Implementation of the ISO15693 Protocol in the TI TRF796x

ShreHarsha Rao

ABSTRACT

This application note discusses the anti-collision sequence of the ISO15693 standard implemented in the MSP430F2370 (a 16-bit ultra-low power microcontroller from the TI MSP430 family) used with Texas Instruments' TRF796x, a fully integrated 13.56-MHz radio frequency identification (RFID) analog front end and data framing reader system.

This document is designed for use by customers who are experienced with RFID and firmware development and want to develop their own application using the TRF796x. This reference guide should be used in conjunction with the ISO15693 standard which specifies the standard protocol, commands and other parameters required for communication between the transponder and the reader.

Contents

List of Figures

List of Tables

1 Anti-Collision Sequence for ISO15693

The following describes the inventory procedure/anti-collision sequence for the ISO15693.

The VCD sends a mask value and number of slots along with the inventory request. The VICC compares the least significant bits of its UID to the slot counter + mask value. If it matches, it sends a response. If only one VICC responds, then there is no collision and the VCD receives the complete UID. If the reader detects a collision, it makes note of the slot number in which collision occurred. The reader sends an EOF to switch to the next slot. The VICC increments its slot counter on reception of EOF. This is repeated for all 16 slots. At the end of 16 slots, the slot pointer contents are examined. If it is not zero, it means that collision has occurred in one or more slots. A new mask value is calculated and the inventory request is sent with the new mask. This is repeated until there are no collisions.

Consider four VICCs with UIDs E00700000000012A, E00700000000032A, E00700000000045A and E007000000000345 in the read range of the reader. For convenience, the UIDs are represented as x12A, x32A, x45A and x345 respectively.

The first row in the tables below represents the slot number and their hexa-decimal representation in brackets. The second row shows the UIDs of the VICCs that respond in that slot.

Round 1: Mask value = 0, Mask length = 0

0	1	2	3	4	5	6	7	8	9	10(A)	11(B)	12(C)	13(D)	14(E)	15(F)
–	–	–	–	–	x345	–	–	–	–	x12A x32A x45A	–	–	–	–	–

The reader sends out an Inventory request with mask value and length zero. The VICC compares the LSB of its UID with the slot number + mask. Since the mask value is zero in Round 1, the VICCs compare their UIDs to the slot number only. Thus, each VICC in the reader's read range will find a match and will respond in one of the 16 slots.

In this example, in slot 5, the VICC with UID x345 responds. Since only one VICC responded in this slot, there is no collision and the reader receives the UID completely. The reader raises an interrupt with the End of RX flag set in the interrupt status register of the TRF796x. For more details, refer to Section 2 on interrupts.

In slot 10, VICCs with UIDs x12A, x32A and x45A respond since the last four bits of their UID match 10(A). This collision is noted by the reader and it raises an interrupt with the collision flag set in the interrupt status register. The microcontroller services this interrupt and makes note of the slot number in which collision occurred.

In all other slots, the VCD does not receive any response from the VICCs. The reader waits for a predefined amount of time before sending the EOF to switch to the next slot. The reader raises an interrupt with the no-response flag set to inform the micro to send the 'transmit next slot' command.

Note: The reader should send an EOF at the end of each slot to switch to the next slot, irrespective of the VICC response. This can be done via a direct command to the reader. This is explained in Section 1.4.

Round 2: Mask value = A, Mask length = 4

0	1	2	3	4	5	6	7	8	9	10(A)	11(B)	12(C)	13(D)	14(E)	15(F)
–	–	x12A x32A	–	–	x45A	–	–	–	–	–	–	–	–	–	–

In round 1, it was noted that collision occurred in slot 10. Hence the new mask value = 10 (new mask value = slot number + old mask value) and the new mask length is incremented by 4 bits. The VCD sends out another inventory request with mask value = A and mask length = 4. Now the VICCs compare the LSB of their UID with slot number + mask value (A). Thus in slot 2, VICCs x12A and x32A respond since the LSB of their UID matches 2A. The VICC with UID x45A responds in the 5th slot.

Round 3: Mask value = 2A, Mask length = 8

0	1	2	3	4	5	6	7	8	9	10(A)	11(B)	12(C)	13(D)	14(E)	15(F)
–	x12A	–	x32A	–	–	–	–	–	–	–	–	–	–	–	–

Since there was another collision in slot 2 of Round 2, the reader sends out another Inventory request with mask value = 2A (collision slot number + old mask value) and mask length = 8 (incremented by 4 bits). VICC with UID x12A responds in slot 1 and VICC with UID x32A responds in slot 3. Since there are no collisions detected in this round, the anti-collision sequence ends here.

1.1 Pseudo-Code for ISO15693 Anti-Collision Algorithm

The default configuration after power-up supports ISO15693, single sub-carrier, high data rate, 1-out-of-4 operation. The low-level option registers (0x02…0x0B) are automatically set to optimally adapt the circuitry to the protocol demands.

1. Check bit B5 of flag in the inventory request. If set, number of slots = 1; else number of slots = 16; If number of slots = 16, enable no response interrupt.
2. Initialize mask length and mask value to zero.
3. Initialize slot number pointer to zero.
4. Send inventory request with mask length and mask value.
5. Wait for end of transmit interrupt.
6. Wait for next interrupt. This can be due to any of the following:
   a. End of RX
   b. Collision
   c. No response

   Check the IRQ status register to determine the cause of the interrupt (for more details, refer to Section 2 on interrupts).

