TEXAS INSTRUMENTS

Application Report
SPRACG9–June 2018

Adding CAN Tx and Rx to an Existing mmWave Project

Raghunandan Kamath

ABSTRACT

This application note describes the steps required to integrate the usage of the CAN (DCAN) interface on the mmWave devices.

Contents

List of Figures

Trademarks

All trademarks are the property of their respective owners.

1 Initializing the Driver

The first step in the integration process is adding code to include and initialize the CAN driver. This driver is required for transmitting and receiving from the CAN interface. The following is C code that initializes the CAN driver. This tested code can be copied into the project.

```
#include <ti/drivers/can/can.h>
/**************************************************************************
 *************************** Global Definitions ***************************
 **************************************************************************/
volatile uint32_t       testSelection = 0;
volatile uint32_t       gTxDoneFlag = 0, gRxDoneFlag = 0, gParityErrFlag = 0;
uint32_t                iterationCount = 0U;
volatile uint32_t       gTxPkts = 0, gRxPkts = 0, gErrStatusInt = 0;
CAN_DCANCfgParams       appDcanCfgParams;
CAN_DCANMsgObjCfgParams appDcanTxCfgParams;
CAN_DCANMsgObjCfgParams appDcanRxCfgParams;
CAN_DCANBitTimeParams   appDcanBitTimeParams;
CAN_DCANData            appDcanTxData;
CAN_DCANData            appDcanRxData;
uint32_t                dataLength = 0U;
uint32_t                msgLstErrCnt = 0U;
uint32_t                dataMissMatchErrCnt = 0U;
uint32_t                rxTicks[DCAN_APP_TEST_MESSAGE_COUNT];
uint32_t                txTicks[DCAN_APP_TEST_MESSAGE_COUNT];
uint32_t                minRxTicks;
uint32_t                maxRxTicks;
uint32_t                minTxTicks;
uint32_t                maxTxTicks;
uint32_t                totalTxTicks;
uint32_t                totalRxTicks;
uint32_t                gDisplayStats = 0;

/**************************************************************************
 *************************** CAN Driver Initialize Function ***********************
 **************************************************************************/
void Can_Initalize(void)
{
    CAN_Handle                canHandle;
    CAN_MsgObjHandle          txMsgObjHandle;
    CAN_MsgObjHandle          rxMsgObjHandle;
    int32_t                   retVal = 0;
    int32_t                   errCode = 0;
    CAN_DCANMsgObjectStats  msgObjStats;
    CAN_OptionTLV             optionTLV;
    CAN_DCANErrorCounter      errCounter;

      /*The pinmux setting for the xWR1443*/
   #if (defined(SOC_XWR14XX))
    /* Setup the PINMUX to bring out the XWR14xx CAN pins */
   Pinmux_Set_OverrideCtrl(SOC_XWR14XX_PINP5_PADAE, PINMUX_OUTEN_RETAIN_HW_CTRL,
PINMUX_INPEN_RETAIN_HW_CTRL);
   Pinmux_Set_FuncSel(SOC_XWR14XX_PINP5_PADAE, SOC_XWR14XX_PINP5_PADAE_CAN_TX);
   Pinmux_Set_OverrideCtrl(SOC_XWR14XX_PINR8_PADAD, PINMUX_OUTEN_RETAIN_HW_CTRL,
PINMUX_INPEN_RETAIN_HW_CTRL);
   Pinmux_Set_FuncSel(SOC_XWR14XX_PINR8_PADAD, SOC_XWR14XX_PINR8_PADAD_CAN_RX);
#else
   /* Setup the PINMUX to bring out the XWR16xx CAN pins */
   Pinmux_Set_OverrideCtrl(SOC_XWR16XX_PINC13_PADAG, PINMUX_OUTEN_RETAIN_HW_CTRL,
PINMUX_INPEN_RETAIN_HW_CTRL);
   Pinmux_Set_FuncSel(SOC_XWR16XX_PINC13_PADAG, SOC_XWR16XX_PINC13_PADAG_CAN_TX);
   Pinmux_Set_OverrideCtrl(SOC_XWR16XX_PINE13_PADAF, PINMUX_OUTEN_RETAIN_HW_CTRL,
PINMUX_INPEN_RETAIN_HW_CTRL);
   Pinmux_Set_FuncSel(SOC_XWR16XX_PINE13_PADAF, SOC_XWR16XX_PINE13_PADAF_CAN_RX);
#endif
/* Configure the divide value for DCAN source clock */
SOC_setPeripheralClock(socHandle, SOC_MODULE_DCAN, SOC_CLKSOURCE_VCLK, 9U, &errCode);
```

```
/* Initialize peripheral memory */
