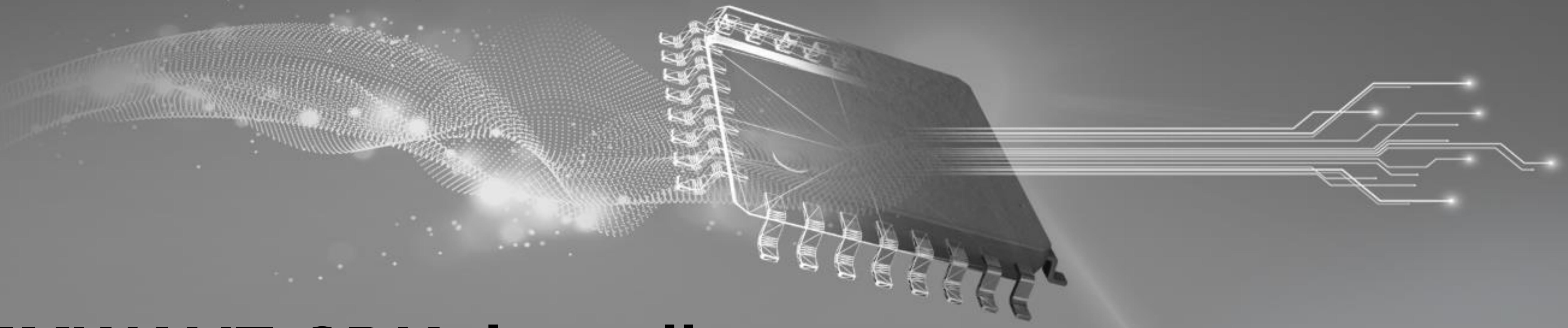
TI TECH DAYS



MMWAVE-SDK deep dive
Easy evaluation and development of mmWave systems with software development kit

Nitin Sakhuja - Industrial mmWave Radar Applications

MMWAVE-SDK deep dive

TI's MMWAVE-SDK (MilliMeter Wave Software Development Kit) is a unified software platform for the TI mmWave Sensing Portfolio, providing easy setup and fast out-of-the-box access to evaluation and development.

This training provides an overview of the MMWAVE-SDK 3.x architecture and the various building blocks such as Data Processing Units (DPUs) and Data Processing Chains(DPCs). It also provides a deeper look into the components with software execution flows accompanied with source code references from the MMWAVE-SDK Out of Box Demo point cloud processing chain.

We also present some example applications where the Out of box point cloud detection chain is extended to develop more complex mmWave applications such as Long Range People Detection and Tracking Demo, Traffic Monitoring Demo and Area Scanner Demo.

What you'll learn:

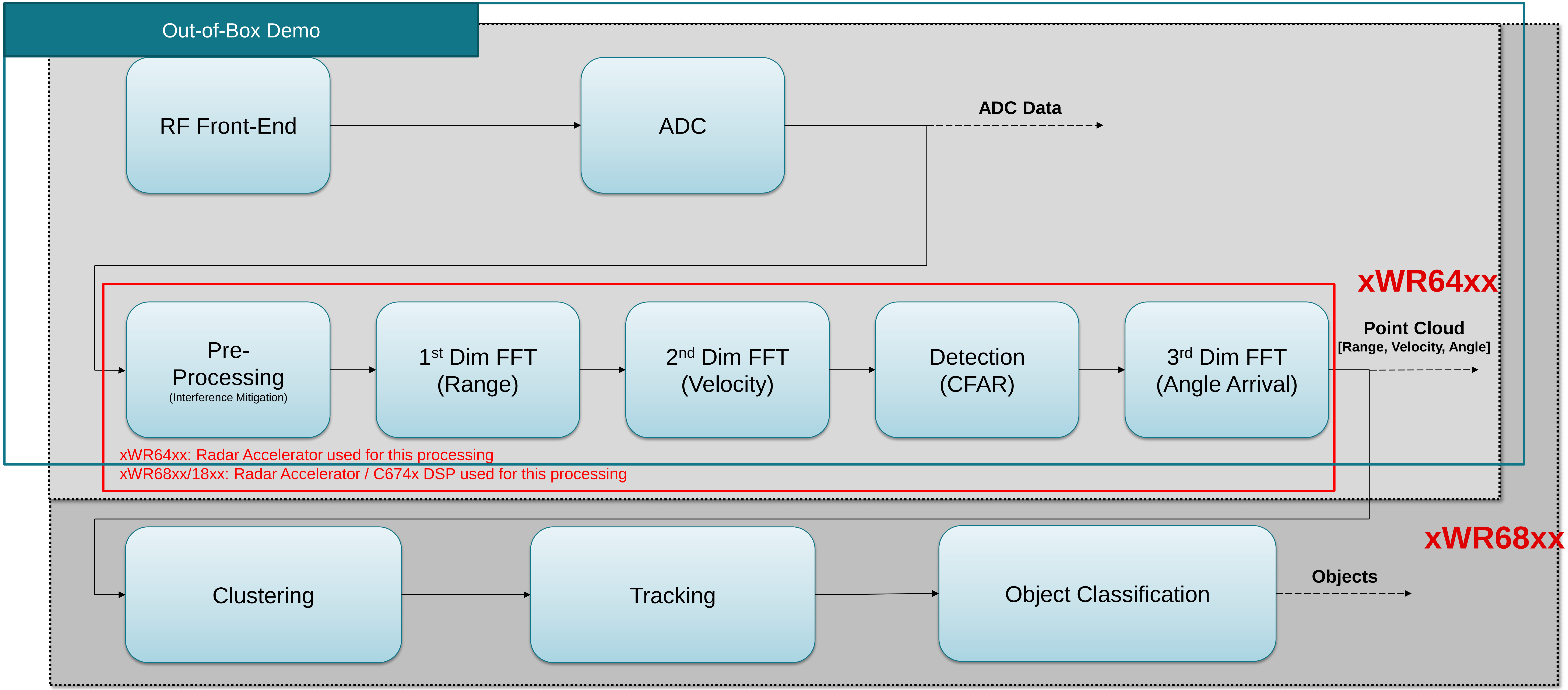
- TI MMWAVE-SDK architecture and it's various building blocks such as DPCs and DPUs
- Understand DPM, DPC and DPU execution flows e.g. initialization and runtime operation using source code references
- Developing custom components to extend the out of box processing chain and available examples.

Agenda

- MMWAVE-SDK
  - Architecture overview
  - Data path design
- Data path deep dive
  - Initialization
  - Configuration
  - Execution (Runtime view)
- DPUs and DPCs in MMWAVE-SDK 3.x
  - DPUs: Range, Static-Clutter removal, Doppler, CFAR-CA and AoA
  - DPCs: HWA and DSP based object detection chains
- Software development and debugging
  - Development resources
  - MMWAVE-SDK debugging
- Extending SDK architecture for advanced applications
  - Considerations for developing custom DPUs and DPCs
  - Custom DPUs and DPCs in Industrial Toolbox
  - Demo: Long Range People Tracking and, Traffic Monitoring
  - Demo: Area Scanner and, Automated Doors and Gates

MMWAVE-SDK - Architecture Overview

mmWave signal processing



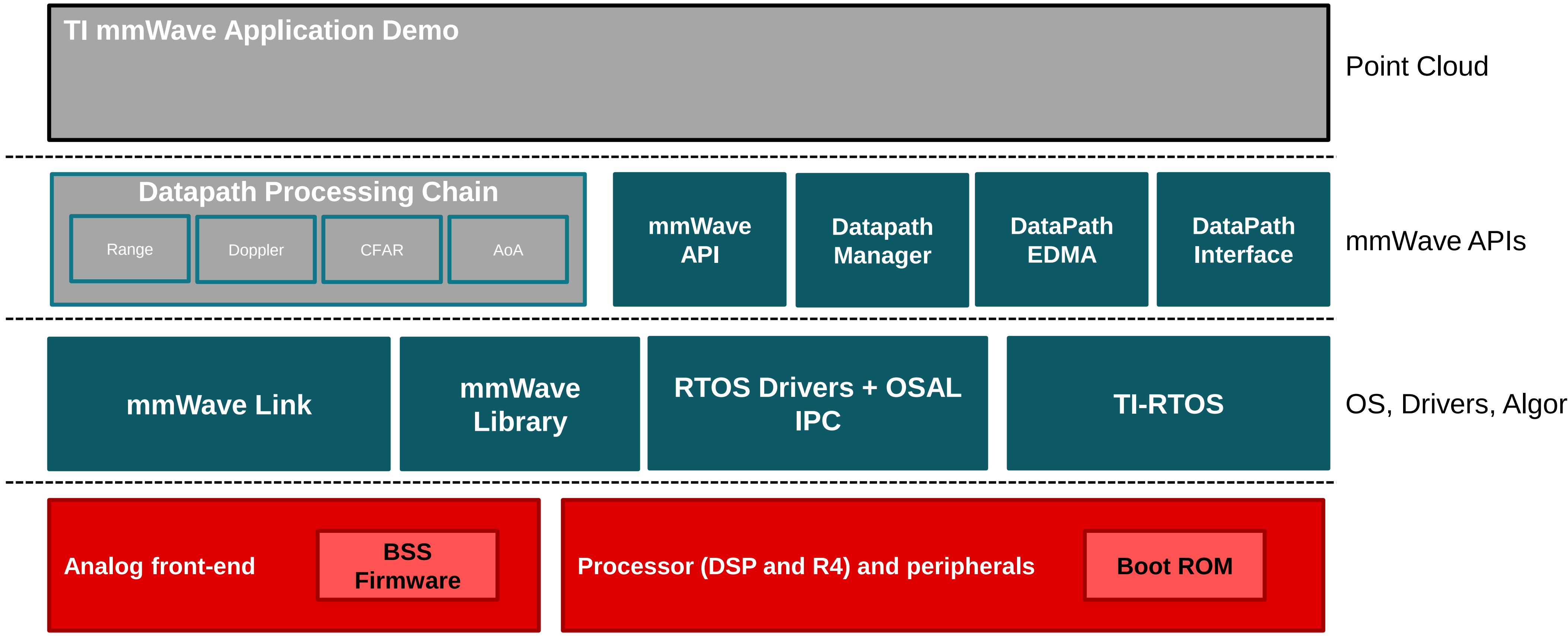
mmWave SDK contents

- Building blocks
  - RTOS, Drivers, and RadarSS firmware
  - Scalable data processing blocks and chains to work on HWA or DSP
  - Layered / API based Radar analog front end (AFE) programming
  - Pre-built software blocks and chains for basic FMCW Radar signal processing
  - Catalog of mmWave signal processing algorithms optimized for C674x DSPs including tracker
  - Package for high-security (HS) devices to enable programming encryption keys and encrypt/authenticate program binaries
- Demonstrations and examples
  - TI RTOS based
  - Out of box demo with easy configurability via TI Cloud-based or offline GUI
  - Representation of point cloud and benchmarking data from demo via GUI
- Documentation
  - Associated tools: Code Composer Studio, TI-RTOS, Uniflash
  - Available at http://www.ti.com/tool/MMWAVE-SDK

mmWave SDK architecture

Main highlights:

- Foundational components for SOC enablement – RTOS, Drivers, mmWaveLink, mmWaveLib
- RF Front-end completely abstracted using mmWaveLink
- mmWave API simplifies device integration of mmWaveLink
- Data path layer is an abstraction over existing driver APIs in the data flow
- Separation of data processing units and chain from the application
- Simpler application that does the instantiation of the Datapath layer



TI mmWave Application Demo

Point Cloud

Datapath Processing Chain

Range

Doppler

CFAR

AoA

mmWave API

Datapath Manager

DataPath EDMA

DataPath Interface

mmWave APIs

mmWave Link

mmWave Library

RTOS Drivers + OSAL IPC

TI-RTOS

OS, Drivers, Algor

Analog front-end

BSS Firmware

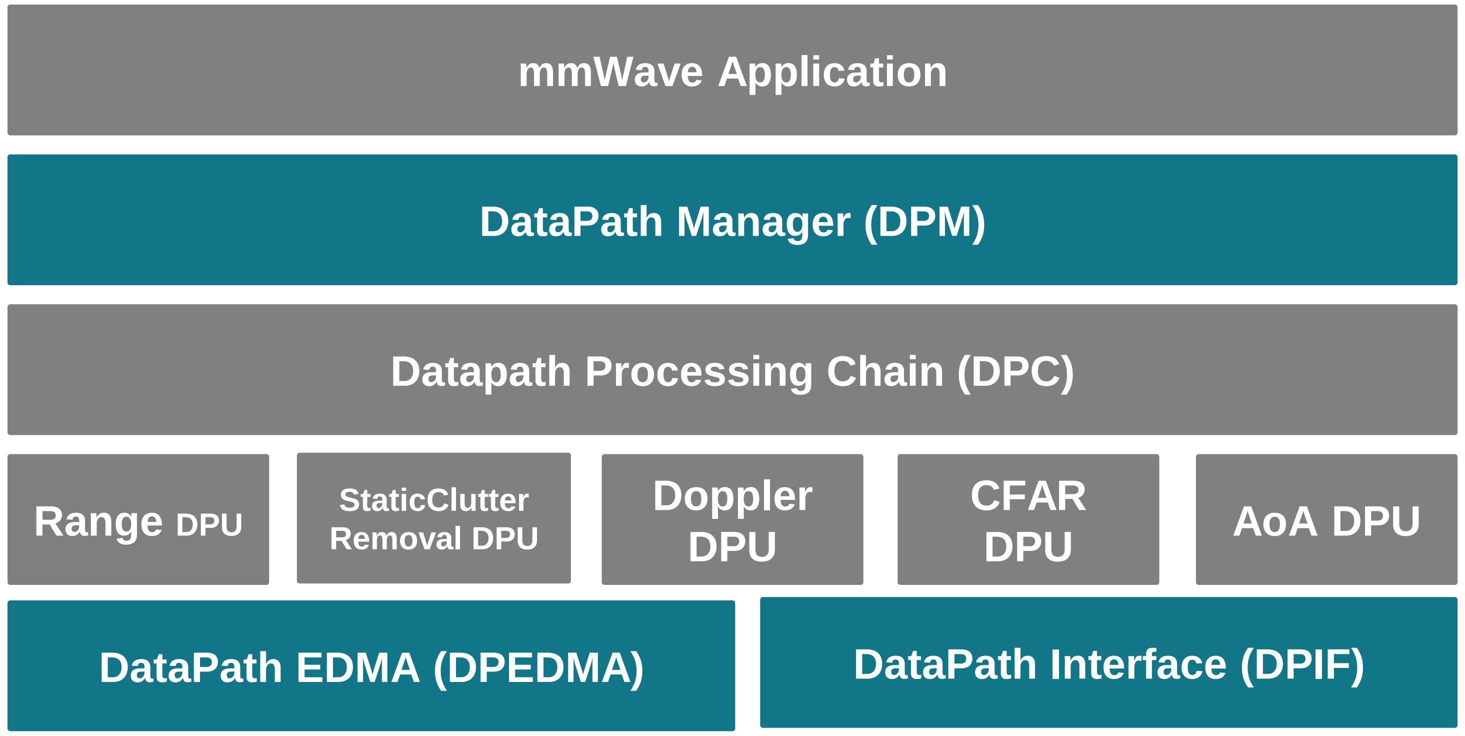
Processor (DSP and R4) and peripherals

Boot ROM

MMWAVE-SDK – Datapath Design

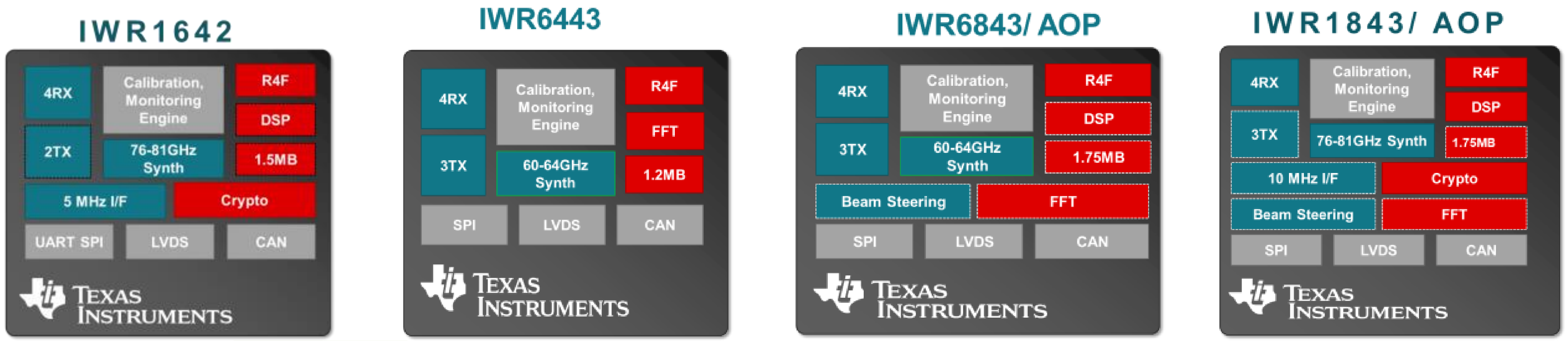
Datapath Layer Design

- **DPM: Datapath manager**
  - Foundation layer that enables the "scalability" aspect of the architecture.
- **DPIF: Standard Interface points in the Detection chain are defined**
  - Input ADC data, Radar Cube, Detection Matrix, Point cloud
- **DPUs: Data Translating function(s) from one interface point to the other are called “Data Processing Units”**
  - Range Processing (ADC data to Radar Cube)
  - Doppler Processing (Radar Cube to Detection Matrix)
  - CFAR and AoA (Detection Matrix to Point Cloud)
- **DPC: Data Processing Chain**
  - Chain of “data processing units” is called a data processing Chain. Ex: Detection DPC (ADC to Point Cloud).
  - This conforms to the DPM dictated API definition

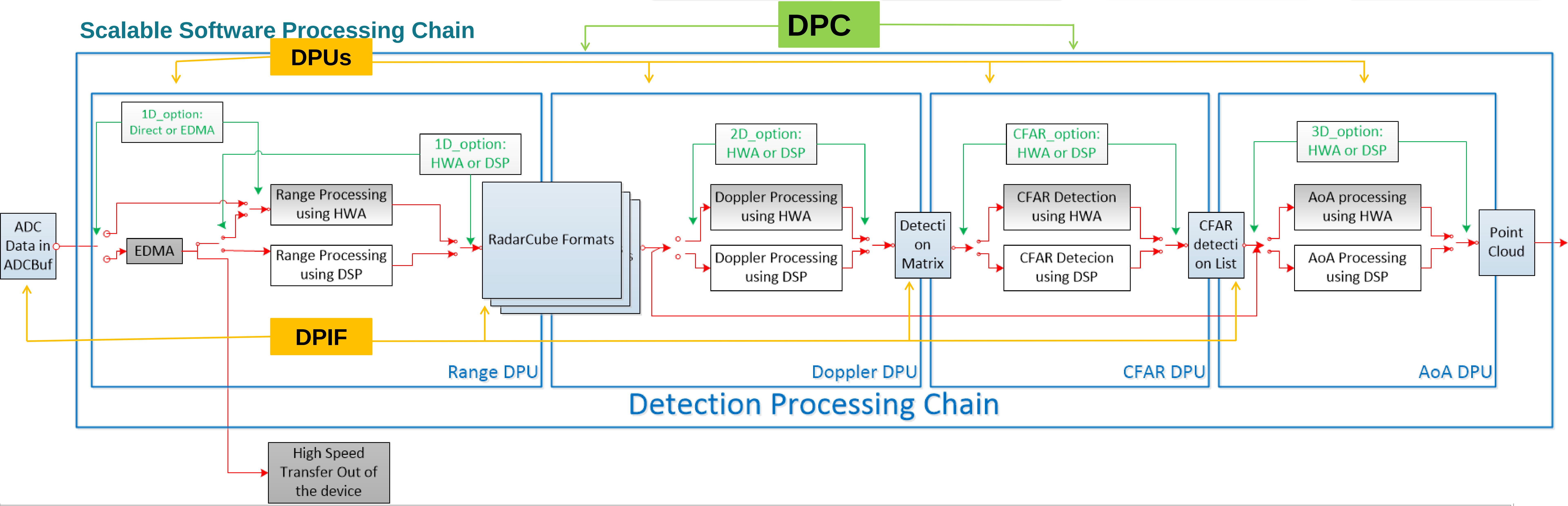


Scalable SW Chain

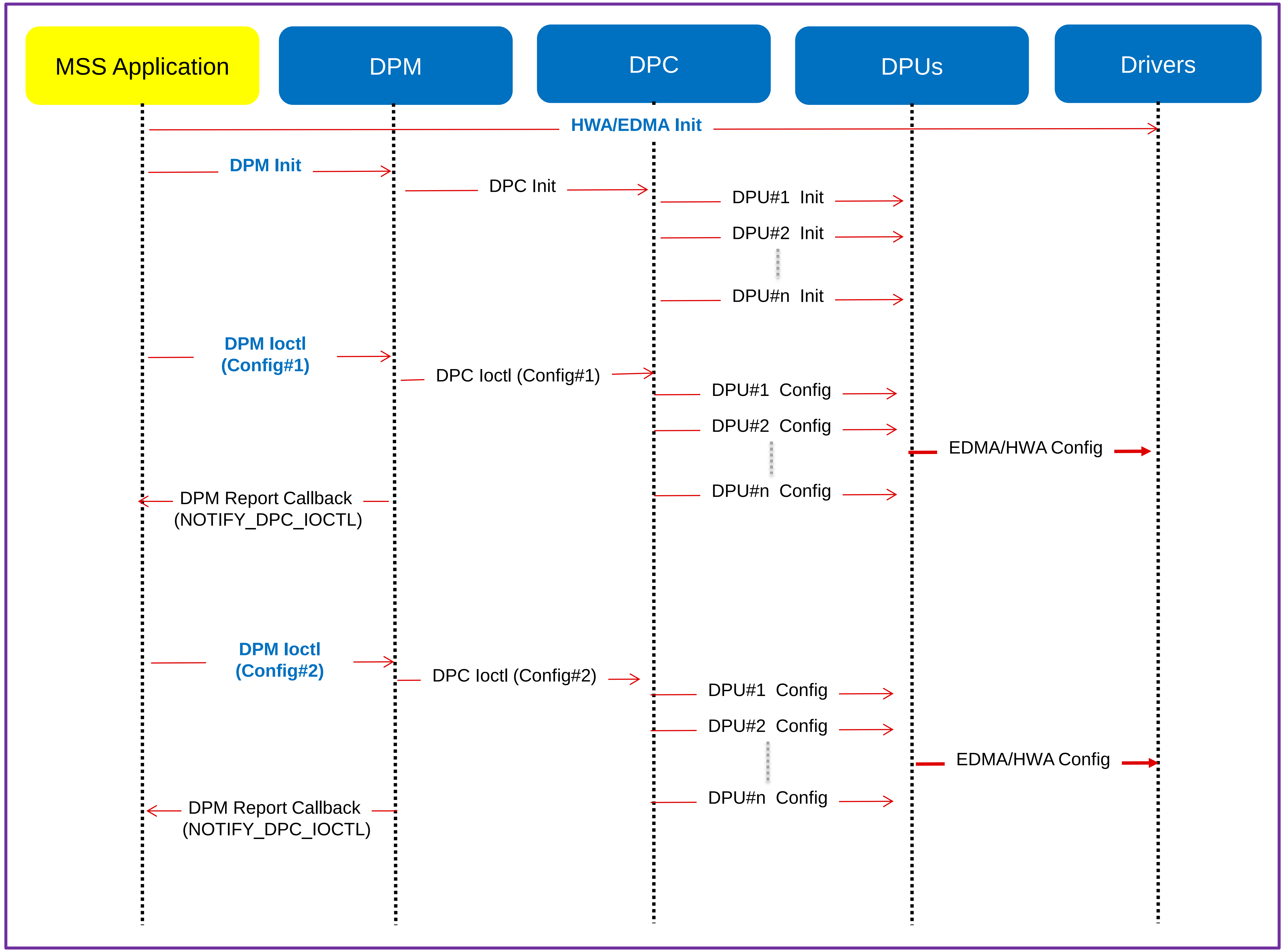
TI mmWave Sensor Portfolio



Scalable Software Processing Chain

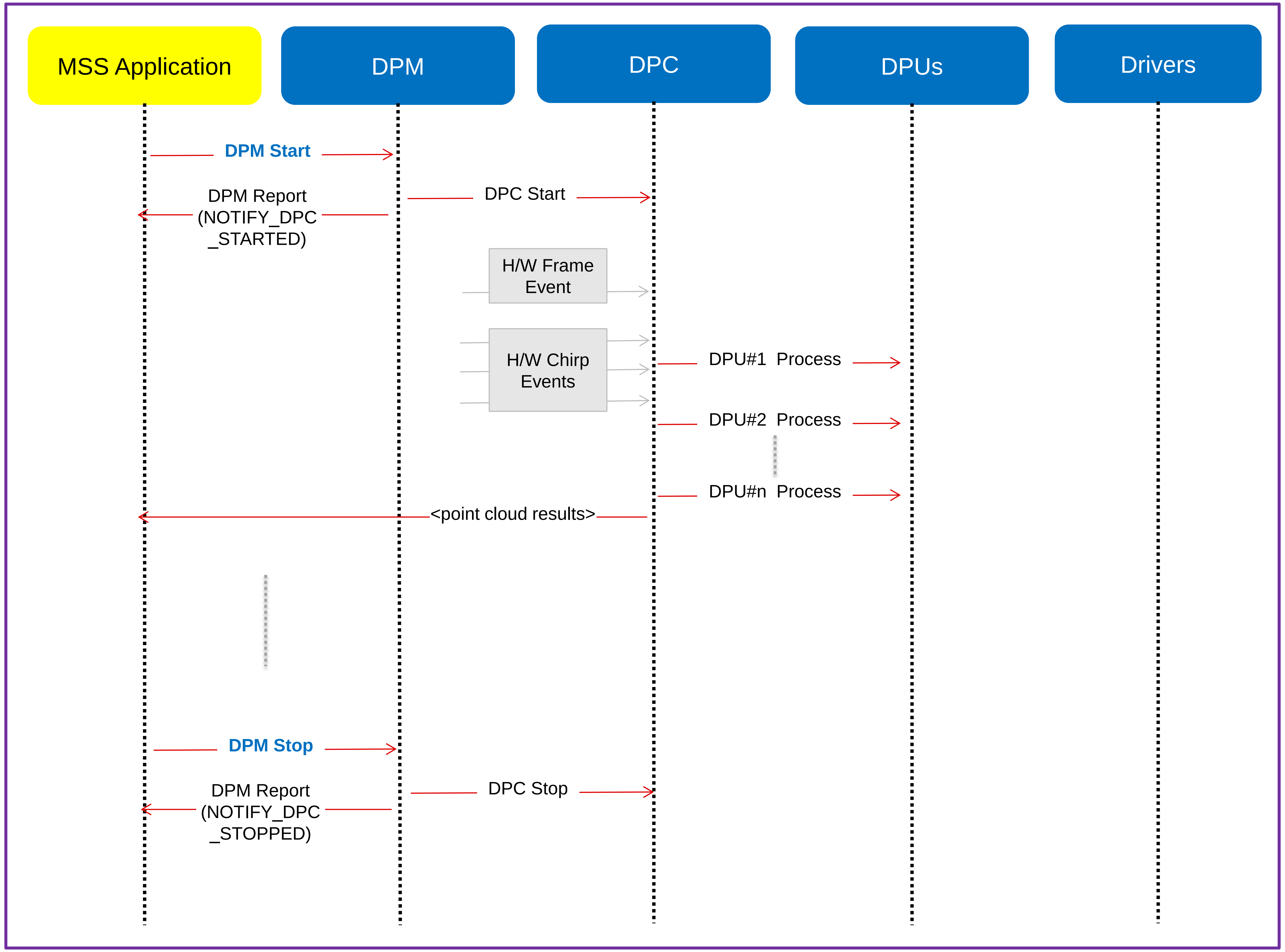


Typical call flow (1/2)



Config#1 and Config#2 are shown as examples here on how a config can be split into multiple shorrter structures as needed by a DPC. It is not mandatory to split into two. On the contrary, config can be split into even more smaller structures.

