MMWAVE SDK Unit Test Procedure

Document Version: 1.2

DOCUMENT LICENSE

COPYRIGHT

CONTENTS

1. Driver Module and Library Module Unit Tests

Each mmWave SDK driver module includes a unit test program that demonstrates API use case examples and implements functional validation. These programs are located in the directory:

mmwave_sdk_<ver>/packages/ti/drivers/<module>/test/

The unit test programs generally setup all necessary program configurations, exercise the driver module, and then check the execution outcomes with minimal to no user intervention.

1. 1. Understanding unit test logs in SDK package

The execution logs of these tests are provided in the release package. See the directories:

<mmwave_sdk_<ver>/docs/test/<device>/module_test

and

<mmwave_sdk_<ver>/docs/test/<device>/alglib_test

All of the test log filenames are in the following format:

<device>_<R4|DSS>_<module/testcase>_<test execution date-time>.log

All tests are single core execution except those with '(TwoCore)' appended after the '_<R4|DSS>_' field.

1. 1. 1. Single core test execution that need no user intervention

Generally these are single core tests, with executables being <device>_<driver>_mss.xer4f for the R4 and <device>_<driver>_dss.xe674 for the DSP where applicable.
The executable name for a given test is found in the corresponding log file in the line containing:

Loading binary to the DUT

The test logs include any associated test setup configuration and the CCS console I/O for each test. Any variables updated prior to execution in order to enable specific configurations are shown prior to the log line:

Parameters found in the Binary arguments

and followed by the variable/symbol, and its assigned value.

The log also includes all execution functional results, each prefaced with "Feature:" and the validation result, for example:

Feature: CRC Type: 16bit Data Length: 32bit: Passed

1. 1. 2. Dual core test execution that need no user intervention

Dual core tests - these test case logs include '(Two Core)' in the log file name - exercise the same basic flow as the single core case; loading executable, loading necessary configuration variable, and logging results. In these test cases there is a 'slave' core, which is loaded, initialized, and set to execution prior to the 'master' core. Then the 'master' flow processes as in the single core case.

2. Unit Tests that require User intervention and/or special setup

2. 1. R4 SPI back-to-back Test

2. 1. 1. Test Summary

Two EVMs are wired for SPI interface cross-connection. Tests on each EVM are run in both Master mode and in Slave mode. The Master mode sends a test sequence to the Slave mode EVM, and it echoes the sequence back to the Master mode EVM. That EVM does a data integrity test.

For the back-to-back tests, the Slave EVM must have completed initialization and be ready to receive prior to the Master EVM beginning transmission. A tester can either monitor the 'gXWR1xxxSlaveReady' variable or view the console output to know when the Slave is ready. At this point the Master device can start the test sequence (i.e. hit run in CCS), and the test sequence flows programmatically to test completion.

To connect 2 mmWave EVMs to the same machine and execute via CCS, refer to instructions mentioned here: http://software-dl.ti.com/ccs/esd/documents/sdto_ccs_multi-probe-debug.html

2. 1. 2. Variable Assignment

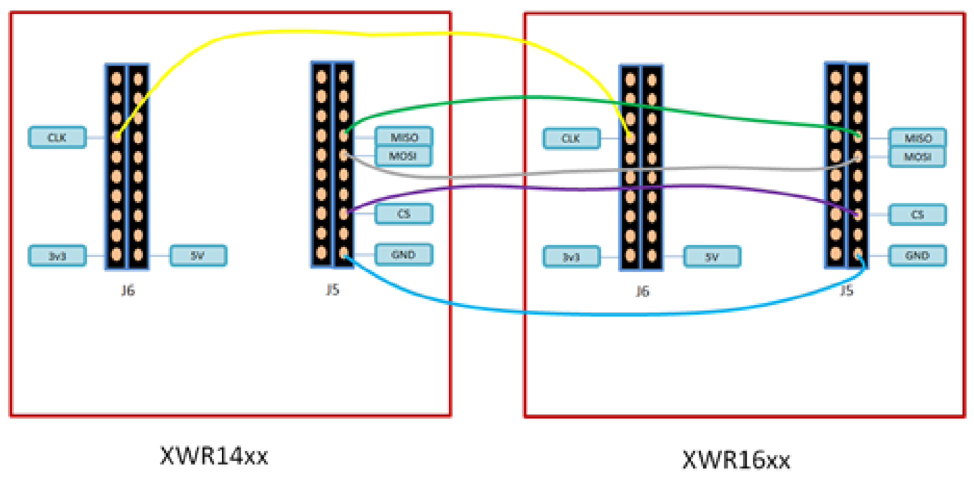
2. 1. 2. 1. Slave EVM

Parameter	Loopback Value
gXWR1xxxLoopbackTest	0
gXWR1xxxMasterWithXWR1xxx	0
gXWR1xxxSlaveWithXWR1xxx	1
gXWR1xxxSlaveWithFTDITest	0