SOC_initPeripheralRam(socHandle, SOC_MODULE_DCAN, &errCode);

/* Initialize the DCAN parameters that need to be specified by the application */
DCANAppInitParams(&appDcanCfgParams, &errCode);
    if (canHandle == NULL)
    {
        System_printf ("Error: CAN Module Initialization failed [Error code %d]\n", errCode);
        return -1;
    }

    /* Set the desired bit rate based on input clock */
    retVal = DCANAppCalcBitTimeParams(DCAN_APP_INPUT_CLK / 1000000,
                                            DCAN_APP_BIT_RATE / 1000,
                                            DCAN_APP_SAMP_PT,
                                            DCAN_APP_PROP_DELAY,
                                            &appDcanBitTimeParams);
    if (retVal < 0)
    {
        System_printf ("Error: CAN Module bit time parameters are incorrect \n");
        return -1;
    }

    /* Configure the CAN driver */
    retVal = CAN_configBitTime (canHandle, & appDcanBitTimeParams, &errCode);
    if (retVal < 0)
    {
        System_printf ("Error: CAN Module configure bit time failed [Error code %d]\n", errCode);
        return -1;
    }

    /* Setup the transmit message object */
    txMsgObjHandle = CAN_createMsgObject (canHandle, DCAN_TX_MSG_OBJ, &appDcanTxCfgParams,
&errCode);
    if (txMsgObjHandle == NULL)
    {
        System_printf ("Error: CAN create Tx message object failed [Error code %d]\n", errCode);
        return -1;
    }

    /* Setup the receive message object */
    rxMsgObjHandle = CAN_createMsgObject (canHandle, DCAN_RX_MSG_OBJ, &appDcanRxCfgParams,
&errCode);
    if (rxMsgObjHandle == NULL)
    {
        System_printf ("Error: CAN create Rx message object failed [Error code %d]\n", errCode);
        return -1;
    }
 }

/***************************************************************************
 ******************** CAN Parameters initialize Function *****************
 ***************************************************************************/
static void DCANAppInitParams(CAN_DCANCfgParams*        dcanCfgParams,
                              CAN_DCANMsgObjCfgParams*  dcanTxCfgParams,
                              CAN_DCANMsgObjCfgParams*  dcanRxCfgParams,
                              CAN_DCANData*             dcanTxData)
{
    /*Intialize DCAN Config Params*/
    dcanCfgParams->parityEnable         = 0;
    dcanCfgParams->intrLine0Enable      = 1;
    dcanCfgParams->intrLine1Enable      = 1;
    dcanCfgParams->testModeEnable       = 0;
    dcanCfgParams->eccModeEnable        = 0;
    dcanCfgParams->stsChangeIntrEnable  = 0;
    dcanCfgParams->autoRetransmitDisable = 1;
    dcanCfgParams->autoBusOnEnable       = 0;
    dcanCfgParams->errIntrEnable         = 1;
    dcanCfgParams->autoBusOnTimerVal     = 0;
    dcanCfgParams->if1DmaEnable          = 0;
    dcanCfgParams->if2DmaEnable          = 0;
    dcanCfgParams->if3DmaEnable          = 0;
    dcanCfgParams->ramAccessEnable       = 0;
    dcanCfgParams->appCallBack           = DCANAppErrStatusCallback;

    /*Intialize DCAN tx Config Params*/
    dcanTxCfgParams->xIdFlagMask       = 0x1;
    dcanTxCfgParams->dirMask           = 0x1;
    dcanTxCfgParams->msgIdentifierMask = 0x1FFFFFFF;

    dcanTxCfgParams->msgValid      = 1;
    dcanTxCfgParams->xIdFlag       = CAN_DCANXidType_11_BIT;
    dcanTxCfgParams->direction     = CAN_Direction_TX;
    dcanTxCfgParams->msgIdentifier = 0xC1;

    dcanTxCfgParams->uMaskUsed    = 1;
    dcanTxCfgParams->intEnable    = 1;

    dcanTxCfgParams->remoteEnable = 0;
    dcanTxCfgParams->fifoEOBFlag  = 1;
    dcanTxCfgParams->appCallBack  = DCANAppCallback;

    /*Intialize DCAN Rx Config Params*/
    dcanRxCfgParams->xIdFlagMask       = 0x1;
    dcanRxCfgParams->msgIdentifierMask = 0x1FFFFFFF;
