BM019 Serial to NFC Converter

Serial Data to Near Field Communication Converter

hardware made easy

Solutions Cubed

designservices@solutions-cubed.com
phone – 530.891.8045
256 East First Street
Chico, CA 95928

Contact Solutions Cubed, LLC for your custom designs:

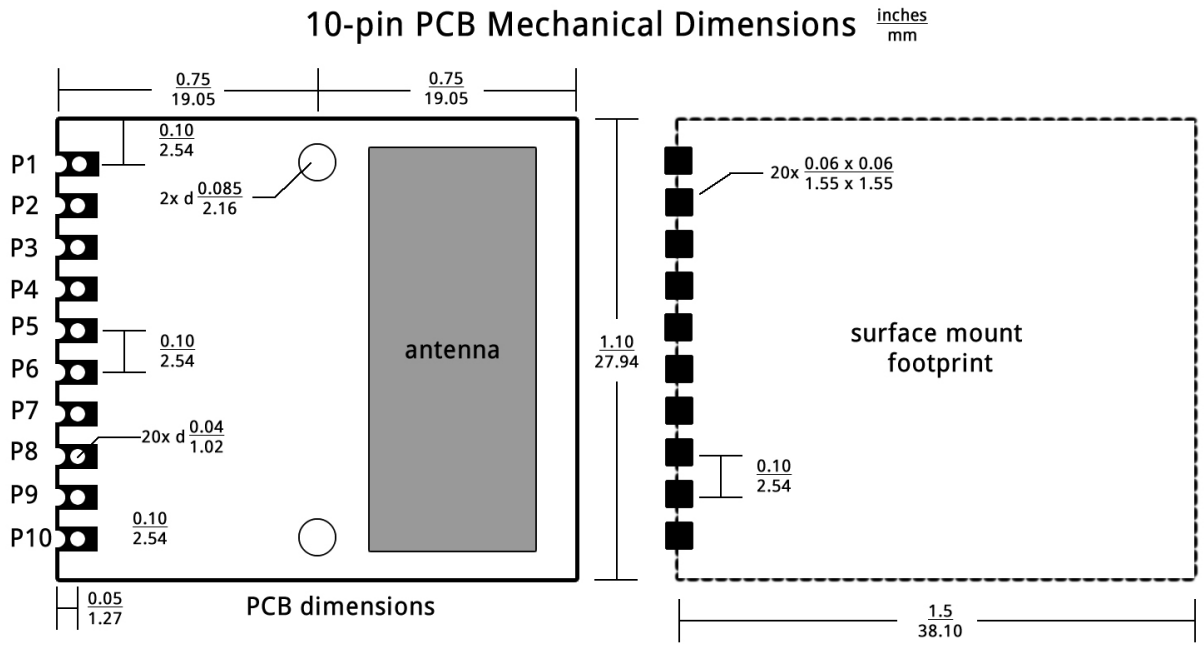
Solutions Cubed is an innovative electronic design firm. We have created successful designs for a myriad of industries including mass produced consumer products, deep-sea robotic components, and encrypted encoders for the banking industry. We love meeting new customers and are interested in hearing about your design needs.

Product Description:

This module carries an ST Micro CR95HF NFC interface chip. The module allows you to communicate with Near Field Communication enabled smart cards and EEPROM via 8N1 serial or SPI interfaces. The module carries a tuned PCB antenna for coupling to NFC enabled devices.

- RF communication at 13.56MHz
- ISO/IEC 14443 Type A and B
- ISO/IEC 15693
- ISO/IEC 18092
- NFC Forum tags: Types 1,2,3 and 4
- ST Micro short-range interface (SRI) tags
- ST Micro long-range interface (LRI) tags
- ST Micro dual interface EEPROM
- On board 3.3V regulator
- Tuned PCB antenna
- Open source hardware design

Dimensions:



Specifications:

Characteristic	Min	Typ	Max	Unit	Notes
VDD output voltage		3.3		V	VDD pin
Operating current		7		mA	
VIN Operating voltage	5		15	V	Supplies onboard regulator
Max voltage on input pins			3.6	V	
Input Low Voltage			0.66	V	
Input High Voltage	2.3			V	
Output Low Voltage			0.5	V	
Output High Voltage	2.3			V	
Default baud rate		57.6		BPS	Adjustable between 26.48KBPS and 2.26MBPS
RF Coupling Range		1	2	Inch	
Operating temperature	-40		+85	°C	