2. 1. 2. 2. Master EVM

Parameter	Loopback Value
gXWR1xxxLoopbackTest	0
gXWR1xxxMasterWithXWR1xxx	1
gXWR1xxxSlaveWithXWR1xxx	0
gXWR1xxxSlaveWithFTDITest	0

2. 1. 3. Physical Setup: Hardware EVM to EVM cross-connections



2. 1. 4. Example Test Log

Please refer to release document file "mmwave_sdk_<ver>\docs\test\IWR16\module_test\xwr16_R4_SPI_bk2bk_Slave_(BackToBack)_<timestamp>.log" for the detailed test log.

Basic Test Sequence

- Back to Back test, one EVM in Master mode, one EVM in Slave mode
  - Sync and transmission tests, Master-to-Slave, echoed back Slave-to-Master
  - Test executes data integrity test at various supported supported bit rates and modes
  - Throughput results reported
- Switch Master/Sync modes between EVMs, Slave mode, Master mode
  - Sync and transmission tests, Master-to-Slave, echoed back Slave-to-Master
  - Test executes data integrity test at various supported supported bit rates and modes
  - Throughput results reported

2. 2. R4 SPI/FTDI Test

2. 2. 1. Test Summary

The EVM sends a test sequence to a Unix PC test application via FTDI. That application echoes the sequence back to the EVM. The EVM does a data integrity test.

2. 2. 2. Unix PC Utilities

The Unix PC application is available in the 'mmwave_sdk_<ver>/packages/ti/drivers/spi/test/unix' directory. Please see README.txt for build and execute instructions.

2. 2. 3. Windows PC Utilities

Similarly, a test application is available for Windows as well. Look for the source and batch file in the 'mmwave_sdk_<ver>/packages/ti/drivers/spi/test/windows/' directory. Note: the FTDI driver needs to be installed for windows (driver is in mmwave_sdk_<ver>/tools/ftdi folder).

2. 2. 4. Variable Assignment

Parameter	Value
gXWR1xxxLoopbackTest	0
gXWR1xxxMasterWithXWR1xxx	0
gXWR1xxxSlaveWithXWR1xxx	0
gXWR1xxxSlaveWithFTDITest	1

2. 2. 5. Physical setup

The EVM is connected to PC USB port.

2. 2. 6. Example Test Log

Please refer to release document file "mmwave_sdk_<ver>\docs\test\AWR16\module_test\xwr16_R4_spiFTDI_<timestamp>.log" for the detailed test log.

2. 2. 7. Basic Test Sequence

- Configure driver for FTDI test
- Sync and transmission tests, PC application echoes back to EVM via SPI/FTDI
- Multiple test transmissions.
- The EVM does a data integrity test.

2. 3. CAN Unit Tests

Executable

mmwave_sdk_<ver>/packages/ti/drivers/can/test/<device>/<device>_can_mss.xer4f

CAN Test Configuration Variables

Test Scenario	`testSelection` Value	Interface	Test Scenario
Internal loopback test	1	internal	Internal, digital loopback test
external loopback test	2	internal	external, analog loopback test
Parity test	3	PCAN	Parity test
Tx/Rx test	4	PCAN	Tx/Rx test

When the unit test is executed using CCS, the test variable 'testSelection' can be manipulated via the 'expression' window to the required value before running the test (i.e. after the loading of unit test binary (code reaches main) but before hitting run in CCS).

2. 3. 1. R4 CAN Tx/Rx Test to PCAN

2. 3. 1. 1. Test Summary

In CAN mode the unit test implements a Transmit and Receive test with the PCAN test utility. The test exercises multiple iterations. The unit test does a data integrity test. Please refer to the "PCAN-USB Test Utility" section below for the physical test requirements using the PCAN system.

Transmit frame:

["a1 1a ff ff c1 1c b1 1b","a2 2a ff ff c2 2c b2 2b","a3 3a ff ff c3 3c b3 3b", "a4 4a ff ff c4 4c b4 4b","a5 5a ff ff c5 5c b5 5b","a6 6a ff ff c6 6c b6 6b","a7 7a ff ff c7 7c b7 7b","a8 8a ff ff c8 8c b8 8b"]

2. 3. 1. 2. PCAN Unix command line

"~/peak-linux-driver-8.5.1/test/pcanfdtst rx -t 10 -n 9 -c 40M -b 1M -d 5M /dev/pcan32"

2. 3. 1. 3. Example Test Log

Please refer to release document file "mmwave_sdk_<ver>\docs\test\IWR16\can_test\xwr16_R4_CAN_Tx_Rx_<timestamp>.log" for the detailed test log.

