TRF7970A RFID Reader Software Example

API Guide

DOCNUM-00.01.00

Contents

Chapter 1

Copyright

Please be aware that an important notice concerning availability, standard warranty, and use in critical applications of Texas Instruments semiconductor products and disclaimers thereto appears at the end of this document.

Texas Instruments
13532 N. Central Expressway
Dallas, TX 75243
www.ti.com

Chapter 2

Introduction

This Example detects ISO15693 NFC/RFID tags and then checks if an RF430FRL15xH transponder has been presented.

If an RF430FRL15xH transponder has been detected, then it is configured for sensor measurements using the Write Single Block NFC command. The example firmware sends configurations specifically for thermistors, but these configurations can be modified to support other custom applications

Once the raw sensor data is received by the TRF7970A, then the MSP430FR4133 uses the IQmathLib in order to calculate the temperature result and display it on the on-board LCD of the MSP-EXP430FR4133 LaunchPad.

This example also can optionally output that data to UART along with information such as the tag UID.

The TRF7970A is an integrated analog front end and data framing system for a 13.56 MHz RFID reader system. Built-in programming options make it suitable for a wide range of applications both in proximity and vicinity RFID systems. The reader is configured by selecting the desired protocol in the control registers. Direct access to all control registers allows fine tuning of various reader parameters as needed.

The TRF7970A is interfaced to a MSP430FR4133 through a SPI (serial) interface using a hardware USCI. The MCU is the master device and initiates all communication with the reader.

The anti-collision procedures (as described in the ISO15693 standard) are implemented in the MCU firmware to help the reader detect and communicate with one VICC among several VICCs.

Chapter 3

Disclaimer

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.

You can obtain information on other Texas Instruments products and application solutions from www.ti.com.

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

Chapter 4

Module Index

4.1 Modules

Here is a list of all modules:

Chapter 5

Module Documentation

5.1 ISO15693 Read/Write API's

Functions

- uint8_t ISO15693_sendSingleSlotInventory (void)
- uint8_t ISO15693_runAnticollision (uint8_t ui8ReqFlags, uint8_t ui8MaskLength, uint8_t ui8Afi)
- uint16_t ISO15693_sendGetSystemInfo (uint8_t ui8ReqFlag)
- uint16_t ISO15693_sendGetSystemInfoExtended (uint8_t ui8ReqFlag)
- uint8_t ISO15693_sendReadSingleBlock (uint8_t ui8ReqFlag, uint8_t ui8BlockNumber)
- uint8_t ISO15693_sendReadMultipleBlocks (uint8_t ui8ReqFlag, uint8_t ui8FirstBlock, uint8_t ui8NumberOfBlocks)
- uint8_t ISO15693_sendReadSingleBlockExtended (uint8_t ui8ReqFlag, uint16_t ui16BlockNumber)
- uint8_t ISO15693_sendWriteSingleBlock (uint8_t ui8ReqFlag, uint8_t ui8BlockNumber, uint8_t ui8BlockSize, uint8_t *pui8BlockData)
- uint8_t * ISO15693_getUid (uint8_t ui8Index)
- uint8_t ISO15693_getTagCount (void)
- void ISO15693_resetTagCount (void)
- uint8_t ISO15693_getSelectedTagIndex (void)
- uint8_t ISO15693_setSelectedTagIndex (uint8_t ui8Index)
- void ISO15693_resetRecursionCount (void)
- void ISO15693_init (void)

Variables

- uint8_t **g_pui8TrfBuffer** [NFC_FIFO_SIZE]

5.1.1 Detailed Description

A ISO15693 Reader/Writer issues commands based on the ISO15693 specifications. The ISO15693 specification is for Vicinity Integrated Circuit Cards (VICCs) and communication with VICCs that are Type V tags occur at bitrates of 6.62 kbps (low data rate) or 26.48 kbps (high data rate).

For more information on ISO15693 Readers and Tags please read the ISO15693 Specifications.

5.1.2 Function Documentation

5.1.2.1 uint8_t ISO15693_getSelectedTagIndex (void)

ISO15693_getSelectedTagIndex - Fetches the ISO15693 UID buffer index.

This function allows for higher layers to fetch the index for the ISO15693 UID array. The index is used to determine which tag UID will be used for all issued addressed ISO15693 commands.

Returns

g_ui8SelectedTagIndex returns the current index for the ISO15693 UID array.

5.1.2.2 uint8_t ISO15693_getTagCount (void)

ISO15693_getTagCount - Fetches the number of ISO15693 tags which have been detected.