Pin Functions and Notes

#	Name	Maximum Voltage	Notes
1	DOUT	VDD	UART (serial 8N1) data output from CR95HF, 270Ω resistor in series
2	DIN	VDD	UART (serial 8N1) data input to CR95HF (NOT 5V TOLERANT)
3	SS	VDD	SPI slave select pin (NOT 5V TOLERANT)
4	MISO	VDD	SPI master in slave output data line, 270Ω resistor in series
5	MOSI	VDD	SPI master out slave input data line (NOT 5V TOLERANT), 3.3KΩ resistor pulled to 3.3V
6	SCK	VDD	SPI clock output from master to CR95HF (NOT 5V TOLERANT)
7	SS_0	VDD	Serial select, 0V = UART (serial 57.6KBPS 8N1) interface 3.3V = SPI interface
8	VDD	VDD	Output from on board 3.3V regulator
9	VIN	15V	Input voltage to on board regulator
10	GND	0V	Ground return for module

User Notes/Tips

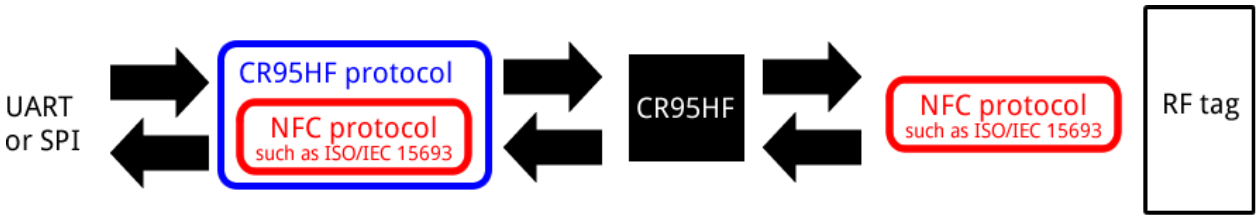
1. This module operates at 3.3V as does **NOT** have 5V tolerant pins.

2. A wake up pulse (low) is required after power –up to set the communication interface (SPI or UART).

3. Communication flow: The communication flow to and from the CR95HF can appear somewhat confusing. The CR95HF handles NFC (RF) communication to and from RF tags. The CR95HF will generate start of frames (SOF), cyclical redundancy checks (error detection - CRC16) and end of frames (EOF) that are required by the NFC protocol you are using. It also handles timing and specific requirements of the RF communication.

   Portions of the NFC protocol must be provided by the user. These portions are embedded in a separate protocol specific to the CR95HF. The user must correctly embed the NFC protocol into the CR95HF protocol.

   In this document we provide several examples of how to do this using common commands and an NFC protocol used by ST Micro, the manufacturer of the CR95HF.



4. The ST Micro datasheet for the CR95HF has a detailed description of the serial communication protocol used by that chip. Additionally, the specific protocol you use (ISO/IEC 15693, ISO/IEC 18092, etc.) are very detailed and not covered in this document. It is recommended that you familiarize yourself with the protocols specific to your application.

5. You must set the parameters of the RF protocol you are using with the CR95HF's Protocol Select command before you can communicate with any RF tags.

6. The CR95HF command set allows you to modify the RF gain and modulation. The commands are detailed in the CR95HF datasheet and not in this document. However, we found that the Read Register Command response differs from the ST Micro datasheet. Their document describes the process as…

Write the ARC_B register index at 0x01	>>>0x09 0x03 0x68 0x00 0x01
CR95HF reply	<<<0x00 0x00
Read the ARC_B register details	>>>0x08 0x03 0x69 0x01 0x00
CR95HF reply	<<<0x01 0x53
The response we saw had a leading 0	**<<<0x00 0x01 0x53**

Configuring for UART or SPI Communication:

The CR95HF can communicate with a serial 8N1 57.6KBPS UART interface, or with an SPI interface.

There are tasks that must be completed to set the communication option after power up. You must set the state of the SS_0 pin and send a wake-up pulse on the DIN line.

