Developer's Guide

Interference Lab

Overview

This lab demonstrates interference detection and mitigation on TI's mmWave devices. The lab is built with a philosophy of reuse in mind. Existing SDK 3.5.0.4 components were modified to support interference mitigation. The rangeproc DPU for DSP and HWA, the DPC and the mmWave out of box demo were modified to support interference.

1. Support for interference detection and mitigation was added in the rangeproc DPU for DSP and HWA. Support for interference mitigation was added to the modified rangeproc DPU available in the victim/<DEVICE_VARIANT>/mmw/datapath/dpu folder. Interference mitigation can be enabled and disabled at the DPU level by added configurations. The rest of the DPUs remain unchanged, and are reused from the SDK.
2. The DPC was modified to support added resource requirement for the interference mitigation integration with the range DPU. This modified DPC is available in the victim/<DEVICE_VARIANT>/mmw/datapath/dpc folder. Some extra memory is required for interference mitigation. Interference mitigation runs on the DSP on both AWR1642 and AWR1843 devices. For AWR1843, an extra EDMA channel is required to direct this data to DSP memory first, and then pipe it to HWA for further range processing. This extra piping is handled in the rangeproc HWA DPU. The extra resource allocation is done in the DPC.
3. If interference mitigation is disabled as a static configuration (interference mitigation checkbox in the interference mmWave Demo Visualizer), the DPC will not allocate these extra resources, and hence be virtually similar to the datapath in SDK 3.5.0.4. In this case, interference mitigation cannot be enabled dynamically (advanced commands under plots tab of the interference mmWave Demo Visualizer)
4. The mmwave out of box demo was modified to use the modified datapath and to pass appropriate interference mitigation configurations from CLI to the DPC.

For details of each component including DPC and DPU packaged with this lab, see their respective doxygen documentation.

Rebuild Application & Tool

Steps to rebuild the mmWave Sensor application for different device variants

1. Using CCS:
    a. CCS project files are provided for the interference mitigation demo running on the victim for AWR1642 and AWR1843.
    b. For AWR1642, import as CCS project xwr16xx_intf_demo_dss.projectspec and xwr16xx_intf_demo_mss.projectspec available in victim/xwr16xx/mmw folder.
    c. For AWR1843, import as CCS project xwr18xx_intf_demo_dss.projectspec and xwr18xx_intf_demo_mss.projectspec available in victim/xwr18xx/mmw folder.
    d. For the selected victim device variant, build both DSS and MSS projects. Make sure to build the DSS project before building MSS project.
    e. The image binary and CCS debug binaries will be available in <CCS_WORKSPACE_LOC>/xwrxx_intf_demo_dss/Debug
    and <CCS_WORKSPACE_LOC>/xwrxx_intf_demo_dss/Debug folders after the build process completes.
    f. Note: projectspec files are not provided for the interferer code, testcases for DPU/DPC and building of the rangeproc library (which is later used by the interference mitigation demo running on the victim). To build these, use their makefiles.

2. Using Makefiles:
    a. Open command prompt to
    'C:\ti\mmwave_sdk_03_05_xx_xx\packages\scripts\windows' path.
        i. Make sure you have installed the appropriate SDK version as mentioned in the release notes and other dependency tools.
    b. Set required device type in setenv.bat file.

```
@REM Select your device. Options (case sensitive) are: awr14xx, iwr14xx, awr16xx, iwr16xx, awr18xx, iwr18xx, awr68xx, iwr68xx
set MMWAVE_SDK_DEVICE=awr18xx
```

    c. Run setenv.bat in command prompt which should set environment variables for the mmWave build environment.
    d. In the same command prompt screen, navigate to the folder containing the targeted makefile. use "make all" command in the following directories:
        i. interference_lab/interferer/
        ii. interference_lab/victim/xwr16xx/mmw/datapath/dpu/rangeproc/
        iii. interference_lab/victim/xwr16xx/mmw/datapath/dpc/objectdetection/objdetdsp/ (builds testcases only)
        iv. interference_lab/victim/xwr16xx/mmw/
        v. interference_lab/victim/xwr18xx/mmw/datapath/dpu/rangeproc/
        vi. interference_lab/victim/xwr18xx/mmw/datapath/dpc/objectdetection/objdethwa/ (builds testcases only)
        vii. interference_lab/victim/xwr18xx/mmw/

e. Ensure that the rangeproc DPU is compiled before the DPC and the mmwave demo. Only testcases are compiled in the DPC folder as objectdetection.c is included as a source file in the interference demo.

