

TI IWR6432 mmWave Radar

Generated by Doxygen 1.13.2

1 Data Structure Index	1
1.1 Data Structures	1
2 File Index	3
2.1 File List	3
3 Data Structure Documentation	5
3.1 DPC_ObjectDetection_MemCfg_t Struct Reference	5
3.1.1 Detailed Description	5
3.2 MemPoolObj_t Struct Reference	5
3.2.1 Detailed Description	6
3.3 Mmw_calibData_t Struct Reference	6
3.3.1 Detailed Description	6
3.4 T_SensPerChirpLut Struct Reference	6
3.4.1 Detailed Description	7
4 File Documentation	9
4.1 include/defines.h File Reference	9
4.1.1 Detailed Description	10
4.2 defines.h	10
4.3 include/dpu_res.h File Reference	11
4.3.1 Detailed Description	14
4.4 dpu_res.h	15
4.5 include/factory_cal.h File Reference	17
4.5.1 Detailed Description	18
4.5.2 Macro Definition Documentation	18
4.5.2.1 MMWDEMO_CALIB_STORE_MAGIC	18
4.5.3 Typedef Documentation	19
4.5.3.1 Mmw_calibData	19
4.5.4 Function Documentation	19
4.5.4.1 restoreFactoryCal()	19
4.6 factory_cal.h	19
4.7 include/mem_pool.h File Reference	20
4.7.1 Detailed Description	20
4.7.2 Macro Definition Documentation	20
4.7.2.1 MEM_ALIGN	20
4.7.3 Function Documentation	20
4.7.3.1 DPC_ObjDet_MemPoolAlloc()	20
4.7.3.2 DPC_ObjDet_MemPoolGetMaxUsage()	21
4.7.3.3 DPC_ObjDet_MemPoolReset()	21
4.8 mem_pool.h	22
4.9 mmwave_basic.h	22
4.10 mmwave_control_config.h	23

4.11 include/rangeproc_dpc.h File Reference	24
4.11.1 Detailed Description	25
4.11.2 Function Documentation	25
4.11.2.1 chirpStartISR()	25
4.11.2.2 dpcTask()	26
4.11.2.3 RangeProc_config()	26
4.11.2.4 rangeProc_dpulnit()	26
4.11.2.5 rangeproc_main()	26
4.11.2.6 registerChirpAvailableInterrupts()	27
4.11.2.7 registerChirpInterrupt()	27
4.11.2.8 registerFrameStartInterrupt()	27
4.11.2.9 uartTask()	28
4.11.3 Variable Documentation	28
4.11.3.1 rangeProcHWADpuHandle	28
4.12 rangeproc_dpc.h	28
4.13 include/uart_transmit.h File Reference	29
4.13.1 Detailed Description	29
4.13.2 Function Documentation	30
4.13.2.1 uart_transmit_loop()	30
4.13.3 Variable Documentation	30
4.13.3.1 uart_tx_done_sem	30
4.13.3.2 uart_tx_start_sem	30
4.14 uart_transmit.h	30
4.15 src/factory_cal.c File Reference	31
4.15.1 Detailed Description	31
4.15.2 Function Documentation	32
4.15.2.1 restoreFactoryCal()	32
4.16 src/main.c File Reference	33
4.16.1 Detailed Description	34
4.16.2 Variable Documentation	35
4.16.2.1 uart_tx_done_sem	35
4.16.2.2 uart_tx_start_sem	35
4.17 src/mem_pool.c File Reference	35
4.17.1 Detailed Description	35
4.17.2 Function Documentation	35
4.17.2.1 DPC_ObjDet_MemPoolAlloc()	35
4.17.2.2 DPC_ObjDet_MemPoolGetMaxUsage()	36
4.17.2.3 DPC_ObjDet_MemPoolReset()	36
4.18 src/mmwave_basic.c File Reference	37
4.18.1 Detailed Description	38
4.18.2 Function Documentation	38
4.18.2.1 mempool_init()	38

4.18.2.2 Mmwave_populateDefaultCalibrationCfg()	39
4.18.2.3 Mmwave_populateDefaultChirpControlCfg()	40
4.18.2.4 Mmwave_populateDefaultOpenCfg()	40
4.18.2.5 Mmwave_populateDefaultStartCfg()	40
4.19 src/mmwave_control_config.c File Reference	41
4.19.1 Detailed Description	42
4.19.2 Function Documentation	42
4.19.2.1 MMWave_populateChannelCfg()	42
4.19.2.2 Mmwave_populateDefaultCalibrationCfg()	42
4.19.2.3 Mmwave_populateDefaultChirpControlCfg()	43
4.19.2.4 Mmwave_populateDefaultOpenCfg()	43
4.19.2.5 Mmwave_populateDefaultStartCfg()	43
4.20 src/rangeproc_dpc.c File Reference	44
4.20.1 Detailed Description	45
4.20.2 Function Documentation	46
4.20.2.1 chirpStartISR()	46
4.20.2.2 dpcTask()	46
4.20.2.3 RangeProc_config()	46
4.20.2.4 rangeProc_dpulnit()	46
4.20.2.5 registerChirpAvailableInterrupts()	47
4.20.2.6 registerChirpInterrupt()	47
4.20.2.7 registerFrameStartInterrupt()	47
4.20.2.8 uartTask()	47
4.20.3 Variable Documentation	48
4.20.3.1 rangeProcHWADpuHandle	48
4.21 src/uart_transmit.c File Reference	48
4.21.1 Detailed Description	49
4.21.2 Function Documentation	49
4.21.2.1 uart_transmit_loop()	49
4.21.3 Variable Documentation	49
4.21.3.1 header	49

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

DPC_ObjectDetection_MemCfg_t	
Memory Configuration used during init API	5
MemPoolObj_t	
Memory pool object to manage memory based on DPC_ObjectDetection_MemCfg_t	5
Mmw_calibData_t	
Structure holds calibration save configuration used during sensor open	6
T_SensPerChirpLut	
Sensor Perchirp LUT, total 64 bytes used, 4 values per params	6

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

include/defines.h	Configuration macros for the radar system	9
include/dpu_res.h	Resource definitions for Data Processing Unit (DPU) configurations	11
include/factory_cal.h	Factory calibration related functions	17
include/mem_pool.h	20
include/mmwave_basic.h	22
include/mmwave_control_config.h	23
include/rangeproc_dpc.h	Range Processing Data Path Unit (DPU) Interface	24
include/uart_transmit.h	UART Transmission Interface for Radar Data	29
src/factory_cal.c	Factory Calibration Restoration and Configuration	31
src/main.c	Main Program for Radar Signal Processing	33
src/mem_pool.c	35
src/mmwave_basic.c	Initialization, configuration, and control functions for mmWave sensor	37
src/mmwave_control_config.c	Configuration and initialization for mmWave chirp profiles	41
src/rangeproc_dpc.c	Implementation of the Range Processing DPC (Data Processing Chain)	44
src/uart_transmit.c	UART Transmission Implementation for Radar Data	48

Chapter 3

Data Structure Documentation

3.1 DPC_ObjectDetection_MemCfg_t Struct Reference

Memory Configuration used during init API.

```
#include <mem_pool.h>
```

Data Fields

- void * **addr**
Start address of memory provided by the application from which DPC will allocate.
- uint32_t **size**
Size limit of memory allowed to be consumed by the DPC.

3.1.1 Detailed Description

Memory Configuration used during init API.

The documentation for this struct was generated from the following file:

- include/[mem_pool.h](#)

3.2 MemPoolObj_t Struct Reference

Memory pool object to manage memory based on [DPC_ObjectDetection_MemCfg_t](#).

```
#include <mem_pool.h>
```

Data Fields

- [DPC_ObjectDetection_MemCfg](#) **cfg**
Memory configuration.
- `uintptr_t` **currAddr**
Pool running adress.
- `uintptr_t` **maxCurrAddr**
Pool max address. This pool allows setting address to desired (e.g for rewinding purposes), so having a running maximum helps in finding max pool usage.

3.2.1 Detailed Description

Memory pool object to manage memory based on [DPC_ObjectDetection_MemCfg_t](#).

The documentation for this struct was generated from the following file:

- `include/mem_pool.h`

3.3 Mmw_calibData_t Struct Reference

Structure holds calibration save configuration used during sensor open.

```
#include <factory_cal.h>
```

Data Fields

- `uint32_t` **magic**
Magic word for calibration data.
- `T_RL_API_FECSS_RXTX_CAL_DATA` **calibData**
RX TX Calibration data.

3.3.1 Detailed Description

Structure holds calibration save configuration used during sensor open.

The structure holds calibration save configuration.

The documentation for this struct was generated from the following file:

- `include/factory_cal.h`

3.4 T_SensPerChirpLut Struct Reference

Sensor Perchirp LUT, total 64 bytes used, 4 values per params.

```
#include <mmwave_control_config.h>
```

Data Fields

- uint32_t **StartFreqHighRes** [4]
- uint32_t **StartFreqLowRes** [4]
- int16_t **ChirpSlope** [4]
- uint16_t **ChirpIdleTime** [4]
- uint16_t **ChirpAdcStartTime** [4]
- int16_t **ChirpTxStartTime** [4]
- uint8_t **ChirpTxEn** [4]
- uint8_t **ChirpBpmEn** [4]

3.4.1 Detailed Description

Sensor Perchirp LUT, total 64 bytes used, 4 values per params.

The documentation for this struct was generated from the following file:

- include/mmwave_control_config.h

Chapter 4

File Documentation

4.1 include/defines.h File Reference

Configuration macros for the radar system.

Macros

- `#define LOW_POWER_MODE 2`
- `#define NUM_TX_ANTENNAS 2`
- `#define NUM_RX_ANTENNAS 3`
- `#define NUM_VIRT_ANTENNAS (NUM_TX_ANTENNAS * NUM_RX_ANTENNAS)`
- `#define NUM_ADC_SAMPLES 256`
- `#define NUM_BURSTS_PER_FRAME 1`
- `#define NUM_CHIRPS_PER_BURST 8`
- `#define NUM_CHIRPS_PER_FRAME (NUM_BURSTS_PER_FRAME * NUM_CHIRPS_PER_BURST)`
- `#define RX_CH_CTRL_BITMASK 7`
- `#define TX_CH_CTRL_BITMASK 3`
- `#define CHANNEL_CFG_MISC_CTRL 0`
- `#define NUM_RANGE_BINS (NUM_ADC_SAMPLES / 2)`
- `#define NUM_DOPPLER_CHIRPS_PER_FRAME (NUM_CHIRPS_PER_FRAME / NUM_TX_ANTENNAS)`
- `#define NUM_DOPPLER_CHIRPS_PER_PROC NUM_DOPPLER_CHIRPS_PER_FRAME`
- `#define CHIRPCOMNCFG_DIG_OUTPUT_SAMP_RATE 20`
- `#define CHIRPCOMNCFG_DIG_OUTPUT_BITS_SEL 0`
- `#define CHIRPCOMNCFG_DFE_FIR_SEL 0`
- `#define CHIRPCOMNCFG_NUM_OF_ADC_SAMPLES NUM_ADC_SAMPLES`
- `#define CHIRPCOMNCFG_CHIRP_TX_MIMO_PAT_SEL 4`
- `#define CHIRPCOMNCFG_MISC_SETTINGS M_RL_SENS_MISC_HPF_FAST_INIT_DIS_BIT`
- `#define CHIRPCOMNCFG_HPF_FAST_INIT_DURATION 15U`
- `#define CHIRPCOMNCFG_CHIRP_RAMP_END_TIME 60.0`
- `#define CHIRPCOMNCFG_CHIRP_RX_HPF_SEL M_RL_SENS_RX_HPF_SEL_300KHZ`
- `#define CHIRPTIMINGCFG_CHIRP_IDLE_TIME 6`
- `#define CHIRPTIMINGCFG_CHIRP_ADC_START_TIME (28 << 10)`
- `#define CHIRPTIMINGCFG_CHIRP_TX_START_TIME 0`
- `#define CHIRPTIMINGCFG_CHIRP_RF_FREQ_SLOPE 65`
- `#define CHIRPTIMINGCFG_CHIRP_RF_FREQ_START 59.75`
- `#define CHIRPTIMINGCFG_CHIRP_TX_EN_SEL 0x3U`
- `#define CHIRPTIMINGCFG_CHIRP_TX_BPM_EN_SEL 0x0U`

- #define NUM_CHIRPS_ACCUM 0
- #define BURST_PERIOD 643
- #define W_BURST_PERIOD (10.0 * BURST_PERIOD)
- #define FRAME_PERIOD (((float)(250.0) * 40000000.0)/1000.0)
- #define NUM_FRAMES 0
- #define SENSOR_START_FRAME_TRIG_MODE 0
- #define SENSOR_START_CHIRP_START_SIG_LB_ENABLE 0
- #define SENSOR_START_FRAME_LIVE_MON_ENABLE 0
- #define SENSOR_START_FRAME_TRIG_TIMER_VAL 0
- #define CLI_FACCALCFG_RES_EN 0
- #define CLI_FACCALCFG_RX_GAIN 40
- #define CLI_FACCALCFG_TX_BACKOFF_SEL 0
- #define CLI_FACCALCFG_FLASH_OFFSET 0x1FF000
- #define SYS_COMMON_NUM_RX_CHANNEL 3U
- #define SYS_COMMON_CQ_MAX_CHIRP_THRESHOLD 8U
- #define SYS_COMMON_CP_SIZE_CBUFF_UNITS 2U
- #define DMA_TRIG_SRC_CHAN_0 0
- #define DMA_TRIG_SRC_CHAN_1 1

4.1.1 Detailed Description

Configuration macros for the radar system.