   If interrupt is due to End of RX, this means that UID is received in the FIFO without any error/collision. Read the FIFO to obtain the complete UID.

   If interrupt is due to collision, note the slot number in the slot number pointer. Increment the slot number pointer.

   If interrupt is due to no response from the VICC, ignore.
7. Reset FIFO.
8. If number of slots is 16, transmit EOF. If number of slots is 1, exit.
9. Repeat steps 5 and 6 for all 16 slots. At the end of 16 slots, disable no response interrupt.
10. Examine slot number pointer. If not zero, calculate new mask. If zero, exit.
    A. Increment mask length by 4 bits.
    B. Calculate new mask = slot number (in which collision occurred) + old mask
11. Go to step 4 (new mask value and length).
12. Decrement slot pointer by 1.
13. Go to step 10.

1.2 Procedure to Initiate Transmission (Send Inventory Request Command)

Note: The general procedure to start transmission is described below. **This is applicable to all commands that need to be transmitted to the tag.**

The data/command that is to be transmitted is written in to the FIFO, a 12 byte buffer. Transmission starts when the first data byte is written into FIFO. The reader adds SOF, EOF and CRC to the request packet before transmitting.

1. Start condition
2. Send reset command 0x0F (command mode – 0x8F)
3. Send transmission command (0x90 - without CRC or 0x91 – with CRC)
4. Continuous write to register 0x1D (0x3D)
5. Data for register 0x1D (upper and middle nibble of the number of bytes to be transmitted)
6. Data for register 0x1E (lower nibble of the number of bytes to be transmitted)
7. Data byte(s) for FIFO
8. Stop condition

The FIFO can be written to (and read from) in continuous mode only.

For details on the Start and Stop conditions, refer to the timing diagrams for SPI/Parallel mode.

The inventory request format (according to the ISO 15693-3 spec) is as follows:

SOF	FLAGS	INVENTORY COMMAND	MASK LENGTH	MASK VALUE	CRC	EOF
	8 bits	8 bits	8 bits	0 to 8 bytes	16 bits	

As mentioned earlier, the SOF, CRC and EOF will be added automatically by the reader. Only the flags, inventory command, mask length and value have to written to the FIFO for transmission.

Pseudo-code:

buf is an array that holds all the command/data bytes that are to be sent to the reader.

size is the number of bytes to be transmitted.

flags is the ISO15693 flag byte in the Inventory Request command format.

length is the mask length.

mask is the mask value.

```
buf[0] = 0x8f;                          /* Reset FIFO command */
buf[1] = 0x91;                          /* Send with CRC */
buf[2] = 0x3d;                          /* Write continuous from register 1D */
buf[3] = (char) (size >> 8);            /* Data for register 1D */
buf[4] = (char) (size << 4);            /* Data for register 1E */
buf[5] = 0x05;                          /* ISO15693 flag with 16 slots bit set*/
buf[6] = 0x01;                          /* ISO15693 anti collision command code */
buf[7] = length;                        /* Mask length */
If (length > 0)
{
      for (i = 0; i < masksize; i++)
      buf[i + 8] = *(mask + i);         /* Mask value */
}
```

Write buf[0] to buf[i + 8] to TRF796x via SPI or Parallel mode (refer to the Parallel/SPI timing diagrams in the TRF7960-61 data sheet, SLOU186).

1.3 Procedure to Send EOF (to Switch to Next Slot)

The VCD sends an EOF command to the VICC at the end of every slot to switch to the next slot. If you are using SPI mode, special procedures need to be followed to send EOF. Please refer to the application note (SLOA140),Using the SPI Interface with TRF7960, for detailed information.

1. Send Block RX command (0x16) (command byte – 0x96)
2. Send Enable RX command (0x17) (command byte – 0x97)
3. Send Transmit Next Slot (EOF) command (0x14) (command byte – 0x94)

1.4 Procedure to Enable No Response Interrupt

During an inventory process, when the VCD receives no response from the VICC, it shall wait for a preset time (defined in RX no response wait time register) before sending the EOF. The TRF796x informs the microcontroller when to send the EOF command to the TRF796x via a no response interrupt. This is enabled by setting bit B0 in the collision position and Interrupt mask register (CPIM). Thus if there is no response within the defined time, an interrupt request is sent and a flag is set in the IRQ status register.

Pseudo-code:

Read the CPIM register value.

Set bit B0.

Write the final value to the CPIM register.

The CPIM register has masks for other interrupts like collision, errors, etc. To ensure that the mask status is maintained for other interrupts, the CPIM register is read first and then the bit zero (corresponding to no response interrupt) alone is set in the value obtained from the read. Alternately, if the mask status is known for all the interrupts, the register can be directly written to with the desired value, thereby avoiding the read operation.

A detailed description of the firmware implementation of the anti-collision sequence is given in Figure 1 and Figure 2 in the form of a flowchart.

Note: Due to the recursive nature of the anti-collision algorithm, there is a risk of stack overflow when collision occurs. It is highly recommended that the user implement stack overflow check in the firmware.

