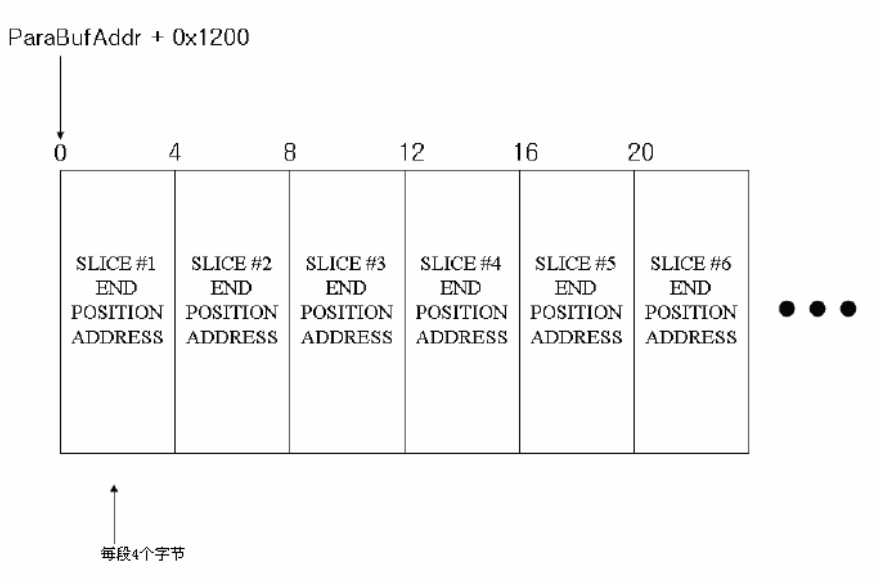

图 21-45 编码的段的信息

2.顺序/图片参数设置RBSP

这是输入变元（对于DEC_PARA_SET指令）或者输出被编码的RBSP流（对于ENC_PARA_SET指令）。

3.运转的缓冲器管理

对于编码/解码操作，该区域用于 SDRAM 内部运转的缓冲器。例如，用于 MPEG4 AC/DC 预测或者 H.264 内部预测的重组的像素行缓冲器、用于运行多处理器的内容保存缓冲器、用于 MPEG4 数据分区的位流再排序缓冲器或者 H.264 FMO/ASO 等等。例如，运转的缓冲器的大小是根据编码/解码大小和容量变化的。例如，AC/DC 预测缓冲区大小由图片宽度决定的，用于数据分区的最大位流再排序缓冲器由一张图片的最大位流大小决定。

固件的当前版本支持全 D1 大小图片（(720 x 576)）的图片解码和编码，位速率达到 10 Mb/s。在 720 x 576 情况下，如果运转的缓冲器配置选项被禁止，运转的缓冲器配置如表 21-19 所示。

表 21-19 运转的缓冲器配置

类型	名称	描述	大小（KB）
STATIC	STATIC_PRC_DMEM	用于内容交换的每个处理过程的BIT处理器数据存储器。	40^{14}
	STATIC_PRC_SEQ	每个队列的静态数据存储。	160^{15}
TEMP_PIC	MP4_DEC_ACDC	Y/Cb/Cr的AC/DC预测缓冲器。	picWidth×8