This function allows for higher layers to fetch the tag detected count in order to know how many ISO15693 tags have been detected by the anticollsion process.

Returns

g_ui8TagDetectedCount returns the count of ISO15693 tags detected.

5.1.2.3 uint8_t* ISO15693_getUid (uint8_t *ui8Index*)

ISO15693_getUid - Fetches the ISO15693 Tag UID.

Parameters

ui8Index	is the index for the ISO15693 UID array.

This function allows for higher layers to fetch the tag UID of one ISO15693 tag. An index corresponding to the array must be provided.

Returns

g_pui8Iso15693UId returns a pointer to the UID buffer.

5.1.2.4 void ISO15693_init (void)

ISO15693_init - Initialize all Global Variables for the ISO15693 layer.

This function will initialize all the global variables for the ISO15693 layer with the appropriate starting values or empty values/buffers.

Returns

None

5.1.2.5 void ISO15693_resetRecursionCount (void)

ISO15693_resetRecursionCount - Clear the recursion counter for the anticollision process.

This function allows for higher layers to reset the anticollision process recursion counter.

Returns

None

5.1.2.6 void ISO15693_resetTagCount (void)

ISO15693_resetTagCount - Reset number of the ISO15693 tags detected.

This function allows for higher layers to reset the tag detected counter.

Returns

None

5.1.2.7 uint8_t ISO15693_runAnticollision (uint8_t *ui8ReqFlags,* uint8_t *ui8MaskLength,* uint8_t *ui8Afi*)

ISO15693_runAnticollision - Issue an Inventory command for thhe 16 slot anticollision process for ISO15693 tags.

Parameters

ui8ReqFlag	are the request flags for ISO15693 commands.
ui8MaskLength	is the number of significant bits in the mask value.
ui8Afi	is the AFI to be issued with command (if AFI flag is included in ui8ReqFlag).

Function issues a sixteen slot Inventory command for the detection of ISO15693 tags. If a tag is found, the UID is stored in the g_ui8Iso15693UId buffer. The process will run until all ISO15693 detected have responded with their UID's.

The function uses a recursive call for the anticollision process. In order to avoid stack overflows, a recursion counter is used to limit the maximum number of recursions.

The number of UID's which can be stored is set by g_ui8MaximumTagCount and the declaration of the g_ui8Iso15693UId buffer. Once the buffer for UID's is filled then no other UID's responses will be stored.

If UART is enabled, the UID of each ISO15693 tag detected is sent out to a host via UART.

Returns

ui8Status returns STATUS_SUCCESS if the anticollision function resulted in a successful tag detection. Otherwise, returns STATUS_FAIL.

5.1.2.8 uint16_t ISO15693_sendGetSystemInfo (uint8_t *ui8ReqFlag*)

ISO15693_sendGetSystemInfo - Issue the Get System Information command for ISO15693 tags.

Parameters

ui8ReqFlag	are the request flags for ISO15693 commands.

This function issues a Get System Information command for ISO15693 tags. This can be used to determine how many blocks of data can be read from the tag.

If UART is enabled, the contents of the Get System Information response is output via UART.

Returns

ui16NumberOfBlocks returns the number of blocks contained in the ISO15693 tag.

5.1.2.9 uint16_t ISO15693_sendGetSystemInfoExtended (uint8_t *ui8ReqFlag*)

ISO15693_sendGetSystemInfoExtended - Issue the Get System Information command for ISO15693 tags with the protocol extention flag set.

Parameters

ui8ReqFlag	are the request flags for ISO15693 commands.

This function issues a Get System Information command for ISO15693 tags with the Protocol Extension bit set in the request flags. This can be used to determine how many blocks of data can be read from the tag.

If UART is enabled, the contents of the Get System Information response is output via UART.

Returns

ui16NumberOfBlocks returns the number of blocks contained in the ISO15693 tag.

5.1.2.10 uint8_t ISO15693_sendReadMultipleBlocks (uint8_t *ui8ReqFlag,* uint8_t *ui8FirstBlock,* uint8_t *ui8NumberOfBlocks*)

ISO15693_sendReadMultipleBlocks - Issue the Read Multiple Block command for ISO15693 tags.

Parameters

ui8ReqFlag	are the request flags for ISO15693 commands.
ui8FirstBlock	is the starting block number to read data from.
ui8NumberOfBlocks	is the amount of blocks to read data from.

This function issues a Read Multiple Block with the specified request flags, the starting block number, and the number blocks to read data from.