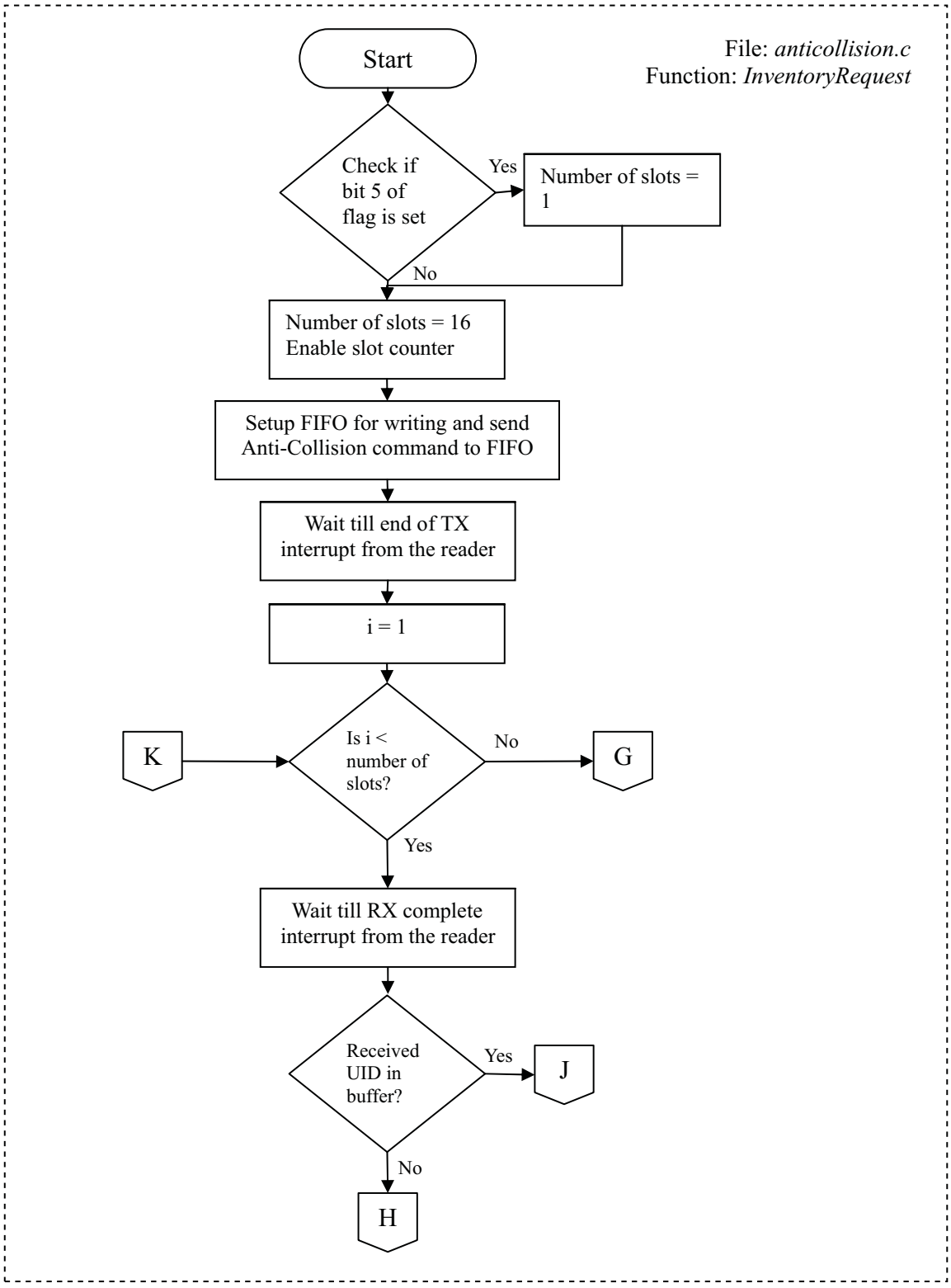

Figure 1. InventoryRequest (1)