    dcanRxCfgParams->dirMask           = 0x1;

    dcanRxCfgParams->msgValid      = 1;
    dcanRxCfgParams->xIdFlag       = CAN_DCANXidType_11_BIT;
    dcanRxCfgParams->direction     = CAN_Direction_RX;
    dcanRxCfgParams->msgIdentifier = 0xC1;

    dcanRxCfgParams->uMaskUsed    = 1;
    dcanRxCfgParams->intEnable    = 1;

    dcanRxCfgParams->remoteEnable = 0;
    dcanRxCfgParams->fifoEOBFlag  = 1;
    dcanRxCfgParams->appCallBack  = DCANAppCallback;
 /*Intialize DCAN Tx transfer Params*/
    dcanTxData->dataLength = DCAN_MAX_MSG_LENGTH;
    dcanTxData->msgData[0] = 0xA5;
    dcanTxData->msgData[1] = 0x5A;
    dcanTxData->msgData[2] = 0xFF;
    dcanTxData->msgData[3] = 0xFF;
    dcanTxData->msgData[4] = 0xC3;
    dcanTxData->msgData[5] = 0x3C;
    dcanTxData->msgData[6] = 0xB4;
    dcanTxData->msgData[7] = 0x4B;
}

/***************************************************************************
 ******************** CAN Bit Timing caluculation *****************
 ***************************************************************************/
int32_t DCANAppCalcBitTimeParams(uint32_t               clkFreq,
                                uint32_t                bitRate,
                                uint32_t                refSamplePnt,
                                uint32_t                propDelay,
                                CAN_DCANBitTimeParams*  bitTimeParams)
{
    Double  tBitRef = 1000 * 1000 / bitRate;
    Double  newBaud = 0, newNProp = 0, newNSeg = 0, newSjw = 0, newP = 0;
    Double  nQRef, nProp, fCan, nQ, nSeg, baud, sp, p, newSp = 0;
    float   tQ;

    for (p = 1; p <= 1024; p++)
    {
        tQ    = ((p / clkFreq) * 1000.0);
        nQRef = tBitRef / tQ;

        if ((nQRef >= 8) && (nQRef <= 25))
        {
            nProp = ceil(propDelay / tQ);
            fCan  = clkFreq / p;
            nQ    = fCan / bitRate * 1000;
            nSeg  = ceil((nQ - nProp - 1) / 2);

            if ((nProp <= 8) && (nProp > 0) && (nSeg <= 8) && (nSeg > 0))
            {
                baud = fCan / (1 + nProp + 2 * nSeg) * 1000;

                sp = (1 + nProp + nSeg) / (1 + nProp + nSeg + nSeg) * 100;

                if ((abs(baud - bitRate)) < (abs(newBaud - bitRate)))
                {
                    newBaud  = baud;
                    newNProp = nProp;
                    newNSeg  = nSeg;
                    newSjw   = (nSeg < 4) ? nSeg : 4;
                    newP     = p - 1;
                    newSp    = sp;
                }
                else if ((abs(baud - bitRate)) == (abs(newBaud - bitRate)))
                {
                    if ((abs(sp - refSamplePnt)) < (abs(newSp - refSamplePnt)))
                    {
                        newBaud  = baud;
                        newNProp = nProp;
                        newNSeg  = nSeg;
                        newSjw   = (nSeg < 4) ? nSeg : 4;
                        newP     = p - 1;
                        newSp    = sp;
                    }
                }
            }
        }
    }
    if ((newBaud == 0) || (newBaud > 1000))
    {
        return -1;
    }
    bitTimeParams->baudRatePrescaler    = (((uint32_t) newP) & 0x3F);
    bitTimeParams->baudRatePrescalerExt =
        ((((uint32_t) newP) & 0x3C0) ? (((uint32_t) newP) &0x3C0) >> 6 : 0);
    bitTimeParams->syncJumpWidth = ((uint32_t) newSjw) - 1;

    /* propSeg = newNProp, phaseSeg = newNSeg, samplePoint = newSp
     * nominalBitTime = (1 + newNProp + 2 * newNSeg), nominalBitRate = newBaud
     * brpFreq  = clkFreq / (brp + 1), brpeFreq = clkFreq / (newP + 1)
     * brp      = bitTimeParams->baudRatePrescaler;
     */

    bitTimeParams->timeSegment1 = newNProp + newNSeg - 1;
    bitTimeParams->timeSegment2 = newNSeg - 1;
    return 0;
}
```