If UART is enabled, the data read from the ISO15693 tag is output via UART.

Returns

ui8Status returns either STATUS_SUCCESS or STATUS_FAIL to indicate if the Read Multiple Block was successful or not.

5.1.2.11 uint8_t ISO15693_sendReadSingleBlock (uint8_t *ui8ReqFlag,* uint8_t *ui8BlockNumber*)

ISO15693_sendReadSingleBlock - Issue the Read Single Block command for ISO15693 tags.

Parameters

ui8ReqFlag	are the request flags for ISO15693 commands.
ui8BlockNumber	is the block number to read data from.

This function issues a Read Single Block with the specified request flags and block number to read data from.

If UART is enabled, the data read from the ISO15693 tag is output via UART.

Returns

ui8Status returns either STATUS_SUCCESS or STATUS_FAIL to indicate if the Read Single Block was successful or not.

5.1.2.12 uint8_t ISO15693_sendReadSingleBlockExtended (uint8_t *ui8ReqFlag,* uint16_t *ui16BlockNumber*)

ISO15693_sendReadSingleBlockExtended - Issue the Read Single Block command for ISO15693 tags with the protocol extention flag set

Parameters

ui8ReqFlag	are the request flags for ISO15693 commands.
ui16BlockNumber	is the block number to read data from.

This function issues a Read Single Block with the block number and the specified request flags, including the Protocol Extension bit, to read data from ISO15693 tags which require the use of extended protocol commands.

If UART is enabled, the data read from the ISO15693 tag is output via UART.

Returns

ui8Status returns either STATUS_SUCCESS or STATUS_FAIL to indicate if the Read Single Block was successful or not.

5.1.2.13 uint8_t ISO15693_sendSingleSlotInventory (void)

ISO15693_sendSingleSlotInventory - Issue a single slot Inventory command for ISO15693 tags.

This function issues a single slot Inventory command for tag detection of ISO15693 tags. If a tag is found, the UID is stored in the g_pui8Iso15693UId buffer.

If UART is enabled, the tag ID is sent out to a host via UART.

Returns

ui8Status returns either STATUS_SUCCESS or STATUS_FAIL to indicate if the Inventory command resulted in a successful tag detection or not.

5.1.2.14 uint8_t ISO15693_sendWriteSingleBlock (uint8_t *ui8ReqFlag,* uint8_t *ui8BlockNumber,* uint8_t *ui8BlockSize,* uint8_t * *pui8BlockData*)

ISO15693_sendWriteSingleBlock - Issue the Write Single Block command for ISO15693 tags.

Parameters

ui8ReqFlag	are the request flags for ISO15693 commands.
ui8BlockNumber	is the block number to write data to.
ui8BlockSize	is the tag block size.
pui8BlockData	is the data to be written.

Function issues an addressed Write Single Block with the specified request flags as well as the address flag. The write single block command writes the provided data to the addressed ISO15693 tag.

The function will always add the address flag to the command as a good practice to avoid overwriting other ISO15693 tags in the vicinity.

This function natively supports writing to tags with more than 4 bytes of data per block through the ui8BlockSize variable.

Returns

ui8Status returns either STATUS_SUCCESS or STATUS_FAIL to indicate if the Write Single Block was successful or not.

5.1.2.15 uint8_t ISO15693_setSelectedTagIndex (uint8_t *ui8Index*)

ISO15693_setSelectedTagIndex - Set the ISO15693 UID buffer index.

Parameters

ui8Index	is the new index for the ISO15963 UID array.

This function allows for higher layers to set the index for the ISO15693 UID array. The index is used to determine which tag UID will be used for all issued addressed ISO15693 commands.

Returns

Either STATUS_SUCCESS if the new index was set or STATUS_FAIL if an out of range index was provided.

5.2 LCD Application Specific API's

Functions

- void LCD_displayTINFCMsg (void)
- void LCD_printRSSI (void)
- void LCD_displayTemperature (float fTemperature)
- void LCD_convertFloatToAscii (float fFloatInput, uint8_t *pui8OutputBuffer)
- void LCD_displayUID (uint8_t *pui8ISO15693UID)

5.2.1 Detailed Description

Application specific API's to display messages to the LCD of a TI MSP-EXP430FR4133 LaunchPad Module.

5.2.2 Function Documentation

5.2.2.1 void LCD_convertFloatToAscii (float *fFloatInput*, uint8_t * *pui8OutputBuffer*)

LCD_convertFloatToAscii - Function to convert a floating point input value into ASCII characters which are then stored in an unsigned 8 bit integar array.