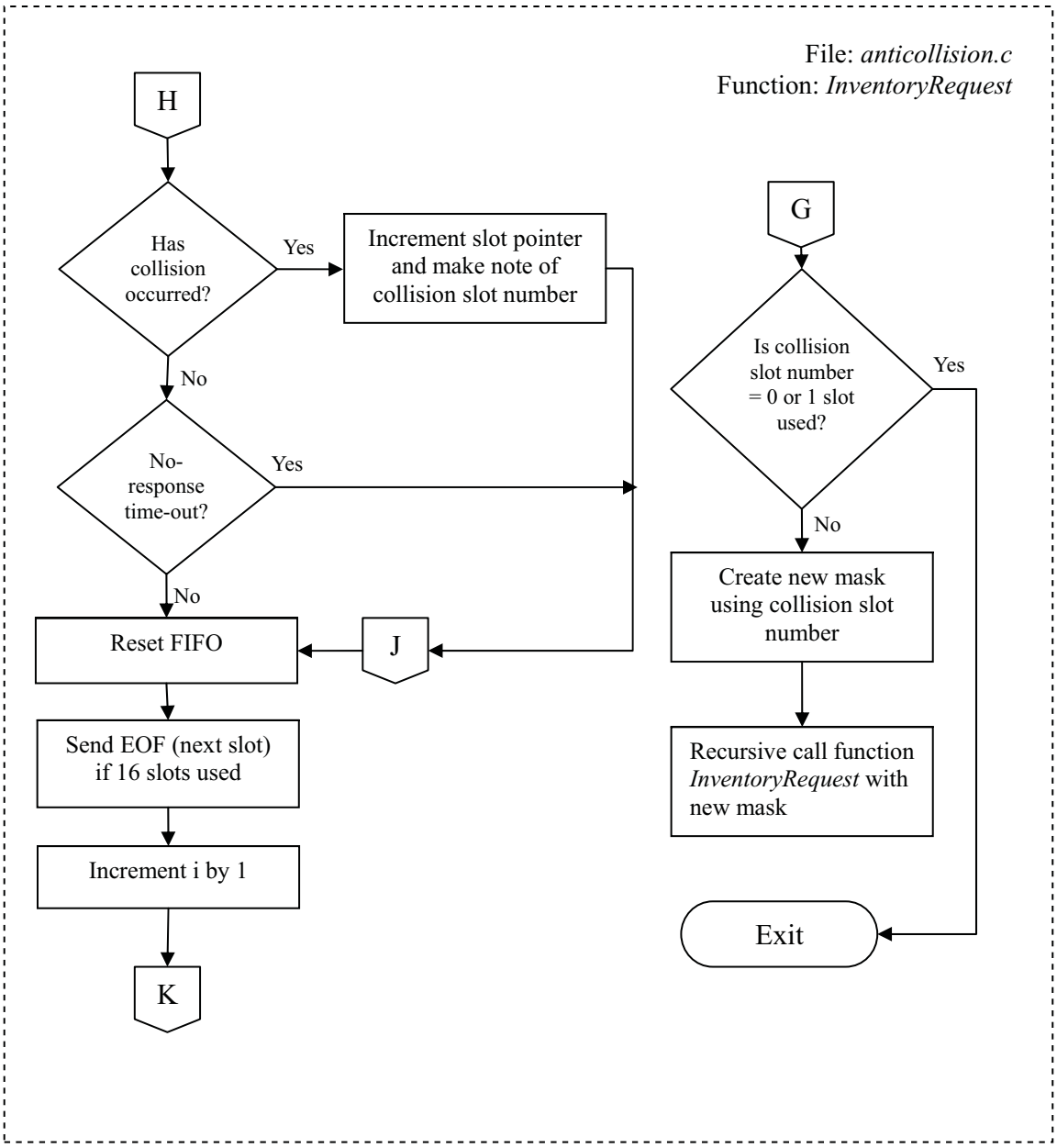

Figure 2. InventoryRequest (2)

1.5 Read Multiple Blocks

The physical memory of an ISO15693 VICC is organized in the form of blocks or pages of fixed size. Up to 256 blocks can be addressed and a block size can be up to 32 bytes.

According the ISO15693-3 spec, the format for the read multiple blocks command is as follows:

1	2	3	4	5	6	7	8
SOF	**FLAGS**	**READ MULTIPLE BLOCKS**	**UID**	**FIRST BLOCK NUMBER**	**NUMBER OF BLOCKS**	**CRC**	**EOF**
	8 bits	8 bits	64 bits	8 bits	8 bits	16 bits	

Note: UID is optional. If the UID is not included, the command can be executed by any VICC in the vicinity of the reader. While if UID field is included, only that VICC whose UID matches the UID specified in the command will respond.

The Number of Blocks field is one less than the number of blocks that the VICC shall return its response.

For more details on the Flags field, please refer to the ISO15693-3 spec.

Except for the SOF, CRC and EOF, all the command fields (2 to 7) have to be placed in the FIFO for transmission.

Pseudo-code (for read multiple blocks command with no UID):

buf is an array that holds all the command/data bytes that are to be sent to the reader.

size is the number of bytes to be transmitted.

In this case, size = 4 (flags + command code + first block number + number of blocks)

flags is the ISO15693 flags byte.

```
buf[0] = 0x8f;                  /* Reset FIFO command */
buf[1] = 0x91;                  /* Send with CRC */
buf[2] = 0x3d;                  /* Write continuous from register 1D */
buf[3] = (char) (size >> 8);    /* Data for register 1D */
buf[4] = (char) (size << 4);    /* Data for register 1E */
buf[5] = 0x00;                  /* ISO15693 flag with option flag not set*/
buf[6] = 0x23;                  /* Read multiple blocks command code */
buf[7] = First block number
buf[8] = Number of blocks
```

1. Write buf[0] to buf[8] to TRF796x in a continuous write mode via SPI or Parallel mode (refer to the Parallel/SPI timing diagrams in the TRF7960-61 data sheet, SLOU186).
2. Wait for a End of TX interrupt (use a timer for timeout).
3. Wait for next interrupt (use a timer for timeout). This can be due to any of the following:
   a. End of RX
   b. Collision

   Check the IRQ status register to determine the cause of the interrupt (for more details, refer to Section 2 on interrupts).

   If interrupt is due to End of RX, this means that the response is received in the FIFO without any error/collision. Read the FIFO to obtain the block data.

   If interrupt is due to collision, the user can choose what to do next – try again (repeat from step 1) or ignore.

Pseudo-code (for read multiple blocks command with UID):

buf is an array that holds all the command/data bytes that are to be sent to the reader.

size is the number of bytes to be transmitted.