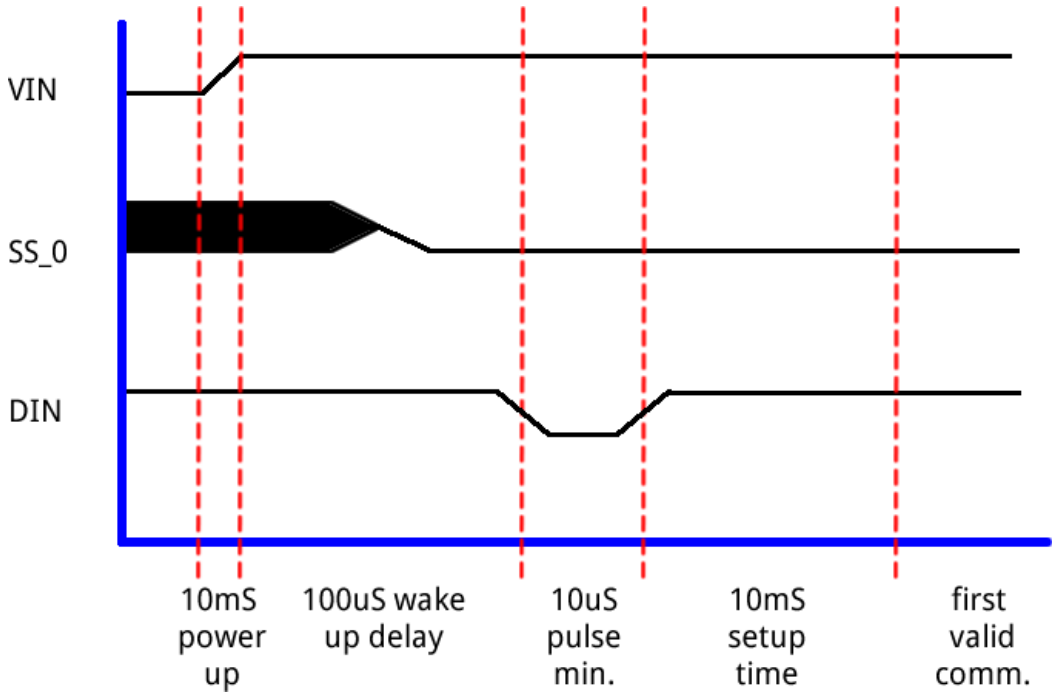
SS_0 State:

SS_0 = 3.3V	SPI mode, although a wake-up pulse must still be sent on the DIN line
SS_0 = 0V	UART mode, sending a 0x00 via the serial interface can work as a wake-up pulse.

Wake-Up Pulse:

The wake-up pulse requirement for the CR95HF makes the SPI interface less desirable since it requires an additional i/o pin to create the pulse. However, some devices may not be capable of generating accurate 57.6KBPS serial data but can still communicate with the CR95HF using SPI.

The image below shows signals for setting the CR95HF to UART mode. Tie SS_0 to 3.3V to set the CR95HF for SPI mode.



Communication Protocol

Overview:

The BM019 carries ST Micro's CR95HF 13.56MHz multi-protocol contactless transceiver IC with SPI and UART serial access. The datasheet for the CR95HF (located at ST Micro's web site) has detailed descriptions for the communication protocol the device uses. In addition, users should be familiar with the RF protocol you are attempting to implement.

In this document we will summarize some of the commands used with the CR95HF. This information is specific to the UART serial interface and ISO/IEC 15693. This does not cover all possible commands but does provide a bare bones introduction to using the CR95HF with the protocol ISO/IEC 15693.

The CR95HF acts as a pass-through device converting serial commands to RF signals and vice-versa. RF protocol commands are wrapped within CR95HF commands.

Hardware Usage:

The asynchronous serial communication takes place at a default value of 57600BPS using eight data bits, no parity, and one stop bit (57600, n, 8, 1). Data is sent LSB first. There is no hardware flow control. Serial data is received by the module on its DIN pin. It sends data on its DOUT pin.

CR95HF Commands:

These commands are sent to the CR95HF.

Code (hexadecimal)	Command	Description
0x01	Identification Number	Provides information about the CR95HF IC
0x02	Protocol Select	Selects the RF communication protocol
0x04	Send Receive	This command sends and receives data from the RF tag and is the most commonly used
0x07	Idle	Puts the CR95HF in a low current mode waiting for a wake up event. *Not covered here, see the ST Micro datasheet for details.*
0x08	Read Register	Reads values from registers internal to the CR95HF. *Not covered here, see the ST Micro datasheet for details.*
0x09	Write Register	Writes values to registers internal to the CR95HF. *Not covered here, see the ST Micro datasheet for details.*
0x0A	Baud Rate	Can be used to change the UART baud rate from the default setting of 57.6KBPS. Not that the lowest baud rate allowed in is in the area of 27KBPS. *Not covered here, see the ST Micro datasheet for details.*
0x55	Echo	CR95HF responds to an echo with 0x55

Identification Number Command:
This command returns information about the CR95HF. This command takes up to 6mS for a response to be returned.

