**mss\_main.c**

**UART üzerinden veri aktarımının gerçekleştirildiği fonksiyon**

/\*\*

\* @b Description

\* @n

\* The Task is used to handle the mmw demo messages received from

\* Mailbox virtual channel.

\*

\* @param[in] arg0

\* arg0 of the Task. Not used

\* @param[in] arg1

\* arg1 of the Task. Not used

\*

\* @retval

\* Not Applicable.

\*/

**static** **void** **MmwDemo\_mboxReadTask**(UArg arg0, UArg arg1)

{

MmwDemo\_message message;

int32\_t retVal = 0;

/\* wait for new message and process all the messsages received from the peer \*/

**while**(1)

{

Semaphore\_pend(gMrrMSSMCB.mboxSemHandle, BIOS\_WAIT\_FOREVER);

/\* Read the message from the peer mailbox: We are not trying to protect the read

\* from the peer mailbox because this is only being invoked from a single thread \*/

retVal = **Mailbox\_read**(gMrrMSSMCB.peerMailbox, (uint8\_t\*)&message, **sizeof**(MmwDemo\_message));

**if** (retVal < 0)

{

/\* Error: Unable to read the message. Setup the error code and return values \*/

System\_printf ("Error: Mailbox read failed [Error code %d]\n", retVal);

}

**else** **if** (retVal == 0)

{

/\* We are done: There are no messages available from the peer execution domain. \*/

**continue**;

}

**else**

{

/\* Flush out the contents of the mailbox to indicate that we are done with the message. This will

\* allow us to receive another message in the mailbox while we process the received message. \*/

**Mailbox\_readFlush** (gMrrMSSMCB.peerMailbox);

/\* Process the received message: \*/

**switch** (message.type)

{

**case** *MMWDEMO\_DSS2MSS\_DETOBJ\_READY*:

/\* Got detected objects , to be shipped out shipped out through UART \*/

**#ifdef** USE\_LVDS\_INTERFACE\_FOR\_OBJECT\_DATA\_TX

{

int32\_t errCode;

**volatile** uint32\_t totalPacketLen = 0;

uint32\_t itemIdx;

uint8\_t \*swUserBuffer = (uint8\_t \*)&gSwUserBuffer[0];

/\* header \*/

memcpy(&swUserBuffer[totalPacketLen], (uint8\_t\*)&message.body.detObj.header, **sizeof**(MmwDemo\_output\_message\_header));

totalPacketLen = **sizeof**(MmwDemo\_output\_message\_header);

/\* TLVs \*/

**for** (itemIdx = 0; itemIdx < message.body.detObj.header.numTLVs; itemIdx++)

{

memcpy(&swUserBuffer[totalPacketLen], (uint8\_t\*)&message.body.detObj.tlv[itemIdx], **sizeof**(MmwDemo\_output\_message\_tl));

totalPacketLen += **sizeof**(MmwDemo\_output\_message\_tl);

memcpy(&swUserBuffer[totalPacketLen],

(uint8\_t\*)SOC\_translateAddress(message.body.detObj.tlv[itemIdx].address,

SOC\_TranslateAddr\_Dir\_FROM\_OTHER\_CPU,NULL), message.body.detObj.tlv[itemIdx].length);

totalPacketLen += message.body.detObj.tlv[itemIdx].length;

}

DebugP\_assert(totalPacketLen < **sizeof**(gSwUserBuffer));

/\* Deactivate the hardware session \*/

DebugP\_assert (CBUFF\_deactivateSession (gMrrMSSMCB.swSessionHandle, &errCode) == 0);

/\* Activate the hardware session \*/

DebugP\_assert (CBUFF\_activateSession (gMrrMSSMCB.swSessionHandle, &errCode) == 0);

}

**#else**

{

uint32\_t totalPacketLen;

uint32\_t numPaddingBytes;

uint32\_t itemIdx;

/\* Send header \*/

totalPacketLen = **sizeof**(MmwDemo\_output\_message\_header);