MP4_DEC	MP4_DEC_DP1	数据分区的位流再排序缓冲器。	48
	MP4_DEC_DP2	数据分区的位流再排序缓冲器。	48
TEMP_PIC 48 MP4_ENC	MP4_ENC_ACDC	Y/Cb/C的AC/DC预测缓冲器。	picWidth×8
	MP4_ENC_DP1	数据分区的位流再排序缓冲器。	48
	MP4_ENC_DP2	数据分区的位流再排序缓冲器。	48
TEMP_PIC AVC_DEC	AVC_DEC_IP	Y/Cb/Cr的内部预测缓冲器。	72
	AVC_DEC_FMO	FMO组状态缓冲器。	5.5
	AVC_DEC_SLICE_INFO	段信息缓冲器 每张图片最多1620段。	12.66
	AVC_DEC_NAL_BUF	保存NAL单元缓冲器。	0.5
	AVC_DEC_SLICE	段数据RBSP缓冲器 一张图片的所有的段数据RBPS被存储。 最坏情况下，图片的原始数据大小。	XSIZE×YSIZE×1.5/1024
TEMP_PIC VC1_DEC	VC1_DEC_ACDC	Y/Cb/Cr的AC/DC预测缓冲器	6
	VC1_DEC_DEBLK	交叠/解块过滤器运转缓冲器。	8
	VC1_DEC_DIRECTMV	直接MV 运转缓冲器。	32
TEMP_PIC AVC_ENC	AVC_ENC_IP	Y/Cb/Cr内部预测缓冲器。	72
	AVC_ENC_FMO	FMO组状态缓冲器。	256

在整个处理/编解码过程，静态缓冲器被普遍使用，并且临时图片缓冲器被每个处理过程（编解码器）重用。

当使运转缓冲器配置选项有效，为了避免不可预料的故障，必须谨慎配置它。运转缓冲器能分成三个部分：固定静态缓冲器、可配置的静态缓冲器和可配置的临时缓冲器。固定静态缓冲器用于内容交换和静态数据存储，其大小是78 KB。可配置的静态缓冲器是在寄存器描述部分定义的处理缓冲器，在处于解码的AVC和VC-1情况下使用。每种情况都必须分派可配置的静态缓冲器，在每中情况之前不做其它应用。可配置的临时缓冲器是临时性的缓冲器，在寄存器描述部分定义了它，并且必须分派说明。一张图片处理结束并且被每中状况重用后，使用可配置的临时缓冲器。

例如，如果应用执行 MPEG4 解码的全 D1 处理，两个 AVC 解码、一个 VC-1 解码、一个 MPEG4 编和一个 AVC 编码同时进行，那么建议将运转缓冲器做如表 21-20 所示的配置。

表 21-20 运转缓冲器配置

类型	名称	描述	大小（KB）
STATIC	STATIC_PRC_DMEM	用于内容交换的每个处理的BIT处理器数据存储器。	40
	STATIC_PRC_SEQ	每个队列的静态数据存储	32
	STATIC_AVC_DEC_FMO	Dec FMO组缓冲器。	5.5
	STATIC_AVC_DEC_NAL_BUF	保存NAL单元缓冲器。	0.5
STATIC(configurable)	AVC_DEC_PS	PS数据保存缓冲器。	128
	VC1_DEC_DirectMV	直接MV预测缓冲器。	6.33
TEMP_PIC MP4_DEC	MP4_DEC_ACDC	Y/Cb/Cr的AC/DC预测缓冲器。	picWidth×8
	MP4_DEC_DP1	用于数据分区的位流再排序缓冲器。	48
	MP4_DEC_DP2	用于数据分区的位流再排序缓冲器。	48
TEMP_PIC MP4_ENC	MP4_ENC_ACDC	Y/Cb/Cr的AC/DC预测缓冲器。	picWidth×8
	MP4_ENC_DP1	用于数据分区的位流再排序缓冲器。	48
	MP4_ENC_DP2	用于数据分区的位流再排序缓冲器。	48
TEMP_PIC AVC_DEC	AVC_DEC_IP_Y	Y的内部预测缓冲器。	(FrameBufferStride×picHieght/16)/1024
	AVC_DEC_IP_Cb	Cb的内部预测缓冲器。	[{FrameBufferStride/2×(picHieght/2)}/8]/1024
	AVC_DEC_IP_Cr	Cr的内部预测缓冲器。	[{FrameBufferStride/2×(picHieght/2)}/8]/1024
	AVC_DEC_SLICE_INFO	段信息缓冲器 每张图片最大1620段。	12.66
	AVC_DEC_SLICE	段数据RBSP缓冲器	XSIZE×YSIZE×1.5/1024

