Revision Control

BRF Integration on WinCE OS, Revision 1.0

Abstract

This document covers the Bluetooth power management integration for Windows CE operating systems. The document describes the BRF power management driver and integration requirements when integrating the BRF driver. This document is based on integration of the BRF driver with the OMAP architecture as an example for general BRF driver - power management integration. The document includes description of the test modes and specific changes in the serial driver that are required when supporting the BRF driver.

Contents

Table of Contentbb

1. Introduction

This document describes the BRF chip support on WinCE devices.

- HCILL power management support for Window CE OS (i.e. Smartphone/PPC). The power management functionality described herein primarily describes support for devices using the HCILL power management protocol and is not intended to provide power management support for serial port devices in general, although some of the modifications to the serial driver may provide general power management support.
- Support for *.bts initialization scripts for all BRF chip versions.

This document describes in detail the TI BRF driver (as released in BSP 8.1.1), this BRF driver will be referenced by name as the Alpha driver.

This document describes the serial driver support for the deep sleep power mode and waking up via system GPIO as implemented on the F-Sample reference design (OMAP850).

This document describes how to support BRF chip on other platforms than OMAP850, based on the Alpha driver code.

Terminology: Alpha driver which known as hcill driver will be called BRF driver since it supports not only hcill but also init script, and in general it is supporting all that BRF chip needed.

Note: Due to the inability of some of the OMAP UART hardware to be awakened by transitions on the RX pin, the proposed design may not be optimized for future devices that do not suffer from this erratum.

The following document relies that the user is acquainted to the HCILL protocol logic and has read the related documentation references.

2. References

Document	Revision
BRF6150 Firmware Release Vendor Specific Commands Host controller Interface (HCI)	BT-SW-0026, Rev 0.91
BRF6150 HCILL – 4-Wire Power Management Protocol	BT-SW-0024, Rev 0.2
WinCE .Net 4.2 Documentation	

3. BRF6100 vs. BRF6150

The BRF6100 and BRF6150 both support the HCILL power management protocol. The difference between the two devices, regarding power management, is that the BRF6150 supports NShut-Down mode.

The BRF6150 can be placed in the NShut-Down mode, providing better power consumption.

The NShut-Down feature is an optional feature that is highly recommended to be integrated into the system.

Supporting the NSHUT-DOWN requires software support on the Host's OS. The control over the NSHUT-DOWN function is done via the host OS and is requires integration with the BRF driver.

