要注意不同的毫米波芯片支持的最大 DMM 数据总线宽度不同，用户需要根据使用的具体芯片来对寄存器配置进行相应调整。

5 参考文献

1. AWR1843 datasheet (<https://www.ti.com/lit/gpn/awr1843>)
2. AWR1843BOOST 原理图 (<https://www.ti.com/lit/zip/sprr370>)
3. AWR18xx/16xx/14xx/68xx Technical Reference Manual (<https://www.ti.com/lit/pdf/swru520>)
4. DMM interface V0.3.pdf (<https://e2e.ti.com/support/sensors/f/1023/t/861828>)
5. DMM_Configuration_Pseudo_code_rev2.pdf (<https://e2e.ti.com/support/sensors/f/1023/t/861828>)

6 附录 - 测试中 MSS 相关代码

6.1 宏定义和全局变量

```
#define WR_MEM_32(addr, data)      *(unsigned int*)(addr) = (unsigned int)(data)
#define RD_MEM_32(addr)           *(unsigned int*)(addr)
#define DSS_REG_DMMSWINT1 0x50000660
#define MSS_CPCFG_REG 0xFFFFF800
```

```

#define DMMSWINT1 0x14c
#define DMMSWINTSEL1 0x154
/*---DMM register address*/
#define DMM1_baseaddress 0xFCFFF700
#define DMM2_baseaddress 0xFCFFF600
//offset address
#define DMMGLBCTRL 0
#define DMMDDMDEST 0x1C
#define DMMDDMBL 0x20
#define DMMPC0 0x6C
#define DMMPC8 0x8C
#define DEST0REG1 0x2C
#define DEST1REG1 0x3C
#define DEST2REG1 0x4C
#define DEST0BL1 0x30
#define DEST1BL1 0x40
#define DEST2BL1 0x50
#pragma DATA_SECTION(gDataCube, ".dataCubeMemory");
#pragma DATA_ALIGN(gDataCube, 8);
uint8_t gDataCube[2048];
#define BYTES_PER_SAMP_1D (2*sizeof(int16_t)) /*16 bit real, 16 bit imaginary => 4 bytes */

```

6.2 中断处理函数

```

static void MRR_MSS_chirpIntCallback(uintptr_t arg)
{
    gMrrMSSMCB.chirpInt++;
}
static void MRR_MSS_frameStartIntCallback(uintptr_t arg)
{
    gMrrMSSMCB.frameStartToken++;
}

```

6.3 主要配置和控制代码

```

static void DMM_MSS_mmWaveCtrlTask (UArg arg0, UArg arg1)
{
    int ping=0;//pong =1;
    Pinmux_Set_OverrideCtrl(SOC_XWR18XX_PINP4_PADBB, PINMUX_OUTEN_RETAIN_HW_CTRL,
PINMUX_INPEN_RETAIN_HW_CTRL);
    Pinmux_Set_FuncSel(SOC_XWR18XX_PINP4_PADBB,SOC_XWR18XX_PINP4_PADBB_DMM_MUXIN);
    Pinmux_Set_OverrideCtrl(SOC_XWR18XX_PINR4_PADBF, PINMUX_OUTEN_RETAIN_HW_CTRL,
PINMUX_INPEN_RETAIN_HW_CTRL);
    Pinmux_Set_FuncSel(SOC_XWR18XX_PINR4_PADBF,SOC_XWR18XX_PINR4_PADBF_DMM0);
    Pinmux_Set_OverrideCtrl(SOC_XWR18XX_PINP5_PADBG, PINMUX_OUTEN_RETAIN_HW_CTRL,
PINMUX_INPEN_RETAIN_HW_CTRL);
    Pinmux_Set_FuncSel(SOC_XWR18XX_PINP5_PADBG,SOC_XWR18XX_PINP5_PADBG_DMM1);
    Pinmux_Set_OverrideCtrl(SOC_XWR18XX_PINR5_PADBH, PINMUX_OUTEN_RETAIN_HW_CTRL,
PINMUX_INPEN_RETAIN_HW_CTRL);
    Pinmux_Set_FuncSel(SOC_XWR18XX_PINR5_PADBH,SOC_XWR18XX_PINR5_PADBH_DMM2);
    Pinmux_Set_OverrideCtrl(SOC_XWR18XX_PINP6_PADBI , PINMUX_OUTEN_RETAIN_HW_CTRL,
PINMUX_INPEN_RETAIN_HW_CTRL);
    Pinmux_Set_FuncSel(SOC_XWR18XX_PINP6_PADBI,SOC_XWR18XX_PINP6_PADBI_DMM3);
    Pinmux_Set_OverrideCtrl(SOC_XWR18XX_PINR7_PADBJ, PINMUX_OUTEN_RETAIN_HW_CTRL,
PINMUX_INPEN_RETAIN_HW_CTRL);
}

