TI Designs

NFC Configuration and Logging Interface

TI Designs

TI Designs provide the foundation that you need including methodology, testing and design files to quickly evaluate and customize the system. TI Designs help *you* accelerate your time to market.

Design Resources

TIDA-00230	Design Folder
MSP430FR5969	Product Folder
RF430CL330H	Product Folder
TPD1E10B06DPYR	Product Folder
TCA9517A	Product Folder

TI E2E™ Community

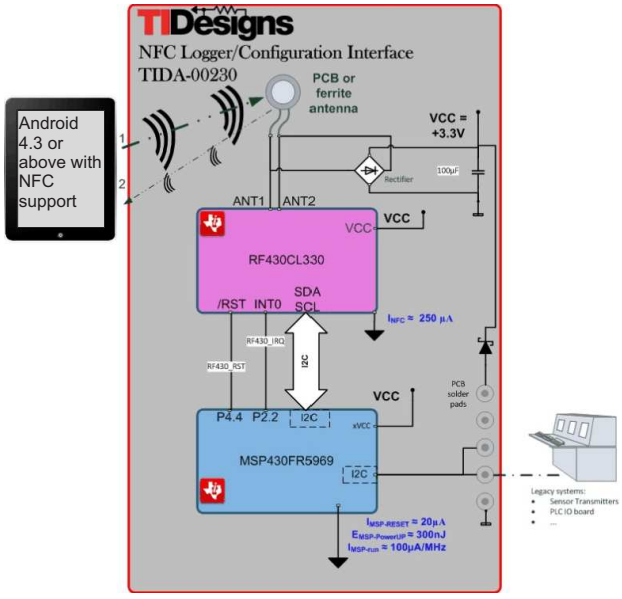
ASK Our E2E Experts
WEBENCH® Calculator Tools

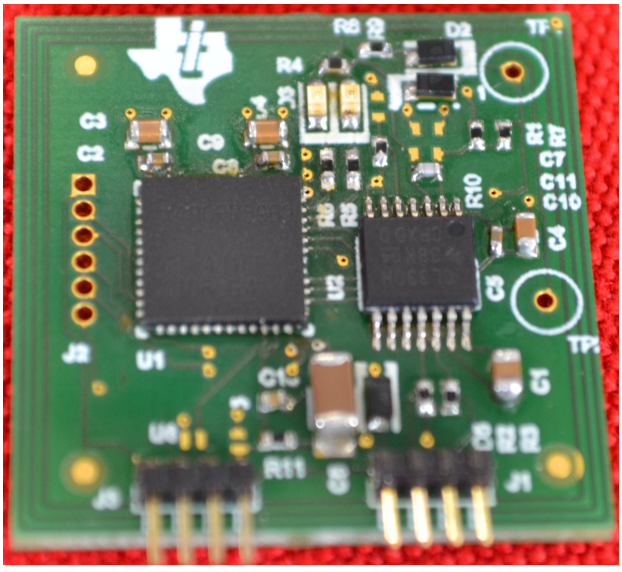
Design Features

- NFC Tag Type 4B
- I2C Interface (Default) or SPI
- 64-kB FRAM (MSP430FR5969)
- Up to 8-Mbps Throughput to or from FRAM over Serial
- 800-kbps Throughput to and from FRAM over NFC
- Power ORing Between NFC and Main System
- Can be Powered by NFC when Main System is Down
- PCB Selectable Option for PCB Antenna or Ferrite
- System Firmware Update over NFC
- System Configuration Settings over NFC
- System Logs Reading over NFC

Featured Applications

- Factory Automation and Process Control
- Building Automation
- Sensors and Field Transmitters
- Portable Instrumentation





An IMPORTANT NOTICE at the end of this TI reference design addresses authorized use, intellectual property matters and other important disclaimers and information.

1 NFC Description

The overall scope of this design is to provide a contactless service interface. Service interface allows reading logs from the system and configuring system (calibration data, firmware upgrades, and so on).

For additional benefits, the contactless is also available when the system is not powered. This benefit is made possible by isolating the MCU and NFC supply from the power of the rest of the system.

1.1 Introduction to NFC

This section provides a quick overview of the NFC interface towards facilitating a grounds-up NFC logger and configuration sub-system project. This section should under no conditions be considered a reference, and only the reference documents should be used once the initial phase of the project has passed.

NFC is a bidirectional, short-range wireless technology communicating over a few centimeters over the ISB band (13.56 MHz) with data rates from 106 to 848 kbps based on the RFID contactless standards.

1.1.1 NFC Versus RFID

NFC is an industry standard that builds on top of RFID international standards (ISOs) and ensures interoperability. The NFC standard is defined by the NFC forum and is based on the existing 13.56-MHz RFID standards (ISO 14443 A/B, JIS-X 6319-4 and ISO15693).

NFC forum specifications for extended functionality and interoperability are standardized by ECMA, ETSI, and ISO/IEC groups NFCIP-1 Standard (ECMA-340, ISO/IEC 18092 , ETSI TS 102 190) and NFCIP-2 Standard (ECMA-352, ISO/IEC 21481, ETSI TS 102 312). The list of specifications is provided in Section 10.1.

1.1.2 NFC Data Transfers Modes

NFC extends the RFID data transfer possibilities by offering three different options:

- *Peer-to-Peer mode* provides a communication between two devices where both devices are able to communicate when required. This mode allows content to be easily shared between two devices.
- *Reader/Writer mode* enables a device to read data from and write data to RFID and contactless smartcards or tags. The NFC forum has mandated four types of tags to be operable in reader/writer mode with NFC devices. The NFC forum specifications define how NDEF messages are to be read from and written to NFC tags.
  - **NFC Forum Type 1 Tag Operation Specification**
    The Type 1 tag is based on ISO/IEC 14443A. Tags are read and re-write capable; users can configure the tag to become read only. The memory available is 96 bytes and expandable to 2kb.
  - **NFC Forum Type 2 Tag Operation Specification**
    The Type 2 tag is based on ISO/IEC 14443A. Tags are read and re-write capable; users can configure the tag to become read only. The memory available is 48 bytes and expandable to 2kb.
  - **NFC Forum Type 3 Tag Operation Specification**
    The Type 3 tag is based on the Japanese Industrial Standard (JIS) X 6319-4, also known as FeliCa. Tags are pre-configured at the manufacture to be either read and re-writable or read only. The memory available is variable, but the theoretical memory limit is 1 MB per service.
  - **NFC Forum Type 4 Tag Operation Specification 2.0**
    The Type 4 tag is fully compatible with the ISO/IEC 14443 standard series. Tags are pre-configured at the manufacture to be either read and re-writable or read only. The memory available is variable up to 32KB per service. The communication interface is either Type-A or Type-B compliant.

NOTE: Given the broader reach of Type 4 tags, support for Type 4 tags was selected for this project.

- *Card emulation mode* enables a device to behave like a contactless smart card. An NFC device may have the ability to emulate more than one card.

NOTE: At the time of writing, the details of the iPhone 6 seem to indicate support for card emulation in the iOS but does not allow peer-to-peer or reader/writer mode.

1.1.3 NDEF Message

1.1.3.1 NDEF Message Overview

The International Standard ISO/IEC 18092, Near Field Communication—Interface and Protocol (NFCIP-1) defines an interface and protocol for simple wireless interconnection of closely coupled devices operating at 13.56 MHz.

NDEF is strictly a message format. The format is a lightweight, binary message encapsulation format to exchange information between an NFC forum device and another NFC forum device or an NFC forum tag. The following are some of the key specifications of NDEF:

- The application payloads may be of arbitrary type and size. All the payload will be encapsulated in one NDEF message. The type identifiers may be URIs, MIME types, or NFC-specific types.
- The length of the payload is an unsigned integer indicating the number of octets in the payload. NDEF has an identifier field for optional payloads that helps associate and cross reference multiple payloads.
- NDEF payloads may include NDEF messages or set of data chunks whose length is unknown at the time data is generated.

1.1.3.2 NDEF Message Objectives

- Provide a data structure format to exchange application data in an inter-operable way for an NFC forum device and NFC forum tag.
- Provide rules to construct a valid NDEF message as an ordered and unbroken collection of NDEF records and define the mechanism for specifying the types of application data encapsulated in NDEF.
- Provide a data format that is independent of data exchange mechanisms. NDEF assumes a reliable underlying protocol for the transfer and exchange of data and only specifies the message format.

1.1.3.3 NDEF Message Goals

The design goal of NDEF is to provide an efficient and simple message format that can accommodate the following:

- Encapsulating arbitrary documents and entities, including encrypted data, XML documents, XML fragments, image data like GIF and JPEG files, and so on.
- Encapsulating documents and entities initially of unknown size. This capability can be used to encapsulate dynamically generated content or very large entities as a series of chunks.
- Aggregating multiple documents and entities that are logically associated in some manner into a single message. For example, NDEF can be used to encapsulate an NFC-specific message and a set of attachments of standardized types referenced from that NFC-specific message.
- Accommodating compact encapsulation of small payloads without introducing unnecessary complexity to parsers.

NOTE: To achieve efficiency and simplicity, the mechanisms provided by this specification have been deliberately limited to serve these purposes. NDEF has not been designed as a general message description or document format such as MIME or XML. Instead, NFC applications can take advantage of such formats by encapsulating them in NDEF messages.

1.1.3.4 Anti-Goals

The following list identifies items outside the scope of NDEF:

- NDEF does not assume the types of payloads that are carried within NDEF messages or about the message exchange patterns implied by such messages.
- NDEF does not in any way introduce the notion of a connection or a logical circuit (virtual or otherwise).
- NDEF does not attempt to deal with head-of-line blocking problems that can occur when using stream-oriented protocols like TCP.

1.1.3.5 NDEF Message Description

The NFC data exchange format is a lightweight binary message format. The format is designed to encapsulate one or more application payloads into a single message construct called an NDEF message. Each NDEF message consists of one or more NDEF records. Each NDEF record can carry a payload of an arbitrary type and up to 2^31 – 1 octets in size. If the payload is larger, then the records can be chained to support bigger data.

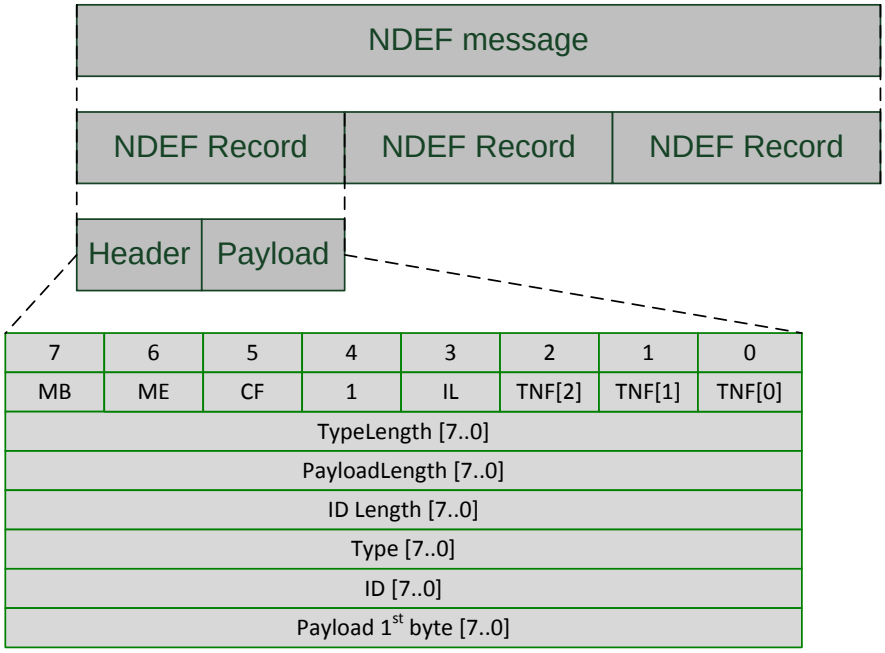


Figure 1. NDEF Message with Short Record Example

The first record in the NDEF message is marked with a message begin (MB) flag set and the last record is marked with a message end (ME) set. The minimum record length is one, which can be constructed by setting MB and ME in the first record. An NDEF message has no limit to the maximum number of NDEF records that can be carried.

NDEF messages must not overlap; for example, do not use MB and ME to nest NDEF messages. If the user wants to send more than one NDEF message, then each NDEF message must be encapsulated in the form of an NDEF record.

NOTE: NDEF records do not carry any index. The ordering of the records are given by the way they are serialized. If an intermediate application repacks the data, then it must take care of the ordering as well.

1.1.3.6 NDEF Record

NDEF records vary in length with the common format. The following is an NDEF record layout.

MB is a 1-bit field that, when set, indicates the start of NDEF Message.

ME is a 1-bit field that, when set, indicates the end of NDEF Message.

CF is a 1-bit field that indicates whether the record is either the first record chunk or a middle record chunk of a chunked payload.

An NDEF message can contain zero or more chunked payloads. A record chunk carries a chunk of payload. Chunked payloads can be used to partition dynamically generated content or very large entities into multiple subsequent record chunks serialized within the same NDEF message.

Each chunked payload is encoded as an initial chunk, followed by zero or more middle chunks, and finally by the terminating chunk. Each record chunk is encoded as an NDEF record using the following guidelines:

- The initial record chunk is an NDEF record with the chunk flag (CF) flag set. The type of the entire chunked payload must be indicated in the *TYPE* field regardless of whether the *PAYLOAD_LENGTH* field value is zero or not. The *PAYLOAD_LENGTH* field of this initial record indicates the size of the data carried in the *PAYLOAD* field of the this record only, not the entire payload size. Use the *ID* field to carry an identifier of the entire chunked payload.
- Each middle record chunk is an NDEF record with the CF flag set indicating that this record chunk contains the next chunk of data of the same type and with the same identifier as the initial record chunk. The value of the *TYPE_LENGTH* and the *IL* fields must be zero, and the type name format (TNF) field value must be 0x06 (unchanged).The *PAYLOAD_LENGTH* field indicates the size of the data carried in the *PAYLOAD* field of this single middle record only.
- The terminating record chunk is an NDEF record with the CF flag cleared, indicating that this record chunk contains the last chunk of data of the same type and with the same identifier as the initial record chunk. The value of the *TYPE_LENGTH* and the *IL* fields must be zero and the *TNF* field value must be 0x06 (Unchanged). The *PAYLOAD_LENGTH* field indicates the size of the data carried in the *PAYLOAD* field of this single terminating record only.

NOTE: A chunked payload must be entirely encapsulated within a single NDEF message. That is, a chunked payload must not span multiple NDEF messages. As a consequence, neither an initial nor a middle record chunk can have the ME flag set.

SR is a 1-bit field. If this flag is set, then the *PAYLOAD_LENGTH* field is a single octet. The short record layout is intended to compact encapsulation of small payloads, which will fit within *PAYLOAD* fields ranging between 0 and 255. (see Figure 1)

In the above layout, SR = 1. The *PAYLOAD_LENGTH* filed is only 1 octet. The max value is 2^8 – 1.

IL is a 1-bit field. If this field is set, then the *ID_LENGTH* field is present in the header as a single octet. If the *IL* flag is zero, the *ID_LENGTH* field is omitted from the record header and the *ID* field is also omitted from the record.

TNF is a 3-bit value that indicates the structure of the value of *TYPE* field. TNF values are defined below:

- 0x00 (Empty) indicates that there is no type or payload associated with this record. When used, the *TYPE_LENGTH*, *ID_LENGTH*, and *PAYLOAD_LENGTH* fields must be zero, and the *TYPE*, *ID*, and *PAYLOAD* fields are then omitted from the record. This TNF value can be used whenever an empty record is needed; for example, to terminate an NDEF message in cases where there the user application does not define a payload.
- 0x01 (NFC forum, well-known type) indicates that the *TYPE* field contains a value that follows the RTD TNF defined in NFC Forum RTD specification [NFC RTD].
- 0x02 (media-type) indicates that the *TYPE* field contains a value that follows the media-type BNF construct defined in RFC 2046 [RFC 2046].
- 0x03 (absolute-URI) indicates that the *TYPE* field contains a value that follows the absolute-URI BNF construct defined in RFC 3986 [RFC 3986]
- 0x04 (NFC forum, external type) indicates that the *TYPE* field contains a value that follows the RTD TNF defined in NFC Forum RTD specification [NFC RTD] for external type names.
- 0x05 (unknown) should be used to indicate that the type of the payload is unknown. This value is similar to the “application/octet-stream” media type defined by MIME RFC 2046 [RFC 2046]. When used, the *TYPE_LENGTH* field must be zero, which omits the *TYPE* field from the NDEF record.
- 0x06 (unchanged) must be used in all middle record chunks and terminating record chunks used in chunked payloads, but not in any other record. When used, the *TYPE_LENGTH* field must be zero, which omits the *TYPE* field the NDEF record.
- 0x07 (reserved)

The **TYPE_LENGTH** field is an unsigned 8-bit integer that specifies the length in octets of the *TYPE* field.

The **ID_LENGTH** field is an unsigned 8-bit integer that specifies the length in octets of the *ID* field. This field is present only if the IL flag is set to 1 in the record header. An *ID_LENGTH* of zero octets is allowed and, in such cases, the *ID* field is omitted from the NDEF record.

The **PAYLOAD_LENGTH** field is an unsigned integer that specifies the length in octets of the *PAYLOAD* field (the application payload).

- If the SR flag is set, the *PAYLOAD_LENGTH* field is a single octet representing an 8-bit unsigned integer. The max size is 2^8 – 1 octets.
- If the SR flag is clear, the *PAYLOAD_LENGTH* field is four octets representing a 32-bit unsigned integer. The max size will be 2^32 – 1 octets.

NOTE: A payload length of 0 is allowed, which omits the *PAYLOAD* field from the NDEF record. Application payloads larger than 2^32 – 1 octets can be accommodated by using chunked payloads.

The **TYPE** field describes the type of the payload. The value of the *TYPE* field must follow the structure, encoding, and format implied by the value of the *TNF* field as described in TNF section above.

NOTE: An NDEF parser receives an NDEF record with a *TNF* field value that it supports, but an unknown *TYPE* field value should interpret the type identifier of that record as if the *TNF* field value were 0x05 (unknown).

The type identifier must be globally unique and maintained with stable and well-defined semantics over time.

The **ID** field is an identifier in the form of a URI reference described in RFC 3986 [RFC 3986] . The required uniqueness of the message identifier is guaranteed by the generator.

NOTE: The URI reference can be either relative or absolute. NDEF does not define a base URI, which means user applications using relative URIs must provide an actual or a virtual base URI.

Middle and terminating record chunks (such as records containing other than the initial chunk of a chunked payload) must not have an *ID* field. All other records may have an *ID* field.

The **PAYLOAD** field carries the payload intended for the NDEF user application. Any internal structure of the data carried within the *PAYLOAD* field is opaque to NDEF.

1.2 System Overview

The system consists of a FRAM memory (embedded in an MCU), which is available over serial link or NFC. The MCU's advanced security is enabled thanks to AES encryption. The benefits of such a system split are highlighted in Section 1.3.

From a visual standpoint, the high-level description of the system is shown in Figure 2.

1.3 Supported Use Cases

Before moving on to architecture discussion and design, the following lists use cases supported by the design:

1.3.1 Use Case 1: Pervasive Readers

Situation: A system undergoes failure that requires emergency shutdown.

- Before:
  - Maintenance service might have spent many hours assessing the cause of failure from visual clues as powering system back on for debug is not an option in many installations.
- Value of TIDA-00230:
  - The logs stored in the memory of the MSP430FR5969 can be read via NFC, which is available in all recent Android smartphones.
  - The critical mass of NFC reader was only reached recently. Now, 300-Mu Android phones have used NFC since Android 4.4 (released 2013).
  - The ubiquity of NFC interfaces in factory automation is enabled.

1.3.2 Use Case 2: Pervasive Reader 2

Situation: A customer returns a system that is under guarantee.

- Before
  - Maintenance service might have spent many hours opening the system and assessing the cause of failure and applicability of guarantee.
- Value of TIDA-00230
  - The logs stored in the memory of the MSP430FR5969 can be read via NFC, which is available in all Android smartphones.
  - The logs may prove the system was operated out of the recommended temperature range.
  - Information is available within a few seconds and saves the return department's costs.

1.3.3 Use Case 3: Data Loggers

Situation: A system requires logs to be written on high duty cycle.

- Before
  - Single-level cells flash had endurances of 50 to few 100's k program or erase cycles (multi-level cells have endurances of 3- to 5-k program)
  - Writing logs every second were equivalent to 86.4k write cycles in a day.
  - Solution was to either under sample avoid aging of the Flash or over-design the size of the Flash.
- Value of TIDA-00230
  - FRAM can write a trillion of write cycles without losing data [1]. Now systems can log as and when needed and not be limited by what is possible
  - Loggers can now input all required information as often as needed.

1.3.4 Use Case 4: High-Bandwidth Loggers

Situation: A system needs to log a huge amount of memory when an emergency situation arises.

- Before
  - Existing dual channel solutions had very slow serial interfaces that need a lot of power to write.
  - The system needed to either design a secondary back-up power system (back-up battery, huge capacitor, and so on) or log less memory than the system truly needed.
- Value of TIDA-00230
  - FRAM enables high speed write at ten times less energy allowing all system logs without compromises

1.3.5 Use Case 5: Pervasive Writers

Situation: A system needs to be configured and programmed as fast as possible when installed in a factory. New settings (or calibration information) and new firmware patches are near impossible when installed in the factory.

- Before
  - Firmware patches required dedicated programming stations.
  - Configuration and settings updates required dedicated programming stations.
  - Hardware-only dual-port EEPROM did not offer enough memory for firmware or allow for strong authentication of data.
- Value of TIDA-00230
  - The device offers parameter and firmware patches (FOTA) from any modern smartphone.
  - MSP430 and AES allow authentication before data are read/written and offers stronger protection than basic password protection like current solutions on the market.

NOTE: This use case was enabled by hardware and will not be implemented in the software available for download.

2 Architecture and Block Diagrams

The total system consists of multiple components that have been already mentioned: a legacy system or mother board, which should have a spare serial connector or interface to interact with this design (TIDA-00230).

The design in itself focuses on the embedded aspect with two main parts: embedded hardware and embedded software, described in Section 2.1 and Section 2.2, respectively.