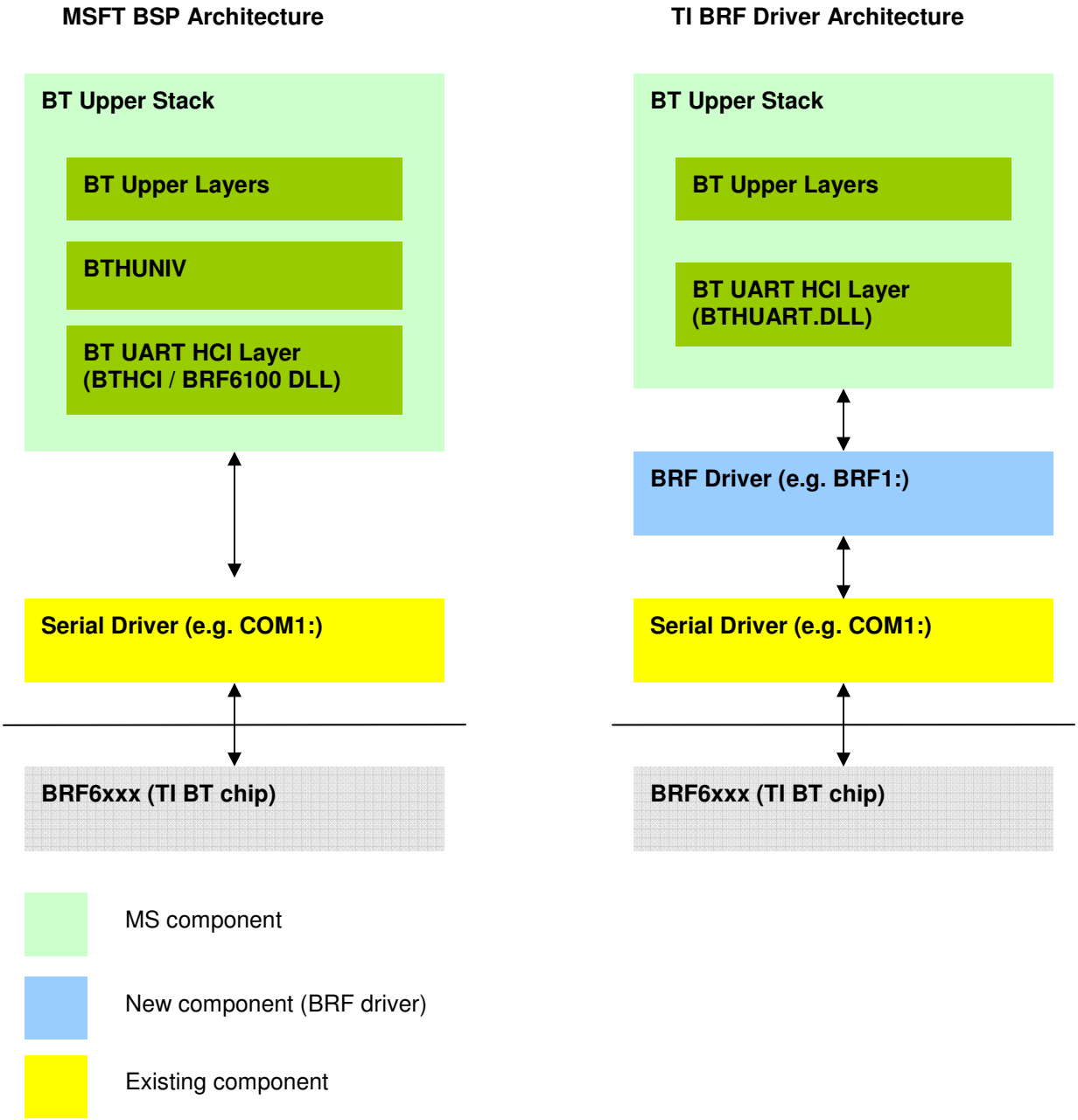
The NSHUT-DOWN feature provides exceptional power saving compared to normal sleep mode and should be integrated into the system when possible.

TI alpha driver handles NshutDown line referenced for OMAP850.

(The BRF6100 device does not support NSHUT-DOWN and does not require extra integration regarding NSHUT-DOWN software support).

4. BRF Driver Architecture

The figure below displays the BRF driver architecture layers above and below the BRF driver. The BRF driver (named hcill.dll) will act as a shim layer between the BT stack's HCI layer and the actual serial driver. Its primary responsibility will be to parse the incoming and outgoing HCI traffic in order to support the HCILL protocol and to run an init script when needed. To that end it will understand the HCILL protocol and its commands as well as the basic structure of HCI packets. **The BT stack will communicate with the HCILL driver as a normal comport**. Instead of using the normal comport named COMx: the BT stack will use the BRF1: as the serial driver.



4.1 Com Port Mapping

To allow the Bluetooth stack to communicate via the BRF driver, the BT stack should open BRF1: as its com port (opens BRF driver), replacing the default mapping to COMx (serial driver).

Example of registry settings to configure the BT stack to open BRF1 and the BRF driver to open COM1 as shown in the diagram (this registry settings are at hcill.reg file provided with TI BRF driver):

```
[HKEY_LOCAL_MACHINE\Software\Microsoft\Bluetooth\Transports\BuiltIn\1]
   "driver"="bthuart.dll"
   "flags"=dword:0
   "name"="BRF1:"
   "baud"=dword:1c200
   "resetdelay"=dword:DAC

[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\HCILL]
   "Prefix"="BRF"                  ; Provides the "BRF" of "BRF1"
   "Dll"="HCILL.Dll"
   "Order"=dword:1                 ; Must be > then actual COM's order value
   "Index"=dword:1                 ; Provides the '1' of "BRF1"
   "name"="COM1:"                  ; Name of actual COM port
   "Priority256" = dword:78        ; Read thread priority
```

5. Using TI Alpha Driver on Magneto

The following actions should be done in order to remove the bthci/brf6100 dll (BSP's original architecture) and add TI alpha driver.