**UART\_writePolling** (gMrrMSSMCB.loggingUartHandle,

(uint8\_t\*)&message.body.detObj.header,

**sizeof**(MmwDemo\_output\_message\_header));

txMsgObjectParams.msgIdentifier = Get\_CanMessageIdentifier((MmwDemo\_output\_message\_type)MMWDEMO\_HEADER);

Can\_Transmit\_Schedule( txMsgObjectParams.msgIdentifier,

(uint8\_t\*)&message.body.detObj.header,**sizeof**(MmwDemo\_output\_message\_header));

/\* Send TLVs \*/

**for** (itemIdx = 0; itemIdx < message.body.detObj.header.numTLVs; itemIdx++)

{

**UART\_writePolling** (gMrrMSSMCB.loggingUartHandle,

(uint8\_t\*)&message.body.detObj.tlv[itemIdx],

**sizeof**(MmwDemo\_output\_message\_tl));

txMsgObjectParams.msgIdentifier = Get\_CanMessageIdentifier((MmwDemo\_output\_message\_type)message.body.detObj.tlv[itemIdx].type);

Can\_Transmit\_Schedule(txMsgObjectParams.msgIdentifier,

(uint8\_t\*)&message.body.detObj.tlv[itemIdx],**sizeof**(MmwDemo\_output\_message\_tl));

**UART\_writePolling** (gMrrMSSMCB.loggingUartHandle,

(uint8\_t\*)**SOC\_translateAddress**(message.body.detObj.tlv[itemIdx].address, *SOC\_TranslateAddr\_Dir\_FROM\_OTHER\_CPU*,NULL),

message.body.detObj.tlv[itemIdx].length);

txMsgObjectParams.msgIdentifier = Get\_CanMessageIdentifier((MmwDemo\_output\_message\_type)message.body.detObj.tlv[itemIdx].type);

Can\_Transmit\_Schedule( txMsgObjectParams.msgIdentifier,

(uint8\_t\*)**SOC\_translateAddress**(message.body.detObj.tlv[itemIdx].address,*SOC\_TranslateAddr\_Dir\_FROM\_OTHER\_CPU*,NULL),

message.body.detObj.tlv[itemIdx].length);

totalPacketLen += **sizeof**(MmwDemo\_output\_message\_tl) + message.body.detObj.tlv[itemIdx].length;

}

/\* Send padding to make total packet length multiple of MMWDEMO\_OUTPUT\_MSG\_SEGMENT\_LEN \*/

numPaddingBytes = MMWDEMO\_OUTPUT\_MSG\_SEGMENT\_LEN - (totalPacketLen & (MMWDEMO\_OUTPUT\_MSG\_SEGMENT\_LEN-1));

**if** (numPaddingBytes<MMWDEMO\_OUTPUT\_MSG\_SEGMENT\_LEN)

{

uint8\_t padding[MMWDEMO\_OUTPUT\_MSG\_SEGMENT\_LEN];

/\*DEBUG:\*/ **memset**(&padding, 0xf, MMWDEMO\_OUTPUT\_MSG\_SEGMENT\_LEN);

txMsgObjectParams.msgIdentifier = Get\_CanMessageIdentifier((MmwDemo\_output\_message\_type)MMWDEMO\_PADDING);

**UART\_writePolling** (gMrrMSSMCB.loggingUartHandle, padding, numPaddingBytes);

Can\_Transmit\_Schedule( txMsgObjectParams.msgIdentifier,

padding,numPaddingBytes);

}

}

**#endif**

/\* Send a message to MSS to log the output data \*/

**memset**((**void** \*)&message, 0, **sizeof**(MmwDemo\_message));

message.type = *MMWDEMO\_MSS2DSS\_DETOBJ\_SHIPPED*;

retVal = MmwDemo\_mboxWrite(&message);

**if** (retVal != 0)

{

System\_printf ("Error: Mailbox send message id=%d failed \n", message.type);

}

**break**;

**case** *MMWDEMO\_DSS2MSS\_ASSERT\_INFO*:

/\* Send the received DSS assert info through CLI \*/

**CLI\_write** ("DSS Exception: %s, line %d.\n", message.body.assertInfo.file,

message.body.assertInfo.line);

**break**;

**default**:

{

/\* Message not support \*/

System\_printf ("Error: unsupport Mailbox message id=%d\n", message.type);

**break**;

}

}

}

}

}

**Verilerin tam olarak nasıl gönderildiğini görmek için UART\_writePolling fonksiyonunun tanımı aşağıdadır.**

/\*!