2. 4. CANFD Unit Tests

2. 4. 1. Executable

mmwave_sdk_<ver>/packages/ti/drivers/can/test/<device>/<device>_canfd_mss.xer4f

2. 4. 2. CANFD Test Configuration Variables

Test Scenario	`testSelection` Value	Interface	Test Scenario
Internal loopback test	1	internal	Internal, digital loopback test
external loopback test	2	internal	external, analog loopback test
Multiple Tx test	3	external to PCAN	Multiple Tx transmission to PCAN
Tx/Rx test	4	external to PCAN	Tx transmission to PCAN
EVM-EVM test	5	external to another EVM	Two EVMs do transmission tests
Tx Cancel test	6	internal	
Power down test	7	internal	
Message Id Range test	8	external to PCAN	
Dual Multiple Tx test	9	external to two separate PCAN	Multiple Tx test concurrently to 2 interfaces
Dual Message Id Range test	10	external to two separate PCAN	Message Id Range test concurrently to 2 interfaces

When the unit test is executed using CCS, the test variable 'testSelection' can be manipulated via the 'expression' window to the required value before running the test (i.e. after the loading of unit test binary (code reaches main) but before hitting run in CCS).

2. 4. 3. CANFD Instances

The CANFD driver supports two instances on some devices (see device datasheet). By default 'gInstanceId' = 0 is used in the unit tests. For devices/EVMs that support a second CANFD, the unit tests can be modified to use this interface by assigning 'gInstanceId' = 1.

Two unit test menu options execute concurrent tests with the PCAN test set, 'Dual MCAN Multiple Tx test' and the 'Dual MCAN Message Id Range test.'

2. 4. 4. R4 CANFD Tx/Rx Test to PCAN

Test Summary

In CANFD mode the unit test implements a Transmit and Receive test with the PCAN test utility. The test exercises multiple iterations. The unit test does a data integrity test. Please refer to the "PCAN-USB Test Utility" section below for the physical test requirements using the PCAN system.

Transmit frame:

["a1 1a ff ff c1 1c b1 1b","a2 2a ff ff c2 2c b2 2b","a3 3a ff ff c3 3c b3 3b", "a4 4a ff ff c4 4c b4 4b","a5 5a ff ff c5 5c b5 5b","a6 6a ff ff c6 6c b6 6b","a7 7a ff ff c7 7c b7 7b","a8 8a ff ff c8 8c b8 8b"]

PCAN Unix command line

"~/peak-linux-driver-8.5.1/test/pcanfdtst rx -t 10 -n 9 -c 40M -b 1M -d 5M /dev/pcan32"

Example Test Log

Please refer to release document file "mmwave_sdk_<ver>\docs\test\IWR16\can_test\xwr16_R4_CAN_Tx_Rx_<timestamp>.log" for the detailed test log.

2. 0. 1. R4 CANFD Multiple Tx

Test Summary

In CANFD mode the unit test implements multiple Transmit sequences to the PCAN test utility. PCAN returns on the command line the Receive data. The test exercises multiple iterations. The user needs to check the received data. Please refer to the "PCAN-USB Test Utility" section below for the physical test requirements using the PCAN system.

Transmit frame:

["a1 1a ff ff c1 1c b1 1b","a2 2a ff ff c2 2c b2 2b","a3 3a ff ff c3 3c b3 3b", "a4 4a ff ff c4 4c b4 4b","a5 5a ff ff c5 5c b5 5b","a6 6a ff ff c6 6c b6 6b","a7 7a ff ff c7 7c b7 7b","a8 8a ff ff c8 8c b8 8b"]

PCAN Unix command line

"~/peak-linux-driver-8.5.1/test/pcanfdtst rx -t 10 -n 2 -c 40M -b 1M -d 5M /dev/pcan32"

Example Test Log

Please refer to release document file "mmwave_sdk_<ver>\docs\test\IWR16\can_test\xwr16_R4_CANFD_Multiple_Tx_<timestamp>.log"

2. 0. 1. R4 CANFD External Tx/Rx Classic

Test Summary

In CAN mode (note: not CAN FD) the unit test implements multiple Transmit sequences to the PCAN test utility. PCAN returns on the command line the Receive data. The test exercises multiple iterations. The user needs to check the received data. Please refer to the "PCAN-USB Test Utility" section below for the physical test requirements using the PCAN system.