- Delete from platform.bib file the BTHCI.DLL / BRF6100.DLL reference.
- Add to the platform.bib file the HCILL.DLL.
- Remove from platform.reg the registry keys of the BTHCI/ brf6100 driver.
- Add to platform.reg the registry keys as in hcill.reg (hcill.reg contains registry keys for BTHUART.DLL and HCILL.DLL).

Note: TI alpha driver can be used also with OzUp.

6. BRF Driver – SW Implantation

The BRF driver (hcill.dll) will act as a shim layer between the Bluetooth stack's HCI layer and the actual serial driver. Its primary responsibility will be to parse the incoming and outgoing HCI traffic in order to support the HCILL protocol. The HCILL driver will understand the HCILL protocol and its commands as well as the basic structure of HCI packets.

The Bluetooth stack will communicate with the HCILL driver (BRF1:) in the same manner as other COMx: ports.

Since sleep and wake commands can be received asynchronously from the HCILL enabled BT device, the HCILL driver will have a separate thread that will constantly be monitoring incoming traffic for sleep and wake-up requests.

Most of the HCILL protocol support functionality will reside in the COM_Read, COM_Write and the incoming read thread.

6.1 BRF Driver Behavior

The HCILL driver supports Init , Open and Close functions that are called by the BT stack.

6.1.1 BRF_Init

During system start-up the BRF_Init function is called, this functions role is to put the BRF chip in a low power state and wait for the BT stack to be opened.

When BRF6100 is used the device is put to sleep.

When BRF6150 is used the device is turned OFF by setting the NShutDown pin to low.

The BRF_Init function will be called when the mobile device is booting up and performs the following:

- Set NshutDown to high enabling the power-up of the BT device
- Initialize BRF driver SW structure using hcill.reg information.
- Initialize BRF chip :
    - Open COMx:
    - Wait 150msec for BRF boot-up
    - Verify BRF version using Read_Local_Version HCI command
        - Checking whether we use BRF6100 or BRF6150
        - Verify specific version of the chip in order to look for the right init script
    - (IF BRF6100 – enable deep sleep mode , put device to sleep)
    - IF BRF6150 – set NshutDown to low
    - Set COM state to D3 – serial driver in low power mode

6.1.2 BRF_Open

When user turns BT ON / Discoverable, BT stack opens the com port (BRF1:) and BRF_Open function is called.

- If BRF6150 detected during BRF_Init set NShutDown to high, enabling the BRF chip.
- Open comport using the registry entry [HKEY_LOCAL_MACHINE\Drivers\BuiltIn\HCILL] with value "name"="COM1:"
- By choosing COM1: the COM_Open function will be called, serial driver will be used.
- Serial driver mode set to D0
- Wait 150msec to allow the BRF hardware to boot-up
- (IF BRF6100 was detected during BRF_Init , device is woken-up and sleep mode disabled)

6.1.3 BRF_Close

When user turns BT off, BT stack close the co mport (BRF1:) and BRF_Close function is called.

- Wake-up BRF chip
- Disable sleep mode
- Set default baud rate (115200 bits/sec)
- enable sleep mode , BRF will goto sleep
- Set serial state to D3
- If BRF6150 detected set NShutDown to low

6.1.4 HCI_Reset

Every time BT stack sends HCI_Reset, the BRF driver performs the following :

- Detects HCI_Reset command
- Wake-up BRF chip if asleep
- Disable sleep mode
- Send HCI_Reset to BRF chip
- Send initscript according to the BRF chip version found in BRF_Init (read_local_version)
- Enable sleep mode

6.2 BRF driver - Testing Compilation Modes

6.2.1 Using the BRF Driver to Change Baud Rate

This mode of operation is for testing purposes.