\* @brief Function that writes data to a UART, polling the peripheral to

\* wait until new data can be written. Usage of this API is mutually

\* exclusive with usage of UART\_write().

\*

\* This function initiates an operation to write data to a UART controller.

\*

\* UART\_writePolling() will not return until all the data was written to the

\* UART (or to its FIFO if applicable).

\*

\* @sa UART\_write()

\*

\* @param handle A UART\_Handle

\*

\* @param buffer A pointer to the buffer containing the data to

\* be written to the UART.

\*

\* @param size The number of bytes in the buffer that should be written

\* to the UART.

\*

\* @return Returns the number of bytes that have been written to the UART.

\* If an error occurs, one of the UART Error codes is returned.

\*

\* \ingroup UART\_DRIVER\_EXTERNAL\_FUNCTION

\*/

**extern** int32\_t **UART\_writePolling**(UART\_Handle handle, uint8\_t \*buffer,

uint32\_t size);

**dss\_main.c**

**Çıktı, DSS üzerinde çalışan MRR\_DSS\_SendProcessOutputToMSS işlevi tarafından oluşturulur. UART üzerinden iletilmek üzere MSS’ye gönderilir.**

**Tanım:**

/\* Output logging. \*/

**static** int32\_t **MRR\_DSS\_SendProcessOutputToMSS**(uint8\_t \*ptrHsmBuffer,

uint32\_t outputBufSize,

MmwDemo\_DSS\_DataPathObj \*obj);

**Fonksiyon:**

int32\_t **MRR\_DSS\_SendProcessOutputToMSS**(uint8\_t \*ptrHsmBuffer,

uint32\_t outputBufSize,

MmwDemo\_DSS\_DataPathObj \*obj)

{

uint8\_t \*ptrCurrBuffer;

uint32\_t totalHsmSize = 0;

uint32\_t totalPacketLen = **sizeof**(MmwDemo\_output\_message\_header);

uint32\_t itemPayloadLen;

int32\_t retVal = 0;

MmwDemo\_message message;

MmwDemo\_output\_message\_dataObjDescr descr;

uint32\_t tlvIdx = 0;

/\* Set pointer to HSM buffer \*/

ptrCurrBuffer = ptrHsmBuffer;

/\* Clear message to MSS \*/

**memset**((**void** \*) &message, 0, **sizeof**(MmwDemo\_message));

message.type = *MMWDEMO\_DSS2MSS\_DETOBJ\_READY*;

/\* Header: \*/

message.body.detObj.header.platform = 0xA1642 ;

message.body.detObj.header.magicWord[0] = 0x0102;

message.body.detObj.header.magicWord[1] = 0x0304;

message.body.detObj.header.magicWord[2] = 0x0506;

message.body.detObj.header.magicWord[3] = 0x0708;

message.body.detObj.header.numDetectedObj = obj->numDetObj;

message.body.detObj.header.version = MMWAVE\_SDK\_VERSION\_BUILD | (MMWAVE\_SDK\_VERSION\_BUGFIX << 8) | (MMWAVE\_SDK\_VERSION\_MINOR << 16) | (MMWAVE\_SDK\_VERSION\_MAJOR << 24);

/\* Put detected Objects in HSM buffer: sizeof(MmwDemo\_objOut\_t) \* numDetObj \*/

**if** (obj->numDetObj > 0)

{

/\* Add objects descriptor \*/

descr.numDetetedObj = obj->numDetObj;

descr.xyzQFormat = obj->xyzOutputQFormat;

itemPayloadLen = **sizeof**(MmwDemo\_output\_message\_dataObjDescr);

totalHsmSize += itemPayloadLen;

**if** (totalHsmSize > outputBufSize)

{

retVal = -1;

**goto** Exit;

}

**memcpy**(ptrCurrBuffer, (**void** \*) &descr, itemPayloadLen);

/\* Add array of objects \*/

itemPayloadLen = **sizeof**(MmwDemo\_detectedObjForTx) \* obj->numDetObj;

totalHsmSize += itemPayloadLen;

