Data_Uart original

Transmit analysis

transmit use `void MmwDemo_transmitProcessedOutput(UART_Handle uartHandle, MmwDemo_DataPathObj* obj)` at `"main.c"`

```
/** @brief Transmits detection data over UART
*
*    The following data is transmitted:
*    1. Header (size = 32bytes), including "Magic word", (size = 8 bytes)
*       and icluding the number of TLV items
*    TLV Items:
*    2. If detectedObjects flag is set, pbjOut structure containing range,
*       Doppler, and X,Y,Z location for detected objects,
*       size = sizeof(objOut_t) * number of detected objects
*    3. If logMagRange flag is set,  rangeProfile,
*       size = number of range bins * sizeof(uint16_t)
*    4. If noiseProfile flag is set,  noiseProfile,
*       size = number of range bins * sizeof(uint16_t)
*    5. If rangeAzimuthHeatMap flag is set, the zero Doppler column of the
*       range cubed matrix, size = number of Rx Azimuth virtual antennas *
*       number of chirps per frame * sizeof(uint32_t)
*    6. If rangeDopplerHeatMap flag is set, the log magnitude range-Doppler
matrix,
*       size = number of range bins * number of Doppler bins * sizeof(uint16_t)
*    7. If statsInfo flag is set, the stats information
*   @param[in] uartHandle   UART driver handle
*   @param[in] obj          Pointer data path object MmwDemo_DataPathObj
*/
void MmwDemo_transmitProcessedOutput(UART_Handle uartHandle, MmwDemo_DataPathObj*
obj);
```

`UART_Handle` defind at `<ti\drivers\uart\UART.h>`

```
/*!
 *  @brief  UART Global configuration
 *
 *  The UART_Config structure contains a set of pointers used to characterize
 *  the UART driver implementation.
 *
 *  This structure needs to be defined before calling UART_init() and it must
 *  not be changed thereafter.
  *
 *  @sa     UART_init()
 */
typedef struct UART_Config_t
{
    /*! Pointer to a table of driver-specific implementations of UART APIs */
    UART_FxnTable const    *fxnTablePtr;

    /*! Pointer to a driver specific data object */
    void                   *object;

    /*! Pointer to a driver specific hardware attributes structure */
    void                   *hwAttrs;
} UART_Config;
/*!
 *  @brief      A handle that is returned from a UART_open() call.
 */
typedef struct UART_Config_t     *UART_Handle;
```