The BRF driver includes a `#define INIT_TEST` compilation flag. Usage of this mode allows the driver to run the init script for the sole purpose of changing the BRF chip baud rate (high rate testing). The BRF driver will perform a one-time initialization of the init script (Included in BRF_Init function) and then close the port, allowing the BT stack to communicate directly with the BRF chip (Comx).

The init script should include a change baud rate command with the required rate and settings. Please note that all other operation performed in the init script will be cleared due to the fact that the BT stack will send HCI_Reset when opening the COMx.

Using this method we can measure throughput with and with out BRF driver for comparison.

Please note that the BT stack should open the COMx with the new default baud rate. The registry key must be updated to support this testing mode.

```
[HKEY_LOCAL_MACHINE\Software\Microsoft\Bluetooth\Transports\BuiltIn\1]
   "driver"="bthuart.dll"
   "flags"=dword:0
   "name"="COM1:"
   "baud"=dword:e1000                    ; Baud rate in registry key matches Initscript
   "resetdelay"=dword:DAC
```

6.2.2 Disabling Sleep Mode

This mode of operation is for testing.

The BRF driver includes a compilation flag to disable the sleep mode of the BRF device. This mode allows testing of the BRF driver without enabling sleep mode. The BRF driver is active and the read thread and parsing takes place, but without HCILL transactions.

To enable this mode the `#define NO_SLEEP` compilation flag should be enabled.

6.2.3 Disabling the BRF driver

This mode of operation is for testing.

The BRF driver can be disabled to allow the BT stack to communicate directly to the BRF device without passing through the BRF driver and HCILL logic. To disable the driver the following actions are required:

- Disabling the [HKEY_LOCAL_MACHINE\Drivers\BuiltIn\HCILL]
  Remark all registry keys regarding HCILL.dll (add ";" at the beginning of each registry key).

```
; "Prefix"="BRF"              ; Provides the "BRF" of "BRF1"
; "Dll"="HCILL.Dll"
; "Order"=dword:1             ; Must be > then actual COM's order value
; "Index"=dword:1             ; Provides the '1' of "BRF1"
; "name"="COM1:"              ; Name of actual COM port
; "Priority256" = dword:78    ; Read thread priority
;  "IClass"="{A32942B7-920C-486b-B0E6-92A702A99B35}"
```

- Direct BTHUART (HCI Transport Layer) to communicate directly with COMx instead of BRF1.

```
[HKEY_LOCAL_MACHINE\Software\Microsoft\Bluetooth\Transports\BuiltIn\1]
"driver"="bthuart.dll"
"flags"=dword:0
"name"="COM1:"              ; Point to appropriate comport
"baud"=dword:1c200
"resetdelay"=dword:DAC
```

7. Serial Port Driver Modifications

7.1 System Integration of the BRF Driver

The BRF driver is not a stand alone driver, but requires some special functionality from the serial driver to allow deep sleep and wake-up operation.

7.1.1 UART RX to GPIO Interrupt

Since the OMAP devices are not interruptible on the RX pin, muxing to RX to an interruptible GPIO is required to allow detection of a wake-up request from the BRF. The GPIO has to be associated with a UART system interrupt (pHWObj->dwIntID).

Pseudo code example :

```
GPIOSetMode(pPdd->hGPIO, 41, GPIO_DIR_INPUT|GPIO_INT_HIGH_LOW);
size = sizeof(irqs);
irqs[3] = GPIOGetIrq(pPdd->hGPIO, 41);
KernelIoControl(IOCTL_HAL_REQUEST_SYSINTR, irqs, size, &pPdd->sysIntr,
sizeof(pPdd->sysIntr), NULL)
pHWObj->dwIntID = pPdd->sysIntr;
```

7.1.2 Interrupt Type

The PDD GetIntType function when called from the MDD will return an interrupt type of INTR_RX when in wakeup mode. The wakeup_mode flag is active during serial driver transition to D3.