**if** (totalHsmSize > outputBufSize)

{

retVal = -1;

**goto** Exit;

}

**memcpy**(&ptrCurrBuffer[**sizeof**(MmwDemo\_output\_message\_dataObjDescr)],

(**void** \*) obj->detObjFinal, itemPayloadLen);

message.body.detObj.tlv[tlvIdx].length = itemPayloadLen

+ **sizeof**(MmwDemo\_output\_message\_dataObjDescr);

message.body.detObj.tlv[tlvIdx].type =

*MMWDEMO\_OUTPUT\_MSG\_DETECTED\_POINTS*;

message.body.detObj.tlv[tlvIdx].address = (uint32\_t) ptrCurrBuffer;

tlvIdx++;

/\* Incrementing pointer to HSM buffer \*/

ptrCurrBuffer += itemPayloadLen

+ **sizeof**(MmwDemo\_output\_message\_dataObjDescr);

totalPacketLen += **sizeof**(MmwDemo\_output\_message\_tl) + itemPayloadLen

+ **sizeof**(MmwDemo\_output\_message\_dataObjDescr);

}

**if** ((obj->processingPath == POINT\_CLOUD\_PROCESSING) && (obj->dbScanReport.numCluster > 0))

{

/\* Add objects descriptor \*/

/\* In the point cloud processing path, the dbScanReport holds the number of clusters. \*/

descr.numDetetedObj = obj->dbScanReport.numCluster;

descr.xyzQFormat = obj->xyzOutputQFormat;

itemPayloadLen = **sizeof**(MmwDemo\_output\_message\_dataObjDescr);

totalHsmSize += itemPayloadLen;

**if** (totalHsmSize > outputBufSize)

{

retVal = -1;

**goto** Exit;

}

**memcpy**(ptrCurrBuffer, (**void** \*) &descr, itemPayloadLen);

/\* Add array of cluster reports. \*/

itemPayloadLen = **sizeof**(clusteringDBscanReportForTx) \* obj->dbScanReport.numCluster;

totalHsmSize += itemPayloadLen;

**if** (totalHsmSize > outputBufSize)

{

retVal = -1;

**goto** Exit;

}

**memcpy**(&ptrCurrBuffer[**sizeof**(MmwDemo\_output\_message\_dataObjDescr)],

(**void** \*) obj->clusterOpFinal, itemPayloadLen);

message.body.detObj.tlv[tlvIdx].length = itemPayloadLen

+ **sizeof**(MmwDemo\_output\_message\_dataObjDescr);

message.body.detObj.tlv[tlvIdx].type =

MMWDEMO\_OUTPUT\_MSG\_CLUSTERS;

message.body.detObj.tlv[tlvIdx].address = (uint32\_t) ptrCurrBuffer;

tlvIdx++;

/\* Incrementing pointer to HSM buffer \*/

ptrCurrBuffer += itemPayloadLen

+ **sizeof**(MmwDemo\_output\_message\_dataObjDescr);

totalPacketLen += **sizeof**(MmwDemo\_output\_message\_tl) + itemPayloadLen

+ **sizeof**(MmwDemo\_output\_message\_dataObjDescr);

}

**if** (obj->processingPath == POINT\_CLOUD\_PROCESSING)

{

/\* Add objects descriptor \*/

descr.numDetetedObj = obj->parkingAssistNumBins;

descr.xyzQFormat = obj->xyzOutputQFormat;

itemPayloadLen = **sizeof**(MmwDemo\_output\_message\_dataObjDescr);

totalHsmSize += itemPayloadLen;

**if** (totalHsmSize > outputBufSize)

{

retVal = -1;

**goto** Exit;

}

**memcpy**(ptrCurrBuffer, (**void** \*) &descr, itemPayloadLen);

/\* Add array of c;uster reports. \*/

itemPayloadLen = **sizeof**(uint16\_t) \* obj->parkingAssistNumBins;

totalHsmSize += itemPayloadLen;

**if** (totalHsmSize > outputBufSize)

{

retVal = -1;