This file contains macro definitions for antenna settings, chirp configurations, timing parameters, and system-level settings.

The configuration settings can be generated with the "mmWave Sensing Estimator" tool (<https://dev.ti.com/gallery/view/mmwave/mmWaveSensingEstimator/ver/2.4.0/>)

4.2 defines.h

[Go to the documentation of this file.](#)

```
00001
00012
00013
00014 #define LOW_POWER_MODE 2
00015
00016 /* basic configuration (frameCfg and others)*/
00017 #define NUM_TX_ANTENNAS 2
00018 #define NUM_RX_ANTENNAS 3
00019 #define NUM_VIRT_ANTENNAS (NUM_TX_ANTENNAS * NUM_RX_ANTENNAS)
00020 #define NUM_ADC_SAMPLES 256 // 128 // number of adc samples per chirp hardcoded in
// https://dev.ti.com/gallery/view/mmwave/mmWaveSensingEstimator/ver/2.4.0/ and also used in
// MOTION_AND_PRESENCE_DETECTION_DEMO
00021 #define NUM_BURSTS_PER_FRAME 1 // from MOTION_AND_PRESENCE_DETECTION_DEMO
00022 #define NUM_CHIRPS_PER_BURST 8 // from MOTION_AND_PRESENCE_DETECTION_DEMO
00023 #define NUM_CHIRPS_PER_FRAME (NUM_BURSTS_PER_FRAME * NUM_CHIRPS_PER_BURST)
00024
00025 // channelCfg
00026 #define RX_CH_CTRL_BITMASK 7 // all 3 RX antennas active => 7 (0b111)
00027 #define TX_CH_CTRL_BITMASK 3 // all 2 TX antennas active => 3 (0b11)
00028 #define CHANNEL_CFG_MISC_CTRL 0
00029
00030 // calculated defines, not in config
00031 #define NUM_RANGE_BINS (NUM_ADC_SAMPLES / 2)
00032 #define NUM_DOPPLER_CHIRPS_PER_FRAME (NUM_CHIRPS_PER_FRAME / NUM_TX_ANTENNAS)
00033 #define NUM_DOPPLER_CHIRPS_PER_PROC NUM_DOPPLER_CHIRPS_PER_FRAME
00034
00035 /* chirpComnCfg */
00036 #define CHIRPCOMNCFG_DIG_OUTPUT_SAMP_RATE 20 // 5 MHz //
// M_RL_SENS_DIG_OUT_SAMP_RATE_MAX_12P5M
00037
00038 #define CHIRPCOMNCFG_DIG_OUTPUT_BITS_SEL 0 // M_RL_SENS_DIG_OUT_12BITS_4LSB_ROUND
00039 #define CHIRPCOMNCFG_DFE_FIR_SEL 0 // M_RL_SENS_DFE_FIR_LONG_FILT
```



```

00040 #define CHIRPCOMNCFG_NUM_OF_ADC_SAMPLES      NUM_ADC_SAMPLES // 256U; /* 2.56us */
00041 #define CHIRPCOMNCFG_CHIRP_TX_MIMO_PAT_SEL    4 // 0; M_RL_SENS_TX_MIMO_PATRN_DIS
00042 // not in .cfg file:
00043 #define CHIRPCOMNCFG_MISC_SETTINGS            M_RL_SENS_MISC_HPF_FAST_INIT_DIS_BIT // 0U; /* HPF
        FINIT, CRD ena, PA blank dis */
00044 #define CHIRPCOMNCFG_HPF_FAST_INIT_DURATION  15U // 15U; /* 1.5us */
00045 #define CHIRPCOMNCFG_CHIRP_RAMP_END_TIME      60.0 //30.0 // 600; 250U; /* 25us low res */
00046 #define CHIRPCOMNCFG_CHIRP_RX_HPF_SEL        M_RL_SENS_RX_HPF_SEL_300KHZ //
        M_RL_SENS_RX_HPF_SEL_350KHZ
00048
00049 /* chirpTimingCfg */
00050 #define CHIRPTIMINGCFG_CHIRP_IDLE_TIME        6 // 400; 65U; /* 6.5us low res */
00051 #define CHIRPTIMINGCFG_CHIRP_ADC_START_TIME  (28 < 10) // 30770;
00052 #define CHIRPTIMINGCFG_CHIRP_TX_START_TIME    0 // -10; /* -0.2us */
00053 #define CHIRPTIMINGCFG_CHIRP_RF_FREQ_SLOPE    65 // 699; 3495; /* 100MHz/us , 77G - 2621 */
00054 #define CHIRPTIMINGCFG_CHIRP_RF_FREQ_START    59.75 // calculation from defines
        (M_RL_SENS_CHIRP_RFFREQ_LR_59G) are unclear, so use fix value
00055 // not in .cfg file:
00056 #define CHIRPTIMINGCFG_CHIRP_TX_EN_SEL        0x3U // 2 TX enable in chirp
00057 #define CHIRPTIMINGCFG_CHIRP_TX_BPM_EN_SEL    0x0U // 0; 0x2U; /* TX1 BPM enable in chirp */
00058 #define NUM_CHIRPS_ACCUM                      0
00059 #define BURST_PERIOD                          643 // 403
00060 #define W_BURST_PERIOD                        (10.0 * BURST_PERIOD)
00061 #define FRAME_PERIOD                          ((float) (250.0) * 40000000.0) / 1000.0)
00062 #define NUM_FRAMES                            0
00063
00064 /* sensorStart */
00065 #define SENSOR_START_FRAME_TRIG_MODE          0
00066 #define SENSOR_START_CHIRP_START_SIG_LB_ENABLE 0
00067 #define SENSOR_START_FRAME_LIVE_MON_ENABLE    0
00068 #define SENSOR_START_FRAME_TRIG_TIMER_VAL     0
00069
00070 /* Factory Calibration */
00071 #define CLI_FACCALCFG_RES_EN 0
00072 #define CLI_FACCALCFG_RX_GAIN 40
00073 #define CLI_FACCALCFG_TX_BACKOFF_SEL 0
00074 #define CLI_FACCALCFG_FLASH_OFFSET 0x1FF000
00075
00076 // derived from common/syscommon.h
00077 // *****
00078 * MMWAVE System level defines
00079 *****
00080 #define SYS_COMMON_NUM_RX_CHANNEL              3U
00081 #define SYS_COMMON_CQ_MAX_CHIRP_THRESHOLD      8U
00082
00083 /* This is the size of the Chirp Parameters (CP) in CBUFF Units */
00084 #define SYS_COMMON_CP_SIZE_CBUFF_UNITS        2U
00085
00086
00087
00088
00089
00090
00091 // DMA channel defines
00092 #define DMA_TRIG_SRC_CHAN_0 0
00093 #define DMA_TRIG_SRC_CHAN_1 1

```

4.3 include/dpu_res.h File Reference

Resource definitions for Data Processing Unit (DPU) configurations.

```

#include <drivers/edma.h>
#include <drivers/hw_include/cslr_soc.h>

```

Macros

- #define **DPC_OBJDET_HWA_WINDOW_RAM_OFFSET** 0
- #define **DPC_OBJDET_EDMA_SHADOW_BASE** SOC_EDMA_NUM_DMACH
- #define **DPC_OBJDET_DPU_RANGEPROC_EDMAIN_CH** EDMA_APPSS_TPCC_B_EVT_CHIRP_↵
AVAIL_IRQ
- #define **DPC_OBJDET_DPU_RANGEPROC_EDMAIN_SHADOW_PING** (DPC_OBJDET_EDMA_↵
SHADOW_BASE + 0)

- #define **DPC_OBJDET_DPU_RANGEPROC_EDMAIN_SHADOW_PONG** (DPC_OBJDET_EDMA_↔
SHADOW_BASE + 1)
- #define **DPC_OBJDET_DPU_RANGEPROC_EDMAIN_EVENT_QUE** 0
- #define **DPC_OBJDET_DPU_RANGEPROC_EDMAIN_SIG_CH** EDMA_APPSS_TPCC_B_EVT_FREE_0
- #define **DPC_OBJDET_DPU_RANGEPROC_EDMAIN_SIG_SHADOW** (DPC_OBJDET_EDMA_↔
SHADOW_BASE + 2)
- #define **DPC_OBJDET_DPU_RANGEPROC_EDMAIN_SIG_EVENT_QUE** 0
- #define **DPC_OBJDET_DPU_RANGEPROC_EDMAOUT_MAJOR_PING_CH** EDMA_APPSS_TPCC_B_↔
EVT_HWA_DMA_REQ0
- #define **DPC_OBJDET_DPU_RANGEPROC_EDMAOUT_MAJOR_PING_SHADOW** (DPC_OBJDET_↔
EDMA_SHADOW_BASE + 3)
- #define **DPC_OBJDET_DPU_RANGEPROC_EDMAOUT_MAJOR_PING_EVENT_QUE** 0
- #define **DPC_OBJDET_DPU_RANGEPROC_EDMAOUT_MINOR_PING_CH** EDMA_APPSS_TPCC_B_↔
EVT_FREE_1
- #define **DPC_OBJDET_DPU_RANGEPROC_EDMAOUT_MINOR_PING_SHADOW** (DPC_OBJDET_↔
EDMA_SHADOW_BASE + 4)
- #define **DPC_OBJDET_DPU_RANGEPROC_EDMAOUT_MINOR_PING_EVENT_QUE** 0
- #define **DPC_OBJDET_DPU_RANGEPROC_EVT_DECIM_PING_CH** EDMA_APPSS_TPCC_B_EVT_↔
FREE_2
- #define **DPC_OBJDET_DPU_RANGEPROC_EVT_DECIM_PING_SHADOW_0** (DPC_OBJDET_EDMA_↔
SHADOW_BASE + 5)
- #define **DPC_OBJDET_DPU_RANGEPROC_EVT_DECIM_PING_SHADOW_1** (DPC_OBJDET_EDMA_↔
SHADOW_BASE + 6)
- #define **DPC_OBJDET_DPU_RANGEPROC_EVT_DECIM_PING_EVENT_QUE** 0
- #define **DPC_OBJDET_DPU_RANGEPROC_EDMAOUT_MAJOR_PONG_CH** EDMA_APPSS_TPCC_B_↔
_EVT_HWA_DMA_REQ1
- #define **DPC_OBJDET_DPU_RANGEPROC_EDMAOUT_MAJOR_PONG_SHADOW** (DPC_OBJDET_↔
EDMA_SHADOW_BASE + 7)
- #define **DPC_OBJDET_DPU_RANGEPROC_EDMAOUT_MAJOR_PONG_EVENT_QUE** 0
- #define **DPC_OBJDET_DPU_RANGEPROC_EDMAOUT_MINOR_PONG_CH** EDMA_APPSS_TPCC_B_↔
_EVT_FREE_3
- #define **DPC_OBJDET_DPU_RANGEPROC_EDMAOUT_MINOR_PONG_SHADOW** (DPC_OBJDET_↔
EDMA_SHADOW_BASE + 8)
- #define **DPC_OBJDET_DPU_RANGEPROC_EDMAOUT_MINOR_PONG_EVENT_QUE** 0
- #define **DPC_OBJDET_DPU_RANGEPROC_EVT_DECIM_PONG_CH** EDMA_APPSS_TPCC_B_EVT_↔
FREE_4
- #define **DPC_OBJDET_DPU_RANGEPROC_EVT_DECIM_PONG_SHADOW_0** (DPC_OBJDET_EDMA_↔
_SHADOW_BASE + 9)
- #define **DPC_OBJDET_DPU_RANGEPROC_EVT_DECIM_PONG_SHADOW_1** (DPC_OBJDET_EDMA_↔
_SHADOW_BASE + 10)
- #define **DPC_OBJDET_DPU_RANGEPROC_EVT_DECIM_PONG_EVENT_QUE** 0
- #define **DPC_OBJDET_DPU_DOAPROC_EDMAIN_PING_CH** EDMA_APPSS_TPCC_B_EVT_FREE_5
- #define **DPC_OBJDET_DPU_DOAPROC_EDMAIN_PING_SHADOW** (DPC_OBJDET_EDMA_SHADOW_↔
_BASE + 11)
- #define **DPC_OBJDET_DPU_DOAPROC_EDMAIN_PING_EVENT_QUE** 0
- #define **DPC_OBJDET_DPU_DOAPROC_EDMAIN_PONG_CH** EDMA_APPSS_TPCC_B_EVT_FREE_6
- #define **DPC_OBJDET_DPU_DOAPROC_EDMAIN_PONG_SHADOW** (DPC_OBJDET_EDMA_SHADOW_↔
_BASE + 12)
- #define **DPC_OBJDET_DPU_DOAPROC_EDMAIN_PONG_EVENT_QUE** 0
- #define **DPC_OBJDET_DPU_DOAPROC_EDMA_HOT_SIG_CH** EDMA_APPSS_TPCC_B_EVT_FREE_7
- #define **DPC_OBJDET_DPU_DOAPROC_EDMA_HOT_SIG_SHADOW** (DPC_OBJDET_EDMA_↔
SHADOW_BASE + 13)
- #define **DPC_OBJDET_DPU_DOAPROC_EDMA_HOT_SIG_EVENT_QUE** 0
- #define **DPC_OBJDET_DPU_DOAPROC_EDMAOUT_DET_MATRIX_CH** EDMA_APPSS_TPCC_B_↔
EVT_HWA_DMA_REQ2