Direction	Data (hexadecimal)	Comments
Sent to CR95HF	0x01	Command code
	0x00	Length of data
Response from CR95HF	0x00	Result code
	<len>	Length of data
	<device ID>	Device ID in ASCII, example - 'NFC FS2JAST2'
	<ROM CRC>	Two bytes in length, example '0x75D2

Protocol Select Command:
This command selects the RF protocol to use. It must be sent prior to communicating with tags. Different protocols have different requirements and parameters. In this example only ISO/IEC 15693 is shown. See the CR95HF datasheet details regarding other RF protocols.

Direction	Data (hexadecimal)	Comments
Sent to CR95HF	0x02	Command code
	<Len>	Length of data
	<Protocol>	0x00 = field off 0x01 = ISO/IEC 15693 0x02 = ISO/IEC 14443-A 0x03 = ISO/IEC 14442-B 0x04 = ISO/IEC 18092 / NFC forum tag type 3
	<parameters>	See below
Response from CR95HF	<Result>	Result code: 0x00 = accepted 0x82 = invalid command length 0x83 = invalid protocol
	0x00	Length of data

ISO/IEC 15693 Parameters Byte

Bit	Function
0	0: do not add CRC16 to data sent 1: generate and add CRC16 to commands
1	0: single subcarrier 1: dual subcarrier
2	0: 100% modulation 1: 30% modulation
3	0: respect 312uS delay 1: Wait for SOF
5:4	00: 26KBPS 01: 52KBPS 10: 6KBPS 11: reserved
7:6	reserved

Example: To set the CR95HF for ISO/IEC 15693 with CRC appended, single subcarrier, 100% modulation, wait for SOF, and 52KBPS you would send 0x02, 0x02, 0x01, 0x19 with the last byte being the parameter byte. If accepted the CR95HF would return 0x00, 0x00.

Send Receive Command:
This command sends data to a RF tag and returns the response. This is the command that will be used to read-write the RF tags as well as implement other commands specific to the protocol the tag uses.

Direction	Data (hexadecimal)	Comments
Sent to CR95HF	0x04	Command code
	<Len>	Length of data
	<Data>	Data being sent. This data will match the format of the protocol you are using. Note that the CR95HF will generate things like the SOF, CRC16, and EOF required by the protocol if those options are selected.
Response from CR95HF	0x80	Result code returned by the CR95HF Result codes: 0x80 – no error 0x86 – communication error 0x87 – frame wait timeout or no tag 0x88 – invalid SOF (start of frame) 0x89 – receive buffer overflow (CR95HF received too many bytes) 0x8A – framing error 0x8B – EGT timeout (for ISO/IEC 14443-B) 0x8C – invalid length (NFC forum tag type 3) 0x8D – CRC error (NFC forum tag type 3) 0x8E – reception lost without EOF (end of frame)
	<Len>	Length of data
	<Data>	Data response from the RF tag. This will differ depending on the command being sent/received.
	<CRC16>	Original CRC16 received by the CR95HF
	<Error>	If bit 0 set then collision was detected, if bit 1 is set then a CRC16 error was detected.

Echo Command:
The CR95HF will echo 0x55 if it is received. Sending the 0x55 is a good way to verify the CR95HF is connected and powered correctly.

Direction	Data (hexadecimal)	Comments
Sent to CR95HF	0x55	Command code
Response from CR95HF	0x55	Result code

Additional Commands:
Commands not described here are detailed in the CR95HF datasheet at ST Micro's web site. These commands can allow you to adjust RF settings and to calibrate the RF receiver thresholds.

ISO/IEC 15693 Protocol Commands:
The various NFC RF tag protocols have diverse command sets. Each protocol is described in detail in ISO International Standard documents. For example, details of the command set for the ISO/EIC 15693 can be found in the ISO/IEC JTC 1/SC 17 1692 document ISO/IEC FCD 15693-3.

Generally speaking the CR95HF is configured to operate with specific RF hardware protocol settings using the Set Protocol command (command code 0x02). After the protocol is selected commands associated with that protocol are embedded in the CR95HF Send Receive command (command code 0x04).