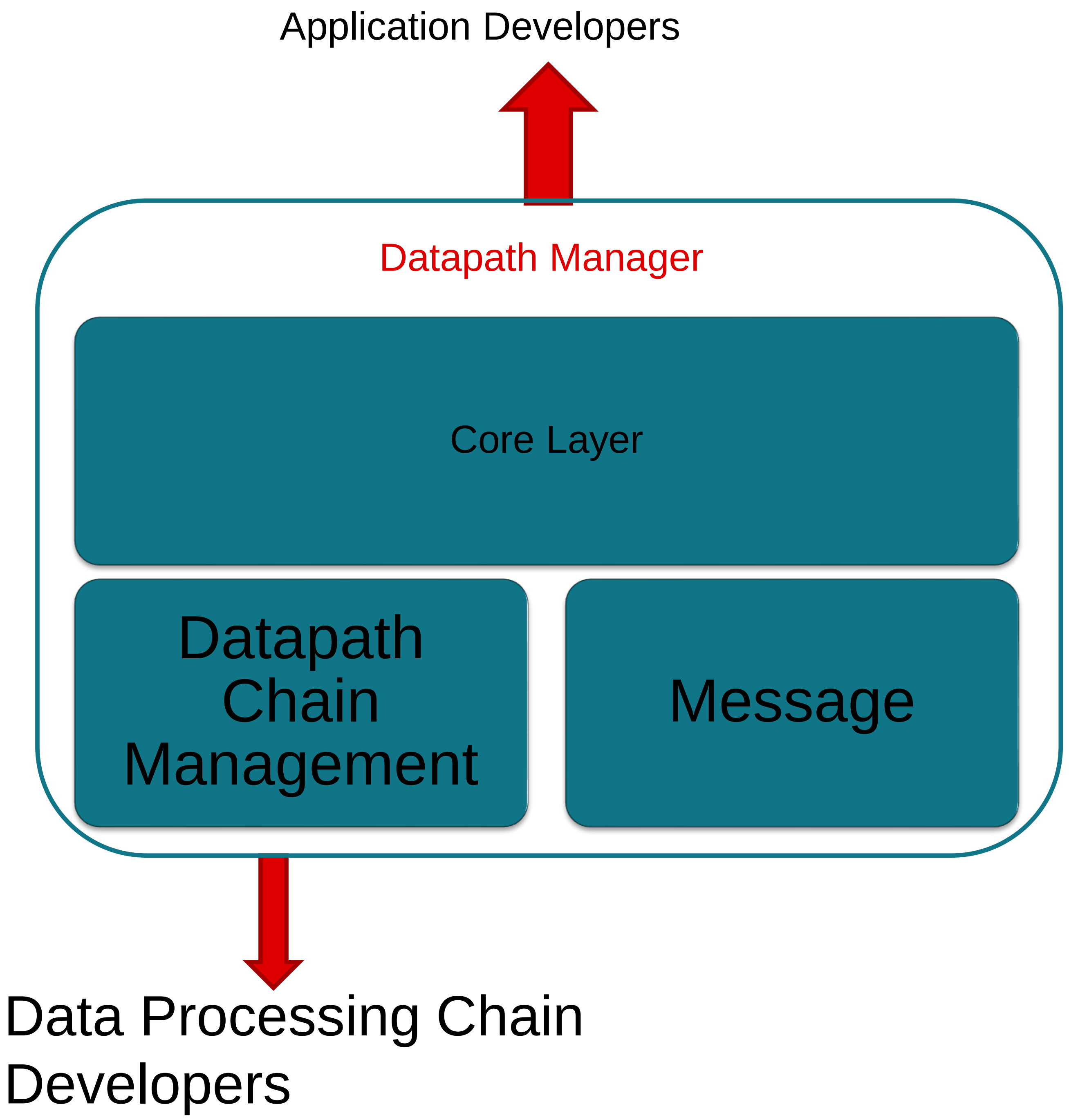
Typical call flow (2/2)



Data path deep dive

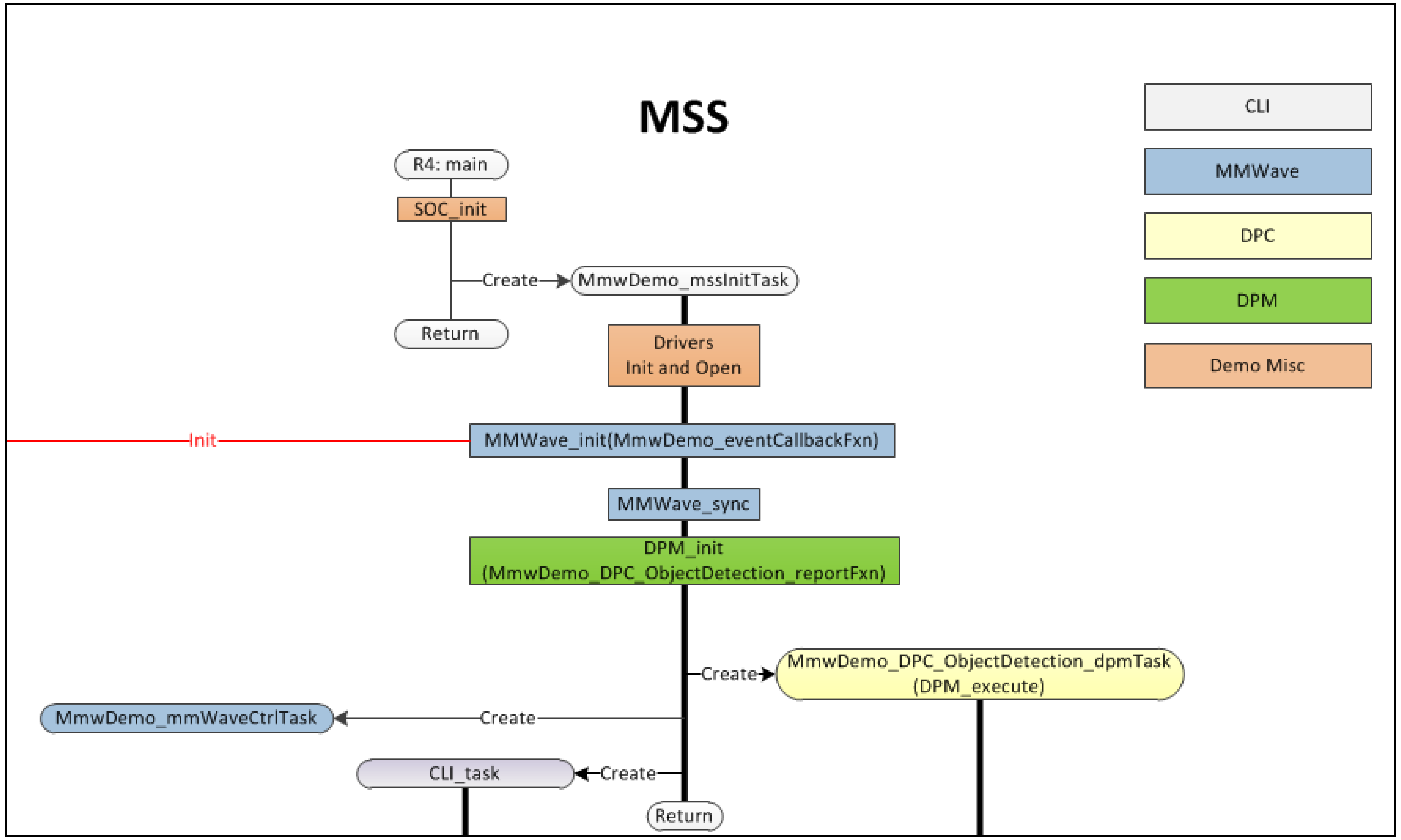
DPM: Datapath Manager

- Modular SW Architecture which provides an abstraction between the "Datapath Processing Chain" and the customer application.
  - Main task context which encapsulates the execution of DPC and DPUs
  - Application code instantiates DPM at start-up and registers a DPC.
- Provides a well-defined API
  - Exposed to the application to interface with the DPM
  - Exposed to the "*Data processing chain" developers* to be able to write their own code.
- Messaging mechanism
  - Send/Receive Configuration
  - Extends to Multiple-Thread/Core
  - Synchronized execution (No critical section required)
  - Response Mechanism with error code passing
- *Reporting mechanism* which allows applications to be notified about the status of the DPM/Datapath Processing Chain.



DPM Initialization

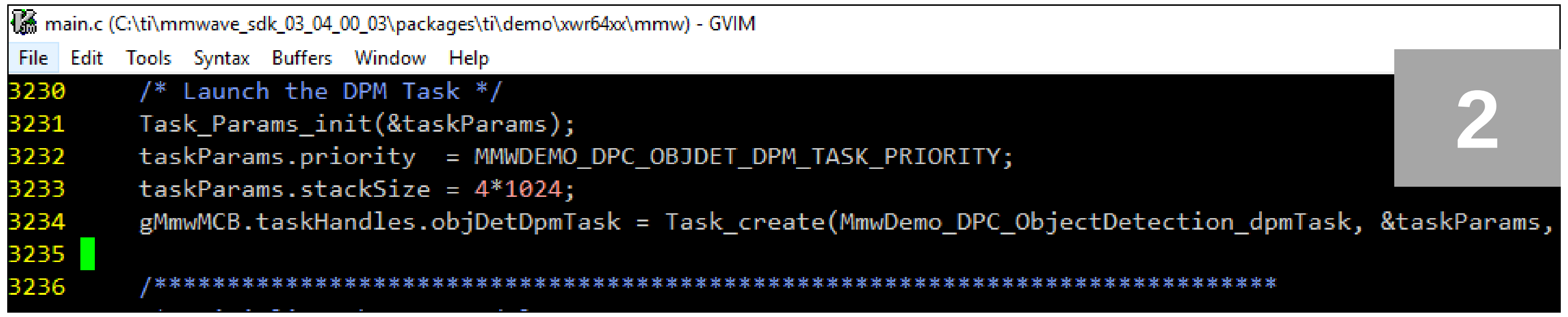
- Application code creates and initializes a DPM instance using the DPM_init function
- Application also creates a DPM task



Source: MMWAVE-SDK HTML documentation C:\ti\mmwave_sdk_03_xx_xx_xx\docs\mmwave_sdk_module_documentation.html

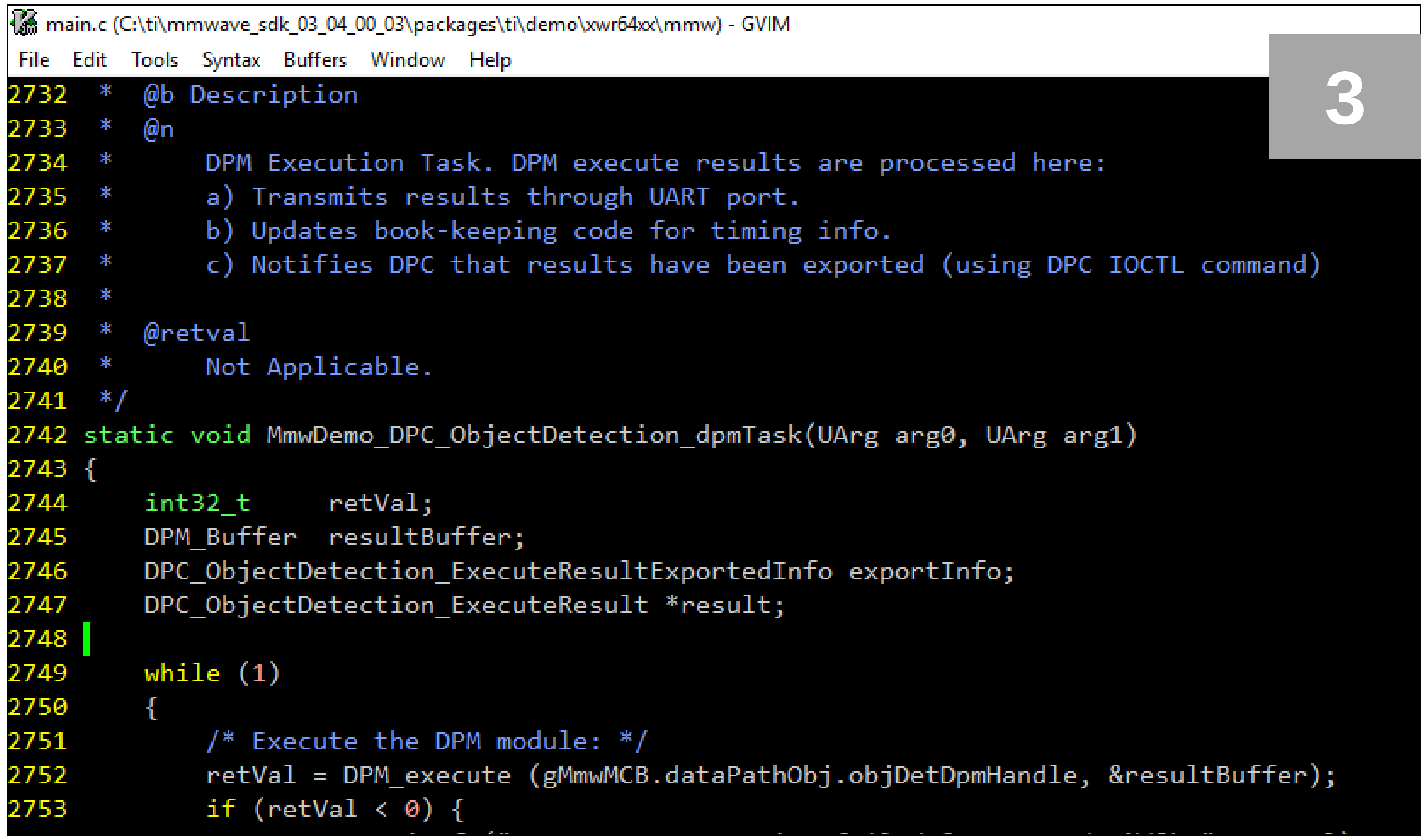
DPM Initialization

- Application code creates and initializes a DPM instance using the DPM_init function
- Application also creates a DPM task



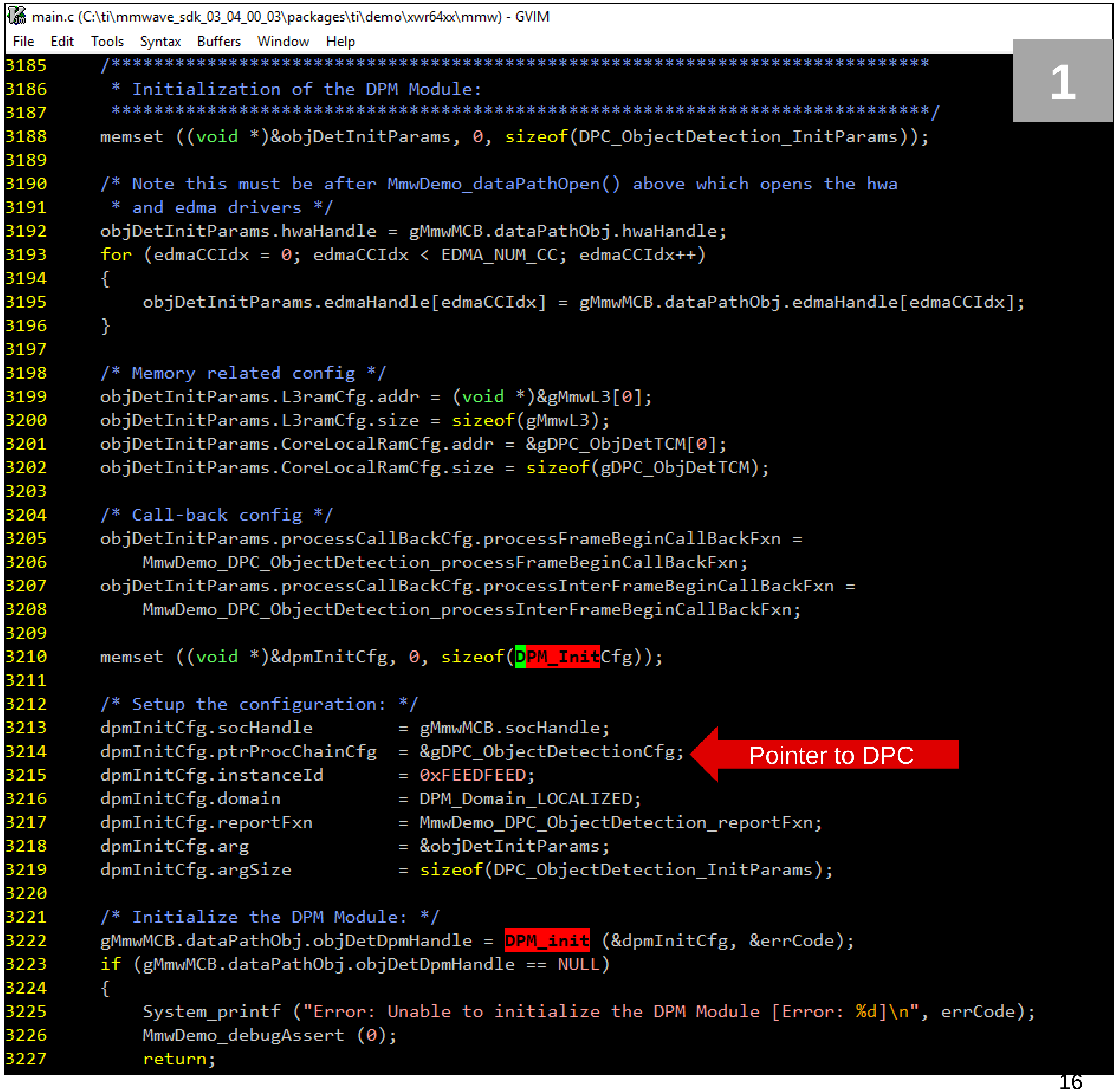
main.c (C:\ti\mmwave_sdk_03_04_00_03\packages\ti\demo\xwr64xx\mmw) - GVIM
File Edit Tools Syntax Buffers Window Help
2

```
    /* Launch the DPM Task */
    Task_Params_init(&taskParams);
    taskParams.priority  = MMWDEMO_DPC_OBJDET_DPM_TASK_PRIORITY;
    taskParams.stackSize = 4*1024;
    gMmwMCB.taskHandles.objDetDpmTask = Task_create(MmwDemo_DPC_ObjectDetection_dpmTask, &taskParams,

    /**************************************************************************
```



main.c (C:\ti\mmwave_sdk_03_04_00_03\packages\ti\demo\xwr64xx\mmw) - GVIM
File Edit Tools Syntax Buffers Window Help
3

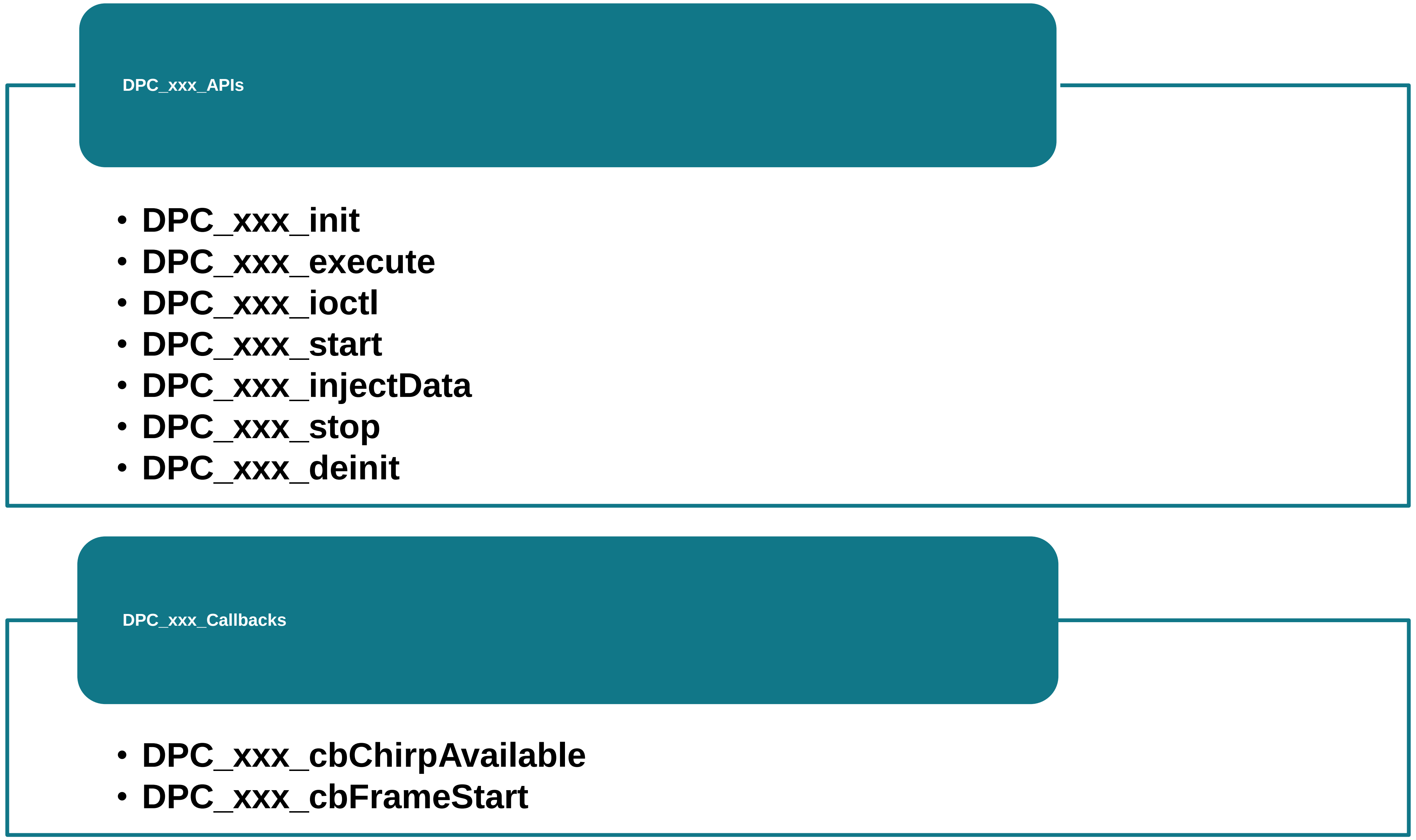
```
 *  @b Description
 *  @n
 *      DPM Execution Task. DPM execute results are processed here:
 *      a) Transmits results through UART port.
 *      b) Updates book-keeping code for timing info.
 *      c) Notifies DPC that results have been exported (using DPC IOCTL command)
 *
 *  @retval
 *      Not Applicable.
 */
static void MmwDemo_DPC_ObjectDetection_dpmTask(UArg arg0, UArg arg1)
{
    int32_t     retVal;
    DPM_Buffer  resultBuffer;
    DPC_ObjectDetection_ExecuteResultExportedInfo exportInfo;
    DPC_ObjectDetection_ExecuteResult *result;

    while (1)
    {
        /* Execute the DPM module: */
        retVal = DPM_execute (gMmwMCB.dataPathObj.objDetDpmHandle, &resultBuffer);
        if (retVal < 0) {
```



main.c (C:\ti\mmwave_sdk_03_04_00_03\packages\ti\demo\xwr64xx\mmw) - GVIM
File Edit Tools Syntax Buffers Window Help
1
Pointer to DPC

```
    /*******************************************************************************
     * Initialization of the DPM Module:
     *******************************************************************************/
    memset ((void *)&objDetInitParams, 0, sizeof(DPC_ObjectDetection_InitParams));

    /* Note this must be after MmwDemo_dataPathOpen() above which opens the hwa
     * and edma drivers */
    objDetInitParams.hwaHandle = gMmwMCB.dataPathObj.hwaHandle;
    for (edmaCCIdx = 0; edmaCCIdx < EDMA_NUM_CC; edmaCCIdx++)
    {
        objDetInitParams.edmaHandle[edmaCCIdx] = gMmwMCB.dataPathObj.edmaHandle[edmaCCIdx];
    }

    /* Memory related config */
    objDetInitParams.L3ramCfg.addr = (void *)&gMmwL3[0];
    objDetInitParams.L3ramCfg.size = sizeof(gMmwL3);
    objDetInitParams.CoreLocalRamCfg.addr = &gDPC_ObjDetTCM[0];
    objDetInitParams.CoreLocalRamCfg.size = sizeof(gDPC_ObjDetTCM);

    /* Call-back config */
    objDetInitParams.processCallBackCfg.processFrameBeginCallBackFxn =
        MmwDemo_DPC_ObjectDetection_processFrameBeginCallBackFxn;
    objDetInitParams.processCallBackCfg.processInterFrameBeginCallBackFxn =
        MmwDemo_DPC_ObjectDetection_processInterFrameBeginCallBackFxn;

    memset ((void *)&dpmInitCfg, 0, sizeof(DPM_InitCfg));

    /* Setup the configuration: */
    dpmInitCfg.socHandle        = gMmwMCB.socHandle;
    dpmInitCfg.ptrProcChainCfg  = &gDPC_ObjectDetectionCfg;
    dpmInitCfg.instanceId       = 0xFEEDFEED;
    dpmInitCfg.domain           = DPM_Domain_LOCALIZED;
    dpmInitCfg.reportFxn        = MmwDemo_DPC_ObjectDetection_reportFxn;
    dpmInitCfg.arg              = &objDetInitParams;
    dpmInitCfg.argSize          = sizeof(DPC_ObjectDetection_InitParams);

    /* Initialize the DPM Module: */
    gMmwMCB.dataPathObj.objDetDpmHandle = DPM_init (&dpmInitCfg, &errCode);
    if (gMmwMCB.dataPathObj.objDetDpmHandle == NULL)
    {
        System_printf ("Error: Unable to initialize the DPM Module [Error: %d]\n", errCode);
        MmwDemo_debugAssert (0);
        return;
```

DPC: Data Processing Chain



DPC_xxx_APIs

- DPC_xxx_init
- DPC_xxx_execute
- DPC_xxx_ioctl
- DPC_xxx_start
- DPC_xxx_injectData
- DPC_xxx_stop
- DPC_xxx_deinit

DPC_xxx_Callbacks

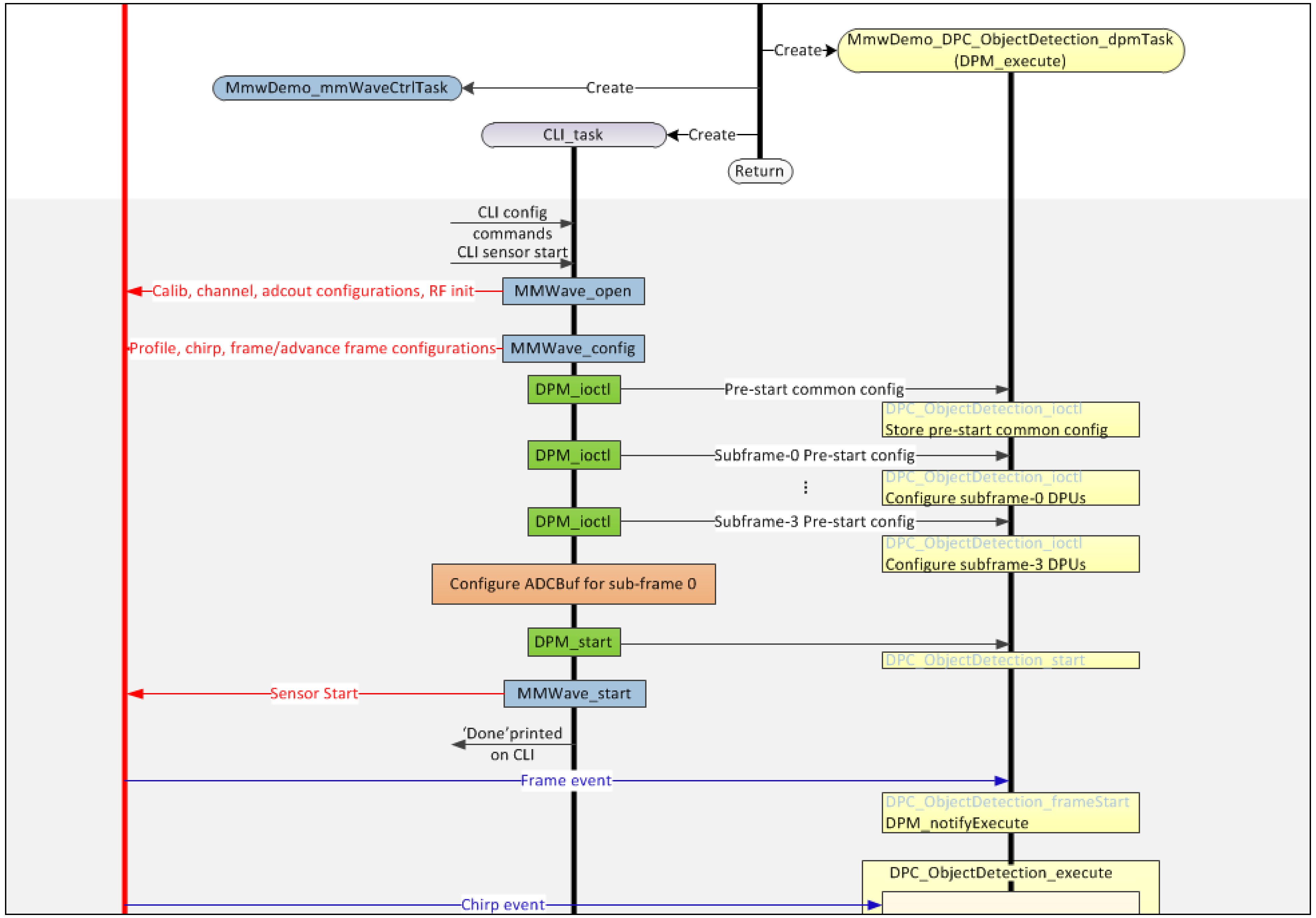
- DPC_xxx_cbChirpAvailable
- DPC_xxx_cbFrameStart

- All external DPC APIs starts with DPC_. DPC unique name follows next (follows coding guidelines).
    - DPC_ObjectDetection_Init
- External Mandatory APIs follows the prototype defined by the “Datapath Manager”
- DPCs have flexibility in defining their own content within the individual structure
- DPC that is split between MSS and DSS will expose two set of APIs - one for MSS and one for DSS. Depending on the functionality split between the two domains, not all APIs need to be implemented on the two domains

For more details, refer to docs folder in each of the DPCs

DPC Initialization

- Application registers the DPC with DPM during DPM creation.
- Application calls DPM init which calls the DPC's registered init function
- Application calls DPM_ioctl with different message types
- DPM_ioctl invokes the DPC's registered ioctl function with:
  - Pre-start common config message (common to all sub-frames) and
  - Sub-frame specific Pre-start config messages.
- DPC handles the messages in the ioctl function and performs the corresponding configuration.