`MmwDemo_DataPathObj` defind at `"data_path.h"`

```
/**
 * @brief
 *  Millimeter Wave Demo Data Path Information.
 *
 * @details
 *  The structure is used to hold all the relevant information for
 *  the data path.
 */
typedef struct MmwDemo_DataPathObj_t
{
    /*! Pointer to cliCfg */
    MmwDemo_CliCfg_t *cliCfg;

    /*! @brief Pointer to cli config common to all subframes*/
    MmwDemo_CliCommonCfg_t *cliCommonCfg;

    /*! @brief   Number of receive channels */
    uint32_t numRxAntennas;

    /*! @brief   ADCBUF handle. */
    ADCBuf_Handle adcbufHandle;

    /*! @brief   Handle of the EDMA driver. */
    EDMA_Handle edmaHandle;

    /*! @brief   EDMA error Information when there are errors like missing events */
    EDMA_errorInfo_t  EDMA_errorInfo;

    /*! @brief EDMA transfer controller error information. */
    EDMA_transferControllerErrorInfo_t EDMA_transferControllerErrorInfo;

    /*! @brief Semaphore handle for 1D EDMA completion. */
    Semaphore_Handle EDMA_1Ddone_semHandle;

    /*! @brief Semaphore handle for 2D EDMA completion. */
    Semaphore_Handle EDMA_2Ddone_semHandle;

    /*! @brief Semaphore handle for CFAR EDMA completion. */
    Semaphore_Handle EDMA_CFARdone_semHandle;

    /*! @brief Handle to hardware accelerator driver. */
    HWA_Handle  hwaHandle;

    /*! @brief Semaphore handle for Hardware accelerator completion. */
    Semaphore_Handle HWA_done_semHandle;

    /*! @brief Hardware accelerator Completion Isr count for debug purposes. */
    uint32_t hwaDoneIsrCounter;

    /*! @brief Frame counter incremented in frame start interrupt handler*/
    uint32_t frameStartIntCounter;

    /*! @brief Semaphore handle for Frame start indication. */
    Semaphore_Handle frameStart_semHandle;

    /*! @brief Number of CFAR detections by Hardware accelerator for debug
purposes. */
    uint32_t numHwaCfarDetections;

    /*! @brief Number of detected objects. */
    uint32_t numObjOut;

    /*! @brief output object array */
    MmwDemo_detectedObj objOut[MMW_MAX_OBJ_OUT];

    /*! @brief noise energy */
    uint32_t noiseEnergy;

    /*! @brief Pointer to Radar Cube memory in L3 RAM */
    uint32_t *radarCube;

    /*! @brief Pointer to range-doppler log magnitude matrix memory in L3 RAM */
    uint16_t *rangeDopplerLogMagMatrix;

    /*! @brief pointer to CFAR detection output in L3 RAM */
    cfarDetOutput_t *cfarDetectionOut;

    /*! @brief Pointer to 2D FFT array in range direction, at doppler index 0,
     * for static azimuth heat map */
    cmplx16ImRe_t *azimuthStaticHeatMap;

    /*! @brief valid Profile index */
    uint32_t validProfileIdx;

    /*! @brief number of transmit antennas */
    uint32_t numTxAntennas;

    /*! @brief number of virtual antennas */
    uint32_t numVirtualAntennas;
    /*! @brief number of virtual azimuth antennas */
    uint32_t numVirtualAntAzim;

    /*! @brief number of virtual elevation antennas */
    uint32_t numVirtualAntElev;

    /*! @brief number of ADC samples */
    uint32_t numAdcSamples;

    /*! @brief number of range bins */
    uint32_t numRangeBins;

    /*! @brief number of chirps per frame */
    uint32_t numChirpsPerFrame;

    /*! @brief number of angle bins */
    uint32_t numAngleBins;

    /*! @brief number of doppler bins */
    uint32_t numDopplerBins;

    /*! @brief number of range bins per transfer */
    uint32_t numRangeBinsPerTransfer;

    /*! @brief range resolution in meters */
    float rangeResolution;

    /*! @brief Q format of the output x/y/z coordinates */
    uint32_t xyzOutputQFormat;

    /*! @brief Timing information */
    MmwDemo_timingInfo_t timingInfo;

    /*! @brief Data path mode */
    DataPath_mode    dataPathMode;

    /*! @brief Detected objects azimuth index for debugging */
    uint8_t detObj2dAzimIdx[MMW_MAX_OBJ_OUT];

    /*! @brief Detected object elevation angle for debugging */
    float detObjElevationAngle[MMW_MAX_ELEV_OBJ_DEBUG];

    /*! @brief  Used for checking that inter frame processing finshed on time */
    int32_t interFrameProcToken;

    /*! @brief  Datapath stopped flag */
    bool datapathStopped;

    /*! @brief Pointer to DC range signature compensation buffer */
    cmplx32ImRe_t *dcRangeSigMean;

    /*! @brief DC range signature calibration counter */
    uint32_t dcRangeSigCalibCntr;
    /*! @brief DC range signature calibration forced disable counter */
    uint32_t dcRangeForcedDisableCntr;


    /*! @brief log2 of number of averaged chirps */
    uint32_t log2NumAvgChirps;

    /*! @brief data path chain selection */
    DataPath_chain2DFftSel datapathChainSel;

    /*! @brief Rx channel gain/phase offset compensation coefficients */
    MmwDemo_compRxChannelBiasCfg_t compRxChanCfg;

    /*! @brief Rx channel Chirp Quality config & data */
    MmwDemo_dataPathCQ          datapathCQ;
} MmwDemo_DataPathObj;
```