2 Register Callbacks

2.1 Tx Complete and Rx Interrupt Callback

The application must implement a callback function to handle transmit complete and receive interrupts.

```
static void DCANAppCallback(CAN_MsgObjHandle handle, uint32_t msgObjectNum, CAN_Direction
direction)
{
    int32_t     errCode, retVal;

    if (direction == CAN_Direction_TX)
    {
        if (msgObjectNum != DCAN_TX_MSG_OBJ)
        {
            System_printf ("Error: Tx callback received for incorrect Message Object %d\n",
msgObjectNum);
            return;
        }
        else
        {
            gTxPkts++;
            gTxDoneFlag = 1;
            return;
        }
    }
    if (direction == CAN_Direction_RX)
    {
        if (msgObjectNum != DCAN_RX_MSG_OBJ)
        {
            System_printf ("Error: Rx callback received for incorrect Message Object %d\n",
msgObjectNum);
            return;
        }
        else
        {
            /* Reset the receive buffer */
            memset(&appDcanRxData, 0, sizeof (appDcanRxData));
            dataLength = 0;

            retVal = CAN_getData (handle, &appDcanRxData, &errCode);
            if (retVal < 0)
            {
                System_printf ("Error: CAN receive data for iteration %d failed [Error code
%d]\n", iterationCount, errCode);
                return;
            }
            /* Check if sent data is lost or not */
            if (appDcanRxData.msgLostFlag == 1)
            {
                msgLstErrCnt++;
            }
             while (dataLength < appDcanRxData.dataLength)
             {
               if (appDcanRxData.msgData[dataLength] != appDcanTxData.msgData[dataLength])
                    {
                       dataMissMatchErrCnt++;
                       System_printf ("Error: CAN receive data mismatch for iteration %d
                at byte %d\n", iterationCount, dataLength);
                    }
                    dataLength++;
                }
            gRxPkts++;
            gRxDoneFlag = 1;
            return;
        }
        }
    }
```

2.2 Error and Status Interrupt Callback

The application must implement a callback function to handle error and status interrupts.

```
static void DCANAppErrStatusCallback(CAN_Handle handle, CAN_ErrStatusResp* errStatusResp)
{
    gErrStatusInt++;
    if (errStatusResp->parityError == 1)
    {
        gParityErrFlag = 1;
    }
    return;
}
```