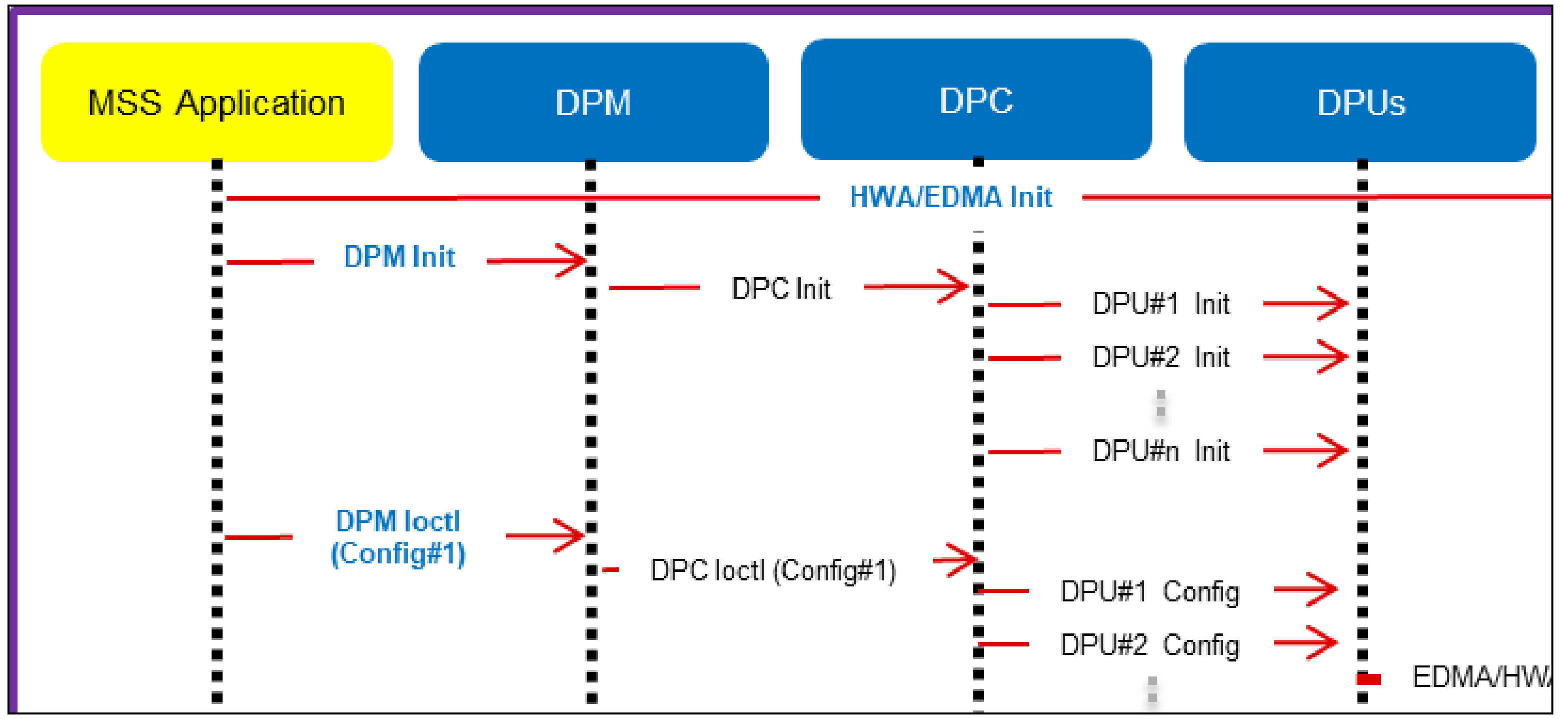
Create
MmwDemo_DPC_ObjectDetection_dpmTask (DPM_execute)
MmwDemo_mmWaveCtrlTask
Create
CLI_task
Create
Return
CLI config commands
CLI sensor start
Calib, channel, adcout configurations, RF init
MMWave_open
Profile, chirp, frame/advance frame configurations
MMWave_config
DPM_ioctl
Pre-start common config
DPC_ObjectDetection_ioctl
Store pre-start common config
DPM_ioctl
Subframe-0 Pre-start config
⋮
DPC_ObjectDetection_ioctl
Configure subframe-0 DPUs
DPM_ioctl
Subframe-3 Pre-start config
DPC_ObjectDetection_ioctl
Configure subframe-3 DPUs
Configure ADCBuf for sub-frame 0
DPM_start
DPC_ObjectDetection_start
Sensor Start
MMWave_start
'Done'printed on CLI
Frame event
DPC_ObjectDetection_frameStart
DPM_notifyExecute
DPC_ObjectDetection_execute
Chirp event

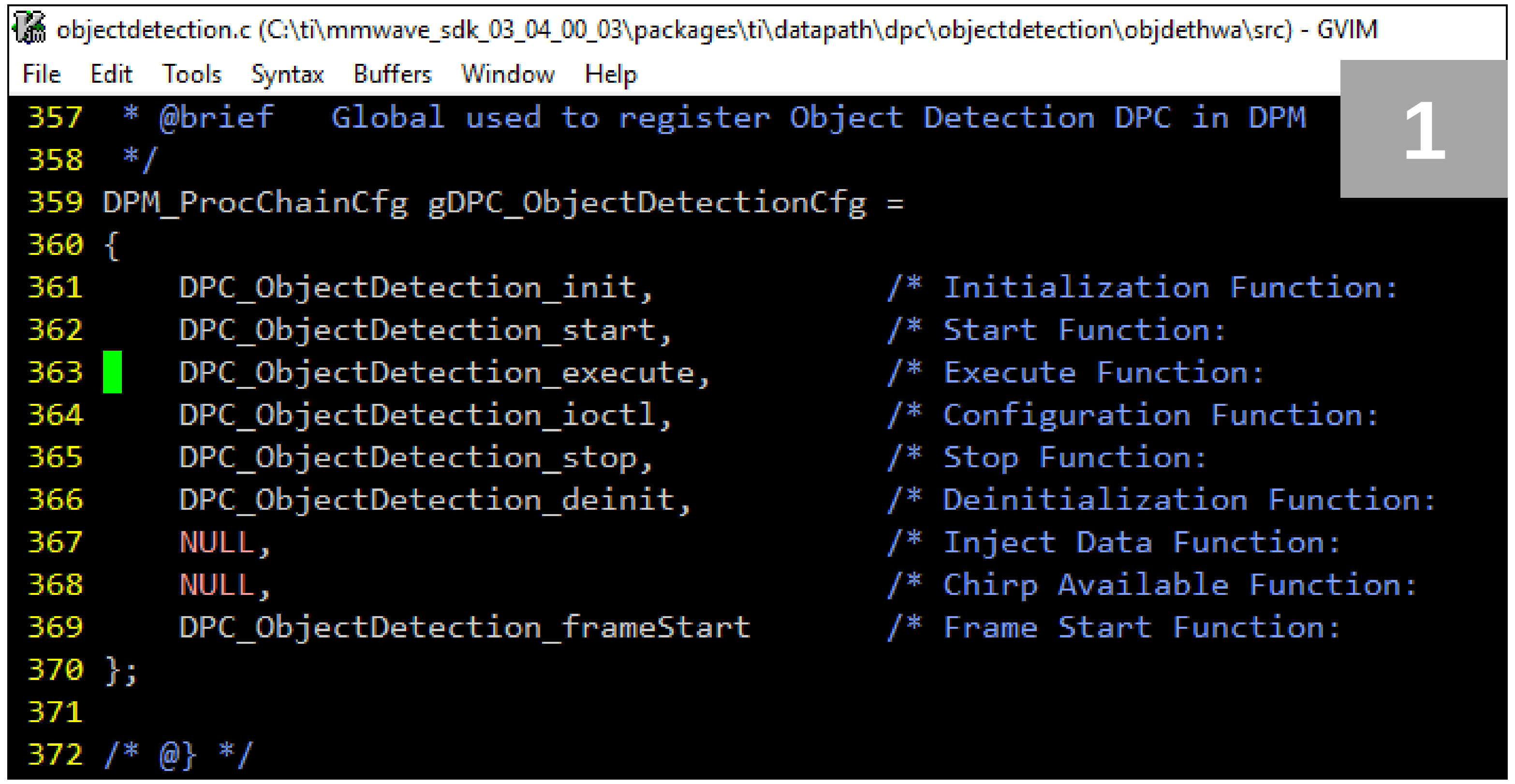
Source: MMWAVE-SDK HTML documentation

C:\ti\mmwave_sdk_03_xx_xx_xx\docs\mmwave_sdk_module_documentation.html

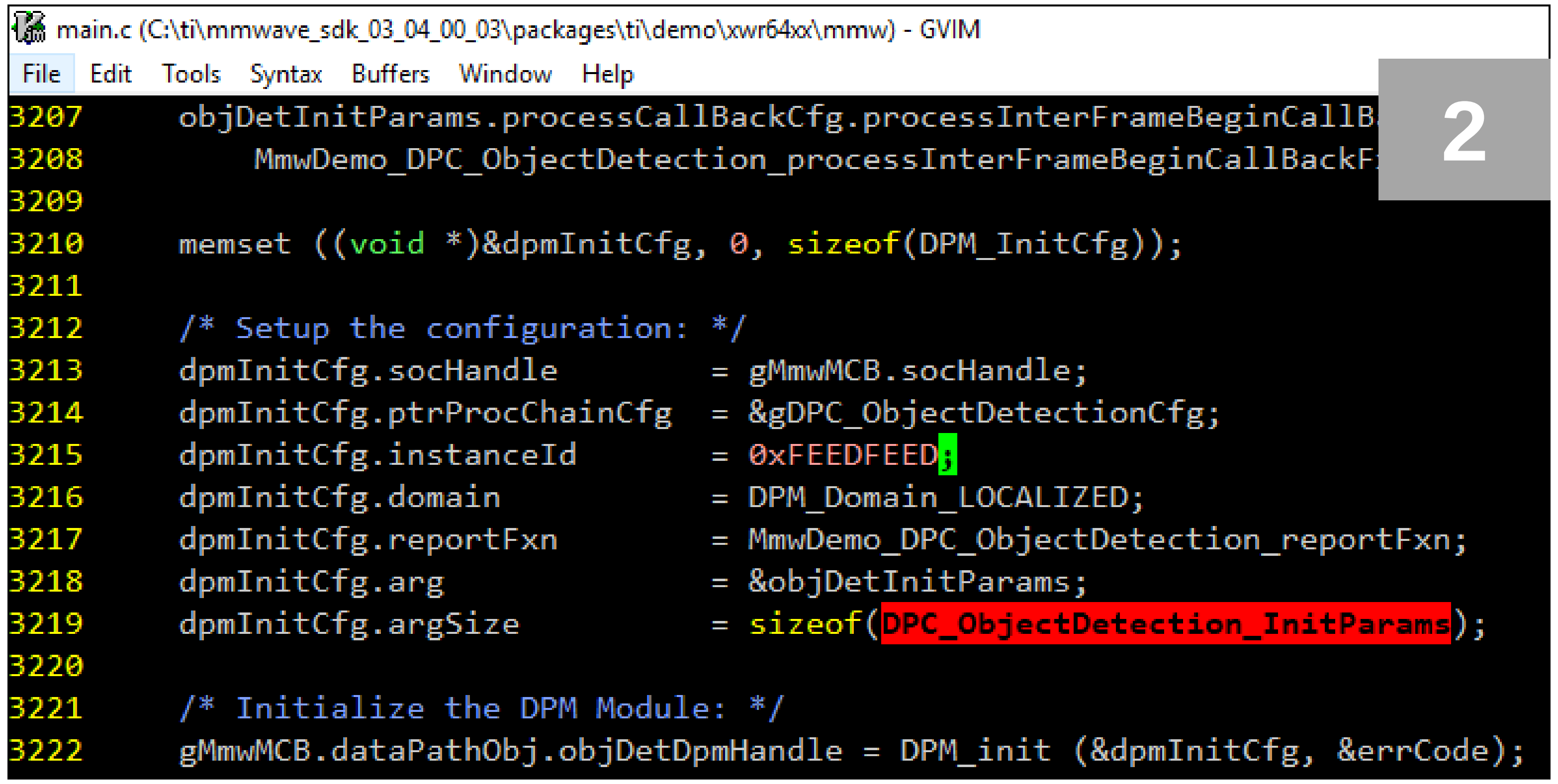
DPC Initialization

- Application registers the DPC with DPM during DPM creation.
- Application calls DPM init which calls the DPC's registered init function
- Application calls DPM_ioctl with different message types
- DPM_ioctl internally invokes the DPC's registered ioctl function and passes the message to it
- DPC handles the message in the ioctl function and performs the corresponding configuration
- Refer to the call flow below




objectdetection.c (C:\ti\mmwave_sdk_03_04_00_03\packages\ti\datapath\dpc\objectdetection\objdethwa\src) - GVIM
File Edit Tools Syntax Buffers Window Help
1

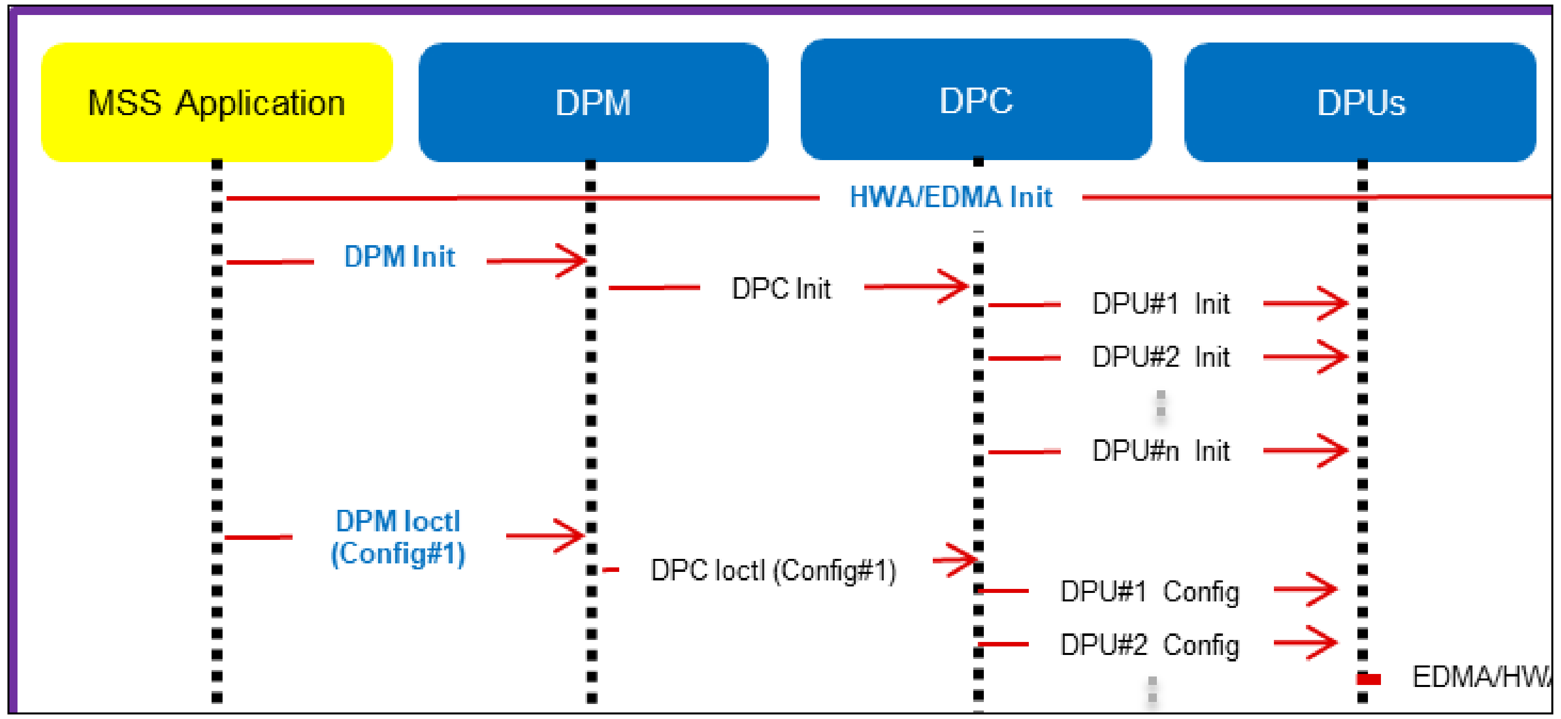
```
 * @brief   Global used to register Object Detection DPC in DPM
 */
DPM_ProcChainCfg gDPC_ObjectDetectionCfg =
{
    DPC_ObjectDetection_init,            /* Initialization Function:
    DPC_ObjectDetection_start,           /* Start Function:
    DPC_ObjectDetection_execute,         /* Execute Function:
    DPC_ObjectDetection_ioctl,           /* Configuration Function:
    DPC_ObjectDetection_stop,            /* Stop Function:
    DPC_ObjectDetection_deinit,          /* Deinitialization Function:
    NULL,                                /* Inject Data Function:
    NULL,                                /* Chirp Available Function:
    DPC_ObjectDetection_frameStart       /* Frame Start Function:
};

/* @} */
```

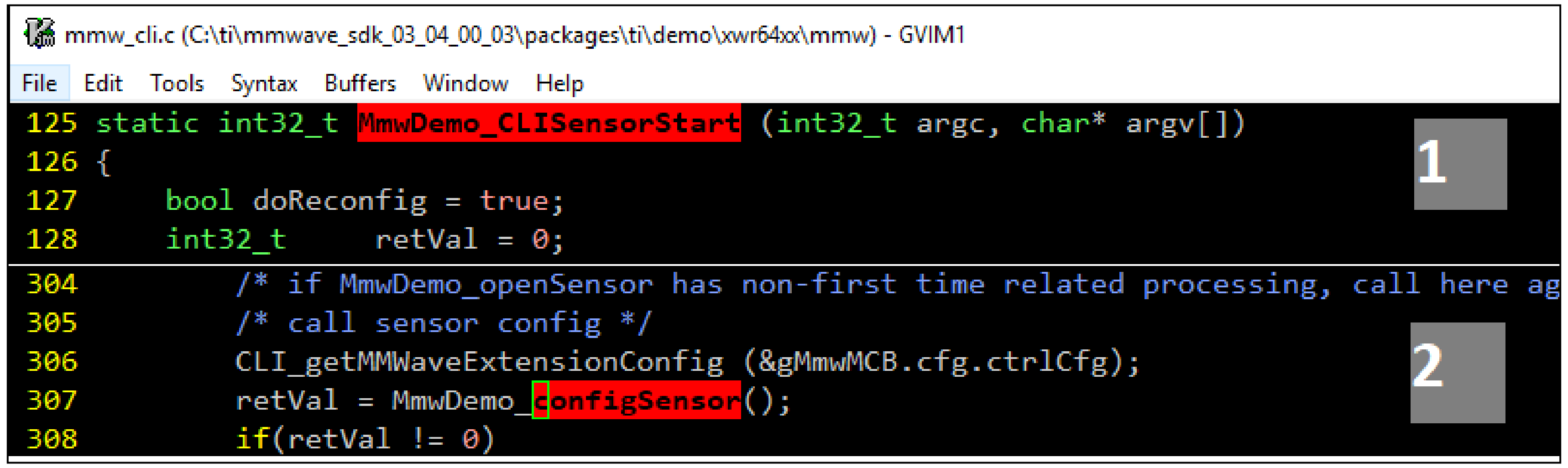

main.c (C:\ti\mmwave_sdk_03_04_00_03\packages\ti\demo\xwr64xx\mmw) - GVIM
File Edit Tools Syntax Buffers Window Help
2

```
    objDetInitParams.processCallBackCfg.processInterFrameBeginCallB
        MmwDemo_DPC_ObjectDetection_processInterFrameBeginCallBackF

    memset ((void *)&dpmInitCfg, 0, sizeof(DPM_InitCfg));

    /* Setup the configuration: */
    dpmInitCfg.socHandle           = gMmwMCB.socHandle;
    dpmInitCfg.ptrProcChainCfg     = &gDPC_ObjectDetectionCfg;
    dpmInitCfg.instanceId          = 0xFEEDFEED;
    dpmInitCfg.domain              = DPM_Domain_LOCALIZED;
    dpmInitCfg.reportFxn           = MmwDemo_DPC_ObjectDetection_reportFxn;
    dpmInitCfg.arg                 = &objDetInitParams;
    dpmInitCfg.argSize             = sizeof(DPC_ObjectDetection_InitParams);

    /* Initialize the DPM Module: */
    gMmwMCB.dataPathObj.objDetDpmHandle = DPM_init (&dpmInitCfg, &errCode);
```

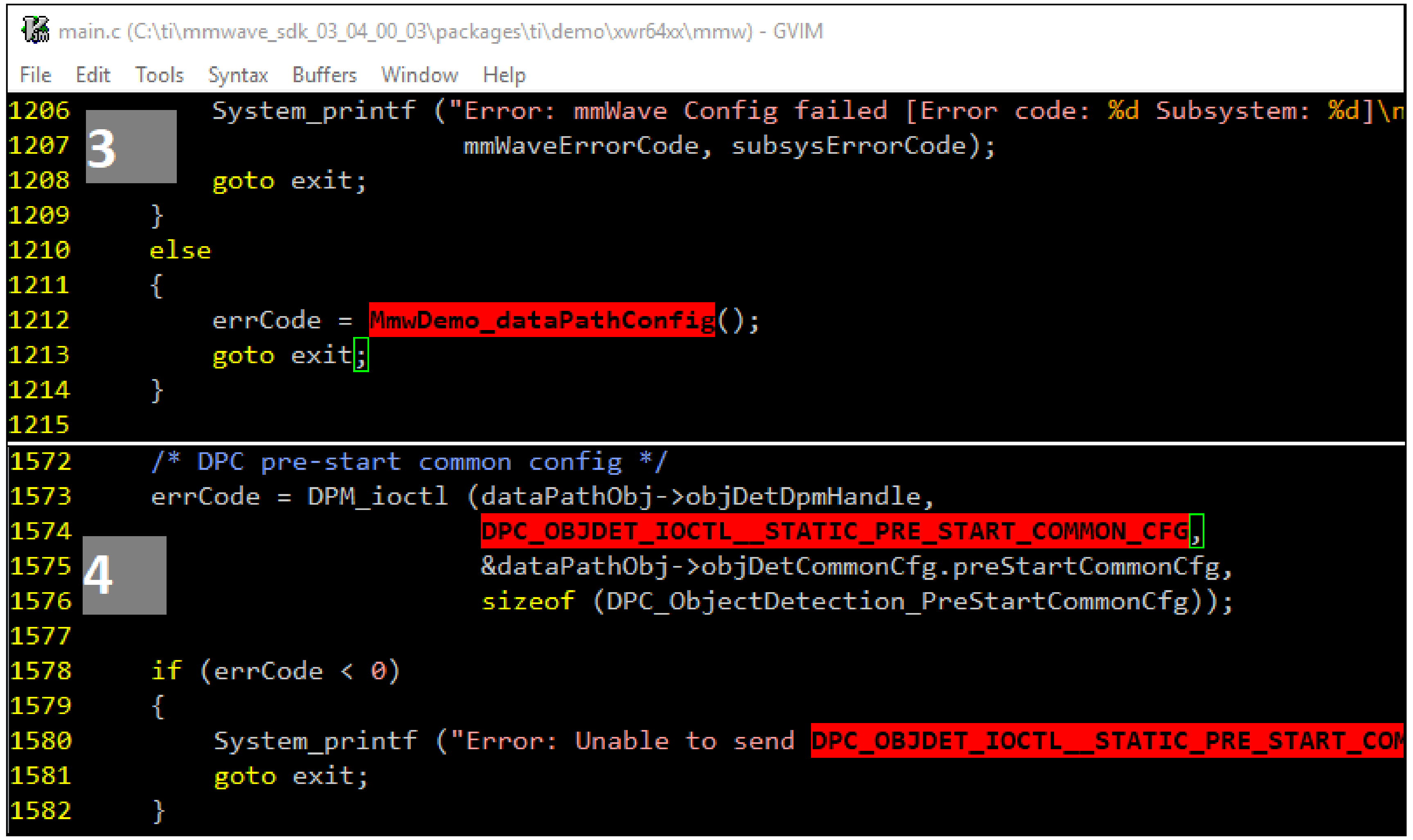
DPC Configuration

- Application registers the DPC with DPM during DPM creation.
- Application calls DPM init which calls the DPC's registered init function
- Application calls DPM_ioctl with different message types (i.e. commands)
- DPM_ioctl internally invokes the DPC's registered ioctl function and passes the message to it
- DPC handles the message in the ioctl function and performs the corresponding configuration
- Refer to the call flow below




mmw_cli.c (C:\ti\mmwave_sdk_03_04_00_03\packages\ti\demo\xwr64xx\mmw) - GVIM1
File Edit Tools Syntax Buffers Window Help
1
2

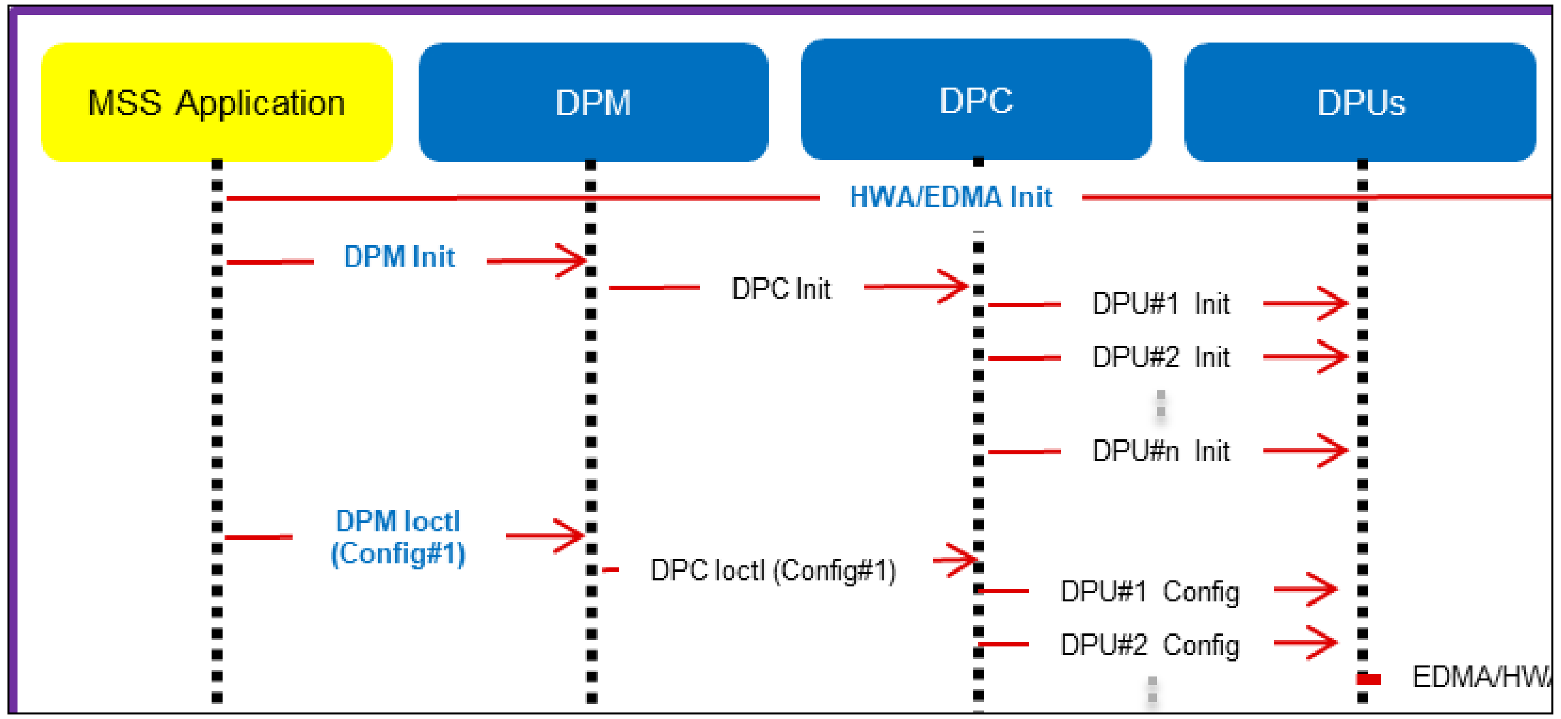
```
static int32_t MmwDemo_CLISensorStart (int32_t argc, char* argv[])
{
    bool doReconfig = true;
    int32_t     retVal = 0;
        /* if MmwDemo_openSensor has non-first time related processing, call here ag
        /* call sensor config */
        CLI_getMMWaveExtensionConfig (&gMmwMCB.cfg.ctrlCfg);
        retVal = MmwDemo_configSensor();
        if(retVal != 0)
```

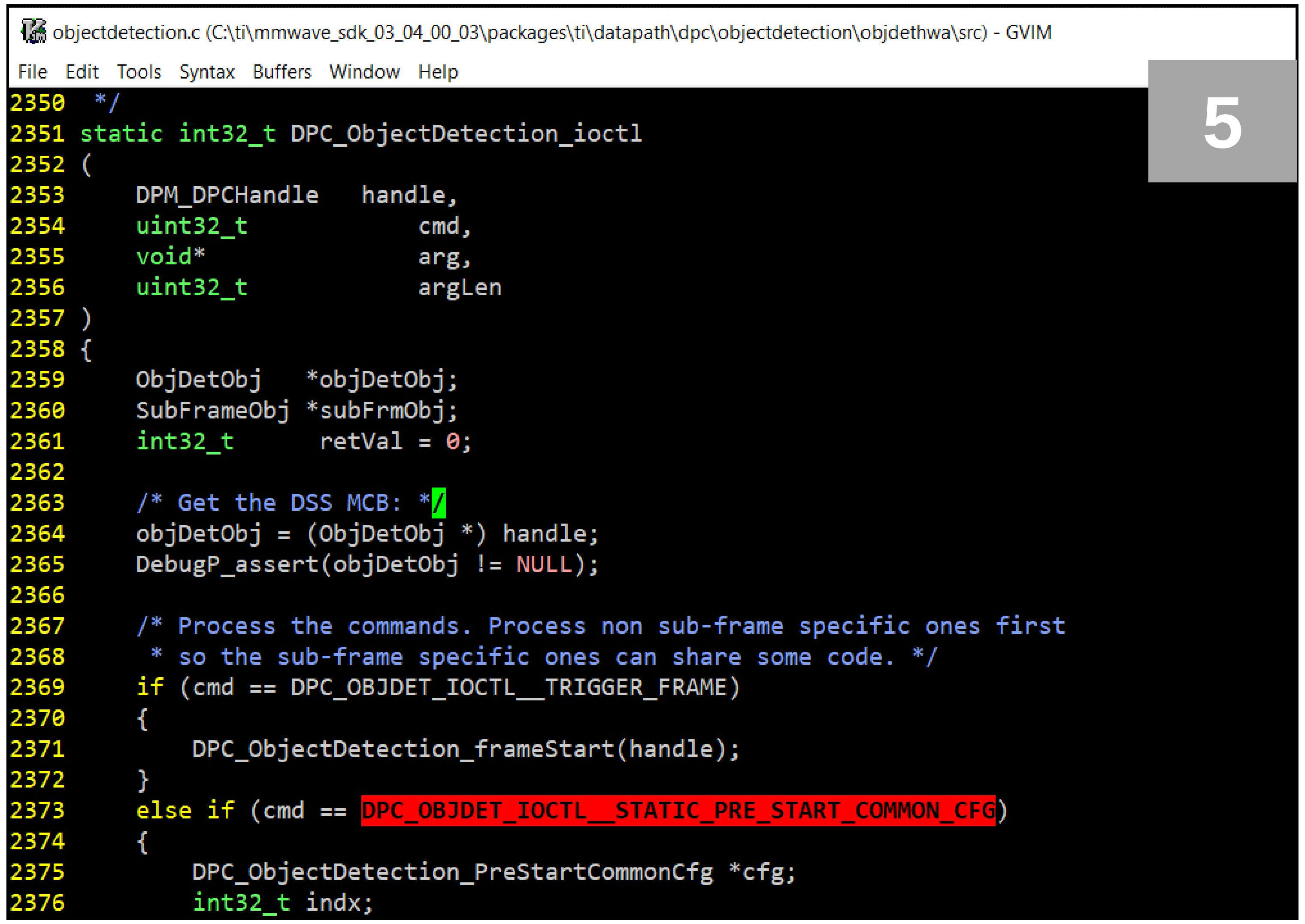

main.c (C:\ti\mmwave_sdk_03_04_00_03\packages\ti\demo\xwr64xx\mmw) - GVIM
File Edit Tools Syntax Buffers Window Help
3
4

```
        System_printf ("Error: mmWave Config failed [Error code: %d Subsystem: %d]\n
                        mmWaveErrorCode, subsysErrorCode);
        goto exit;
    }
    else
    {
        errCode = MmwDemo_dataPathConfig();
        goto exit;
    }

    /* DPC pre-start common config */
    errCode = DPM_ioctl (dataPathObj->objDetDpmHandle,
                         DPC_OBJDET_IOCTL__STATIC_PRE_START_COMMON_CFG,
                         &dataPathObj->objDetCommonCfg.preStartCommonCfg,
                         sizeof (DPC_ObjectDetection_PreStartCommonCfg));

    if (errCode < 0)
    {
        System_printf ("Error: Unable to send DPC_OBJDET_IOCTL__STATIC_PRE_START_COM
        goto exit;
    }
```

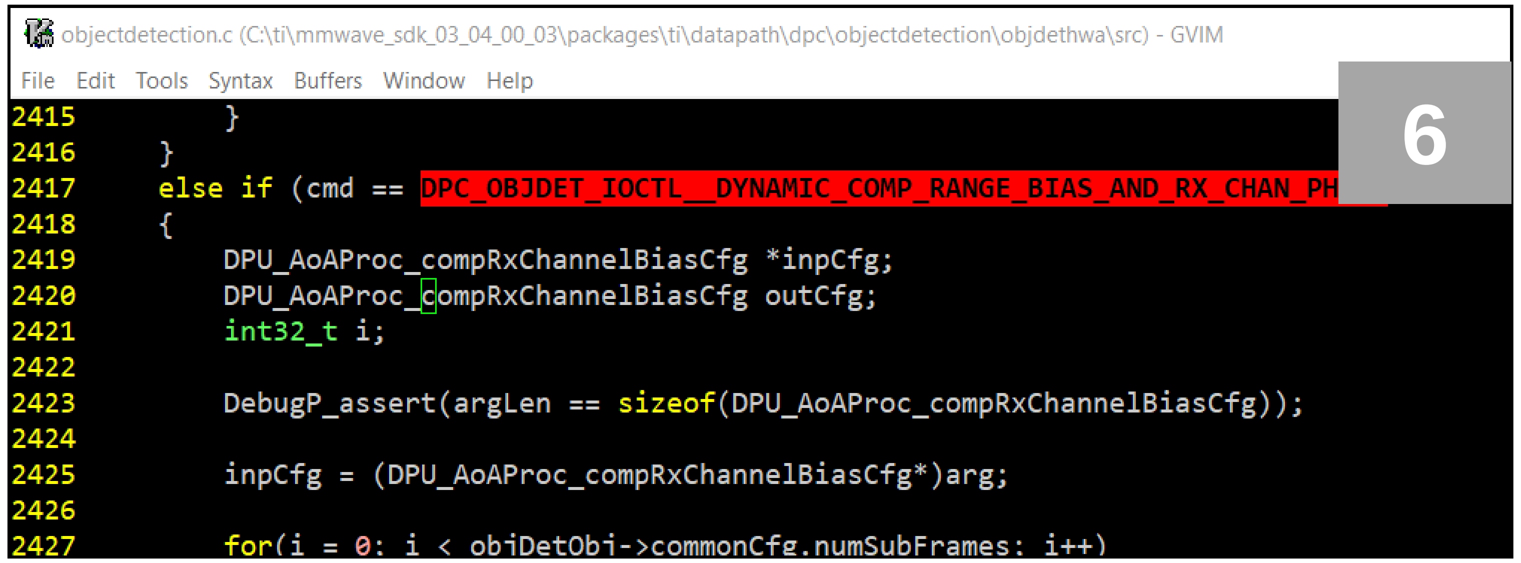
DPC Configuration (continued)