Directory Structure

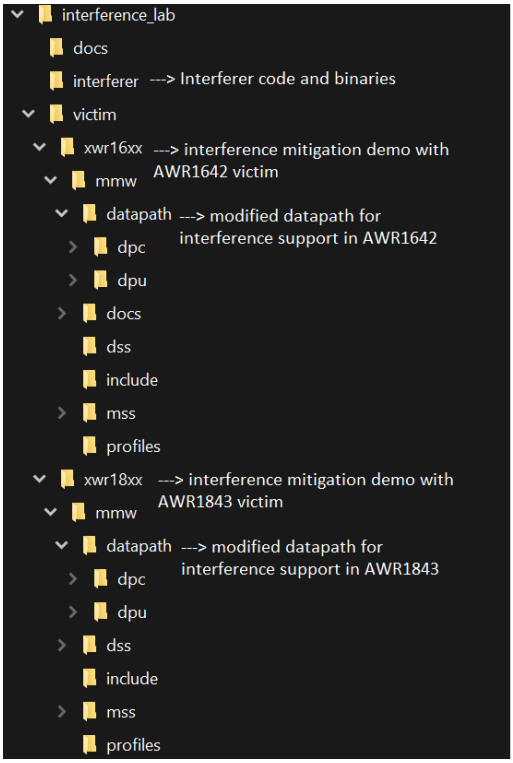

Fig. 1: Directory structure

Changing CW frequency of interferer

The interferer code provided with this lab has the interferer running in continuous wave mode, set at a fixed frequency of 78.000 gHz. To see effects of interference, the interferer frequency must lie in the band of the victim. If this is not the case for your specific configuration of the victim, the interferer/main_mss.c file needs to be recompiled. Change the startFreqConst in the global contModeCfg structure to the needed frequency and recompile it using the provided makefiles.

```
177   const rlContModeCfg_t contModeCfg =
178   {
179       .startFreqConst = 78000*START_FREQ_CONST_VAL_PER_MHZ, //~78.000 gHz
180       .txOutPowerBackoffCode = 0,
181       .txPhaseShifter = 0,
182       .digOutSampleRate = 10000,
183       .hpfCornerFreq1 = 0,
184       .hpfCornerFreq2 = 0,
185       .rxGain = 30,
186       .vcoSelect = 0x0,
187       .reserved  = 0x0
188   };
```

Fig. 2: global structure for CW mode config in interferer/main_mss.c

Details of interference configuration in demo

The modified mmWave demo accepts an additional CLI configuration for interference detection and mitigation. This configuration is sent by the interference mmWave Demo Visualizer to the device. This is the format of the CLI configuration:

intfMitg <subFrameIdx> <enabled> <calcThresh> <performLinearInterp> <threshFacAbs> <threshFacAbsDiff> <threshFacBitwidth> < threshAbs > <threshAbsDiff>

- <subFrameIdx>: it is the subframe index for the configuration. If set to -1, it applies the configuration to all subframes.
- <enabled> field enables/disables the interference detection/mitigation ability altogether. If this is set to 0 in the pre-start configuration, the DPC will not allocate extra resources needed for interference detection and mitigation.
- <calcThresh> enables dynamic per-chirp calculation of the absolute threshold and absolute difference threshold used for interference detection.
- <performLinearInterp> when set to 1 performs linear interpolation on affected regions, using the last good ADC sample before the region of interference and the first good ADC sample after the region of interference.
- <threshFacAbs>, <threshFacAbsDiff>, and <threshFacBitwidth>: If <calcThresh> is set to 1, these factors will be used to calculate the absolute threshold and the absolute difference threshold on a per chirp basis. Details of this calculation is given in the next section.
- <threshAbs>, and <threshAbsDiff>: If <calcThresh> is set to 0, these thresholds will be used as the absolute threshold and the absolute difference threshold respectively for all chirps. If <calcThresh> is set to 1, these fields are ignored.

The interference mmWave Demo Visualizer uses the following configuration by default: "intfMitg -1 1 1 1 64 100 3 0 0". To change this, one of the following two methods can be used:

- For a different configuration as static pre-start configuration (before sensorStart), use the "Save config to PC" option, change the interference configuration in the .cfg file and load this file with the "Load config from PC and send" button in the plots tab.
- To change the configuration dynamically (in between frames after senorStart), use the "send command" button under plots->Advanced commands tab.

This configuration is passed from the CLI to the DPC via the DPC_ObjectDetection_IntfMitgCfg structure, which passes it to the DPU via the DPU_RangeProc_IntfMitgCfg structure. This is done either with the DPC_OBJDET_IOCTL__STATIC_PRE_START_CFG IOCTL command as a static configuration, or with DPC_OBJDET_IOCTL__DYNAMIC_INTF_MITG_CFG IOCTL command (DPU_RangeProcDSP_Cmd_intfMitg or DPU_RangeProcHWA_Cmd_intfMitg command at the DPU level) as a dynamic configuration.

Interference detection and mitigation algorithm details

To detect interference, two thresholds are used. threshAbs is applied on the absolute value of the complex ADC samples, threshAbsDiff is applied on the absolute value of the difference of two consecutive complex ADC samples, i.e. abs(x[i]-x[i-1]).

If the absolute value of a sample or the absolute value of the difference of a sample exceeds their respective threshold, the sample is flagged as being affected by interference via an array of interference indicator bits (IIB array). If i'th sample exceeds any of the two thresholds, all samples in the window [i-N_HYST, i+N_HYST] are also flagged. N_HYST is set to 15.

The thresholds threshAbs and threshAbsDiff are computed on a per chirp basis by the following formulae:

$$threshAbs = \frac{\sum_1^{len} |x_i|}{len} \times \frac{threshFacAbs}{2^{threshFacBitwidth}}$$

And

$$threshAbsDiff = \frac{\sum_2^{len} |x_i - x_{i-1}|}{len} \times \frac{threshFacAbsDiff}{2^{threshFacBitwidth}}$$

Here, x_i is the i'th complex ADC sample in a chirp with 'len' ADC samples. Only one RX channel is used for threshold computation. A sample is a 16-bit (each for real and imaginary part) complex number. threshAbs and threshAbsDiff are 16-bit numbers, and are both capped at a maximum of 0x7FFF.

During interference mitigation, all of the samples flagged as being affected by interference are zeroed out across all RX channels. These blanked out regions are then filled with linear interpolation.

This computation is performed directly on the raw ADC data. All of the data processing steps follow this step.

Refer to this application note for theoretical details on interference mitigation with AWR devices.