- #define **DPC_OBJDET_DPU_DOAPROC_EDMAOUT_DET_MATRIX_SHADOW** (DPC_OBJDET_EDMA↔
_SHADOW_BASE + 14)
- #define **DPC_OBJDET_DPU_DOAPROC_EDMAOUT_DET_MATRIX_EVENT_QUE** 0
- #define **DPC_OBJDET_DPU_DOAPROC_EDMAOUT_ELEVIND_MATRIX_CH** EDMA_APPSS_TPCC_B↔
_EVT_FREE_8
- #define **DPC_OBJDET_DPU_DOAPROC_EDMAOUT_ELEVIND_MATRIX_SHADOW** (DPC_OBJDET↔
EDMA_SHADOW_BASE + 15)
- #define **DPC_OBJDET_DPU_DOAPROC_EDMAOUT_ELEVIND_MATRIX_EVENT_QUE** 0
- #define **DPC_OBJDET_DPU_DOAPROC_EDMAOUT_DOPIND_MATRIX_CH** EDMA_APPSS_TPCC_B↔
_EVT_FREE_9
- #define **DPC_OBJDET_DPU_DOAPROC_EDMAOUT_DOPIND_MATRIX_SHADOW** (DPC_OBJDET↔
EDMA_SHADOW_BASE + 16)
- #define **DPC_OBJDET_DPU_DOAPROC_EDMAOUT_DOPIND_MATRIX_EVENT_QUE** 0
- #define **DPC_OBJDET_DPU_DOAPROC_INTER_LOOP_EDMAOUT_DET_MATRIX_CH** EDMA↔
APPSS_TPCC_B_EVT_HWA_DMA_REQ3
- #define **DPC_OBJDET_DPU_DOAPROC_INTER_LOOP_EDMAOUT_DET_MATRIX_SHADOW** (DPC↔
OBJDET_EDMA_SHADOW_BASE + 17)
- #define **DPC_OBJDET_DPU_DOAPROC_INTER_LOOP_EDMAOUT_DET_MATRIX_EVENT_QUE** 0
- #define **DPC_OBJDET_DPU_DOAPROC_INTER_LOOP_EDMAIN_CH** EDMA_APPSS_TPCC_B_EVT↔
FREE_10
- #define **DPC_OBJDET_DPU_DOAPROC_INTER_LOOP_EDMAIN_SHADOW** (DPC_OBJDET_EDMA↔
SHADOW_BASE + 18)
- #define **DPC_OBJDET_DPU_DOAPROC_INTER_LOOP_EDMAIN_EVENT_QUE** 0
- #define **DPC_OBJDET_DPU_DOAPROC_INTER_LOOP_EDMA_HOT_SIG_CH** EDMA_APPSS_TPCC↔
B_EVT_FREE_11
- #define **DPC_OBJDET_DPU_DOAPROC_INTER_LOOP_EDMA_HOT_SIG_SHADOW** (DPC_OBJDET↔
EDMA_SHADOW_BASE + 19)
- #define **DPC_OBJDET_DPU_DOAPROC_INTER_LOOP_EDMA_HOT_SIG_EVENT_QUE** 0
- #define **DPC_OBJDET_DPU_DOAPROC_INTER_LOOP_EDMA_CHAIN_BACK_CH** EDMA_APPSS↔
TPCC_B_EVT_FREE_12
- #define **DPC_OBJDET_DPU_DOAPROC_INTER_LOOP_EDMA_CHAIN_BACK_SHADOW** (DPC↔
OBJDET_EDMA_SHADOW_BASE + 20)
- #define **DPC_OBJDET_DPU_DOAPROC_INTER_LOOP_EDMA_CHAIN_BACK_EVENT_QUE** 0
- #define **DPC_OBJDET_DPU_CFAR_PROC_EDMAIN_CH** EDMA_APPSS_TPCC_B_EVT_FREE_13
- #define **DPC_OBJDET_DPU_CFAR_PROC_EDMAIN_SHADOW** (DPC_OBJDET_EDMA_SHADOW↔
BASE + 21)
- #define **DPC_OBJDET_DPU_CFAR_PROC_EDMAIN_EVENT_QUE** 0
- #define **DPC_OBJDET_DPU_CFAR_PROC_EDMAIN_SIG_CH** EDMA_APPSS_TPCC_B_EVT_FREE_14
- #define **DPC_OBJDET_DPU_CFAR_PROC_EDMAIN_SIG_SHADOW** (DPC_OBJDET_EDMA_SHADOW↔
_BASE + 22)
- #define **DPC_OBJDET_DPU_CFAR_PROC_EDMAIN_SIG_EVENT_QUE** 0
- #define **DPC_OBJDET_DPU_CFAR_PROC_EDMAOUT_RNG_PROFILE_CH** EDMA_APPSS_TPCC_B↔
_EVT_HWA_DMA_REQ4
- #define **DPC_OBJDET_DPU_CFAR_PROC_EDMAOUT_RNG_PROFILE_SHADOW** (DPC_OBJDET↔
EDMA_SHADOW_BASE + 23)
- #define **DPC_OBJDET_DPU_CFAR_PROC_EDMAOUT_RNG_PROFILE_EVENT_QUE** 0
- #define **DPC_OBJDET_DPU_UDOP_PROC_EDMA_RESET_CH** EDMA_APPSS_TPCC_B_EVT_FREE↔
_15
- #define **DPC_OBJDET_DPU_UDOP_PROC_EDMA_RESET_SHADOW** (DPC_OBJDET_EDMA↔
SHADOW_BASE + 24)
- #define **DPC_OBJDET_DPU_UDOP_PROC_EDMARESET_EVENT_QUE** 0
- #define **DPC_OBJDET_DPU_UDOP_PROC_EDMAIN_CH** EDMA_APPSS_TPCC_B_EVT_FREE_16
- #define **DPC_OBJDET_DPU_UDOP_PROC_EDMAIN_SHADOW** (DPC_OBJDET_EDMA_SHADOW↔
BASE + 25)
- #define **DPC_OBJDET_DPU_UDOP_PROC_EDMAIN_EVENT_QUE** 0
- #define **DPC_OBJDET_DPU_UDOP_PROC_EDMAIN_SIG_CH** EDMA_APPSS_TPCC_B_EVT_FREE_17

- `#define DPC_OBJDET_DPU_UDOP_PROC_EDMAIN_SIG_SHADOW (DPC_OBJDET_EDMA_SHADOW_↵`
`_BASE + 26)`
- `#define DPC_OBJDET_DPU_UDOP_PROC_EDMAIN_SIG_EVENT_QUE 0`
- `#define DPC_OBJDET_DPU_UDOP_PROC_EDMAOUT_CHAIN_CH EDMA_APPSS_TPCC_B_EVT_↵`
`HWA_DMA_REQ5`
- `#define DPC_OBJDET_DPU_UDOP_PROC_EDMAOUT_CHAIN0_SHADOW (DPC_OBJDET_EDMA_↵`
`SHADOW_BASE + 27)`
- `#define DPC_OBJDET_DPU_UDOP_PROC_EDMAOUT_CHAIN1_SHADOW (DPC_OBJDET_EDMA_↵`
`SHADOW_BASE + 28)`
- `#define DPC_OBJDET_DPU_UDOP_PROC_EDMAOUT_CHAIN_EVENT_QUE 0`
- `#define DPC_OBJDET_DPU_UDOP_PROC_EDMAOUT_UDOPPLER_CH EDMA_APPSS_TPCC_B_↵`
`EVT_FREE_18`
- `#define DPC_OBJDET_DPU_UDOP_PROC_EDMAOUT_UDOPPLER_SHADOW (DPC_OBJDET_↵`
`EDMA_SHADOW_BASE + 29)`
- `#define DPC_OBJDET_DPU_UDOP_PROC_EDMAOUT_UDOPPLER_EVENT_QUE 0`

4.3.1 Detailed Description

Resource definitions for Data Processing Unit (DPU) configurations.

This file defines the hardware resources and configurations for the Data Processing Units (DPUs) used in the radar signal processing pipeline. It includes EDMA (Enhanced Direct Memory Access) channel assignments, shadow configurations, and event queue mappings for various DPUs such as Range Processing, Direction of Arrival (DoA), CFAR (Constant False Alarm Rate), and Micro Doppler.

The configurations are derived from the Motion and Presence Detection Demo provided in the TI mmWave SDK. These resources are critical for ensuring proper signal processing and data transfer between the radar front-end and the processing units.

Note

This file is adapted from the Motion and Presence Detection Demo: `${MMWAVE_SDK_INSTALL_↵`
`PATH}\examples\mmw_demo\motion_and_presence_detection\source\mmw_res.h`

Copyright

Copyright (C) 2022-24 Texas Instruments Incorporated

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the name of Texas Instruments Incorporated nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

4.4 dpu_res.h

[Go to the documentation of this file.](#)

```

00001 #ifndef DPU_RES_H
00002 #define DPU_RES_H
00003
00050
00051
00052
00053 #ifdef __cplusplus
00054 extern "C" {
00055 #endif
00056
00057 #include <drivers/edma.h>
00058 #include <drivers/hw_include/cslr_soc.h>
00059
00060 /*****
00061  * Resources for Object Detection DPC, currently the only DPC and hwa/edma
00062  * resource used in the demo.
00063  *****/
00064
00065 #define DPC_OBJDET_HWA_WINDOW_RAM_OFFSET 0
00066
00067 #define DPC_OBJDET_EDMA_SHADOW_BASE SOC_EDMA_NUM_DMACH
00068
00069 /* Range DPU */
00070 #define DPC_OBJDET_DPU_RANGEPROC_EDMAIN_CH
00071         EDMA_APPSS_TPCC_B_EVT_CHIRP_AVAIL_IRQ
00072 #define DPC_OBJDET_DPU_RANGEPROC_EDMAIN_SHADOW_PING 0
00073 #define DPC_OBJDET_DPU_RANGEPROC_EDMAIN_SHADOW_PONG 1
00074 #define DPC_OBJDET_DPU_RANGEPROC_EDMAIN_EVENT_QUE 0
00075 #define DPC_OBJDET_DPU_RANGEPROC_EDMAIN_SIG_CH EDMA_APPSS_TPCC_B_EVT_FREE_0
00076 #define DPC_OBJDET_DPU_RANGEPROC_EDMAIN_SIG_SHADOW 2
00077 #define DPC_OBJDET_DPU_RANGEPROC_EDMAIN_SIG_EVENT_QUE 0
00078 #define DPC_OBJDET_DPU_RANGEPROC_EDMAOUT_MAJOR_PING_CH
00079         EDMA_APPSS_TPCC_B_EVT_HWA_DMA_REQ0
00080 #define DPC_OBJDET_DPU_RANGEPROC_EDMAOUT_MAJOR_PING_SHADOW 3
00081 #define DPC_OBJDET_DPU_RANGEPROC_EDMAOUT_MAJOR_PING_EVENT_QUE 0
00082 #define DPC_OBJDET_DPU_RANGEPROC_EDMAOUT_MINOR_PING_CH EDMA_APPSS_TPCC_B_EVT_FREE_1
00083 #define DPC_OBJDET_DPU_RANGEPROC_EDMAOUT_MINOR_PING_SHADOW 4
00084 #define DPC_OBJDET_DPU_RANGEPROC_EDMAOUT_MINOR_PING_EVENT_QUE 0
00085
00086 #define DPC_OBJDET_DPU_RANGEPROC_EVT_DECIM_PING_CH EDMA_APPSS_TPCC_B_EVT_FREE_2
00087 #define DPC_OBJDET_DPU_RANGEPROC_EVT_DECIM_PING_SHADOW_0 5
00088 #define DPC_OBJDET_DPU_RANGEPROC_EVT_DECIM_PING_SHADOW_1 6
00089 #define DPC_OBJDET_DPU_RANGEPROC_EVT_DECIM_PING_EVENT_QUE 0
00090
00091 #define DPC_OBJDET_DPU_RANGEPROC_EDMAOUT_MAJOR_PONG_CH
00092         EDMA_APPSS_TPCC_B_EVT_HWA_DMA_REQ1
00093 #define DPC_OBJDET_DPU_RANGEPROC_EDMAOUT_MAJOR_PONG_SHADOW 7
00094 #define DPC_OBJDET_DPU_RANGEPROC_EDMAOUT_MAJOR_PONG_EVENT_QUE 0
00095 #define DPC_OBJDET_DPU_RANGEPROC_EDMAOUT_MINOR_PONG_CH EDMA_APPSS_TPCC_B_EVT_FREE_3
00096 #define DPC_OBJDET_DPU_RANGEPROC_EDMAOUT_MINOR_PONG_SHADOW 8
00097 #define DPC_OBJDET_DPU_RANGEPROC_EDMAOUT_MINOR_PONG_EVENT_QUE 0
00098
00099 #define DPC_OBJDET_DPU_RANGEPROC_EVT_DECIM_PONG_CH EDMA_APPSS_TPCC_B_EVT_FREE_4
00100 #define DPC_OBJDET_DPU_RANGEPROC_EVT_DECIM_PONG_SHADOW_0 9
00101 #define DPC_OBJDET_DPU_RANGEPROC_EVT_DECIM_PONG_SHADOW_1 10
00102 #define DPC_OBJDET_DPU_RANGEPROC_EVT_DECIM_PONG_EVENT_QUE 0
00103
00104 /* DoA DPU */
00105 #define DPC_OBJDET_DPU_DOAPROC_EDMAIN_PING_CH EDMA_APPSS_TPCC_B_EVT_FREE_5
00106 #define DPC_OBJDET_DPU_DOAPROC_EDMAIN_PING_SHADOW 11
00107 #define DPC_OBJDET_DPU_DOAPROC_EDMAIN_PING_EVENT_QUE 0
00108
00109 #define DPC_OBJDET_DPU_DOAPROC_EDMAIN_PONG_CH EDMA_APPSS_TPCC_B_EVT_FREE_6
00110 #define DPC_OBJDET_DPU_DOAPROC_EDMAIN_PONG_SHADOW 12
00111 #define DPC_OBJDET_DPU_DOAPROC_EDMAIN_PONG_EVENT_QUE 0
00112