Transmit frame:

["a1 1a ff ff c1 1c b1 1b","a2 2a ff ff c2 2c b2 2b","a3 3a ff ff c3 3c b3 3b", "a4 4a ff ff c4 4c b4 4b","a5 5a ff ff c5 5c b5 5b","a6 6a ff ff c6 6c b6 6b","a7 7a ff ff c7 7c b7 7b","a8 8a ff ff c8 8c b8 8b"]

PCAN Unix command line

"~/peak-linux-driver-8.5.1/test/pcanfdtst rx -t 10 -n 9 -c 40M -b 1M -d 5M /dev/pcan32"

Example Test Log

Please refer to release document file "mmwave_sdk_<ver>\docs\test\IWR16\can_test\xwr16_R4_CANFD_External_Tx_Rx_Classic_<timestamp>.log"

2. 0. 1. R4 CANFD External Tx/Rx FD

Test Summary

In CANFD mode the unit test implements multiple Transmit sequences to the PCAN test utility. PCAN returns on the command line the Receive data. The test exercises multiple iterations. The user needs to check the received data. Please refer to the "PCAN-USB Test Utility" section below for the physical test requirements using the PCAN system.

Transmit frame:

["a1 1a ff ff c1 1c b1 1b","a2 2a ff ff c2 2c b2 2b","a3 3a ff ff c3 3c b3 3b", "a4 4a ff ff c4 4c b4 4b","a5 5a ff ff c5 5c b5 5b","a6 6a ff ff c6 6c b6 6b","a7 7a ff ff c7 7c b7 7b","a8 8a ff ff c8 8c b8 8b"]

PCAN Unix command line

"~/peak-linux-driver-8.5.1/test/pcanfdtst rx -t 10 -n 2 -c 40M -b 1M -d 5M /dev/pcan32"

Example Test Log

Please refer to release document file "mmwave_sdk_<ver>\docs\test\IWR16\can_test\xwr16_R4CANFD_External_Tx_Rx_FD_<timestamp>.log"

2. 0. 1. R4 CANFD EVM-EVM

Test Summary

Two EVMs with connected CANFD signals Transmit sequences to each other. The test exercises multiple iterations. The unit test does a data integrity test. The two EVMs must have their respective CAN signals connected per below:

EVM-1	EVM-2
CANFD-L	CANFD-L
GND	GND
CANFD-H	CANFD-H

Transmit frame:

["a1 1a ff ff c1 1c b1 1b","a2 2a ff ff c2 2c b2 2b","a3 3a ff ff c3 3c b3 3b", "a4 4a ff ff c4 4c b4 4b","a5 5a ff ff c5 5c b5 5b","a6 6a ff ff c6 6c b6 6b","a7 7a ff ff c7 7c b7 7b","a8 8a ff ff c8 8c b8 8b"]

Example Test Log

Please refer to release document file "mmwave_sdk_<ver>\docs\test\IWR16\can_test\xwr16_R4_CANFD_EVM-EVM_<timestamp>.log"

2. 0. 1. R4 CANFD Range ID - Using PCAN GUI Interface

Test Summary

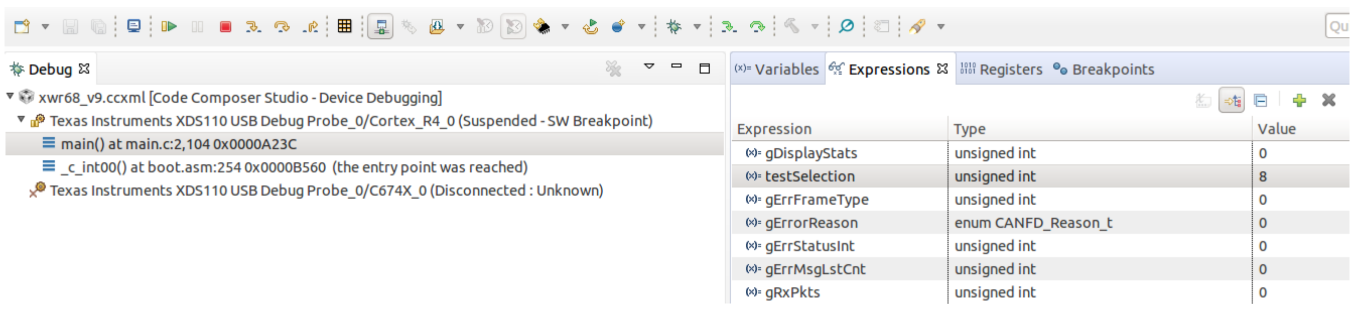
The CANFD driver receiver is conditioned by the unit test program to recognize Range IDs in specified ranges:

range ID start	range ID end
C1	C1
C2	C4
CC	CE
D6	D8

The Range ID tests consist of transmitting CANFD messages with various Range IDs, and check the statistics of the unit test. Messages with IDs within the defined receive ranges are logged by the unit test.