2.1 Embedded Design Hardware

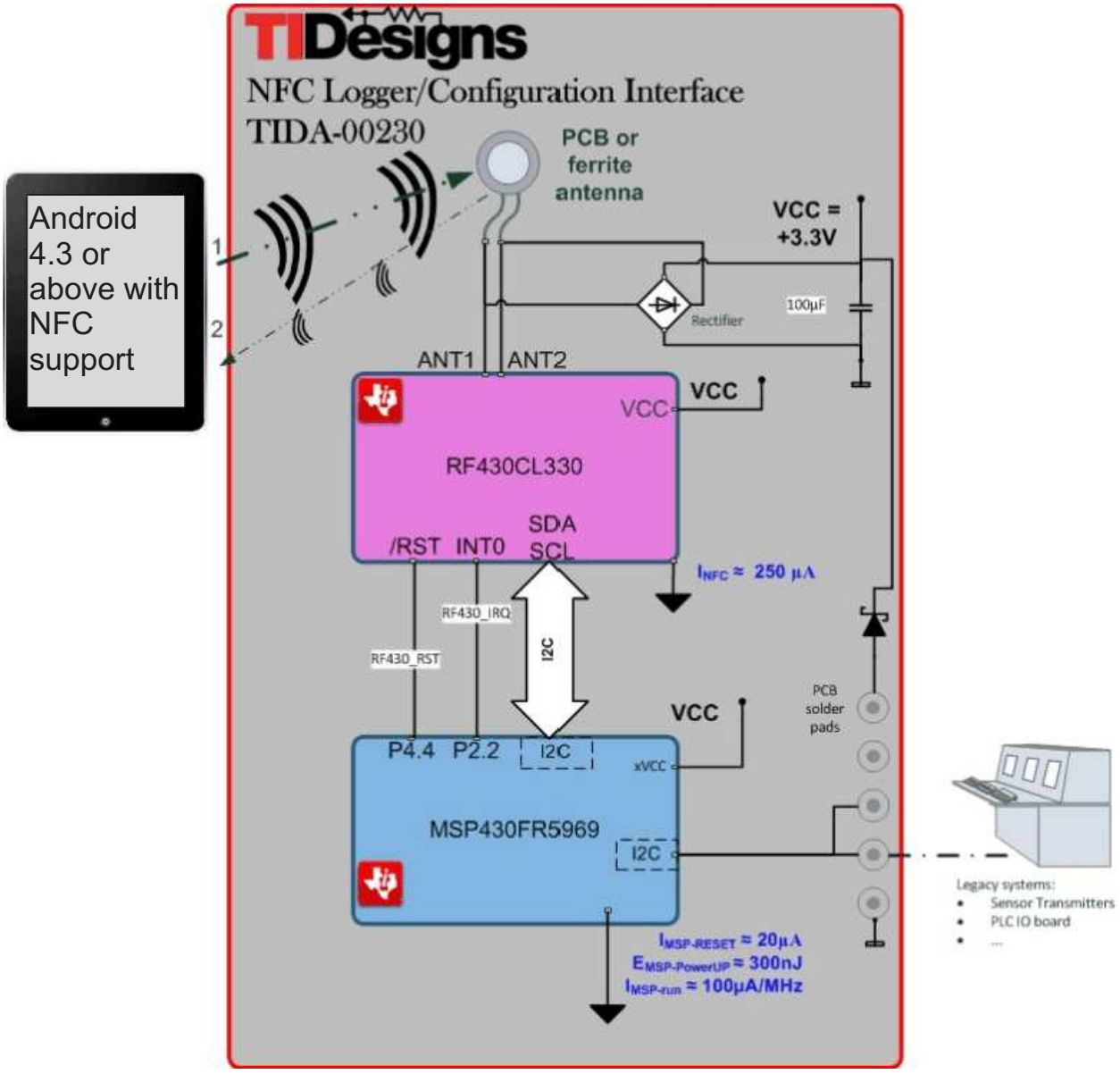


Figure 2. Hardware Block Diagram

2.1.1 Throughput Calculations

Since SPI only goes as high as 100 kHz, I2C was selected for interfacing with RF430CL330H as I2C goes as high as 400 kHz.

2.2 Embedded Design Software

The embedded software is a OS-free design leveraging the TI real-time support (RTS) library to initialize C environment.

Once the C environment is set up, the BSP will configure the MSP430FR5969 and the RF430CL330H (I2C interface, interrupts, and NDEF default content).

Once the NDEF memory of the dynamic tag has been set up, the MSP430FR5969 switches to slave mode to listen for possible logging commands.

LEDs are used to display different states and help solve problems during demonstrations. More details are provided in Section 4.

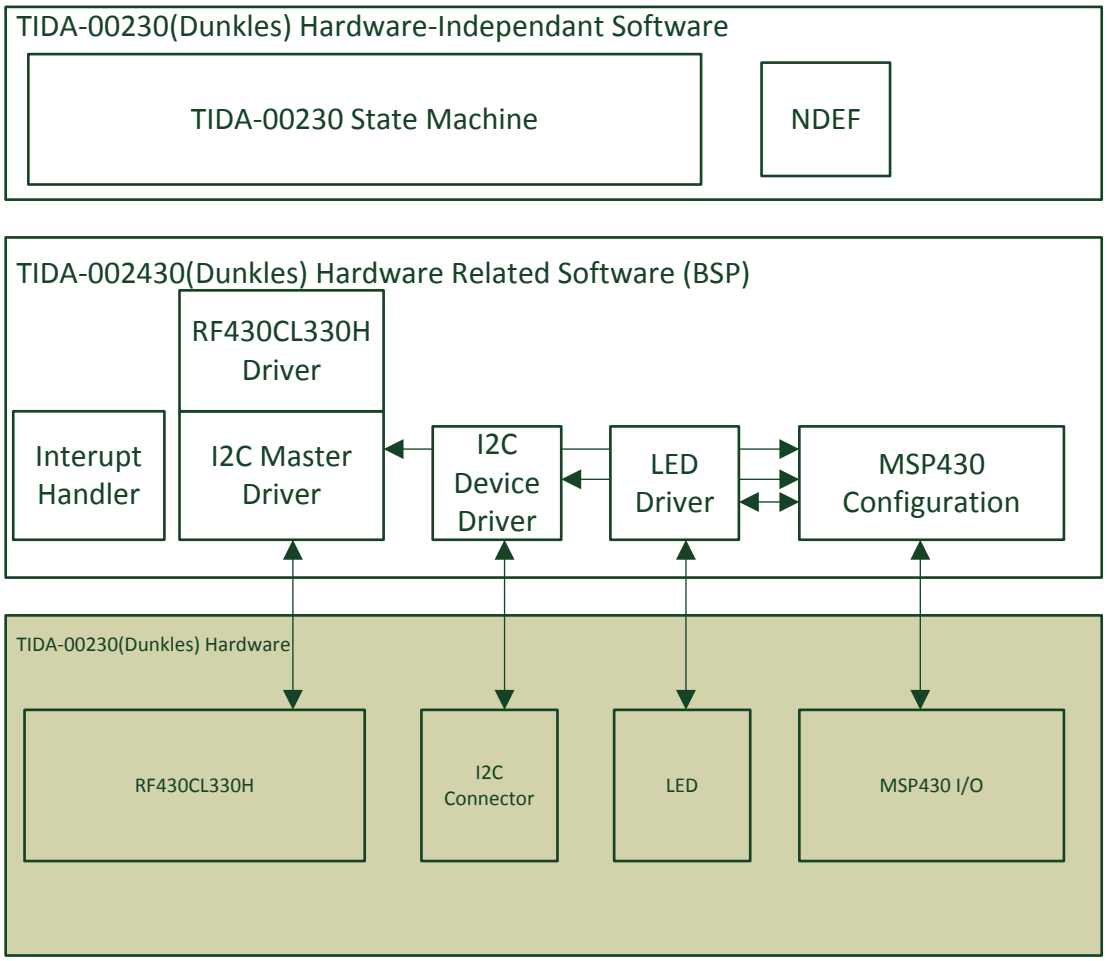


Figure 3. Software Block Diagram

3 Embedded System Design

3.1 NFC Antenna

The NFC antenna plays a dual role in the design. The antenna provides the reader/writer interface to the NDEF memory embedded in the RF430CL330H (NDEF memory that is programmed by the MSP430FR5969) and acts as the source of power for the design when the VCC_System (available on the I2C connector) is not available.

This NFC antenna has a low inductance since it operates at 13.56 MHz. The antenna can be air-core, ferrite core, or with a conductive trace on the PCB.

For maximum flexibility, the design offers three different antennas options:

- PCB antenna, which offers the easiest and most convenient option when the board can be aligned freely with the reader. In this option, the antenna is the RF interface as well as the power source.
- Surface mount (or through hole) antenna to offer orthogonal field capability (which requires to remove the 0-Ω resistors to avoid the PCB antenna to interfere)
- PCB antenna for communication and surface mount antenna for the power (which requires to mount another set of 0-Ω resistors).

Table 1 summarizes the different options.

Table 1. NFC Antenna PCB Options

COMPONENT OPTIONS	PCB ANTENNA: RF AND POWER	SMT ANTENNA: RF AND POWER	PCB ANTENNA: RF SMT ANTENNA: POWER
R1-R4	Mounted	Not mounted	Mounted
R7-R8	Not mounted	Mounted	Not mounted
L1-C12	Not mounted	Mounted	Mounted

3.1.1 PCB Antenna

Formulas for calculating (or approximating) PCB antenna values have been covered in many papers. [2][3][4]

For this design, the following formulas provide generic guidelines for the inductor value.

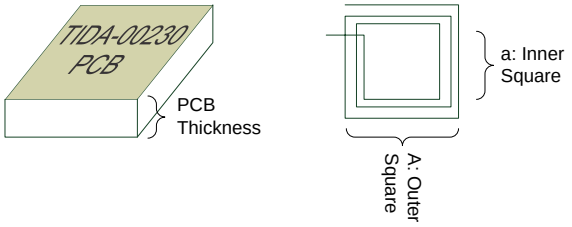


Figure 4. Dimensions for Calculating Antenna Inductance

$$L(\mu H) = \frac{11.04 \times \left[\left(A_{mm} + PCB_{th-mm} \right) \times N \right]^2}{100 \times \left[38.16 \times \left(A_{mm} + PCB_{th-mm} \right) + 90 \times \left(A_{mm} - a_{mm} \right) + 100 \times PCB_{th-mm} \right]} \quad (1)$$

where

- L(µH) is the PCB antenna inductance (in µH)
- A is the outer square dimension (in mm)
- a is the inner square dimension (in mm)
- PCB_{th}: the PCB thickness containing the antenna traces (in mm)

It is also possible to use online resources for those calculations (Missouri University of Science and Technology Inductance Calculators or others). However, Texas Instruments does not endorse any of these web calculators.

Use those formulas and calculators only to secure the right order of magnitude to achieve instead of making pure theoretical calculations and fine tuning the passive components on the real PCB (see Section 6.1).

3.1.2 Surface Mount or Through-Hole Ferrite Antenna

Given the directivity of PCB antenna and the absence of ferrite by design, consider a surface mount antenna (or through hole). To address those needs, the design provides a footprint for the 1D antenna, SDTR1103-HF2 (Figure 5), and the 3D antenna, 3DC15-HF (Figure 5). Table 2 summarizes the main characteristics of these antennas:

Table 2. Premo NFC Antenna Options

P/N	L (µH)	LENGTH (mm)	WIDTH (mm)	HEIGHT (mm)
3DC15HF-0003K	3	17.5	16	4
3DC15HF-0006K	6	17.5	16	4
3DC15HF-0009K	9	17.5	16	4
3DC15HF-0018K	18	17.5	16	4
SDTR1103-HF2-0001K	1	11.8	3.6	2.5
SDTR1103-HF2-0002K	2	11.8	3.6	2.5
SDTR1103-HF2-0003K	3	11.8	3.6	2.5
SDTR1103-HF2-0006K	6	11.8	3.6	2.5
SDTR1103-HF2-0020K	20	11.8	3.6	2.5

Figure 5. 3DC1515-HF SMD 3D Coil 17.5 × 16.0 × 4.0

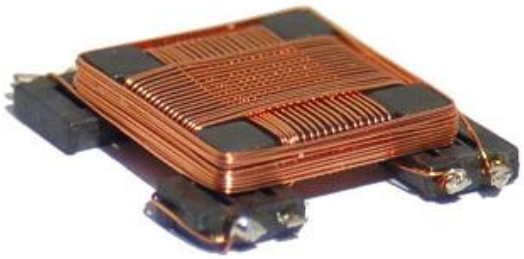


Figure 6. SDTR1103-HF2

3.2 Power Topology

There are two main different topologies possible: either using the same antenna for RF and power, or using two different antennas for each option. Figure 7 lists these options under the names Default, Ferrite, and Distinct Power. The first two options use the same antenna for RF and power, and the third option uses separate antennas due to the easily separated power generation (two Schottky diodes rectifying the RF field) from the RF input of the RF430CL330H.

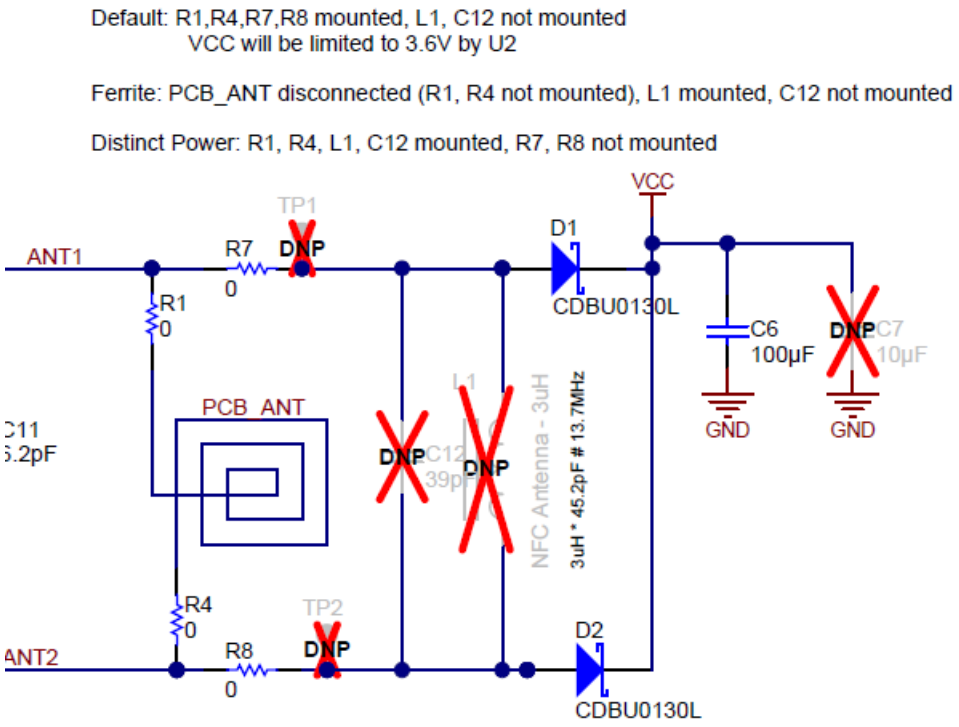


Figure 7. Antenna Schematics Variants, Extracted from Main Schematics (Default Variant)

The Schottky diodes were selected to minimize losses, whereas BAT54 classically have a 400-mV drop at 10 mA. The selected CDBU0130L has a VF of 350 mV at 10mA.

3.3 IC Selections

The system is leveraging

RF430CL330H, a dedicated product which is an NFC Tag Type 4 with combined NFC and NDEF memory which can be accessed and updated wirelessly via integrated ISO14443B-compliant RF interface that supports up to 848 kbps.

- RF430CL330H product folder: RF430CL330HTB
- RF430CL330H Dynamic NFC Interface Transponder Data Sheet (SLAS916)
- RF430CL330H Target Board User's Guide (SLOU373)

MSP430FR5969, from the family devices featuring embedded FRAM nonvolatile memory which brings unique capabilities for field updates and logging thanks to a write endurance of over 100 trillion cycles [1].

- This device was selected as it offers the best compromise between MCU clock frequency and total amount of FRAM

TPD1E10B06 is a single channel ESD protection device in a small 0402 package. The device offers over ±30KV IEC air-gap, over ±30KV contact ESD protection, and has an ESD clamp circuit with a back-to-back diode for bipolar or bidirectional signal support. The 10pF line capacitance is suitable for a wide range of applications supporting data rates up to 400Mbps. Typical application areas of the TPD1E10B06 include audio lines (microphone, earphone and speakerphone), SD interfacing, keypad or other buttons, and VBUS pins of USB ports (ID).The 0402 package is industry standard and convenient for component placement in space saving applications. The TPD1E10B06 is characterized for operation over ambient air temperature of –40°C to 125°C.

- This device was selected for its performance and 0402 package

TCA9517 While for simplicity and ease of interconnection it was decided not to include a I2C buffer like TCA9517, robust design should consider having such a buffer to secure that when the main system is un-powered, there is no low impedance introduce on the SDA or SCL lines which would prevent communication between the MSP430FR5969 MCU and the RF430CL330H dynamic tag.

This dual bidirectional I2C buffer is operational at 2.7 V to 5.5 V. The TCA9517A is a BiCMOS integrated circuit intended for I2C bus and SMBus systems. It can provide bidirectional voltage-level translation (up-translation/down-translation) between low voltages (down to 0.9 V) and higher voltages (2.7 V to 5.5 V) in mixed-mode applications. This device enables I2C and similar bus systems to be extended, without degradation of performance even during level shifting. The TCA9517A buffers both the serial data (SDA) and the serial clock (SCL) signals on the I2C bus, thus allowing two buses of 400-pF bus capacitance to be connected in an I2C application. This device can also be used to isolate two halves of a bus for voltage and capacitance. The TCA9517A has two types of drivers—A-side drivers and B-side drivers. All inputs and I/Os are overvoltage tolerant to 5.5 V, even when the device is unpowered (VCCB and VCCA = 0 V). The TCA9517A offers a higher contention level threshold, VILC, than the TCA9517 and can be used in applications where a larger input logic low is required on the B-side.

3.4 Hardware Power-Up Sequence

To power-up the MSP430FR5969, the charge is estimated to be approximately 75nAs. If the RF field charged the C6, C7 enough then the MSP430RF5969 will power-up. In the case of a DC source this will always be the case.

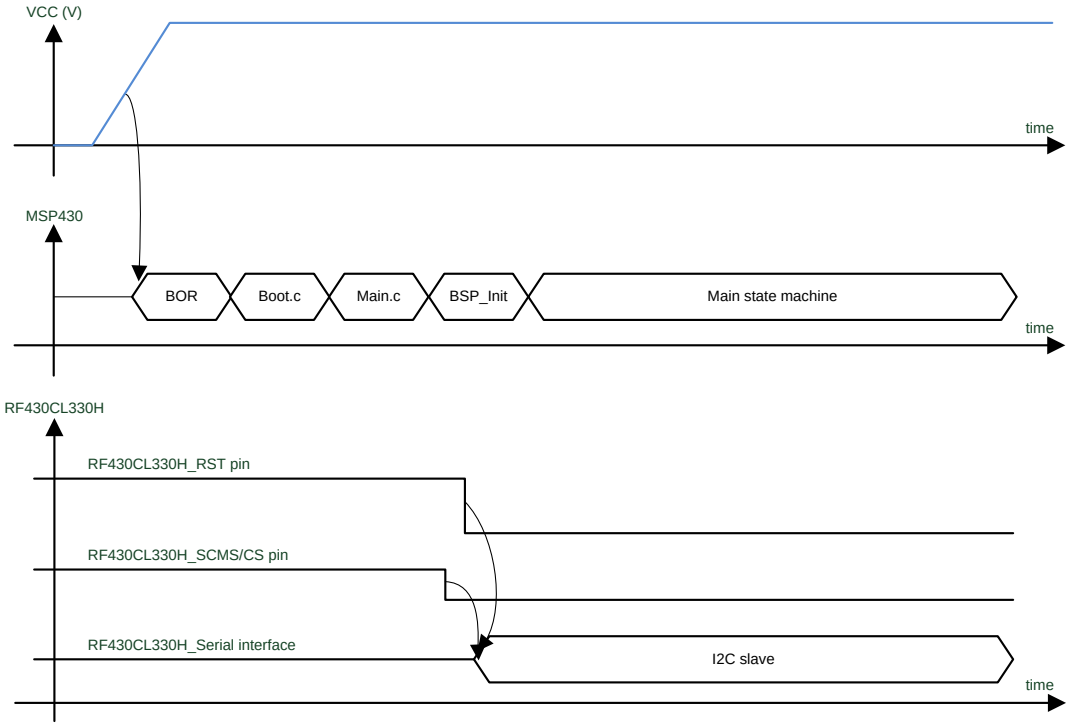


Figure 8. Hardware Power-Up Sequence

To power-up the MSP430FR5969, the charge is estimated to be approximately 75nAs. If the RF field charged the C6, C7 enough then the MSP430RF5969 will power-up. In the case of a DC source this will always be the case.

Once the device powers-up, a BOR (brown-out reset), a device reset is generated.

After a BOR, the initial device conditions are:

- The RST/NMI pin is configured in the reset mode. See User's Guide [5] Section 1.7 for details on configuring the RST/NMI pin.
- I/O pins are switched to input mode
  - Configuration of Digital I/Os: After BOR Reset To prevent any cross currents during start-up of the device, all port pins are high-impedance with Schmitt triggers, and their module functions disabled. To enable the I/O functionality after a BOR reset, the ports must be configured first and then the LOCKLPM5 bit must be cleared. as described in the User's Guide [5] Digital I/O chapter.
- Other peripheral modules and registers are initialized as described in their respective chapters.
- Status register (SR) is reset.
- The watchdog timer powers up active in watchdog mode.
- Program counter (PC) is loaded with the address contained at the SYSRSTIV reset location (0FFFEh).

NOTE: A device that is unprogrammed or blank is defined as having its reset vector value, residing at memory address FFFEh, equal to FFFFh. Upon system reset of a blank device, the device automatically enters operating mode LPM4. See Section 1.4 of *MSP430 Hardware Tools User's Guide* (SLAU278T) for information on operating modes and Table 6-3 of *MSP430FR59xx Mixed-Signal Microcontrollers* (SLAS704D) for details on interrupt vectors.[5][6]

3.4.1 RF430CL330H Power-Up

For the RF430CL330H, The SCMS/CS pin is sampled after tSPIvsI2C(MIN:1 ms) at the earliest and after tSPIvsI2C(MAX: 10 ms) at the latest.

After power-up or reset, the host should wait until device is ready to communicate using SPI or I2C which is at most 20 ms (tREADY)

3.4.2 Low Power Design Considerations

It is recommended that any unused pin with a secondary function that is shared with general-purpose I/O should follow the Px.0 to Px.7 unused pin connection guidelines as per Table 6-3 of *MSP430FR59xx Mixed-Signal Microcontrollers* (SLAS704D), “Connection of Unused Pins” [6].

While the RF430CL330H has a pull-up on its CS pin, and the design operates in I2C mode (meaning that CS pin will be grounded), the pull-up is disabled after the pin has been sampled.

While it may be surprising at first to use I2C between MSP430FR5969 and RF430CL330H, this serial bus has been selected over SPI for energy conservation reasons as the SCL clock can be as high as 400kHz in I2C and goes only up to 100kHz in SPI mode .

The software follows those principles for low power apps:

- Maximizing the time in LPM3 mode
- Use interrupts to wake the processor from LPM3
- Switch on peripherals only when needed
- Use low-power integrated peripherals

3.5 Design Variants

3.5.1 Default: PCB Antenna

By default, the PCB antenna is used to power the system and to communicate with the readers. Accordingly the SMT antenna (L1) and resonant cap (C12) are not mounted.