**goto** Exit;

}

**memcpy**(&ptrCurrBuffer[**sizeof**(MmwDemo\_output\_message\_dataObjDescr)],

(**void** \*) obj->parkingAssistBins, itemPayloadLen);

message.body.detObj.tlv[tlvIdx].length = itemPayloadLen

+ **sizeof**(MmwDemo\_output\_message\_dataObjDescr);

message.body.detObj.tlv[tlvIdx].type =

MMWDEMO\_OUTPUT\_MSG\_PARKING\_ASSIST;

message.body.detObj.tlv[tlvIdx].address = (uint32\_t) ptrCurrBuffer;

tlvIdx++;

/\* Incrementing pointer to HSM buffer \*/

ptrCurrBuffer += itemPayloadLen

+ **sizeof**(MmwDemo\_output\_message\_dataObjDescr);

totalPacketLen += **sizeof**(MmwDemo\_output\_message\_tl) + itemPayloadLen

+ **sizeof**(MmwDemo\_output\_message\_dataObjDescr);

}

**if** ((obj->processingPath == MAX\_VEL\_ENH\_PROCESSING) &&

(obj->numActiveTrackers > 0))

{

/\* Add objects descriptor \*/

descr.numDetetedObj = obj->numActiveTrackers;

descr.xyzQFormat = obj->xyzOutputQFormat;

itemPayloadLen = **sizeof**(MmwDemo\_output\_message\_dataObjDescr);

totalHsmSize += itemPayloadLen;

**if** (totalHsmSize > outputBufSize)

{

retVal = -1;

**goto** Exit;

}

**memcpy**(ptrCurrBuffer, (**void** \*) &descr, itemPayloadLen);

/\* Add array of tracked objects. \*/

itemPayloadLen = **sizeof**(trackingReportForTx) \* obj->numActiveTrackers;

totalHsmSize += itemPayloadLen;

**if** (totalHsmSize > outputBufSize)

{

retVal = -1;

**goto** Exit;

}

**memcpy**(&ptrCurrBuffer[**sizeof**(MmwDemo\_output\_message\_dataObjDescr)],

(**void** \*) obj->trackerOpFinal, itemPayloadLen);

message.body.detObj.tlv[tlvIdx].length = itemPayloadLen

+ **sizeof**(MmwDemo\_output\_message\_dataObjDescr);

message.body.detObj.tlv[tlvIdx].type =

MMWDEMO\_OUTPUT\_MSG\_TRACKED\_OBJECTS;

message.body.detObj.tlv[tlvIdx].address = (uint32\_t) ptrCurrBuffer;

tlvIdx++;

/\* Incrementing pointer to HSM buffer \*/

ptrCurrBuffer += itemPayloadLen + **sizeof**(MmwDemo\_output\_message\_dataObjDescr);

totalPacketLen += **sizeof**(MmwDemo\_output\_message\_tl) + itemPayloadLen + **sizeof**(MmwDemo\_output\_message\_dataObjDescr);

}

**if** (tlvIdx >= *MMWDEMO\_OUTPUT\_MSG\_MAX*)

{

retVal = -1;

}

**if** (retVal == 0)

{

message.body.detObj.header.numTLVs = tlvIdx;

/\* Round up packet length to multiple of MMWDEMO\_OUTPUT\_MSG\_SEGMENT\_LEN. \*/

message.body.detObj.header.totalPacketLen =

MMWDEMO\_OUTPUT\_MSG\_SEGMENT\_LEN

\* ((totalPacketLen

+ (MMWDEMO\_OUTPUT\_MSG\_SEGMENT\_LEN - 1))

/ MMWDEMO\_OUTPUT\_MSG\_SEGMENT\_LEN);

message.body.detObj.header.timeCpuCycles = Cycleprofiler\_getTimeStamp();

message.body.detObj.header.frameNumber = gMrrDSSMCB.stats.frameStartIntCounter;

/\* The GUI reads the subframe number to decide on the plotting type.

\* a 0 => MAX\_VEL\_ENH\_PROCESSING.