In this case, size = 12 (flags + command code + UID + first block number + number of blocks)

flags is the ISO15693 flags byte.

```
buf[0] = 0x8f;                          /* Reset FIFO command */
buf[1] = 0x91;                          /* Send with CRC */
buf[2] = 0x3d;                          /* Write continuous from register 1D */
buf[3] = (char) (size >> 8);            /* Data for register 1D */
buf[4] = (char) (size << 4);            /* Data for register 1E */
buf[5] = 0x00;                          /* ISO15693 flag with option flag not set*/
buf[6] = 0x23;                          /* Read multiple blocks command code */
buf[7] to buf[14] contains UID
buf[15] = First block number
buf[16] = Number of blocks
```

According to the ISO 15693 protocol, a multiple-byte field (here, the UID field) is transmitted least significant byte first. For example, consider a tag with UID = E0007000006D6AC1C, then:

```
buf[7] = 1C;
buf[8] = AC;
buf[9] = D6;
buf[10] = 06;
buf[11] = 00;
buf[12] = 00;
buf[13] = 07;
buf[15] = E0;
```

1. Write buf[0] to buf[16] to TRF796x in a continuous write mode via SPI or Parallel mode (Refer to the Parallel/SPI timing diagrams in the TRF7960-61 data sheet, SLOU186).
2. Wait for a End of TX interrupt (yse a timer for timeout).
3. Wait for next interrupt (use a timer for timeout). This can be due to any of the following:
   a. End of RX
   b. Collision

   Check the IRQ status register to determine the cause of the interrupt (for more details, refer to Section 2 on interrupts).

   If interrupt is due to End of RX, this means that the response is received in the FIFO without any error/collision. Read the FIFO to obtain the data received from the tag. Check for the Error_flag in the Flags field. If set, the error code gives information about the type of error that occurred. Otherwise, data has been received without any error.

   If interrupt is due to collision, the user can choose how to act – try again (repeat from step 1) or ignore.

1.6 Write Multiple Blocks

Note: Most tags do not support the write multiple blocks command. Hence the user should know beforehand if the tag supports this command. If not, the 'write single block command can be used multiple times to write to many blocks.

The procedure to write multiple blocks is described below.

The write multiple blocks request format is as shown:

1	2	3	4	5	6	7	8	9
SOF	**FLAGS**	**READ MULTIPLE BLOCKS**	**UID**	**FIRST BLOCK NUMBER**	**NUMBER OF BLOCKS**	**DATA**	**CRC**	**EOF**
	8 bits	8 bits	64 bits	8 bits	8 bits	Block length	16 bits	
						Repeat as needed		

Note: UID is optional. If the UID is not included, the command can be executed by any VICC in the vicinity of the reader. While if UID field is included, only that VICC whose UID matches the UID specified in the command will respond.

The Number of Blocks field is one less than the number of blocks that the VICC shall return its response.

For more details on the Flags field, please refer to the ISO15693-3 spec.

Except for the SOF, CRC and EOF, all the command fields (2 to 7) have to be placed in the FIFO for transmission.

Pseudo-code (for write multiple blocks command with no UID):

buf is an array that holds all the command/data bytes that are to be sent to the reader.

size is the number of bytes to be transmitted.

flags is the ISO15693 flags byte.

datalength is the number of the data bytes to be written to the tag.

```
buf[0] = 0x8f;                              /* Reset FIFO command */
buf[1] = 0x91;                              /* Send with CRC */
buf[2] = 0x3d;                              /* Write continuous from register 1D */
buf[3] = (char) (size >> 8);                /* Data for register 1D */
buf[4] = (char) (size << 4);                /* Data for register 1E */
buf[5] = 0x00;                              /* ISO15693 flag with option flag not set*/
buf[6] = 0x24;                              /* Read multiple blocks command code */
buf[7] = First block number
buf[8] = Number of blocks
buf[9] to buf[8 + datalength] = Data
```

1. Write buf[0] to buf[8 + datalength] to TRF796x via SPI or Parallel mode (Refer to the Parallel/SPI timing diagrams in the TRF7960-61 data sheet, SLOU186).

The FIFO buffer in the TRF796x is only 12 bytes long. Hence if the total number of bytes to be transmitted is greater than 12, then:

1. Transmit the first 12 bytes.
2. Wait for TX active and 3 bytes left in FIFO interrupt (refer to Section 2 on interrupts for more details).
3. Transmit next 9 bytes or less.

Repeat from step 2 until all the bytes have been transmitted.

To transmit first 12 bytes:

```
buf[0] = 0x8f;                                      /* Reset FIFO command */
buf[1] = 0x91;                                      /* Send with CRC */
buf[2] = 0x3d;                                      /* Write continuous from register 1D */
buf[3] = (char) (total number of bytes to be TX >> 8);   /* Data for register 1D */
buf[4] = (char) (total number of bytes to be TX << 4);   /* Data for register 1E */
buf[5] = 0x00;                                      /* ISO15693 flag with option flag not set*/
buf[6] = 0x24;                                      /* Read multiple blocks command code */
buf[7] = First block number
buf[8] = Number of blocks
buf[9] to buf[16] = Data
```

Write buf[0] to buf[16] to TRF796x via SPI or Parallel mode.

To transmit next 9 bytes:

```
buf[0] = 0x3F;                    /* Continuous write to FIFO 0x1F */
buf[1] to buf[9] = Data;          /* 9 data bytes for transmission */
```

1. Write buf[0] to buf[9] to TRF796x via SPI or Parallel mode.
2. Wait for a End of TX interrupt (use a timer for timeout).
3. If the Option_flag (in the Flags field in the request packet) is set, the VICC waits for the reception of an EOF and upon such reception shall return its response.
   a. Send reset command (0x0F) to FIFO (command byte - 0x8F)
   b. Send transmit next slot (EOF) command (0x14) (command byte – 0x94)

   If the Option_flag is not set, proceed to step 4.
4. Wait for next interrupt (use a timer for timeout). This can be due to any of the following:
   a. End of RX
   b. Collision

   Check the IRQ status register to determine the cause of the interrupt (for more details, refer to Section 2 on interrupts).