For brevity this document only covers four commands associated with the ISO/IEC 15693 protocol. Additionally, it will not cover methods of communicating with one RF tag when more than one might be in range of the CR95HF (addressed mode and select mode).

Commands italicized and bolded are described in this document

Code (hexadecimal)	Command	Description
0x01	***Inventory***	***Receives information about RF tags in range. Can be used to get UID.***
0x02	Stay quiet	Selected unit will not send back a response
0x03 to 0x1F	reserved	Reserved for future use
0x20	***Read single block***	***Read a single block of memory from RF tag***
0x21	***Write single block***	***Write a single block of memory to RF tag***
0x22	Lock block	Permanently locks a block of memory
0x23	Read multiple blocks	Reads multiple blocks of memory from RF tag
0x24	Write multiple blocks	Writes multiple blocks of memory from RF tag
0x25	Select	Used to select a specific card via the UID to communicate with
0x26	Reset to ready	Returns RF tag to ready state
0x27	Write AFI	Writes application family identifier to RF tag
0x28	Lock AFI	Permanently locks application family identifier
0x29	Write DSFID	Writes data storage format identifier to RF tag
0x2A	Lock DSFID	Permanently locks data storage format identifier
0x2B	***Get system information***	***Gets information from RF tag that includes the memory block size in bytes and the number of memory blocks.***
0x2C	Get multiple block security status	
0x2D to 0x9F	Reserved	Reserved for future use
0xA0 to 0xDF	Custom	Depends on Manufacturer product
0xE0 to 0xFF	Proprietary	Depends on Manufacturer product

Command Fields:
ISO/IEC 15693 has common bit fields in commands and responses. There are also data fields that are optional. The optional fields tend to relate to methods of segregating RF tags by ID or family. After these fields are described we'll begin looking at specific commands and their structure.

UID Format:
Each RF tag has a universal identifier, or UID. These can be used to ensure one-on-one communication where multiple tags might be present. The UID could also be used to track tag use and movement.

The UID consists of a 64-bit field.

MSB		LSB
bits 64-57	bits 56-49	bits 48-1
0xE0	IC manufacturer's code	IC manufacturer's serial number

For example, an ST Micro LRi2K card might have a UID of 'E0 02 204238196951' in hexadecimal (spaces are shown for clarity and not part of the UID).

Request Flags:
Each command will populate an 8-bit field of 'Request Flags'. The upper four bit definitions change based on the value of bit 3 (Inventory flag).

Bit	Flag name	Description
1	Sub-carrier flag	0: single sub-carrier used 1: two sub-carriers used
2	Data rate flag	0: low data rate is used 1: high data rate is used
3	Inventory flag	State determines how bits 5-8 are defined
4	Protocol extension flag	0: no protocol extension 1: protocol format is extended
If Inventory flag = 0 (not set)		
5	Select flag	0: request shall be executed based on setting of the address flag 1: request shall be executed only by devices in the selected state
6	Address flag	0: request is not addressed 1: request is addressed, optional UID field is present
7	Option flag	Meaning is defined by the command description, if not used set to 0
8	reserved	
If Inventory flag = 1 (set)		
5	AFI flag	0: AFI field is not present 1: AFI field is present
6	Nb slots flag	0: 16 slots 1: 1 slot
7	Option flag	Meaning is defined by the command description, if not used set to 0
8	reserved	

Response Flags:
When an RF tag responds it will include the Response Flag byte. This byte indicates whether or not there was an error and if the protocol has been extended. A protocol might be extended based on the RF device you are communicating with. For example, devices with more than 256 memory blocks will use the protocol extension.

Bit	Flag name	Description
1	Error flag	0: no error occurred 1: error detected, error is indicated in the error field
2	reserved	Shall be set to 0
3	reserved	Shall be set to 0
4	Extension flag	0: no protocol format extension 1: protocol format is extended
5	reserved	Shall be set to 0
6	reserved	Shall be set to 0
7	reserved	Shall be set to 0
8	reserved	Shall be set to 0

Response Error Codes:
If an error is detected an error code will be returned as part of the response packet. Therefore, if bit 0 of the Response Flags byte is set the next byte in the response will be the Error Code. Here are the codes. Examples will be shown later.

