Developer's Guide

mmWave Diagnostic & Monitoring TI Design

Overview

This demo is a reference application to showcase the implementation of mmWave SafeTI Diagnostic Library (SDL). This demo assists in the development of software applications involving functional safety. Here is the list of features this demo provides

1. Demonstrating easy implementation of safety & monitoring functionality in mmWave Radar Sensors
2. Demonstrating System level safety feature
3. Make use of safety features to build ASIL-B/SIL2 mmWave sensor
4. Secondary Bootloader (SBL) runs few of destructive diagnostic tests.
5. After diagnostic test execution, SBL boots the main application.
6. Main application executes remaining diagnostic test and device monitor features.
7. TI-RTOS based MSS implementation of diagnostic tests.
8. Non-OS (bare metal) based DSS implementation of diagnostic tests.
9. All the test status is reported over UART.
10. MSS runs periodic diagnostic tests post frame trigger.

IS	IS NOT
Reference design using mmWave Radar and PMIC	ASIL-B/D Safety design
Reference example to demonstrate the safety features mentioned in Safety manual	Safety Software deliverable
Tool to estimate monitoring threshold	Monitoring recommendation
Based on TIRTOS and provides OSAL for Safety OS integration	Safety OS

Pre-requisite

1. mmWave Sensor device board or EVM.
2. Micro USB & power adaptors for above board.
3. PC running on Windows 10 OS.
4. XDS110 driver for UART COM port.
5. mmWave SDK 3.5
6. mmWave SDL 1.0

It's highly recommended to refer following document from SDL installation to get acquainted with SDL design and architectures.

<SDL_install_path>/csp/mmwave_sdl_user_guide.pdf	Information on the software, installation, how to build, and other relevant information
<SDL_install_path>/csp/API_doc/mmwave_sdl_api.pdf	API, module, data structure information generated by doxygen. HTML version at ti/docs/doxygen/html/index.html
<SDL_install_path>/csp/Architecture_Design_Integration/ mmwave_sdl_architecture.pdf	Architecture doc. HTML version at ti/docs/doxygen/html/index.html
<SDL_install_path>/csp/Architecture_Design_Integration/ mmwave_sdl_design_integration.pdf	Design and Integration doc. HTML version at ti/docs/doxygen/html/index.html

Directory Structure

Package contains the SBL, OSAL and main application source codes.