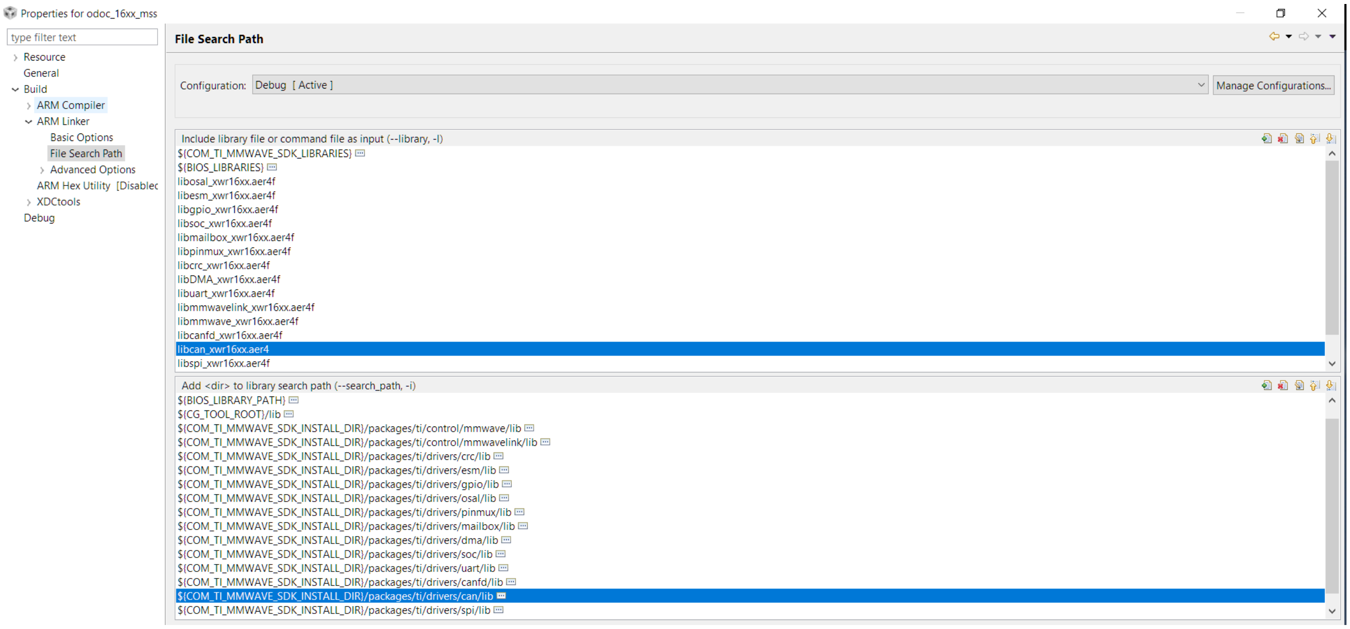
3 CAN Transmit

The following code can be used to transmit CAN data during the initialization and the length message.

```
/* Send data over Tx message object */
        retVal = CAN_transmitData (txMsgObjHandle, &appDcanTxData, &errCode);
        if (retVal < 0)
        {
          System_printf ("Error: CAN transmit data for iteration %d failed [Error code
    %d]\n", iterationCount, errCode);
            return -1;
        }
```

4 Linking the CAN Driver

The final step is to build the executable by linking with the CAN drivers. If using a CCS project, the CAN drivers can be added to the project’s linker properties, as shown in Figure 1.



File Search Path

Configuration: Debug [Active]

Include library file or command file as input (--library, -l)

```
${COM_TI_MMWAVE_SDK_LIBRARIES}
${BIOS_LIBRARIES}
libosal_xwr16xx.aer4f
libesm_xwr16xx.aer4f
libgpio_xwr16xx.aer4f
libsoc_xwr16xx.aer4f
libmailbox_xwr16xx.aer4f
libpinmux_xwr16xx.aer4f
libcrc_xwr16xx.aer4f
libDMA_xwr16xx.aer4f
libuart_xwr16xx.aer4f
libmmwavelink_xwr16xx.aer4f
libmmwave_xwr16xx.aer4f
libcanfd_xwr16xx.aer4f
libcan_xwr16xx.aer4
libspi_xwr16xx.aer4f
```