   If interrupt is due to End of RX, this means that the response is received in the FIFO without any error/collision. Read the FIFO to obtain the data received from the tag. Check for the Error_flag bit in the Flags field of the response. The VICC reports the success of the operation in this bit.

   If interrupt is due to collision, the user can choose what to do next – try again (repeat from step 1) or ignore.

Pseudo-code (for write multiple blocks command with UID):

The procedure to write multiple blocks with UID is the similar to that of write multiple blocks except that the 8 bytes of the UID is sent along with the request packet. Please note that the UID being a multiple-byte field, has to be sent with its least significant byte first. Please refer to Section 1.5 on Pseudo-code for read multiple blocks command with UID for details.

2 Interrupt Handler Routine

The reader which is a slave device has an IRQ pin to prompt/flag the MCU for attention in cases when the reader detects a response from the PICC/VICC. The interrupt handler routine described below determines how the IRQ should be handled.

The TRF796x IRQ status register (Table 2) is read to determine the cause of the IRQ. The following conditions (Table 1) are checked and appropriate actions taken:

Table 1. Interrupt Conditions

NO.	CONDITION	ACTION
1	Transmission complete	Reset FIFO
2	Collision occurred	1. Read collision position register (TRF796x).(1)(2)(3) 2. Determine the number of valid bytes and bits. 3. Read the valid received bytes and bits in FIFO and write to local buffer. 4. Reset FIFO.

(1) Though registers 0Dh and 0Eh give the collision position, only register 0Eh is used because the anti-collision command in ISO 14443A is maximum only 7 bytes long. Hence 8 bits (0Dh) are enough to determine the position.

(2) The lower nibble of the Collision register (0Eh) has the bit count and that the upper nibble has the byte count. For example, if the Collision Position register holds the value 0x40, it means that the collision happened in the 4th byte on the bit 0.

(3) The anti-collision procedure in the ISO14443A standard is done in such a way, that the reader sends at least 2 bytes (cascade level and length information) in the anti-collision command. The collision position is counted from this reader command on. Therefore to know the number of valid bytes and bits, subtract 0x20 from the Collision Position register.

Table 1. Interrupt Conditions (continued)

NO.	CONDITION	ACTION
3	RX flag set (at End of RX)	1. Read FIFO status register (TRF796x) to determine the number of unread bytes and bits in FIFO. 2. Read the data in FIFO and write to local buffer. 3. Reset FIFO.
4	RX active and 9 bytes in FIFO	1. Read 9 bytes from FIFO. 2. Check if IRQ pin is still high. If yes, go to condition No. 3.
5	CRC error	Set error flag.
6	Byte framing error	Set error flag.
7	No-response time-out	
8	Any other	1. Reset FIFO. 2. Clear interrupt flag.

Table 2. IRQ Status Register

BIT	BIT NAME	FUNCTION	COMMENTS
B7	Irq_tx	IRQ set due to end of TX	Signals the TX is in progress. The flag is set at the start of TX, but the interrupt request is sent when TX is finished.
B6	Irq_srx	IRQ set due to RX start	Signals that RX SOF was received and RX is in progress. The flag is set at the start of RX, but the interrupt request is sent when RX is finished.
B5	Irq_fifo	Signals the FIFO is 1/3 > FIFO > 2/3	Signals FIFO high or low (less than 4 or more than 8).
B4	Irq_err1	CRC error	Reception CRC
B3	Irq_err2	Parity error	
B2	Irq_err3	Byte framing or EOF error	
B1	Irq_col	Collision error	For ISO14443A and ISO15693 single sub-carrier
B0	Irq_noresp	No response interrupt	Signal to MCU that next slot command can be sent.