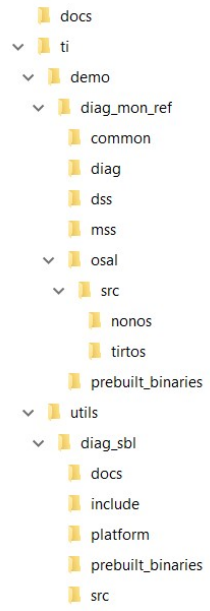


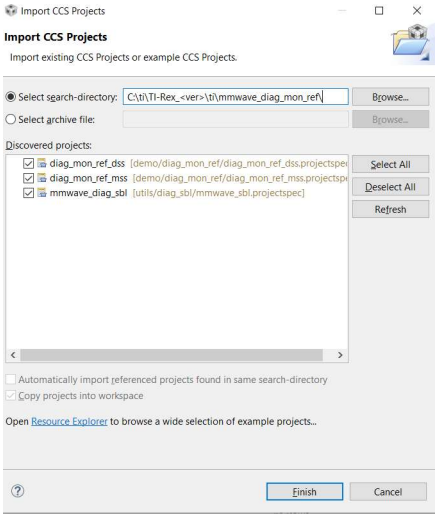
Figure 2 Directory Structure

Rebuild Application

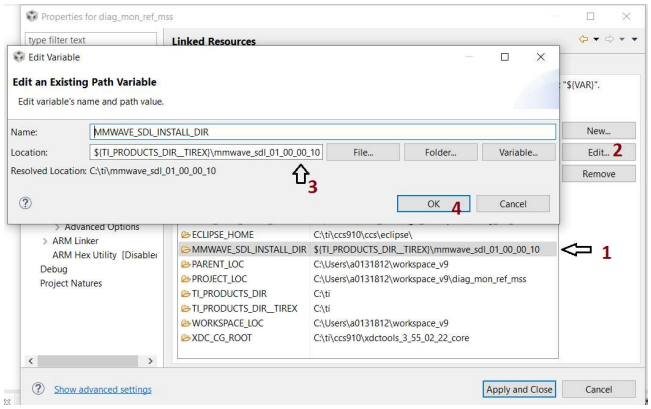
Make sure that you have install mmWave SDL (Safety Diagnostic Library). It will be available under mySecure weblink which requires safety NDA with TI, contact your local sales person for the same.

mmWave SDK 3.5 and all other dependencies come along with SDK installation.

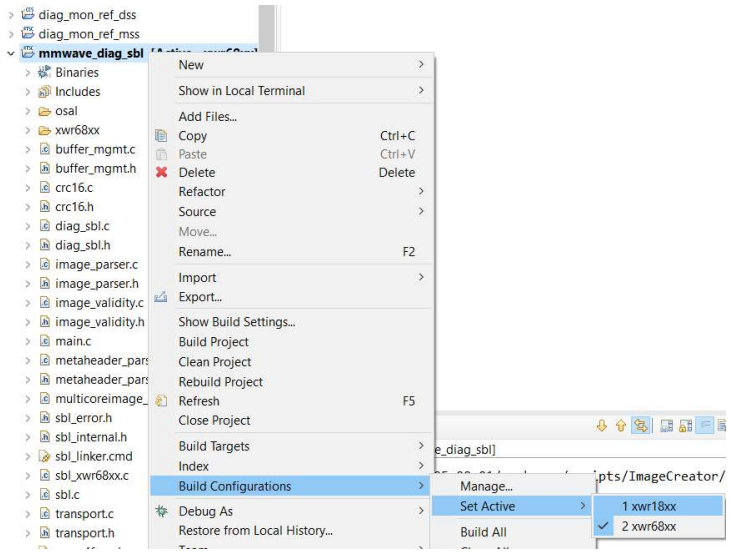
- Import MSS and DSS project to CCS.



- To execute MSS/DSS diagnostic test, application uses SDL libraries. So after projects are imported, set mmWave SDL installation path, by default that path is set to "C:/ti/mmwave_sdl_01_00_00_10" in CCS project properties.



- SDL provides source files for all the DIAG, CSL components not libraries. So user needs to build SDL libraries first for specific device variants (AWR1843/xWR6843), please refer SDL user guide for compilation instructions.
- Same Projectspec contains CCS project for both of device variants (18xx/68xx), user can change that from CCS project properties for SBL as well as Main-application.



Application Overview

This application contains two set of MetaImage binaries to run, first SBL (ti\utils\diag_sbl) and second diagnostic & Monitor reference app (ti\demo\diag_mon_ref). Following figure shows the high level of flow diagram of application.