Unit Test Running in CCS

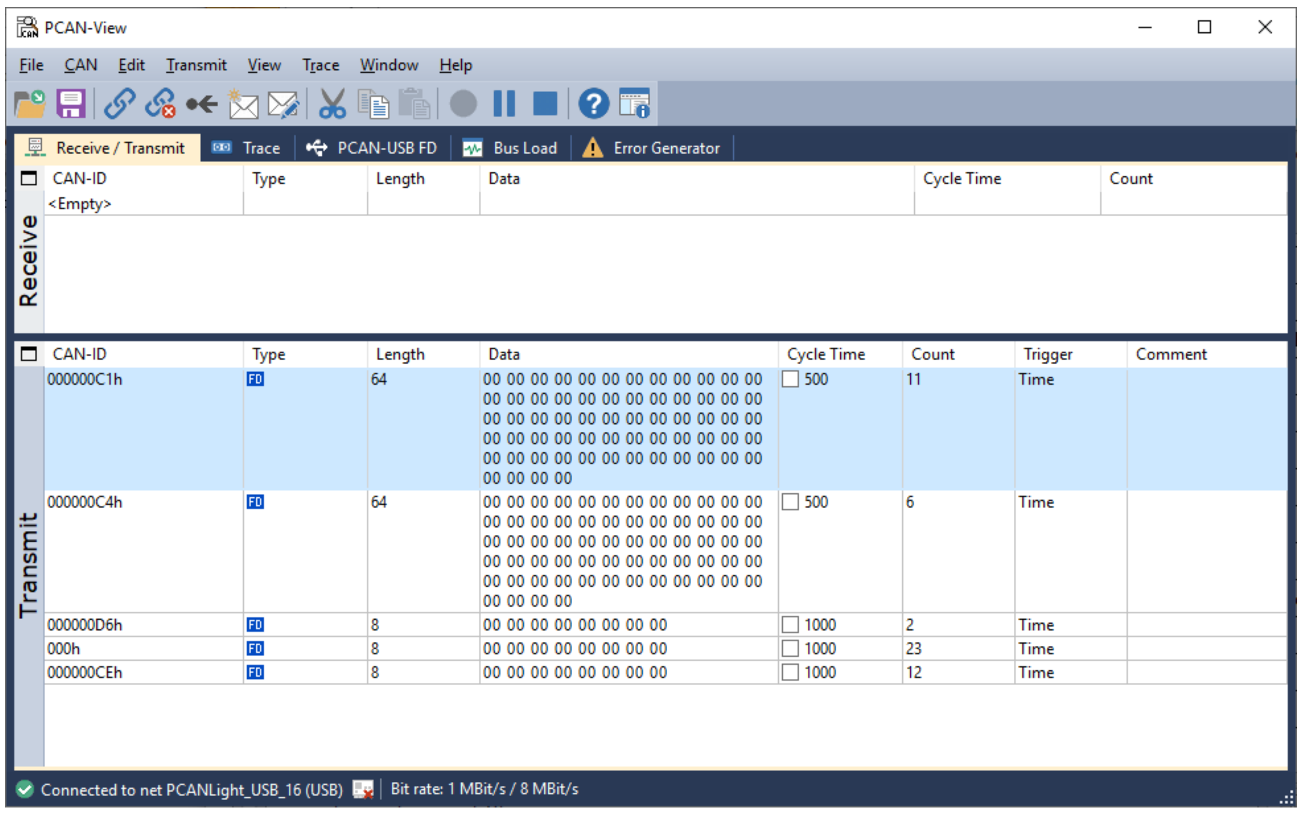
Bring up the CCS GUI, connect to target, and load the executable for the device unit test, in this case 'xwr68xx_canfd_mss.xer4f'. Then set the global variables 'gDisplayStats' = 0 and 'testSelection' = 8, and run.



```
Debug
xwr68_v9.ccxml [Code Composer Studio - Device Debugging]
  Texas Instruments XDS110 USB Debug Probe_0/Cortex_R4_0 (Suspended - SW Breakpoint)
    main() at main.c:2,104 0x0000A23C
    _c_int00() at boot.asm:254 0x0000B560  (the entry point was reached)
  Texas Instruments XDS110 USB Debug Probe_0/C674X_0 (Disconnected : Unknown)
```