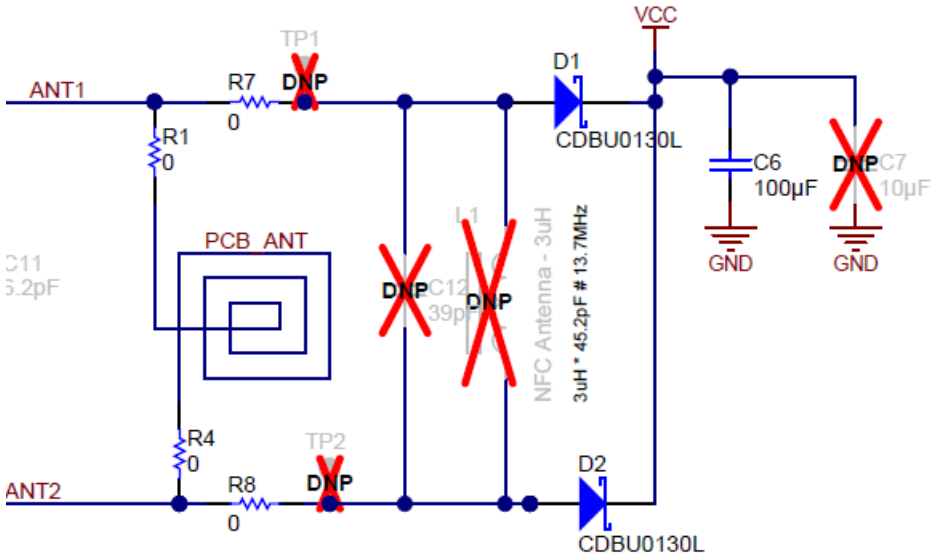


Figure 9. PCB Antenna (Default Variant)

3.5.2 Ferrite

In occasions where the PCB antenna is not desired (magnetic lflux directivity), efficiency, PCB area, and so on, a SMT antenna can be mounted: L1 and R1 and R4 should be removed to prevent the inductance of the PCB antenna to interfere with L1.

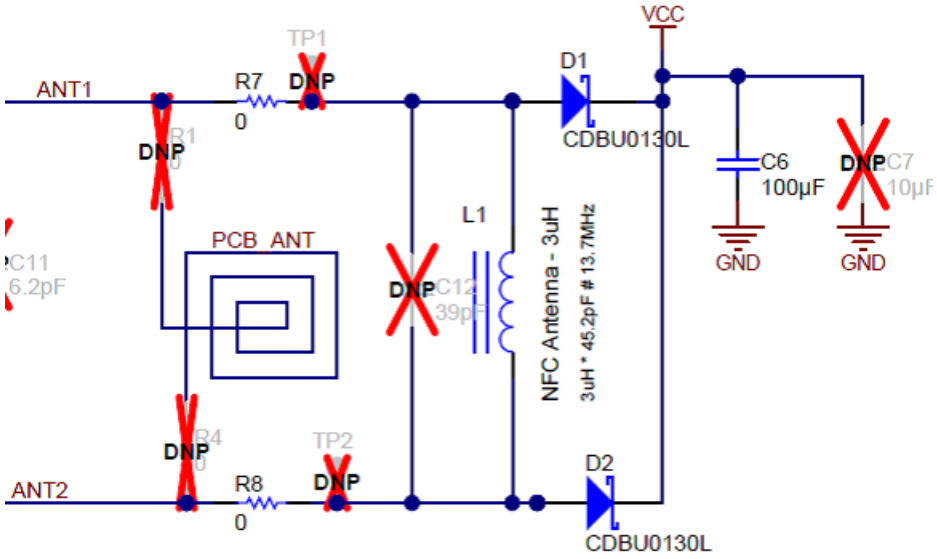


Figure 10. Ferrite NFC Antenna

3.5.3 Split Power and RF

In occasion where more power is needed, the SMT antenna and the PCB antenna can be isolated so that the PCB antenna will be used for communication and the SMT antenna will be used for power harvesting. In this case C12, L1 should be mounted and R7 and R8 should be removed.

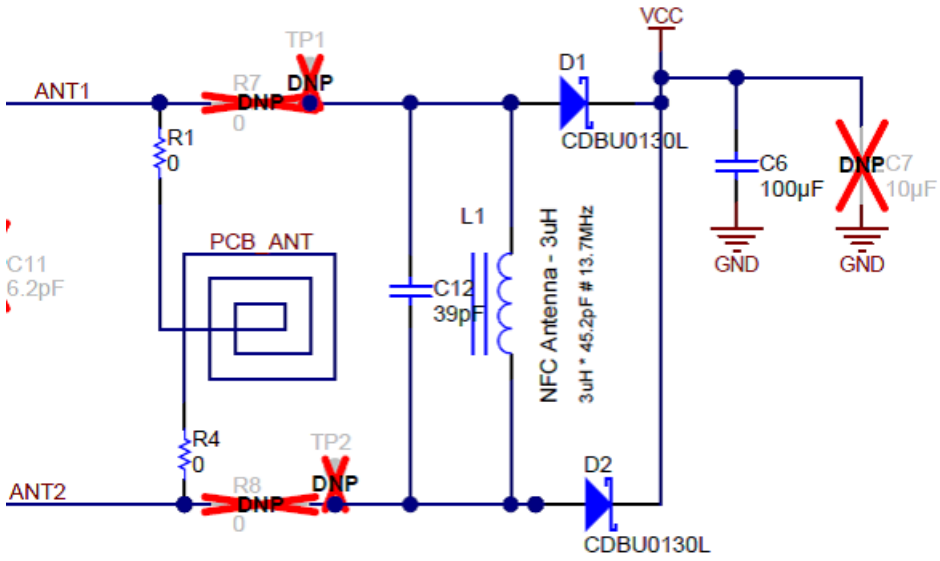


Figure 11. Split Power and RF Antenna

3.5.4 Buffered I2C

When adding an extension card to an existing system, many assumptions are needed on the mother board - legacy system. In occasions where the system would present a low impedance when not powered which could prevent the TIDA-00230 design from operating, a I2C buffer could be added.

Other reasons for adding an I2C buffer would include cases where the total capacitance of the mother board is already at the maximum allowed by I2C specs.

For those reasons and others, it is recommended to extend the design with an I2C buffer such as TCA9517.

3.6 Software Power-Up Sequence

When building the project with CCS, the linker will link rts.lib (real time support library) and will set the content of the reset vector with the address at which the entry point to the rts.lib: the _c_int00 symbol.

At _c_int00 (linked from boot.c in rts.lib), the code will code will do the following:

- Initialize global and static variables
- Initialize C environment variables
- Setup stack (SP)
- Call _main – at which point the code in the main function from the project will start being executed

Once entered in main(),Configuring the watchdog must be among the first actions taken by the program. Indeed it is not handled quickly, the watchdog expires, and a system reset occurs.

Figure 12. Software Power-Up Sequence

3.6.1 Software Architecture

The code architecture is interrupt driven, because doing so provides the most opportunities to power down the device. The device sleeps until an interrupt is received, thereby maximizing power efficiency modes. The power modes are controlled by bits within the status register (SR). The advantage of this is that the power mode in place prior to ISR execution is saved onto the stack. When the ISR reloads that value upon completing execution, program flow returns to that saved power mode.

However, by manipulating the saved SR value on the stack from within the ISR, program flow after the ISR can be diverted to a different power mode. This mechanism is an integral part of the MSP430 low-power operation, because it allows the device to quickly wake up in response to an interrupt. As an example, suppose a device is in the LPM0 low-power mode when an interrupt occurs. The MSP430 prepares for ISR execution, including the saving of the SR to the stack and clearing the SR. Clearing the SR causes an exit from LPM0 into active mode.

Within the ISR, the code developer places a statement that modifies the saved SR value by clearing the low-power bits. When the ISR completes, it reloads the values from the stack to their respective registers. Without having modified the bits, this action would put the device back into LPM0. Because the SR value has been modified to reflect a fully active device, the device stays active, and execution resumes at the PC value that had been saved to the stack prior to ISR execution.

Given this ability to change the power mode from the ISR, the developer could choose to implement the full functionality of the ISR within the routine itself, or use the ISR to wake up the processor and let the main loop handle all or part of the resulting functionality. Handling within the ISR ensures that the response to the interrupt event is immediate, provided that interrupts are enabled at the time of the event. However, while handling an ISR, interrupts are not enabled and will not be enabled until the ISR is completed. As a result, long ISRs may decrease system responsiveness.

The author choose which to have the ISR wake-up the processor and have the handling done in the main loop handle, as illustrated in Figure 17.

3.7 IC Data Sheet Extracts

For ease of understanding, data sheets extracts from the selected components are inserted in following section. It is however strongly recommended to read the data sheets of those components as following sections only serve for illustration and initial understanding. Critical details for those components are located in the data sheets and for brevity reasons were not included in this design documentation.

3.7.1 RF430CL330H

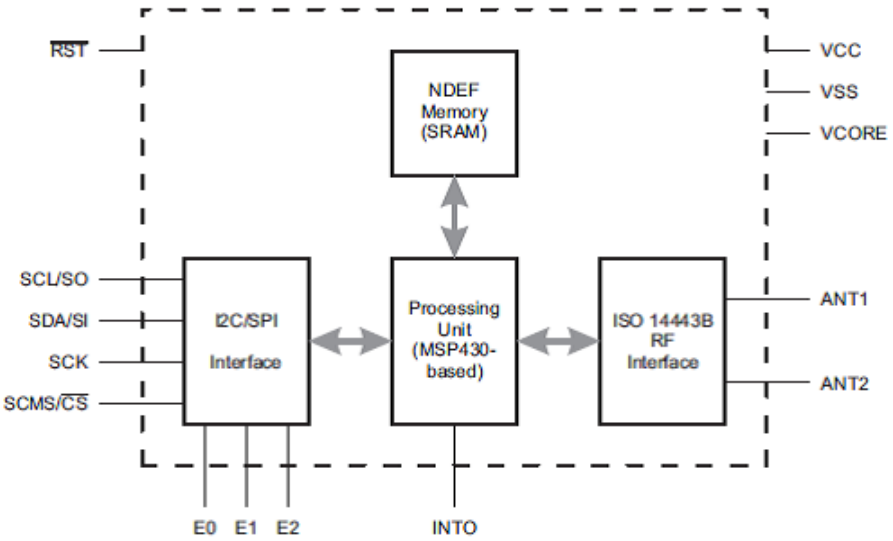


Figure 13. RF430CL330H Functional Block Diagram

NFC Tag Type 4

ISO14443B Compliant 13.56-MHz RF Interface supports up to 848 kbps

SPI or I2C Interface to Write and Read NDEF Messages to Internal SRAM

3kB SRAM for NDEF Messages

Automatic Checking of NDEF Structure

Interrupt Register and Output Pin to Indicate

NDEF Read or Write Completion

3.7.2 MSP430FR5969

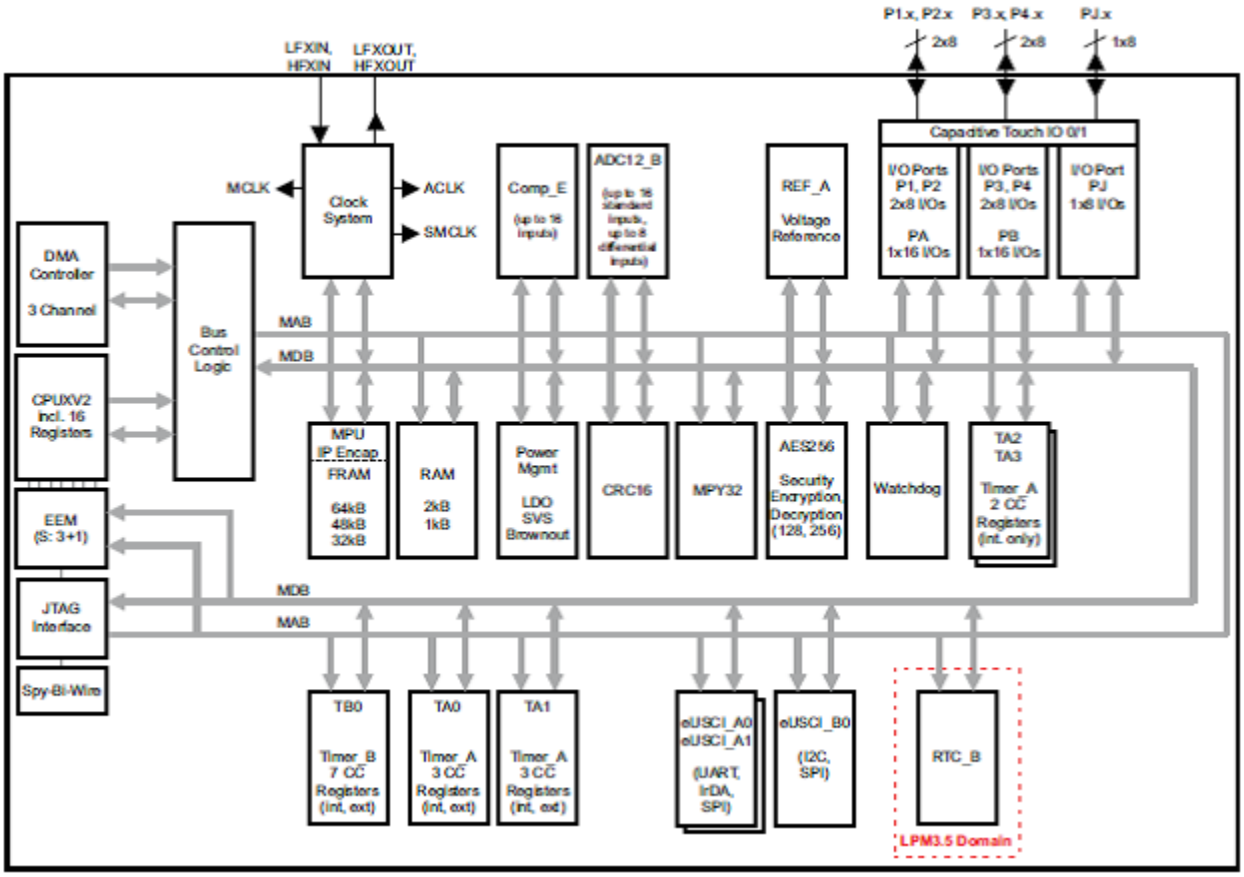


Figure 14. MSP430FR5969 Functional Block Diagram

- Embedded Microcontroller
  - 16-Bit RISC Architecture up to 16-MHz Clock
  - Wide Supply Voltage Range (1.8 to 3.6 V)
- Optimized Ultralow-Power Modes

Table 3. MSP430FR5969 Ultra Low Power Modes

MODE	CONSUMPTION (TYPICAL)
Active mode	103 µA/MHz
Standby (LPM3 with VLO)	0.4 µA
Real-time clock (LPM3.5 with Crystal)	0.5 µA
Shutdown (LPM4.5)	0.02 µA

- Ultralow-Power Ferroelectric RAM (FRAM)
  - Up to 64KB Nonvolatile Memory
  - Ultralow-Power Writes
  - Fast Write at 125 ns Per Word (64KB in 4 ms)
  - Unified Memory = Program + Data + Storage in One Single Space
  - 1015 Write Cycle Endurance
  - Radiation Resistant and Nonmagnetic

- Intelligent Digital Peripherals
  - 32-Bit Hardware Multiplier (MPY)
  - Three-Channel Internal DMA
  - Real-Time Clock (RTC) With Calendar and Alarm Functions
  - Five 16-Bit Timers With up to Seven Capture/Compare Registers Each
  - 16-Bit Cyclic Redundancy Checker (CRC)
- High-Performance Analog
  - 16-Channel Analog Comparator
  - 14-Channel 12-Bit Analog-to-Digital Converter (ADC) With Internal Reference and Sample-and-Hold (200 ksps at 75-µA Consumption)
  - Multifunction Input/Output Ports
  - All Pins Support Capacitive Touch Capability With No Need for External Components
  - Accessible Bit-, Byte-, and Word-Wise (in Pairs)
  - Edge-Selectable Wake From LPM on All Ports
  - Programmable Pullup and Pulldown on All Ports
- Code Security and Encryption
  - 128-Bit or 256-Bit AES Security Encryption and Decryption Coprocessor
  - Random Number Seed for Random Number Generation Algorithms
- Enhanced Serial Communication
  - eUSCI_A0 and eUSCI_A1 Support
    - UART With Automatic Baud-Rate Detection
    - IrDA Encode and Decode
    - SPI at Rates up to 10 Mbps
  - eUSCI_B0 Supports
    - I2C With Multiple Slave Addressing
    - SPI at Rates up to 8 Mbps
  - Hardware UART and I2C Bootstrap Loader (BSL)
- Flexible Clock System
  - Fixed-Frequency DCO With 10 Selectable Factory-Trimmed Frequencies
  - Low-Power Low-Frequency Internal Clock Source (VLO)
  - 32-kHz Crystals (LFXT)
  - High-Frequency Crystals (HFXT)
- Development Tools and Software
  - Professional Development Environments
  - Development Kit (MSP-TS430RGZ48C)
- Family Members
  - Table 2 in the datasheet *MSP430FR59xx Mixed-Signal Microcontrollers* summarizes 18 variants in three available package types [6].

For complete module descriptions, see the MSP430FR59xx and MSP430FR58xx Family User's Guide (SLAU367)

3.7.3 TPD1E10B06DPYR (Recommended Solution for ESD on All Interfaces)

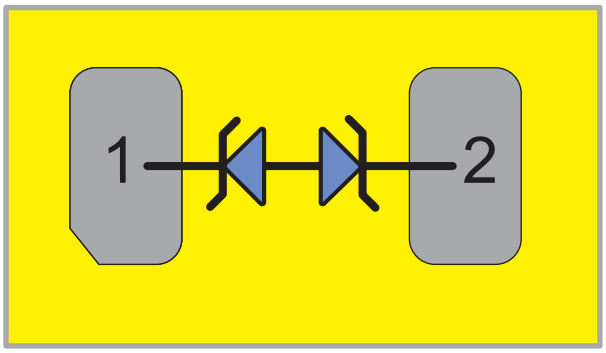


Figure 15. TPD1E10B06DPYR Functional View

Features:

- Provides System Level ESD Protection for Low-voltage IO Interface
- IEC 61000-4-2 Level 4
  - ±30kV (Air-Gap Discharge)
  - ±30kV (Contact Discharge)
- IEC 61000-4-5 (Surge): 6 A (8/20 µs)
- IO Capacitance 12 pF (Typ)
- RDYN 0.4 Ω (Typ) DC Breakdown Voltage ±6 V (Min)
- Ultra Low Leakage Current 100 nA (Max)
- 10V Clamping Voltage (Max at IPP = 1A)
- Industrial Temperature Range: –40°C to 125°C
- Space Saving 0402 Footprint (1 × 0.6 × 0.5mm)

3.7.4 TCA9517A (Included Only in Variants)

Table 1. FUNCTION TABLE

INPUT EN	FUNCTION
L	Outputs disabled
H	SDAA = SDAB SCLA = SCLB

Figure 1. FUNCTIONAL BLOCK DIAGRAM

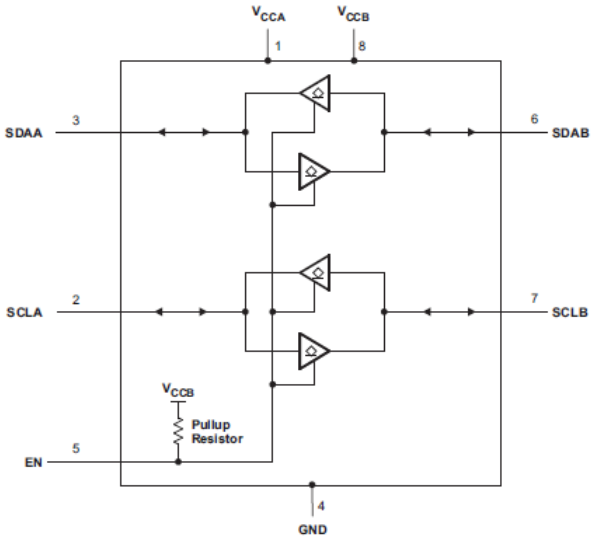


Figure 16. TCA9517 Functional View

Features:

- Two-Channel Bidirectional Buffer
- I2C Bus and SMBus Compatible
- Operating Supply Voltage Range of 0.9 to 5.5 V on A Side
- Operating Supply Voltage Range of 2.7 to 5.5 V on B side
- Voltage-Level Translation From 0.9 to 5.5 V and 2.7 to 5.5 V
- Footprint and Function Replacement for PCA9515B
- Active-High Repeater-Enable Input
- Open-Drain I2C I/O
- 5.5-V Tolerant I2C and Enable Input Support Mixed-Mode Signal Operation
- Lockup-Free Operation
- Accommodates Standard Mode and Fast Mode I2C Devices and Multiple Masters
- Powered-Off High-Impedance I2C Pins
- 400-kHz Fast I2C Bus
- Latch-Up Performance Exceeds 100 mA Per JESD 78, Class II
- ESD Protection Exceeds JESD 22
- 5500-V Human-Body Model (A114-A)
- 200-V Machine Model (A115-A)
- 1000-V Charged-Device Model (C101)

3.8 Design Failure Mode Effects and Analysis (DFMEA)

A successful FMEA activity helps to identify potential failure modes based on experience with similar products and processes - or based on common physics of failure logic. It is widely used in development and manufacturing industries in various phases of the product life cycle. Effects analysis refers to studying the consequences of those failures on different system levels.

It should be noted the judging criteria for the FMEA have been based around the functional ability to use the design as a demonstrator or a rapid prototyping vehicle.

Should the same schematics and layout be used for any other intention, a formal FMEA team should be assembled and reassess the findings of the below FMEA and also identify if the new conditions of usage bring new possible failure modes.

This FMEA being at system level, it does not intend nor provides any integrated circuits failure modes information.

Table 4. DFMEA

ITEM OR FUNCTION	POTENTIAL FAILURE MODES	POTENTIAL EFFECTS	SEVERITY	CLASS	POTENTIAL CAUSES	OCCURRENCE	DETECTION	RPN	RECOMMENDED ACTIONS	TARGET COMPLETION DATE
PWR	Crash before end of transmission	System stuck	6	SP	Cap too small or not enough energy harvested					
PWR	VCC stuck below VCCmin	System stuck	6							
Serial-IF	SCL or SDA not toggling	No communication to TIDA-00230	8		Resistive load too low on lines (when main system is not powered)	3	8	192	Check theoretical possibility	6/1/2014
Serial-IF	SCL or SDA edges too slow	No communication to TIDA-00230	8		System cap too high for our MSP430 USCI driver	3	8	192	Check theoretical possibility	6/1/2014
PWR	VCC too high	IC dead	6		RF field too strong	1	10	60	N/A	6/1/2014
PWR	VCC too high	Reduced lifetime	4		RF field too strong	1	10	40		6/1/2014

Table 5. DFMEA Action Results

ITEM OR FUNCTION	ACTIONS TAKEN	SEVERITY	OCCURRENCE	DETECTION	RPN
PWR					
PWR					
Serial-IF	Recommend I2C buffer in documentation tca9517a, …	1	3	8	24
Serial-IF	Recommend I2C buffer in documentation tca9517a, …	1	3	8	24
PWR					0
PWR					0

4 Software Description

This section does not intend to cover generic aspects of coding on MSP430 but rather specific design characteristics of the TIDA-00230 software.