`tl` content

```
/**
 * @brief
 *  Message for reporting detected objects from data path.
 *
 * @details
 *  The structure defines the message body for detected objects from from data
path.
 */
typedef struct MmwDemo_output_message_tl_t
{
    /*! @brief   TLV type */
    uint32_t    type;

    /*! @brief   Length in bytes */
    uint32_t    length;

} MmwDemo_output_message_tl;
```

type `enum`

```
/*!
 * @brief
 *  Message types used in Millimeter Wave Demo for the communication between
 *  target and host, and also for Mailbox communication
 *  between MSS and DSS on the XWR16xx platform. Message types are used to
indicate
 *  different type detection information sent out from the target.
 *
 */
```

```
typedef enum MmwDemo_output_message_type_e
{
    /*! @brief   List of detected points */
    MMWDEMO_OUTPUT_MSG_DETECTED_POINTS = 1,

    /*! @brief   Range profile */
    MMWDEMO_OUTPUT_MSG_RANGE_PROFILE,

    /*! @brief   Noise floor profile */
    MMWDEMO_OUTPUT_MSG_NOISE_PROFILE,

    /*! @brief   Samples to calculate static azimuth  heatmap */
    MMWDEMO_OUTPUT_MSG_AZIMUT_STATIC_HEAT_MAP,

    /*! @brief   Range/Doppler detection matrix */
    MMWDEMO_OUTPUT_MSG_RANGE_DOPPLER_HEAT_MAP,

    /*! @brief   Stats information */
    MMWDEMO_OUTPUT_MSG_STATS,

    MMWDEMO_OUTPUT_MSG_MAX
} MmwDemo_output_message_type;
```

header

call `UART_writePolling`

```
    UART_writePolling (uartHandle,
                       (uint8_t*)&header,
                       sizeof(MmwDemo_output_message_header));
```

`MmwDemo_output_message_header` defind at `<ti\demo\io_interface\mmw_output.h>`

```
/*!
 * @brief
 *  Message header for reporting detection information from data path.
 *
 * @details
 *  The structure defines the message header.
 */
typedef struct MmwDemo_output_message_header_t
{
    /*! @brief   Output buffer magic word (sync word). It is initialized to {0x0102,0x0304,0x0506,0x0708} */
    uint16_t    magicWord[4];

    /*! brief   Version: : MajorNum * 2^24 + MinorNum * 2^16 + BugfixNum * 2^8 + BuildNum   */
    uint32_t     version;
    /*! @brief   Total packet length including header in Bytes */
    uint32_t    totalPacketLen;

    /*! @brief   platform type */
    uint32_t    platform;

    /*! @brief   Frame number */
    uint32_t    frameNumber;

    /*! @brief   Time in CPU cycles when the message was created. For XWR16xx: DSP
CPU cycles, for XWR14xx: R4F CPU cycles */
    uint32_t    timeCpuCycles;

    /*! @brief   Number of detected objects */
    uint32_t    numDetectedObj;

    /*! @brief   Number of TLVs */
    uint32_t    numTLVs;

#ifdef SOC_XWR16XX
    /*! @brief   For Advanced Frame config, this is the sub-frame number in the
range
     * 0 to (number of subframes - 1). For frame config (not advanced), this is
always
     * set to 0. */
    uint32_t    subFrameNumber;
#endif
} MmwDemo_output_message_header;
```

set header content

```
    /* Clear message header */
    memset((void *)&header, 0, sizeof(MmwDemo_output_message_header));
    /* Header: */
    header.platform = 0xA1443;
    header.magicWord[0] = 0x0102;
    header.magicWord[1] = 0x0304;
    header.magicWord[2] = 0x0506;
    header.magicWord[3] = 0x0708;
    header.numDetectedObj = obj->numObjOut;
    header.version =    MMWAVE_SDK_VERSION_BUILD |   //DEBUG_VERSION
                        (MMWAVE_SDK_VERSION_BUGFIX << 8) |
                        (MMWAVE_SDK_VERSION_MINOR << 16) |
                        (MMWAVE_SDK_VERSION_MAJOR << 24);
```

```
    header.numTLVs = tlvIdx;
    /* Round up packet length to multiple of MMWDEMO_OUTPUT_MSG_SEGMENT_LEN */
    header.totalPacketLen = MMWDEMO_OUTPUT_MSG_SEGMENT_LEN *
```

```
            ((packetLen + (MMWDEMO_OUTPUT_MSG_SEGMENT_LEN-
1))/MMWDEMO_OUTPUT_MSG_SEGMENT_LEN);
    header.timeCpuCycles =  Pmu_getCount(0);
    header.frameNumber = obj->frameStartIntCounter;
```