		存储一张图片的所有段数据。最坏情况下图片原始数据的大小。	
TEMP_PIC AVC_ENC	AVC_ENC_IP_Y	Y的内部预测缓冲器。	(FrameBufferStride×picHieght/16)/1024
	AVC_ENC_IP_Cb	Cb的内部预测缓冲器。	[{FrameBufferStride/2×(picHieght/2)}/8]/1024
	AVC_ENC_IP_Cr	Cr的内部预测缓冲器。	[{FrameBufferStride/2×(picHieght/2)}/8]/1024
	AVC_ENC_FMO	Enc FMO组缓冲器。	32 × SliceGroupNum
TEMP_PIC VC1_DEC	VC1_DEC_ACDC	Y/Cb/Cr的AC/DC预测缓冲器。	6
	VC1_DEC_DEBLK	交叠/解块过滤器运转缓冲器。	8

根据目标应用的要求，通过选择当前缓冲器的总线设置，应用软件能保留运转缓冲区空间。

运行过程的描述：

BIT处理器能同时执行最多8个进程。每个进程有不同的编解码标准（MPEG4 DECODE、H.264 ENCODE等）。每个进程有不同的位流缓冲器和相关读/写指针寄存器（BitStreamRdPtr0、BitStreamWrPtr0等）。进程通过DEC_SEQ_INIT/ENC_SEQ_INIT指令产生，通过DEC_SEQ_END/ENC_SEQ_END指令结束。每个进程通过运行索引寄存器来鉴别。位处理器通过执行指令确定每个进程的时间。对于BIT处理器内容的交换，交换不同的处理需要一些周期/带宽的系统开销。在每个内容交换大约8 KB的存储数据被读出或写入SDRAM，如果编解码在前面的和当前的进程不相同，另外88 KB的存储代码被从SDRAM读出。因此，最坏的情况下，内容交换的开销花费的SDRAM带宽大约是{8KB(R)+ 8KB(W) + 8KB(R)} × 30 Hz × 8 进程 = 5760 KB / s (30 帧/ s)。

对于以脉冲的形式读或写SDRAM数据，BIT处理器要求大约0.75周期/字节。因此最坏的情况下，关于系统开销，交换周期大约是{8K + 8K + 8K} × 0.75 × 30 Hz × 8进程～= 4.4 M 周期

4.**预/后旋转的描述**

编码源图像可能优先于编码处理（预旋转）被旋转，解码输出图像可能在解码处理后被旋转。在每次编码/解码一张图片时，主机告知预/后旋转模式。

在预旋转条件下，在读取源图像处理时，旋转被执行。因此不再需要SDRAM带宽。读取旋转源图像后，保持编码进程和非预旋转情况下相同。

在后旋转情况下，在存储解码图像处理时，旋转被执行。但是解码图像（优先于旋转）用于以后的帧解码，用作运动补偿参考。因此当后旋转有效，解码图像必须被存储，进一步需要SDRAM带宽用于额外的后旋转图像存储。。

预/后旋转模式由4位区域组成。第一位（最有意义的位）是水平镜像，下一位是垂直镜像，最后两位（不太重要的两位）是逆时针旋转角度。每个镜像/旋转域（水平、垂直、旋转）都独立应用。

在预旋转情况下，镜像域优先于旋转使用，接下来逆时针旋转应用到水平/垂直触发镜像。

在后旋转器中，旋转区域优先于水平/垂直镜像旋转，接下来镜像应用到逆时针旋转镜像。

如果逆时针旋转区域是 1（90 度）或者 3（270 度），旋转的镜像图片宽读/高度将被改变。例如，CIF 图像大小是（352 x 288），旋转后，将是 288 x 352，并且在后旋转的情况下，解码图像和旋转图像将存储为不同的格式。如图 21-46 所示，显示预/后旋转的描述图。