When modifying the project files, TI provides header files for every MSP430 device in production. These header files have constants for all the registers and bits in a given device, which match the names provided in the user's guides. Using these constants within code greatly enhances its readability. It also gives any code that uses them a similar look-and-feel that enables another engineer familiar with MSP430, including TI's support team, to more quickly grasp the code. Every code example and application report from TI uses these headers where applicable.

Several intrinsic functions are made available in MSP430 development environments when writing in C. Sometimes, the only way to accomplish a critical task is to use an intrinsic function. Other intrinsics provide an opportunity to do things more efficiently.

The most common example of a critical task that can only be done using intrinsic functions is entering/exiting low power modes. Doing so requires manipulation of bit values not otherwise accessible at the level of C, since they reside within the CPU's status register. If entering LPM3 within the IAR development environment, an intrinsic function is used:
__bis_SR_register(LPM3_bits + GIE);

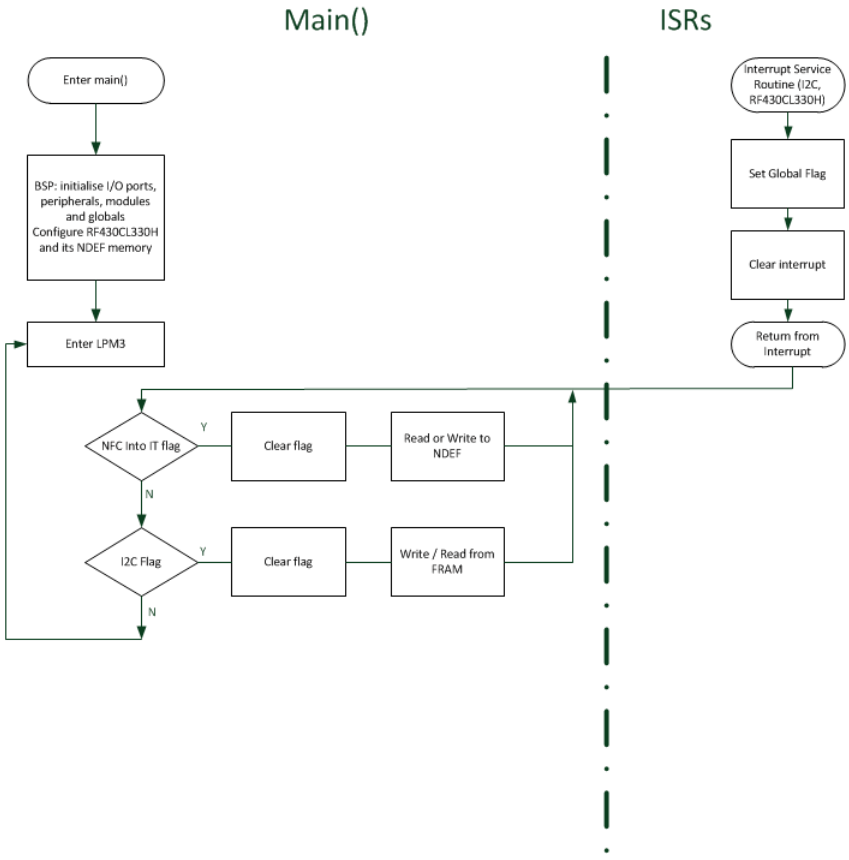


Figure 17. Software Flow Diagram

Another way to look at the program execution is to look at it through a sequence diagram, showing different event of logging, followed by a system failure and then the Android reader powering the system to access the log memory over NFC. This sequence is detailed in Figure 18).

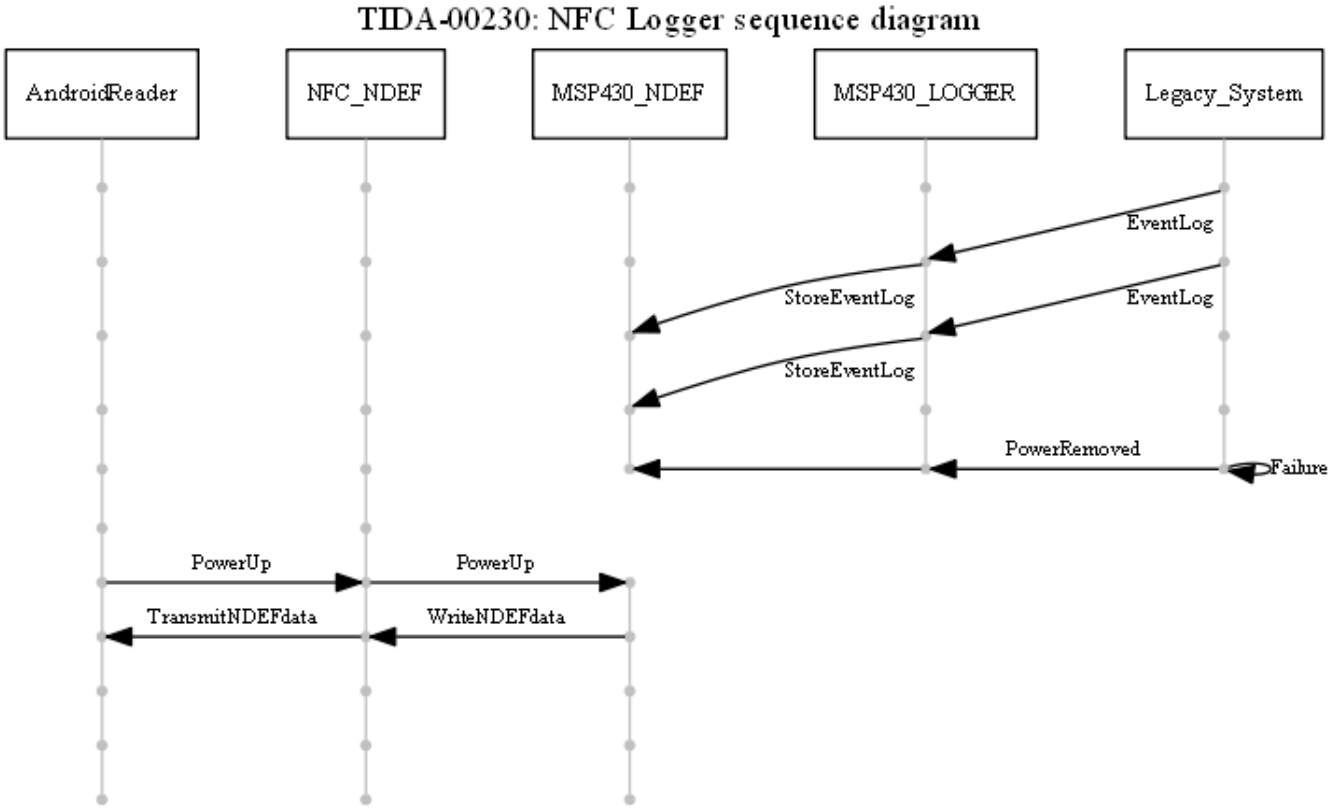


Figure 18. Sequence Diagram

4.1 NDEF Memory Flow

A specific flow is to be followed to ensure that the RF430CL330H NDEF memory is only accessed at once by either the MSP430FR5969 or by the NFC reader/writer. The sequence is detailed below in Figure 19.

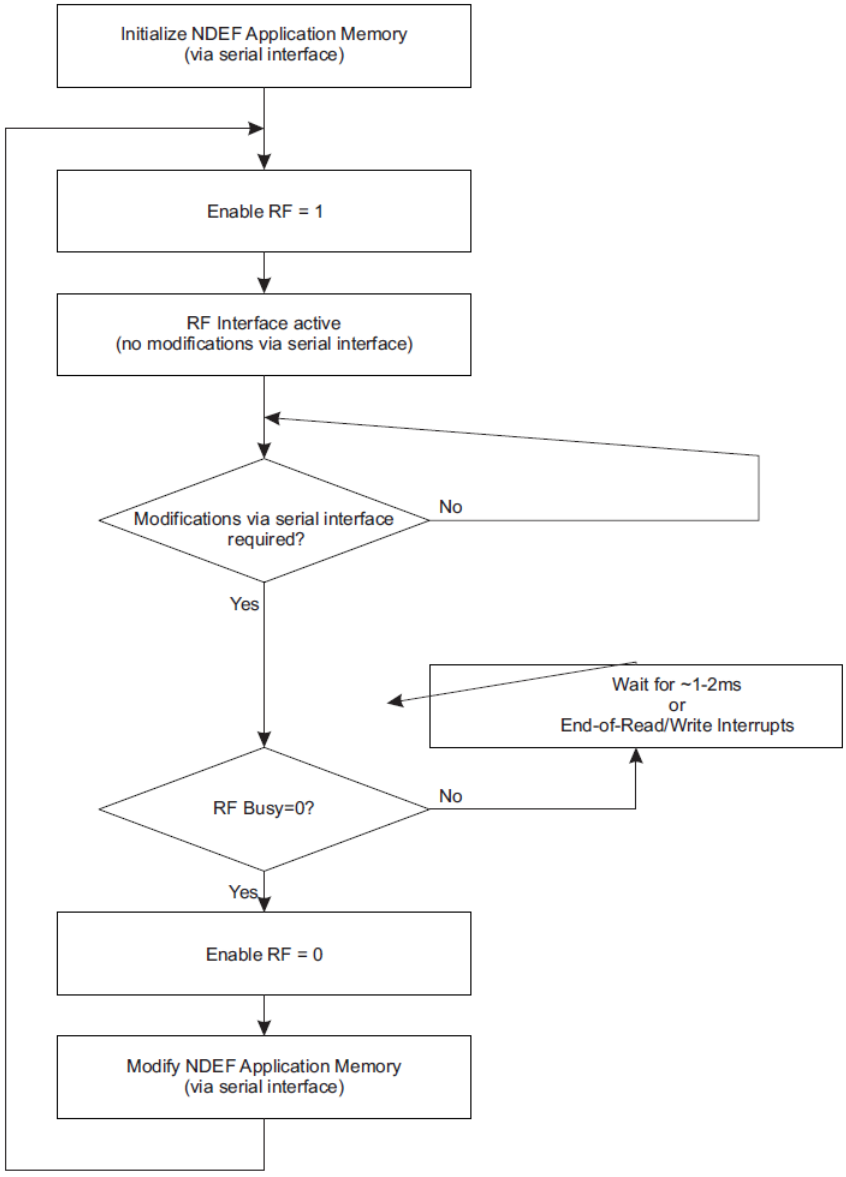


Figure 19. Recommended NDEF Data Memory Flow

4.2 Software Documentation from Doxygen

4.2.1 File Index

Here is a list of all files with brief descriptions:

- [Installation folder]/src/BSP.c
- [Installation folder]/src/BSP.h
- [Installation folder]/src/main.c
- [Installation folder]/src/MSP430_logger.c
- [Installation folder]/src/MSP430_NDEF.c
- [Installation folder]/src/MSP430_NDEF.h
- [Installation folder]/src/RF430CL330.c
- [Installation folder]/src/RF430CL330.h

4.2.2 Data Structure Documentation

4.2.2.1 MSP430_NDEF_image Struct Reference

#include <MSP430_NDEF.h>

Data Fields

- unsigned char ApplicationName
- unsigned char File_ID
- unsigned char CCLEN
  File ID : 0xE103h
- unsigned char MappingVersion
  CCLen
- unsigned char MLe
- unsigned char MLc
- unsigned char Tag_n_Length
- unsigned char FileIdentifier
- unsigned char MaxNDEFSize
- unsigned char NDEFReadAccess_n_WriteAccess
- unsigned char NDEFHeader
- unsigned char NDEFPayLoad

Detailed Description

Definition at line 172 of file MSP430_NDEF.h.

Field Documentation

unsigned char ApplicationName

- Definition at line 173 of file MSP430_NDEF.h

unsigned char CCLEN

- File ID: 0xE103h
- Definition at line 176 of file MSP430_NDEF.h

unsigned char File_ID

- Definition at line 175 of file MSP430_NDEF.h

unsigned char FileIdentifier

- Definition at line 182 of file MSP430_NDEF.h

unsigned char MappingVersion

- CCLen
- Definition at line 177 of file MSP430_NDEF.h

unsigned char MaxNDEFSize

- Definition at line 183 of file MSP430_NDEF.h

unsigned char MLc

- Definition at line 179 of file MSP430_NDEF.h

unsigned char MLe

- Definition at line 178 of file MSP430_NDEF.h

unsigned char NDEFHeader

- Definition at line 186 of file MSP430_NDEF.h

unsigned char NDEFPayLoad

- Definition at line 187 of file MSP430_NDEF.h

unsigned char NDEFReadAccess_n_WriteAccess

- Definition at line 184 of file MSP430_NDEF.h

unsigned char Tag_n_Length

- Definition at line 180 of file MSP430_NDEF.h

The documentation for this struct was generated from the following file:

- [Installation folder]/src/MSP430_NDEF.h

4.2.2.2 NDEF_File_Headers Struct Reference

#include <MSP430_NDEF.h>

Data Fields

- unsigned char NDEF_File_ID
- unsigned char NDEF_Len
- unsigned char RecordHEader_TypeLength
- unsigned char PayloadLength_PayloadType
- unsigned char LanguageLength_Language

Detailed Description

Definition at line 190 of file MSP430_NDEF.h

Field Documentation

unsigned char LanguageLength_Language

- Definition at line 195 of file MSP430_NDEF.h

unsigned char NDEF_File_ID

- Definition at line 191 of file MSP430_NDEF.h

unsigned char NDEF_Len

- Definition at line 192 of file MSP430_NDEF.h

unsigned char PayloadLength_PayloadType

- Definition at line 194 of file MSP430_NDEF.h

unsigned char RecordHEader_TypeLength

- Definition at line 193 of file MSP430_NDEF.h

The documentation for this struct was generated from the following file:

- [Installation folder]/src/MSP430_NDEF.h

4.2.3 File Documentation

4.2.3.1 [Installation folder]/src/BSP.c File Reference

```
#include "BSP.h"
#include "msp430.h"
#include "RF430CL330.h"
```

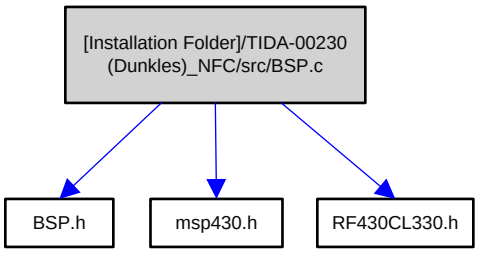


Figure 20. BSP.c Dependency Graph

Functions

- void BSP_MSPEXP430FR5739_TIDMDYNAMICNFCTAG (void)
- void I2C_to_RF430CL330H_Init ()
  - Configures the UCBO in I2C slave mode (to communicate with the RF430CL330
- void BSP_TIDA00230 (void)
  - Provides all the hardware initialization for the TIDA-00230 board
- void Timer_Init ()
- __interrupt void Timer1_A0_ISR (void)
- void Low_Power_Delay_ms (unsigned int ms)
- void BSP_Init (void)
  - Provides all the hardware initialization see BSP.h for supported hardware platforms
- void BSP_Enable_INTO_IT ()
  - Uses the hardware definition to clear then enable the relevant MSP430 PORT interrupts

Function Documentation

void BSP_Enable_INTO_IT (void)

- Uses the hardware definition to clear then enable the relevant MSP430 PORT interrupts. This function enables the interrupts on the MSP430 INTO line (see Board Configuration to see which line this is)
- Clears any pending flags
- Enables interrupts
- Definition at line 278 of file BSP.c

Figure 21 is the caller graph for this function:

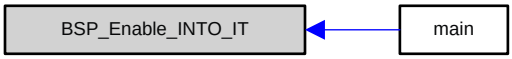


Figure 21. BSP_Enable Caller Graph

void BSP_Init (void)

- Provides all the hardware initialization see BSP.h for supported hardware platforms
- While multiple hardware configuration are supported, only TIDA-00230 is used to validate. Mileage on other configurations may vary.
- Definition at line 263 of file BSP.c.

Figure 22 is the call graph for this function:

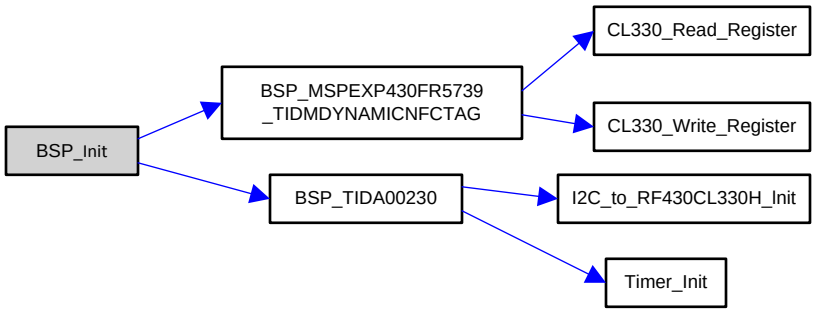


Figure 22. BSP_Init Call Graph

Here is the caller graph for this function:

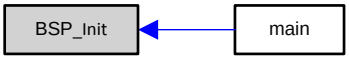


Figure 23. BSP_Init Caller Graph

void BSP_MSPEXP430FR5739_TIDMDYNAMICNFCTAG (void)

```
*
*                                /|\  /|\     (Host/Tester)
*                  RF430         10k  10k     MSP430FR5739
*                 (Slave)         |    |        Master
*                                 |    |
*             |                SDA|<-|----+->|P1.6/UCB0SDA  XIN|-+
*             |                   |   | I2C   |                 |
*             |                SCL|<-+------>|P1.7/UCB0SCL XOUT|-
*             |                   |          |                 |
*       GND<--|E(2-0)         /RST|<---------|P3.7         P4.0|<----S1
*             |               INTO|--------->|P2.0         P4.1|<----S2
*             |                   |          |                 |
*             |                   |          |            P3.4-6|---->LED5-7
*             |                   |          |                 |
*             |                   |          |                 |
*
*           TIDM-DYNAMICNFCTAG              MSP-EXP430FR5739
```

Figure 24. Overview of Hardware Configuration MSPEXP430FR5739_TIDMDYNAMICNFCTAG

- Definition at line 47 of file BSP.c.

Here is the call graph for this function:

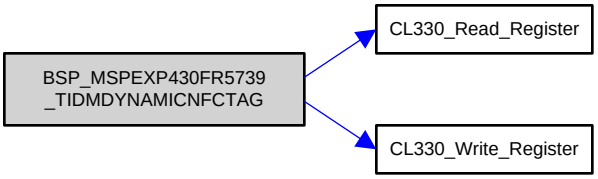


Figure 25. BSP_MSPEXP430FR5739_TIDMDYNAMICMFCTAG Call Graph

Here is the caller graph for this function:

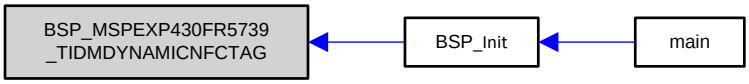


Figure 26. BSP_MSPEXP430FR5739_TIDMDYNAMICMFCTAG Caller Graph

void BSP_TIDA00230 (void)

- Provides all the hardware initialization for the TIDA-00230 board.

```
*
*                   NFC chip      /|\  /|\         Host
*                                4.75k  4.75k
*                  (I2C Slave)     |    |         I2C Master for RF430, I2C Slave for Mother-
Board / Legacy systems
*                  _____________   |    |   _________________
*             |               SDA|<-|----+->|P1.6/UCB0SDA     |
*             |                   |  | I2C   |                 |
*             |               SCL|<-+------>|P1.7/UCB0SCL     |
*             |                   |          |                 |
*       GND<--|E(2-0)         /RST|<---------|P3.7         P4.0|<----S1
*             |               INTO|--------->|P2.2         P4.1|<----S2
*             |                   |          |                 |
*             |                   |          |             P4.5|---->LED_D3
*             |                   |          |             P4.6|---->LED D4
*             |                   |          |                 |
*
*           RF430CL330H                     MSP430FR5969IRGZ
```

Figure 27. Overview of the Hardware Corresponding to TIDA-00230 Configuration

- DRIVE CS Low to indicate I2C Mode - should be pulled low within 1ms of boot, so we pull-it low before sending the reset to the RF430CL330H
- P4.4 set to '0'
- Reset RF430 to secure the sampling for CS low is done properly (set CS low above)
- Definition at line 171 of file BSP.c

Here is the call graph for this function:

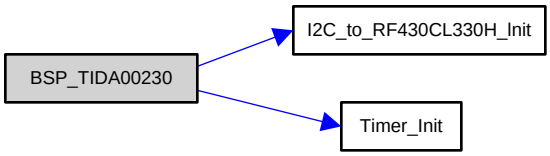


Figure 28. BSP_TIDA00230 Call Graph

Here is the caller graph for this function:

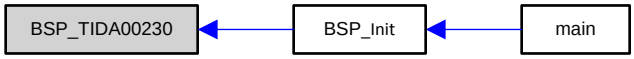


Figure 29. BSP_TIDA00230 Caller Graph

void I2C_to_RF430CL330H_Init ()

- Configures the UCBO in I2C slave mode (to communicate with the RF430CL330
- Definition at line 149 of file BSP.c.