```

```

00113 #define DPC_OBJDET_DPU_DOAPROC_EDMA_HOT_SIG_CH          EDMA_APPSS_TPCC_B_EVT_FREE_7
00114 #define DPC_OBJDET_DPU_DOAPROC_EDMA_HOT_SIG_SHADOW      (DPC_OBJDET_EDMA_SHADOW_BASE +
13)
00115 #define DPC_OBJDET_DPU_DOAPROC_EDMA_HOT_SIG_EVENT_QUE    0
00116
00117 #define DPC_OBJDET_DPU_DOAPROC_EDMAOUT_DET_MATRIX_CH     EDMA_APPSS_TPCC_B_EVT_HWA_DMA_REQ2
00118 #define DPC_OBJDET_DPU_DOAPROC_EDMAOUT_DET_MATRIX_SHADOW (DPC_OBJDET_EDMA_SHADOW_BASE +
14)
00119 #define DPC_OBJDET_DPU_DOAPROC_EDMAOUT_DET_MATRIX_EVENT_QUE 0
00120
00121 #define DPC_OBJDET_DPU_DOAPROC_EDMAOUT_ELEVIND_MATRIX_CH EDMA_APPSS_TPCC_B_EVT_FREE_8
00122 #define DPC_OBJDET_DPU_DOAPROC_EDMAOUT_ELEVIND_MATRIX_SHADOW (DPC_OBJDET_EDMA_SHADOW_BASE +
15)
00123 #define DPC_OBJDET_DPU_DOAPROC_EDMAOUT_ELEVIND_MATRIX_EVENT_QUE 0
00124
00125 #define DPC_OBJDET_DPU_DOAPROC_EDMAOUT_DOPIND_MATRIX_CH   EDMA_APPSS_TPCC_B_EVT_FREE_9
00126 #define DPC_OBJDET_DPU_DOAPROC_EDMAOUT_DOPIND_MATRIX_SHADOW (DPC_OBJDET_EDMA_SHADOW_BASE +
16)
00127 #define DPC_OBJDET_DPU_DOAPROC_EDMAOUT_DOPIND_MATRIX_EVENT_QUE 0
00128
00129 #define DPC_OBJDET_DPU_DOAPROC_INTER_LOOP_EDMAOUT_DET_MATRIX_CH EDMA_APPSS_TPCC_B_EVT_HWA_DMA_REQ3
00130 #define DPC_OBJDET_DPU_DOAPROC_INTER_LOOP_EDMAOUT_DET_MATRIX_SHADOW (DPC_OBJDET_EDMA_SHADOW_BASE
+ 17)
00131 #define DPC_OBJDET_DPU_DOAPROC_INTER_LOOP_EDMAOUT_DET_MATRIX_EVENT_QUE 0
00132
00133 #define DPC_OBJDET_DPU_DOAPROC_INTER_LOOP_EDMAIN_CH       EDMA_APPSS_TPCC_B_EVT_FREE_10
00134 #define DPC_OBJDET_DPU_DOAPROC_INTER_LOOP_EDMAIN_SHADOW   (DPC_OBJDET_EDMA_SHADOW_BASE
+ 18)
00135 #define DPC_OBJDET_DPU_DOAPROC_INTER_LOOP_EDMAIN_EVENT_QUE 0
00136
00137 #define DPC_OBJDET_DPU_DOAPROC_INTER_LOOP_EDMA_HOT_SIG_CH EDMA_APPSS_TPCC_B_EVT_FREE_11
00138 #define DPC_OBJDET_DPU_DOAPROC_INTER_LOOP_EDMA_HOT_SIG_SHADOW (DPC_OBJDET_EDMA_SHADOW_BASE
+ 19)
00139 #define DPC_OBJDET_DPU_DOAPROC_INTER_LOOP_EDMA_HOT_SIG_EVENT_QUE 0
00140
00141 #define DPC_OBJDET_DPU_DOAPROC_INTER_LOOP_EDMA_CHAIN_BACK_CH EDMA_APPSS_TPCC_B_EVT_FREE_12
00142 #define DPC_OBJDET_DPU_DOAPROC_INTER_LOOP_EDMA_CHAIN_BACK_SHADOW (DPC_OBJDET_EDMA_SHADOW_BASE
+ 20)
00143 #define DPC_OBJDET_DPU_DOAPROC_INTER_LOOP_EDMA_CHAIN_BACK_EVENT_QUE 0
00144
00145 /* CFAR DPU */
00146 #define DPC_OBJDET_DPU_CFAR_PROC_EDMAIN_CH               EDMA_APPSS_TPCC_B_EVT_FREE_13
00147 #define DPC_OBJDET_DPU_CFAR_PROC_EDMAIN_SHADOW           (DPC_OBJDET_EDMA_SHADOW_BASE
+ 21)
00148 #define DPC_OBJDET_DPU_CFAR_PROC_EDMAIN_EVENT_QUE        0
00149
00150 #define DPC_OBJDET_DPU_CFAR_PROC_EDMAIN_SIG_CH           EDMA_APPSS_TPCC_B_EVT_FREE_14
00151 #define DPC_OBJDET_DPU_CFAR_PROC_EDMAIN_SIG_SHADOW       (DPC_OBJDET_EDMA_SHADOW_BASE
+ 22)
00152 #define DPC_OBJDET_DPU_CFAR_PROC_EDMAIN_SIG_EVENT_QUE    0
00153
00154 #define DPC_OBJDET_DPU_CFAR_PROC_EDMAOUT_RNG_PROFILE_CH   EDMA_APPSS_TPCC_B_EVT_HWA_DMA_REQ4
00155 #define DPC_OBJDET_DPU_CFAR_PROC_EDMAOUT_RNG_PROFILE_SHADOW (DPC_OBJDET_EDMA_SHADOW_BASE
+ 23)
00156 #define DPC_OBJDET_DPU_CFAR_PROC_EDMAOUT_RNG_PROFILE_EVENT_QUE 0
00157
00158 /* Micro Doppler DPU */
00159 #define DPC_OBJDET_DPU_UDOP_PROC_EDMA_RESET_CH           EDMA_APPSS_TPCC_B_EVT_FREE_15
00160 #define DPC_OBJDET_DPU_UDOP_PROC_EDMA_RESET_SHADOW       (DPC_OBJDET_EDMA_SHADOW_BASE
+ 24)
00161 #define DPC_OBJDET_DPU_UDOP_PROC_EDMARESET_EVENT_QUE     0
00162
00163 #define DPC_OBJDET_DPU_UDOP_PROC_EDMAIN_CH               EDMA_APPSS_TPCC_B_EVT_FREE_16
00164 #define DPC_OBJDET_DPU_UDOP_PROC_EDMAIN_SHADOW           (DPC_OBJDET_EDMA_SHADOW_BASE
+ 25)
00165 #define DPC_OBJDET_DPU_UDOP_PROC_EDMAIN_EVENT_QUE        0
00166
00167 #define DPC_OBJDET_DPU_UDOP_PROC_EDMAIN_SIG_CH           EDMA_APPSS_TPCC_B_EVT_FREE_17
00168 #define DPC_OBJDET_DPU_UDOP_PROC_EDMAIN_SIG_SHADOW       (DPC_OBJDET_EDMA_SHADOW_BASE
+ 26)
00169 #define DPC_OBJDET_DPU_UDOP_PROC_EDMAIN_SIG_EVENT_QUE    0
00170
00171 #define DPC_OBJDET_DPU_UDOP_PROC_EDMAOUT_CHAIN_CH        EDMA_APPSS_TPCC_B_EVT_HWA_DMA_REQ5
00172 #define DPC_OBJDET_DPU_UDOP_PROC_EDMAOUT_CHAIN0_SHADOW   (DPC_OBJDET_EDMA_SHADOW_BASE
+ 27)
00173 #define DPC_OBJDET_DPU_UDOP_PROC_EDMAOUT_CHAIN1_SHADOW   (DPC_OBJDET_EDMA_SHADOW_BASE
+ 28)
00174 #define DPC_OBJDET_DPU_UDOP_PROC_EDMAOUT_CHAIN_EVENT_QUE 0
00175
00176 #define DPC_OBJDET_DPU_UDOP_PROC_EDMAOUT_UDOPPLER_CH     EDMA_APPSS_TPCC_B_EVT_FREE_18
00177 #define DPC_OBJDET_DPU_UDOP_PROC_EDMAOUT_UDOPPLER_SHADOW (DPC_OBJDET_EDMA_SHADOW_BASE
+ 29)
00178 #define DPC_OBJDET_DPU_UDOP_PROC_EDMAOUT_UDOPPLER_EVENT_QUE 0

```

```

00179
00180 #ifdef __cplusplus
00181 }
00182 #endif
00183
00184 #endif /* DPU_RES_H */
00185

```

4.5 include/factory_cal.h File Reference

Factory calibration related functions.

Data Structures

- struct [Mmw_calibData_t](#)
Structure holds calibration save configuration used during sensor open.

Macros

- #define [MMWDEMO_CALIB_STORE_MAGIC](#) (0x7CB28DF9U)
Magic word for factory calibration data validation.

Typedefs

- typedef struct [Mmw_calibData_t](#) [Mmw_calibData](#)
Structure holds calibration save configuration used during sensor open.

Functions

- int32_t [restoreFactoryCal](#) (void)
Restores factory calibration data from flash.

Variables

- T_RL_API_SENS_CHIRP_PROF_COMN_CFG **profileComCfg**
- T_RL_API_SENS_CHIRP_PROF_TIME_CFG **profileTimeCfg**
- T_RL_API_FECSS_RF_PWR_CFG_CMD **channelCfg**
- T_RL_API_SENS_FRAME_CFG **frameCfg**
- MMWave_Handle **gCtrlHandle**
This is the mmWave control handle which is used to configure the BSS.
- T_RL_API_FECSS_RUNTIME_TX_CLPC_CAL_CMD **fecTxclpcCalCmd**

4.5.1 Detailed Description

Factory calibration related functions.

This header defines data structure and function to hold calibration data.

Note

This function in this file is adapted from the Motion and Presence Detection Demo: `$(MMWAVE_SDK_↵
INSTALL_PATH)\examples\mmw_demo\motion_and_presence_detection\source\calibration\factory_cal.c`

Copyright

Copyright (C) 2022-24 Texas Instruments Incorporated

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the name of Texas Instruments Incorporated nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

4.5.2 Macro Definition Documentation

4.5.2.1 MMWDEMO_CALIB_STORE_MAGIC

```
#define MMWDEMO_CALIB_STORE_MAGIC (0x7CB28DF9U)
```

Magic word for factory calibration data validation.

This value is stored in flash alongside calibration data and checked upon restoration to verify data integrity. It acts as a simple checksum to ensure the calibration data is valid.