Parameters

fFloatInput	is the floating point value to be converted.
pui8OutputBuffer	is a user provided buffer to store the converted value into.

Function converts a provided floating point input to ASCII characters and stores each converted digit into a user provided buffer.

Function is limited to maximum 4 digit conversion with one decimal point precision as it is designed to handle temperatures in the range of -40 to 125 degree Celcius.

Returns

None

5.2.2.2 void LCD_displayTemperature (float *fTemperature*)

LCD_displayTemperature - Function to display a temperature value to an LCD.

Parameters

fTemperature	is a floating point temperature to display.

Function displays the provided floating point temperature from an RF430FRL152H Sensor Patch to the LCD of the MSP-EXP430FR4133 LaunchPad.

Returns

None

5.2.2.3 void LCD_displayTINFCMsg (void)

LCD_displayTINFCMsg - Display "TI NFC" message to LCD

Function sends commands to display a simple message ("TI NFC") on the LCD.

Returns

None

5.2.2.4 void LCD_displayUID (uint8_t * *pui8ISO15693UID*)

LCD_displayUID - Function to read RSSI Level and display it on an LCD.

Parameters

ui8SelectedTag	is a pointer to the array which contains the ISO15693 tag UID to be displayed.

Function displays the UID to the LCD of the FR4133 LaunchPad.

Returns

None

5.2.2.5 void LCD_printRSSI (void)

LCD_printRSSI - Function to read RSSI Level and display it on an LCD.

Function reads the RSSI level from the TRF7970A, masks the relevant bits, and then display a number of bars to indicate the signal strength on the LCD of the MSP-EXP430FR4133 LaunchPad.

Returns

None

5.3 General MSP430FR4133 API's

Functions

- void MCU_setCounter (uint16_t ui16mSecTimeout)
- void MCU_delayMillisecond (uint32_t n_ms)
- void MCU_initClock (uint32_t ui32Freq)
- void MCU_setHeartbeat (void)
- __interrupt void **TimerA1Handler** (void)

5.3.1 Detailed Description

This section contains descriptions for general MSP430FR4133 functions including but not limited to clock configuration and timers for the MSP430FR4133.

5.3.2 Function Documentation

5.3.2.1 void MCU_delayMillisecond (uint32_t *n_ms*)

MCU_delayMillisecond - Delay for inputted number of millisecond

Parameters

n_ms	is the number of milliseconds to delay by

Blocking function to delay is approximately one millisecond, looping until the inputted number of milliseconds have gone by. DELAY_1ms must be calibrated based on the clock speed of the MCU.

Returns

None.

5.3.2.2 void MCU_initClock (uint32_t *ui32Freq*)

MCU_initClock - Calibrate the Oscillator

Parameters

ui32Freq	is the clock frequency (in kHz) to set in the MSP430FR4133

Function to calibrate the DCO of the MSP430FR4133.

Returns

None.

5.3.2.3 void MCU_setCounter (uint16_t *ui16mSecTimeout*)

MCU_setCounter - Set up the counter for Timer A0

Set up the counter for Timer A0 and enable the interrupt

Returns

None.

5.3.2.4 void MCU_setHeartbeat (void)

MCU_setHeartbeat - Handle the timer for the LED Heartbeat

Function starts the 500 millisecond timer which is interrupt driven in order to blink the heartbeat on the MSP-EXP430FR4133 LaunchPad LCD.

Returns

None.

5.4 NFC Application Specific API's

Functions

- void NFC_runAppReadFRLTag (void)
- uint8_t NFC_runRF430FRLStateMachine (uint8_t ui8TagIndex)
- uint8_t NFC_findISO15693Tag (void)
- uint8_t NFC_searchRF430FRL (uint8_t ui8TagIndex)
- uint8_t NFC_pollRF430FRLDataReady (void)
- uint8_t NFC_writeRF430FRLSensorConfig (void)
- float NFC_calculateTemperature (uint16_t ui16ThermValue, uint16_t ui16RefResValue)
- void NFC_updateRSSI (void)

5.4.1 Detailed Description

Application specific API's to handle NFC communication between the TRF7970A and the RF430FRL152H and calculate temperature measurements from received sensor data.

5.4.2 Function Documentation

5.4.2.1 float NFC_calculateTemperature (uint16_t *ui16ThermValue,* uint16_t *ui16RefResValue*)

NFC_calculateTemperature - Function to calculate a temperature based on thermistor and reference resistor inputs.