Here is the caller graph for this function:

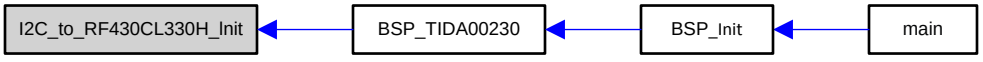


Figure 30. I2C_to_RF430CL330H_Init Caller Graph

void Low_Power_Delay_ms (unsigned int ms)

- Definition at line 248 of file BSP.c

__interrupt void Timer1_A0_ISR (void)

- Definition at line 241 of file BSP.c

void Timer_Init ()

- Definition at line 233 of file BSP.c

Here is the caller graph for this function:

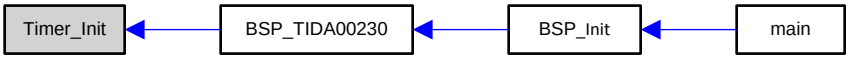


Figure 31. Timer_Init Caller Graph

4.2.3.2 [Installation folder]/src/BSP.h File Reference

This graph shows which files directly or indirectly include this file:

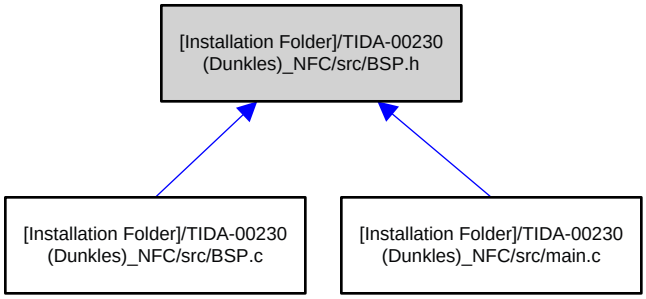


Figure 32. BSP.h Dependency Graph

Macros

- #define TIDA00230
- #define RF430CL330_I2C_ADD = 0x0028
- #define PORT_INTO_IN P2IN
  RF430CC330::INTO pin -------— >MSP430FR5969::Port 2, #2.
- #define PORT_INTO_OUT P2OUT
- #define PORT_INTO_DIR P2DIR
- #define PORT_INTO_SEL0 P2SEL0
- #define PORT_INTO_SEL1 P2SEL1
- #define PORT_INTO_REN P2REN
- #define PORT_INTO_IE P2IE
- #define PORT_INTO_IES P2IES
- #define PORT_INTO_IFG P2IFG
- #define INTO BIT2
- #define PORT_RST_OUT P4OUT
  RF430CC330::RST pin <———MSP430FR5969::Port 4, #4.
- #define PORT_RST_DIR P4DIR
- #define PORT_RST_SEL0 P4SEL0
- #define PORT_RST_SEL1 P4SEL1
- #define RST BIT4
- #define PORT_LED_OUT P4OUT
  LEDs : D3<–> P4.5, D4<–>P4.6.
- #define PORT_LED_DIR P4DIR
- #define PORT_LED_SEL0 P4SEL0
- #define PORT_LED_SEL1 P4SEL1
- #define LED_D3 BIT5
- #define LED_D4 BIT6
- #define PORT_I2C_OUT P1OUT
- #define PORT_I2C_DIR P1DIR
- #define PORT_I2C_SEL0 P1SEL0
- #define PORT_I2C_SEL1 P1SEL1
- #define SDA BIT6
- #define SCL BIT7
- #define PORT_CS P3OUT
- #define PORT_CS_SEL0 P3SEL0
- #define PORT_CS_DIR P3DIR
- #define PORT_CS_PIN BIT4

Functions

- void BSP_Init (void)
  Provides all the hardware initialization see BSP.h for supported hardware platforms
- void BSP_Enable_INTO_IT (void)
  Enables the interrupts on the MSP430 INTO line (see Board Configuration to see which line this is)
- void BSP_MSPEXP430FR5739_TIDMDYNAMICNFCTAG (void)
- void BSP_TIDA00230 (void)
  Provide all the hardware initialization for the TIDA-00230 board
- void Timer_Init ()

Macro Definition Documentation

#define INTO BIT2

- Definition at line 85 of file BSP.h

#define LED_D3 BIT5

- Definition at line 99 of file BSP.h

#define LED_D4 BIT6

- Definition at line 100 of file BSP.h

#define PORT_CS P3OUT

- Definition at line 111 of file BSP.h

#define PORT_CS_DIR P3DIR

- Definition at line 113 of file BSP.h

#define PORT_CS_PIN BIT4

- Definition at line 114 of file BSP.h

#define PORT_CS_SEL0 P3SEL0

- Definition at line 112 of file BSP.h

#define PORT_I2C_DIR P1DIR

- Definition at line 104 of file BSP.h

#define PORT_I2C_OUT P1OUT

- Definition at line 103 of file BSP.h

#define PORT_I2C_SEL0 P1SEL0

- Definition at line 105 of file BSP.h

#define PORT_I2C_SEL1 P1SEL1

- Definition at line 106 of file BSP.h

#define PORT_INTO_DIR P2DIR

- Definition at line 78 of file BSP.h

#define PORT_INTO_IE P2IE

- Definition at line 82 of file BSP.h

#define PORT_INTO_IES P2IES

- Definition at line 83 of file BSP.h

#define PORT_INTO_IFG P2IFG

- Definition at line 84 of file BSP.h

#define PORT_INTO_IN P2IN

- RF430CC330::INTO pin ———>MSP430FR5969::Port 2, #2
- Definition at line 76 of file BSP.h

#define PORT_INTO_OUT P2OUT

- Definition at line 77 of file BSP.h

#define PORT_INTO_REN P2REN

- Definition at line 81 of file BSP.h

#define PORT_INTO_SEL0 P2SEL0

- Definition at line 79 of file BSP.h.

#define PORT_INTO_SEL1 P2SEL1

- Definition at line 80 of file BSP.h.

#define PORT_LED_DIR P4DIR

- Definition at line 96 of file BSP.h

#define PORT_LED_OUT P4OUT

- LEDs : D3 <–> P4.5, D4<–>P4.6
- Definition at line 95 of file BSP.h

#define PORT_LED_SEL0 P4SEL0

- Definition at line 97 of file BSP.h

#define PORT_LED_SEL1 P4SEL1

- Definition at line 98 of file BSP.h

#define PORT_RST_DIR P4DIR

- Definition at line 89 of file BSP.h

#define PORT_RST_OUT P4OUT

- RF430CC330::RST pin <———MSP430FR5969::Port 4, #4
- Definition at line 88 of file BSP.h

#define PORT_RST_SEL0 P4SEL0

- Definition at line 90 of file BSP.h

#define PORT_RST_SEL1 P4SEL1

- Definition at line 91 of file BSP.h

#define RF430CL330_I2C_ADD = 0x0028

- Definition at line 72 of file BSP.h

#define RST BIT4

- Definition at line 92 of file BSP.h

#define SCL BIT7

- Definition at line 108 of file BSP.h

#define SDA BIT6

- Definition at line 107 of file BSP.h

#define TIDA00230

- Definition at line 12 of file BSP.h

Function Documentation

void BSP_Enable_INTO_IT (void)

- Enables the interrupts on the MSP430 INTO line (see Board Configuration to see which line this is)
- Clears any pending flags
- Enables interrupts
- Definition at line 278 of file BSP.c

Here is the caller graph for this function:

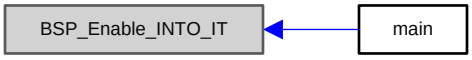


Figure 33. BSP_Enable_INTO_IT Caller Graph

void BSP_Init (void)

- Provides all the hardware initialization see BSP.h for supported hardware platforms.
- While multiple hardware configurations are supported, only TIDA-00230 is used to validate. Mileage on other configurations may vary
- Definition at line 263 of file BSP.c.

Here is the call graph for this function:

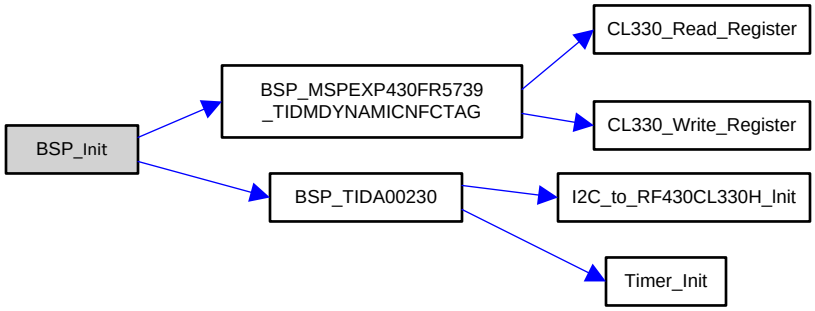


Figure 34. BSP_Init Call Graph

Here is the caller graph for this function:

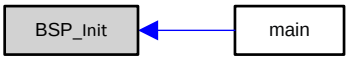


Figure 35. BSP_Init Caller Graph

void BSP_MSPEXP430FR5739_TIDMDYNAMICNFCTAG (void)

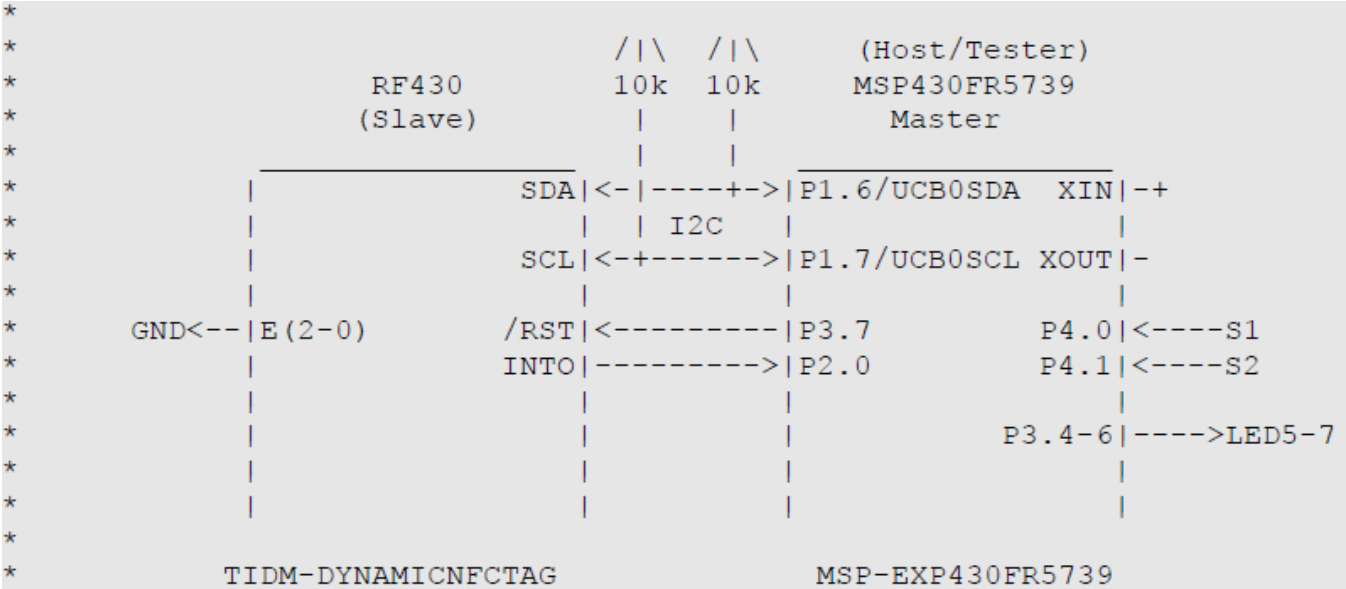


Figure 36. Overview of Hardware Configuration for Software Configuration MSPEXP430FR5739_TIDMDYNAMICNFCTAG

- Definition at line 47 of file BSP.c

Here is the call graph for this function:

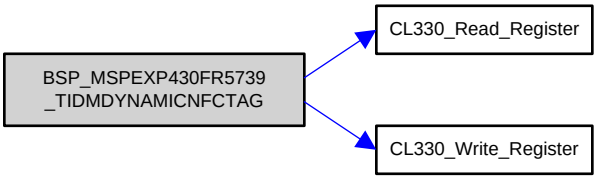


Figure 37. BSP_MSPEXP430FR5739_TIDMDYNAMICNFCTAG Call Graph

Here is the caller graph for this function:

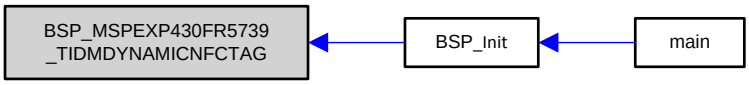


Figure 38. BSP_MSPEXP430FR5739_TIDMDYNAMICNFCTAG Caller Graph

void BSP_TIDA00230 (void)

- Provides all the hardware initialization for the TIDA-00230 board

```
*
*                   NFC chip      /|\  /|\         Host
*                                4.75k  4.75k
*                  (I2C Slave)     |    |         I2C Master for RF430, I2C Slave for Mother-
Board / Legacy systems
*                                  |    |
*             |               SDA|<-|----+->|P1.6/UCB0SDA      |
*             |                   |  | I2C   |                  |
*             |               SCL|<-+------>|P1.7/UCB0SCL      |
*             |                   |          |                  |
*      GND<--|E(2-0)          /RST|<---------|P3.7          P4.0|<----S1
*             |               INTO|--------->|P2.2          P4.1|<----S2
*             |                   |          |                  |
*             |                   |          |              P4.5|---->LED D3
*             |                   |          |              P4.6|---->LED D4
*             |                   |          |                  |
*
*           RF430CL330H                      MSP430FR5969IRGZ
```

Figure 39. TIDA-00230 Hardware Configuration

- DRIVE CS Low to indicate I2C Mode — should be pulled low within 1 ms of boot, so pull-it low before sending the reset to the RF430CL330H
- P4.4 set to '0'
- Reset RF430 to secure the sampling for CS low is done properly (we just set CS low above)
- Definition at line 171 of file BSP.c

Here is the call graph for this function:

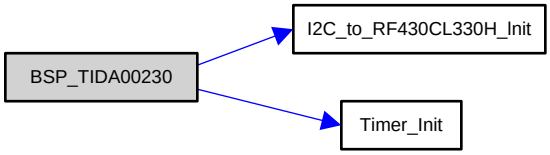


Figure 40. BSP_TIDA00230 Call Graph

Here is the caller graph for this function:

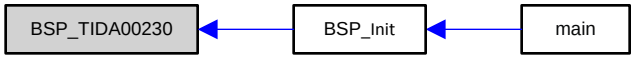


Figure 41. BSP_TIDA00230 Caller Graph

void Timer_Init ()

- Definition at line 233 of file BSP.c

Here is the caller graph for this function:

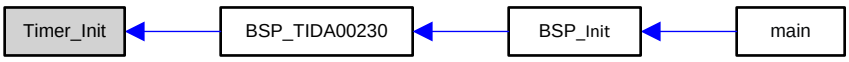


Figure 42. Timer_Init Caller Graph

4.2.3.3 [Installation folder]/src/main.c File Reference

```
#include "msp430.h"
#include "RF430CL330.h"
#include "MSP430_NDEF.h"
#include "BSP.h"
#include "stdlib.h"
#include "string.h"
```

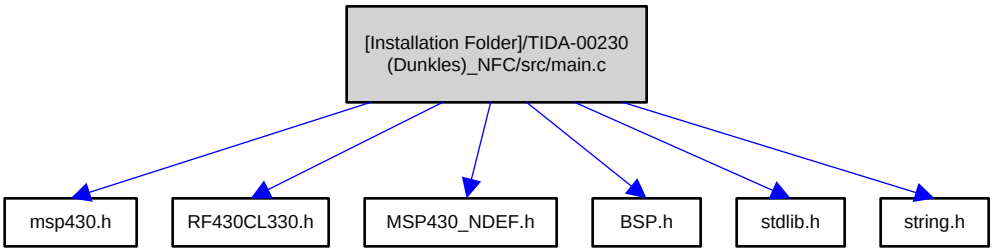


Figure 43. main.c Dependency Graph

Functions

- void main (void)
  flag indicating I2C transaction from master system was received (logging or request for information)
- __interrupt void PORT2_ISR (void)
- __interrupt void USCIB0_ISR (void)

Variables

- unsigned char func_retrim_osc []
- unsigned char test_data [] = {0x01, 0x23, 0x45, 0x67, 0x89, 0xAB, 0xCD, 0xEF}
- unsigned char CRC_Data [] = {1,2,3,4,5,6,7,8,9}
- unsigned char Cmd = 0
- unsigned char read_complete = 0
- unsigned char rx_byte_count = 0
- unsigned char tx_byte_count = 0
- unsigned int Results [11] = {0,0,0,0,0,0,0,0,0,0,0}
- unsigned char into_fired = 0
- unsigned char i2c_master_fired = 0
  flag indicating interrupt from RF430CL330H was received

Function Documentation

void main (void)

- Flag indicating I2C transaction from master system was received (logging or request for information)
- Enables NFC read or write notifications
- Enables CL330 RF
- Intrinsic functions to go to low power mode and enable interrupts. Here we are waiting for an RF read or write
- If flag is set, the user received an interrupt from RF430CL330H, which needs to be serviced
- Serviced INTO, clear the flag
- Definition at line 94 of file main.c

Here is the call graph for this function:

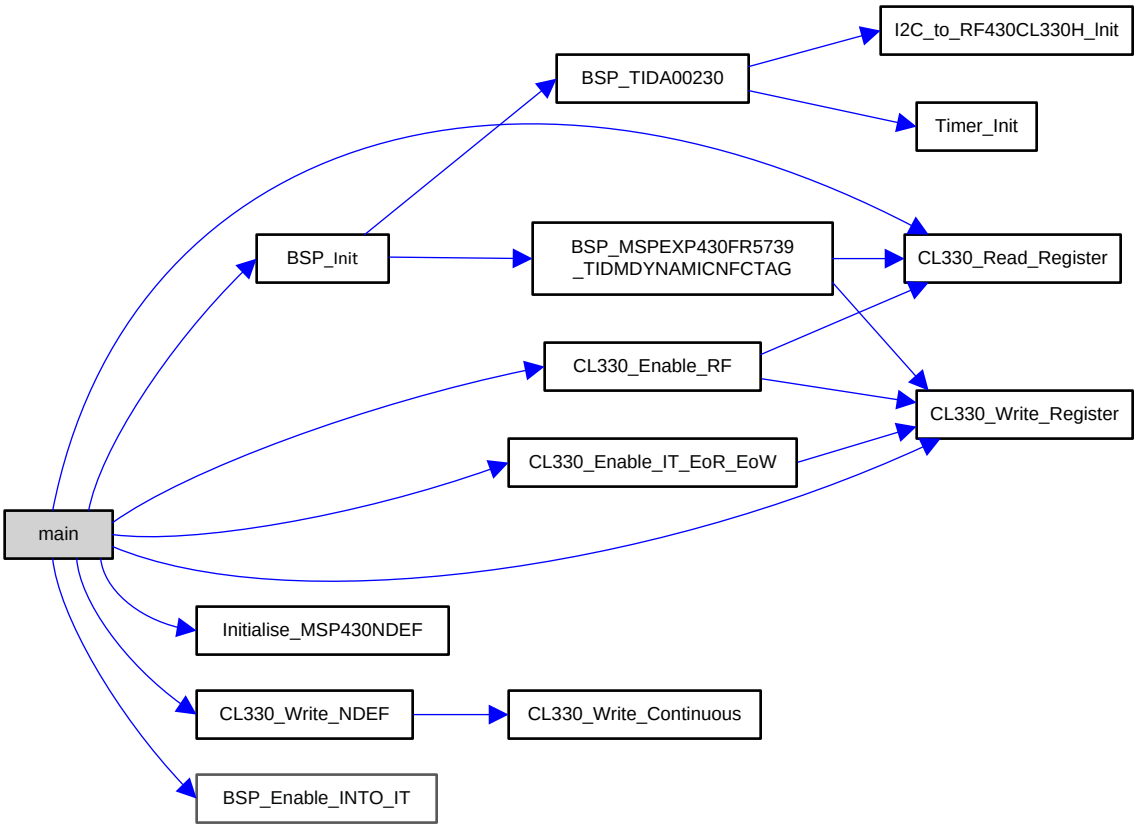


Figure 44. main Call Graph

__interrupt void PORT2_ISR (void)

- Updates into flag to indicate the user received an interrupt from RF430CL330H
- Intrisinc function to wake up MSP430 to handle INTO (that is, resume execution in main), otherwise MSP430 would go back to LPM3 when exiting ISR
- Definition at line 194 of file main.c

__interrupt void USCIB0_ISR (void)

- Intrisinc function to wake up MSP430 to handle INTO (that is, resume execution in main), otherwise MSP430 would go back to LPM3 when exiting ISR
- Definition at line 224 of file main.c.

Variable Documentation

unsigned char Cmd = 0

- Definition at line 76 of file main.c

unsigned char CRC_Data[] = {1,2,3,4,5,6,7,8,9}

- Definition at line 74 of file main.c

unsigned char func_retrim_osc[]

- Initial value:= {
  0xB2, 0x40, 0x11, 0x96, 0x10, 0x01,
  0xB2, 0x40, 0x60, 0x03, 0x18, 0x01,
  0x30, 0x41
  }
- Definition at line 66 of file main.c

unsigned char i2c_master_fired = 0

- Flag indicating interrupt from RF430CL330H was received
- Definition at line 92 of file main.c

unsigned char into_fired = 0

- Definition at line 91 of file main.c

unsigned char read_complete = 0

- Definition at line 77 of file main.c

unsigned int Results[11] = {0,0,0,0,0,0,0,0,0,0,0}

- Definition at line 80 of file main.c

unsigned char rx_byte_count = 0

- Definition at line 78 of file main.c

unsigned char test_data[] = {0x01, 0x23, 0x45, 0x67, 0x89, 0xAB, 0xCD, 0xEF}

- Definition at line 72 of file main.c

unsigned char tx_byte_count = 0

- Definition at line 79 of file main.c

4.2.3.4 [Installation folder]/src/MSP430_logger.c File Reference

NOTE: This section is intentionally left blank.

4.2.3.5 [Installation folder]/src/MSP430_NDEF.c File Reference

```
#include "MSP430_NDEF.h"
#include "string.h"
```

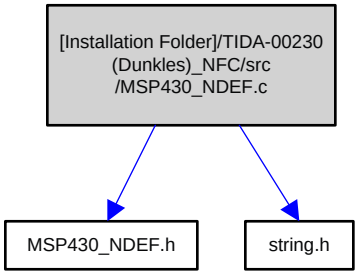


Figure 45. MSP430_NDEF.c Dependency Graph

Functions

unsigned int Initialise_MSP430NDEF (MSP430_NDEF_image *MSP_NDEF, char *NDEFPayload)

Function Documentation

unsigned int Initialise_MSP430NDEF (MSP430_NDEF_image * MSP_NDEF, char * NDEFPayload)

- : merge NDEF PAyload in NDEFFILE
- NDEF Application Name = D2_7600_0085_0101h
- File ID = E103h
- CCLEN
- Mapping version 2.0
- MLe (49 bytes); Maximum R-APDU data size
- MLc (52 bytes); Maximum C-APDU data size
- Tag, File Control TLV (4 = NDEF file)
- NDEF File Identifier
- Max NDEF size (3037 bytes of useable memory)
- NDEF file read access condition, read access without any security
- : remove temp variable
- : make this more robust if size of NDEFPayload is longer 255 characters
- Record Header TNF = 0x01 (Well Known Type). SR=1, MB=1, ME=1, IL=0, TypeLength = 1
- : same as above here
- PayloadType:0x54 = text
- write the NDEF Application name
- Capability Container ID
- need to have header size generated dynamically but today it is static and always 33 = 0x21
- Definition at line 45 of file MSP430_NDEF.c.