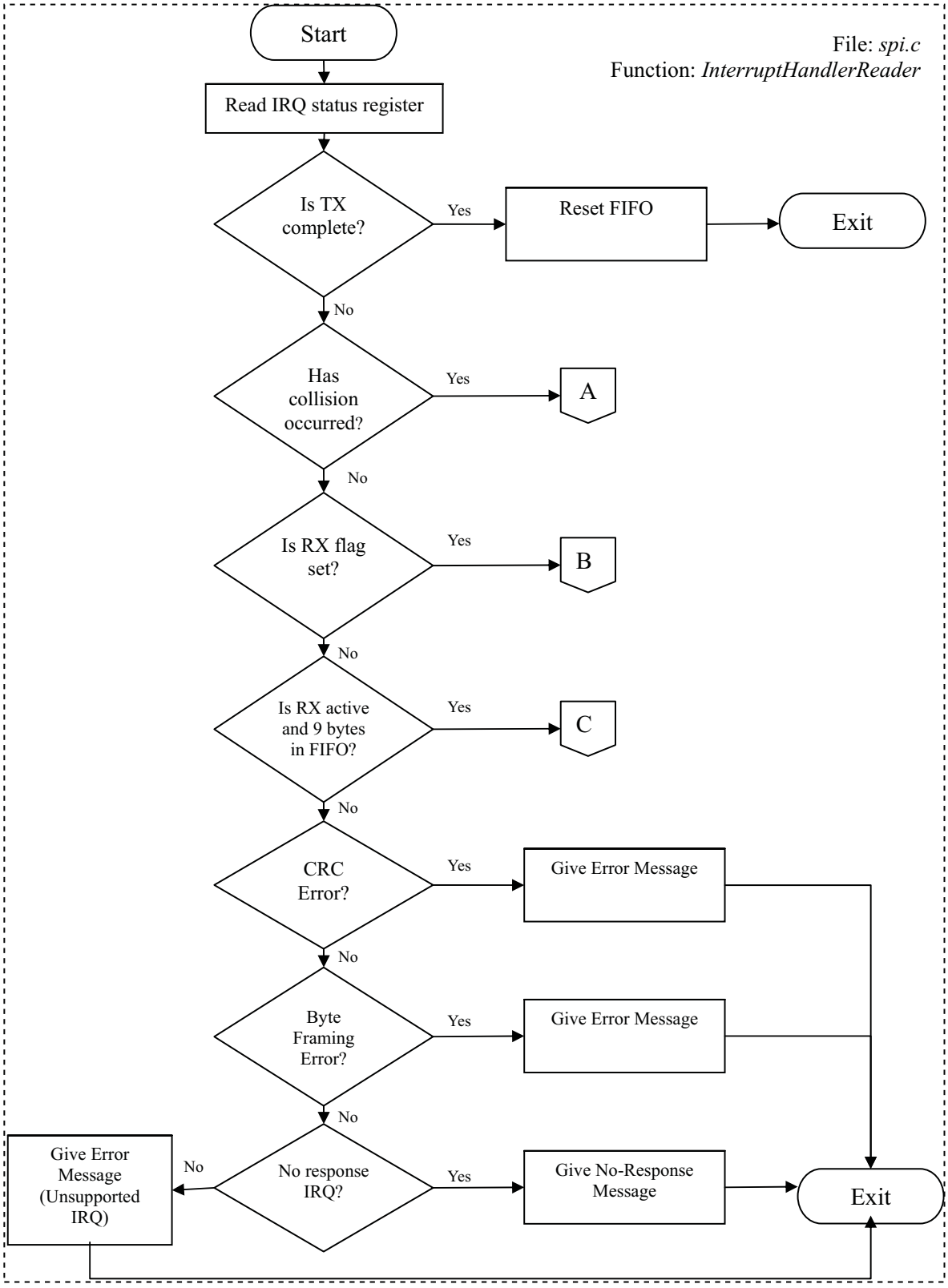

Figure 3. Interrupt Handler Routine (1)

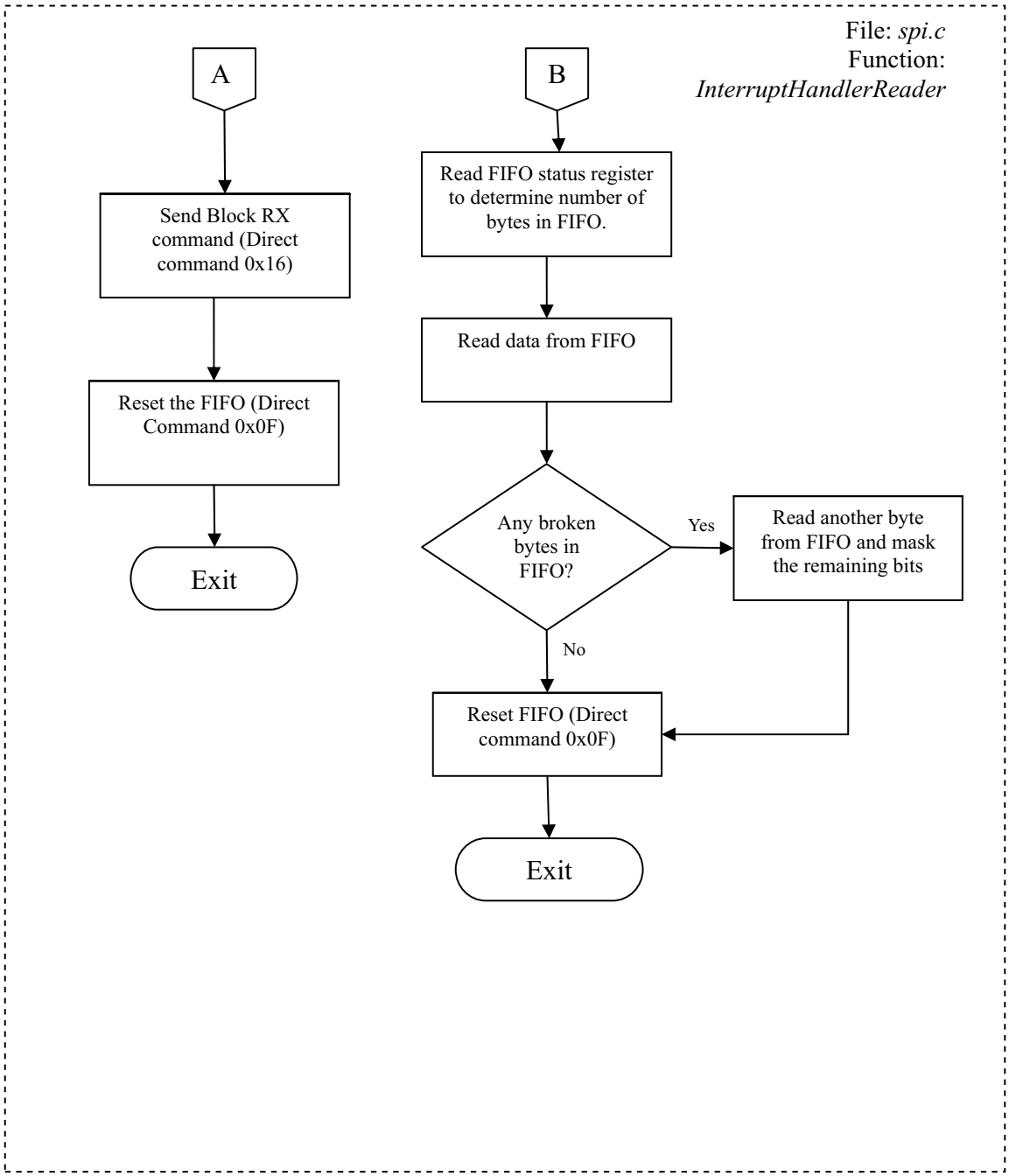

Figure 4. Interrupt Handler Routine (2)