Parameters

ui16ThermValue	is the raw ADC value after reading the FRL152H Sensor Patch thermistor
ui16RefResValue	is the raw ADC value after reading the FRL152H Sensor Patch reference resistor

This function calculates the temperature result from the sensors on the RF430FRL152H Sensor Patch by using the MSP430 IQmathLib Fixed Point mathematics library.

The IQmathLib offers various grades of decimal precision versus maximum possible value stored. In order to support the temperature conversion while maintaining a high level of precision, the IQ10 setting is used.

The process of calculating the temperature from the thermistor value is specific to the thermistor used on the RF430-TMPSNS-EVM board (ERT-J1VS104FA).

Returns

fTemperatureResult is the floating point result of the temperature conversion.

5.4.2.2 uint8_t NFC_findISO15693Tag (void)

NFC_findISO15693Tag - Application to search for ISO15693 compliant tags.

This function configures TRF79xxA for ISO1593, waits for the guard time to allow VICC to power up, and then searches for ISO15693 tags.

Returns

ui8Status returns either STATUS_SUCCESS or STATUS_FAIL to indicate if an ISO15693 tag was found or not.

5.4.2.3 uint8_t NFC_pollRF430FRLDataReady (void)

NFC_pollRF430FRLDataReady - Checks for sensor data to be ready on the RF430FRL15xH.

This function reads Block 0 of the RF430FRL15xH repeatedly and checks the result to determine when the ADC conversion is completed and sensor data is available to be read.

Returns

ui8Status returns either STATUS_SUCCESS or STATUS_FAIL to indicate if the sensor data is ready or if an error occurred.

5.4.2.4 void NFC_runAppReadFRLTag (void)

NFC_runAppReadFRLTag - Application designed to detect ISO15693 tags and read data from them.

This function detects ISO15693 tags, determines how many tags (if any) were detected, and then for each of these tags, run the application state machine that attempts to find and read data from an RF430FRL15xH tag.

Returns

None.

5.4.2.5 uint8_t NFC_runRF430FRLStateMachine (uint8_t *ui8TagIndex*)

NFC_runRF430FRLStateMachine - Simple state machine to pull data from an RF430FRL15xH tag.

Parameters

ui8TagIndex	is the tag index marker for the UID buffer.

This function runs through the process of searching for RF430FRL15xH tag, configuring the sensors, and pulling the sensor data from the tag.

It also has states that call the API's needed to convert and display the data from a thermistor onto the LCD of the MSP-EXP430FR4133 LaunchPad

The ui8TagIndex variable allows for the commands to be issued with addressed commands directed at a single ISO15693 tag.

Returns

ui8Status to indicate whether or not an RF430FRL15xH tag was found and had it's sensor data read.

5.4.2.6 uint8_t NFC_searchRF430FRL (uint8_t *ui8TagIndex*)

NFC_searchRF430FRL - Issues ISO15693 Commands which can identify if an ISO15693 tag is an RF430FRL15xH tag.

This function first checks the tag UID to determine if the ISO15693 is made by Texas Instruments. If so, then the Get System Information command is issued to get information about the memory size of the tag.

The only Texas Instruments ISO15693 transponders that have a memory size of of 0xF2 are from the RF430FRL15xH family.

Returns

ui8Status returns either STATUS_SUCCESS or STATUS_FAIL to indicate if the sequence of commands were successful or not.

5.4.2.7 void NFC_updateRSSI (void)

NFC_updateRSSI - Updates the RSSI reading.

This function sends an ISO15693 Read Single Block command to get a new RSSI reading in and then updates the LCD display of the MSP-EXP430FR4133 LaunchPad module.

This is an application specific function designed for ISO15693, but can be modified to support other tag technologies by changing the RF command issued to a command which will trigger a reply from the desired tag technology.

Returns

None.

5.4.2.8 uint8_t NFC_writeRF430FRLSensorConfig (void)

NFC_writeRF430FRLSensorConfig - Configures the RF430FRL152H to sample a thermistor.

This function is used to write the data blocks of the RF430FRL152H with the necessary configuration parameters to begin sampling of the thermistor and reference resistor for temperature measurements.

Returns

ui8Status returns either STATUS_SUCCESS or STATUS_FAIL to indicate if the sequence of commands were successful or not.