Add <dir> to library search path (--search_path, -i)

```
${BIOS_LIBRARY_PATH}
${CG_TOOL_ROOT}/lib
${COM_TI_MMWAVE_SDK_INSTALL_DIR}/packages/ti/control/mmwave/lib
${COM_TI_MMWAVE_SDK_INSTALL_DIR}/packages/ti/control/mmwavelink/lib
${COM_TI_MMWAVE_SDK_INSTALL_DIR}/packages/ti/drivers/crc/lib
${COM_TI_MMWAVE_SDK_INSTALL_DIR}/packages/ti/drivers/esm/lib
${COM_TI_MMWAVE_SDK_INSTALL_DIR}/packages/ti/drivers/gpio/lib
${COM_TI_MMWAVE_SDK_INSTALL_DIR}/packages/ti/drivers/osal/lib
${COM_TI_MMWAVE_SDK_INSTALL_DIR}/packages/ti/drivers/pinmux/lib
${COM_TI_MMWAVE_SDK_INSTALL_DIR}/packages/ti/drivers/mailbox/lib
${COM_TI_MMWAVE_SDK_INSTALL_DIR}/packages/ti/drivers/dma/lib
${COM_TI_MMWAVE_SDK_INSTALL_DIR}/packages/ti/drivers/soc/lib
${COM_TI_MMWAVE_SDK_INSTALL_DIR}/packages/ti/drivers/uart/lib
${COM_TI_MMWAVE_SDK_INSTALL_DIR}/packages/ti/drivers/canfd/lib
${COM_TI_MMWAVE_SDK_INSTALL_DIR}/packages/ti/drivers/can/lib
${COM_TI_MMWAVE_SDK_INSTALL_DIR}/packages/ti/drivers/spi/lib
```

Figure 1. CCS Project Linker Properties

If using the makefile, perform the same procedure.

```
###################################################################################
# Additional libraries which are required to build the DEMO:
###################################################################################
MSS_MMW_DEMO_STD_LIBS = $(R4F_COMMON_STD_LIB)                           \
            -llibpinmux_$(MMWAVE_SDK_DEVICE_TYPE).$(R4F_LIB_EXT)        \
            -llibdma_$(MMWAVE_SDK_DEVICE_TYPE).$(R4F_LIB_EXT)           \
            -llibcrc_$(MMWAVE_SDK_DEVICE_TYPE).$(R4F_LIB_EXT)           \
            -llibuart_$(MMWAVE_SDK_DEVICE_TYPE).$(R4F_LIB_EXT)          \
            -llibgpio_$(MMWAVE_SDK_DEVICE_TYPE).$(R4F_LIB_EXT)          \
            -llibmailbox_$(MMWAVE_SDK_DEVICE_TYPE).$(R4F_LIB_EXT)       \
            -llibmmwavelink_$(MMWAVE_SDK_DEVICE_TYPE).$(R4F_LIB_EXT)    \
            -llibmmwave_$(MMWAVE_SDK_DEVICE_TYPE).$(R4F_LIB_EXT)        \
            -llibcli_$(MMWAVE_SDK_DEVICE_TYPE).$(R4F_LIB_EXT)       \
            -llibcan_$(MMWAVE_SDK_DEVICE_TYPE).$(R4F_LIB_EXT)
MSS_MMW_DEMO_LOC_LIBS = $(R4F_COMMON_LOC_LIB)                       \
            -i$(MMWAVE_SDK_INSTALL_PATH)/ti/drivers/pinmux/lib      \
            -i$(MMWAVE_SDK_INSTALL_PATH)/ti/drivers/uart/lib        \
            -i$(MMWAVE_SDK_INSTALL_PATH)/ti/drivers/dma/lib         \
            -i$(MMWAVE_SDK_INSTALL_PATH)/ti/drivers/crc/lib         \
            -i$(MMWAVE_SDK_INSTALL_PATH)/ti/drivers/gpio/lib        \
            -i$(MMWAVE_SDK_INSTALL_PATH)/ti/drivers/mailbox/lib     \
            -i$(MMWAVE_SDK_INSTALL_PATH)/ti/control/mmwavelink/lib  \
            -i$(MMWAVE_SDK_INSTALL_PATH)/ti/control/mmwave/lib      \
            -i$(MMWAVE_SDK_INSTALL_PATH)/ti/utils/cli/lib   \
            -i$(MMWAVE_SDK_INSTALL_PATH)/ti/drivers/can/lib
```