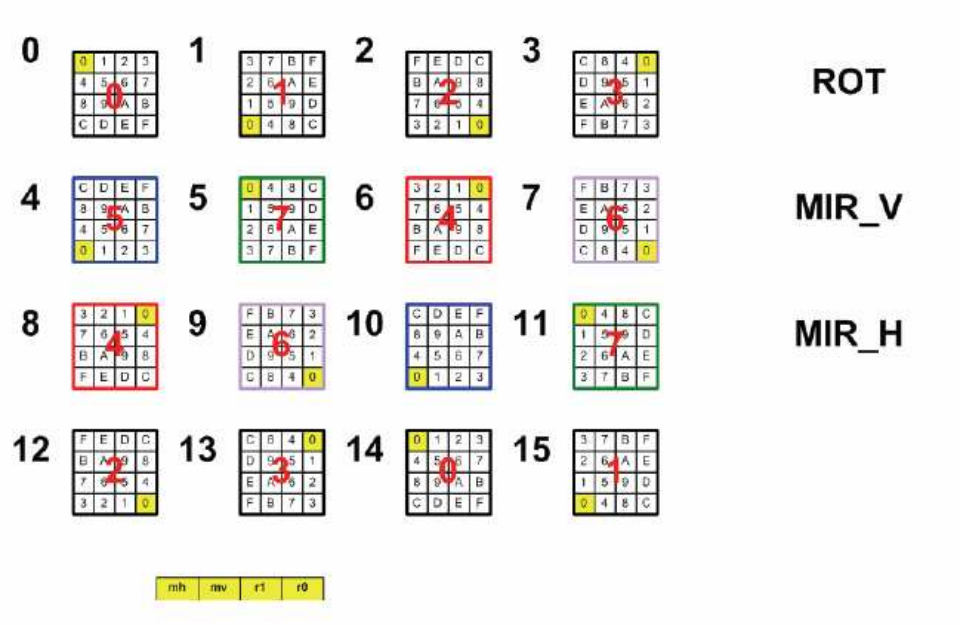

图21-46 预/后旋转的描述图

5. 操作流程的实例

对于执行 BIT 处理器的具体运行情况，如图 21-47 所示，显示流程是简单的例子。

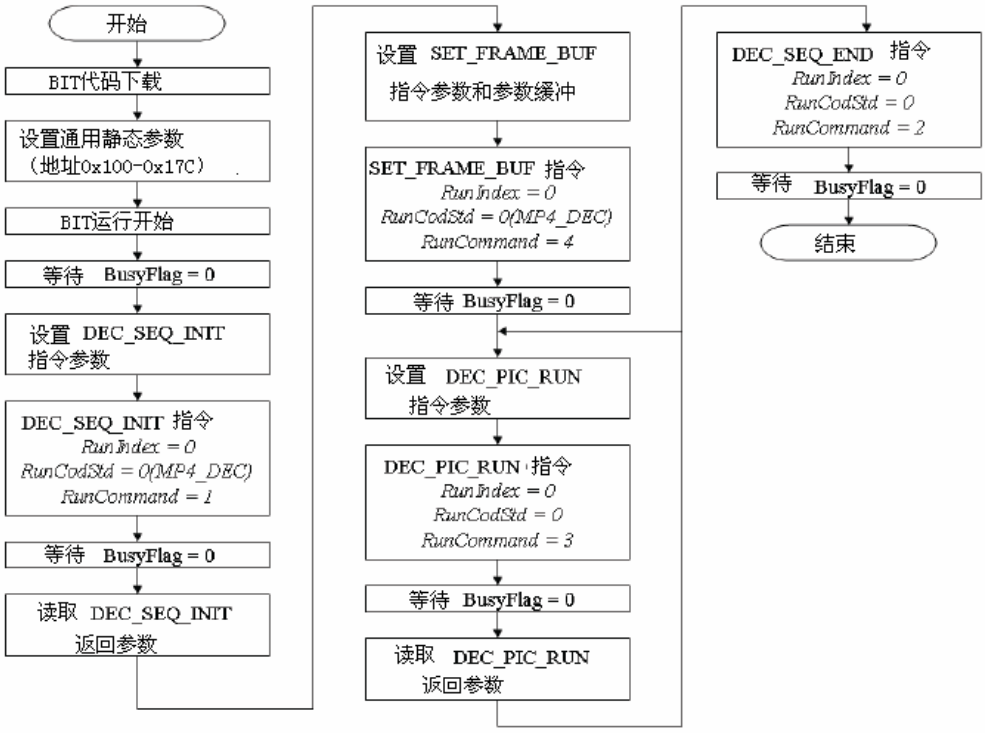

图 21-47（a）单个 MPEG4 解码器的例子

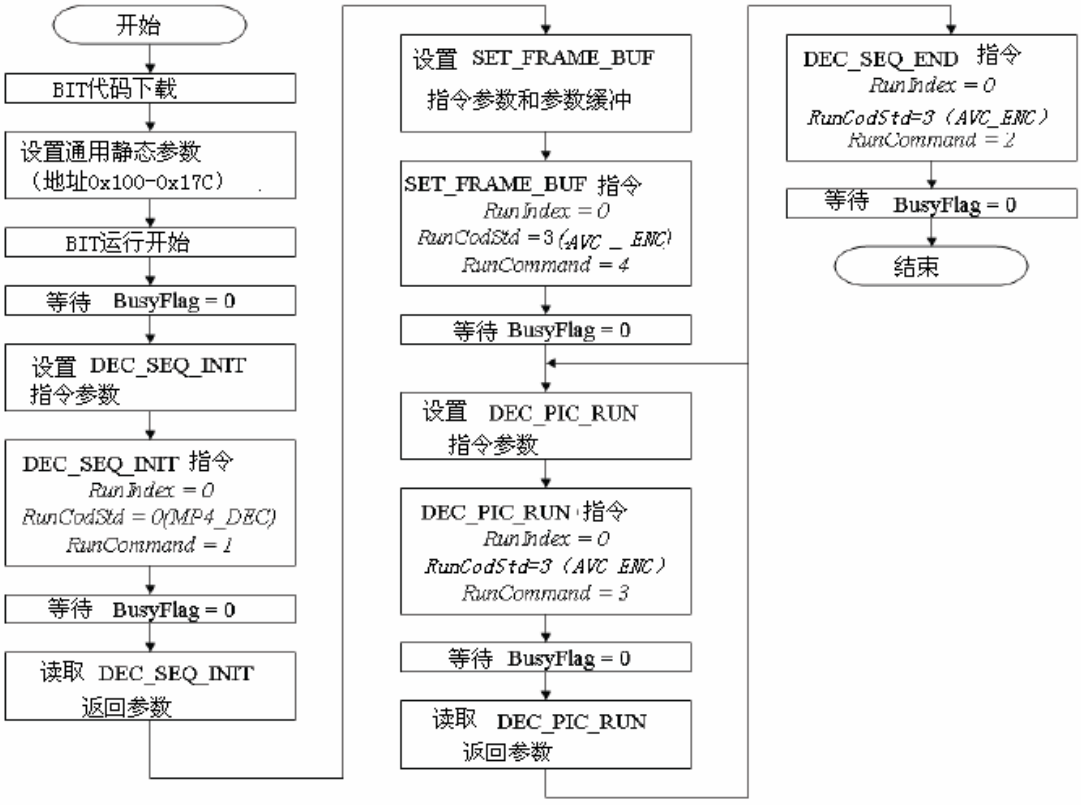

图21-47（b）单个H.264 编码器的例子

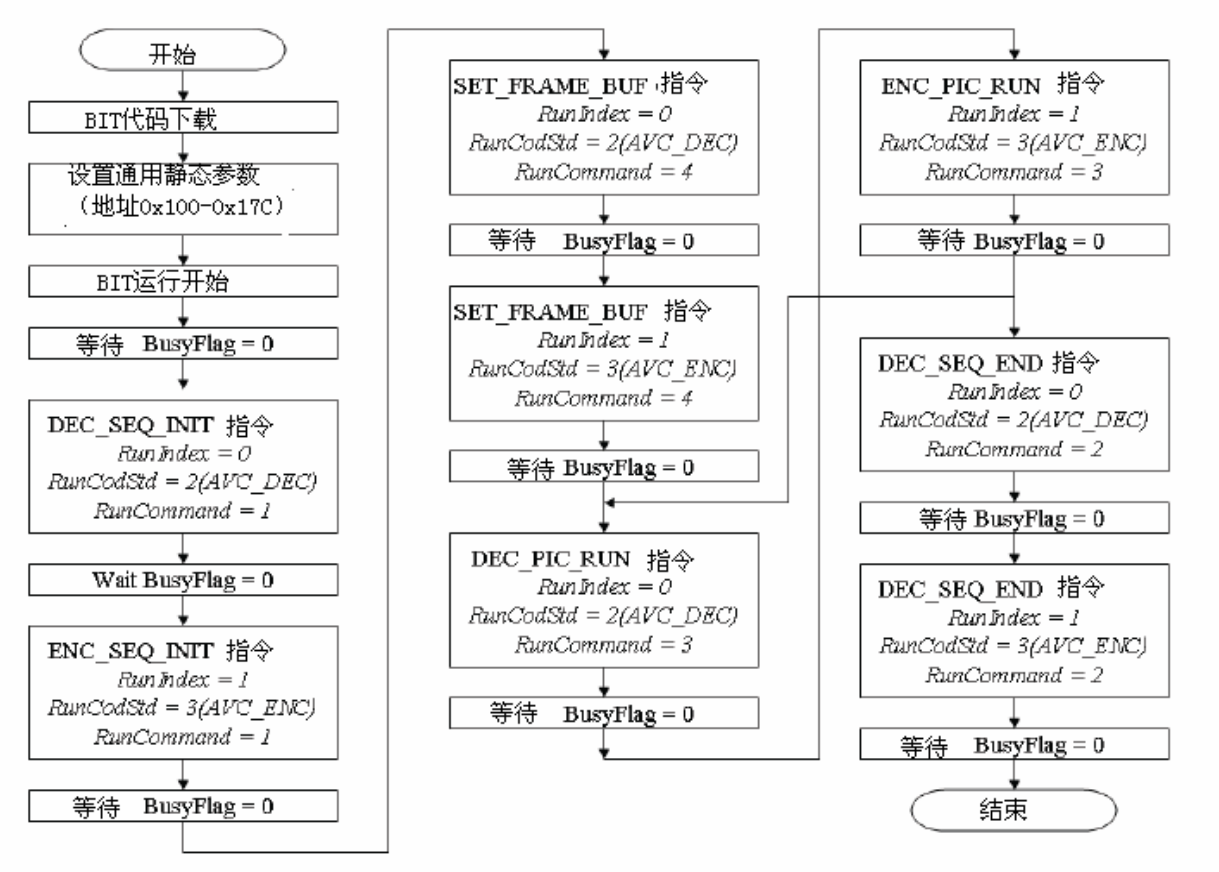

图 21-47（c）全双工的例子

单个 MPEG4 解码器的例子

以下位多格式视频编解码器的编码解码处理的初始化的一些代码，请大家参考。

多格式视频编解码器的编码处理初始化代码：

```
void MFC_InitProcessForDecoding(
    u32 uProcessIdx, MFC_CODEC_MODE eCodecMode, u32 uStreamBufStAddr, u32 uStreamBufSize,
    u32 uFrameBufStAddr, bool bDecRotEn, bool bMp4DeblkEn, bool bH264ReorderEn)
{
    u32 uStreamBufSizeCeilingToKbMultiple;
    u32 uMp4DecDeblkMode;
    u32 uH264DecReorderMode;
```

```
    bool stat;
    float frameRate;