5.5 TRF7970A Driver API's

Functions

- void TRF79xxA_sendDirectCommand (uint8_t ui8Value)
- void TRF79xxA_disableSlotCounter (void)
- void TRF79xxA_enableSlotCounter (void)
- void TRF79xxA_resetFIFO (void)
- void TRF79xxA_initialSettings (void)
- void TRF79xxA_processIRQ (uint8_t *pui8IrqStatus)
- void TRF79xxA_writeRaw (uint8_t *pui8Payload, uint8_t ui8Length)
- void TRF79xxA_readContinuous (uint8_t *pui8Payload, uint8_t ui8Length)
- uint8_t TRF79xxA_readIrqStatus (void)
- uint8_t TRF79xxA_readRegister (uint8_t ui8TrfRegister)
- void TRF79xxA_reset (void)
- void TRF79xxA_resetIrqStatus (void)
- void TRF79xxA_turnRfOff (void)
- void TRF79xxA_turnRfOn (void)
- void TRF79xxA_writeContinuous (uint8_t *pui8Payload, uint8_t ui8Length)
- void TRF79xxA_writeRegister (uint8_t ui8TrfRegister, uint8_t ui8Value)
- void TRF79xxA_setupInitiator (uint8_t ui8IsoControl)
- void TRF79xxA_waitIRQ (uint8_t ui8TxTimeout, uint8_t ui8RxTimeout)
- void TRF79xxA_waitIRQ_txOnly (uint8_t ui8TxTimeout)
- void TRF79xxA_waitIRQ_rxOnly (uint8_t ui8RxTimeout)
- tTrfStatus TRF79xxA_getTrfStatus (void)
- void TRF79xxA_setTrfStatus (tTrfStatus sTrfStatus)
- void TRF79xxA_setTrfPowerSetting (tTrfPowerOptions sTrfPowerSetting)
- uint8_t TRF79xxA_getCollisionPosition (void)
- void TRF79xxA_setCollisionPosition (uint8_t ui8ColPos)
- uint8_t TRF79xxA_getRxBytesReceived (void)
- uint8_t * TRF79xxA_getTrfBuffer (void)
- uint8_t TRF79xxA_getIsoControlValue (void)
- bool TRF79xxA_checkExternalRfField (void)
- __interrupt void TRF79xxA_timerHandler (void)
- __interrupt void TRF79xxA_irqHandler (void)

5.5.1 Detailed Description

This section describes all the functions contained within the TRF79xxA driver.

5.5.2 Function Documentation

5.5.2.1 bool TRF79xxA_checkExternalRfField (void)

Trf79xxA_checkExternalRfField - Checks if an external RF field is present.

This function performs RF collision avoidance as outlined in TI application notes (sloa192, sloa227) in order to determine if an external RF field is present.

Returns

bExtFieldOn retuns the external RF field status

5.5.2.2 void TRF79xxA_disableSlotCounter (void)

TRF79xxA_disableSlotCounter - Disables interrupts from 15693 slot counter function.

This function will configure the TRF79xxA to disable the firing of the IRQ interrupt when an ISO15693 slot marker hits the No Response timeout threshold.

Returns

None.

5.5.2.3 void TRF79xxA_enableSlotCounter (void)

TRF79xxA_enableSlotCounter - Enables interrupts from 15693 slot counter function.

This function will configure the TRF79xxA to enable the firing of the IRQ interrupt when an ISO15693 slot marker hits the No Response timeout threshold.

Returns

None.

5.5.2.4 uint8_t TRF79xxA_getCollisionPosition (void)

Trf79xxA_getCollisionPosition - Return the current Collision Position value

This function will return the current Collision Position value. This is only used for the ISO14443 Type A anti-collision process.

Returns

g_ui8CollisionPosition returns the current Collision Position value

5.5.2.5 uint8_t TRF79xxA_getIsoControlValue (void)

Trf79xxA_getIsoControlValue - Fetch the latest Iso Control Register value

This function returns the current TRF79xxA ISO Control Register setting .

The ISO Control Register value is updated whenever a read or write to the ISO Control Register occurs.

Returns

g_ui8IsoControlValue returns the current ISO Control Register value

5.5.2.6 uint8_t TRF79xxA_getRxBytesReceived (void)

Trf79xxA_getRxBytesReceived - Returns the Number of RX Bytes received by the TRF79xxA FIFO

This function returns the number of bytes received by the TRF79xxA during the last packet reception.

This is used to check for how much data has been received and to ensure packets with expected lengths were fully received.

Returns

g_ui8FifoRxLength returns the current FIFO RX Length

5.5.2.7 uint8_t* TRF79xxA_getTrfBuffer (void)

TRF79xxA_getTrfBuffer - Returns a point to the start of the TRF Buffer.

This function will return a pointer for the start address of the TRF79xxA Data Buffer.