4.5.3 Typedef Documentation

4.5.3.1 Mmw_calibData

```
typedef struct Mmw_calibData_t Mmw_calibData
```

Structure holds calibration save configuration used during sensor open.

The structure holds calibration save configuration.

4.5.4 Function Documentation

4.5.4.1 restoreFactoryCal()

```
int32_t restoreFactoryCal (
    void )
```

Restores factory calibration data from flash.

This function reads the calibration data stored in flash memory and restores it. It requires that the system has been previously calibrated.

Derived from mmwDemo_factoryCal and MmwDemo_calibRestore in [factory_cal.c](#) from the demo project.

Returns

SystemP_SUCCESS on success, -1 on failure.

4.6 factory_cal.h

[Go to the documentation of this file.](#)

```
00001 #ifndef FACTORY_CAL_H
00002 #define FACTORY_CAL_H
00003
00004
00005
00006
00007 #define MMWDEMO_CALIB_STORE_MAGIC (0x7CB28DF9U)
00008
00009 // externals
00010 extern T_RL_API_SENS_CHIRP_PROF_COMN_CFG profileComCfg;
00011 extern T_RL_API_SENS_CHIRP_PROF_TIME_CFG profileTimeCfg;
00012 extern T_RL_API_FECSS_RF_PWR_CFG_CMD channelCfg;
00013 extern T_RL_API_SENS_FRAME_CFG frameCfg;
00014
00015 extern MMWave_Handle gCtrlHandle;
00016 extern T_RL_API_FECSS_RUNTIME_TX_CLPC_CAL_CMD fecTxclpcCalCmd;
00017
00018 typedef struct Mmw_calibData_t
00019 {
00020     uint32_t magic;
00021
00022     T_RL_API_FECSS_RXTX_CAL_DATA calibData;
00023 } Mmw_calibData;
00024
00025 int32_t restoreFactoryCal(void);
00026
00027 #endif //FACTORY_CAL_H
```

4.7 include/mem_pool.h File Reference

```
#include <stdint.h>
#include <stddef.h>
```

Data Structures

- struct [DPC_ObjectDetection_MemCfg_t](#)
Memory Configuration used during init API.
- struct [MemPoolObj_t](#)
Memory pool object to manage memory based on [DPC_ObjectDetection_MemCfg_t](#).

Macros

- #define [MEM_ALIGN](#)(addr, align)

Typedefs

- typedef struct [DPC_ObjectDetection_MemCfg_t](#) [DPC_ObjectDetection_MemCfg](#)
Memory Configuration used during init API.
- typedef struct [MemPoolObj_t](#) [MemPoolObj](#)
Memory pool object to manage memory based on [DPC_ObjectDetection_MemCfg_t](#).

Functions

- void [DPC_ObjDet_MemPoolReset](#) ([MemPoolObj](#) *pool)
- uint32_t [DPC_ObjDet_MemPoolGetMaxUsage](#) ([MemPoolObj](#) *pool)
- void * [DPC_ObjDet_MemPoolAlloc](#) ([MemPoolObj](#) *pool, uint32_t size, uint8_t align)

4.7.1 Detailed Description

This header defines Memory Pool management functions and datatypes.

4.7.2 Macro Definition Documentation

4.7.2.1 MEM_ALIGN

```
#define MEM_ALIGN(  
    addr,  
    align)
```

Value:

```
((addr) + ((align)-1)) & ~((align)-1))
```

4.7.3 Function Documentation

4.7.3.1 DPC_ObjDet_MemPoolAlloc()

```
void * DPC_ObjDet_MemPoolAlloc (  
    MemPoolObj * pool,  
    uint32_t size,  
    uint8_t align)
```

Description

Utility function for allocating from a static memory pool.

Parameters

in	<i>pool</i>	Handle to pool object.
in	<i>size</i>	Size in bytes to be allocated.
in	<i>align</i>	Alignment in bytes

Return values

<i>pointer</i>	to beginning of allocated block. NULL indicates could not allocate.
----------------	---

4.7.3.2 DPC_ObjDet_MemPoolGetMaxUsage()

```
uint32_t DPC_ObjDet_MemPoolGetMaxUsage (  
    MemPoolObj * pool)
```

Description

Utility function for getting maximum memory pool usage.

Parameters

in	<i>pool</i>	Handle to pool object.
----	-------------	------------------------

Return values

<i>Amount</i>	of pool used in bytes.
---------------	------------------------

4.7.3.3 DPC_ObjDet_MemPoolReset()

```
void DPC_ObjDet_MemPoolReset (  
    MemPoolObj * pool)
```

Description

Utility function for resetting memory pool.

Parameters

in	<i>pool</i>	Handle to pool object.
----	-------------	------------------------

Return values

<i>none.</i>	
--------------	--

4.8 mem_pool.h

[Go to the documentation of this file.](#)

```

00001 #ifndef MEM_POOL_H
00002 #define MEM_POOL_H
00003
00010
00011
00012
00013 #include <stdint.h>
00014 #include <stddef.h>
00015
00016 /* Macro for alignment */
00017 #define MEM_ALIGN(addr, align) (((addr) + ((align)-1)) & ~((align)-1))
00018 // #define MAX(a, b) ((a) > (b) ? (a) : (b))
00019
00023 typedef struct DPC_ObjectDetection_MemCfg_t
00024 {
00028     void *addr;
00029
00031     uint32_t size;
00032 } DPC_ObjectDetection_MemCfg;
00033
00037 typedef struct MemPoolObj_t
00038 {
00040     DPC_ObjectDetection_MemCfg cfg;
00041
00043     uintptr_t currAddr;
00044
00049     uintptr_t maxCurrAddr;
00050 } MemPoolObj;
00051
00064 void DPC_ObjDet_MemPoolReset(MemPoolObj *pool);
00065
00080 static void DPC_ObjDet_MemPoolSet(MemPoolObj *pool, void *addr);
00081
00095 static void *DPC_ObjDet_MemPoolGet(MemPoolObj *pool);
00096
00109 uint32_t DPC_ObjDet_MemPoolGetMaxUsage(MemPoolObj *pool);
00110
00126 void *DPC_ObjDet_MemPoolAlloc(MemPoolObj *pool, uint32_t size, uint8_t align);
00127
00128 #endif /* MEM_POOL_H */

```

4.9 mmwave_basic.h

```

00001 #ifndef MMWAVE_BASIC_H
00002 #define MMWAVE_BASIC_H
00003
00004 /*
00005  * Copyright (C) 2022-24 Texas Instruments Incorporated
00006  *
00007  * Redistribution and use in source and binary forms, with or without
00008  * modification, are permitted provided that the following conditions
00009  * are met:
00010  *
00011  *   Redistributions of source code must retain the above copyright
00012  *   notice, this list of conditions and the following disclaimer.
00013  *
00014  *   Redistributions in binary form must reproduce the above copyright
00015  *   notice, this list of conditions and the following disclaimer in the
00016  *   documentation and/or other materials provided with the
00017  *   distribution.
00018  *
00019  *   Neither the name of Texas Instruments Incorporated nor the names of
00020  *   its contributors may be used to endorse or promote products derived
00021  *   from this software without specific prior written permission.
00022  *
00023  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
00024  * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
00025  * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
00026  * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
00027  * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00028  * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
00029  * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
00030  * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
00031  * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
00032  * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
00033  * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00034  */

```

```

00035
00036 #include <control/mmwave/mmwave.h>
00037 #include <kernel/dpl/DebugP.h>
00038 #include <kernel/dpl/SystemP.h>
00039 #include <string.h>
00040
00041 #include "drivers/hwa.h"
00042 #include "kernel/dpl/SystemP.h"
00043 #include "ti_drivers_open_close.h"
00044 #include "ti_board_open_close.h"
00045 #include "kernel/dpl/DebugP.h"
00046 #include "defines.h"
00047 #include "mem_pool.h"
00048
00049 HWA_Handle hwaHandle;
00050
00052 MMWave_Handle gCtrlHandle;
00053
00055 MMWave_OpenCfg mmwOpenCfg;
00056
00058 MMWave_CtrlCfg mmwCtrlCfg;
00059
00061 MMWave_StrtCfg sensorStartCfg;
00062
00064 MemPoolObj    L3RamObj;
00065
00067 MemPoolObj    CoreLocalRamObj;
00068
00075
00079 void mempool_init(void);
00080
00084 int32_t hwa_open_handler(void);
00085
00089 int32_t mmwave_initSensor(void);
00090
00094 int32_t mmwave_openSensor(void);
00095
00099 int32_t mmwave_configSensor(void);
00100
00104 int32_t mmwave_startSensor(void);
00105
00109 int32_t mmwave_stop_close_deinit(void);
00110
00111 void Mmwave_HwaConfig_custom (void);
00112
00113 #endif //MMWAVE_BASIC_H

```

4.10 mmwave_control_config.h

```

00001 #ifndef MMWAVE_CONTROL_CONFIG_H
00002 #define MMWAVE_CONTROL_CONFIG_H
00003
00004
00008 typedef struct
00009 {
00010     uint32_t StartFreqHighRes[4]; /* LUT address 0 */
00011     uint32_t StartFreqLowRes[4]; /* LUT address 16 */
00012     int16_t ChirpSlope[4]; /* LUT address 32 */
00013     uint16_t ChirpIdleTime[4]; /* LUT address 40 */
00014     uint16_t ChirpAdcStartTime[4]; /* LUT address 48 */
00015     int16_t ChirpTxStartTime[4]; /* LUT address 56 */
00016     uint8_t ChirpTxEn[4]; /* LUT address 64 */
00017     uint8_t ChirpBpmEn[4]; /* LUT address 68 */
00018 } T_SensPerChirpLut;
00019
00020
00021 extern MMWave_Handle gCtrlHandle;
00022 extern T_SensPerChirpLut* sensPerChirpLuTable;
00023
00024 T_RL_API_SENS_CHIRP_PROF_COMN_CFG profileComCfg;
00025 T_RL_API_SENS_CHIRP_PROF_TIME_CFG profileTimeCfg;
00026 T_RL_API_FECSS_RF_PWR_CFG_CMD channelCfg;
00027 T_RL_API_SENS_FRAME_CFG frameCfg;
00028
00029
00030
00031 static void Mmwave_populateDefaultProfileCfg (T_RL_API_SENS_CHIRP_PROF_COMN_CFG* ptrProfileCfg,
T_RL_API_SENS_CHIRP_PROF_TIME_CFG* ptrProfileTimeCfg);
00032 static void Mmwave_populateDefaultChirpCfg (T_RL_API_SENS_PER_CHIRP_CFG* ptrChirpCfg,
T_RL_API_SENS_PER_CHIRP_CTRL* ptrChirpCtrl);
00033 void MMWave_populateChannelCfg();
00034 void Mmwave_populateDefaultCalibrationCfg (MMWave_CalibrationCfg* ptrCalibrationCfg);
00035 void Mmwave_populateDefaultStartCfg (MMWave_StrtCfg* ptrStartCfg);

```

```

00036 void Mmwave_populateDefaultOpenCfg (MMWave_OpenCfg* ptrOpenCfg);
00037 void Mmwave_populateDefaultChirpControlCfg (MMWave_CtrlCfg* ptrCtrlCfg);
00038
00039 #endif /* MMWAVE_CONTROL_CONFIG_H */

```

4.11 include/rangeproc_dpc.h File Reference

Range Processing Data Path Unit (DPU) Interface.

```

#include <stdint.h>
#include <kernel/dpl/DebugP.h>
#include "kernel/dpl/SemaphoreP.h"
#include "ti_drivers_config.h"
#include "ti_drivers_open_close.h"
#include "ti_board_open_close.h"
#include <datapath/dpu/rangeproc/v0/rangeprochwa.h>
#include "drivers/hwa.h"
#include "dpu_res.h"
#include "defines.h"

```

Macros

- #define **DPC_OBJDET_QFORMAT_RANGE_FFT** 17

Functions

- void [rangeProc_dpulnit](#) (void)
Initializes the Range Processing DPU.
- void [RangeProc_config](#) ()
Configures the Range Processing DPU.
- void [rangeproc_main](#) (void *args)
Main function for Range Processing DPU.
- void [dpcTask](#) ()
Task function for Data Processing Chain (DPC).
- void [uartTask](#) ()
Task function for UART transmission.
- int32_t [registerFrameStartInterrupt](#) (void)
Registers an interrupt for the frame start event.
- int32_t [registerChirpInterrupt](#) (void)
Registers the Chirp Interrupt.
- void [chirpStartISR](#) (void *arg)
ISR for Chirp Start event.
- uint32_t **Cycleprofiler_getTimeStamp** (void)
- int32_t [registerChirpAvailableInterrupts](#) (void)
Registers the Chirp Available Interrupt.

Variables

- SemaphoreP_Object **dpcCfgDoneSemHandle**
- SemaphoreP_Object **uart_tx_start_sem**
- SemaphoreP_Object **uart_tx_done_sem**
- DPU_RangeProchWA_Handle [rangeProchWADpuHandle](#)

Handle for Range Processing DPU.