```

```

Pinmux_Set_FuncSel(SOC_XWR18XX_PINR7_PADBJ,SOC_XWR18XX_PINR7_PADBJ_DMM4);
Pinmux_Set_OverrideCtrl(SOC_XWR18XX_PINP7_PADBK, PINMUX_OUTEN_RETAIN_HW_CTRL,
PINMUX_INPEN_RETAIN_HW_CTRL);
Pinmux_Set_FuncSel(SOC_XWR18XX_PINP7_PADBK,SOC_XWR18XX_PINP7_PADBK_DMM5);
Pinmux_Set_OverrideCtrl(SOC_XWR18XX_PINR8_PADBL, PINMUX_OUTEN_RETAIN_HW_CTRL,
PINMUX_INPEN_RETAIN_HW_CTRL);
Pinmux_Set_FuncSel(SOC_XWR18XX_PINR8_PADBL,SOC_XWR18XX_PINR8_PADBL_DMM6);
Pinmux_Set_OverrideCtrl(SOC_XWR18XX_PINP8_PADBM, PINMUX_OUTEN_RETAIN_HW_CTRL,
PINMUX_INPEN_RETAIN_HW_CTRL);
Pinmux_Set_FuncSel(SOC_XWR18XX_PINP8_PADBM,SOC_XWR18XX_PINP8_PADBM_DMM7);
Pinmux_Set_OverrideCtrl(SOC_XWR18XX_PINN15_PADBV, PINMUX_OUTEN_RETAIN_HW_CTRL,
PINMUX_INPEN_RETAIN_HW_CTRL);
Pinmux_Set_FuncSel(SOC_XWR18XX_PINN15_PADBV,SOC_XWR18XX_PINN15_PADBV_DMM_CLK);
Pinmux_Set_OverrideCtrl(SOC_XWR18XX_PINN14_PADBW, PINMUX_OUTEN_RETAIN_HW_CTRL,
PINMUX_INPEN_RETAIN_HW_CTRL);
Pinmux_Set_FuncSel(SOC_XWR18XX_PINN14_PADBW,SOC_XWR18XX_PINN14_PADBW_DMM_SYNC);
//DMM1 reset
Task_sleep(1);
WR_MEM_32(DMM1_baseaddress+ DMMGLBCTRL , 0x10000U);
WR_MEM_32(DMM1_baseaddress+ DMMGLBCTRL , 0x0000U);
/* DMM1 Configuration in Functional Mode */
//8-bit data line for AWR1843: 0x403FFU, 16-bit lines for AWR1642: 0x7FFFF
WR_MEM_32(DMM1_baseaddress+ DMMPC0 , 0x403FFU);
WR_MEM_32(DMM1_baseaddress+ DMMPC8 , 0x403FFU);

/* DMM1 Configuration in TRACE-Mode */
//WR_MEM_32(DMM1_baseaddress+ DMMGLBCTRL , 0x0000000AU);
WR_MEM_32(DMM1_baseaddress+ DMMGLBCTRL , 0x0002000AU);//2000A - enable in debug mode
//set DMM1 DEST address for trace mode
WR_MEM_32(DMM1_baseaddress+ DEST0REG1, 0xFCFC0000U);
WR_MEM_32(DMM1_baseaddress+ DEST0BL1, 0x0009U);
WR_MEM_32(DMM1_baseaddress+ DEST1REG1 , 0x50000000U);
WR_MEM_32(DMM1_baseaddress+ DEST1BL1 , 0x0009U);
WR_MEM_32(DMM1_baseaddress+ DEST2REG1 , 0xFFFF0000U);
WR_MEM_32(DMM1_baseaddress+ DEST2BL1 , 0x0009U);

/* DMM2 Configuration in direct access -Mode */
//DMM2 reset
WR_MEM_32(DMM2_baseaddress+ DMMGLBCTRL , 0x10000U);
WR_MEM_32(DMM2_baseaddress+ DMMGLBCTRL , 0x0000U);
/* DMM2 Configuration in Functional Mode */ //8-bit data line for AWR1843: 0x403FFU, 16-bit lines for
AWR1642: 0x7FFFF
WR_MEM_32(DMM2_baseaddress+ DMMPC0 , 0x403FFU);
WR_MEM_32(DMM2_baseaddress+ DMMPC8 , 0x403FFU);
WR_MEM_32(DMM2_baseaddress+ DMMGLBCTRL , 0x0002050AU);
WR_MEM_32(DMM2_baseaddress+ DMMDDMDEST,0x52090000);//0x52090000U);
WR_MEM_32(DMM2_baseaddress+ DMMDDMBL, 0xB);
//init to select ping buffer
WR_MEM_32(DSS_REG_DMMSWINT1,0x420000);
while (1)
{
if (gMrrMSSMCB.frameStartToken == 1)
{
gMrrMSSMCB.frameStartToken = 0;
}
}