This is used to access the data received from successful RF commands in files which do not have naturally have access to the g_pui8TrfBuffer.

Returns

&g_pui8TrfBuffer[0] returns the start address of g_pui8TrfBuffer.

5.5.2.8 tTrfStatus TRF79xxA_getTrfStatus (void)

Trf79xxA_getTrfStatus - Returns current TRF79xxA driver status

Returns

g_sTrfStatus returns the current TRF79xxA drive status

5.5.2.9 void TRF79xxA_initialSettings (void)

Trf79xxA_initialSettings - Initialize TRF79xxA

This function configures the TRF79xxA upon power up. Steps include:

- Setup SPI communication
- Send Soft Init and Idle Direct Commands
- Reset the FIFO
- Configure TRF79xxA modulator and regulator registers
- TRF7970A only: Write to regiser 0x18 per errata

Returns

None.

5.5.2.10 __interrupt void TRF79xxA_irqHandler (void)

TRF79xxA_irqHandler- Interrupt handler for IRQ interrupts

Handles receiving IRQ's, getting IRQ status, and maintaining timers/global variables

Returns

None.

5.5.2.11 void TRF79xxA_processIRQ (uint8_t * *pui8IrqStatus*)

Trf79xxA_processIRQ - Services TRF79xxA IRQ interrupts

Parameters

pui8IrqStatus	is the received IRQ Status flags

The Interrupt Service Routine determines how the IRQ should be handled. The TRF79xxA IRQ status register is read to determine the cause of the IRQ. Conditions are checked and appropriate actions taken.

Returns

None.

5.5.2.12 void TRF79xxA_readContinuous (uint8_t * *pui8Payload*, uint8_t *ui8Length*)

Trf79xxA_readContinuous - Read multiple TRF79xxA registers

Parameters

pui8Payload	is the address of the first register as well as the pointer for buffer where the results will be
ui8Length	is the number of registers to read

This function reads a specified number of TRF79xxA registers from a specified address.

Returns

None.

5.5.2.13 uint8_t TRF79xxA_readIrqStatus (void)

Trf79xxA_readIrqStatus - Read out the IRQ Status Register

This function reads the IRQ Status register of the TRF79xxA and store the result into the location pointed to by the input.

Returns

pui8Value returns the value of the IRQ Status Register

5.5.2.14 uint8_t TRF79xxA_readRegister (uint8_t *ui8TrfRegister*)

Trf79xxA_readRegister - Read out a single TRF79xxA register

This function reads a specific TRF79xxA register.

Returns

pui8Value returns the value of the TRF79xxA Register

5.5.2.15 void TRF79xxA_reset (void)

Trf79xxA_reset - Resets TRF79xxA

This function will reset the TRF79xxA through the Software Init direct command followed by reinitializing basic settings and clearing affected global variables.

Returns

None.

5.5.2.16 void TRF79xxA_resetFIFO (void)

TRF79xxA_resetFIFO - Resets TRF79xxA FIFO

This function used to reset the TRF79xxA FIFO.

Returns

None.

5.5.2.17 void TRF79xxA_resetIrqStatus (void)

Trf79xxA_resetIrqStatus - Resets the IRQ Status Register of the TRF79xxA

This function resets/clears the TRF79xxA IRQ Status Register

Returns

None.

5.5.2.18 void TRF79xxA_sendDirectCommand (uint8_t *ui8Value*)

TRF79xxA_sendDirectCommand - Send a Direct Command to TRF79xxA.

Parameters

ui8Value	is the direct command to be issued

This function is used to transmit a Direct Command to TRF79xxA.

Returns

None.

5.5.2.19 void TRF79xxA_setCollisionPosition (uint8_t *ui8ColPos*)

Trf79xxA_setCollisionPosition - Set the Collision Position variable

Parameters

ui8ColPos	is the new Collision Position value

This function sets the Collision Position variable. This is only used for the ISO14443 Type A anti-collision process.

Returns

None.

5.5.2.20 void TRF79xxA_setTrfPowerSetting (tTrfPowerOptions *sTrfPowerSetting*)

Trf79xxA_setTrfPowerSetting - Set the TRF79xxA power setting

Parameters

sTrfPowerSetting	is the new TRF79xxA Power Setting

This function allows for configuration of the TRF79xxA power setting.