4.11.1 Detailed Description

Range Processing Data Path Unit (DPU) Interface.

This file defines the interface for the Range Processing Data Path Unit (DPU) and its associated functions. It includes declarations for DPU initialization, configuration, and task management, as well as interrupt handling for frame start, chirp start, and chirp available events. The module is designed to facilitate radar signal processing using the Hardware Accelerator (HWA) and UART communication for data transmission.

The range processing DPU is responsible for processing radar data frames, including FFT computation, object detection, and data transmission over UART. This module integrates with the TI mmWave SDK and is derived from the Motion and Presence Detection Demo provided in the SDK.

Note

This module relies on the SemaphoreP API for synchronization and the HWA for hardware-accelerated signal processing.

Copyright

(C) 2022-24 Texas Instruments Incorporated

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the name of Texas Instruments Incorporated nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

4.11.2 Function Documentation

4.11.2.1 chirpStartISR()

```
void chirpStartISR (  
    void * arg)
```

ISR for Chirp Start event.

This ISR is triggered when the chirp start event occurs, indicating that the RF time control is functioning correctly. It does not necessarily mean the RF is actively chirping.

Parameters

<i>arg</i>	Unused optional argument.
------------	---------------------------

Description

Chirp Start ISR

4.11.2.2 dpcTask()

```
void dpcTask ()
```

Task function for Data Processing Chain (DPC).

This function initializes and manages the data processing chain, including configuring the DPUs, registering interrupts, and triggering UART transmission. It runs in an infinite loop, processing data frames and triggering the next frame.

4.11.2.3 RangeProc_config()

```
void RangeProc_config ()
```

Configures the Range Processing DPU.

This function configures the range processing DPU (Data Path Unit) and the HWA. It is derived from the dpc.c of the mmwavedemo project.

Return values

<i>None</i>	
-------------	--

4.11.2.4 rangeProc_dpuInit()

```
void rangeProc_dpuInit (
    void )
```

Initializes the Range Processing DPU.

This function initializes the range processing DPU (Data Path Unit) and sets up the required handles for HWA (Hardware Accelerator) and DPU. It is derived from the dpc.c of the mmwavedemo project.

Return values

<i>None</i>	
-------------	--

4.11.2.5 rangeproc_main()

```
void rangeproc_main (
    void * args)
```

Main function for Range Processing DPU.

This function opens the necessary drivers, initializes the HWA, calls the range processing DPU initialization, and closes all drivers after execution. It is the entry point for the range processing task.

Parameters

<i>in</i>	<i>args</i>	Arguments passed to the function (if any).
-----------	-------------	--

Return values

<i>None</i>	
-------------	--

4.11.2.6 registerChirpAvailableInterrupts()

```
int32_t registerChirpAvailableInterrupts (
    void )
```

Registers the Chirp Available Interrupt.

This function registers the interrupt for the chirp available event.

Returns

int32_t Returns SystemP_SUCCESS on success, SystemP_FAILURE on failure.

Description

This is to register Chirp Available Interrupt

4.11.2.7 registerChirpInterrupt()

```
int32_t registerChirpInterrupt (
    void )
```

Registers the Chirp Interrupt.

This function registers the interrupt for the chirp start and end events.

Returns

int32_t Returns SystemP_SUCCESS on success, SystemP_FAILURE on failure.

Description

This is to register Chirpt Interrupt

4.11.2.8 registerFrameStartInterrupt()

```
int32_t registerFrameStartInterrupt (
    void )
```

Registers an interrupt for the frame start event.

This function configures and registers the interrupt handler for the frame start event, which is triggered by the frame timer. It sets up the interrupt priority and callback function, then enables the interrupt to trigger the corresponding handler. Copied from \${MMWAVE_SDK_INSTALL_PATH}\examples\mmw_demo\motion_and_↵ presence_detection\source\interrupts.c

Parameters

None	
------	--

Return values

<i>SystemP_SUCCESS</i>	on successful registration, SystemP_FAILURE on error.
------------------------	---

4.11.2.9 uartTask()

```
void uartTask ()
```

Task function for UART transmission.

This function continuously transmits data over UART.

4.11.3 Variable Documentation**4.11.3.1 rangeProcHWADpuHandle**

```
DPU_RangeProcHWA_Handle rangeProcHWADpuHandle [extern]
```

Handle for Range Processing DPU.

Description

This header file defines the interface for range processing DPU initialization and related functions.

4.12 rangeproc_dpc.h

[Go to the documentation of this file.](#)

```
00001 #ifndef RANGEPROC_DPC_H
00002 #define RANGEPROC_DPC_H
00003
00052
00053 #include <stdint.h>
00054 #include <kernel/dpl/DebugP.h>
00055 #include "kernel/dpl/SemaphoreP.h"
00056 #include "ti_drivers_config.h"
00057 #include "ti_drivers_open_close.h"
00058 #include "ti_board_open_close.h"
00059 #include <datapath/dpu/rangeproc/v0/rangeprochwa.h>
00060 #include "drivers/hwa.h"
00061
00062 #include "dpu_res.h"
00063 #include "defines.h"
00064
00065 #define DPC_OBJDET_QFORMAT_RANGE_FFT 17
00066
00067 extern SemaphoreP_Object dpcCfgDoneSemHandle;
00068 extern SemaphoreP_Object uart_tx_start_sem;
00069 extern SemaphoreP_Object uart_tx_done_sem;
00070
00077
00081 extern DPU_RangeProcHWA_Handle rangeProcHWADpuHandle;
00082
00092 void rangeProc_dpuInit(void);
00093
00102 void RangeProc_config();
00103
```

```

00115 void rangeproc_main(void *args);
00116
00124 void dpcTask();
00125
00131 void uartTask();
00132
00145 int32_t registerFrameStartInterrupt(void);
00146
00155 static void frameStartISR(void *arg);
00156
00164 int32_t registerChirpInterrupt(void);
00165
00173 void chirpStartISR(void *arg);
00174
00175 uint32_t Cycleprofiler_getTimeStamp(void);
00176
00184 int32_t registerChirpAvailableInterrupts(void);
00185 static void ChirpAvailISR(void *arg);
00186
00187
00188 #endif /* RANGEPROC_DPC_H */

```

4.13 include/uart_transmit.h File Reference

UART Transmission Interface for Radar Data.

```
#include "kernel/dpl/SemaphoreP.h"
```

Functions

- void [uart_transmit_loop](#) ()
UART transmission loop function.

Variables

- SemaphoreP_Object [uart_tx_start_sem](#)
Semaphore to signal the start of UART transmission.
- SemaphoreP_Object [uart_tx_done_sem](#)
Semaphore to signal the completion of UART transmission.

4.13.1 Detailed Description

UART Transmission Interface for Radar Data.

This file defines the interface for transmitting radar cube data over UART. It includes the declaration of semaphores used to synchronize the UART transmission process and the function prototype for the UART transmission loop.

The UART transmission loop waits for a signal to start transmission, sends radar cube data over UART, and signals completion when done. This module is designed to facilitate communication between the radar processing unit and external systems via UART.

Note

This module relies on the SemaphoreP API from the kernel/dpl library for synchronization.

4.13.2 Function Documentation

4.13.2.1 `uart_transmit_loop()`

```
void uart_transmit_loop ()
```

UART transmission loop function.

This function continuously waits for a signal to start UART transmission, sends radar cube data over UART, and signals completion when done.

4.13.3 Variable Documentation

4.13.3.1 `uart_tx_done_sem`

```
SemaphoreP_Object uart_tx_done_sem [extern]
```

Semaphore to signal the completion of UART transmission.

This semaphore is posted when the UART transmission is complete.

4.13.3.2 `uart_tx_start_sem`

```
SemaphoreP_Object uart_tx_start_sem [extern]
```

Semaphore to signal the start of UART transmission.

This semaphore is used to trigger the UART transmission process.

4.14 `uart_transmit.h`

[Go to the documentation of this file.](#)

```
00001 #ifndef UART_TRANSMIT_H
00002 #define UART_TRANSMIT_H
00003
00020
00021 #include "kernel/dpl/SemaphoreP.h"
00022
00028 extern SemaphoreP_Object uart_tx_start_sem;
00029
00035 extern SemaphoreP_Object uart_tx_done_sem;
00036
00043 void uart_transmit_loop();
00044
00045 #endif /* UART_TRANSMIT_H */
```

4.15 src/factory_cal.c File Reference

Factory Calibration Restoration and Configuration.

```
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include "kernel/dpl/DebugP.h"
#include "kernel/dpl/SystemP.h"
#include "control/mmwave/mmwave.h"
#include "ti_board_open_close.h"
#include <board/flash.h>
#include <mmwavelink/mmwavelink.h>
#include <mmwavelink/include/rl_device.h>
#include <mmwavelink/include/rl_sensor.h>
#include <kernel/dpl/CacheP.h>
#include "factory_cal.h"
#include "mmwave_basic.h"
#include "mmwave_control_config.h"
#include "defines.h"
```

Functions

- [Mmw_calibData](#) calibData **__attribute__** ((aligned(8)))
- [int32_t restoreFactoryCal](#) (void)

Restores factory calibration data from flash.

4.15.1 Detailed Description

Factory Calibration Restoration and Configuration.

This file implements the functionality to restore factory calibration data for the radar system. It reads calibration data from flash memory, validates its integrity using a magic number, and configures the radar front-end (FECSS) with the restored calibration parameters. The calibration process ensures optimal performance of the radar system by compensating for manufacturing variations and environmental factors.

Note

This function in this file is adapted from the Motion and Presence Detection Demo: `$(MMWAVE_SDK_↵
INSTALL_PATH)\examples\mmw_demo\motion_and_presence_detection\source\calibration\factory_cal.c`

Copyright

(C) 2022-24 Texas Instruments Incorporated

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the name of Texas Instruments Incorporated nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

4.15.2 Function Documentation

4.15.2.1 `restoreFactoryCal()`

```
int32_t restoreFactoryCal (
    void )
```

Restores factory calibration data from flash.

This function reads the calibration data stored in flash memory and restores it. It requires that the system has been previously calibrated.

Derived from `mmwDemo_factoryCal` and `MmwDemo_calibRestore` in [factory_cal.c](#) from the demo project.

Returns

`SystemP_SUCCESS` on success, -1 on failure.

4.16 src/main.c File Reference

Main Program for Radar Signal Processing.

```
#include <stdlib.h>
#include <kernel/dpl/DebugP.h>
#include "board/flash.h"
#include "drivers/uart/v0/uart_sci.h"
#include "kernel/dpl/SystemP.h"
#include "ti_drivers_config.h"
#include "ti_board_config.h"
#include "ti_drivers_open_close.h"
#include "FreeRTOS.h"
#include "task.h"
#include <mmwavelink/mmwavelink.h>
#include <mmwavelink/include/rl_device.h>
#include <kernel/dpl/SemaphoreP.h>
#include <datapath/dpu/rangeproc/v0/rangeprochwa.h>
#include "rangeproc_dpc.h"
#include "mmwave_basic.h"
#include "mmwave_control_config.h"
#include "factory_cal.h"
```

Macros

- #define **MAIN_TASK_PRI** 1
- #define **MAIN_TASK_SIZE** (16384U/sizeof(configSTACK_DEPTH_TYPE))
- #define **DPC_TASK_STACK_SIZE** 8192
- #define **UART_TASK_STACK_SIZE** 2048
- #define **DPC_TASK_PRI** 5
- #define **UART_TASK_PRI** 10

Functions

- StackType_t gMainTaskStack[MAIN_TASK_SIZE] **__attribute__** ((aligned(32)))
- void **rangeproc_main** (void *args)
- void **freertos_main** (void *args)
- int **main** ()

Variables

- StaticTask_t **gMainTaskObj**
- TaskHandle_t **gMainTask**
- StaticTask_t **gDpcTaskObj**
- TaskHandle_t **gDpcTask**
- StaticTask_t **gUartTaskObj**
- TaskHandle_t **gUartTask**
- T_RL_API_FECSS_RUNTIME_TX_CLPC_CAL_CMD **fecTxclpcCalCmd**
- SemaphoreP_Object **pend_main_sem**
- SemaphoreP_Object **dpcCfgDoneSemHandle**
- SemaphoreP_Object **uart_tx_start_sem**
Semaphore to signal the start of UART transmission.
- SemaphoreP_Object **uart_tx_done_sem**
Semaphore to signal the completion of UART transmission.
- T_RL_API_FECSS_RF_PWR_CFG_CMD **channelCfg**

4.16.1 Detailed Description

Main Program for Radar Signal Processing.

This file contains the main entry point and initialization logic for the radar signal processing application. It sets up the hardware, initializes the radar sensor, configures the Data Processing Units (DPUs), and starts the FreeRTOS scheduler to manage tasks for radar processing and UART communication.

The application performs the following key steps:

1. Initializes the hardware and drivers.
2. Configures the radar sensor and performs factory calibration.
3. Initializes the DPUs for range processing.
4. Creates FreeRTOS tasks for radar processing (DPC) and UART communication.
5. Starts the radar sensor and enters the FreeRTOS scheduler.