```

```

    /* Increment event stats */
    gMrrMSSMCB.subframeCntFromFrameStart++;
}
else if (gMrrMSSMCB.chirpInt == 1)
{
    gMrrMSSMCB.chirpIntcumSum++;
    if (ping==0)
    {
        WR_MEM_32(DSS_REG_DMMSWINT1,0x630000);
        ping=1;
    }
    else
    {
        WR_MEM_32(DSS_REG_DMMSWINT1,0x420000);
        ping=0;
    }
    //add code to trigger EDMA
    EDMA_startDmaTransfer(gMrrMSSMCB.dmaHandle, EDMA_TPCC0_REQ_FREE_0);
    gMrrMSSMCB.chirpInt--;
    //DMM2 reset
    WR_MEM_32(DMM2_baseaddress+ DMMGLBCTRL , 0x10000U);
    WR_MEM_32(DMM2_baseaddress+ DMMGLBCTRL , 0x0000U);
    /* DMM2 Configuration in Functional Mode */
    //8-bit data line for AWR1843: 0x403FFU, 16-bit lines for AWR1642: 0x7FFFF
    WR_MEM_32(DMM2_baseaddress+ DMMPC0 , 0x403FFU);
    WR_MEM_32(DMM2_baseaddress+ DMMPC8 , 0x403FFU);
    WR_MEM_32(DMM2_baseaddress+ DMMGLBCTRL , 0x0002050AU);
    WR_MEM_32(DMM2_baseaddress+ DMMDDMDEST,0x52090000);
    WR_MEM_32(DMM2_baseaddress+ DMMDDMBL, 0xB);
    if (gMrrMSSMCB.chirpIntcumSum ==4)
    {
        goto exit;
    }
}
}
exit: return;
}

```

6.4 EDMA 初始化函数

```

int32_t EDMAutil_configType1(EDMA_Handle handle,
    uint8_t *srcBuff,
    uint8_t *dstBuff,
    uint8_t chld,
    bool isEventTriggered,
    uint16_t shadowParamId,
    uint16_t aCount,
    uint16_t bCount,
    int16_t srcBIdx,
    int16_t dstBIdx,
    uint8_t eventQueueId,
    EDMA_transferCompletionCallbackFxn_t transferCompletionCallbackFxn,
    uintptr_t transferCompletionCallbackFxnArg)
{
    EDMA_channelConfig_t config;