IMPORTANT NOTICE FOR TI DESIGN INFORMATION AND RESOURCES

Texas Instruments Incorporated (‘TI”) technical, application or other design advice, services or information, including, but not limited to, reference designs and materials relating to evaluation modules, (collectively, “TI Resources”) are intended to assist designers who are developing applications that incorporate TI products; by downloading, accessing or using any particular TI Resource in any way, you (individually or, if you are acting on behalf of a company, your company) agree to use it solely for this purpose and subject to the terms of this Notice.

TI’s provision of TI Resources does not expand or otherwise alter TI’s applicable published warranties or warranty disclaimers for TI products, and no additional obligations or liabilities arise from TI providing such TI Resources. TI reserves the right to make corrections, enhancements, improvements and other changes to its TI Resources.

You understand and agree that you remain responsible for using your independent analysis, evaluation and judgment in designing your applications and that you have full and exclusive responsibility to assure the safety of your applications and compliance of your applications (and of all TI products used in or for your applications) with all applicable regulations, laws and other applicable requirements. You represent that, with respect to your applications, you have all the necessary expertise to create and implement safeguards that (1) anticipate dangerous consequences of failures, (2) monitor failures and their consequences, and (3) lessen the likelihood of failures that might cause harm and take appropriate actions. You agree that prior to using or distributing any applications that include TI products, you will thoroughly test such applications and the functionality of such TI products as used in such applications. TI has not conducted any testing other than that specifically described in the published documentation for a particular TI Resource.

You are authorized to use, copy and modify any individual TI Resource only in connection with the development of applications that include the TI product(s) identified in such TI Resource. NO OTHER LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE TO ANY OTHER TI INTELLECTUAL PROPERTY RIGHT, AND NO LICENSE TO ANY TECHNOLOGY OR INTELLECTUAL PROPERTY RIGHT OF TI OR ANY THIRD PARTY IS GRANTED HEREIN, including but not limited to any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information regarding or referencing third-party products or services does not constitute a license to use such products or services, or a warranty or endorsement thereof. Use of TI Resources may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

TI RESOURCES ARE PROVIDED “AS IS” AND WITH ALL FAULTS. TI DISCLAIMS ALL OTHER WARRANTIES OR REPRESENTATIONS, EXPRESS OR IMPLIED, REGARDING TI RESOURCES OR USE THEREOF, INCLUDING BUT NOT LIMITED TO ACCURACY OR COMPLETENESS, TITLE, ANY EPIDEMIC FAILURE WARRANTY AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

TI SHALL NOT BE LIABLE FOR AND SHALL NOT DEFEND OR INDEMNIFY YOU AGAINST ANY CLAIM, INCLUDING BUT NOT LIMITED TO ANY INFRINGEMENT CLAIM THAT RELATES TO OR IS BASED ON ANY COMBINATION OF PRODUCTS EVEN IF DESCRIBED IN TI RESOURCES OR OTHERWISE. IN NO EVENT SHALL TI BE LIABLE FOR ANY ACTUAL, DIRECT, SPECIAL, COLLATERAL, INDIRECT, PUNITIVE, INCIDENTAL, CONSEQUENTIAL OR EXEMPLARY DAMAGES IN CONNECTION WITH OR ARISING OUT OF TI RESOURCES OR USE THEREOF, AND REGARDLESS OF WHETHER TI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

You agree to fully indemnify TI and its representatives against any damages, costs, losses, and/or liabilities arising out of your non-compliance with the terms and provisions of this Notice.

This Notice applies to TI Resources. Additional terms apply to the use and purchase of certain types of materials, TI products and services. These include; without limitation, TI’s standard terms for semiconductor products http://www.ti.com/sc/docs/stdterms.htm), evaluation modules, and samples (http://www.ti.com/sc/docs/sampterms.htm).

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