Error Code	Meaning
0x01	Command not supported
0x02	Command is not recognized, could be a format error
0x03	Option is not supported
0x0F	Unknown error
0x10	Specified memory block does exist
0x11	Specified memory block is locked and cannot be locked again
0x12	Specified memory block is locked and cannot be changed
0x13	Specified memory block was not successfully programmed
0x14	Specified memory block was not successfully locked
0xA0 to 0xDF	Custom command codes
0xE0 to 0xFF	Reserved

Inventory Command:
Below is the format for the Inventory command. Earlier we mentioned that we'd simplify our examples to non-addressed commands. Additionally, the CR95HF handles the SOF and EOF generation. It can also be set to generate the CRC16 if the Set Protocol command is configured correctly.

So the shaded boxes are data we don't need to worry about. That means the Request Flag and Inventory command need to be embedded in the CR95HF Send Receive commands

ISO/IEC 15693 Inventory command format

SOF	Request Flags	Inventory Command Code	Optional AFI	Mask Length	Mask Value	CRC16	EOF
	8 bits	8 bits	8 bits	8 bits	0-64 bits	16 bits	
CR95HF takes care of this	Bit 4 needs to be set to enable the protocol extension	0x01	Not used in non-addressed modes	0x00 since no mask follows	Not used in non-addressed modes	CR95HF takes care of this	CR95HF takes care of this

CR95HF Send Receive command format

Command code	**Length**	**Data**		
		Request Flags	**Inventory Command**	**Mask Length**
0x04	0x03	0x26	0x01	0x00

The Inventory command is therefore executed by sending 0x04, 0x03, 0x26, 0x01, 0x00 to the CR95HF.

ISO/IEC 15693 Inventory command response format with no error

SOF	Response Flags	DSFID	UID	CRC16	Error
	8 bits	8 bits	64 bits	16 bits	8 bits
CR95HF takes care of this			Not used in non-addressed modes	CRC16 received by CR95HF	CRC16 or collision error bits

CR95HF Send Receive response format example shown in the order data would be received. Obviously the actual data received would be based on the RF tag UID etc..

Result code	**Length**	**Data**				
		Response flags	**DSFID**	**UID**	**CRC16**	**Error**
0x80	0x0D	0x00	0x00	0x51 0x69 0x19 0x38 0x42 0x20 0x02 0xE0	0x84 0x28	0x00

Read Single Block Command:
This command reads a single block of memory from the RF tag.

ISO/IEC 15693 Read Single Block command format

SOF	Request Flags	Read Single Block Command Code	Optional UID	Block Number	CRC16	EOF
	8 bits	8 bits	64 bits	8 bits	16 bits	
CR95HF takes care of this		0x20	Not used in non-addressed modes	The block of memory you want to read	CR95HF takes care of this	CR95HF takes care of this

CR95HF Send Receive command format

Command code	**Length**	**Data**		
		Request Flags	**Read Block Command**	**Block Number**
0x04	0x03	0x02	0x20	0x00

A Read Single Block from block 0x00 is executed by sending 0x04, 0x03, 0x02, 0x20, 0x00 to the CR95HF.

ISO/IEC 15693 Read Single Block response format with no error

SOF	Response Flags	Optional Sector Security Status	Data	CRC16	Error
	8 bits	8 bits	Block length	16 bits	8 bits
CR95HF takes care of this		Will be returned of the option bit is set in the Request Flags	Data will be returned LSB first	CRC16 received by CR95HF	CRC16 or collision error bits

CR95HF Send Receive response format example shown in the order data would be received.

Result code	**Length**	**Data**			
		Response Flags	**Data**	**CRC16**	**Error**
0x80	0x08	0x00	0x00 0x00 0x00 0x00	0x77 0xCF	0x00

Read Single Block response format **with error set (bit 0 of Response Flag)**

SOF	Response Flags	Error Code	CRC16	Error
	8 bits	8 bits	16 bits	8 bits
CR95HF takes care of this		Returned when error bit is set	CRC16 received by CR95HF	CRC16 or collision error bits

Read Single Block Command (continued):
For RF tags with more than 256 memory blocks you'll need to set the protocol extension bit (bit 4) of the Request Flags byte. Check the datasheet for the device you are interfacing with. Here are examples of the format required by ST Micro's M24LR64E-R dual interface EEPROM (64K bits of memory).