Here is the caller graph for this function:

Figure 46. Initialise_MSP430NDEF Caller Graph

4.2.3.6 [Installation folder]/src/MSP430_NDEF.h File Reference

This graph shows which files directly or indirectly include this file:

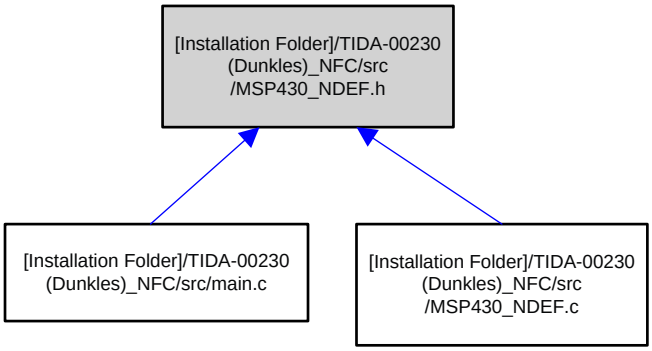


Figure 47. MSP430_NDEF.h Dependency Graph

Data Structures

- struct MSP430_NDEF_image
- struct NDEF_File_Headers

Macros

- #define RF430_NDEF_IMAGE
- #define RF430_NDEF_IMAGE2
- #define RF430_NDEF_IMAGE3
- #define RF430_NDEF_IMAGE_E2E

Typedefs

- typedef struct MSP430_NDEF_image MSP430_NDEF_image
- typedef struct NDEF_File_Headers NDEF_File_Headers

Functions

- unsigned int Initialise_MSP430NDEF (MSP430_NDEF_image *MSP_NDEF, char *NDEFpayload)

Variables

- MSP430_NDEF_image * NDEFFRAM
- MSP430_NDEF_image * NDEFFRAM_END

Macro Definition Documentation

#define RF430_NDEF_IMAGE

Value:{ \
NDEF Tag Application Name
\ 0xD2, 0x76, 0x00, 0x00, 0x85, 0x01, 0x01,

\ \ Capability Container ID \
0xE1, 0x03, \
0x00, 0x0F, CCLEN
\ 0x20, Mapping version 2.0
\ 0x00, 0xF9, MLe (49 bytes); Maximum R-APDU data size
\ 0x00, 0xF6, MLc (52 bytes); Maximum C-APDU data size
\ 0x04, Tag, File Control TLV (4 = NDEF file)
\ 0x06, Length, File Control TLV (6 = 6 bytes of data for this tag)
\ 0xE1, 0x04, File Identifier
\ 0x0B, 0xDF, Max NDEF size (3037 bytes of useable memory)
\ 0x00, NDEF file read access condition, read access without any security
\ 0x00, NDEF file write access condition; write access without any security

\ \ NDEF File ID
\ 0xE1, 0x04, \ \

NDEF File for Hello World (48 bytes total length)
\ 0x00, 0x18, NLEN; NDEF length (3 byte long message) was 0x14
0xD1, 0x01, 0x10,
\ 0x54, T = text
\ 0x02,
\ 0x65, 0x6E, 'e', 'n', \ \
/* 'Hello, Matt!!' NDEF data; Empty NDEF message, length should match NLEN*/
\ 0x48, 0x65, 0x6c, 0x6c, 0x6f, 0x2c, 0x20, 0x4d, 0x61, 0x74, 0x74, 0x21, 0x21, 0x21 \
}

- Definition at line 45 of file MSP430_NDEF.h

#define RF430_NDEF_IMAGE2

Value:{ \
NDEF Tag Application Name
\ 0xD2, 0x76, 0x00, 0x00, 0x85, 0x01, 0x01,

\ \ /Capability Container ID*/
\ 0xE1, 0x03,
\ 0x00, 0x0F, CCLEN
\ 0x20, Mapping version 2.0
\ 0x00, 0xF9, MLe (49 bytes); Maximum R-APDU data size
\ 0x00, 0xF6, MLc (52 bytes); Maximum C-APDU data size
\ 0x04, Tag, File Control TLV (4 = NDEF file)
\ 0x06, Length, File Control TLV (6 = 6 bytes of data for this tag)
\ 0xE1, 0x04, File Identifier
\ 0x0B, 0xDF, Max NDEF size (3037 bytes of useable memory)
\ 0x00, NDEF file read access condition, read access without any security
\ 0x00, NDEF file write access condition; write access without any security

\ \ /* NDEF File ID */
\ 0xE1, 0x04,

\ \ NDEF File for Hello World (48 bytes total length)
\ 0x00, 0x14, NLEN; NDEF length (3 byte long message)
\ 0xD1, 0x01, 0x10,
\ 0x54, T = text
\ 0x02,
\ 0x65, 0x6E, 'e', 'n',

\ \ /* 'Hello, world!' NDEF data; Empty NDEF message, length should match NLEN*/ \ 0x48, 0x65, 0x6c, 0x6c, 0x6f, 0x2c, 0x20, 0x54, 0x49, 0x44, 0x41, 0x32, 0x34 \
}

- Definition at line 76 of file MSP430_NDEF.h

#define RF430_NDEF_IMAGE3

- Definition at line 107 of file MSP430_NDEF.h

#define RF430_NDEF_IMAGE_E2E

Value:{ \
NDEF Tag Application Name
\ 0xD2, 0x76, 0x00, 0x00, 0x85, 0x01, 0x01,

\ \ Capability Container ID
\ 0xE1, 0x03,
\ 0x00, 0x0F, CCLEN
\ 0x20, Mapping version 2.0
\ 0x00, 0xF9, MLe (49 bytes); Maximum R-APDU data size
\ 0x00, 0xF6, MLc (52 bytes); Maximum C-APDU data size
\ 0x04, Tag, File Control TLV (4 = NDEF file)
0x06, /* Length, File Control TLV (6 = 6 bytes of data for this tag) */
\ 0xE1, 0x04, File Identifier
\ 0x0B, 0xDF, Max NDEF size (3037 bytes of useable memory)
\ 0x00, NDEF file read access condition, read access without any security
\ 0x00, NDEF file write access condition; write access without any security
\ /*end of Capability Container ID*/

\ \ NDEF File start
\ 0xE1, 0x04,
\ 0x00, 0x13,
\ 0xD1,
\ 0x01,
\ 0xf, \
0x54, \
0x02, \ 0x65, 0x6E, \
'h', 'a', 0x6c, 0x6c, 0x6f, 0x2c, 0x20, 0x4d, 0x61, 0x74, 0x74, 0x21 \
}

- Definition at line 142 of file MSP430_NDEF.h

Typedef Documentation

- typedef struct MSP430_NDEF_image MSP430_NDEF_image
- typedef struct NDEF_File_Headers NDEF_File_Headers

Function Documentation

unsigned int Initialise_MSP430NDEF (MSP430_NDEF_image * MSP_NDEF, char * NDEFpayload)

- : merge NDEF PAyload in NDEFFILE
- NDEF Application Name = D2_7600_0085_0101h
- File ID = E103h
- CCLEN
- Mapping version 2.0
- MLe (49 bytes); Maximum R-APDU data size
- MLc (52 bytes); Maximum C-APDU data size
- Tag, File Control TLV (4 = NDEF file)
- NDEF File Identifier Max NDEF size (3037 bytes of useable memory)
- NDEF file read access condition, read access without any security
- : remove temp variable
- : make this more robust if size of NDEFPayload is longer 255 characters
- Record Header TNF = 0x01 (Well Known Type). SR=1, MB=1, ME=1, IL=0, TypeLength = 1
- : same as above here
- PayloadType:0x54 = text
- write the NDEF Application name
- Capability Container ID
- need to have header size generated dynamically but today it is static and always 33 = 0x21
- Definition at line 45 of file MSP430_NDEF.c

Here is the caller graph for this function:

Figure 48. Initialise_MSP430NDEF Caller Graph

Variable Documentation

- MSP430_NDEF_image* NDEFFRAM
- MSP430_NDEF_image* NDEFFRAM_END

4.2.3.7 [Installation folder]/src/RF430CL330.c File Reference

```
#include "msp430.h"
#include "RF430CL330.h"
```

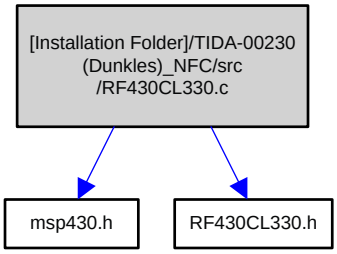


Figure 49. RF430CL330.c Dependency Graph

Functions

- unsigned int CL330_Read_Register (unsigned int reg_addr)
- unsigned int CL330_Read_Register_BIP8 (unsigned int reg_addr)
- void CL330_Read_Continuous (unsigned int reg_addr, unsigned char *read_data, unsigned int data_length)
- void CL330_Write_Register (unsigned int reg_addr, unsigned int value)
- void CL330_Write_Register_BIP8 (unsigned int reg_addr, unsigned int value)
- void CL330_Write_Continuous (unsigned int reg_addr, unsigned char *write_data, unsigned int data_length)
- void CL330_Write_NDEF (unsigned char *NDEF_Image, unsigned int len)
  Writes NDEF memory with Capability Container + NDEF message
- void CL330_Enable_IT_EoR_EoW ()
  Enable interrupts for End of Read and End of Write.
- void CL330_Enable_RF ()
  Sets the bit RF_ENABLE in CL330 register CONTROL_REG.

Variables

- unsigned char RxData [2] = {0,0}
- unsigned char TxData [2] = {0,0}
- unsigned char TxAddr [2] = {0,0}

4.2.3.7.1 Function Documentation

void CL330_Enable_IT_EoR_EoW ()

- Enables interrupts for End of Read and End of Write
- Write enables the IT, which are used (End of Read, End of Write from RF) need to rewrite to secure all IT handlers are added
- Definition at line 271 of file RF430CL330.c

Here is the call graph for this function:

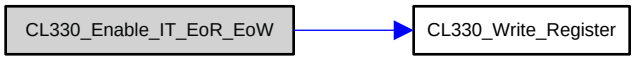


Figure 50. CL330_Enable_IT_EoR_EoW Call Graph

Here is the caller graph for this function:

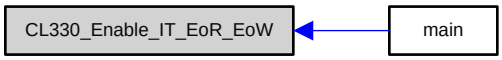


Figure 51. CL330_Enable_IT_EoR_EoW Caller Graph

void CL330_Enable_RF ()

- Sets the bit RF_ENABLE in CL330 register CONTROL_REG.
- Enables the RF interface, after this read or write to NDEF should only be done if RF is not busy and then calling CL330_Disable_RF.
- Disables the RF interface, should only be call if NDEF read or write are needed and after checking that RF is not busy.
- < shadow copy of the CL330 CONTROL_REG
- Definition at line 277 of file RF430CL330.c

Here is the call graph for this function:

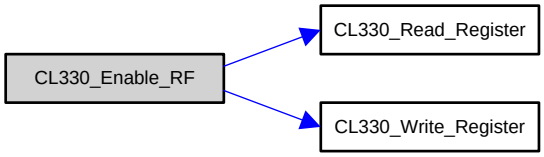


Figure 52. CL330_Enable_RF Call Graph

Here is the caller graph for this function:

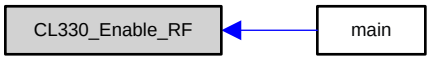


Figure 53. CL330_Enable_RF Caller Graph

void CL330_Read_Continuous (unsigned int reg_addr, unsigned char * read_data, unsigned int data_length)

- Definition at line 119 of file RF430CL330.c

unsigned int CL330_Read_Register (unsigned int reg_addr)

- Definition at line 49 of file RF430CL330.c

Here is the caller graph for this function:

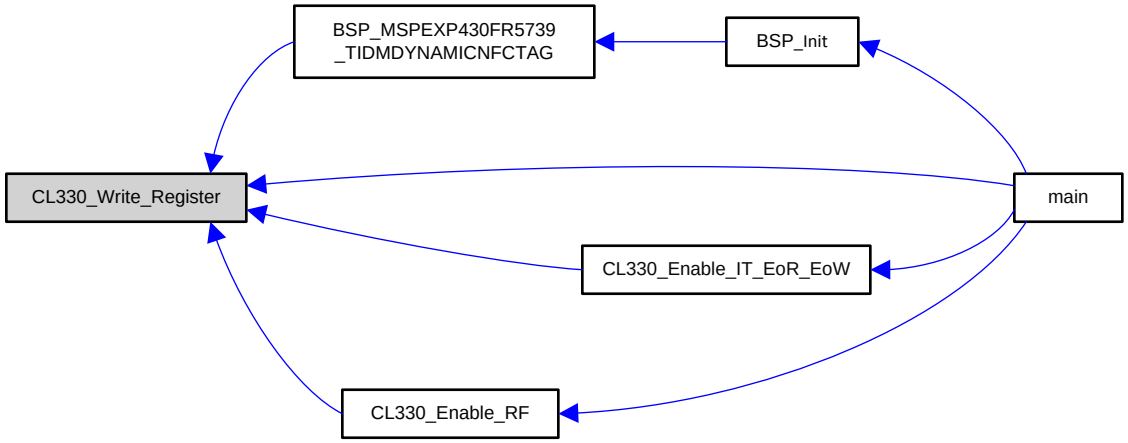


Figure 54. CL330_Write_Register Caller Graph

unsigned int CL330_Read_Register_BIP8 (unsigned int reg_addr)

- Definition at line 77 of file RF430CL330.c

void CL330_Write_Continuous (unsigned int reg_addr, unsigned char * write_data, unsigned int data_length)

- Definition at line 230 of file RF430CL330.c

Here is the caller graph for this function:

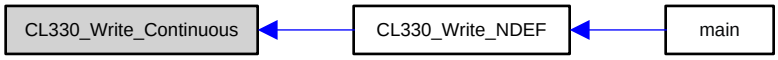


Figure 55. CL330_Write_Continuous Caller Graph

void CL330_Write_NDEF (unsigned char * NDEF_Image, unsigned int len)

- Writes NDEF memory with Capability Container + NDEF message
- Writes the NDEF_Application_Data with all the container information need to rewrite to make sure the container information is added afterwards. Remove the third parameter (length) to make it more generic
- Definition at line 264 of file RF430CL330.c

Here is the call graph for this function:

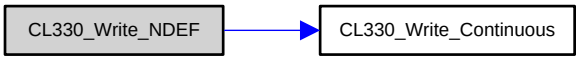


Figure 56. CL330_Write_NDEF Call Graph

Here is the caller graph for this function:

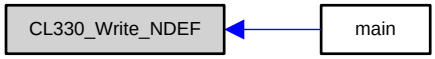


Figure 57. CL330_Write_NDEF

void CL330_Write_Register (unsigned int reg_addr, unsigned int value)

- Definition at line 157 of file RF430CL330.c

Here is the caller graph for this function:

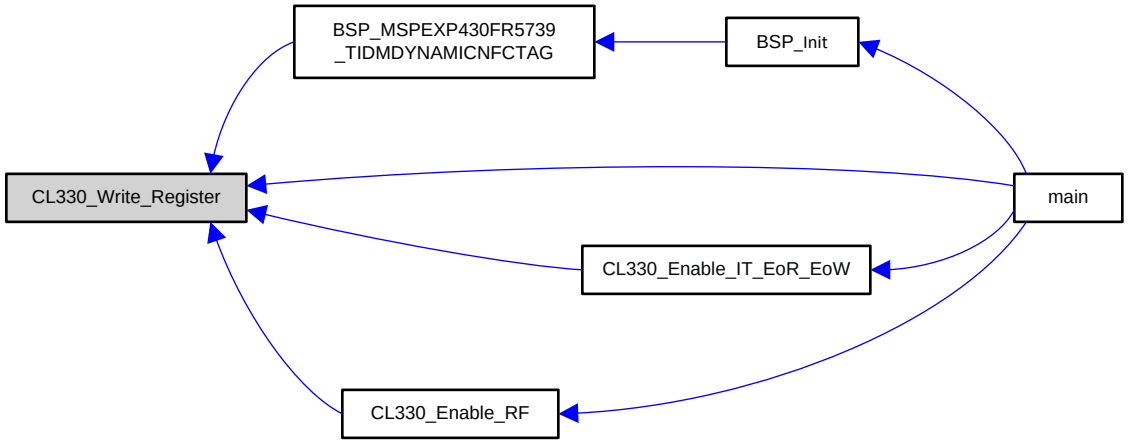


Figure 58. CL330_Write_Register Caller Graph

void CL330_Write_Register_BIP8 (unsigned int reg_addr, unsigned int value)

- Definition at line 188 of file RF430CL330.c

Variable Documentation

unsigned char RxData[2] = {0,0}

- Definition at line 44 of file RF430CL330.c

unsigned char TxAddr[2] = {0,0}

- Definition at line 46 of file RF430CL330.c

unsigned char TxData[2] = {0,0}

- Definition at line 45 of file RF430CL330.c

4.2.3.8 [Installation folder]/src/RF430CL330.h File Reference

This graph shows which files directly or indirectly include this file:

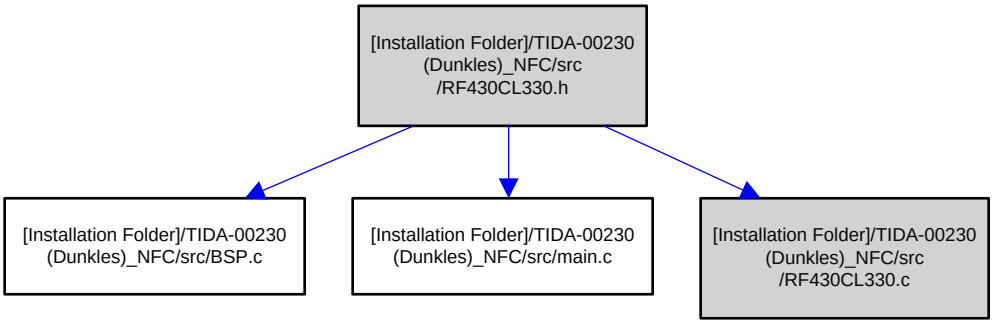


Figure 59. RF430CL330.h Dependency Graph

Macros

- #define CONTROL_REG 0xFFFE
- #define STATUS_REG 0xFFFC
- #define INT_ENABLE_REG 0xFFFA
- #define INT_FLAG_REG 0xFFF8
- #define CRC_RESULT_REG 0xFFF6
- #define CRC_LENGTH_REG 0xFFF4
- #define CRC_START_ADDR_REG 0xFFF2
- #define COMM_WD_CTRL_REG 0xFFF0
- #define VERSION_REG 0xFFEE
- #define TEST_FUNCTION_REG 0xFFE2
- #define TEST_MODE_REG 0xFFE0
- #define SW_RESET BIT0
- #define RF_ENABLE BIT1
- #define INT_ENABLE BIT2
- #define INTO_HIGH BIT3
- #define INTO_DRIVE BIT4
- #define BIP8_ENABLE BIT5
- #define STANDBY_ENABLE BIT6
- #define TEST430_ENABLE BIT7
- #define READY BIT0
- #define CRC_ACTIVE BIT1
- #define RF_BUSY BIT2
- #define EOR_INT_ENABLE BIT1

- #define EOW_INT_ENABLE BIT2
- #define CRC_INT_ENABLE BIT3
- #define BIP8_ERROR_INT_ENABLE BIT4
- #define NDEF_ERROR_INT_ENABLE BIT5
- #define GENERIC_ERROR_INT_ENABLE BIT7
- #define EOR_INT_FLAG BIT1
- #define EOW_INT_FLAG BIT2
- #define CRC_INT_FLAG BIT3
- #define BIP8_ERROR_INT_FLAG BIT4
- #define NDEF_ERROR_INT_FLAG BIT5
- #define GENERIC_ERROR_INT_FLAG BIT7
- #define WD_ENABLE BIT0
- #define TIMEOUT_PERIOD_2_SEC 0
- #define TIMEOUT_PERIOD_32_SEC BIT1
- #define TIMEOUT_PERIOD_8_5_MIN BIT2
- #define TIMEOUT_PERIOD_MASK BIT1 + BIT2 + BIT3
- #define TEST_MODE_KEY 0x004E

Functions

- unsigned int CL330_Read_Register (unsigned int reg_addr)
- unsigned int CL330_Read_Register_BIP8 (unsigned int reg_addr)
- void CL330_Read_Continuous (unsigned int reg_addr, unsigned char *read_data, unsigned int data_length)
- void CL330_Write_Register (unsigned int reg_addr, unsigned int value)
- void CL330_Write_Continuous (unsigned int reg_addr, unsigned char *write_data, unsigned int data_length)
- void CL330_Write_Register_BIP8 (unsigned int reg_addr, unsigned int value)
- void CL330_Write_NDEF (unsigned char *NDEF_Image, unsigned int len)
  Writes NDEF memory with Capability Container + NDEF message
- void CL330_Enable_IT_EoR_EoW ()
  Enables interrupts for End of Read and End of Write
- void CL330_Enable_RF ()
  Enables the RF interface, after this read or write to NDEF should only be done if RF is not busy and then calling CL330_Disable_RF.