- Application registers the DPC with DPM during DPM creation.
- Application calls DPM init which calls the DPC’s registered init function
- Application calls DPM_ioctl with different message types
- DPM_ioctl internally invokes the DPC’s registered ioctl function and passes the message to it
- DPC handles the message in the ioctl function and performs the corresponding configuration
- Refer to the call flow below




objectdetection.c (C:\ti\mmwave_sdk_03_04_00_03\packages\ti\datapath\dpc\objectdetection\objdethwa\src) - GVIM
File Edit Tools Syntax Buffers Window Help
5

```
 */
static int32_t DPC_ObjectDetection_ioctl
(
    DPM_DPCHandle   handle,
    uint32_t            cmd,
    void*               arg,
    uint32_t            argLen
)
{
    ObjDetObj   *objDetObj;
    SubFrameObj *subFrmObj;
    int32_t      retVal = 0;

    /* Get the DSS MCB: */
    objDetObj = (ObjDetObj *) handle;
    DebugP_assert(objDetObj != NULL);

    /* Process the commands. Process non sub-frame specific ones first
     * so the sub-frame specific ones can share some code. */
    if (cmd == DPC_OBJDET_IOCTL__TRIGGER_FRAME)
    {
        DPC_ObjectDetection_frameStart(handle);
    }
    else if (cmd == DPC_OBJDET_IOCTL__STATIC_PRE_START_COMMON_CFG)
    {
        DPC_ObjectDetection_PreStartCommonCfg *cfg;
        int32_t indx;
```


objectdetection.c (C:\ti\mmwave_sdk_03_04_00_03\packages\ti\datapath\dpc\objectdetection\objdethwa\src) - GVIM
File Edit Tools Syntax Buffers Window Help
6

```
        }
    }
    else if (cmd == DPC_OBJDET_IOCTL__DYNAMIC_COMP_RANGE_BIAS_AND_RX_CHAN_PH
    {
        DPU_AoAProc_compRxChannelBiasCfg *inpCfg;
        DPU_AoAProc_compRxChannelBiasCfg outCfg;
        int32_t i;

        DebugP_assert(argLen == sizeof(DPU_AoAProc_compRxChannelBiasCfg));

        inpCfg = (DPU_AoAProc_compRxChannelBiasCfg*)arg;

        for(i = 0; i < objDetObj->commonCfg.numSubFrames; i++)
```

DPU: Data Processing Units

DPU_xxx_init

- **DPU_xxx_InitParams_t**
- **errCode**
- **Handle**

DPU_xxx_config

- **DPU_Handle**
- **DPU_xxx_Config_t**
  - H/W Resources (EDMA, HWA, I/O buffer pointers, Scratch buffer pointers)
  - Frame/Sub-frame DPU Static Config (Ex: Num ADC Samples, Chirps/Frame, ADCBuf Config, Data Interface Desc)
  - Frame/Sub-frame DPU Dynamic Config (Ex: DC Range Calibration)
- **errCode**

DPU_xxx_process

- **DPU_Handle**
- **DPU_xxx_OutParams_t**
  - DPU_xxx_Stats_t
  - DPU optional specific Params (Ex: isLastChirp)
- **errCode**

DPU_xxx_control

- **DPU_Handle**
- **cmd**
- **args**
- **argSize**
- **errCode**

DPU_xxx_deinit

- **DPU_handle**
- **errCode**

- All external DPU APIs start with the prefix DPU_. DPU unique name follows next..Ex: DPU_RangeProcHWA_init

- Standard external APIs:
  - **Init**: one time initialization of DPU
  - **Config**: complete configuration of the DPU: hardware resources, static and dynamic (if supported by DPU)
    - **static config**: config that is static during ongoing frames
    - **dynamic config**: config that can be changed from frame to frame but only when process is not ongoing - ideally interframe time after DPC has exported the results for the frame
  - **Process**: the actual processing function of the DPU
  - **Control**: ioctl interface that allows higher layer to switch dynamic configuration during interframe time
  - **De-init**: de-initialization of DPU

- All memory allocations for I/O buffers and scratch buffers are outside the DPU since mmWave applications rely on memory overlay technique for optimization and that is best handled at application level

- All H/W resources must be allocated by application and passed to the DPU. This helps in keeping DPU platform agnostic as well as allows application to share the resources across DPU when DPU processing doesn't overlap in time.

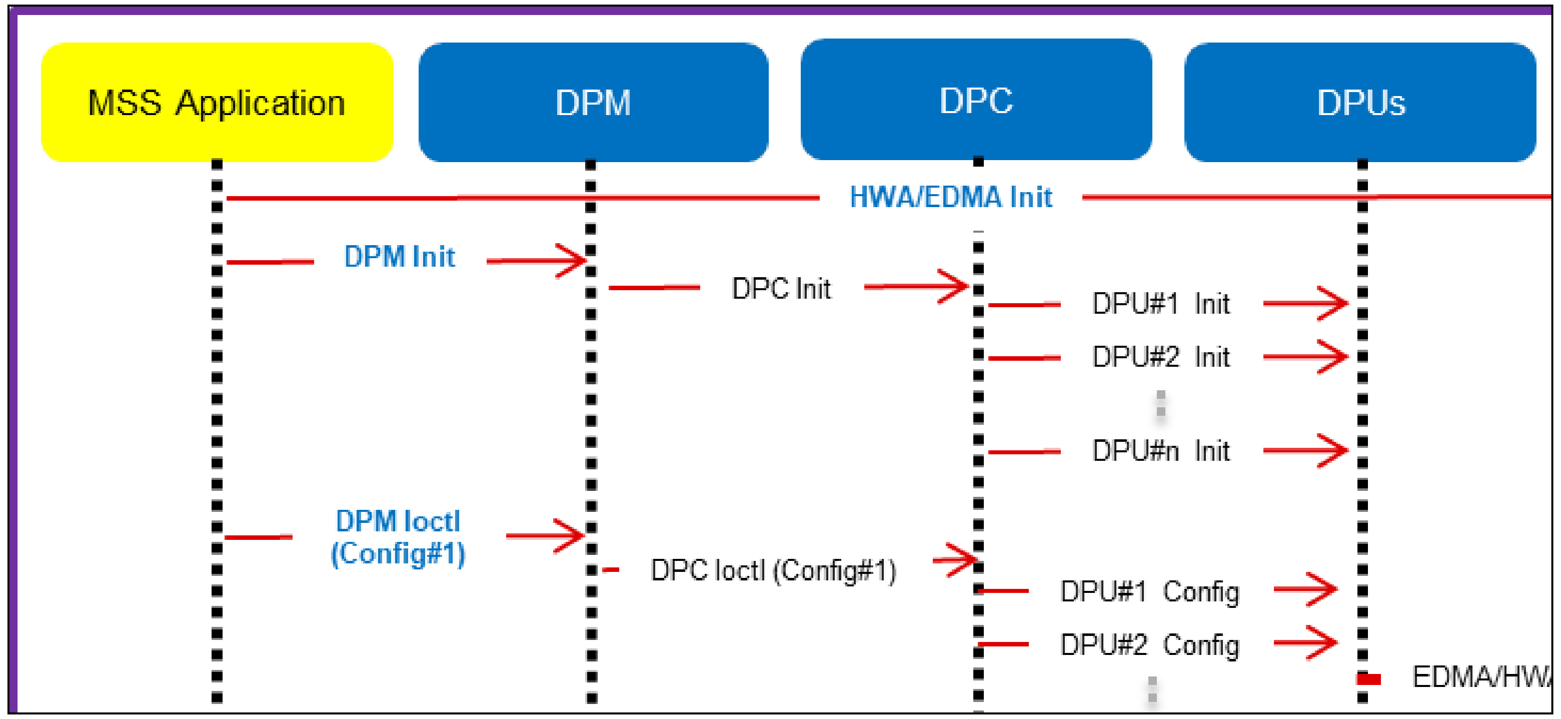
- DPUs are OS agnostic and use OSAL APIs for needed OS services.

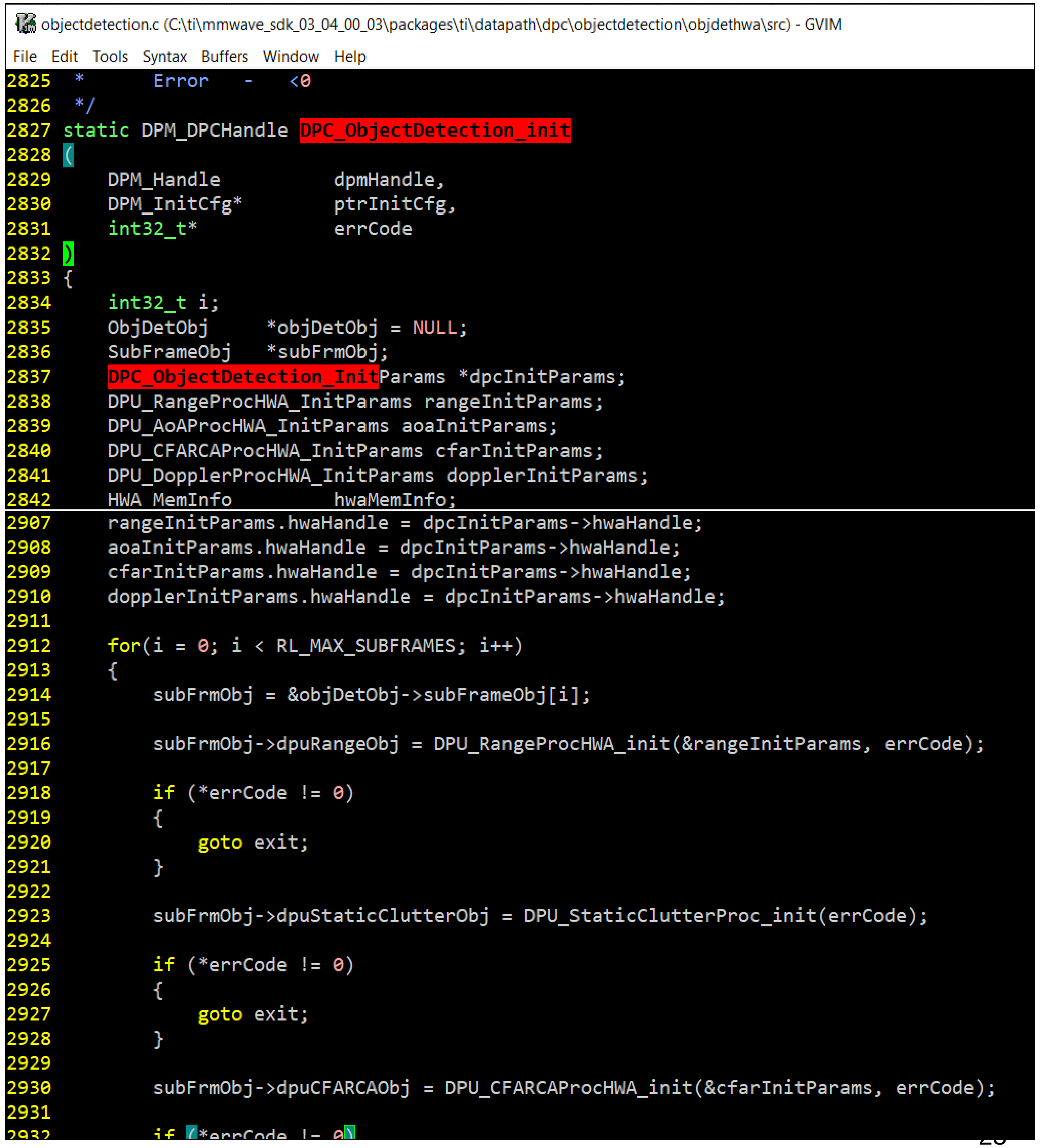
For more details, refer to docs folder in each of the DPUs

DPU Initialization

Recall from DPC Configuration…

- Application calls DPM init which calls the DPC's registered init function
- DPC_init calls the various DPU init functions in sequence, to create separate DPU instances for the number of sub-frames configured.
- Refer to the call flow below



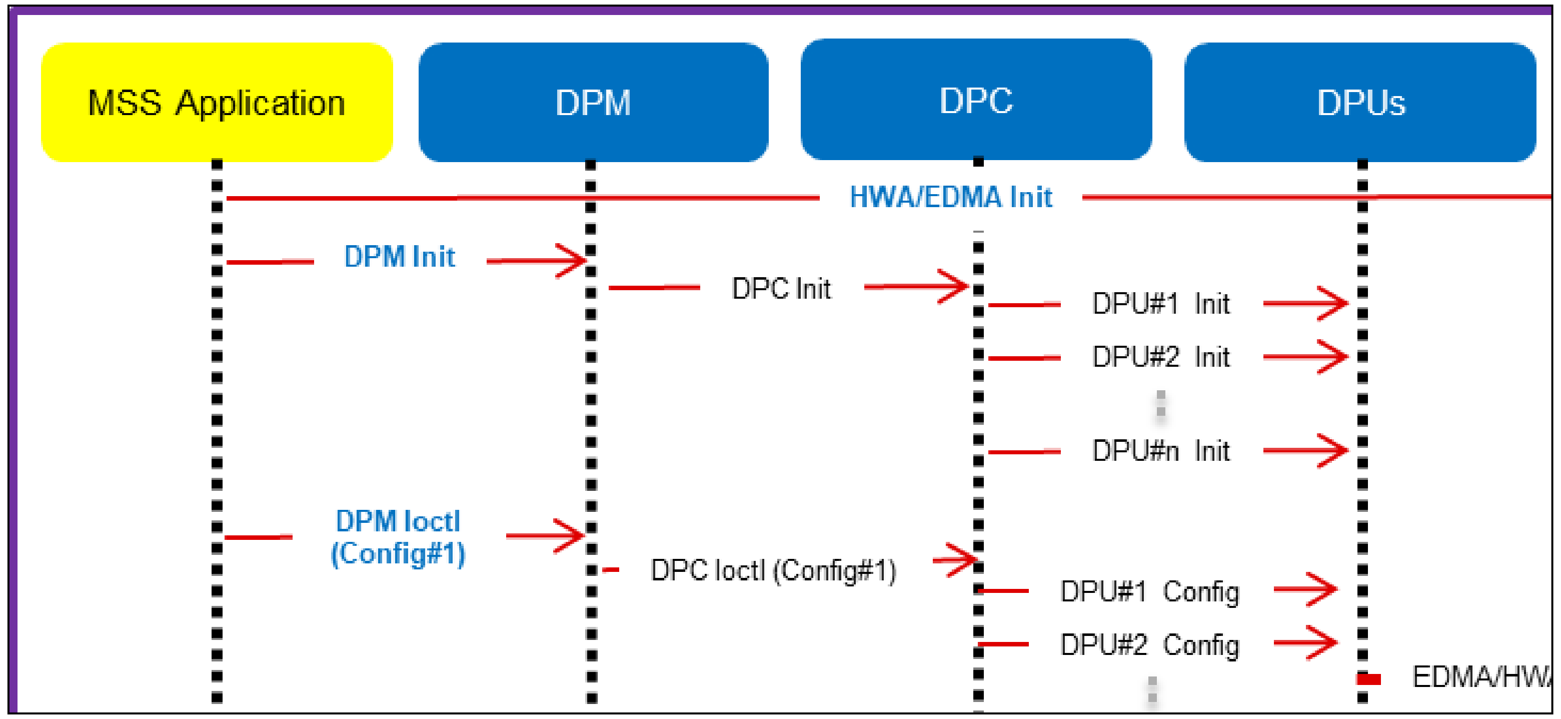

objectdetection.c (C:\ti\mmwave_sdk_03_04_00_03\packages\ti\datapath\dpc\objectdetection\objdethwa\src) - GVIM
File Edit Tools Syntax Buffers Window Help

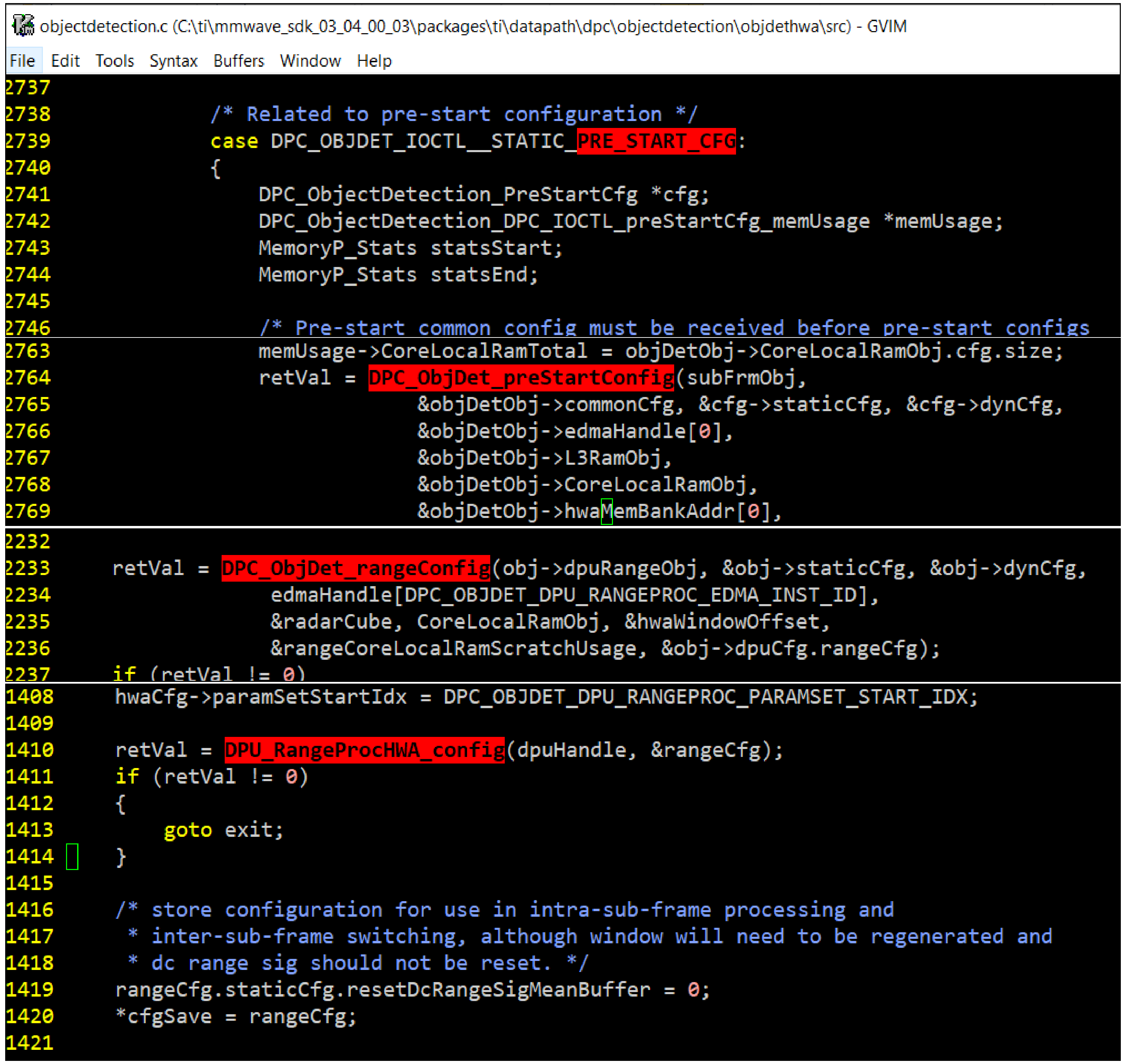
```
 *       Error    -   <0
 */
static DPM_DPCHandle DPC_ObjectDetection_init
(
    DPM_Handle          dpmHandle,
    DPM_InitCfg*        ptrInitCfg,
    int32_t*            errCode
)
{
    int32_t i;
    ObjDetObj     *objDetObj = NULL;
    SubFrameObj   *subFrmObj;
    DPC_ObjectDetection_InitParams *dpcInitParams;
    DPU_RangeProcHWA_InitParams rangeInitParams;
    DPU_AoAProcHWA_InitParams aoaInitParams;
    DPU_CFARCAProcHWA_InitParams cfarInitParams;
    DPU_DopplerProcHWA_InitParams dopplerInitParams;
    HWA_MemInfo         hwaMemInfo;
    rangeInitParams.hwaHandle = dpcInitParams->hwaHandle;
    aoaInitParams.hwaHandle = dpcInitParams->hwaHandle;
    cfarInitParams.hwaHandle = dpcInitParams->hwaHandle;
    dopplerInitParams.hwaHandle = dpcInitParams->hwaHandle;

    for(i = 0; i < RL_MAX_SUBFRAMES; i++)
    {
        subFrmObj = &objDetObj->subFrameObj[i];

        subFrmObj->dpuRangeObj = DPU_RangeProcHWA_init(&rangeInitParams, errCode);

        if (*errCode != 0)
        {
            goto exit;
        }

        subFrmObj->dpuStaticClutterObj = DPU_StaticClutterProc_init(errCode);

        if (*errCode != 0)
        {
            goto exit;
        }

        subFrmObj->dpuCFARCAObj = DPU_CFARCAProcHWA_init(&cfarInitParams, errCode);

        if (*errCode != 0)
```

DPU Configuration

Recall from DPC Configuration…

- Application calls DPM_ioctl with different message types
- DPM_ioctl internally invokes the DPC's registered ioctl function and passes the message to it
- DPC performs DPU configuration when handling the PRE_START_CFG command
  - The PRE_START_CFG handler calls wrapper functions for each DPU, e.g. DPC_ObjDet_rangeConfig
  - The wrapper allocates the resources required for the DPU and calls the corresponding DPU config function e.g. DPU_RangeProcHWA_config.
- Refer to the call flow below




objectdetection.c (C:\ti\mmwave_sdk_03_04_00_03\packages\ti\datapath\dpc\objectdetection\objdethwa\src) - GVIM
File
Edit
Tools
Syntax
Buffers
Window
Help

```
            /* Related to pre-start configuration */
            case DPC_OBJDET_IOCTL__STATIC_PRE_START_CFG:
            {
                DPC_ObjectDetection_PreStartCfg *cfg;
                DPC_ObjectDetection_DPC_IOCTL_preStartCfg_memUsage *memUsage;
                MemoryP_Stats statsStart;
                MemoryP_Stats statsEnd;

                /* Pre-start common config must be received before pre-start configs
                memUsage->CoreLocalRamTotal = objDetObj->CoreLocalRamObj.cfg.size;
                retVal = DPC_ObjDet_preStartConfig(subFrmObj,
                             &objDetObj->commonCfg, &cfg->staticCfg, &cfg->dynCfg,
                             &objDetObj->edmaHandle[0],
                             &objDetObj->L3RamObj,
                             &objDetObj->CoreLocalRamObj,
                             &objDetObj->hwaMemBankAddr[0],
```

```

    retVal = DPC_ObjDet_rangeConfig(obj->dpuRangeObj, &obj->staticCfg, &obj->dynCfg,
                 edmaHandle[DPC_OBJDET_DPU_RANGEPROC_EDMA_INST_ID],
                 &radarCube, CoreLocalRamObj, &hwaWindowOffset,
                 &rangeCoreLocalRamScratchUsage, &obj->dpuCfg.rangeCfg);
    if (retVal != 0)
```

```
    hwaCfg->paramSetStartIdx = DPC_OBJDET_DPU_RANGEPROC_PARAMSET_START_IDX;

    retVal = DPU_RangeProcHWA_config(dpuHandle, &rangeCfg);
    if (retVal != 0)
    {
        goto exit;
    }

    /* store configuration for use in intra-sub-frame processing and
     * inter-sub-frame switching, although window will need to be regenerated and
     * dc range sig should not be reset. */
    rangeCfg.staticCfg.resetDcRangeSigMeanBuffer = 0;
    *cfgSave = rangeCfg;

```

DPIF: Datapath Interface

Input ADC data

- Property
  - numADCSamples
  - RX interleaved/non-interleaved
  - Complex/Real
- Buffer Pointer

Radar Cube

- Property
  - Layout – RADAR CUBE RANGE DOPPLER RX TX (1)
- Buffer Pointer

Detection Matrix

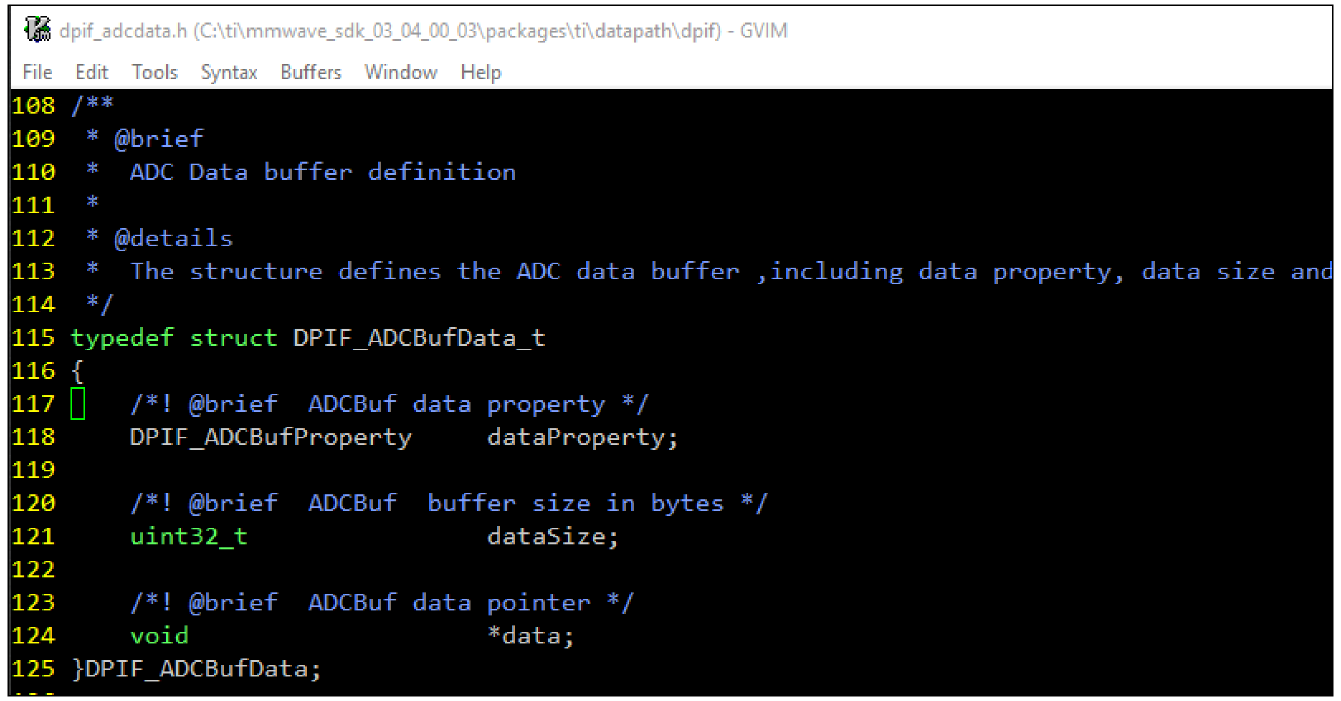
- Property
  - Layout – DET MATRIX RANGE DOPPLER (1)
- Buffer Pointer

Point cloud

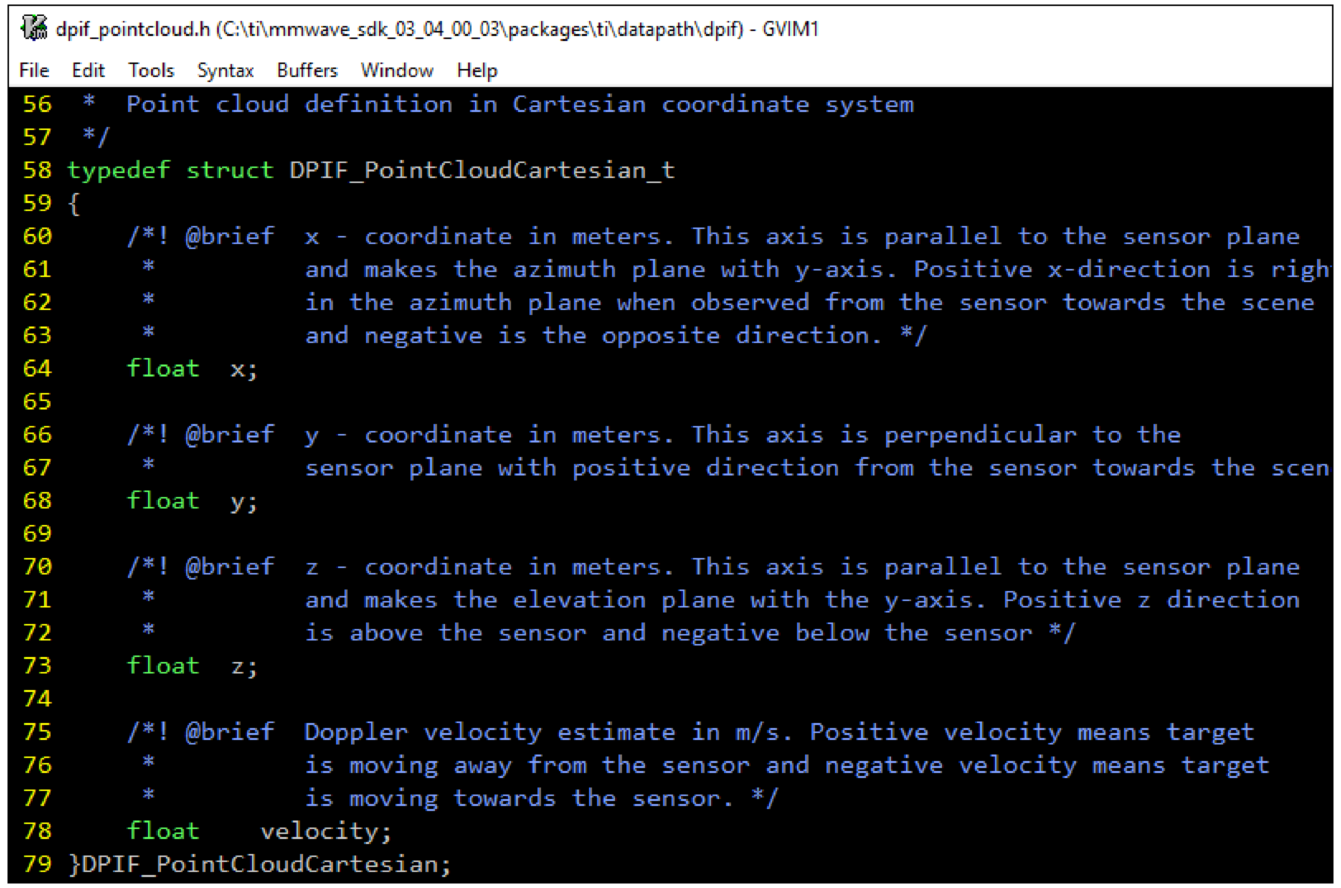
- Property
  - Format – XYZV, RAEV
  - Float