Expression	Type	Value
gDisplayStats	unsigned int	0
testSelection	unsigned int	8
gErrFrameType	unsigned int	0
gErrorReason	enum CANFD_Reason_t	0
gErrStatusInt	unsigned int	0
gErrMsgLstCnt	unsigned int	0
gRxPkts	unsigned int	0

Test messages can be defined in the PCAN test utility GUI, each with a specific message ID:



PCAN-View

Receive

CAN-ID	Type	Length	Data	Cycle Time	Count
<Empty>					

Transmit

CAN-ID	Type	Length	Data	Cycle Time	Count	Trigger	Comment
000000C1h	FD	64	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	☐ 500	11	Time	
000000C4h	FD	64	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	☐ 500	6	Time	
000000D6h	FD	8	00 00 00 00 00 00 00 00	☐ 1000	2	Time	
000h	FD	8	00 00 00 00 00 00 00 00	☐ 1000	23	Time	
000000CEh	FD	8	00 00 00 00 00 00 00 00	☐ 1000	12	Time	

Connected to net PCANLight_USB_16 (USB) | Bit rate: 1 MBit/s / 8 MBit/s

Toggling the checkbox in the 'cycle time' field for each message transmits the number of messages indicated in the Count field.

After transmitting the desired messages, enable the stats display with 'gDisplayStats' = 1. This triggers the stats output after the next receive message. A single message must then be transmitted to trigger the stats output

CCS console output for test sequence:

```
Debug: Single Message Identifier
Debug: Start Message Identifier : 0xc1
Debug: End Message Identifier : 0xc1
Debug: Direction : Receive
Debug: Number of interrupts received : 11
Debug: Number of messages processed : 11


Debug: Range Message Identifier
Debug: 0 Start Message Identifier : 0xc2
Debug: 0 End Message Identifier : 0xc4
Debug: 0 Direction : Receive
Debug: 0 Number of interrupts received : 6
Debug: 0 Number of messages processed : 6


Debug: Range Message Identifier
Debug: 1 Start Message Identifier : 0xcc
Debug: 1 End Message Identifier : 0xce
Debug: 1 Direction : Receive
Debug: 1 Number of interrupts received : 12
Debug: 1 Number of messages processed : 12


Debug: Range Message Identifier
Debug: 2 Start Message Identifier : 0xd6
Debug: 2 End Message Identifier : 0xd8
Debug: 2 Direction : Receive
Debug: 2 Number of interrupts received : 2
Debug: 2 Number of messages processed : 2


Debug: Number of Frame mismatch : 0
Debug: Number of Get_Data errors : 23
Debug: Error Status Interrupt : 0
```

2. 0. 1. R4 CANFD Range ID - Using test automation

Test Summary

The CANFD driver receiver is conditioned by the unit test program to recognize Range IDs in specified ranges:

range ID start	range ID end
C1	C1
C2	C4
CC	CE
D6	D8

The Range ID tests consist of transmitting CANFD messages with various Range IDs, and check the statistics of the unit test. Messages with IDs within the defined receive ranges are logged by the unit test.

Unit Test Running in automation

Automation conditions the unit test for Range ID test and then using the command line PCAN utility, it transmits multiple messages both within and outside of the configured unit test ID ranges. The format for the PCAN command line:

```
echo Send Message Identifier; 0xc1
~/peak-linux-driver-8.5.1/test/pcanfdtst tx -n 1 --fd -ie 0x000000C1 -c 40M -b 1M -d 8M -l 64 -v -P 500m /dev/pcan32 | tee PCAN_Logfile.txt
```

This command sends 1 (number defined by "-n") message with Range ID = 0xC1.

PCAN produces the log file PCAN_Logfile.txt. An example log file is shown below:

```
start opening 1 devices:

opening "/dev/pcan32" with flags=ca000004h bitrate=1000000 bps sample_pt=0 dbitrate=8000000 bps dsample_pt=0 clock=40000000 Hz
"/dev/pcan32" opened (fd=3)
running 1 loops
1584470708.689208 /dev/pcan32 > BUS STATE=ACTIVE [Rx:0 Tx:0]
1584470708~699470 /dev/pcan32 < 000000c1 .e... [00 00 00 00 00 00 00 00 00 00]
[00 00 00 00 00 00 00 00 00 00]
[00 00 00 00 00 00 00 00 00 00]
[00 00 00 00 00 00 00 00 00 00]
[00 00 00 00 00 00 00 00 00 00]
[00 00 00 00 00 00 00 00 00 00]
[00 00 00 00]
stop test after 1 loops
end of test loop (tst=1).
/dev/pcan32 < [packets=1 calls=1 bytes=64 eagain=0]
all 1 devices closed
--- stop logging ---
sent frames: 1
```

Test automation then enables the stats output and transmits a triggering message. The stats output is then checked for the correct number of messages received.