Pseudo code example :

```
if (pPdd->wakeUpMode) {
    if (!pPdd->wakeUpSignaled) {
        type = INTR_RX;
        pPdd->wakeUpSignaled = TRUE;
    }
    goto cleanUp;
}
```

7.1.3 Faking the wake-up byte

Since the wake-up byte sent to the host is not physically received by the RX buffer but used to create an interrupt on the GPIO the serial driver is required to create a fake wakeup_char in the RX buffer (length = 1 byte). This wakeup char is parsed by the BRF driver as a normal wakeup request, responding with wakeup ack.

Pseudo code example :

```
if (pPdd->wakeUpMode) {
   *pRxBuffer = (UCHAR)pPdd->wakeUpChar;
   *pLength = 1;
}
```

7.2 Power Modes States

The basic UART driver does not support HCILL and some modifications are to be made in order to support HCILL.

The serial interface supports power management modes D0 and D3. This mode can be read as a status from the serial driver as follows:

WinCE OS terminology uses the following power states:

D0 – Device fully awake

D3 – Device in sleep mode

7.2.1 D3 Mode

Ideally, when the serial driver receives an IOCTL_POWER_SET command requesting that the power state be changed to D3, the serial driver will disable the associated UART clock. Subsequently a transition on the RX pin would wake the OMAP and UART communication would commence.

Since the current generation of OMAP chips have an erratum that does not allow a transition on the RX pin to wake up the device, we must provide a software work around to allow a device connected to the serial port to be able to wake up the OMAP. During this transition to D3, we must map the RX pin of the associated UART to its GPIO mode and to enable an IST associated with the GPIO.

Once in GPIO mode, the OMAP can be awakened by a high to low transition of the GPIO pin. After the OMAP is awaken by the GPIO interrupt, the UART driver will re-map the RX pin to UART mode and the UART clock will be enabled when serial driver changes state to D0. Since the HCILL protocol specifies that no other byte will be sent until the BT device receives the HCILL_WAKE_UP_ACK (the wakeup acknowledgement), we can drop the first byte and safely assume that it was an HCILL_WAKE_UP_IND. If we wait one bytes time to enable the UART clocks, we should be able to safely assume that the UART will not detect a received byte of data until after the HCILL_WAKE_UP_ACK is sent.

After being awaken, serial driver should fake a reception of a the wake up indication byte (0x32 HCI_WAKE_UP_IND.

Note: The only byte that is allowed to send when HCILL protocol synchronized both sides to enter sleep mode is the 0x32 HCILL_WAKE_UP_ACK .

Note: When using other CPUs than OMAP one can use mux as in OMAP or connect UART Tx line from BRF chip to Host UART Rx **and** to an additional interruptible GPIO.

7.2.2 D0 Mode

When the serial driver receives an IOCTL_POWER_SET command requesting that the power state be changed to D0, the serial driver will enable the associated UART clock. When the UART RX pin is currently mapped to GPIO mode, the driver will re-map (disable the GPIO functionality) the RX pin to UART mode.

8. MSFT BT Stack Behavior

The BT stack at start-up will open automatically to check if BT hardware device is attached to the system.
The BT stack will communicate with the BRF driver as a normal comport (stream driver API).
At start-up the following sequence is performed:

- BT stack opens comport using registry entry [HKEY_LOCAL_MACHINE\Software\Microsoft\Bluetooth\Transports\BuiltIn\2]
- "name"="BRF1:"
- By choosing BRF1 the BRF_Open function will be called.
- BT stack sends HCI_Reset (Which triggered BRF driver to send init script and enables deep sleep mode)
- BT stack sends configuration commands and verifies BT hardware is present.
- After Initialization the BT stack closes the comport (BRF_Close function is called).
- The com port will be reopened when user enables BT in GUI (BT On or BT discoverable).
- The com port will be closed when the user turns BT OFF.

- Please note: on some occasions of BT OFF the MSFT BT stack will toggle Open com port / Close com port one more time after the comport has been closed.