    u32 picX, picY;
    u32 uNumOfRefReconFrame;
    u32 uStride;
    u32 uHeight;
    u32 uFrameBufNumTemp;
    u32 uFrameBufNum;

    Assert(uProcessIdx < MAX_PROCESS_NUM);
    Assert(eCodecMode == MP4_DEC || eCodecMode == AVC_DEC || eCodecMode == VC1_DEC);

    oMfc.m_eCodecMode[uProcessIdx] = eCodecMode;
    uStreamBufSizeCeilingToKbMultiple = (uStreamBufSize+1023)/1024*1024;

    oMfc.m_uStreamBufStAddr[uProcessIdx] = uStreamBufStAddr;
    oMfc.m_uStreamBufEndAddr[uProcessIdx]           =           uStreamBufStAddr           +
uStreamBufSizeCeilingToKbMultiple;
    oMfc.m_uStreamBufByteSize[uProcessIdx] = uStreamBufSizeCeilingToKbMultiple;

    oMfc.m_uBitRdPtr[uProcessIdx] =
        (uProcessIdx == 0) ? BITS_RD_PTR0 :
        (uProcessIdx == 1) ? BITS_RD_PTR1 :
        (uProcessIdx == 2) ? BITS_RD_PTR2 :
        (uProcessIdx == 3) ? BITS_RD_PTR3 :
        (uProcessIdx == 4) ? BITS_RD_PTR4 :
        (uProcessIdx == 5) ? BITS_RD_PTR5 :
        (uProcessIdx == 6) ? BITS_RD_PTR6 : BITS_RD_PTR7;
    oMfc.m_uBitWrPtr[uProcessIdx] =
```

```
        (uProcessIdx == 0) ? BITS_WR_PTR0 :
        (uProcessIdx == 1) ? BITS_WR_PTR1 :
        (uProcessIdx == 2) ? BITS_WR_PTR2 :
        (uProcessIdx == 3) ? BITS_WR_PTR3 :
        (uProcessIdx == 4) ? BITS_WR_PTR4 :
        (uProcessIdx == 5) ? BITS_WR_PTR5 :
        (uProcessIdx == 6) ? BITS_WR_PTR6 : BITS_WR_PTR7;

    mfcOutp32(oMfc.m_uBitRdPtr[uProcessIdx], oMfc.m_uStreamBufStAddr[uProcessIdx]);
    mfcOutp32(oMfc.m_uBitWrPtr[uProcessIdx],
oMfc.m_uStreamBufStAddr[uProcessIdx]+uStreamBufSize);

    mfcOutp32(DEC_SEQ_BIT_BUF_ADDR, oMfc.m_uStreamBufStAddr[uProcessIdx]);
    mfcOutp32(DEC_SEQ_BIT_BUF_SIZE,  oMfc.m_uStreamBufByteSize[uProcessIdx]/1024);    // KB
unit

    oMfc.m_bMp4DecDeblkMode[uProcessIdx] = (eCodecMode == MP4_DEC) ? bMp4DeblkEn : false;
    uMp4DecDeblkMode   =   (oMfc.m_bMp4DecDeblkMode[uProcessIdx])   ?   MP4_DBK_ENABLE   :
MP4_DBK_DISABLE;
    uH264DecReorderMode = (bH264ReorderEn) ? REORDER_ENABLE : REORDER_DISABLE;
    mfcOutp32(DEC_SEQ_OPTION, uMp4DecDeblkMode|uH264DecReorderMode);

    oMfc.m_bIsNoMoreStream[uProcessIdx] = false;

    MFC_IssueCmd(uProcessIdx, SEQ_INIT);

    stat = MFC_IsCmdFinished();

    if(stat == false)
```