\* a 1 => POINT\_CLOUD\_PROCESSING. \*/

message.body.detObj.header.subFrameNumber = gMrrDSSMCB.dataPathObj[gMrrDSSMCB.subframeIndx].processingPath;

**if** (MmwDemo\_mboxWrite(&message) != 0)

{

retVal = -1;

}

}

Exit: **return** retVal;

}

**src/common/mrr\_output.h**

**Çerçeve başına bir mesaj vardır. Mesaj her zaman sabit boyutlu bir mesaj başlığı ile başlar.**

**Başlık, ileti başlığının dahil olduğu mesajın boyutu olan totalPacketLen’i belirtir.**

**Mesaj paketi TLV’lerden oluşur. Gövdedeki numTLV’ler mesaj başlığında belirtilir.**

/\*!

\* @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/XWR18xx: 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;

**#if** (defined(SOC\_XWR16XX)||defined(SOC\_XWR18XX) || defined(ENABLE\_ADVANCED\_FRAME))

/\* SOC\_XWR18XX has 2 demo modes. In mmw demo mode which is similar to xwr16xx

\* ENABLE\_ADVANCED\_FRAME is enabled while in mmwhwa mode which is similar to xwr14xx

\* it is disabled. Due to these 2 demo modes the SOC\_XWR18XX is not used directly

\* in the above #if.\*/

/\*! @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;

**src/common/mrr\_output.h**

**Her TLV, TLV türünü ve TLV uzunluğunu belirten sabit boyutlu bir TLV başlığı ile başlar. Bu uzunluk yalnızca TLV gövdesine aittir.**

/\*\*

\* @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;

**src/common/mrr\_output.h**

**MRR için TLV gövdesi, takip edilecek dizideki öğelerin sayısını ve verilerin Q biçimini içeren MmwDemo\_output\_message\_dataObjDescr ile başlar.**

/\*!

\* @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;

**src/dss/dss\_datapath.h**

**TLV gövdesinin geri kalanı uygun yapının bir dizisini içerir. Bu dizinin uzunluğu, TLV gövdesinin en başında gönderilen veri nesnesi tanımlayıcısındaki numDetectedObj alanı tarafından verilir. Bu mesaj numDetectedObj’den farklı olabileceğini unutmayın.**

/\*!

\* @brief Detected object estimated parameters to be transmitted out.

\*

\*/

**typedef** **struct** MmwDemo\_detectedObjForTx\_t

{

int16\_t speed; /\*!< @brief Doppler index \*/

uint16\_t peakVal; /\*!< @brief Peak value \*/

int16\_t x; /\*!< @brief x - coordinate in meters. Q format provides the bitwidth. \*/

int16\_t y; /\*!< @brief y - coordinate in meters. Q format provides the bitwidth. \*/

int16\_t z; /\*!< @brief z - coordinate in meters. Q format provides the bitwidth. \*/

**#ifdef** SEND\_SNR\_INFO

uint16\_t rangeSNRdB; /\*!< @brief Range SNR (dB) \*/

uint16\_t dopplerSNRdB; /\*!< @brief Doppler SNR (dB) \*/

**#endif**

} MmwDemo\_detectedObjForTx;

/\*!

\* @brief Structure for tracking report.

\*

\*/

**typedef** **struct** trackingReportForTx\_t

{

int16\_t x; /\*\*< the tracking output -> x co-ordinate \*/

int16\_t y; /\*\*< the tracking output -> y co-ordinate \*/

int16\_t xd; /\*\*< velocity in the x direction \*/

int16\_t yd; /\*\*< velocity in the y direction \*/

int16\_t xSize; /\*\*< cluster size (x direction). \*/

int16\_t ySize; /\*\*< cluster size (y direction). \*/

} trackingReportForTx;

/\*!

\* @brief Structure for each cluster information report .

\*

\*/

**typedef** **struct** clusteringDBscanReportForTx\_t

{

int16\_t xCenter; /\*\*< the clustering center on x direction \*/

int16\_t yCenter; /\*\*< the clustering center on y direction \*/

int16\_t xSize; /\*\*< the clustering size on x direction \*/

int16\_t ySize; /\*\*< the clustering size on y direction \*/

} clusteringDBscanReportForTx;