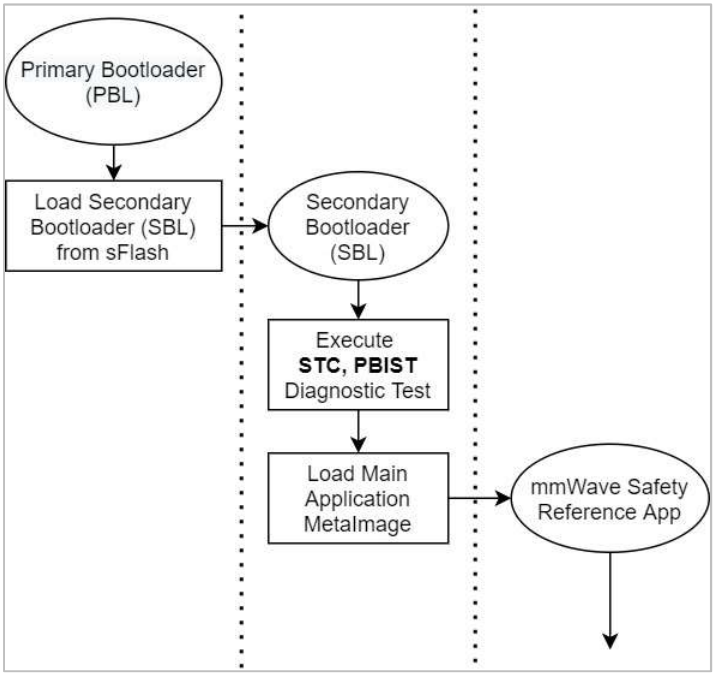


Figure 1 Overall Software Flow

Secondary Bootloader (SBL)

The secondary bootloader primarily is responsible for updating the application meta image in the SFLASH by receiving the image over a serial interface. It then loads and runs the updated application meta image.
The ROM(primary) bootloader always loads the SBL. Application can choose to either update or load and run the application meta image.

For safety applications, the SBL may be used to perform some of destructive tests like PBIST and STC that needs to be run during boot time. These tests are validated before loading the main application. In-case of failure the SBL aborts and exit.

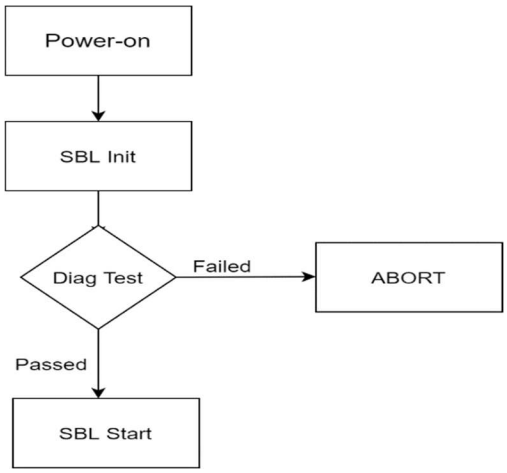


Figure 2 SBL updated flow-chart

As soon as SBL is booted, it performs STC and PBIST on misc. device memory which may cause core reset. SBL uses SPARE register to store the test status during the test as SPARE register retains it value across soft/warm reset of the device which happens due to these destructive DIAG tests. After all these DIAG tests are pass, SBL moves to application loading flow where user can either opt to load new image over UART to sFlash (press space key) or load the existing image from sFlash (press 'a' to skip the counter).

Diagnostic & Monitor Reference Application

Main application contains MSS and DSS core application where different set of diagnostic tests are executed. At the bootup it checks for boot status tests result and unhalts BSS & DSS core. In parallel to DSS diagnostic test execution, MSS runs set of R4F fault injection, self test, I/O and static diagnostic tests. Handshake of test status from DSS to MSS happens over HSRAM.

Later to that MSS configures RadarSS for misc. monitoring features with a fixed set of configuration which user can change as required (rfmonitor_defaultCfg.c). All the monitoring APIs are configured to report only in case of failure.

At every FTTI (monitor report interval), MSS application checks for any failure monitor report received by RadarSS in form of Async-event message and reports over UART. Along with this it

executes set of static diagnostic tests as well periodically at every FTTI interval and reports this result also over UART. Refer MmwDemo_MonReportStreamTask function in diag_mon_main.c file.