detectedObjects

calc length

```
    if (pGuiMonSel->detectedObjects && (obj->numObjOut > 0))
    {
        tl[tlvIdx].type = MMWDEMO_OUTPUT_MSG_DETECTED_POINTS;
        tl[tlvIdx].length = sizeof(MmwDemo_detectedObj) * obj->numObjOut +
                            sizeof(MmwDemo_output_message_dataObjDescr);
        packetLen += sizeof(MmwDemo_output_message_tl) + tl[tlvIdx].length;
        tlvIdx++;
    }
```

call `UART_writePolling`

```
    /* Send detected Objects */
    if ((pGuiMonSel->detectedObjects == 1) && (obj->numObjOut > 0))
    {
        MmwDemo_output_message_dataObjDescr descr;

        UART_writePolling (uartHandle,
                           (uint8_t*)&tl[tlvIdx],
                           sizeof(MmwDemo_output_message_tl));
        /* Send objects descriptor */
        descr.numDetetedObj = (uint16_t) obj->numObjOut;
        descr.xyzQFormat = (uint16_t) obj->xyzOutputQFormat;
        UART_writePolling (uartHandle, (uint8_t*)&descr,
sizeof(MmwDemo_output_message_dataObjDescr));

        /*Send array of objects */
        UART_writePolling (uartHandle, (uint8_t*)obj->objOut,
sizeof(MmwDemo_detectedObj) * obj->numObjOut);
        tlvIdx++;
    }
```

`MmwDemo_output_message_dataObjDescr` defind at `<ti\demo\io_interface\mmw_output.h>`

```
/*!
 * @brief
 * Structure holds information about detected objects.
 *
 * @details
 * This information is sent in front of the array of detected objects
 */
typedef struct MmwDemo_output_message_dataObjDescr_t
{
    /*! @brief   Number of detected objects */
    uint16_t     numDetetedObj;

    /*! @brief   Q format of detected objects x/y/z coordinates */
    uint16_t     xyzQFormat;

} MmwDemo_output_message_dataObjDescr;
```

`MmwDemo_detectedObj` defind at `<ti\demo\io_interface\mmw_output.h>`

```
/*!
 *  @brief    Detected object estimated parameters
 *
 */
typedef volatile struct MmwDemo_detectedObj_t
{
    uint16_t   rangeIdx;     /*!< @brief Range index */
    int16_t   dopplerIdx;   /*!< @brief Doppler index. Note that it is changed
                                 to signed integer in order to handle extended
maximum velocity.
                                 Neagative values correspond to the object moving
toward
                                 sensor, and positive values correspond to the
                                 object moving away from the sensor */
    uint16_t  peakVal;      /*!< @brief Peak value */
    int16_t  x;             /*!< @brief x - coordinate in meters. Q format depends
on the range resolution */
    int16_t  y;             /*!< @brief y - coordinate in meters. Q format depends
on the range resolution */
    int16_t  z;             /*!< @brief z - coordinate in meters. Q format depends
on the range resolution */
} MmwDemo_detectedObj;
```

logMagRange

calc length

```
    if (pGuiMonSel->logMagRange)
    {
        tl[tlvIdx].type = MMWDEMO_OUTPUT_MSG_RANGE_PROFILE;
        tl[tlvIdx].length = sizeof(uint16_t) * obj->numRangeBins;
        packetLen += sizeof(MmwDemo_output_message_tl) + tl[tlvIdx].length;
        tlvIdx++;
    }
```