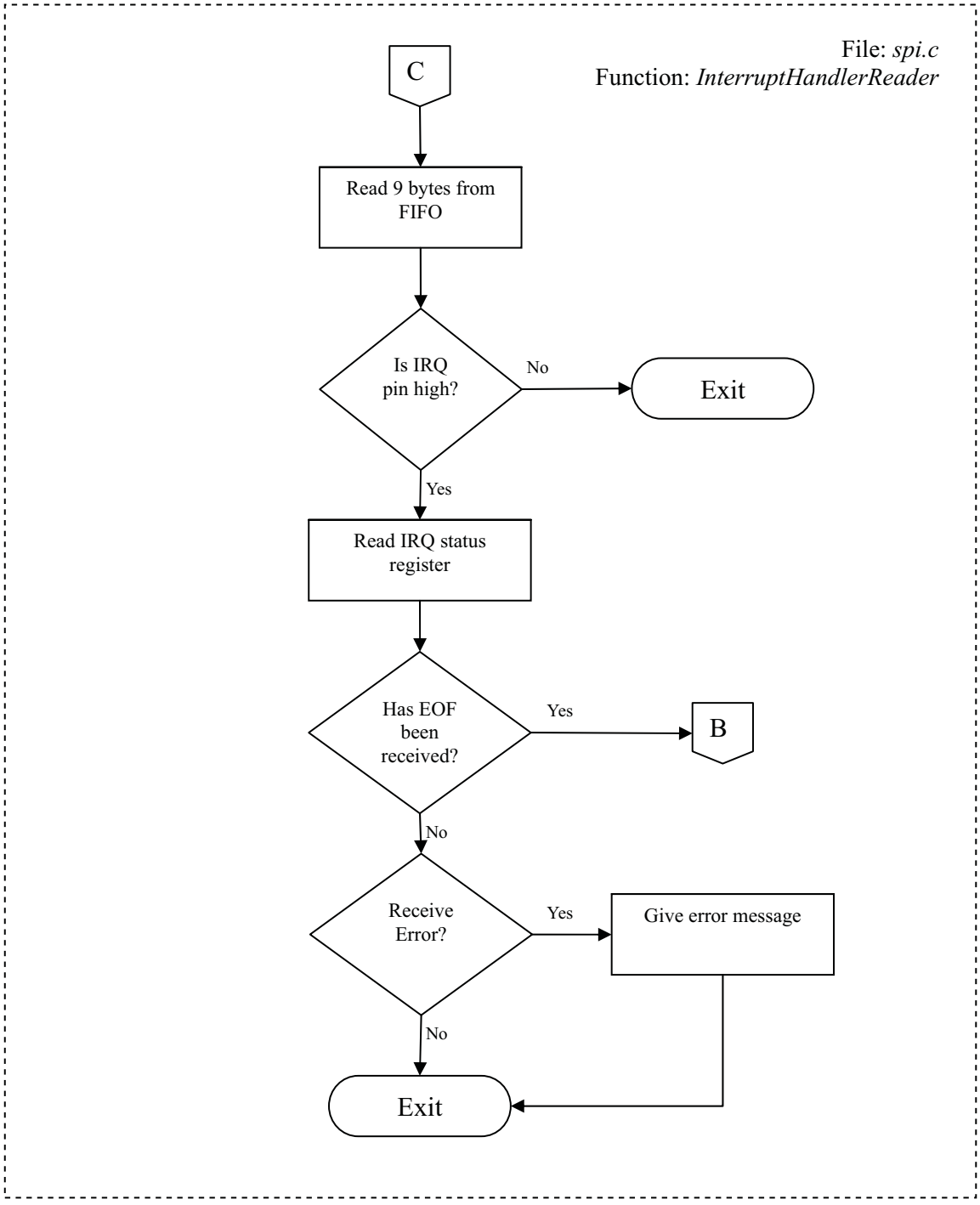

Figure 5. Interrupt Handler Routine (3)

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

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

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information 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.

Reproduction of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DLP® Products	www.dlp.com	Broadband	www.ti.com/broadband
DSP	dsp.ti.com	Digital Control	www.ti.com/digitalcontrol
Clocks and Timers	www.ti.com/clocks	Medical	www.ti.com/medical
Interface	interface.ti.com	Military	www.ti.com/military
Logic	logic.ti.com	Optical Networking	www.ti.com/opticalnetwork
Power Mgmt	power.ti.com	Security	www.ti.com/security
Microcontrollers	microcontroller.ti.com	Telephony	www.ti.com/telephony
RFID	www.ti-rfid.com	Video & Imaging	www.ti.com/video
RF/IF and ZigBee® Solutions	www.ti.com/lprf	Wireless	www.ti.com/wireless

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