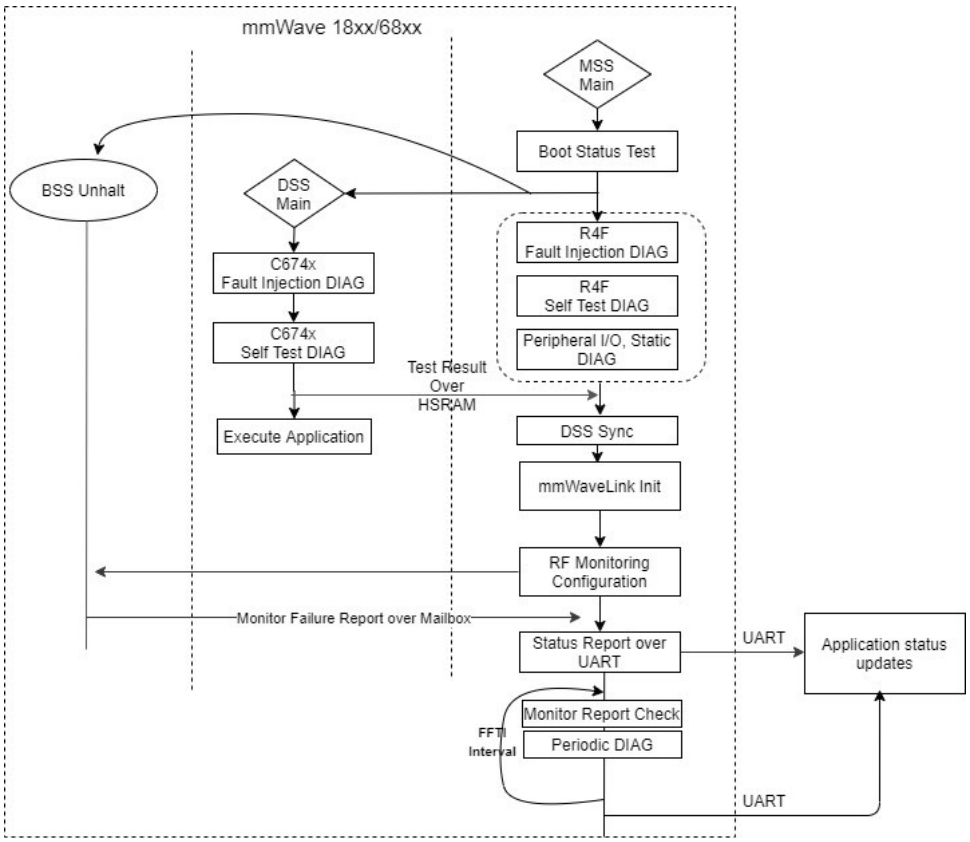


Figure 1 Main Application Flow Diagram

OSAL Implementation

Diagnostic test from SDL uses OSAL to hook its ESM and CPU fault callbacks, so main application flow doesn't get impacted.

Most of diagnostic tests cause CPU or ESM fault, which may hamper main application flow. To avoid that Diagnostic Test attach a hook to OS or Non-OS (bare metal) layer for these CPU or ESM faults. These hooks get called as soon as Fault/Error caused by diagnostic test. Diagnostic test verifies its test flow based on its own hooks and returns the pass or fail of the test. Please refer mmwave_sdl_api.pdf from SDL installation path for more info.

mmWave SDL provides OSAL implementation for Non-OS version whereas mmWave SDK provides OSAL for TI-RTOS. In this application OSAL is modified for NonOS and TI-RTOS according to the application requirements. Same OSAL is being used by the SBL as well. Refer source code at 'ti\demo\diag_mon_ref\osal\src' directory.

- MSS CPU Fault handler integration with TI-RTOS (sysBios)

  SysBios provides an exception hook where user application can attach its own function, so SysBios would invoke user-defined exception handler in lieu of SysBios Internal exception function. It is defined in mmw_mss.cfg file

```
/* to hook user-defined exception handlers */
Exception.excHookFunc =  '&MMW_EXCEPTION_HANDLER';
```

This exception handler is defined in osal_r4f_interrupt.c file where each type exception handler is updated by diagnostic test, so it can check the test status.

Note: Since TI-RTOS aborts the application even if that exception is raised knowingly by the app (DIAG), so currently there is no way we can avoid application termination during this kind of DIAG test. Thus that specific DIAG (TCM 2-bit ECC) is not enabled in this application.

- MSS ESM Fault handler integration with TI-RTOS (sysBios)
  OSAL_R4F_ESMDrv_addHook function in OSAL (osal_r4f_esmdrv.c) is being used to register ESM fault handler with ESM driver of mmWave SDK. During the Diagnostic test, SDL add its own hook to ESM module to avoid any disturbance to main application.