Example Test Log

Please refer to release document file "mmwave_sdk_<ver>\docs\test\AWR68\canfd_test\xwr16_R4_CANFD_Range_ID_<timestamp>.log"

2. 0. 1. R4 CANFD Multiple Tx, Dual Instance

Test Summary

In CAN FD mode the unit test implements multiple Transmit sequences to the PCAN test utility concurrently on both CANFD interfaces using both CANFD instances. Note this test requires two PCAN test system probes operating concurrently and independently. The PCAN test system returns the Receive data on the command line instance for each PCAN system, respectively. The test exercises multiple iterations. The user needs to check the received data. Please refer to the "PCAN-USB Test Utility" section below for the physical test requirements using the PCAN test system.

Transmit frame:

["a1 1a ff ff c1 1c b1 1b","a2 2a ff ff c2 2c b2 2b","a3 3a ff ff c3 3c b3 3b", "a4 4a ff ff c4 4c b4 4b","a5 5a ff ff c5 5c b5 5b","a6 6a ff ff c6 6c b6 6b","a7 7a ff ff c7 7c b7 7b","a8 8a ff ff c8 8c b8 8b"]

PCAN Unix command line

In this example, one PCAN test system probe is available at device '/dev/pcan32' and the other is available at '/dev/pcan33'. The command lines for both are shown here:

- "~/peak-linux-driver-8.5.1/test/pcanfdtst rx -t 10 -n 2 -c 40M -b 1M -d 5M /dev/pcan32"
- "~/peak-linux-driver-8.5.1/test/pcanfdtst rx -t 10 -n 2 -c 40M -b 1M -d 5M /dev/pcan33"

Example Test Log

Please refer to release document file "mmwave_sdk_<ver>\docs\test\AWR68\canfd_test\xwr68_R4_CANFD_Multiple_Tx,_Dual_Instance<timestamp>.log"

NOTE: The test is supported in the xWR68 unit test only. This test requires EVM to support CAN FD transceivers on both interfaces to successfully execute this unit test. Please refer to EVM documentation to understand EVM capabilities.

2. 0. 1. R4 CANFD Range ID, Dual Instance

Test Summary

The unit test programs the CAN FD driver to condition both CANFD interfaces to recognize Range IDs in specified ranges. The test messages are transmitted from the PCAN test system utility. This test uses both CANFD interfaces and both CANFD instances to concurrently receive and detect. Note this test requires two PCAN test system probes operating concurrently and independently.

range ID start	range ID end
C1	C1
C2	C4
CC	CE
D6	D8

The Range ID tests consist of transmitting CANFD messages with various Range IDs, and check the statistics of the unit test. Messages with IDs within the defined receive ranges are logged by the unit test.

Unit Test Running in automation

Automation conditions the unit test for Range ID test and then using the command line PCAN utility on two separate and concurrent OS command lines, transmits multiple messages both within and outside of the configured unit test ID ranges.

In this example, one PCAN test system probe is available at device '/dev/pcan32' and the other is available at '/dev/pcan33'. The command lines for both are shown here,

To one PCAN test system:

```
echo Send Message Identifier; 0xc1

~/peak-linux-driver-8.5.1/test/pcanfdtst tx -n 1 --fd -ie 0x000000C1 -c 40M -b 1M -d 8M -l 64 -v -P 500m /dev/pcan32 | tee PCAN_Logfile.txt
```

And to the other PCAN test system:

```
echo Send Message Identifier; 0xc1

~/peak-linux-driver-8.5.1/test/pcanfdtst tx -n 1 --fd -ie 0x000000C1 -c 40M -b 1M -d 8M -l 64 -v -P 500m /dev/pcan33 | tee PCAN_Logfile.txt
```

These command send 1 (number defined by "-n") message with Range ID = 0xC1.