Point Cloud SideInfo

- Property
  - snr
  - noiseVal


dpif_adcdata.h (C:\ti\mmwave_sdk_03_04_00_03\packages\ti\datapath\dpif) - GVIM
File Edit Tools Syntax Buffers Window Help

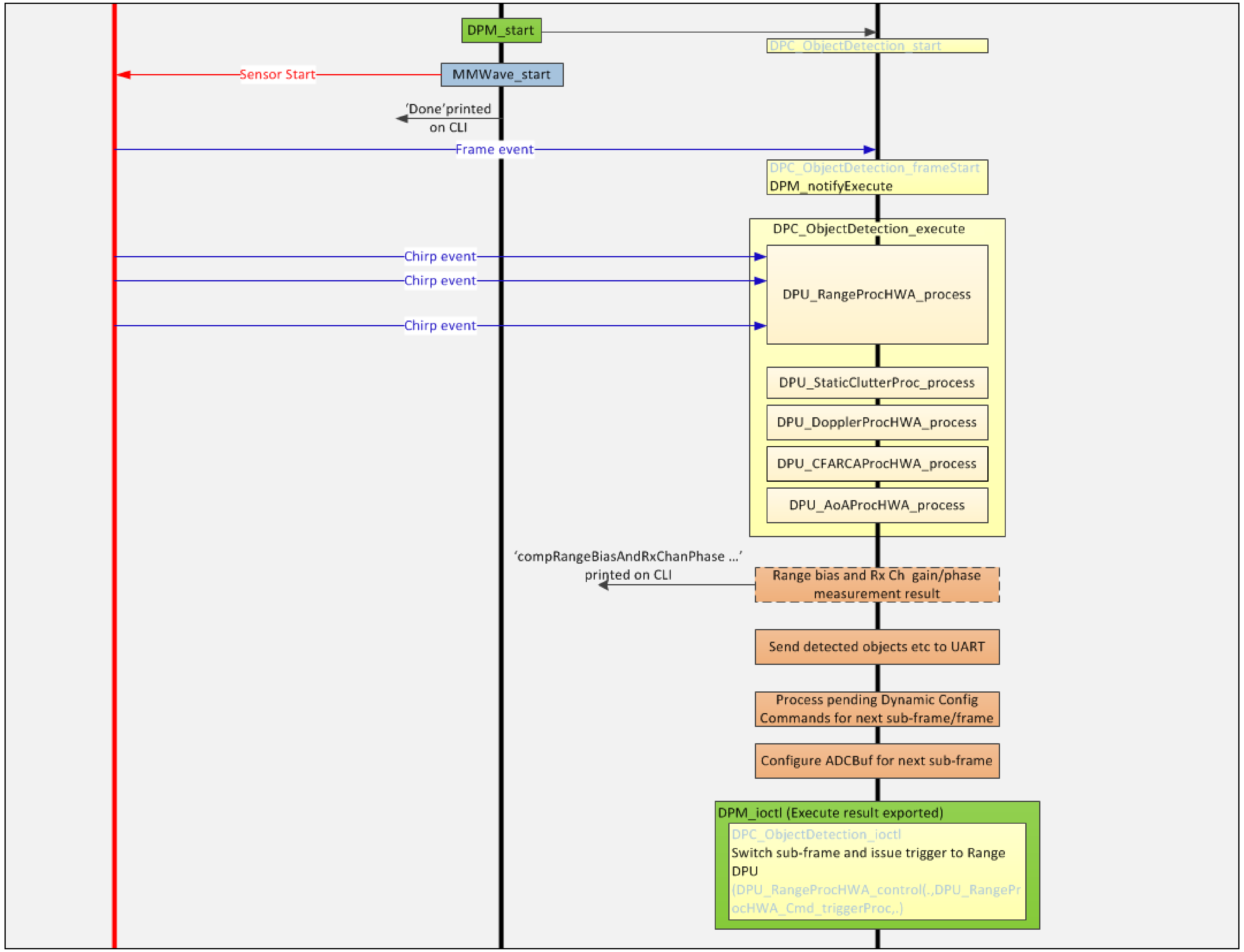
```
/**
 * @brief
 *  ADC Data buffer definition
 *
 * @details
 *  The structure defines the ADC data buffer ,including data property, data size and
 */
typedef struct DPIF_ADCBufData_t
{
    /*! @brief  ADCBuf data property */
    DPIF_ADCBufProperty     dataProperty;

    /*! @brief  ADCBuf  buffer size in bytes */
    uint32_t                dataSize;

    /*! @brief  ADCBuf data pointer */
    void                    *data;
}DPIF_ADCBufData;
```


dpif_pointcloud.h (C:\ti\mmwave_sdk_03_04_00_03\packages\ti\datapath\dpif) - GVIM1
File Edit Tools Syntax Buffers Window Help

```
 *  Point cloud definition in Cartesian coordinate system
 */
typedef struct DPIF_PointCloudCartesian_t
{
    /*! @brief  x - coordinate in meters. This axis is parallel to the sensor plane
     *          and makes the azimuth plane with y-axis. Positive x-direction is righ
     *          in the azimuth plane when observed from the sensor towards the scene
     *          and negative is the opposite direction. */
    float  x;

    /*! @brief  y - coordinate in meters. This axis is perpendicular to the
     *          sensor plane with positive direction from the sensor towards the scen
    float  y;

    /*! @brief  z - coordinate in meters. This axis is parallel to the sensor plane
     *          and makes the elevation plane with the y-axis. Positive z direction
     *          is above the sensor and negative below the sensor */
    float  z;

    /*! @brief  Doppler velocity estimate in m/s. Positive velocity means target
     *          is moving away from the sensor and negative velocity means target
     *          is moving towards the sensor. */
    float    velocity;
}DPIF_PointCloudCartesian;
```

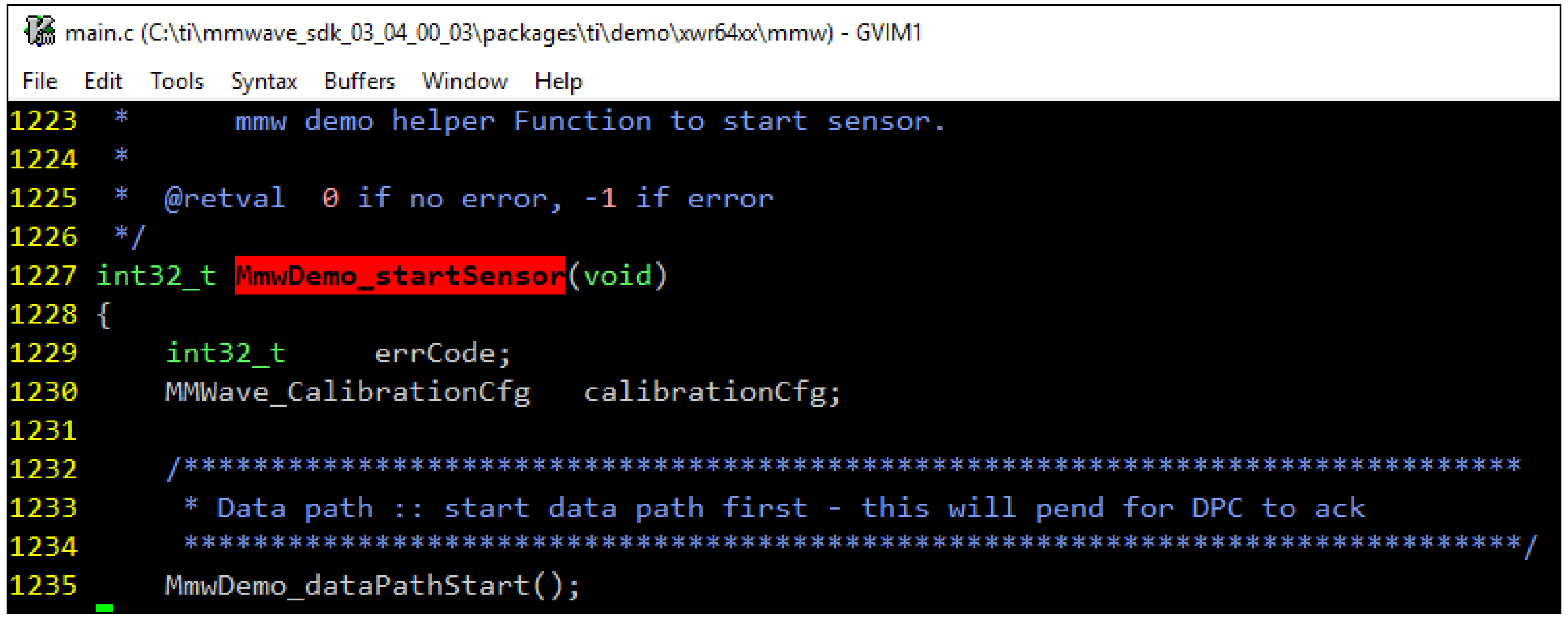
Datapath Execution – Runtime View

- **At sensor start:** Application calls DPM_start which calls the DPC's pre-registered start function DPC_ObjectDetection_start.
  - The mmw demo does this in the sensorStart CLI command handler function.
- **At every frame interrupt:** The DPC's pre-registered frame start call back function, DPC_ObjectDetection_frameStart is invoked.
  - **Recall from DPC_init:** The DPC's frameStart callback is registered with the DPM during DPC initialization
- This invokes the DPC's registered execute function, e.g. DPC_ObjectDetection_execute
  - **Under the hood:** The Frame Start handler calls DPM_notifyExecute which posts the semaphore for DPM_execute
  - DPM_execute then calls the DPC's pre-registered execute function.
- DPC_ObjectDetection_execute performs the frame processing and populates the output in DPC_ObjectDetection_ExecuteResult structure and returns.
- Application reads the output structure and sends it over UART. This completes the frame processing.

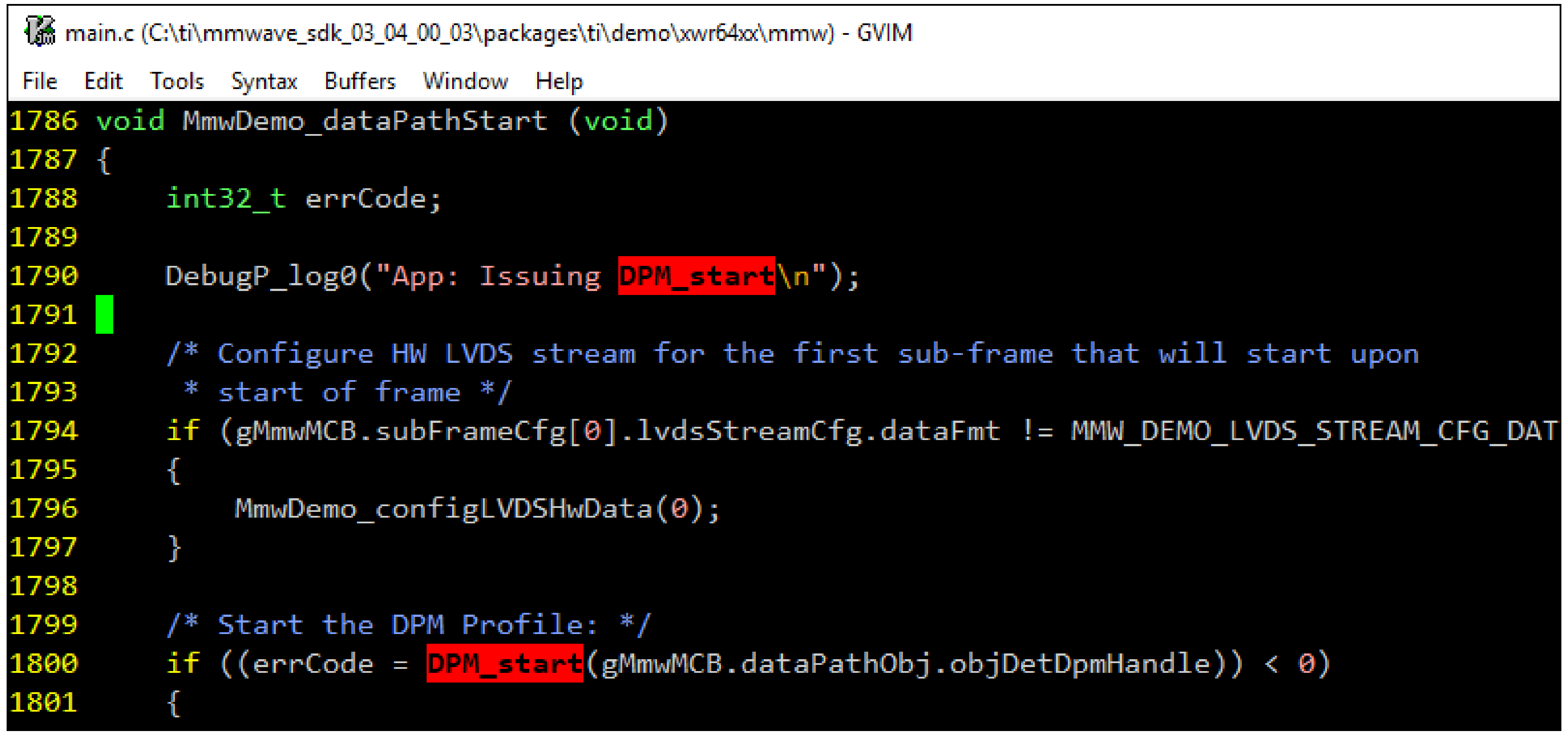

DPM_start
DPC_ObjectDetection_start
Sensor Start
MMWave_start
'Done'printed on CLI
Frame event
DPC_ObjectDetection_frameStart
DPM_notifyExecute
DPC_ObjectDetection_execute
Chirp event
Chirp event
Chirp event
DPU_RangeProcHWA_process
DPU_StaticClutterProc_process
DPU_DopplerProcHWA_process
DPU_CFARCAProcHWA_process
DPU_AoAProcHWA_process
'compRangeBiasAndRxChanPhase ...' printed on CLI
Range bias and Rx Ch gain/phase measurement result
Send detected objects etc to UART
Process pending Dynamic Config Commands for next sub-frame/frame
Configure ADCBuf for next sub-frame
DPM_ioctl (Execute result exported)
DPC_ObjectDetection_ioctl
Switch sub-frame and issue trigger to Range DPU
(DPU_RangeProcHWA_control(.,DPU_RangeProcHWA_Cmd_triggerProc,.)

Datapath Execution – Runtime View

- **At sensor start:** Application calls DPM_start which calls the DPC's pre-registered start function DPC_ObjectDetection_start.
  - The mmw demo does this in the sensorStart CLI command handler function.
- **At every frame interrupt:** The DPC's pre-registered frame start call back function, DPC_ObjectDetection_frameStart is invoked.
  - **Recall from DPC_init:** The DPC's frameStart callback is registered with the DPM during DPC initialization
- This invokes the DPC's registered execute function, e.g. DPC_ObjectDetection_execute
  - **Under the hood:** The Frame Start handler calls DPM_notifyExecute which posts the semaphore for DPM_execute
  - DPM_execute then calls the DPC's pre-registered execute function.
- DPC_ObjectDetection_execute performs the frame processing and populates the output in DPC_ObjectDetection_ExecuteResult structure and returns.
- Application reads the output structure and sends it over UART. This completes the frame processing.


main.c (C:\ti\mmwave_sdk_03_04_00_03\packages\ti\demo\xwr64xx\mmw) - GVIM1
File Edit Tools Syntax Buffers Window Help

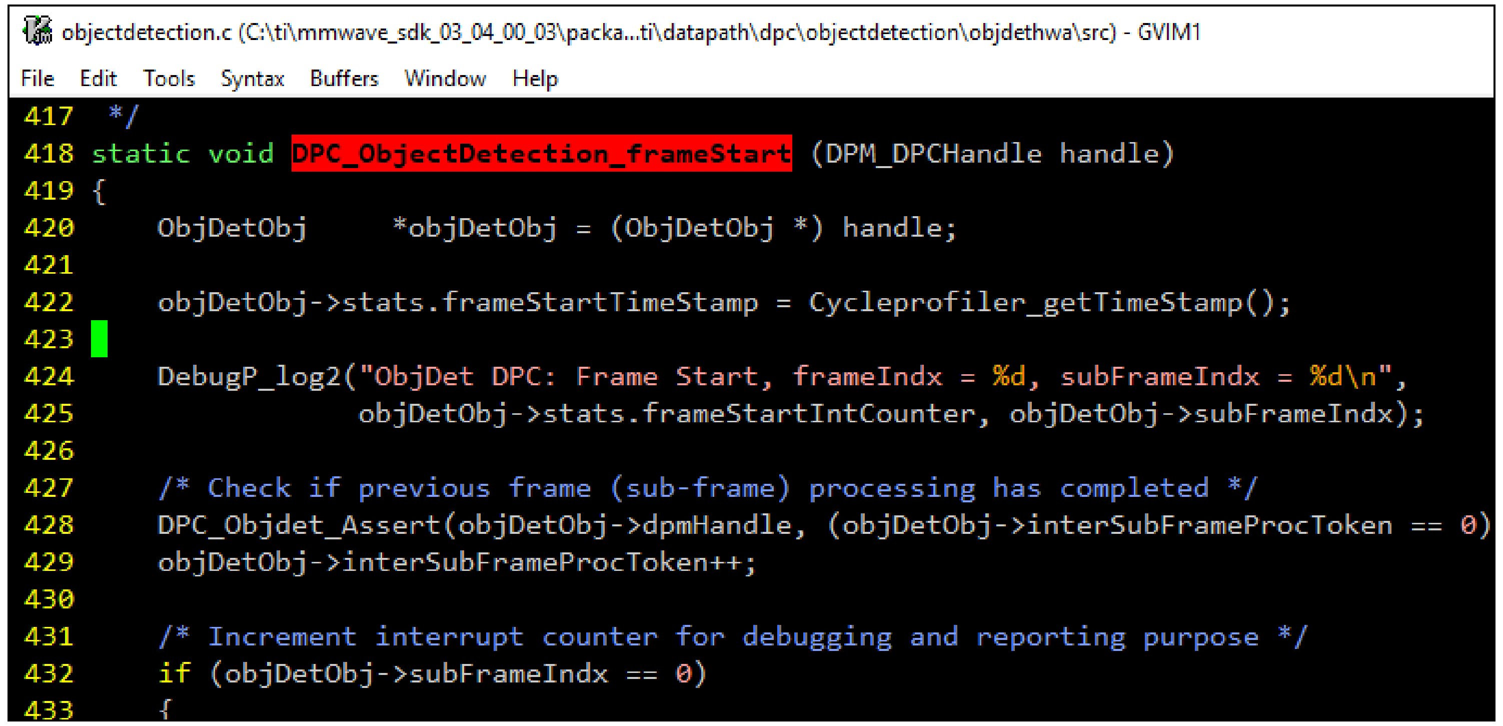
```
 *      mmw demo helper Function to start sensor.
 *
 *  @retval   0 if no error, -1 if error
 */
int32_t MmwDemo_startSensor(void)
{
    int32_t     errCode;
    MMWave_CalibrationCfg   calibrationCfg;

    /****************************************************************************
     * Data path :: start data path first - this will pend for DPC to ack
     ***************************************************************************/
    MmwDemo_dataPathStart();
```


main.c (C:\ti\mmwave_sdk_03_04_00_03\packages\ti\demo\xwr64xx\mmw) - GVIM
File Edit Tools Syntax Buffers Window Help

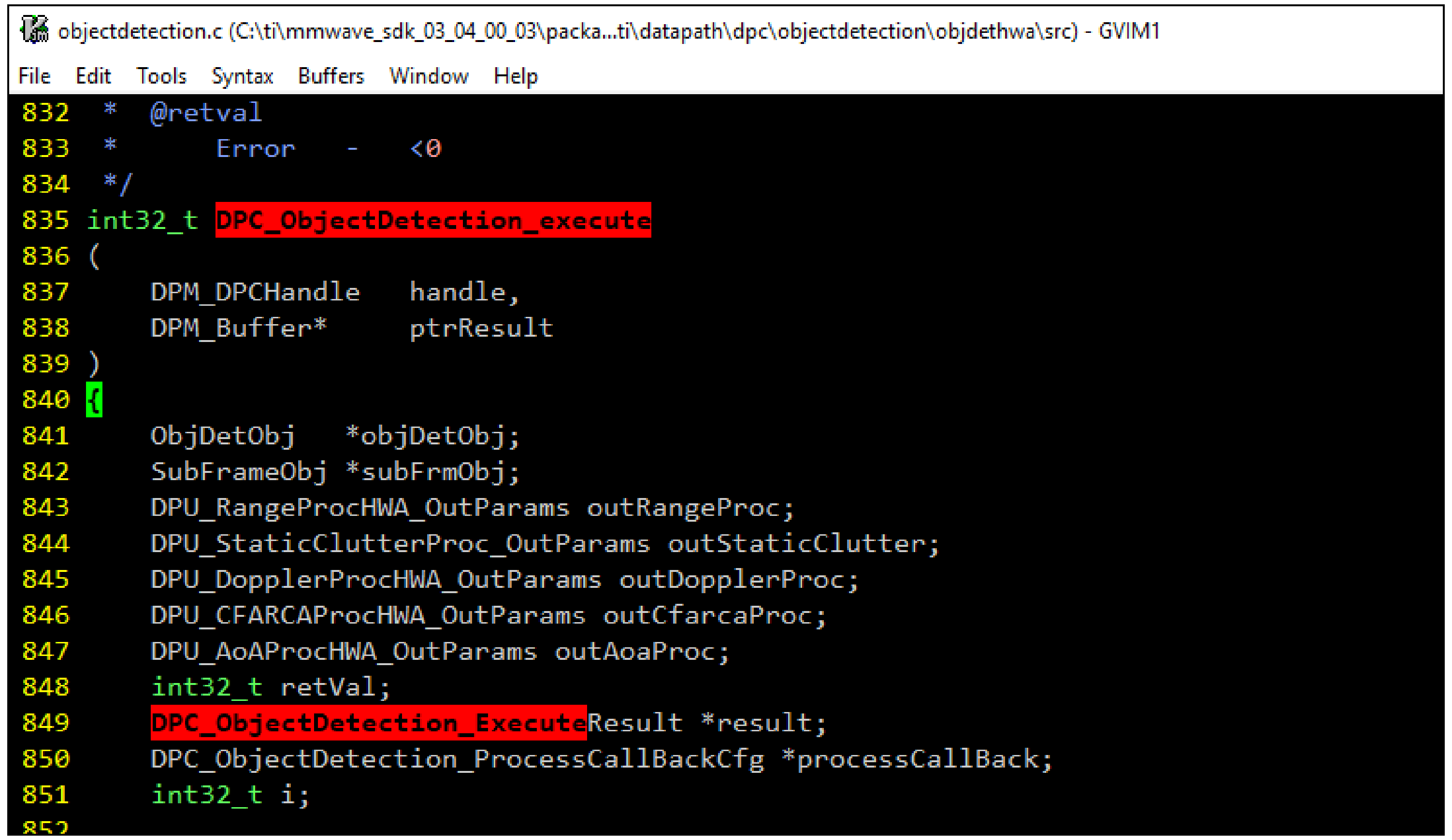
```
void MmwDemo_dataPathStart (void)
{
    int32_t errCode;

    DebugP_log0("App: Issuing DPM_start\n");

    /* Configure HW LVDS stream for the first sub-frame that will start upon
     * start of frame */
    if (gMmwMCB.subFrameCfg[0].lvdsStreamCfg.dataFmt != MMW_DEMO_LVDS_STREAM_CFG_DAT
    {
        MmwDemo_configLVDSHwData(0);
    }

    /* Start the DPM Profile: */
    if ((errCode = DPM_start(gMmwMCB.dataPathObj.objDetDpmHandle)) < 0)
    {
```

Datapath Execution – Runtime View

- **At sensor start:** Application calls DPM_start which calls the DPC's pre-registered start function DPC_ObjectDetection_start.
    - The mmw demo does this in the sensorStart CLI command handler function.
- **At every frame interrupt:** The DPC's pre-registered frame start call back function, DPC_ObjectDetection_frameStart is invoked.
    - **Recall from DPC_init:** The DPC's frameStart callback is registered with the DPM during DPC initialization
- This invokes the DPC's registered execute function, e.g. DPC_ObjectDetection_execute
    - **Under the hood:** The Frame Start handler calls DPM_notifyExecute which posts the semaphore for DPM_execute
    - DPM_execute then calls the DPC's pre-registered execute function.
- DPC_ObjectDetection_execute performs the frame processing and populates the output in DPC_ObjectDetection_ExecuteResult structure and returns.
- Application reads the output structure and sends it over UART. This completes the frame processing.


objectdetection.c (C:\ti\mmwave_sdk_03_04_00_03\packa...ti\datapath\dpc\objectdetection\objdethwa\src) - GVIM1
File
Edit
Tools
Syntax
Buffers
Window
Help

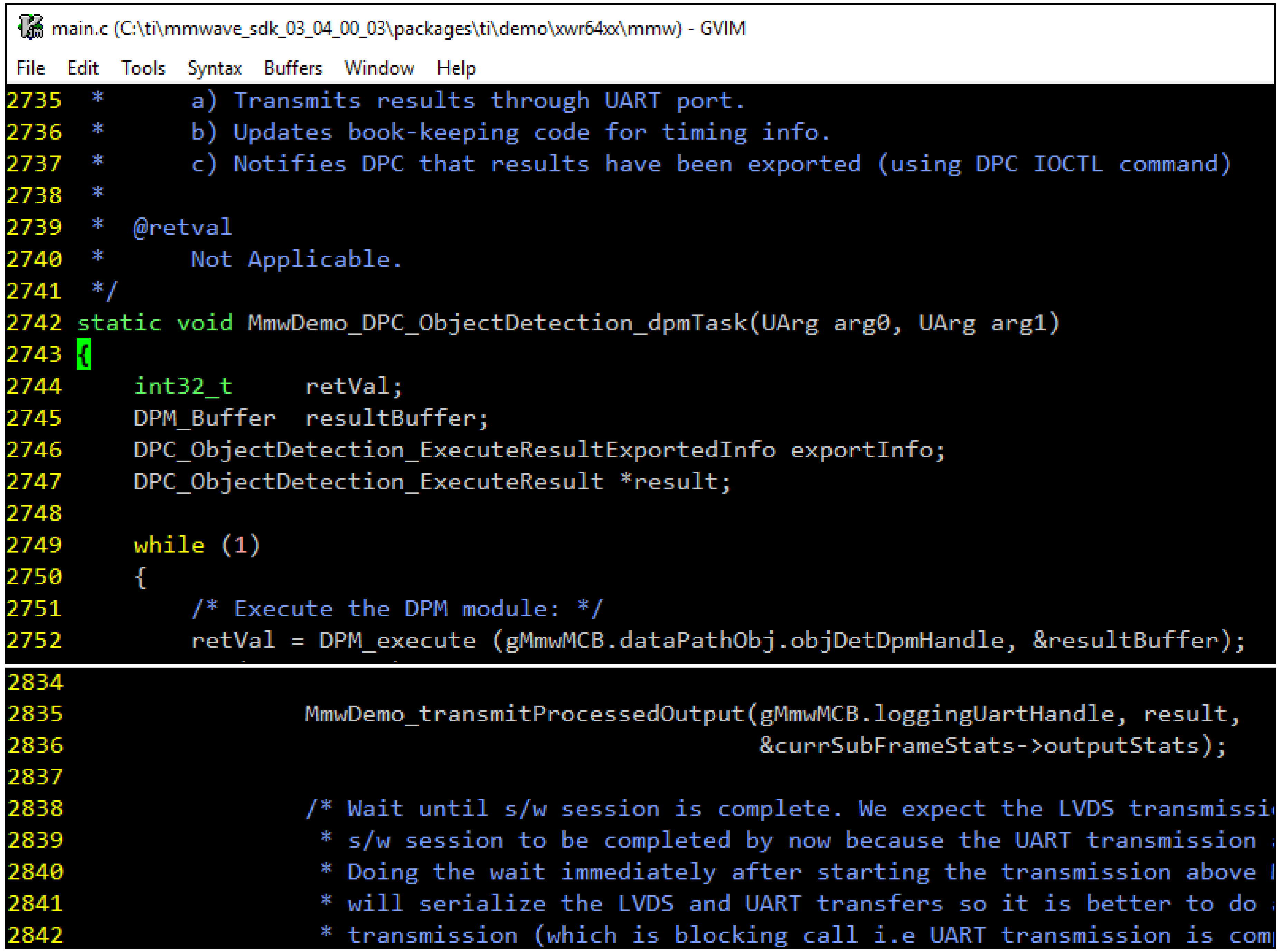
```c
 */
static void DPC_ObjectDetection_frameStart (DPM_DPCHandle handle)
{
    ObjDetObj     *objDetObj = (ObjDetObj *) handle;

    objDetObj->stats.frameStartTimeStamp = Cycleprofiler_getTimeStamp();

    DebugP_log2("ObjDet DPC: Frame Start, frameIndx = %d, subFrameIndx = %d\n",
                objDetObj->stats.frameStartIntCounter, objDetObj->subFrameIndx);

    /* Check if previous frame (sub-frame) processing has completed */
    DPC_Objdet_Assert(objDetObj->dpmHandle, (objDetObj->interSubFrameProcToken == 0)
    objDetObj->interSubFrameProcToken++;

    /* Increment interrupt counter for debugging and reporting purpose */
    if (objDetObj->subFrameIndx == 0)
    {
```


objectdetection.c (C:\ti\mmwave_sdk_03_04_00_03\packa...ti\datapath\dpc\objectdetection\objdethwa\src) - GVIM1
File
Edit
Tools
Syntax
Buffers
Window
Help

```c
 *  @retval
 *      Error    -   <0
 */
int32_t DPC_ObjectDetection_execute
(
    DPM_DPCHandle   handle,
    DPM_Buffer*     ptrResult
)
{
    ObjDetObj    *objDetObj;
    SubFrameObj *subFrmObj;
    DPU_RangeProcHWA_OutParams outRangeProc;
    DPU_StaticClutterProc_OutParams outStaticClutter;
    DPU_DopplerProcHWA_OutParams outDopplerProc;
    DPU_CFARCAProcHWA_OutParams outCfarcaProc;
    DPU_AoAProcHWA_OutParams outAoaProc;
    int32_t retVal;
    DPC_ObjectDetection_ExecuteResult *result;
    DPC_ObjectDetection_ProcessCallBackCfg *processCallBack;
    int32_t i;
```

Datapath Execution – Runtime View

- **At sensor start:** Application calls DPM_start which calls the DPC's pre-registered start function DPC_ObjectDetection_start.
    - The mmw demo does this in the sensorStart CLI command handler function.
- **At every frame interrupt:** The DPC's pre-registered frame start call back function, DPC_ObjectDetection_frameStart is invoked.
    - **Recall from DPC_init:** The DPC's frameStart callback is registered with the DPM during DPC initialization
- This invokes the DPC's registered execute function, e.g. DPC_ObjectDetection_execute
    - **Under the hood:** The Frame Start handler calls DPM_notifyExecute which posts the semaphore for DPM_execute
    - DPM_execute then calls the DPC's pre-registered execute function.
- DPC_ObjectDetection_execute performs the frame processing and populates the output in DPC_ObjectDetection_ExecuteResult structure and returns.
- Application reads the output structure and sends it over UART. This completes the frame processing.