ISO/IEC 15693 Read Single Block command format with protocol extended

SOF	Request Flags	Read Single Block Command Code	Optional UID	Block Number LSB	Block Number MSB	CRC16	EOF
	8 bits	8 bits	64 bits	8 bits		16 bits	
CR95HF takes care of this	Bit 4 must be set to enable the protocol extension	0x20	Not used in non-addressed modes	LSB of block of memory you want to read	MSB of block of memory you want to read	CR95HF takes care of this	CR95HF takes care of this

CR95HF Send Receive command format

Command code	**Length**	**Data**		
		Request Flags	**Read Block Command**	**Block Number**
0x04	0x04	0x0A	0x20	0x00 0x00

The Read Single Block reading from block 0x0000 is executed by sending 0x04, 0x04, 0x0A, 0x20, 0x00, 0x00 to the CR95HF. Note that the protocol extension bit is set in the Request Flags.

ISO/IEC 15693 Read Single Block response format with no error

SOF	Response Flags	Data	CRC16	Error
	8 bits	Block length	16 bits	8 bits
CR95HF takes care of this		Usually 4 bytes, 32 bits	CRC16 received by CR95HF	CRC16 or collision error bits

Write Single Block Command:
This command writes data to a single block of memory on the RF tag.

ISO/IEC 15693 Write Single Block command format

SOF	Request Flags	Write Single Block Command Code	Optional UID	Block Number	Data	CRC16	EOF
	8 bits	8 bits	64 bits	8 bits	Length of block	16 bits	
CR95HF takes care of this		0x21	Not used in non-addressed modes	The block of memory you write to	Usually 4 bytes but depends on RF tag	CR95HF takes care of this	CR95HF takes care of this

CR95HF Send Receive command format

Command code	**Length**	**Data**			
		Request Flags	**Block Command**	**Block Number**	**Data**
0x04	0x07	0x02	0x21	0x14	0x50 0x78 0xA0 0xC8

The Write Single Block writing 4 bytes to block 20 (0x14) is executed by sending 0x04, 0x07, 0x02, 0x21, 0x14 0x50, 0x78, 0xA0, 0xC8 to the CR95HF. Obviously the data shown in this example would change based on your needs.

ISO/IEC 15693 Write Single Block response format with no error

SOF	Response Flags	CRC16	Error
	8 bits	16 bits	8 bits
CR95HF takes care of this		CRC16 received by CR95HF	CRC16 or collision error bits

CR95HF Send Receive response format example shown in the order data would be received.

Result code	**Length**	**Data**		
		Response Flags	**CRC16**	**Error**
0x80	0x04	0x00	0x77 0xCF	0x00

Write Single Block response format **with error (bit 0 of response flag set)**

SOF	Response Flags	Error Code	CRC16	Error
	8 bits	8 bits	16 bits	8 bits
CR95HF takes care of this			CRC16 received by CR95HF	CRC16 or collision error bits

Write Single Block Command (continued):
For RF tags with more than 256 memory blocks you'll need to set the protocol extension bit (bit 4) of the Request Flags byte when writing data. Check the datasheet for the device you are interfacing with. Here are examples of the format required by ST Micro's M24LR64E-R dual interface EEPROM (64K bits of memory).

ISO/IEC 15693 Write Single Block command format with protocol extended

SOF	Request Flags	Write Single Block Command Code	Optional UID	Block Number LSB	Block Number MSB	Data	CRC16	EOF
	8 bits	8 bits	64 bits	8 bits	8 bits	Depends on block size	16 bits	
CR95HF takes care of this	Bit 4 must be set to enable the protocol extension	0x21	Not used in non-addressed modes	LSB of block of memory you want to read	MSB of block of memory you want to read	Usually 4 bytes, or 32 bits	CR95HF takes care of this	CR95HF takes care of this

CR95HF Send Receive command format

Command Code	**Length**	**Data**			
		Request Flags	**Block Command**	**Block Number**	**Data**
0x04	0x08	0x0A	0x21	0x00 0x00	0x50 0x78 0xA0, 0xC8

The Write Single Block to block 0x0000 is executed by sending 0x04, 0x08, 0x0A, 0x21, 0x00, 0x00, 0x50, 0x78, 0xA0, 0xC8 to the CR95HF. Obviously the block number and data would change based on your needs.

ISO/IEC 15693 Write Single Block response format with no error

SOF	Response Flags	CRC16	Error
	8 bits	16 bits	8 bits
CR95HF takes care of this		CRC16 received by CR95HF	CRC16 or collision error bits

Get System Information Command:
This command reads system information from the RF tag. The information includes indicators about the features of the device and important parameters about memory block size, number of memory blocks, and configuration.