This implementation is adapted from the Motion and Presence Detection Demo provided in the TI mmWave SDK.

Note

This module relies on the TI mmWave SDK, FreeRTOS, and various hardware drivers for radar sensor control, DPU configuration, and UART communication.

Copyright

(C) 2022-24 Texas Instruments Incorporated

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the name of Texas Instruments Incorporated nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

4.16.2 Variable Documentation

4.16.2.1 uart_tx_done_sem

SemaphoreP_Object uart_tx_done_sem

Semaphore to signal the completion of UART transmission.

This semaphore is posted when the UART transmission is complete.

4.16.2.2 uart_tx_start_sem

SemaphoreP_Object uart_tx_start_sem

Semaphore to signal the start of UART transmission.

This semaphore is used to trigger the UART transmission process.

4.17 src/mem_pool.c File Reference

```
#include "mem_pool.h"
```

Functions

- void [DPC_ObjDet_MemPoolReset](#) ([MemPoolObj](#) *pool)
- uint32_t [DPC_ObjDet_MemPoolGetMaxUsage](#) ([MemPoolObj](#) *pool)
- void * [DPC_ObjDet_MemPoolAlloc](#) ([MemPoolObj](#) *pool, uint32_t size, uint8_t align)

4.17.1 Detailed Description

This file implements Memory Pool management functions and datatypes.

4.17.2 Function Documentation

4.17.2.1 DPC_ObjDet_MemPoolAlloc()

```
void * DPC_ObjDet_MemPoolAlloc (  
    MemPoolObj * pool,  
    uint32_t size,  
    uint8_t align)
```

Description

Utility function for allocating from a static memory pool.

Parameters

in	<i>pool</i>	Handle to pool object.
in	<i>size</i>	Size in bytes to be allocated.
in	<i>align</i>	Alignment in bytes

Return values

<i>pointer</i>	to beginning of allocated block. NULL indicates could not allocate.
----------------	---

4.17.2.2 DPC_ObjDet_MemPoolGetMaxUsage()

```
uint32_t DPC_ObjDet_MemPoolGetMaxUsage (  
    MemPoolObj * pool)
```

Description

Utility function for getting maximum memory pool usage.

Parameters

in	<i>pool</i>	Handle to pool object.
----	-------------	------------------------

Return values

<i>Amount</i>	of pool used in bytes.
---------------	------------------------

4.17.2.3 DPC_ObjDet_MemPoolReset()

```
void DPC_ObjDet_MemPoolReset (  
    MemPoolObj * pool)
```

Description

Utility function for resetting memory pool.

Parameters

in	<i>pool</i>	Handle to pool object.
----	-------------	------------------------

Return values

<i>none.</i>	
--------------	--

4.18 src/mmwave_basic.c File Reference

Initialization, configuration, and control functions for mmWave sensor.

```
#include "drivers/hwa.h"
#include "kernel/dpl/SystemP.h"
#include "ti_drivers_open_close.h"
#include "ti_board_open_close.h"
#include "kernel/dpl/DebugP.h"
#include "defines.h"
#include "mem_pool.h"
#include "mmwave_basic.h"
```

Macros

- #define **L3_MEM_SIZE** (0x40000 + 160*1024)
L3 RAM buffer for object detection DPC.
- #define **MMWDEMO_OBJDET_CORE_LOCAL_MEM_SIZE** ((8U+6U+4U+2U+8U) * 1024U)
Local RAM buffer for object detection DPC.

Functions

- void [Mmwave_populateDefaultOpenCfg](#) (MMWave_OpenCfg *ptrOpenCfg)
- void [Mmwave_populateDefaultChirpControlCfg](#) (MMWave_CtrlCfg *ptrCtrlCfg)
- void [Mmwave_populateDefaultCalibrationCfg](#) (MMWave_CalibrationCfg *ptrCalibrationCfg)
- void [Mmwave_populateDefaultStartCfg](#) (MMWave_StrtCfg *ptrStartCfg)
- uint8_t gMmwL3[L3_MEM_SIZE] **__attribute** ((section(".I3")))
- void [mempool_init](#) (void)
sets start address and size of shared mempool
- int32_t [hwa_open_handler](#) ()
calls HWA_open() and thereby retrieves the HWA's handle (HWA_Handle)
- int32_t [mmwave_initSensor](#) ()
calls the MMWave_init() function and thereby retrieves the mmwave control handle (MMWave_Handle)
- int32_t [mmwave_openSensor](#) (void)
calls the MMWave_open() function, requires the configuration for open (MMWave_OpenCfg)
- int32_t [mmwave_configSensor](#) (void)
calls the MMWave_config() function, requires the configuration for control (MMWave_CtrlCfg)
- int32_t [mmwave_startSensor](#) (void)
calls the MMWave_start() function, requires the configuration for start (MMWave_StartCfg)
- int32_t [mmwave_stop_close_deinit](#) (void)
calls the MMWave_stop(), MMWave_close() and MMWave_deinit() function

Variables

- uint8_t gMmwCoreLocMem [MMWDEMO_OBJDET_CORE_LOCAL_MEM_SIZE]

4.18.1 Detailed Description

Initialization, configuration, and control functions for mmWave sensor.

This file contains functions to initialize, open, configure, start, stop, and deinitialize the mmWave sensor. It also includes memory pool setup and HWA initialization.

Note

Functionality derived from motion_and_presence_detection_demo_xwrL64xx-evm_m4fss0-0_freertos_↔
ti-arm-clang.

- Provides functions for mmWave sensor control.
- Includes memory pool initialization for object detection processing.
- Handles HWA (Hardware Accelerator) initialization.
- Implements calibration restore and validation using stored magic word.

Copyright

(C) 2022-24 Texas Instruments Incorporated

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the name of Texas Instruments Incorporated nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

4.18.2 Function Documentation

4.18.2.1 mempool_init()

```
void mempool_init (  
    void )
```

sets start address and size of shared mempool

Description

This header file defines the interface for the basic mmwave initialization functionality on the basis of which the DPUs can be used.

4.18.2.2 Mmwave_populateDefaultCalibrationCfg()

```
void Mmwave_populateDefaultCalibrationCfg (  
    MMWave_CalibrationCfg * ptrCalibrationCfg)    [extern]
```

Description

The function is used to populate the default calibration configuration which is passed to start the mmWave module

Return values

<i>Not</i>	applicable
------------	------------

4.18.2.3 Mmwave_populateDefaultChirpControlCfg()

```
void Mmwave_populateDefaultChirpControlCfg (  
    MMWave_CtrlCfg * ptrCtrlCfg) [extern]
```

Description

The function is used to populate the default control configuration in chirp configuration mode

Parameters

out	<i>ptrCtrlCfg</i>	Pointer to the control configuration
-----	-------------------	--------------------------------------

Return values

<i>Not</i>	applicable
------------	------------

4.18.2.4 Mmwave_populateDefaultOpenCfg()

```
void Mmwave_populateDefaultOpenCfg (  
    MMWave_OpenCfg * ptrOpenCfg) [extern]
```

Description

The function is used to populate the default open configuration.

Parameters

out	<i>ptrOpenCfg</i>	Pointer to the open configuration
-----	-------------------	-----------------------------------

Return values

<i>Not</i>	applicable
------------	------------

4.18.2.5 Mmwave_populateDefaultStartCfg()

```
void Mmwave_populateDefaultStartCfg (  
    MMWave_StrtCfg * ptrStartCfg) [extern]
```

Description

The function is used to populate the default start configuration which is passed to start the mmWave module

Return values

Not	applicable
-----	------------

4.19 src/mmwave_control_config.c File Reference

Configuration and initialization for mmWave chirp profiles.

```
#include <stdint.h>
#include <stdlib.h>
#include <stddef.h>
#include <string.h>
#include <stdio.h>
#include <math.h>
#include <drivers/hw_include/cslr_adcbuf.h>
#include <drivers/hw_include/xwrL64xx/cslr_soc_baseaddress.h>
#include <control/mmwave/mmwave.h>
#include <kernel/dpl/DebugP.h>
#include <utils/testlogger/logger.h>
#include "defines.h"
#include "common/sys_defs.h"
#include "mmwave_control_config.h"
```

Macros

- `#define DebugP_LOG_ENABLED 1`
- `#define RDIF_LANE_RATE_UPDATE 1`

Functions

- void [Mmwave_populateDefaultOpenCfg](#) (MMWave_OpenCfg *ptrOpenCfg)
- void [MMWave_populateChannelCfg](#) ()
Populates the channel configuration structure.
- void [Mmwave_populateDefaultChirpControlCfg](#) (MMWave_CtrlCfg *ptrCtrlCfg)
- void [Mmwave_populateDefaultCalibrationCfg](#) (MMWave_CalibrationCfg *ptrCalibrationCfg)
- void [Mmwave_populateDefaultStartCfg](#) (MMWave_StrtCfg *ptrStartCfg)

Variables

- MMWave_Handle **gCtrlHandle**
This is the mmWave control handle which is used to configure the BSS.
- T_RL_API_FECSS_RUNTIME_TX_CLPC_CAL_CMD **fecTxclpcCalCmd**
- T_SensPerChirpLut * **sensPerChirpLuTable** = (T_SensPerChirpLut*)(0x21880000U)

4.19.1 Detailed Description

Configuration and initialization for mmWave chirp profiles.

This file contains functions for setting up default chirp and profile configurations for the Texas Instruments mmWave radar system. It includes parameter definitions, initialization routines, and structure setup for mmWave control.

This file is a modified version of the original from the motion and presence detection demo project. It has been adjusted to fit the current application needs while maintaining compatibility with TI's mmWave SDK.

Copyright

Copyright (C) 2022-24 Texas Instruments Incorporated

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the name of Texas Instruments Incorporated nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

4.19.2 Function Documentation

4.19.2.1 MMWave_populateChannelCfg()

```
void MMWave_populateChannelCfg ()
```

Populates the channel configuration structure.

Returns

None

4.19.2.2 Mmwave_populateDefaultCalibrationCfg()

```
void Mmwave_populateDefaultCalibrationCfg (  
    MMWave_CalibrationCfg * ptrCalibrationCfg)
```

Description

The function is used to populate the default calibration configuration which is passed to start the mmWave module

Return values

<i>Not</i>	applicable
------------	------------

4.19.2.3 Mmwave_populateDefaultChirpControlCfg()

```
void Mmwave_populateDefaultChirpControlCfg (  
    MMWave_CtrlCfg * ptrCtrlCfg)
```

Description

The function is used to populate the default control configuration in chirp configuration mode

Parameters

out	<i>ptrCtrlCfg</i>	Pointer to the control configuration
-----	-------------------	--------------------------------------

Return values

<i>Not</i>	applicable
------------	------------

4.19.2.4 Mmwave_populateDefaultOpenCfg()

```
void Mmwave_populateDefaultOpenCfg (  
    MMWave_OpenCfg * ptrOpenCfg)
```

Description

The function is used to populate the default open configuration.

Parameters

out	<i>ptrOpenCfg</i>	Pointer to the open configuration
-----	-------------------	-----------------------------------

Return values

<i>Not</i>	applicable
------------	------------

4.19.2.5 Mmwave_populateDefaultStartCfg()

```
void Mmwave_populateDefaultStartCfg (  
    MMWave_StrtCfg * ptrStartCfg)
```

Description

The function is used to populate the default start configuration which is passed to start the mmWave module

Return values

Not	applicable
-----	------------

4.20 src/rangeproc_dpc.c File Reference

Implementation of the Range Processing DPC (Data Processing Chain)

```
#include <stdio.h>
#include <kernel/dpl/DebugP.h>
#include <utils/mathutils/mathutils.h>
#include "drivers/edma/v0/edma.h"
#include "kernel/dpl/SemaphoreP.h"
#include "ti_drivers_config.h"
#include "ti_drivers_open_close.h"
#include "ti_board_open_close.h"
#include "drivers/prcm/v0/prcm.h"
#include <datapath/dpu/rangeproc/v0/rangeprochwa.h>
#include "drivers/hwa.h"
#include "mmwave_basic.h"
#include "rangeproc_dpc.h"
#include "mem_pool.h"
#include "uart_transmit.h"
```

Functions

- void [uartTask](#) ()
Task function for UART transmission.
- void [dpcTask](#) ()
Task function for Data Processing Chain (DPC).
- void [rangeProc_dpulnit](#) ()
Initializes the Range Processing DPU.
- void [RangeProc_config](#) ()
Configures the Range Processing DPU.
- int32_t [registerChirpInterrupt](#) (void)
Registers the Chirp Interrupt.
- void [chirpStartISR](#) (void *arg)
ISR for Chirp Start event.
- int32_t [registerFrameStartInterrupt](#) (void)
Registers an interrupt for the frame start event.
- uint32_t [Cycleprofiler_getTimeStamp](#) (void)
- int32_t [registerChirpAvailableInterrupts](#) (void)
Registers the Chirp Available Interrupt.