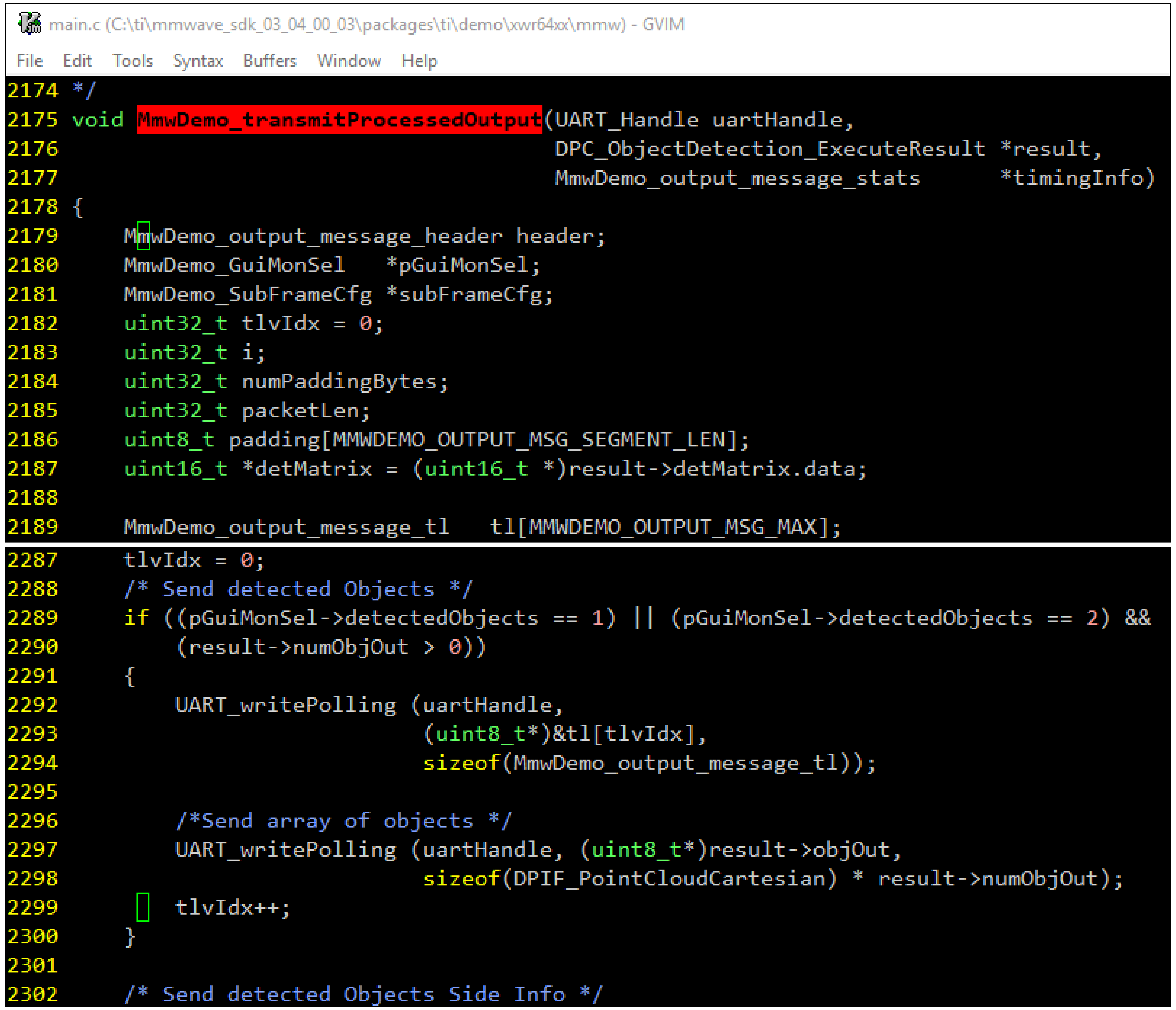

main.c (C:\ti\mmwave_sdk_03_04_00_03\packages\ti\demo\xwr64xx\mmw) - GVIM
File
Edit
Tools
Syntax
Buffers
Window
Help

```
 *      a) Transmits results through UART port.
 *      b) Updates book-keeping code for timing info.
 *      c) Notifies DPC that results have been exported (using DPC IOCTL command)
 *
 *  @retval
 *      Not Applicable.
 */
static void MmwDemo_DPC_ObjectDetection_dpmTask(UArg arg0, UArg arg1)
{
    int32_t     retVal;
    DPM_Buffer  resultBuffer;
    DPC_ObjectDetection_ExecuteResultExportedInfo exportInfo;
    DPC_ObjectDetection_ExecuteResult *result;

    while (1)
    {
        /* Execute the DPM module: */
        retVal = DPM_execute (gMmwMCB.dataPathObj.objDetDpmHandle, &resultBuffer);
```

```

                MmwDemo_transmitProcessedOutput(gMmwMCB.loggingUartHandle, result,
                                                &currSubFrameStats->outputStats);

                /* Wait until s/w session is complete. We expect the LVDS transmissi
                 * s/w session to be completed by now because the UART transmission 
                 * Doing the wait immediately after starting the transmission above 
                 * will serialize the LVDS and UART transfers so it is better to do 
                 * transmission (which is blocking call i.e UART transmission is com
```

Datapath Execution – Runtime View

- **At sensor start:** Application calls DPM_start which calls the DPC's pre-registered start function DPC_ObjectDetection_start.
  - The mmw demo does this in the sensorStart CLI command handler function.
- **At every frame interrupt:** The DPC's pre-registered frame start call back function, DPC_ObjectDetection_frameStart is invoked.
  - **Recall from DPC_init:** The DPC's frameStart callback is registered with the DPM during DPC initialization
- This invokes the DPC's registered execute function, e.g. DPC_ObjectDetection_execute
  - **Under the hood:** The Frame Start handler calls DPM_notifyExecute which posts the semaphore for DPM_execute
  - DPM_execute then calls the DPC's pre-registered execute function.
- DPC_ObjectDetection_execute performs the frame processing and populates the output in DPC_ObjectDetection_ExecuteResult structure and returns.
- Application reads the output structure and sends it over UART. This completes the frame processing.


main.c (C:\ti\mmwave_sdk_03_04_00_03\packages\ti\demo\xwr64xx\mmw) - GVIM
File
Edit
Tools
Syntax
Buffers
Window
Help

```
*/
void MmwDemo_transmitProcessedOutput(UART_Handle uartHandle,
                                     DPC_ObjectDetection_ExecuteResult *result,
                                     MmwDemo_output_message_stats      *timingInfo)
{
    MmwDemo_output_message_header header;
    MmwDemo_GuiMonSel   *pGuiMonSel;
    MmwDemo_SubFrameCfg *subFrameCfg;
    uint32_t tlvIdx = 0;
    uint32_t i;
    uint32_t numPaddingBytes;
    uint32_t packetLen;
    uint8_t padding[MMWDEMO_OUTPUT_MSG_SEGMENT_LEN];
    uint16_t *detMatrix = (uint16_t *)result->detMatrix.data;

    MmwDemo_output_message_tl   tl[MMWDEMO_OUTPUT_MSG_MAX];
```

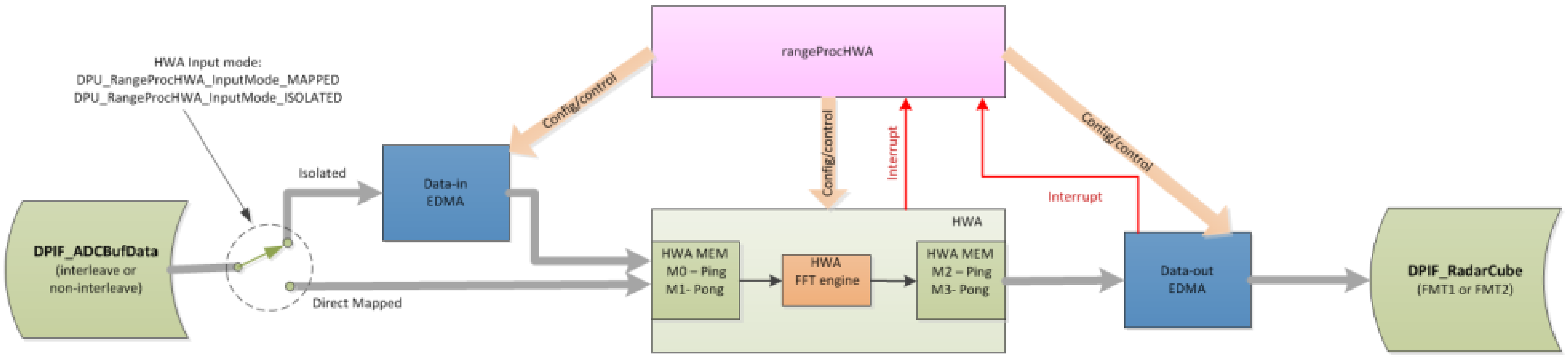
```
    tlvIdx = 0;
    /* Send detected Objects */
    if ((pGuiMonSel->detectedObjects == 1) || (pGuiMonSel->detectedObjects == 2) &&
        (result->numObjOut > 0))
    {
        UART_writePolling (uartHandle,
                           (uint8_t*)&tl[tlvIdx],
                           sizeof(MmwDemo_output_message_tl));

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

    /* Send detected Objects Side Info */
```

DPUs and DPCs in SDK 3.x

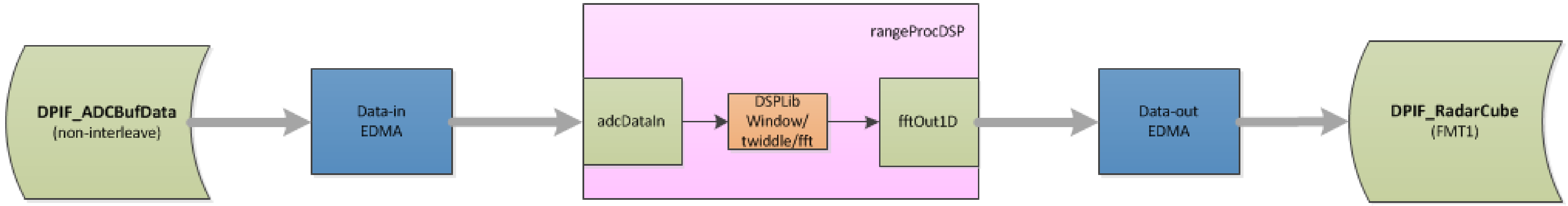
RangeProc DPU

- Purpose: (1D FFT+ DC Range Calib) processing during active frame.
  - Takes ADCBuf as input
    - interleaved or nonInterleaved format
    - Single chirp or multichirp (DSP mode only)
    - Either direct access (HWA mode only) or via EDMA
  - Produces RadarCube in L3 in user requested format (fixed set of formats described in data interface).
  - Performs FFT using HWA or DSP based on configuration.
  - Performs DC range calibration either inline (DSP mode) or at the end of all chirps (HWA mode).

- Supported architecture
  - R4F, C674x
  - Different files for HWA based and S/W based implementation



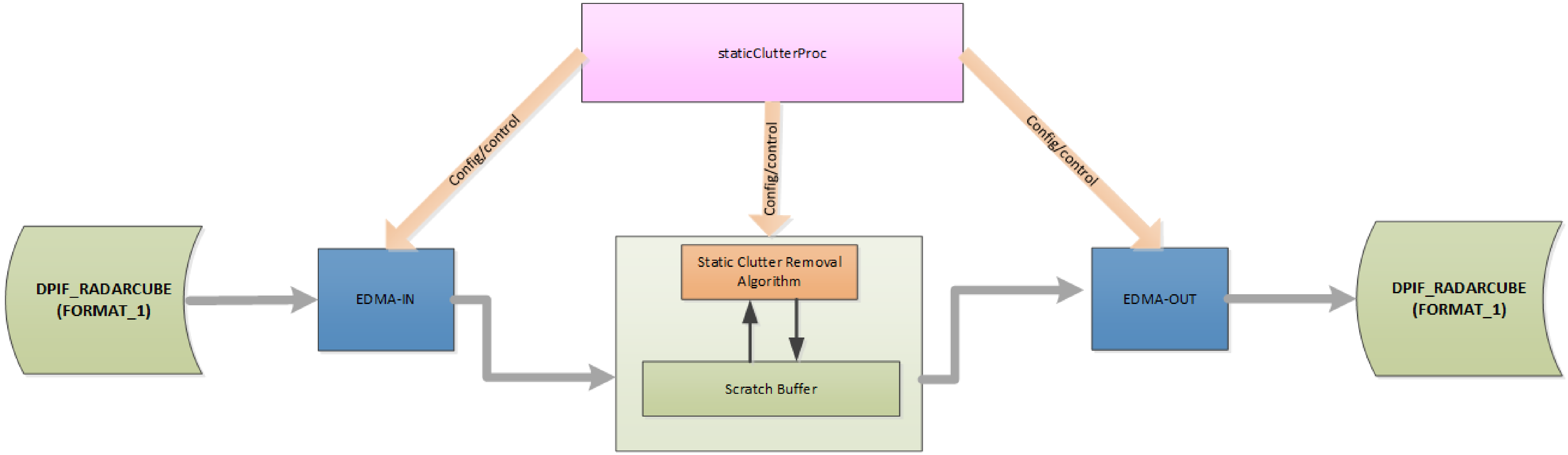
HWA Input mode:
DPU_RangeProcHWA_InputMode_MAPPED
DPU_RangeProcHWA_InputMode_ISOLATED
rangeProcHWA
Config/control
Config/control
Interrupt
Interrupt
Config/control
Isolated
Data-in EDMA
DPIF_ADCBufData (interleave or non-interleave)
Direct Mapped
HWA
HWA MEM M0 – Ping M1- Pong
HWA FFT engine
HWA MEM M2 – Ping M3- Pong
Data-out EDMA
DPIF_RadarCube (FMT1 or FMT2)



Static Clutter DPU

- Purpose: (Clutter Removal) processing during inter frame.
  - Takes non-transposed formatted 1DOUT Radarcube as input
  - Updates the RadarCube in L3 (keeping the same format)
  - S/W based only

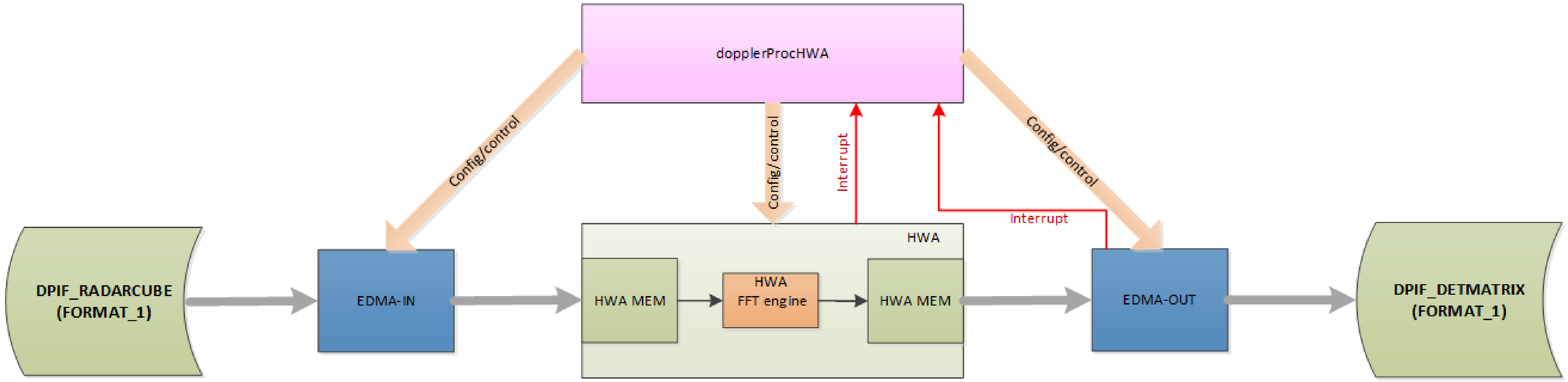
- Supported architecture
  - R4F, C674x



Doppler DPU

- Purpose: (2D FFT + Energy Sum) processing during inter frame.
  - Takes non-transposed formatted 1DOUT Radarcube as input
  - Produces Detection Matrix in L3 in fixed format
  - Performs FFT and Energy Sum using H/W(HWA)
    - S/W(DSP) based implementation would be added in the SDK in future.

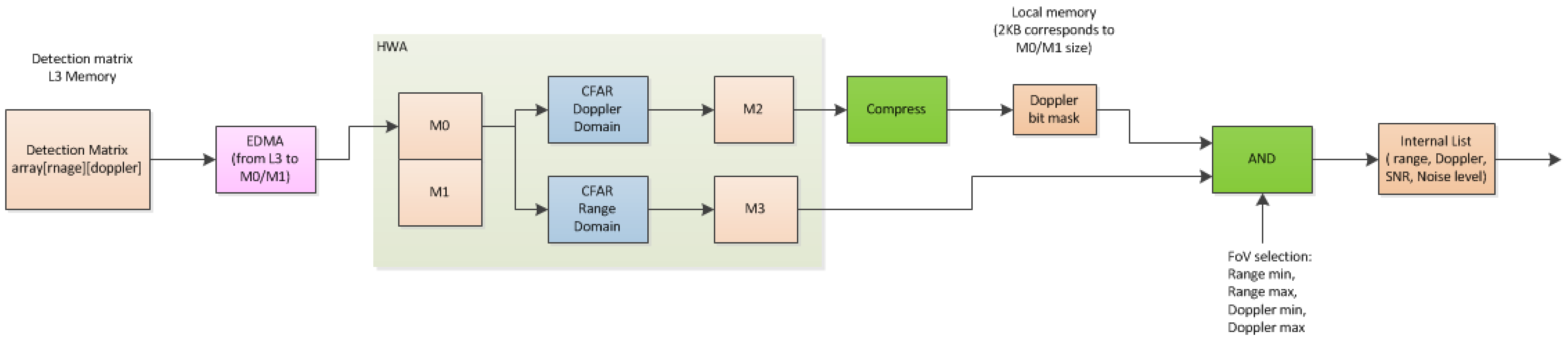
- Supported architecture
  - R4F, C674x



CFAR-CA DPU

- Purpose: (CFAR-CA +peak grouping) processing during inter frame.
  - User can choose between various CFAR-CA implementation: CFAR-CA, CFAR-CASO, CFAR-CAGO
  - Fixed Point implementation
  - 2 pass implementation: first pass (optional) in Doppler direction and then second pass in Range direction
  - Performs CFAR and PeakGrouping using H/W(HWA)
    - S/W(DSP) based implementation would be added in SDK in future
  - Takes Detection Matrix as input
  - Produces bitmask of peaks and SNR information for AoA.

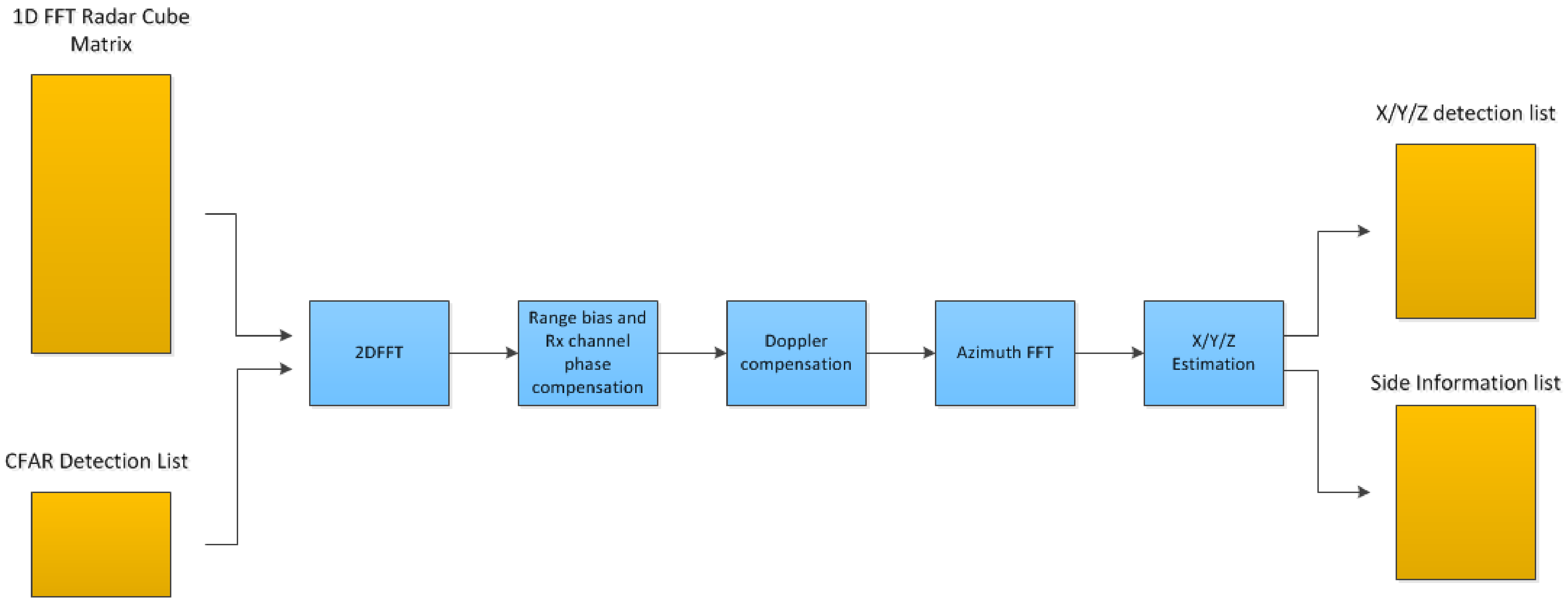
- Supported architecture
  - R4F, C674x



AoA DPU

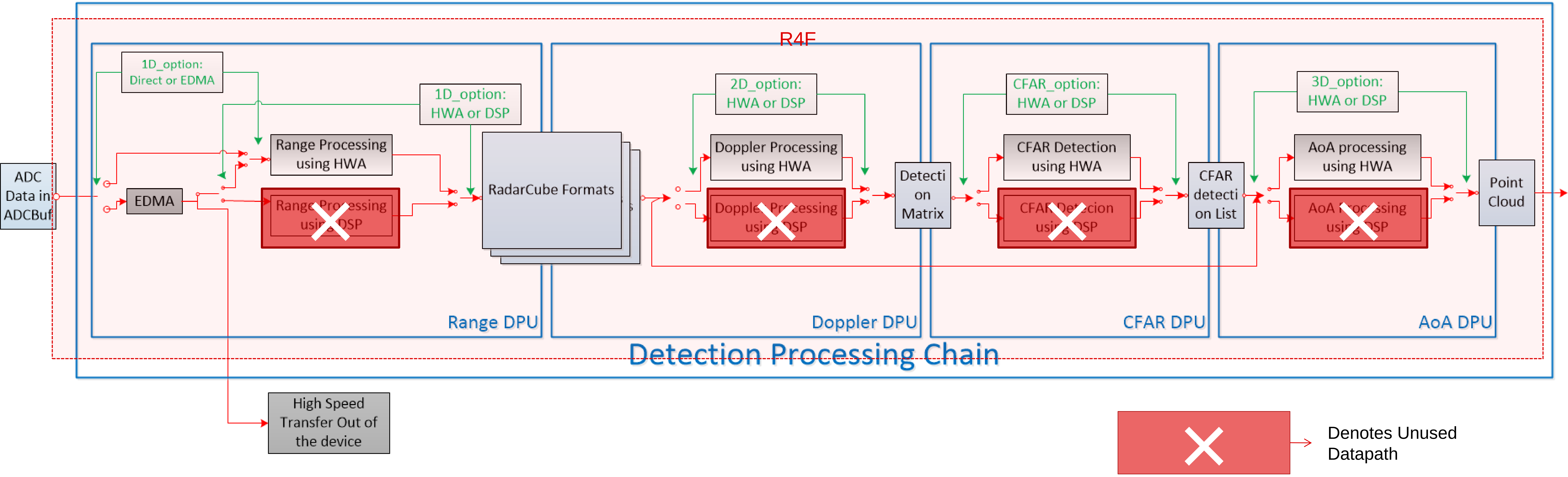
- Purpose: (Range/Phase/Doppler compensation + *Near field correction* + *Max Velocity enhancement* + AoA + FoV filter) processing during inter frame.
  - Takes non-transposed formatted 1DOUT Radarcube, detection matrix and peak bitmask as input
  - Produces Point Cloud
  - Performs any FFT ops using H/W(HWA)
    - S/W(DSP) based implementation will be added in SDK in future
  - All other processing operations are done using S/W.
    - For R4F based implementation, near field and max velocity algos will not be offered
  - All the processing ops other than AoA FFT is optional and driven by user input

- Supported architecture
  - R4F, C674x



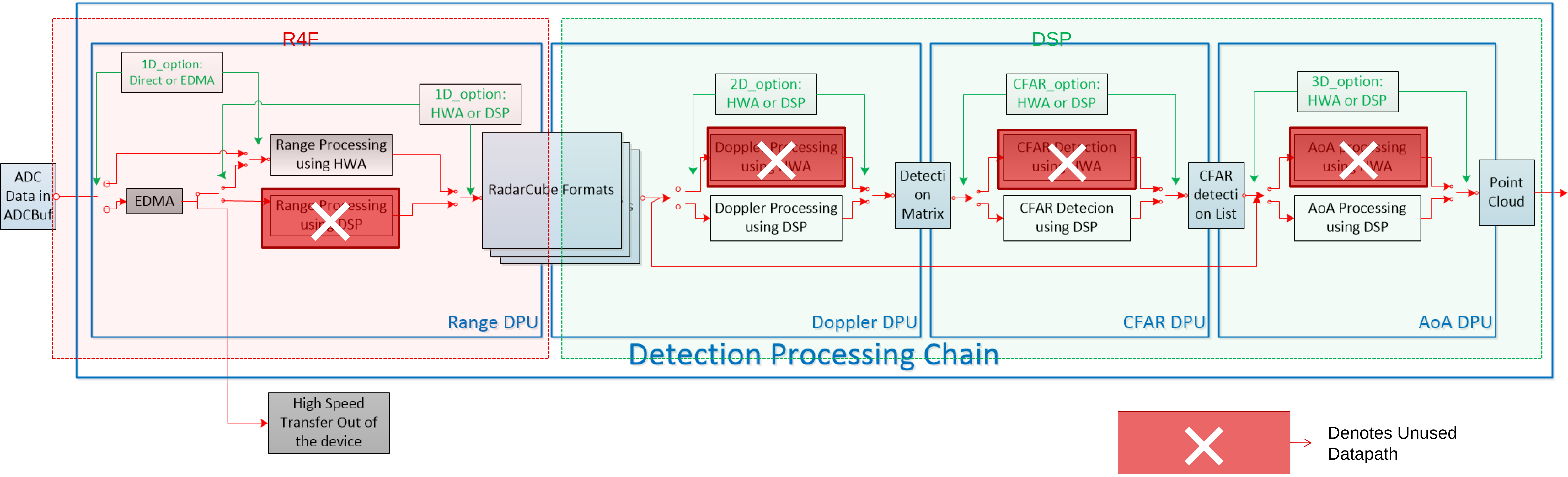
xWR64xx OOB Demo Chain (DPC)

- Detection Processing Chain (DPC) for **xWR64xx** Out of Box demo.
- Demonstrates Radar Signal Processing using Cortex R4F + HWA
- Also runs on **xWR68xx** (C674x DSP not used)

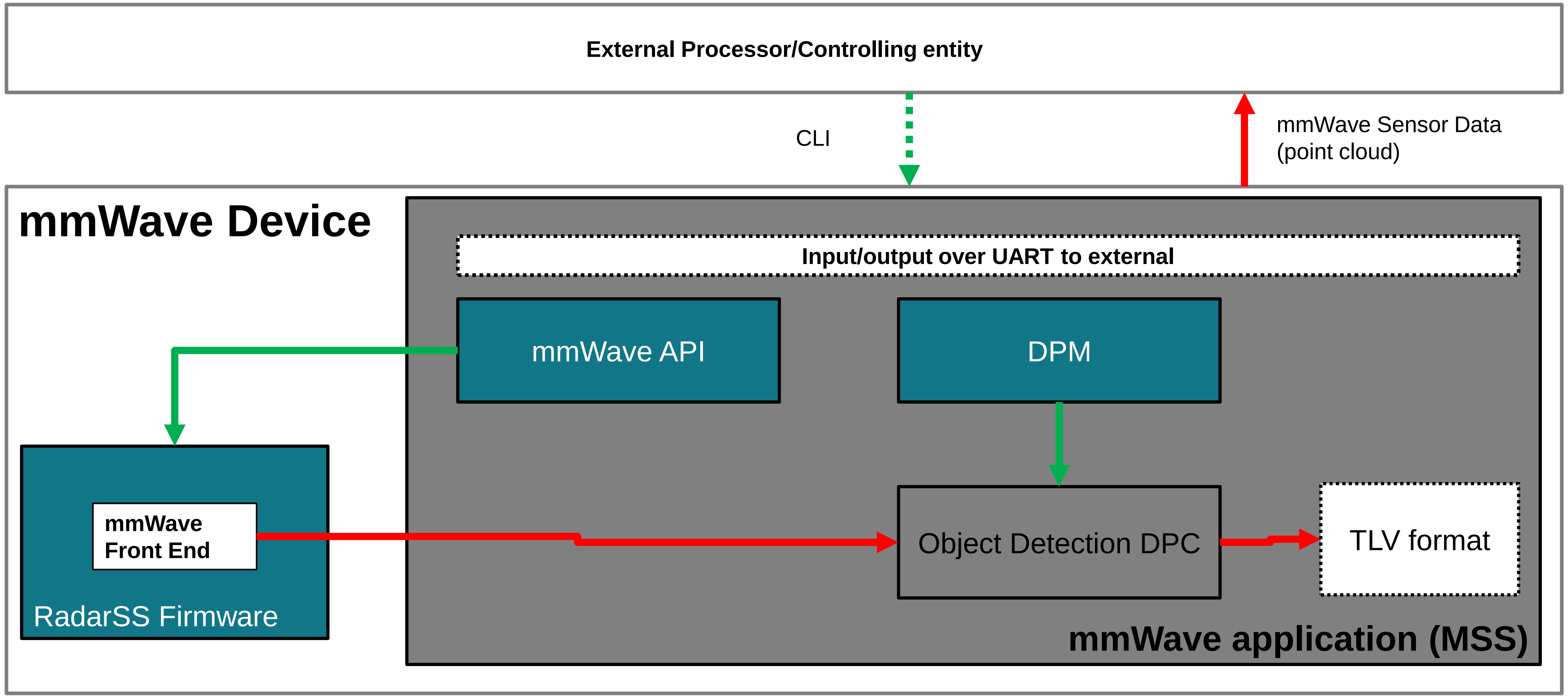


xWR68xx OOB Demo Chain (DPC)

- Detection Processing Chain (DPC) for **xWR68xx** Out of Box demo.
- Demonstrates Radar Signal Processing using Cortex R4F + HWA + C674x DSP
- DSP provides higher performance and frees up R4F for other processing (e.g. Object Tracking)