Options are:

- TRF79xxA_3V_FULL_POWER: 3V TRF79xxA input w/ full power RF output
- TRF79xxA_5V_FULL_POWER: 5V TRF79xxA input w/ full power RF output
- TRF79xxA_3V_HALF_POWER: 3V TRF79xxA input w/ half power RF output
- TRF79xxA_5V_HALF_POWER: 5V TRF79xxA input w/ half power RF output

Returns

None.

5.5.2.21 void TRF79xxA_setTrfStatus (tTrfStatus *sTrfStatus*)

Trf79xxA_setTrfStatus - Set the TRF79xxA driver status

Parameters

sTrfStatus	is the new TRF79xxA driver status

This function sets the TRF79xxA driver status manually. This can be used to modify the TRF driver status without an IRQ event. Use with caution.

Returns

None.

5.5.2.22 void TRF79xxA_setupInitiator (uint8_t *ui8IsoControl*)

Trf79xxA_setupInitiator - Write the initial settings for a set of TRF79xxA registers based on which protocol is to be enabled.

Parameters

ui8IsoControl	is the value to write to the ISO Control register of the TRF79xxA

This function is used to configure a series of TRF79xxA registers based on which RFID technology will be enabled in the ISO control register.

This function will only enables one RFID technology at a time.

Returns

None.

5.5.2.23 __interrupt void TRF79xxA_timerHandler (void)

TRF79xxA_timerHandler - Handler for assigned MCU Timer Interrupt of TX/RX Timeout functions.

This function processes the Timer Interrupt for the TX/RX timeout functions. If a timeout occurs, the IRQ Status register is read out to determine the current TRF79xxA state.

Returns

None.

5.5.2.24 void TRF79xxA_turnRfOff (void)

Trf79xxA_turnRfOff - Turn off the transmission of the TRF79xxA RF Field

This function stops the TRF79xxA transmitting an RF field

Returns

None.

5.5.2.25 void TRF79xxA_turnRfOn (void)

Trf79xxA_turnRfOn - Turns on the transmission of the TRF79xxA RF Field

This function enables the TRF79xxA transmit an RF field

Returns

None.

5.5.2.26 void TRF79xxA_waitIRQ (uint8_t *ui8TxTimeout,* uint8_t *ui8RxTimeout*)

Trf79xxA_waitIRQ - Timeout sequence for both TX and RX

Parameters

ui8TxTimeout	is the TX timeout in milliseconds
ui8RxTimeout	is the RX timeout in milliseconds

This function ensures data has been transmitted correctly and then determines if any data has been received prior to the RX timeout.

When the RX timeout occurs before data is received, then mark the TRF79xxA status as a No Response Received status.

Returns

None.

5.5.2.27 void TRF79xxA_waitIRQ_rxOnly (uint8_t *ui8RxTimeout*)

Trf79xxA_waitIRQ_rxOnly - Timeout sequence for just RX

Parameters

ui8RxTimeout	is the RX timeout in milliseconds

This function determines if any data has been received prior to the RX timeout only.

When the RX timeout occurs before data is received, then mark the TRF79xxA status as a No Response Received status.

Returns

None.

5.5.2.28 void TRF79xxA_waitIRQ_txOnly (uint8_t *ui8TxTimeout*)

Trf79xxA_waitIRQ_txOnly - Timeout sequence for just TX

Parameters

ui8TxTimeout	is the TX timeout in milliseconds

This function ensures data has been transmitted correctly only.

Returns

None.

5.5.2.29 void TRF79xxA_writeContinuous (uint8_t * *pui8Payload,* uint8_t *ui8Length*)

Trf79xxA_writeContinuous - Write to consecutive TRF79xxA registers.

Parameters

pui8Payload	is the address of the first register followed by the contents to write for each register
ui8Length	is the number of registers to write + 1 Minimum value of ui8Length allowed = 2 (a write to 1 register)

This function writes data to a specific number of TRF79xxA registers from a specific address.

Returns

None.

5.5.2.30 void TRF79xxA_writeRaw (uint8_t * *pui8Payload,* uint8_t *ui8Length*)

TRF79xxA_writeRaw - Write data to TRF79xxA

Parameters

pui8Payload	is the buffer with data packet contents

Parameters

ui8Length	is the size of the data packet

This function writes provided data directly to the TRF79xxA.

Returns

None.

5.5.2.31 void TRF79xxA_writeRegister (uint8_t *ui8TrfRegister,* uint8_t *ui8Value*)

Trf79xxA_writeRegister - Write single to a TRF79xxA Register

Parameters

ui8TrfRegister	is the register address for the write
ui8Value	is the value to write to the specified register

This function writes a new value into a single TRF79xxA register.

Returns

None.