call `UART_writePolling`

```
    /* Send Range profile */
    if (pGuiMonSel->logMagRange)
    {
        UART_writePolling (uartHandle,
                           (uint8_t*)&tl[tlvIdx],
                           sizeof(MmwDemo_output_message_tl));

        for(i = 0; i < obj->numRangeBins; i++)
        {
            UART_writePolling (uartHandle,
                    (uint8_t*)&obj->rangeDopplerLogMagMatrix[i*obj-
>numDopplerBins],
                    sizeof(uint16_t));
        }
        tlvIdx++;
    }
```

noiseProfile

calc length

```
    if (pGuiMonSel->noiseProfile)
    {
        tl[tlvIdx].type = MMWDEMO_OUTPUT_MSG_NOISE_PROFILE;
        tl[tlvIdx].length = sizeof(uint16_t) * obj->numRangeBins;
        packetLen += sizeof(MmwDemo_output_message_tl) + tl[tlvIdx].length;
        tlvIdx++;
    }
```

call `UART_writePolling`

```
    /* Send noise profile */
    if (pGuiMonSel->noiseProfile)
    {
        uint32_t maxDopIdx = obj->numDopplerBins/2 -1;
        UART_writePolling (uartHandle,
                           (uint8_t*)&tl[tlvIdx],
                           sizeof(MmwDemo_output_message_tl));

        for(i = 0; i < obj->numRangeBins; i++)
        {
            UART_writePolling (uartHandle,
                    (uint8_t*)&obj->rangeDopplerLogMagMatrix[i*obj->numDopplerBins
+ maxDopIdx],
                    sizeof(uint16_t));
        }
        tlvIdx++;
    }
```

rangeAzimuthHeatMap

calc length

```
    if (pGuiMonSel->rangeAzimuthHeatMap)
    {
        tl[tlvIdx].type = MMWDEMO_OUTPUT_MSG_AZIMUT_STATIC_HEAT_MAP;
        tl[tlvIdx].length = obj->numRangeBins * obj->numVirtualAntAzim *
sizeof(uint32_t);
        packetLen += sizeof(MmwDemo_output_message_tl) +  tl[tlvIdx].length;
        tlvIdx++;
    }
```

call `UART_writePolling`

```
    /* Send data for static azimuth heatmap */
    if (pGuiMonSel->rangeAzimuthHeatMap)
    {
        UART_writePolling (uartHandle,
                           (uint8_t*)&tl[tlvIdx],
                           sizeof(MmwDemo_output_message_tl));

        UART_writePolling (uartHandle,
                (uint8_t *) obj->azimuthStaticHeatMap,
                obj->numRangeBins * obj->numVirtualAntAzim * sizeof(uint32_t));

        tlvIdx++;
    }
```

`cmplx16ImRe_t` defind at `<ti\common\sys_common.h>`

```
/*! @brief  Complex data type, natural for C674x complex
 * multiplication instructions. */
typedef struct cmplx16ImRe_t_
{
    int16_t imag; /*!< @brief imaginary part */
    int16_t real; /*!< @brief real part */
} cmplx16ImRe_t;
```

rangeDopplerHeatMap

calc length

```
    if (pGuiMonSel->rangeDopplerHeatMap)
    {
        tl[tlvIdx].type = MMWDEMO_OUTPUT_MSG_RANGE_DOPPLER_HEAT_MAP;
        tl[tlvIdx].length = obj->numRangeBins * obj->numDopplerBins *
sizeof(uint16_t);
        packetLen += sizeof(MmwDemo_output_message_tl) + tl[tlvIdx].length;
        tlvIdx++;
    }
```

call `UART_writePolling`

```
    /* Send data for range/Doppler heatmap */
    if (pGuiMonSel->rangeDopplerHeatMap == 1)
    {
        UART_writePolling (uartHandle,
                           (uint8_t*)&tl[tlvIdx],
                           sizeof(MmwDemo_output_message_tl));
        UART_writePolling (uartHandle,
                (uint8_t*)obj->rangeDopplerLogMagMatrix,
                tl[tlvIdx].length);
        tlvIdx++;
    }
```