9. TI Alpha Driver vs. OzUp and Magneto

9.1 Ozone Update BSP

At the original BSP releases, the OzUp HCILL driver is based on the BRF6100 BT device. Usage of the OzUp HCILL driver with BRF6150 requires some modifications to operate the NshutDown pin.

TI alpha driver controls NshutDown already line based on the F-Sample reference design.

When using a different platform the NShutDown should be controlled accordingly.

9.2 Magneto BSP

At the original BSP releases, the Magneto HCILL driver, named BTHCI or BRF6100 is yet to be fully tested and usable as provided in the RTM release, in addition current BSP supports BRF6100 and not BRF6150.

Texas Instruments recommends using TI alpha driver instead of the BTHCI / BRF6100 DLL provided in the BSP.

9.3 TI Alpha Driver

It is recommended to use TI alpha driver for projects that are based on the UzUp and Magneto.

For usage of the Alpha driver on other platforms then OMAP F-Sample, minor modifications are required:

- The NShutDown control should be mapped to the correct pin.
- Certain OMAP related .h files should not be included since they contain platform specific variables.

Note: The differences between the driver released in BSP 2.x project (OMAP730+BRF6100 project) and the TI Alpha driver are:

- Checking the BRF chip type (BRF6100/BRF6150) and the specific version and acts accordingly
- The Alpha code had been optimized and tested and is suitable for out-of-the-box usage on F-Sample based projects.

10. Appendix A - Using BRF6150 Without NShutDown

Usage the BRF6150 without NShutDown functionality is NOT recommended.

In order to use the BRF6150 hardware without NShutDown control the BRF functions (Init, open & close) need to be modified slightly to behave like the BRF6100 device.

11. Appendix B – NshutDown Function Fix

SetNShutDown function as provided in alpha driver is not setting nshutdown to low. The fix is to implement 'else' case and set the line to low when the function parameter ' is_set' is FALSE.

12. Appendix C – BRF_Open Function Fix

The BRF_Open function contains a check if the function has allready been run before. This check should be moved to the beginning of the BRF_Open function. If the OpenCount value is not equal to ZERO then return NULL. This will make sure that the function check is before the NShut-Down toggle and will make sure that an BRF_Open tread is not run when BRF_Close is in process.

Code section required to be moved :

```
if (pHcillHead->dwOpenCount)
{
    DEBUGMSG(ZONE_OPEN | ZONE_WARN, (TEXT("B: Device already openned.
    BRF_Open failed\n\r")));
    return NULL;
}
```

13. Appendix D – Optimizing BRF_Close time on BRF6150

Further optimization of the BRF_Close function may be done by removing the change baudrate section. BRF_Close is acompanied by NShutDown low and therfor there is no requirment to change BRF6150 baudrate back to default 115200 before NshutDown High to Low transition.

Code section required to be removed :

```
HcillSetState(pOpenHead, HCILL_STATE_AWAKE, FALSE);
HcillDefUartHCIBaudrate(pHcillHead->hComPort);
SetEvent(pHcillHead->hStateChangeAck);
```

When using BRF6150 and NShutDown is implemented we can futher shorten the Bluetooth ON to OFF time. At BRF_Close function we can remove the function that waits for an event (sleep event , 2000). This function waits for a sleep event or 2000 ms timeout. Since the BRF6150 does not go to sleep the driver will wait 2000 ms!

Important Notice

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgement, including those pertaining to warranty, patent infringement, and limitation of liability.

TI warrants performance of its semiconductor products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain applications using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications"). TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE–SUPPORT DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS. INCLUSION OF TI PRODUCTS IN SUCH APPLICATIONS IS UNDERSTOOD TO BE FULLY AT THE CUSTOMER'S RISK.

In order to minimize risks associated with the customer's applications, the customer to minimize inherent or procedural hazards must provide adequate design and operating safeguards.

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used. TI's publication of information regarding any third party's products or services does not constitute TI's approval, warranty or endorsement thereof.