Variables

- DPU_RangeProcHWA_Handle [rangeProcHWADpuHandle](#)
Handle for Range Processing DPU.
- DPU_RangeProcHWA_Config **rangeProcDpuCfg**
- HwiP_Object **gHwiChirpAvailableHwiObject**
- HwiP_Object **gHwiFrameStartHwiObject**
- Edma_IntrObject **intrObj_Rangeproc** [2]
- volatile unsigned long long **demoStartTime**
- [MemPoolObj](#) **L3RamObj**
L3 ram memory pool object.
- [MemPoolObj](#) **CoreLocalRamObj**
Core Local ram memory pool object.
- HWA_Handle **hwaHandle**
- cmplx16ImRe_t * **gRadarCubeDebugPtr** = NULL
Pointer to radar cube data for easier debugging access.
- void * **gAdcDataDebugPtr** = NULL
- uint32_t **gChirpCount**
- uint32_t **gFrameCount**

4.20.1 Detailed Description

Implementation of the Range Processing DPC (Data Processing Chain)

This file contains the core functions for range processing in the radar signal chain. It manages FFT processing, data handling, and hardware acceleration interfaces.

This file is a modified version of the original from the motion and presence detection demo project. It has been adjusted to fit the current application needs while maintaining compatibility with TI's mmWave SDK.

Copyright

Copyright (C) 2022-24 Texas Instruments Incorporated

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the name of Texas Instruments Incorporated nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

4.20.2 Function Documentation

4.20.2.1 chirpStartISR()

```
void chirpStartISR (  
    void * arg)
```

ISR for Chirp Start event.

Description

Chirp Start ISR

4.20.2.2 dpcTask()

```
void dpcTask ()
```

Task function for Data Processing Chain (DPC).

This function initializes and manages the data processing chain, including configuring the DPUs, registering interrupts, and triggering UART transmission. It runs in an infinite loop, processing data frames and triggering the next frame.

4.20.2.3 RangeProc_config()

```
void RangeProc_config ()
```

Configures the Range Processing DPU.

This function configures the range processing DPU (Data Path Unit) and the HWA. It is derived from the dpc.c of the mmwavedemo project.

Return values

None	
------	--

4.20.2.4 rangeProc_dpuInit()

```
void rangeProc_dpuInit (  
    void )
```

Initializes the Range Processing DPU.

This function initializes the range processing DPU (Data Path Unit) and sets up the required handles for HWA (Hardware Accelerator) and DPU. It is derived from the dpc.c of the mmwavedemo project.

Return values

None	
------	--

4.20.2.5 registerChirpAvailableInterrupts()

```
int32_t registerChirpAvailableInterrupts (
    void )
```

Registers the Chirp Available Interrupt.

Description

This is to register Chirp Available Interrupt

4.20.2.6 registerChirpInterrupt()

```
int32_t registerChirpInterrupt (
    void )
```

Registers the Chirp Interrupt.

Description

This is to register Chirpt Interrupt

4.20.2.7 registerFrameStartInterrupt()

```
int32_t registerFrameStartInterrupt (
    void )
```

Registers an interrupt for the frame start event.

This function configures and registers the interrupt handler for the frame start event, which is triggered by the frame timer. It sets up the interrupt priority and callback function, then enables the interrupt to trigger the corresponding handler. Copied from \${MMWAVE_SDK_INSTALL_PATH}\examples\mmw_demo\motion_and_↵presence_detection\source\interrupts.c

Parameters

None	
------	--

Return values

SystemP_SUCCESS	on successful registration, SystemP_FAILURE on error.
-----------------	---

4.20.2.8 uartTask()

```
void uartTask ()
```

Task function for UART transmission.

This function continuously transmits data over UART.

4.20.3 Variable Documentation

4.20.3.1 rangeProcHWADpuHandle

DPU_RangeProcHWA_Handle rangeProcHWADpuHandle

Handle for Range Processing DPU.

Description

This header file defines the interface for range processing DPU initialization and related functions.

4.21 src/uart_transmit.c File Reference

UART Transmission Implementation for Radar Data.

```
#include "ti_drivers_config.h"
#include <stdio.h>
#include "string.h"
#include <kernel/dpl/DebugP.h>
#include <utils/mathutils/mathutils.h>
#include "kernel/dpl/SemaphoreP.h"
#include "ti_drivers_open_close.h"
#include <datapath/dpu/rangeproc/v0/rangeprochwa.h>
#include "defines.h"
#include "uart_transmit.h"
```

Macros

- #define **APP_UART_BUFSIZE** (1024)
- #define **APP_UART_RECEIVE_BUFSIZE** (8U)

Functions

- void [uart_transmit_loop](#) ()
UART transmission loop function.

Variables

- uint8_t **gUartBuffer** [APP_UART_BUFSIZE]
- uint8_t **gUartReceiveBuffer** [APP_UART_RECEIVE_BUFSIZE]
- volatile uint32_t **gNumBytesRead** = 0U
- volatile uint32_t **gNumBytesWritten** = 0U
- const uint8_t **header** [4] = {0xAA, 0xBB, 0xCC, 0xDD}
Header and footer for each radarCube transmission over UART.
- const uint8_t **footer** [4] = {0xDD, 0xCC, 0xBB, 0xAA}
- DPU_RangeProcHWA_Config **rangeProcDpuCfg**

4.21.1 Detailed Description

UART Transmission Implementation for Radar Data.

This file implements the UART transmission of radar cube data. It manages synchronization using semaphores, waits for transmission signals, sends data over UART, and signals completion when done.

The transmission process is controlled using two semaphores:

- `uart_tx_start_sem`: Signals the start of transmission.
- `uart_tx_done_sem`: Signals the completion of transmission.

The function `uart_transmit_loop()` runs continuously, waiting for `uart_tx_start_sem` to be posted, transmitting radar cube data, and posting `uart_tx_done_sem` upon completion.

Note

This module relies on the SemaphoreP API from the kernel/dpl library for synchronization.

4.21.2 Function Documentation

4.21.2.1 `uart_transmit_loop()`

```
void uart_transmit_loop ()
```

UART transmission loop function.

This function continuously waits for a signal to start UART transmission, sends radar cube data over UART, and signals completion when done.

4.21.3 Variable Documentation

4.21.3.1 `header`

```
const uint8_t header[4] = {0xAA, 0xBB, 0xCC, 0xDD}
```

Header and footer for each radarCube transmission over UART.

These markers are used to indicate the start and end of each radarCube data packet during UART transmission.

Index

- chirpStartISR
 - rangeproc_dpc.c, [46](#)
 - rangeproc_dpc.h, [25](#)
- DPC_ObjDet_MemPoolAlloc
 - mem_pool.c, [35](#)
 - mem_pool.h, [20](#)
- DPC_ObjDet_MemPoolGetMaxUsage
 - mem_pool.c, [36](#)
 - mem_pool.h, [21](#)
- DPC_ObjDet_MemPoolReset
 - mem_pool.c, [36](#)
 - mem_pool.h, [21](#)
- DPC_ObjectDetection_MemCfg_t, [5](#)
- dpcTask
 - rangeproc_dpc.c, [46](#)
 - rangeproc_dpc.h, [26](#)
- factory_cal.c
 - restoreFactoryCal, [32](#)
- factory_cal.h
 - Mmw_calibData, [19](#)
 - MMWDEMO_CALIB_STORE_MAGIC, [18](#)
 - restoreFactoryCal, [19](#)
- header
 - uart_transmit.c, [49](#)
- include/defines.h, [9](#), [10](#)
- include/dpu_res.h, [11](#), [15](#)
- include/factory_cal.h, [17](#), [19](#)
- include/mem_pool.h, [20](#), [22](#)
- include/mmwave_basic.h, [22](#)
- include/mmwave_control_config.h, [23](#)
- include/rangeproc_dpc.h, [24](#), [28](#)
- include/uart_transmit.h, [29](#), [30](#)
- main.c
 - uart_tx_done_sem, [35](#)
 - uart_tx_start_sem, [35](#)
- MEM_ALIGN
 - mem_pool.h, [20](#)
- mem_pool.c
 - DPC_ObjDet_MemPoolAlloc, [35](#)
 - DPC_ObjDet_MemPoolGetMaxUsage, [36](#)
 - DPC_ObjDet_MemPoolReset, [36](#)
- mem_pool.h
 - DPC_ObjDet_MemPoolAlloc, [20](#)
 - DPC_ObjDet_MemPoolGetMaxUsage, [21](#)
 - DPC_ObjDet_MemPoolReset, [21](#)
 - MEM_ALIGN, [20](#)
- mempool_init
 - mmwave_basic.c, [38](#)
- MemPoolObj_t, [5](#)
- Mmw_calibData
 - factory_cal.h, [19](#)
- Mmw_calibData_t, [6](#)
- mmwave_basic.c
 - mempool_init, [38](#)
 - Mmwave_populateDefaultCalibrationCfg, [38](#)
 - Mmwave_populateDefaultChirpControlCfg, [40](#)
 - Mmwave_populateDefaultOpenCfg, [40](#)
 - Mmwave_populateDefaultStartCfg, [40](#)
- mmwave_control_config.c
 - MMWave_populateChannelCfg, [42](#)
 - Mmwave_populateDefaultCalibrationCfg, [42](#)
 - Mmwave_populateDefaultChirpControlCfg, [43](#)
 - Mmwave_populateDefaultOpenCfg, [43](#)
 - Mmwave_populateDefaultStartCfg, [43](#)
- MMWave_populateChannelCfg
 - mmwave_control_config.c, [42](#)
- Mmwave_populateDefaultCalibrationCfg
 - mmwave_basic.c, [38](#)
 - mmwave_control_config.c, [42](#)
- Mmwave_populateDefaultChirpControlCfg
 - mmwave_basic.c, [40](#)
 - mmwave_control_config.c, [43](#)
- Mmwave_populateDefaultOpenCfg
 - mmwave_basic.c, [40](#)
 - mmwave_control_config.c, [43](#)
- Mmwave_populateDefaultStartCfg
 - mmwave_basic.c, [40](#)
 - mmwave_control_config.c, [43](#)
- MMWDEMO_CALIB_STORE_MAGIC
 - factory_cal.h, [18](#)
- RangeProc_config
 - rangeproc_dpc.c, [46](#)
 - rangeproc_dpc.h, [26](#)
- rangeproc_dpc.c
 - chirpStartISR, [46](#)
 - dpcTask, [46](#)
 - RangeProc_config, [46](#)
 - rangeProc_dpulnit, [46](#)
 - rangeProcHWADpuHandle, [48](#)
 - registerChirpAvailableInterrupts, [46](#)
 - registerChirpInterrupt, [47](#)
 - registerFrameStartInterrupt, [47](#)
 - uartTask, [47](#)
- rangeproc_dpc.h
 - chirpStartISR, [25](#)

- dpcTask, [26](#)
- RangeProc_config, [26](#)
- rangeProc_dpulnit, [26](#)
- rangeproc_main, [26](#)
- rangeProcHWADpuHandle, [28](#)
- registerChirpAvailableInterrupts, [27](#)
- registerChirpInterrupt, [27](#)
- registerFrameStartInterrupt, [27](#)
- uartTask, [28](#)
- rangeProc_dpulnit
 - rangeproc_dpc.c, [46](#)
 - rangeproc_dpc.h, [26](#)
- rangeproc_main
 - rangeproc_dpc.h, [26](#)
- rangeProcHWADpuHandle
 - rangeproc_dpc.c, [48](#)
 - rangeproc_dpc.h, [28](#)
- registerChirpAvailableInterrupts
 - rangeproc_dpc.c, [46](#)
 - rangeproc_dpc.h, [27](#)
- registerChirpInterrupt
 - rangeproc_dpc.c, [47](#)
 - rangeproc_dpc.h, [27](#)
- registerFrameStartInterrupt
 - rangeproc_dpc.c, [47](#)
 - rangeproc_dpc.h, [27](#)
- restoreFactoryCal
 - factory_cal.c, [32](#)
 - factory_cal.h, [19](#)
- src/factory_cal.c, [31](#)
- src/main.c, [33](#)
- src/mem_pool.c, [35](#)
- src/mmwave_basic.c, [37](#)
- src/mmwave_control_config.c, [41](#)
- src/rangeproc_dpc.c, [44](#)
- src/uart_transmit.c, [48](#)
- T_SensPerChirpLut, [6](#)
- uart_transmit.c
 - header, [49](#)
 - uart_transmit_loop, [49](#)
- uart_transmit.h
 - uart_transmit_loop, [30](#)
 - uart_tx_done_sem, [30](#)
 - uart_tx_start_sem, [30](#)
- uart_transmit_loop
 - uart_transmit.c, [49](#)
 - uart_transmit.h, [30](#)
- uart_tx_done_sem
 - main.c, [35](#)
 - uart_transmit.h, [30](#)
- uart_tx_start_sem
 - main.c, [35](#)
 - uart_transmit.h, [30](#)
- uartTask
 - rangeproc_dpc.c, [47](#)
 - rangeproc_dpc.h, [28](#)