statsInfo

calc length

```
    if (pGuiMonSel->statsInfo)
    {
        tl[tlvIdx].type = MMWDEMO_OUTPUT_MSG_STATS;
        tl[tlvIdx].length = sizeof(MmwDemo_output_message_stats);
        packetLen += sizeof(MmwDemo_output_message_tl) + tl[tlvIdx].length;
        tlvIdx++;
    }
```

call `UART_writePolling`

```
    /* Send stats information */
    if (pGuiMonSel->statsInfo == 1)
    {
        MmwDemo_output_message_stats stats;
        stats.interChirpProcessingMargin = 0; /* Not applicable */
        stats.interFrameProcessingMargin = (uint32_t) (obj-
>timingInfo.interFrameProcessingEndMargin/R4F_CLOCK_MHZ); /* In micro seconds */
        stats.interFrameProcessingTime = (uint32_t) (obj-
>timingInfo.interFrameProcCycles/R4F_CLOCK_MHZ); /* In micro seconds */
```

```
        stats.transmitOutputTime = (uint32_t) (obj-
>timingInfo.transmitOutputCycles/R4F_CLOCK_MHZ); /* In micro seconds */
        stats.activeFrameCPULoad = obj->timingInfo.activeFrameCPULoad;
        stats.interFrameCPULoad = obj->timingInfo.interFrameCPULoad;

        UART_writePolling (uartHandle,
                           (uint8_t*)&tl[tlvIdx],
                           sizeof(MmwDemo_output_message_tl));
        UART_writePolling (uartHandle,
                           (uint8_t*)&stats,
                           tl[tlvIdx].length);
        tlvIdx++;
    }
```

`MmwDemo_output_message_stats` defind at `<ti\demo\io_interface\mmw_output.h>`

```
/*!
 * @brief
 * Structure holds message stats information from data path.
 *
 * @details
 *  The structure holds stats information. This is a payload of the TLV message
item
 *  that holds stats information.
 */
typedef struct MmwDemo_output_message_stats_t
{
    /*! @brief   Interframe processing time in usec */
    uint32_t     interFrameProcessingTime;

    /*! @brief   Transmission time of output detection informaion in usec */
    uint32_t     transmitOutputTime;

    /*! @brief   Interframe processing margin in usec */
    uint32_t     interFrameProcessingMargin;

    /*! @brief   Interchirp processing margin in usec */
    uint32_t     interChirpProcessingMargin;

    /*! @brief   CPU Load (%) during active frame duration */
    uint32_t     activeFrameCPULoad;

    /*! @brief   CPU Load (%) during inter frame duration */
    uint32_t     interFrameCPULoad;

} MmwDemo_output_message_stats;
```

padding

call `UART_writePolling`

```
    /* Send padding bytes */
    numPaddingBytes = MMWDEMO_OUTPUT_MSG_SEGMENT_LEN - (packetLen &
(MMWDEMO_OUTPUT_MSG_SEGMENT_LEN-1));
    if (numPaddingBytes<MMWDEMO_OUTPUT_MSG_SEGMENT_LEN)
    {
        UART_writePolling (uartHandle,
                            (uint8_t*)padding,
                            numPaddingBytes);
    }
```

Packet analysis

part	illustrate
header	
data	
padding	

header part

part	type	illustrate	illustrate from code
magicWord[0]	uint16	const value `0x0102`	Output buffer magic word (sync word). It is initialized to `0x0102`.
magicWord[1]	uint16	const value `0x0304`	Output buffer magic word (sync word). It is initialized to `0x0304`.
magicWord[2]	uint16	const value `0x0506`	Output buffer magic word (sync word). It is initialized to `0x0506`.
magicWord[3]	uint16	const value `0x0708`	Output buffer magic word (sync word). It is initialized to `0x0708`.
version	uint32	mmWave SDK version (hexadecimal)	Version: : `MajorNum` x 2^24 + `MinorNum` x 2^16 + `BugfixNum` x 2^8 + `BuildNum`.
totalPacketLen	uint32	packet length	Total packet length including header in Bytes.
platform	uint32	const value `0x000A1443` for "xWR1443"	platform type.
frameNumber	uint32		Frame number.
timeCpuCycles	uint32		Time in CPU cycles when the message was created. For XWR16xx: DSP CPU cycles, for XWR14xx: R4F CPU cycles.
numDetectedObj	uint32		Number of detected objects.