Macro Definition Documentation

#define BIP8_ENABLE BIT5

- Definition at line 121 of file RF430CL330.h

#define BIP8_ERROR_INT_ENABLE BIT4

- Definition at line 134 of file RF430CL330.h

#define BIP8_ERROR_INT_FLAG BIT4

- Definition at line 142 of file RF430CL330.h

#define COMM_WD_CTRL_REG 0xFFF0

- Definition at line 109 of file RF430CL330.h

#define CONTROL_REG 0xFFFE

- Definition at line 102 of file RF430CL330.h

#define CRC_ACTIVE BIT1

- Definition at line 127 of file RF430CL330.h

#define CRC_INT_ENABLE BIT3

- Definition at line 133 of file RF430CL330.h

#define CRC_INT_FLAG BIT3

- Definition at line 141 of file RF430CL330.h

#define CRC_LENGTH_REG 0xFFF4

- Definition at line 107 of file RF430CL330.h

#define CRC_RESULT_REG 0xFFF6

- Definition at line 106 of file RF430CL330.h

#define CRC_START_ADDR_REG 0xFFF2

- Definition at line 108 of file RF430CL330.h

#define EOR_INT_ENABLE BIT1

- Definition at line 131 of file RF430CL330.h

#define EOR_INT_FLAG BIT1

- Definition at line 139 of file RF430CL330.h

#define EOW_INT_ENABLE BIT2

- Definition at line 132 of file RF430CL330.h

#define EOW_INT_FLAG BIT2

- Definition at line 140 of file RF430CL330.h

#define GENERIC_ERROR_INT_ENABLE BIT7

- Definition at line 136 of file RF430CL330.h

#define GENERIC_ERROR_INT_FLAG BIT7

- Definition at line 144 of file RF430CL330.h

#define INT_ENABLE BIT2

- Definition at line 118 of file RF430CL330.h

#define INT_ENABLE_REG 0xFFFA

- Definition at line 104 of file RF430CL330.h

#define INT_FLAG_REG 0xFFF8

- Definition at line 105 of file RF430CL330.h

#define INTO_DRIVE BIT4

- Definition at line 120 of file RF430CL330.h

#define INTO_HIGH BIT3

- Definition at line 119 of file RF430CL330.h

#define NDEF_ERROR_INT_ENABLE BIT5

- Definition at line 135 of file RF430CL330.h

#define NDEF_ERROR_INT_FLAG BIT5

- Definition at line 143 of file RF430CL330.h

#define READY BIT0

- Definition at line 126 of file RF430CL330.h

#define RF_BUSY BIT2

- Definition at line 128 of file RF430CL330.h

#define RF_ENABLE BIT1

- Definition at line 117 of file RF430CL330.h

#define STANDBY_ENABLE BIT6

- Definition at line 122 of file RF430CL330.h

#define STATUS_REG 0xFFFC

- Definition at line 103 of file RF430CL330.h

#define SW_RESET BIT0

- Definition at line 116 of file RF430CL330.h

#define TEST430_ENABLE BIT7

- Definition at line 123 of file RF430CL330.h

#define TEST_FUNCTION_REG 0xFFE2

- Definition at line 111 of file RF430CL330.h

#define TEST_MODE_KEY 0x004E

- Definition at line 153 of file RF430CL330.h

#define TEST_MODE_REG 0xFFE0

- Definition at line 112 of file RF430CL330.h

#define TIMEOUT_PERIOD_2_SEC 0

- Definition at line 148 of file RF430CL330.h

#define TIMEOUT_PERIOD_32_SEC BIT1

- Definition at line 149 of file RF430CL330.h

#define TIMEOUT_PERIOD_8_5_MIN BIT2

- Definition at line 150 of file RF430CL330.h

#define TIMEOUT_PERIOD_MASK BIT1 + BIT2 + BIT3

- Definition at line 151 of file RF430CL330.h

#define VERSION_REG 0xFFEE

- Definition at line 110 of file RF430CL330.h

#define WD_ENABLE BIT0

- Definition at line 147 of file RF430CL330.h

Function Documentation

void CL330_Enable_IT_EoR_EoW ()

- Enable interrupts for End of Read and End of Write.
- Write enables the IT, which are used (End of Read, End of Write from RF) need to rewrite to secure all IT handlers are added
- Definition at line 271 of file RF430CL330.c

Here is the call graph for this function:

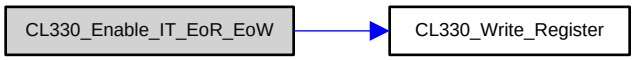


Figure 60. CL330_Enable_IT_EoR_EoW Call Graph

Here is the caller graph for this function:

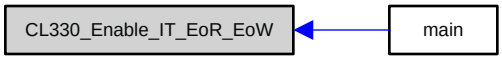


Figure 61. CL330_Enable_IT_EoR_EoW Caller Graph

void CL330_Enable_RF ()

- Enables the RF interface, after this read or write to NDEF should only be done if RF is not busy and then calling CL330_Disable_RF
- Disables the RF interface, should only be call if NDEF read or write are needed and after checking that RF is not busy
- Enables the RF interface, after this read or write to NDEF should only be done if RF is not busy and then calling CL330_Disable_RF
- Disables the RF interface, should only be call if NDEF read or write are needed and after checking that RF is not busy
- < shadow copy of the CL330 CONTROL_REG
- Definition at line 277 of file RF430CL330.c

Here is the call graph for this function:

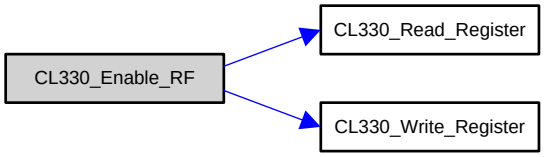


Figure 62. CL330_Enable_RF Call Graph

Here is the caller graph for this function:

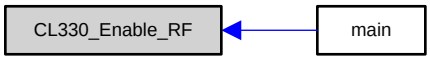


Figure 63. CL330_Enable_RF Caller Graph

void CL330_Read_Continuous (unsigned int reg_addr, unsigned char * read_data, unsigned int data_length)

- Definition at line 119 of file RF430CL330.c

unsigned int CL330_Read_Register (unsigned int reg_addr)

- Definition at line 49 of file RF430CL330.c

Here is the caller graph for this function:

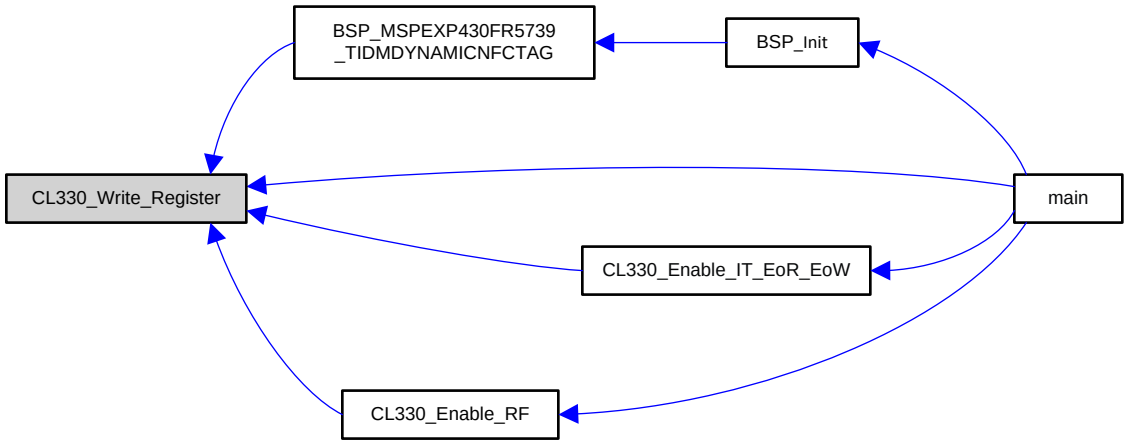


Figure 64. CL330_Write_Register Caller Graph

unsigned int CL330_Read_Register_BIP8 (unsigned int reg_addr)

- Definition at line 77 of file RF430CL330.c

void CL330_Write_Continuous (unsigned int reg_addr, unsigned char * write_data, unsigned int data_length)

- Definition at line 230 of file RF430CL330.c

Here is the caller graph for this function:

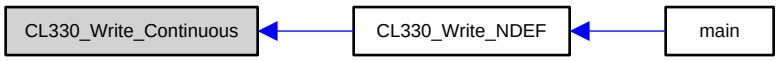


Figure 65. CL330_Write_Continuous Caller Graph

void CL330_Write_NDEF (unsigned char * NDEF_Image, unsigned int len)

- Writes NDEF memory with Capability Container + NDEF message
- Parameters: NDEF_Image = Binary NDEF file content
- Writes the NDEF_Application_Data with all the container information need to rewrite to make sure the container information is added afterwards. Remove the third parameter (length) to make it more generic
- Definition at line 264 of file RF430CL330.c

Here is the call graph for this function:

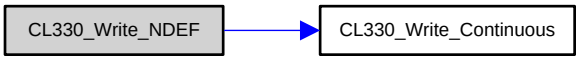


Figure 66. CL330_Write_NDEF Call Graph

Here is the caller graph for this function:

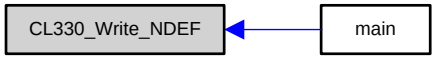


Figure 67. CL330_Write_NDEF

void CL330_Write_Register (unsigned int reg_addr, unsigned int value)

- Definition at line 157 of file RF430CL330.c

Here is the caller graph for this function:

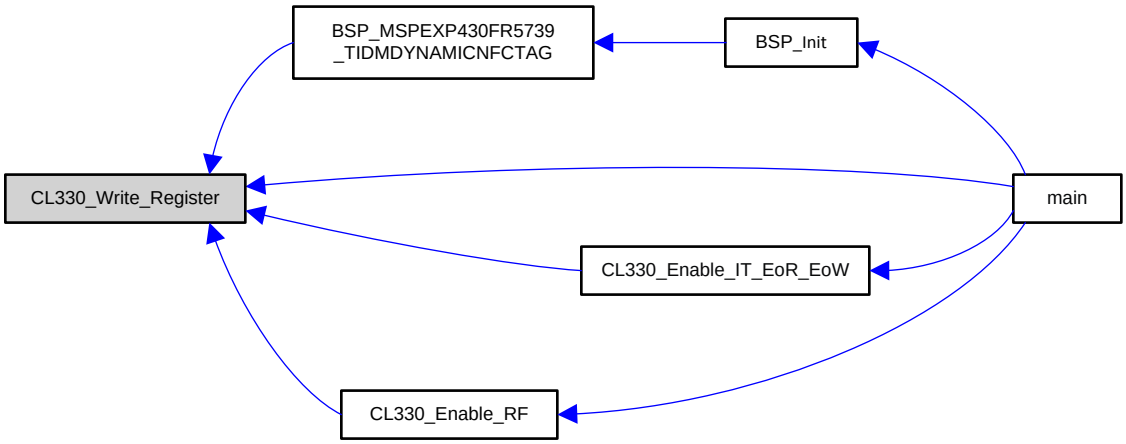


Figure 68. CL330_Write_Register Caller Graph

void CL330_Write_Register_BIP8 (unsigned int reg_addr, unsigned int value)

- Definition at line 188 of file RF430CL330.c

5 Test Setup

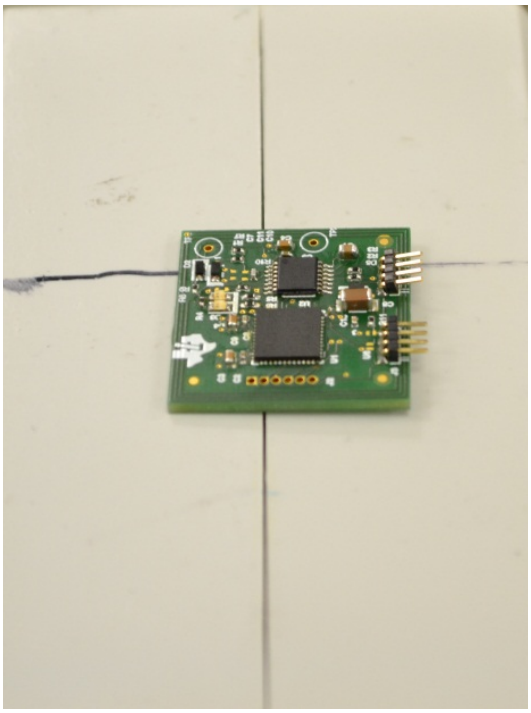


Figure 69. Top View of PCB Antenna Design on Reference Antenna

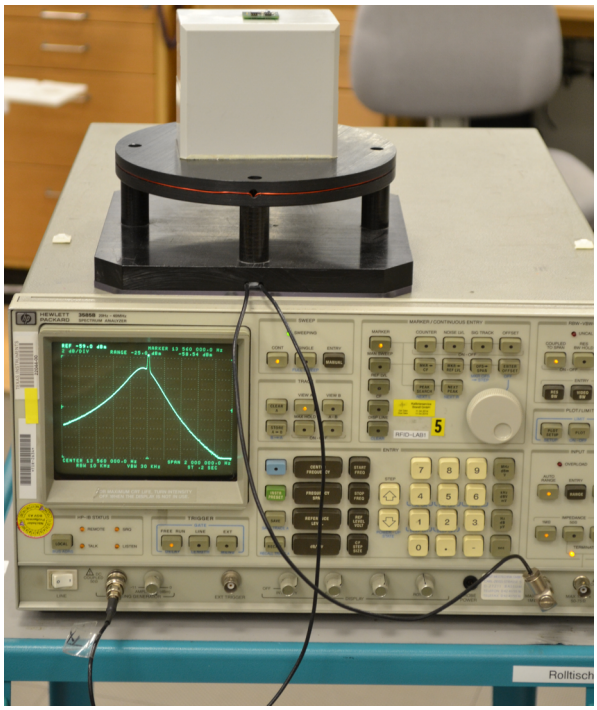


Figure 70. Front View of Design on Test Antenna Connected to Spectrum Analyzer

As per NFC specification, the resonance frequency of the device should be in the 13.7-MHz range. To ensure a close matching in resonance frequency, once the design is manufactured, the LC resonant circuit is tuned to offer maximum resonance at the desired frequency.

There are different ways to measure the resonance capacitance and Q of this design based on RF430CL330H. ISO/IEC 10373-6 describes a standardized concept.

For a simple frequency and Q-factor measurement, a spectrum analyzer with tracking generator is required. As seen in Figure 70, this test uses an Agilent 4402B (though any other similar calibrated spectrum analyzer can be used).

Follow this procedure:

1. Connect the fixture to the analyzer.
2. Place a reference unit on top of the fixture (as shown in Figure 69 and Figure 70).
3. Enable the tracking generator output, center to the expected inlay frequency (13.56 MHz) with a span of 2 MHz.
4. See the resonance curve at about a –60-dBm reference level with a vertical scale of 1 dB/div.
5. Read the inlay frequency at maximum.
6. View the bandwidth through the peak search menu.
7. Calculate the quality factor by dividing the resonant frequency by the measured bandwidth.

For a practical approach, use a tunable capacitor during the design of the system and replace it with a fixed value for the final product. The capacitor value in the schematics is the one given after testing and characterization as described in this section and Section 6.

The recommended operating resonance frequency (fres) is approximately 13.7 MHz for optimal performance. Resonance frequencies greater than 13.7 MHz lead reduce performance. Ensure that the resonant frequency, including all the tolerances, stay above 13.56 MHz.

Considering the classical resonant LC circuit, the resonance frequency (fres) is given by:

$$fres = \frac{1}{2 \times \pi \times \sqrt{L \times C}} \quad (2)$$

However, the capacitance C is the sum of the RF430CL330 (Cint = 35 pF) and the PCB one.

Figure 71 shows the inductance and capacitor values to generate resonance at 13.7 MHz.

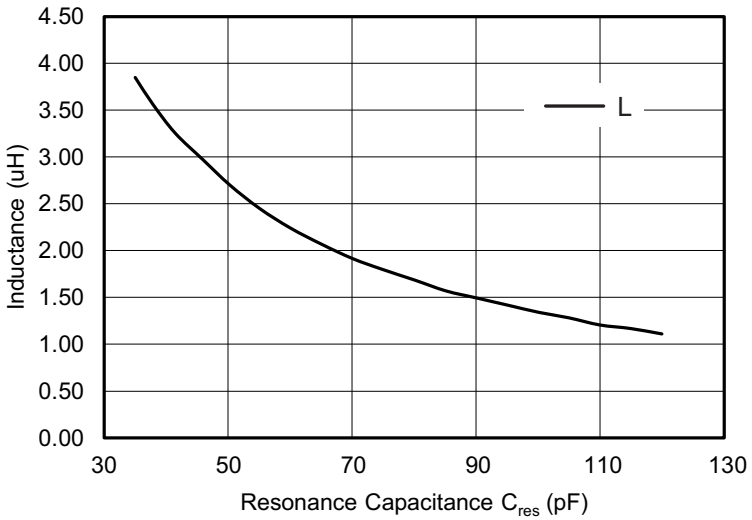


Figure 71. L versus Cres at fres = 13.7MHz

6 Test Results

6.1 Resonance Tuning using Spectrum Analyzer

Following the procedure given in Section 5, select the following components:

- C10: 22 pF
- C11: DNP

6.1.1 Power Generation

Reader behavior and the amount of energy varies from device to device. Figure 72 and Figure 73 show typical energy that can be extracted from 2 cm for two different devices.

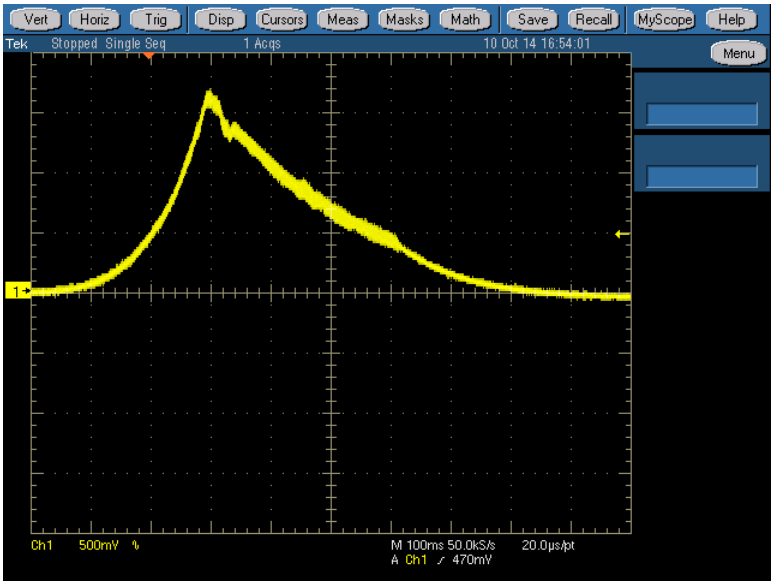


Figure 72. NFC Reader

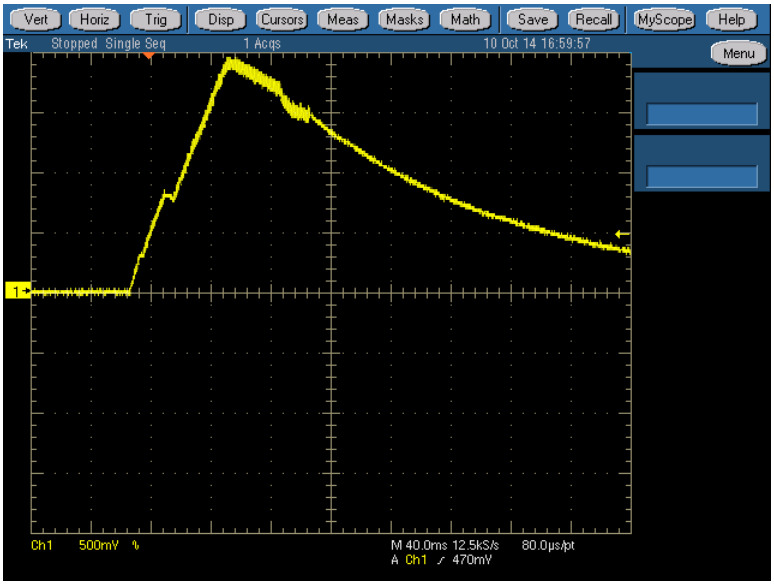


Figure 73. Samsung™ Galaxy S3 LTE

6.2 RF430CL330H: CETECOM Test Results Summary

TEST RESULT SUMMARY

Project No.: 1-6851/13-01-02

Test Laboratory

CETECOM ICT Services GmbH
Untertürkheimer Straße 6 – 10
66117 Saarbrücken/Germany
Phone: + 49 681 5 98 - 0
Fax: + 49 681 5 98 - 9075
Internet: http://www.cetecom.com
e-mail: ict@cetecom.com

Applicant

Texas Instruments Deutschland GmbH
Haggertystrasse 1
85356 Freising
Germany

Manufacturer

Texas Instruments Deutschland GmbH
Haggertystrasse 1
85356 Freising
Germany

Test Standards

NFC Forum Test Cases for DP, v1.0.04	NFC Forum Test Cases for Digital Protocol – 1st Wave of Certification, Version 1.0.04
NFC Forum TCCL, v1.4	NFC Forum Test Case Category List, Version 1.4

Test Item

Kind of test item:	**NFC Forum Tag (Type 4B)**
Model name:	**RF430CL330**
S/N serial number:	1210013410
HW hardware status:	Rev. D / V1.3

Summary of Measurement Results

☒	**No deviations from the technical specifications were ascertained**
☐	There were deviations from the technical specifications ascertained

Note: The test amount was limited to Listen Mode tests for type NFC-B as the DUT doesn't support further modes of operation.

Figure 74. CETECOM Test Results Summary

7 Terminology

FOTA— Firmware upgrade Over The Air, umbrella terminology for over-the-air service provisioning (OTASP), over-the-air provisioning (OTAP) or over-the-air parameter administration (OTAPA). The "over-the-air" aspect referring to the use of wireless instead of requiring the user to connect the device to a host through a serial interface (HART, IO-Link, RS-232, RS-485, and so on)

NFC— Near Field Communication

NDEF— NFC Data Exchange Format

NFC initiator (master)— Generator of the RF field and is the starter of the NFCIP-1 communication

NFC target (slave)— Responds to initiator command either using load modulation scheme (passive mode) or using modulation of self-generated RF field (active mode)

NFC active communication mode (peer-to-peer)— Both devices (initiator and target) generate their own RF field for the communication

NFC passive communication mode— Only the initiator generates the RF field for the communication. The target responds to an initiator command by applying a load modulation on the RF field

RFID— Radio-Frequency Identification

8 Design Files

8.1 Schematics

The following schematics show the default variants. Other variants have different sets of DNI components.