PCAN produces the log file PCAN_Logfile.txt. An example log file is shown below:

```
start opening 1 devices:
```

```
opening "/dev/pcan32" with flags=ca000004h bitrate=1000000 bps sample_pt=0 dbitrate=8000000 bps dsample_pt=0 clock=40000000 Hz
"/dev/pcan32" opened (fd=3)
running 1 loops
1584470708.689208 /dev/pcan32 > BUS STATE=ACTIVE [Rx:0 Tx:0]
1584470708~699470 /dev/pcan32 < 000000c1 .e... [00 00 00 00 00 00 00 00 00 00]
[00 00 00 00 00 00 00 00 00 00]
[00 00 00 00 00 00 00 00 00 00]
[00 00 00 00 00 00 00 00 00 00]
[00 00 00 00 00 00 00 00 00 00]
[00 00 00 00 00 00 00 00 00 00]
[00 00 00 00]
stop test after 1 loops
end of test loop (tst=1).
/dev/pcan32 < [packets=1 calls=1 bytes=64 eagain=0]
all 1 devices closed
--- stop logging ---
sent frames: 1
```

Test automation then enables the stats output and transmits a triggering message. The stats output is then checked for the correct number of messages received at both CAN FD interfaces.

Example Test Log

Please refer to release document file "mmwave_sdk_<ver>\docs\test\AWR68\canfd_test\xwr68_R4_CANFD_Range_ID,_Dual_instance_<timestamp>.log" Note that the test log identifies the message statistics from each instance/interface,

```
Debug: Range Message Identifier Instance 0
```

NOTE: The test is supported in the xWR68 unit test only. This test requires EVM to support CAN FD transceivers on both interfaces to successfully execute this unit test. Please refer to EVM documentation to understand EVM capabilities.

3. PCAN-USB Test Utility

CAN and CAN FD compatibility tests are performed between the mmWave SDK on the EVM to a PC using the PCAN Test Utility.

PCAN is a product of PEAK Systems. Refer to this link for more information, https://www.peak-system.com/PCAN-USB-FD.365.0.html?&L=1.

The product includes an interface cable (see below) and PC software/drivers.

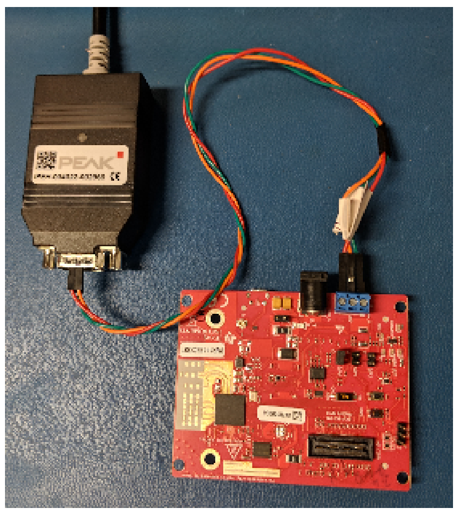
The User Guide is available at https://www.peak-system.com/produktcd/Pdf/English/PCAN-USB-FD_UserMan_eng.pdf

The Unix command line utility is available at https://www.peak-system.com/fileadmin/media/linux/index.htm

The Windows utilities are available at https://www.peak-system.com/quick/DrvSetup

3. 1. PCAN-USB board Setup

The PCAN device connects between a PC USB connector and the EVM CAN interface connector:



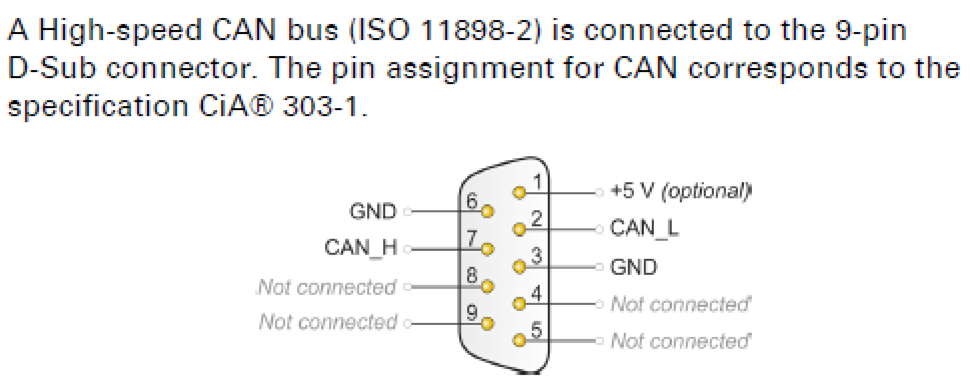

A High-speed CAN bus (ISO 11898-2) is connected to the 9-pin D-Sub connector. The pin assignment for CAN corresponds to the specification CiA® 303-1.

Figure 1: Pin assignment High-speed CAN
(view onto connector of the PCAN-USB FD adapter)

Signal Connections

Signal	EVM connection	PCAN 9-pin D-Sub
CAN-L	CANFD-L, CAN-L	pin 2
GND	GND	pin 3
CAN-H	CANFD-H, CAN-H	pin 7