68xx OOB Demo - Application View



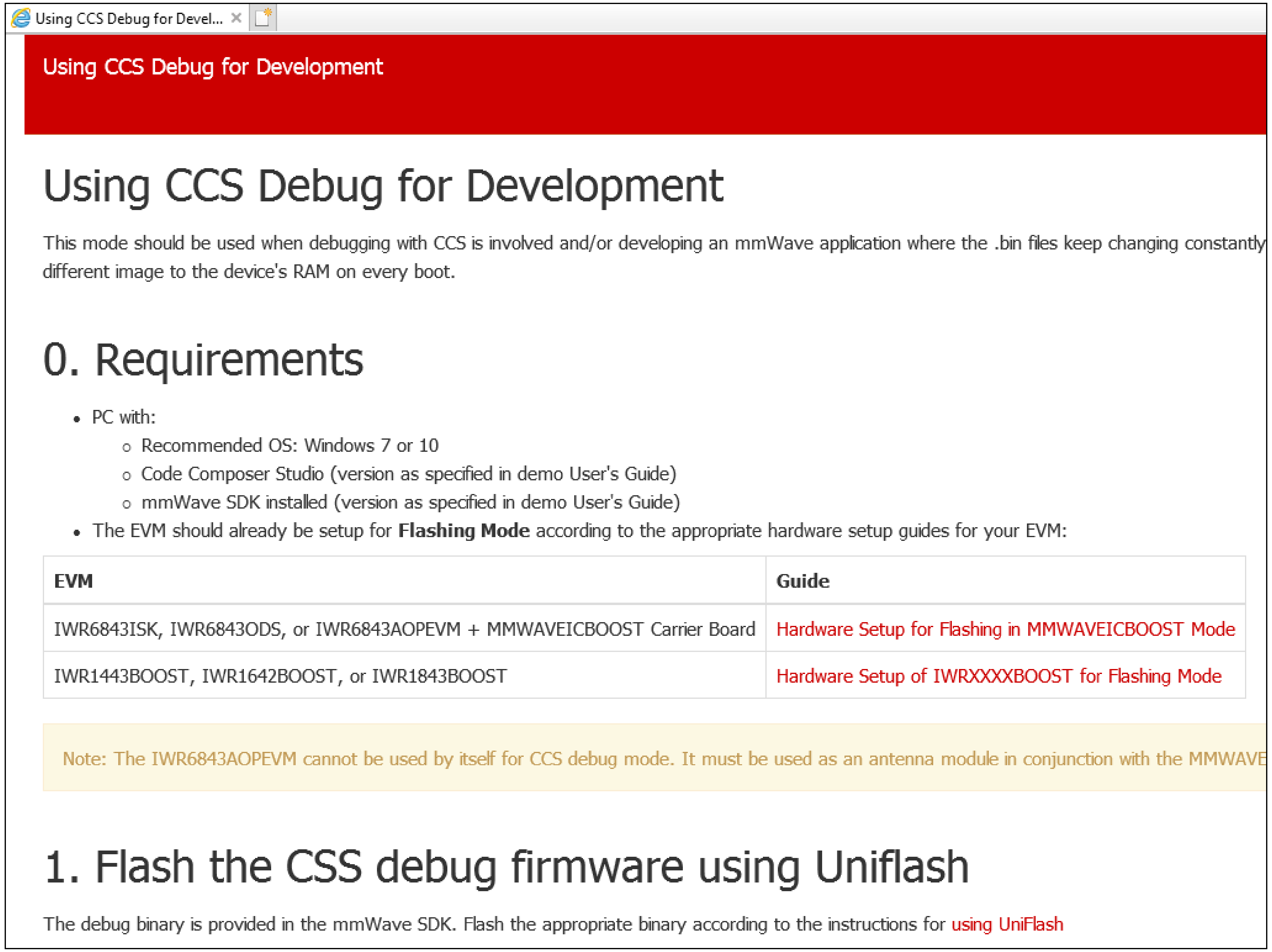
Software Development and Debugging

Software Development Resources

- **Software Development Resources:** The following resources are key to learning about software development for TI processors.
  - **BIOS users guide**: Installed as part of MMWAVE-SDK, it is available in the TI install directory e.g. C:\ti\bios_6_73_01_01\docs\Bios_User_Guide.pdf.
    - This is the one of the best resources for learning about SYSBIOS and software development for TI processors.
  - **Linker command files:** Understanding these is fundamental to developing applications for TI RTOS (SYSBIOS). Refer to the following resources:
    - TI Linker Command File Primer
    - Advanced Linker Techniques for Convenient and Efficient Memory Usage
- **Other helpful resources**
  - Getting Started with Code Composer Studio v7
  - Debugging Common Application Issues with TI-RTOS
  - TI-RTOS Workshop
  - Introduction to C6000 Architecture
  - C6000 Cache - Overview (7 of 15)
  - Using C6000 EDMA3 - Part 1 (13 of 15)

MMWAVE-SDK Debugging

- **Pre-requisites:** The user should be familiar with general CCS debugging techniques e.g. running code in CCS, putting breakpoints, observing the values of variables and memories etc. Following resources can be used to ramp-up on these topics if needed.
  - Getting Started with Code Composer Studio v7
  - Debugging Common Application Issues with TI-RTOS
- In addition to the above, the user must be familiar with basic procedures for running an MMWAVE application in CCS debug mode. This is explained in the mmWave Industrial Toolbox at the following page:
  - Using CCS Debug for Development



Using CCS Debug for Development

Using CCS Debug for Development

This mode should be used when debugging with CCS is involved and/or developing an mmWave application where the .bin files keep changing constantly different image to the device's RAM on every boot.

0. Requirements

- PC with:
  - Recommended OS: Windows 7 or 10
  - Code Composer Studio (version as specified in demo User's Guide)
  - mmWave SDK installed (version as specified in demo User's Guide)
- The EVM should already be setup for **Flashing Mode** according to the appropriate hardware setup guides for your EVM:

EVM	Guide
IWR6843ISK, IWR6843ODS, or IWR6843AOPEVM + MMWAVEICBOOST Carrier Board	Hardware Setup for Flashing in MMWAVEICBOOST Mode
IWR1443BOOST, IWR1642BOOST, or IWR1843BOOST	Hardware Setup of IWRXXXXBOOST for Flashing Mode

Note: The IWR6843AOPEVM cannot be used by itself for CCS debug mode. It must be used as an antenna module in conjunction with the MMWAVE

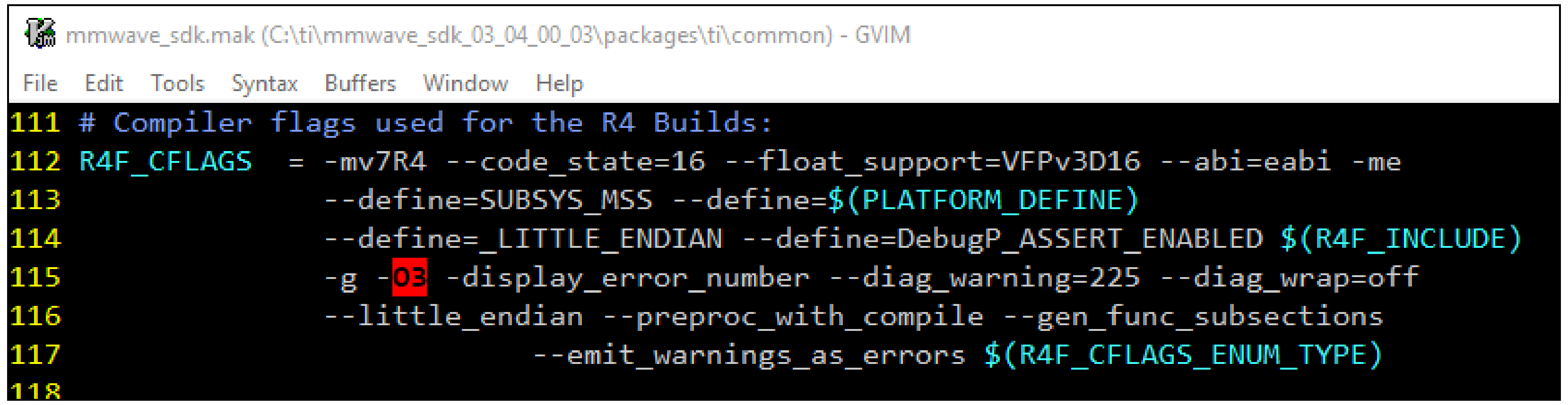
1. Flash the CSS debug firmware using Uniflash

The debug binary is provided in the mmWave SDK. Flash the appropriate binary according to the instructions for using UniFlash

MMWAVE-SDK Debugging

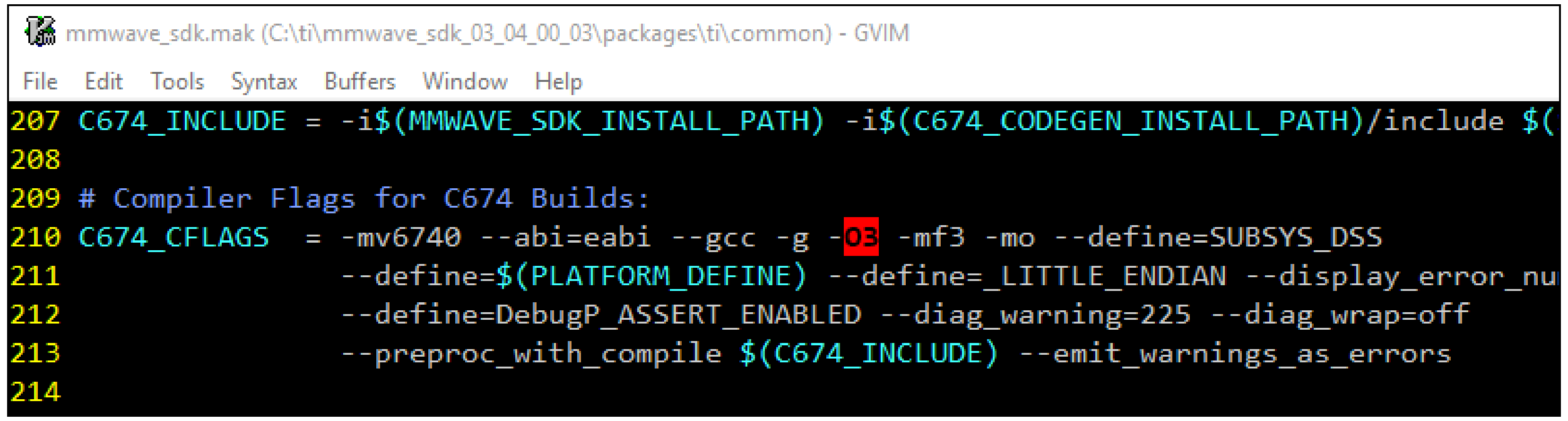
- **Rebuilding SDK code for debug:**
  - All MMWAVE-SDK code (drivers, OOB demo application and DPU libs) is built with –O3 optimization, which does not allow single step debugging in the corresponding components.
  - In order to enable step debugging in CCS, the –O3 optimization options needs to be removed (or changed to a lower level)
    - Remove the –O3 flag in the SDK common makefile from R4F_CFLAGS and/or C674_CFLAGS directives to change build options for ARM and/or DSP code.
    - Re-build the demo and the desired pre-compiled libs (e.g. drivers, DPUs) separately with the non-optimized configuration to enable debugging.
- **A note about DPUs:** All DPUs are linked with the application as pre-compiled libs, so just re-building the application/demo code does not re-build the DPU libs automatically. Any DPUs that need to be debugged, should be re-built first with the new build options and then the applications code re-compiled to link with the new DPU lib(s). This applies to the SDK drivers as well.
- Removing optimization may increase the memory requirements and/or break real-time behavior.

SDK Compile flags for Cortex R4F core


mmwave_sdk.mak (C:\ti\mmwave_sdk_03_04_00_03\packages\ti\common) - GVIM
File Edit Tools Syntax Buffers Window Help

```
# Compiler flags used for the R4 Builds:
R4F_CFLAGS  = -mv7R4 --code_state=16 --float_support=VFPv3D16 --abi=eabi -me
              --define=SUBSYS_MSS --define=$(PLATFORM_DEFINE)
              --define=_LITTLE_ENDIAN --define=DebugP_ASSERT_ENABLED $(R4F_INCLUDE)
              -g -O3 -display_error_number --diag_warning=225 --diag_wrap=off
              --little_endian --preproc_with_compile --gen_func_subsections
                    --emit_warnings_as_errors $(R4F_CFLAGS_ENUM_TYPE)
```

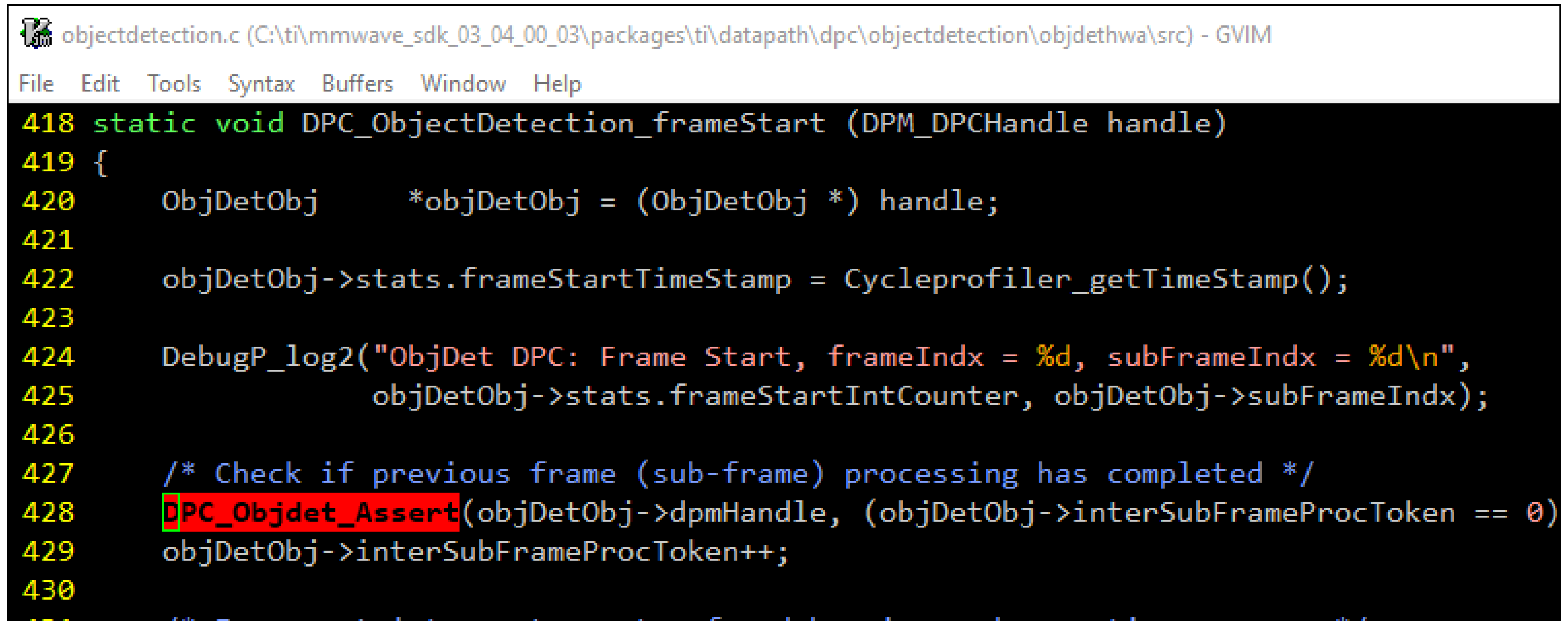
SDK Compile flags for C674x DSP core


mmwave_sdk.mak (C:\ti\mmwave_sdk_03_04_00_03\packages\ti\common) - GVIM
File Edit Tools Syntax Buffers Window Help

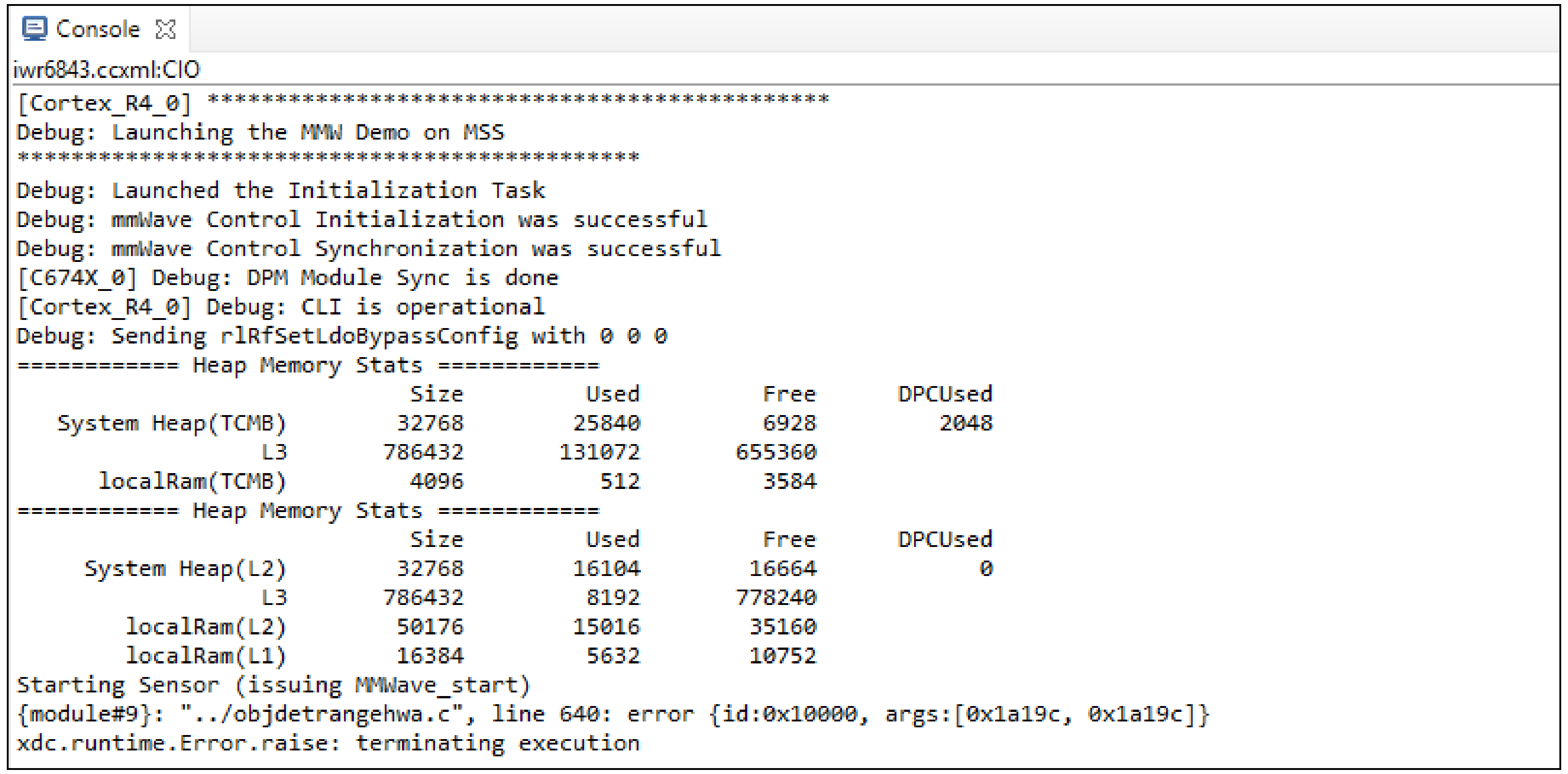
```
C674_INCLUDE = -i$(MMWAVE_SDK_INSTALL_PATH) -i$(C674_CODEGEN_INSTALL_PATH)/include $(

# Compiler Flags for C674 Builds:
C674_CFLAGS  = -mv6740 --abi=eabi --gcc -g -O3 -mf3 -mo --define=SUBSYS_DSS
               --define=$(PLATFORM_DEFINE) --define=_LITTLE_ENDIAN --display_error_nu
               --define=DebugP_ASSERT_ENABLED --diag_warning=225 --diag_wrap=off
               --preproc_with_compile $(C674_INCLUDE) --emit_warnings_as_errors
```

MMWAVE-SDK Debugging

- **Disabling Real time asserts:**
  - If the DPC cannot complete the frame (or sub-frame) processing within the real-time deadline, it raises as assert causing the application to exit
  - When step debugging a distributed DPC which runs on both R4F and DSP cores (e.g. in the 68xx point cloud detection chain), this assert should be disabled on both cores in DPC code as shown otherwise application would crash as shown in this CCS console log.
  - NOTE: Make sure to re-enable the frame timing assert when running the DPC in non debug mode.


objectdetection.c (C:\ti\mmwave_sdk_03_04_00_03\packages\ti\datapath\dpc\objectdetection\objdethwa\src) - GVIM
File
Edit
Tools
Syntax
Buffers
Window
Help

```
static void DPC_ObjectDetection_frameStart (DPM_DPCHandle handle)
{
    ObjDetObj     *objDetObj = (ObjDetObj *) handle;

    objDetObj->stats.frameStartTimeStamp = Cycleprofiler_getTimeStamp();

    DebugP_log2("ObjDet DPC: Frame Start, frameIndx = %d, subFrameIndx = %d\n",
                objDetObj->stats.frameStartIntCounter, objDetObj->subFrameIndx);

    /* Check if previous frame (sub-frame) processing has completed */
    DPC_Objdet_Assert(objDetObj->dpmHandle, (objDetObj->interSubFrameProcToken == 0)
    objDetObj->interSubFrameProcToken++;

```

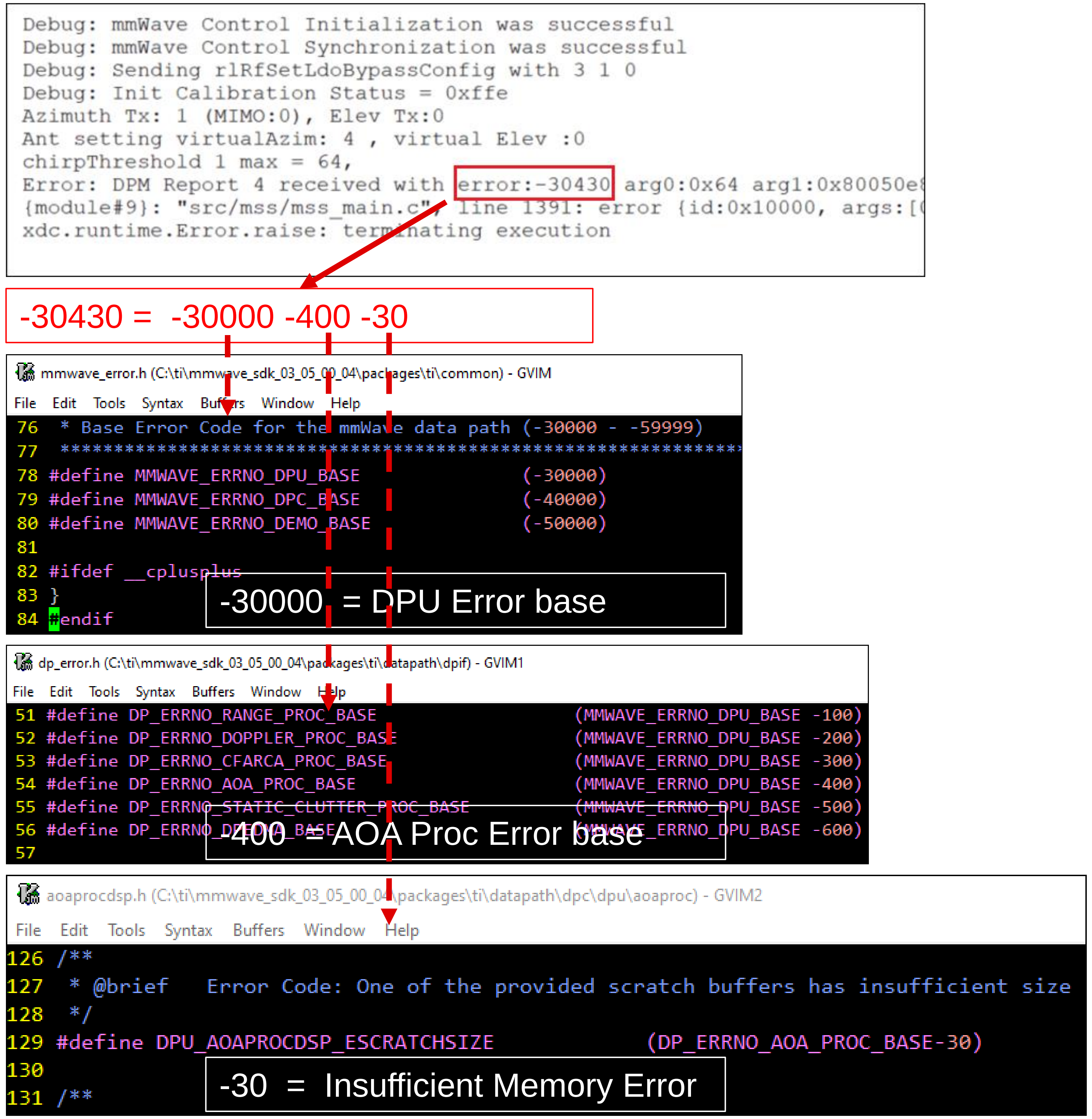


```
[Cortex_R4_0] **********************************************
Debug: Launching the MMW Demo on MSS
**********************************************
Debug: Launched the Initialization Task
Debug: mmWave Control Initialization was successful
Debug: mmWave Control Synchronization was successful
[C674X_0] Debug: DPM Module Sync is done
[Cortex_R4_0] Debug: CLI is operational
Debug: Sending rlRfSetLdoBypassConfig with 0 0 0
============ Heap Memory Stats ============
                            Size         Used         Free      DPCUsed
   System Heap(TCMB)       32768        25840         6928         2048
                  L3      786432       131072       655360
      localRam(TCMB)        4096          512         3584
============ Heap Memory Stats ============
                            Size         Used         Free      DPCUsed
     System Heap(L2)       32768        16104        16664            0
                  L3      786432         8192       778240
        localRam(L2)       50176        15016        35160
        localRam(L1)       16384         5632        10752
Starting Sensor (issuing MMWave_start)
{module#9}: "../objdetrangehwa.c", line 640: error {id:0x10000, args:[0x1a19c, 0x1a19c]}
xdc.runtime.Error.raise: terminating execution
```

Understanding Error Codes: Datapath Errors

- When demo runs into error conditions, an error code is generated and printed out on CCS console
- The error code is a negative integer e.g. shown in the picture.
- Can be from various sources such as Drivers, Control modules, DPC, DPU or demo (i.e. application)
- Error code defined as: (**Module error code base** minus **Module specific error code**)
  - The module error code base values are defined in **packages\ti\common\mmwave_error.h.**
  - The base error codes for DPC and DPU are define in **packages\ti\datapath\dpif\dp_error.h**
  - Individual DPU specific error codes defined in the DPU header files
- **Example:** Parsing the error -30430 shown here
  - The error code is from module with error base "-30000", which indicates it is DPU error
  - Referring to dp_error.h, base "-30400" is from AOA Proc.
  - Then find the error code in aoaprocdsp.h for error(-30) which is DPU_AOAPROCDSP_ESCRATCHSIZE
- **NOTE:** In SDK demos, these error codes are not sent out on UART so the demo must be run in CCS debug mode to get the error code
- Refer to SDK module documentation at the following location (in the SDK install directory) for more details:
  - file:///C:/ti/mmwave_sdk_03_05_00_04/packages/ti/demo/xwr68xx/mmw/docs/doxygen/html/index.html#mmwave_error

Datapath Errors



```
Debug: mmWave Control Initialization was successful
Debug: mmWave Control Synchronization was successful
Debug: Sending rlRfSetLdoBypassConfig with 3 1 0
Debug: Init Calibration Status = 0xffe
Azimuth Tx: 1 (MIMO:0), Elev Tx:0
Ant setting virtualAzim: 4 , virtual Elev :0
chirpThreshold 1 max = 64,
Error: DPM Report 4 received with error:-30430 arg0:0x64 arg1:0x80050e
{module#9}: "src/mss/mss_main.c", line 1391: error {id:0x10000, args:[
xdc.runtime.Error.raise: terminating execution
```

-30430 = -30000 -400 -30

mmwave_error.h (C:\ti\mmwave_sdk_03_05_00_04\packages\ti\common) - GVIM
File Edit Tools Syntax Buffers Window Help

```
 * Base Error Code for the mmWave data path (-30000 - -59999)
 ************************************************************************
#define MMWAVE_ERRNO_DPU_BASE                (-30000)
#define MMWAVE_ERRNO_DPC_BASE                (-40000)
#define MMWAVE_ERRNO_DEMO_BASE               (-50000)

#ifdef __cplusplus
}
#endif
```

-30000 = DPU Error base

dp_error.h (C:\ti\mmwave_sdk_03_05_00_04\packages\ti\datapath\dpif) - GVIM1
File Edit Tools Syntax Buffers Window Help

```
#define DP_ERRNO_RANGE_PROC_BASE                  (MMWAVE_ERRNO_DPU_BASE -100)
#define DP_ERRNO_DOPPLER_PROC_BASE                (MMWAVE_ERRNO_DPU_BASE -200)
#define DP_ERRNO_CFARCA_PROC_BASE                 (MMWAVE_ERRNO_DPU_BASE -300)
#define DP_ERRNO_AOA_PROC_BASE                    (MMWAVE_ERRNO_DPU_BASE -400)
#define DP_ERRNO_STATIC_CLUTTER_PROC_BASE         (MMWAVE_ERRNO_DPU_BASE -500)
#define DP_ERRNO_DPEDMA_BASE                      (MMWAVE_ERRNO_DPU_BASE -600)

```

-400 = AOA Proc Error base

aoaprocdsp.h (C:\ti\mmwave_sdk_03_05_00_04\packages\ti\datapath\dpc\dpu\aoaproc) - GVIM2
File Edit Tools Syntax Buffers Window Help

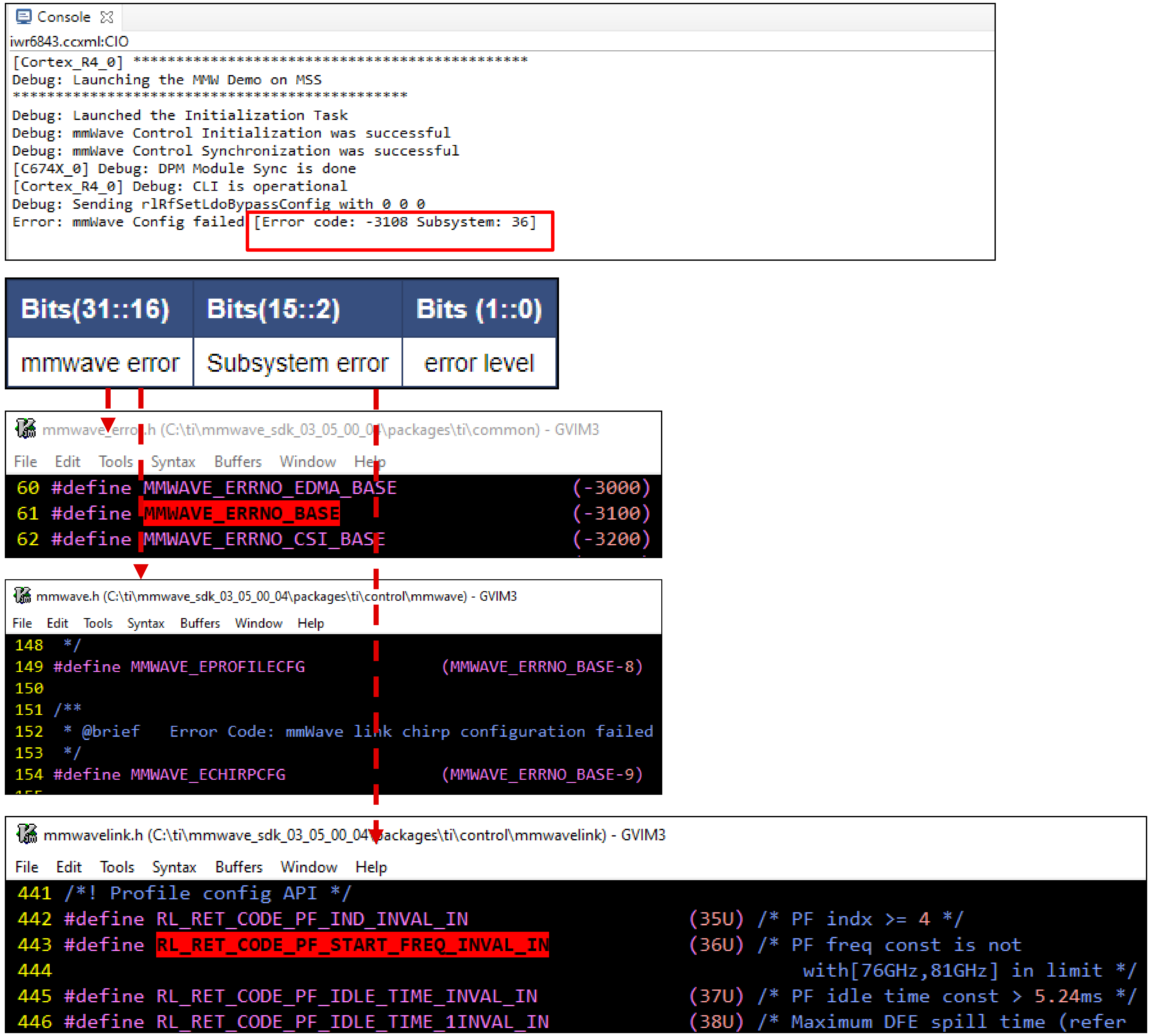
```
/**
 * @brief   Error Code: One of the provided scratch buffers has insufficient size
 */
#define DPU_AOAPROCDSP_ESCRATCHSIZE             (DP_ERRNO_AOA_PROC_BASE-30)

/**
```