```

```

int32_t errorCode = EDMA_NO_ERROR;
config.channelId = chId;
config.channelType = (uint8_t)EDMA3_CHANNEL_TYPE_DMA;
config.paramId = chId;
config.eventQueueId = eventQueueId;
config.paramSetConfig.sourceAddress = (uint32_t) srcBuff;
config.paramSetConfig.destinationAddress = (uint32_t) dstBuff;
config.paramSetConfig.aCount = aCount;
config.paramSetConfig.bCount = bCount;
config.paramSetConfig.cCount = 1U;
config.paramSetConfig.bCountReload = 0U;
config.paramSetConfig.sourceBIndex = srcBIdx;
config.paramSetConfig.destinationBIndex = dstBIdx;
config.paramSetConfig.sourceCIndex = 0U;
config.paramSetConfig.destinationCIndex = 0U;
config.paramSetConfig.linkAddress = EDMA_NULL_LINK_ADDRESS;
config.paramSetConfig.transferType = (uint8_t)EDMA3_SYNC_A;
config.paramSetConfig.transferCompletionCode = chId;
config.paramSetConfig.sourceAddressingMode = (uint8_t) EDMA3_ADDRESSING_MODE_LINEAR;
config.paramSetConfig.destinationAddressingMode = (uint8_t) EDMA3_ADDRESSING_MODE_LINEAR;
/* don't care because of linear addressing modes above */
config.paramSetConfig.fifoWidth = (uint8_t) EDMA3_FIFO_WIDTH_8BIT;
config.paramSetConfig.isStaticSet = false;
config.paramSetConfig.isEarlyCompletion = false;
config.paramSetConfig.isFinalTransferInterruptEnabled = true;
config.paramSetConfig.isIntermediateTransferInterruptEnabled = true;
config.paramSetConfig.isFinalChainingEnabled = false;
config.paramSetConfig.isIntermediateChainingEnabled = false;
config.transferCompletionCallbackFxn = transferCompletionCallbackFxn;
config.transferCompletionCallbackFxnArg = transferCompletionCallbackFxnArg;
if ((errorCode = EDMA_configChannel(handle, &config, isEventTriggered)) != EDMA_NO_ERROR)
{
    //System_printf("Error: EDMA_configChannel() failed with error code = %d\n", errorCode);
    goto exit;
}
exit:
    return(errorCode);
}
static int32_t DMM_initEDMA()
{
    int32_t retVal = 0;
    uint8_t chId;
    cmplx16ReIm_t *ADCdataBuf;
    uint16_t numAdcSamples, ADCBufferoffset, ChirpPerFrame, FrameNum;
    uint16_t shadowParam = EDMA_NUM_DMA_CHANNELS;
    uint32_t eventQueue;

    ADCdataBuf = (cmplx16ReIm_t *) (SOC_XWR18XX_MSS_ADCBUF_BASE_ADDRESS);
    numAdcSamples=128;
    ChirpPerFrame=2;
    FrameNum=2;
    //numTxAntennas=1;
    //numRxAntennas=1;
    ADCBufferoffset=0;

```

```

eventQueue=0;
chId=EDMA_TPCC0_REQ_FREE_0;
/*****
 * Initialize EDMA driver *****/
if (MRR_MSS_initEDMA(EDMA_INSTANCE_A) < 0)
{
    return -1;
}
/*****
 * Open EDMA driver: *****/
if( (gMrrMSSMCB.dmaHandle = MRR_MSS_openEDMA(EDMA_INSTANCE_A) )== NULL)
{
    return -2;
}
retVal =
    EDMAUtil_configType1(gMrrMSSMCB.dmaHandle,
        (uint8_t *) (SOC_translateAddress((uint32_t)&ADCdataBuf[0],
SOC_TranslateAddr_Dir_TO_EDMA, NULL)), //(&ADCdataBuf[0]),
        (uint8_t *) (SOC_translateAddress((uint32_t)&gDataCube[0], SOC_TranslateAddr_Dir_TO_EDMA,
NULL)), //(&gDataCube[0]),
        chId,
        false,
        shadowParam++,
        numAdcSamples * BYTES_PER_SAMP_1D,
        ChirpPerFrame*FrameNum,
        ADCBufferoffset * 2,
        numAdcSamples * BYTES_PER_SAMP_1D,
        eventQueue,
        NULL,
        (uintptr_t) NULL);
if (retVal<0)
{
    return -3;
}
return 0;
}

```

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
Copyright © 2021，德州仪器 (TI) 公司