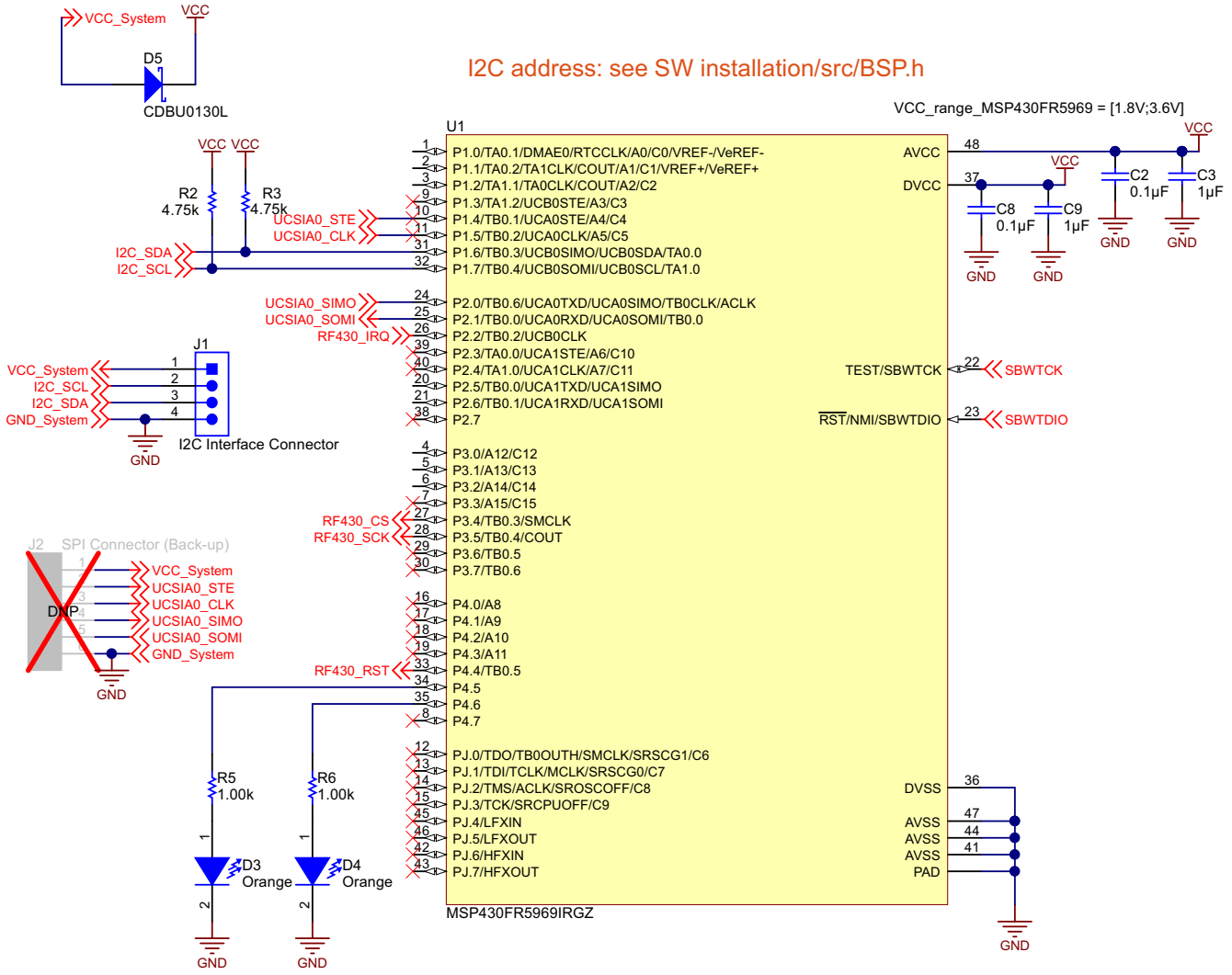


Figure 75. MSP430 Microcontroller

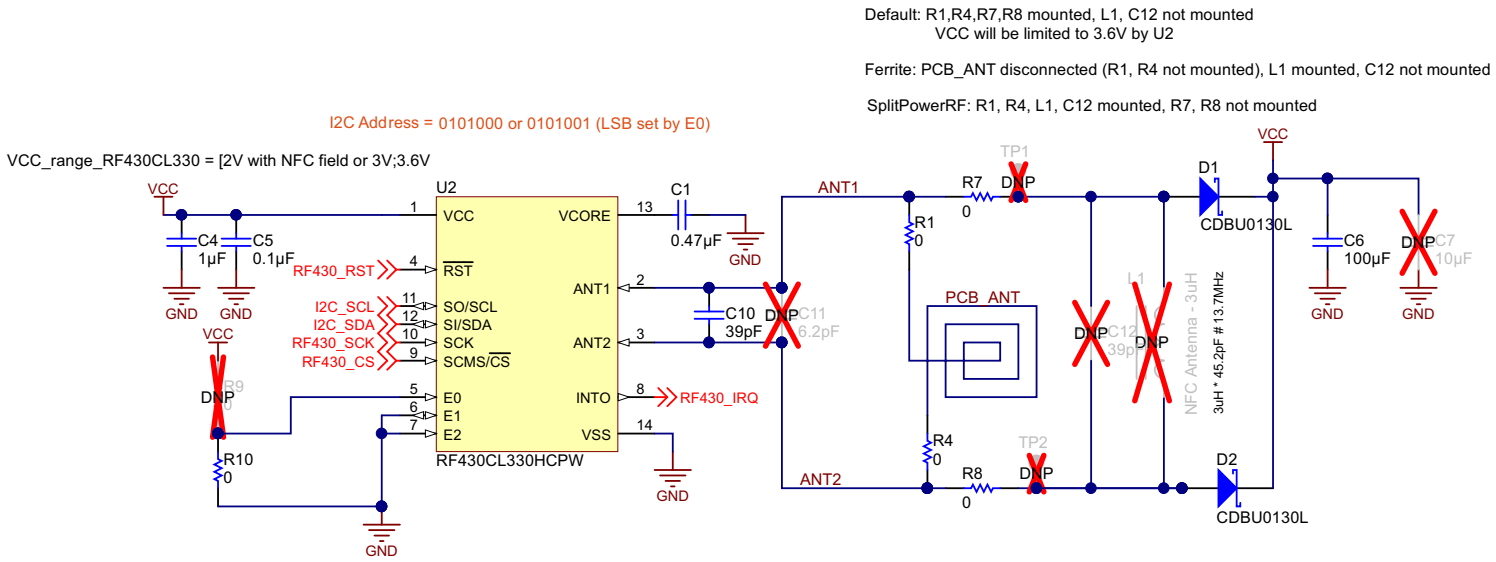


Figure 76. RF430 Dynamic NFC Transponder

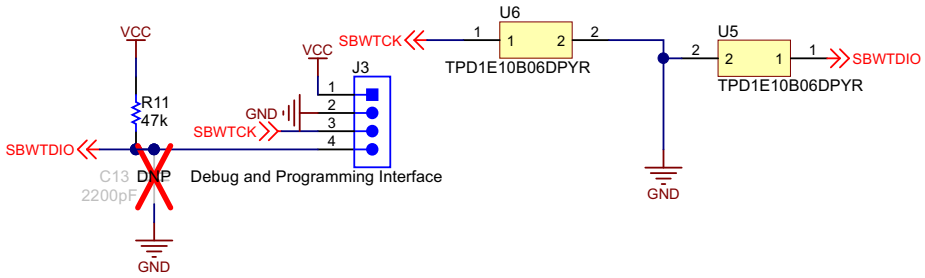


Figure 77. SBW Interface and ESD Protection

8.2 Bill of Materials

To download the bill of materials (BOM), see the design files at TIDA-00230.

Table 6. BOM

DESIGNATOR	DESCRIPTION	MANUFACTURER	PARTNUMBER	QUANTITY
!PCB1	Printed Circuit Board	Any	TIDA00230	1
C1	CAP, CERM, 0.47 µF, 25 V, ±10%, X7R, 0603	MuRata	GRM188R71E474KA12D	1
C2, C5, C8	CAP, CERM, 0.1 µF, 16 V, ±10%, X7R, 0402	MuRata	GRM155R71C104KA88D	3
C3, C4, C9	CAP, CERM, 1 µF, 16 V, ±10%, X7R, 0603	MuRata	GRM188R71C105KA12D	3
C6	CAP, CERM, 100 µF, 10 V, ±20%, X5R, 1206_190	TDK	C3216X5R1A107M160AC	1
C10	CAP, CERM, 39 pF, 50 V, ±5%, C0G/NP0, 0402	MuRata	GRM1555C1H390JA01D	1
D1, D2, D5	Diode, Schottky, 35 V, 0.1 A, SOD-523F	Comchip Technology	CDBU0130L	3
D3, D4	LED, Orange, SMD	Rohm	SML-311DTT86	2
FID1, FID2, FID3, FID4, FID5, FID6	Fiducial mark. There is nothing to buy or mount.	N/A	N/A	6
J1, J3	Header, 50 mil, 4×1, R/A, TH	Sullins Connector Solutions	GRPB041VWCN-RC	2
R1, R4, R7, R8, R10	RES, 0 Ω, 5%, 0.063 W, 0402	Vishay-Dale	CRCW04020000Z0ED	5
R2, R3	RES, 4.75 kΩ, 1%, 0.063 W, 0402	Vishay-Dale	CRCW04024K75FKED	2
R5, R6	RES, 1.00 kΩ, 1%, 0.063 W, 0402	Vishay-Dale	CRCW04021K00FKED	2
R11	RES, 47 kΩ, 5%, 0.063W, 0402	Vishay-Dale	CRCW040247K0JNED	1
U1	Mixed signal microcontroller, RGZ0048B	Texas Instruments	MSP430FR5969IRGZ	1
U2	Dynamic NFC interface transponder, PW0014A	Texas Instruments	RF430CL330HCPW	1
U5, U6	ESD in 0402 Package with 10-pF Capacitance and 6-V Breakdown, 1 Channel, –40°C to 125°C, 2-pin X2SON (DPY), Green (RoHS and no Sb/Br)	Texas Instruments	TPD1E10B06DPYR	2

8.3 Layer Plots

To download the layer plots, see the design files at TIDA-00230.

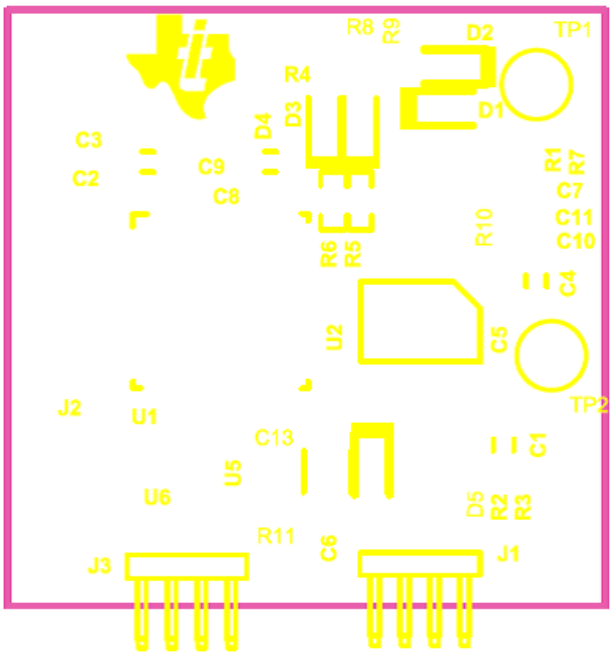


Figure 78. Top Overlay

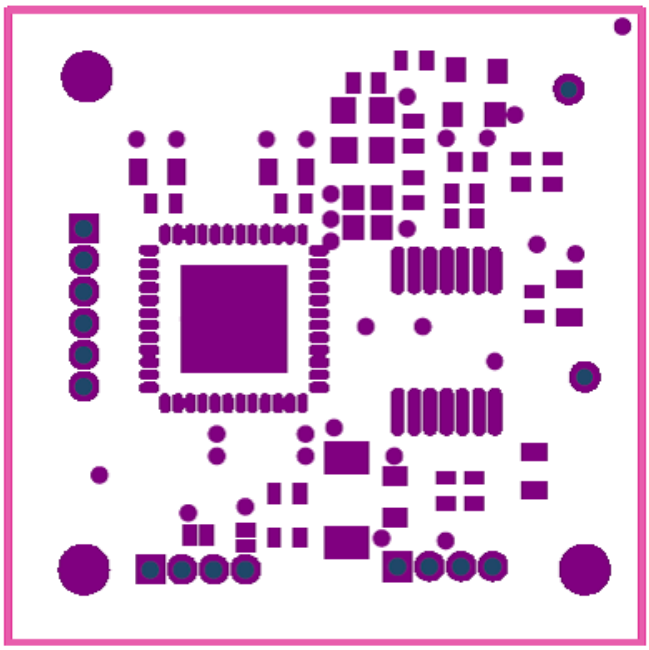


Figure 79. Top Solder Mask

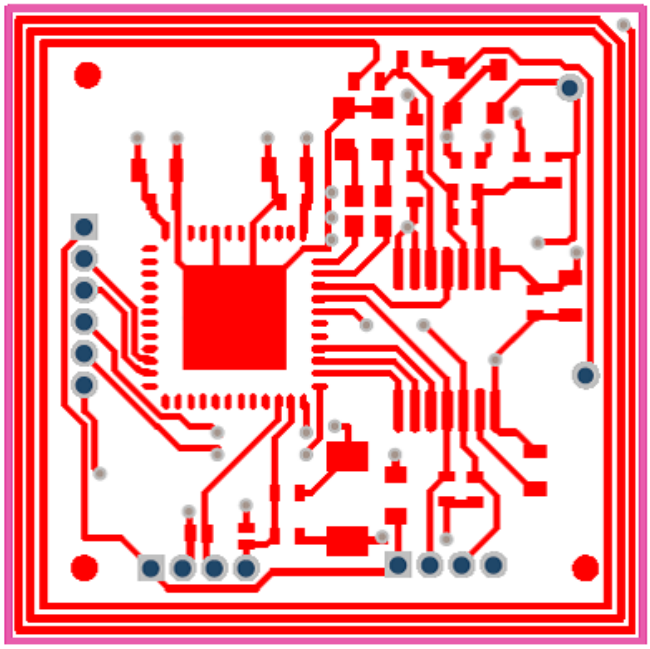


Figure 80. Top Layer

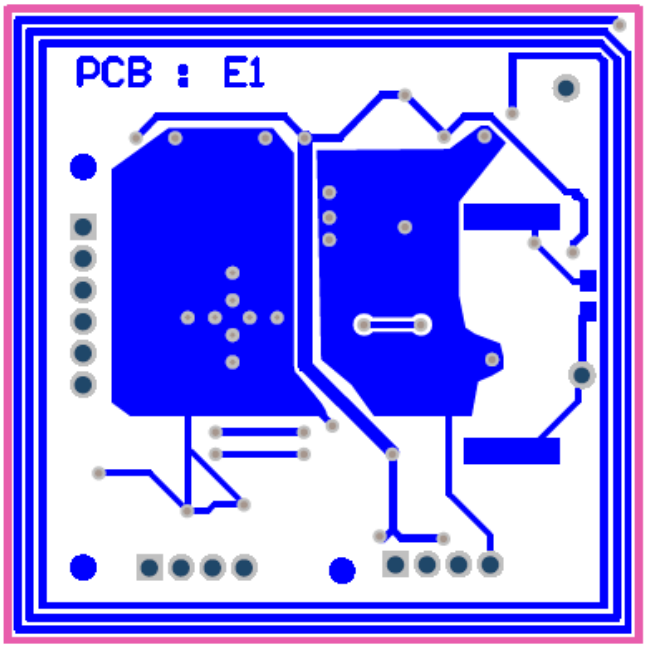


Figure 81. Bottom Layer

8.4 Altium Project

To download the Altium project files, see the design files at TIDA-00230.

8.5 Gerber Files

To download the Gerber files, see the design files at TIDA-00230.

8.6 Assembly Drawings

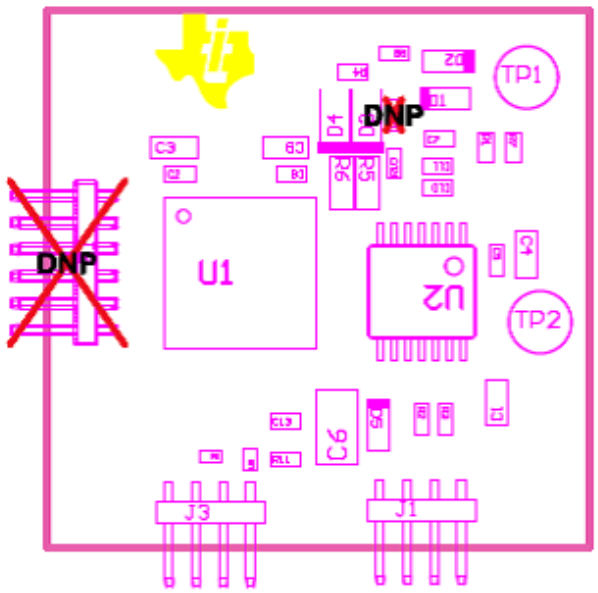


Figure 82. Assembly Drawing: Top

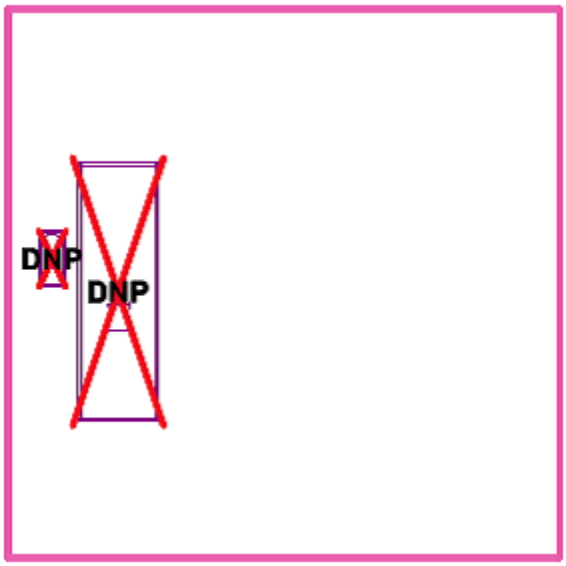


Figure 83. Assembly Drawing: Bottom

9 Software Files

To download the software files, see the design files at TIDA-00230.

10 References

1. J. Rodriguez, K. Remack, J. Gertas, L. Wang, C. Zhou, K. Boku, J. Rodriguez-Latorre, "Reliability of Ferroelectric Random Access Memory Embedded within 130nm CMOS," Reliability Physics Symposium (IRPS), 2010 IEEE International, Vol. May 2010, pp.750, 758.
2. S. S. Mohan, M. del Mar Hershenson, S. P. Boyd and T. H. Lee, "Simple Accurate Expressions for Planar Spiral Inductances," IEEE JOURNAL OF SOLID-STATE CIRCUITS, vol. 34, no. 10, pp. 1419–1424, 1999.
3. H. Greenhouse, "Design of Planar Rectangular Microelectronic Inductors," IEEE Transactions On Parts, Hybrids, And Packaging, Vols. VOL. PHP-10, NO. 2, no. June, pp. 101–109, 1974.
4. H. G. Dill, "Designing inductors for thin-film applications,," Electronic Design, vol. 12, no. 4, pp. 52–59, 1964.
5. Texas Instruments, *MSP430 Hardware Tools User's Guide*, (SLAU278T).
6. Texas Instruments,*MSP430FR59xx Mixed-Signal Microcontrollers*, (SLAS704D).
7. K. Aslanidis and E. LaCost, *RF430CL330H Practical Antenna Design Guide*, (SLOA197).

10.1 NFC Approved Specifications (Extract as of 2014/6/11)

- NFC Data Exchange Format (NDEF) Technical Specification
- NFC Forum Type 1 Tag Operation Specification 1.2
- NFC Forum Type 2 Tag Operation Specification 1.2
- NFC Forum Type 3 Tag Operation Specification 1.2
- NFC Forum Type 4 Tag Operation Specification 3.0
- NFC Record Type Definition (RTD) Technical Specification
- NFC Text Record Type Definition (RTD) Technical Specification
- NFC URI Record Type Definition (RTD) Technical Specification
- NFC Smart Poster Record Type Definition (RTD) Technical Specification
- NFC Generic Control Record Type Definition (RTD) Technical Specification
- NFC Signature Record Type Definition (RTD) Technical Specification
- NFC Signature Record Type Definition (RTD) Candidate Technical Specification 2.0
- NFC Forum Connection Handover Technical Specification 1.3
- NFC Forum Personal Health Device Communication (PHDC) Technical Specification
- NFC Logical Link Control Protocol (LLCP) 1.2 Technical Specification
- NFC Logical Link Control Protocol (LLCP) 1.0 Technical Specification (Deprecated)
- NFC Digital Protocol Technical Specification 1.1
- NFC Activity Technical Specification v1.1
- NFC Simple NDEF Exchange Protocol (SNEP) Technical Specification
- NFC Analog Technical Specification
- NFC Controller Interface (NCI) Technical Specification v1.1
- NFC LLCP to OBEX Protocol Binding Candidate Technical Specification
- Bluetooth Secure Simple Pairing Using NFC

11 Credits

The author wishes to acknowledge key contributions from colleagues that enabled this project to happen on time and in conformance with design specification and market needs. Especially Jason Kriek, who first had the idea of highlighting the benefit of FRAM in conjunction with NFC and realized the first proof of concept.

JASON KRIEK is a digital field applications engineer at Texas Instruments where he is responsible for supporting MCU and wireless connectivity products. His specific areas of expertise include embedded microcontrollers and related analog/digital sensing systems and transceivers operating at frequencies from 13.56 MHz to 2.45 GHz. Jason brings innovative and creative solutions to life that are driven by his passion for the combination of the technologies that he supports in his role.

12 About the Author

MATTHIEU CHEVRIER is a systems architect at Texas Instruments where he is responsible for defining and developing reference design solutions for the industrial segment. Matthieu brings to this role his extensive experience in designing embedded systems with both hardware (power management, mixed signal) and software (low level drivers, RTOS and compilers) competencies. Matthieu earned his master of science in electrical engineering (Ingénieur diplomé) from Supélec, an Ivy League university in France. Matthieu holds patents originally delivered by IPO, EPO, and USPTO.

Revision History

Changes from Original (June 2014) to A Revision	Page
• Added "Credits" section	78

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

IMPORTANT NOTICE FOR TI REFERENCE DESIGNS

Texas Instruments Incorporated ("TI") reference designs are solely intended to assist designers (“Buyers”) who are developing systems that incorporate TI semiconductor products (also referred to herein as “components”). Buyer understands and agrees that Buyer remains responsible for using its independent analysis, evaluation and judgment in designing Buyer’s systems and products.

TI reference designs have been created using standard laboratory conditions and engineering practices. **TI has not conducted any testing other than that specifically described in the published documentation for a particular reference design.** TI may make corrections, enhancements, improvements and other changes to its reference designs.

Buyers are authorized to use TI reference designs with the TI component(s) identified in each particular reference design and to modify the reference design in the development of their end products. HOWEVER, NO OTHER LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE TO ANY OTHER TI INTELLECTUAL PROPERTY RIGHT, AND NO LICENSE TO ANY THIRD PARTY TECHNOLOGY OR INTELLECTUAL PROPERTY RIGHT, 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 components or services are used. Information published by TI regarding third-party products or services does not constitute a license 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.

TI REFERENCE DESIGNS ARE PROVIDED "AS IS". TI MAKES NO WARRANTIES OR REPRESENTATIONS WITH REGARD TO THE REFERENCE DESIGNS OR USE OF THE REFERENCE DESIGNS, EXPRESS, IMPLIED OR STATUTORY, INCLUDING ACCURACY OR COMPLETENESS. TI DISCLAIMS ANY WARRANTY OF TITLE AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, QUIET ENJOYMENT, QUIET POSSESSION, AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHTS WITH REGARD TO TI REFERENCE DESIGNS OR USE THEREOF. TI SHALL NOT BE LIABLE FOR AND SHALL NOT DEFEND OR INDEMNIFY BUYERS AGAINST ANY THIRD PARTY INFRINGEMENT CLAIM THAT RELATES TO OR IS BASED ON A COMBINATION OF COMPONENTS PROVIDED IN A TI REFERENCE DESIGN. IN NO EVENT SHALL TI BE LIABLE FOR ANY ACTUAL, SPECIAL, INCIDENTAL, CONSEQUENTIAL OR INDIRECT DAMAGES, HOWEVER CAUSED, ON ANY THEORY OF LIABILITY AND WHETHER OR NOT TI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, ARISING IN ANY WAY OUT OF TI REFERENCE DESIGNS OR BUYER’S USE OF TI REFERENCE DESIGNS.

TI reserves the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products are sold subject to TI’s terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI’s terms and conditions of sale of semiconductor products. Testing and other quality control techniques for TI components are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

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

Reproduction of significant portions of TI information in TI data books, data sheets or reference designs is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards that anticipate dangerous failures, monitor failures and their consequences, lessen the likelihood of dangerous failures and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in Buyer’s safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI’s goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed an agreement specifically governing such use.

Only those TI components that TI has specifically designated as military grade or “enhanced plastic” are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components that have ***not*** been so designated is solely at Buyer's risk, and Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

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