part	type	illustrate	illustrate from code
numTLVs	uint32	data (TLV) number	Number of TLVs.
subFrameNumber	uint32	Exists ifdef SOC_XWR16XX	This is the sub-frame number in the range 0 to (number of subframes - 1). For frame config (not advanced), this is always set to 0.

data part

part	typeid	illustrate
detectedObjects	1	
logMagRange	2	
noiseProfile	3	
rangeAzimuthHeatMap	4	
rangeDopplerHeatMap	5	
statsInfo	6	

all part is use `TLV`, that is `TLV` format

part	illustrate
type	type ID
length	part length
value	data content

value format of detectedObjects

part	type	illustrate	illustrate from code
DetetedInfomation	MmwDemo_output_message_dataObjDescr	array infomation	Structure holds information about detected objects.
DetetedObjArray	MmwDemo_detectedObj[]	array	Detected object estimated parameters.

`DetetedObjArray` have `numDetetedObj` of `MmwDemo_detectedObj`

value format of `MmwDemo_output_message_dataObjDescr`

part	type	illustrate	illustrate from code
numDetetedObj	uint16	array length	Number of detected objects.

part	type	illustrate	illustrate from code
xyzQFormat	uint16	QFormat infomation	Q format of detected objects x/y/z coordinates .

value format of `MmwDemo_detectedObj`

part	type	illustrate	illustrate from code
rangeIdx	uint16		Range index.
dopplerIdx	int16		Doppler index.
peakVal	uint16		Peak value.
x	int16	use QFormat	x - coordinate in meters. Q format depends on the range resolution.
y	int16	use QFormat	y - coordinate in meters. Q format depends on the range resolution.
z	int16	use QFormat	z - coordinate in meters. Q format depends on the range resolution.

value format of logMagRange

part	type	illustrate
logMagRange	uint16_t[]	1d array

`logMagRange` have `numRangeBins` of `uint16_t`

~~`numRangeBins` equal to (length of TLV / 2)~~

value format of noiseProfile

part	type	illustrate
noiseProfile	uint16_t[]	1d array

`noiseProfile` have `numRangeBins` of `uint16_t`

~~`numRangeBins` equal to (length of TLV / 2)~~

value format of rangeAzimuthHeatMap (azimuthStaticHeatMap)

part	type	illustrate
rangeAzimuthHeatMap	cmplx16ImRe_t[][]	2d array

`rangeAzimuthHeatMap` have (`numRangeBins` x `numVirtualAntAzim`) of `cmplx16ImRe_t`

value format of `cmplx16ImRe_t`

part	type	illustrate from code
imag	int16	imaginary part.
real	int16	real part.

value format of rangeDopplerHeatMap

part	type	illustrate
rangeDopplerHeatMap	uint16_t[][]	2d array

`rangeDopplerHeatMap` have (`numRangeBins` x `numDopplerBins`) of `uint16_t`

value format of statsInfo

part	type	illustrate
statsInfo	MmwDemo_output_message_stats	

value format of `MmwDemo_output_message_stats`

part	type	illustrate from code
interFrameProcessingTime	uint32	Interframe processing time in usec.
transmitOutputTime	uint32	Transmission time of output detection informaion in usec.
interFrameProcessingMargin	uint32	Interframe processing margin in usec.
interChirpProcessingMargin	uint32	Interchirp processing margin in usec.
activeFrameCPULoad	uint32	CPU Load (%) during active frame duration.
interFrameCPULoad	uint32	CPU Load (%) during inter frame duration.

padding part

Adjust the packet length (`header.totalPacketLen`) to be a multiple of `MMWDEMO_OUTPUT_MSG_SEGMENT_LEN`