-30 = Insufficient Memory Error

Understanding Error Codes: mmWave Errors

- Besides Datapath errors, there is another class of errors known as mmWave module errors.
- These represent errors returned by the RF Front-End e.g. incorrect profile configuration.
- Defined as a combination of mmWave error, Susbsystem error and error level as shown.
  - mmwave errors defined in **packages\ti\control\mmwave\mmwave.h**
  - Subsystem errors defined in **packages\ti\control\mmwavelink for mmwavelink.h**
  - Error level represents WARNING or ERROR.
- **Example:** The error shown in the log here indicates
  - The error is from module(-3100 i.e. mmwave) with error -8 (MMWAVE_EPROFILECFG)
  - The Subsystem (mmwavelink) error is 36 which is defined as RL_RET_CODE_PF_START_FREQ_INVAL_IN in mmwavelink.h, which indicates invalid start frequency specified in ProfileCfg API.

mmWave Link errors


Console
iwr6843.ccxml:CIO
mmwave_error.h (C:\ti\mmwave_sdk_03_05_00_04\packages\ti\common) - GVIM3
File Edit Tools Syntax Buffers Window Help
mmwave.h (C:\ti\mmwave_sdk_03_05_00_04\packages\ti\control\mmwave) - GVIM3
File Edit Tools Syntax Buffers Window Help
mmwavelink.h (C:\ti\mmwave_sdk_03_05_00_04\packages\ti\control\mmwavelink) - GVIM3
File Edit Tools Syntax Buffers Window Help

```
[Cortex_R4_0] **********************************************
Debug: Launching the MMW Demo on MSS
**********************************************
Debug: Launched the Initialization Task
Debug: mmWave Control Initialization was successful
Debug: mmWave Control Synchronization was successful
[C674X_0] Debug: DPM Module Sync is done
[Cortex_R4_0] Debug: CLI is operational
Debug: Sending rlRfSetLdoBypassConfig with 0 0 0
Error: mmWave Config failed [Error code: -3108 Subsystem: 36]
```

Bits(31::16)	Bits(15::2)	Bits (1::0)
mmwave error	Subsystem error	error level

```
#define MMWAVE_ERRNO_EDMA_BASE                (-3000)
#define MMWAVE_ERRNO_BASE                     (-3100)
#define MMWAVE_ERRNO_CSI_BASE                 (-3200)
```

```
 */
#define MMWAVE_EPROFILECFG                 (MMWAVE_ERRNO_BASE-8)

/**
 * @brief   Error Code: mmWave link chirp configuration failed
 */
#define MMWAVE_ECHIRPCFG                   (MMWAVE_ERRNO_BASE-9)
```

```
/*! Profile config API */
#define RL_RET_CODE_PF_IND_INVAL_IN             (35U) /* PF indx >= 4 */
#define RL_RET_CODE_PF_START_FREQ_INVAL_IN      (36U) /* PF freq const is not
                                                            with[76GHz,81GHz] in limit */
#define RL_RET_CODE_PF_IDLE_TIME_INVAL_IN       (37U) /* PF idle time const > 5.24ms */
#define RL_RET_CODE_PF_IDLE_TIME_1INVAL_IN      (38U) /* Maximum DFE spill time (refer
```

Extending SDK architecture for advanced applications

Developing a Custom Radar Processing Chain

Key considerations for developing a custom mmWave processing chain with SDK 3.x architecture

- Understand the data processing chain for the target application and model it in terms of DPC and DPUs

- Re-use the SDK Out of Box Detection Processing **OR** develop a new Detection Chain
  - SDK Out of box DPC is a Range-Doppler based processing chain.
  - Lower angular resolution as compared to Range-Azimuth (Capon Beamforming) chain
    - SDK Detection Chain re-used for Long Range People Detection and Area Scanner Demos
    - But a Higher resolution Capon Beamforming Detection Chain was developed for Indoor People Counting Demos
  - Range Processing is still re-used irrespective of the rest of the Detection Processing

- Additional processing requirements beyond the OOB point cloud detection
  - Can the additional processing be added to an existing DPU or do we need to create a new DPU
    - Object Tracking - new DPU
    - Static Object Detection - new DPU
    - 2D AoA (Angle of Arrival) using DSP – Enhancement to existing AoA DPU

- Enhancements needed at the DPC and Application level e.g.
  - Tracking
  - Beam-steering
  - Static object detection

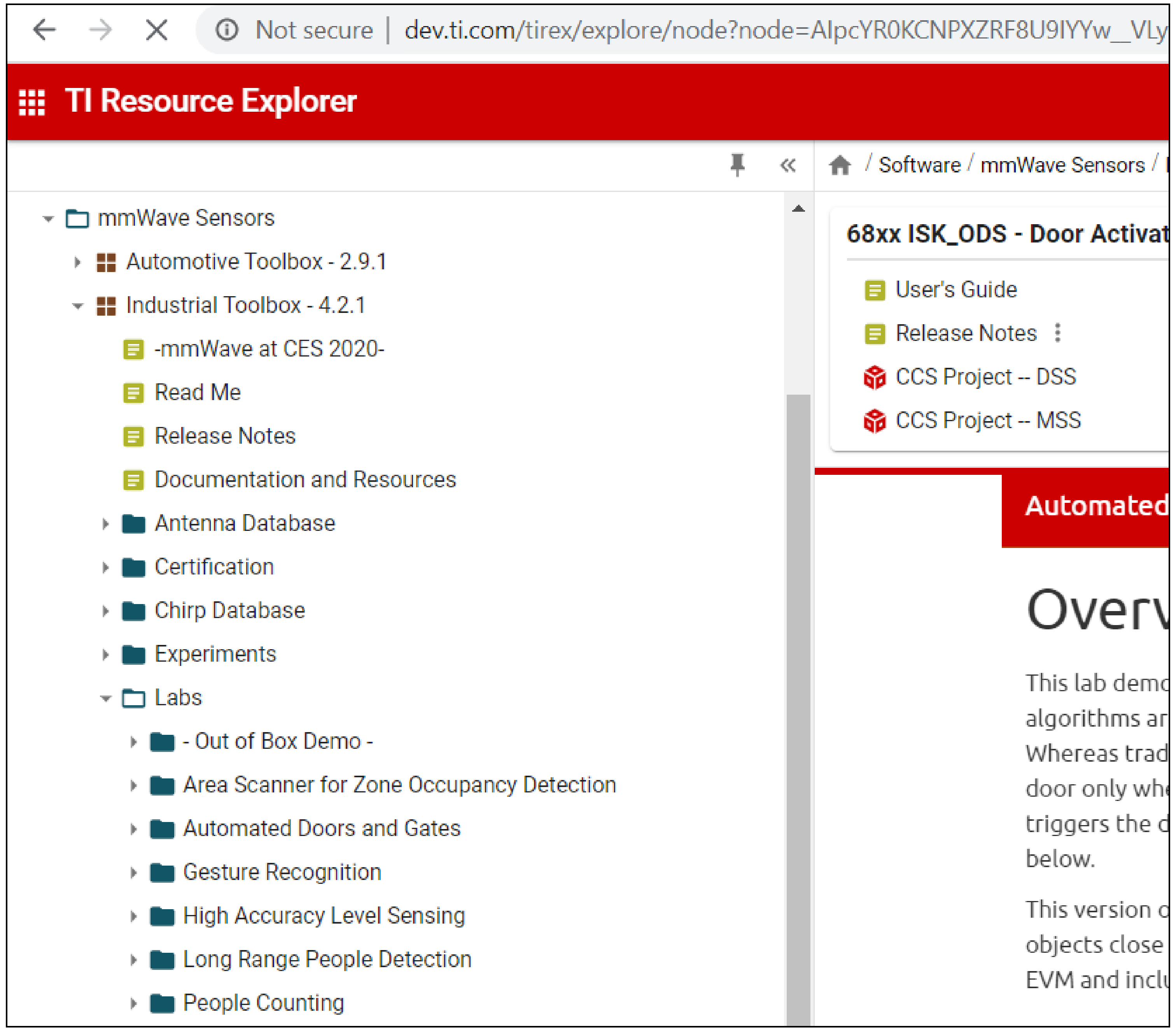
- Memory and MIPS requirements for the additional processing

Examples of Custom DPUs and DPCs

New DPUs and DPCs developed in mmWave Industrial Toolbox

- TrackerProc DPU and ObjectDetectionAndTracking DPC
  - 2D/3D Object tracking using SDK 3.3 Out of Box detection chain
  - Used in the following demos
    - Long Range People detection
    - Traffic Monitoring
    - Area Scanner
    - Automated Doors and Gates
- TrackerProcCapon DPU and Capon3D DPC
  - 2D/3D Object Tracking using Capon beamforming detection chain
  - Used in the following demos
    - 3D People Counting Demo – Side Mount
    - 3D People Counting Demo – Overhead Mount
    - Sense and Direct HVAC Control Demo
- StaticDetProc DPU and StaticObjeDet DPC
  - 2D/3D Object tracking using SDK 3.3 Out of Box detection chain with added Static detection capability
  - Used in the following demos
    - Area Scanner
    - Automated Doors and Gates

- Available in Industrial Toolbox 4.x download under **C:\ti\mmwave_industrial_toolbox_4_x_x\labs\common**



Not secure | dev.ti.com/tirex/explore/node?node=AlpcYR0KCNPXZRF8U9IYYw__VLy

TI Resource Explorer

/ Software / mmWave Sensors /

- mmWave Sensors
  - Automotive Toolbox - 2.9.1
  - Industrial Toolbox - 4.2.1
    - -mmWave at CES 2020-
    - Read Me
    - Release Notes
    - Documentation and Resources
    - Antenna Database
    - Certification
    - Chirp Database
    - Experiments
    - Labs
      - - Out of Box Demo -
      - Area Scanner for Zone Occupancy Detection
      - Automated Doors and Gates
      - Gesture Recognition
      - High Accuracy Level Sensing
      - Long Range People Detection
      - People Counting

68xx ISK_ODS - Door Activat

- User's Guide
- Release Notes
- CCS Project -- DSS
- CCS Project -- MSS

Automated

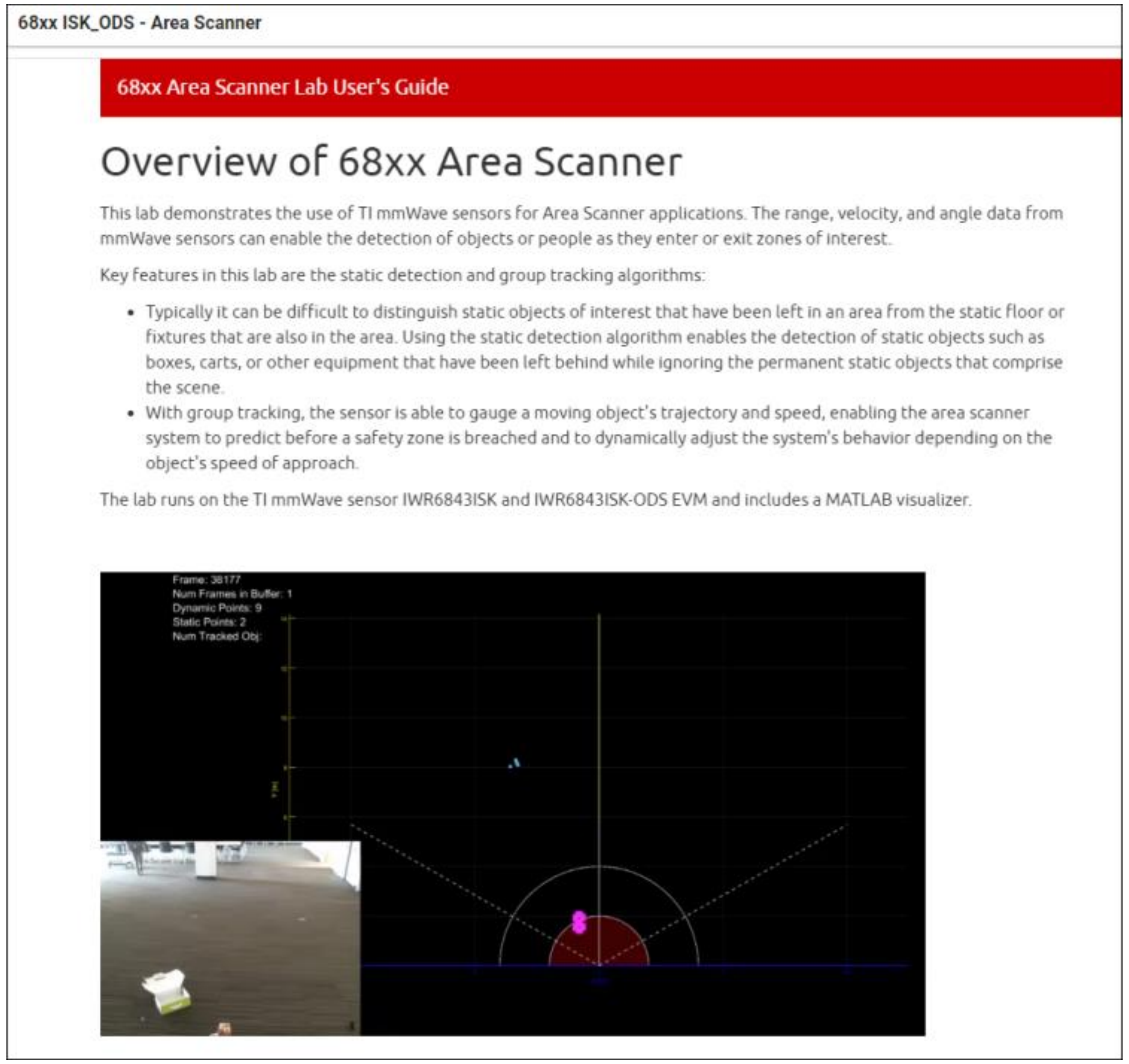
Overv

This lab demo
algorithms ar
Whereas trad
door only wh
triggers the d
below.

This version o
objects close
EVM and inclu

Area Scanner and Automated Doors

- Custom processing chain developed for Area Scanner and Automated Doors Demos.
- Based on xWR68xx OOB Range-Doppler Detection Chain with addition of 2D AoA using DSP
- Adds Object Tracking and Static Detection Capabilities



68xx ISK_ODS - Area Scanner

68xx Area Scanner Lab User's Guide

Overview of 68xx Area Scanner

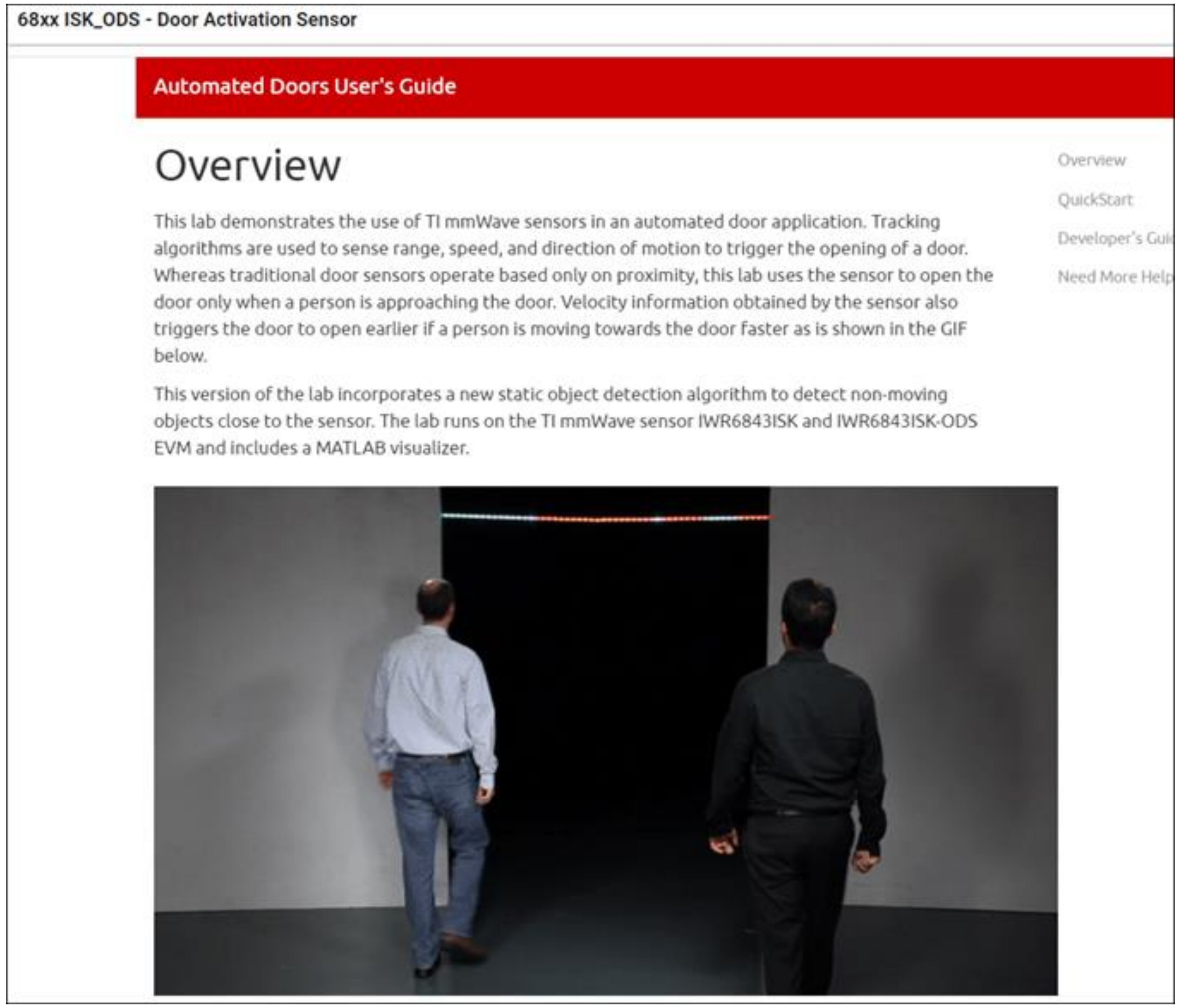
This lab demonstrates the use of TI mmWave sensors for Area Scanner applications. The range, velocity, and angle data from mmWave sensors can enable the detection of objects or people as they enter or exit zones of interest.

Key features in this lab are the static detection and group tracking algorithms:

- Typically it can be difficult to distinguish static objects of interest that have been left in an area from the static floor or fixtures that are also in the area. Using the static detection algorithm enables the detection of static objects such as boxes, carts, or other equipment that have been left behind while ignoring the permanent static objects that comprise the scene.
- With group tracking, the sensor is able to gauge a moving object's trajectory and speed, enabling the area scanner system to predict before a safety zone is breached and to dynamically adjust the system's behavior depending on the object's speed of approach.

The lab runs on the TI mmWave sensor IWR6843ISK and IWR6843ISK-ODS EVM and includes a MATLAB visualizer.

Frame: 38177
Num Frames in Buffer: 1
Dynamic Points: 9
Static Points: 2
Num Tracked Obj:



68xx ISK_ODS - Door Activation Sensor

Automated Doors User's Guide

Overview

This lab demonstrates the use of TI mmWave sensors in an automated door application. Tracking algorithms are used to sense range, speed, and direction of motion to trigger the opening of a door. Whereas traditional door sensors operate based only on proximity, this lab uses the sensor to open the door only when a person is approaching the door. Velocity information obtained by the sensor also triggers the door to open earlier if a person is moving towards the door faster as is shown in the GIF below.

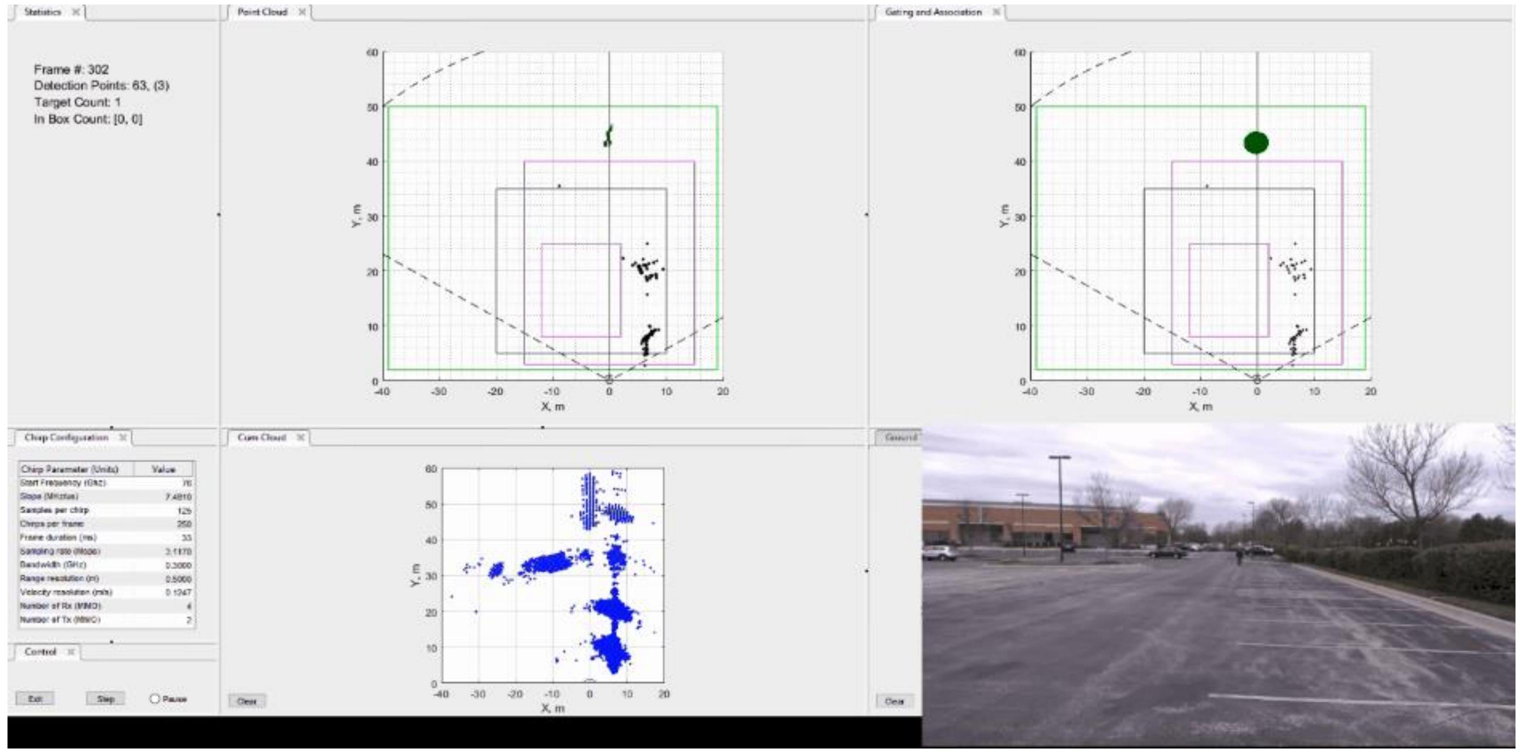
This version of the lab incorporates a new static object detection algorithm to detect non-moving objects close to the sensor. The lab runs on the TI mmWave sensor IWR6843ISK and IWR6843ISK-ODS EVM and includes a MATLAB visualizer.

Demo: Long Range Outdoor People Detection and Tracking

100m People Detection and Tracking

- **Features**
  - SDK 3.3 Range Doppler Detection Chain with Object tracking on R4F
  - Tracker DPU developed on top of SDK 3.x
  - Runs on IWR6843ISK ES2.0 and IWR1843BOOST
  - Supports 2D and 3D tracking
  - People tracking tested upto 100m with IWR6843ISK ES2.0 (3TX SIMO)
  - Re-used for Traffic Monitoring Demo on IWR1843/6843
  - Supports ISK, ODS and AOP antennas (for Indoor applications such as Area Scanner with Static Detection)
    - DSP based AoA DPU modified to add 2D DoA for ODS/AOP
  - Built on SDK 3.x OOB Demo
    - Advanced features included for free e.g. Run time CFAR tuning, FoV filtering, ADC data streaming over LVDS.

Overview



Frame #: 302
Detection Points: 63, (3)
Target Count: 1
In Box Count: [0, 0]

Chirp Parameter (Units)	Value
Start Frequency (GHz)	76
Slope (MHz/us)	7.4810
Samples per chirp	125
Chirps per frame	250
Frame duration (ms)	33
Sampling rate (Msps)	3.1170
Bandwidth (GHz)	0.3000
Range resolution (m)	0.5000
Velocity resolution (m/s)	0.1247
Number of Rx (MIMO)	4
Number of Tx (MIMO)	2

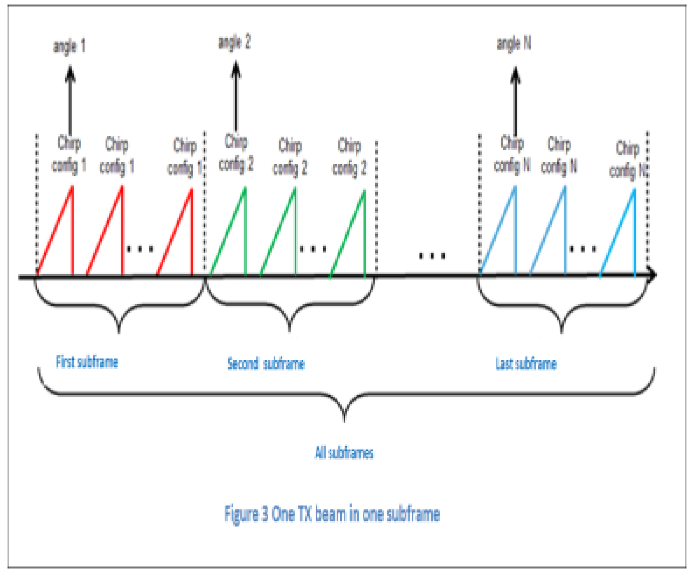
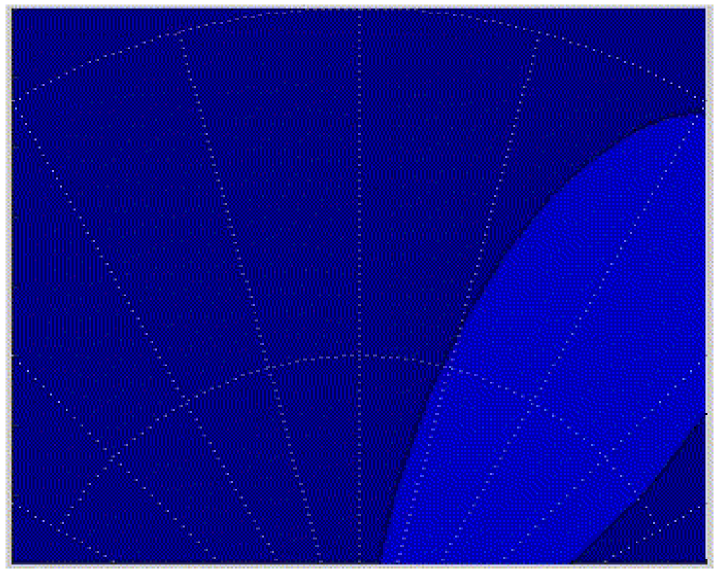
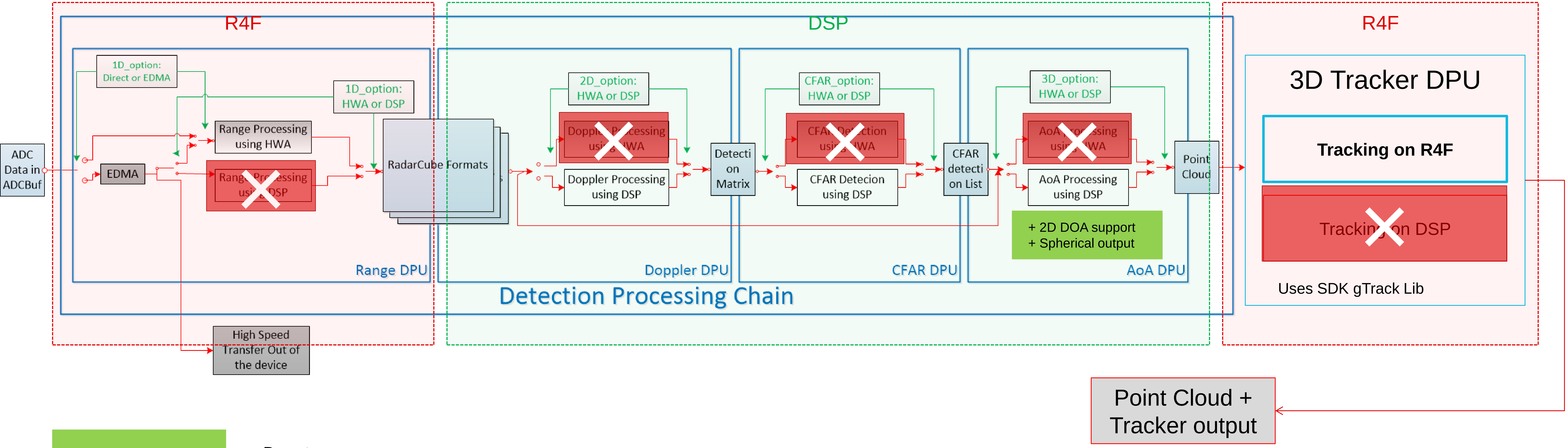


Figure 3 One TX beam in one subframe



Sub frame based TX Beam Steering

3D Long Range Tracker Chain



🟩 → Denotes new or modified components

🟥 → Denotes Unused Datapath

Texas Instruments

The material is provided strictly "as-is" for informational purposes only and without any warranty.
Use of this material is subject to TI's **Terms of Use**, viewable at TI.com

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale (www.ti.com/legal/termsofsale.html) or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265