ISO/IEC 15693 Get System Information command format

SOF	Request Flags	Get System Info	Optional UID	CRC16	EOF
	8 bits	8 bits	64 bits	16 bits	
CR95HF takes care of this		0x2B	Not used in non-addressed modes	CR95HF takes care of this	CR95HF takes care of this

CR95HF Send Receive command format

Command code	**Length**	**Data**	
		Request Flags	**Block Command**
0x04	0x02	0x02	0x2B

A Get System Information is executed by sending 0x04, 0x02, 0x02, 0x2B to the CR95HF.

ISO/IEC 15693 Get System Information response format (no error). Note: light grey columns may not be supported, check Info Flags.

SOF	Response Flags	Info Flags	UID	DSFID	AFI	Memory Size	IC ref.	CRC16	Error
	8 bits	8 bits	64 bits	8 bits	8 bits	16 bits	8 bits	16 bits	8 bits
CR95HF takes care of this		See below		See Info flags	See Info flags	See Info flags	See Info flags	CRC16 received by CR95HF	CRC16 or collision error bits

CR95HF Send Receive response format.

Result code	**Length**	**Data**		
		Response Flags	**Info Flags**	**UID**
0x80	0x12	0x00	0x0F	0x69 0x55 0x19 0x38 0x42 0x20 0x02 0xE0

Data (continued)					
DSFID	**AFI**	**Memory Size**	**IC Ref**	**CRC16**	**Error**
0x00	0x00	0x3F 0x03	0x20	0xB4 0xA9	0x00

Get System Information response format (**with error - bit 0 of Response Flag set)**

SOF	Response Flags	Error Code	CRC16	Error
	8 bits	8 bits	16 bits	8 bits
CR95HF takes care of this		Returned when error bit is set	CRC16 received by CR95HF	CRC16 or collision error bits

Get System Information Command (continued):
For RF tags with more than 256 memory blocks you'll need to set the protocol extension bit (bit 4) of the Request Flags byte. Check the datasheet for the device you are interfacing with. Here are examples of the format required by ST Micro's M24LR64E-R dual interface EEPROM (64K bits of memory).

ISO/IEC 15693 Get System Information command format

SOF	Request Flags	Get System Info	Optional UID	CRC16	EOF
	8 bits	8 bits	64 bits	16 bits	
CR95HF takes care of this	Bit 4 must be set to enable the protocol extension	0x2B	Not used in non-addressed modes	CR95HF takes care of this	CR95HF takes care of this

CR95HF Send Receive command format

Command code	**Length**	**Data**	
		Request Flags	**Block Command**
0x04	0x02	0x0A	0x2B

A Get System Information is executed by sending 0x04, 0x02, 0x0A, 0x2B to the CR95HF.

ISO/IEC 15693 Get System Information response format with no error

SOF	Response Flags	Info Flags	UID	DSFID	AFI	Memory Size	IC ref.	CRC16	Error
	8 bits	8 bits	64 bits	8 bits	8 bits	24 bits	8 bits	16 bits	8 bits
CR95HF takes care of this		See below		See Info flags	See Info flags	See Info flags	See Info flags	CRC16 received by CR95HF	CRC16 or collision error bits

CR95HF Send Receive response format.

Result code	**Length**	**Data**		
		Response Flags	**Info Flags**	**UID**
0x80	0x13	0x00	0x0F	0x7F 0x57 0x96 0x2D 0x28 0x2C 0x02 0xE0

Data (continued)					
DSFID	**AFI**	**Memory Size**	**IC Ref**	**CRC16**	**Error**
0xFF	0x00	0xFF 0x07 0x03	0x2C	0xBA 0x47	0x00

Info Flags: The Get System Information command returns a field with flags that indicate if specific fields are present in the response.

Bit	Flag name	Description
1	DSFID	0: DSFID is not supported, field is not present in response 1: DSFID is supported and field is present in response
2	AFI	0: AFI is not supported, field is not present in response 1: AFI is supported and field is present in response
3	Memory Size	0: Memory size is not supported, field is not present in response 1: Memory size is supported and field is present in response
4	IC ref	0: IC ref. is not supported, field is not present in response 1: IC ref. is supported and field is present in response
5	reserved	Shall be set to 0
6	reserved	Shall be set to 0
7	reserved	Shall be set to 0
8	reserved	Shall be set to 0

Memory Size: Get System Information will return memory size information when that field is supported by the RF tag. The field is typically 16 bits wide. For devices with more than 256 blocks you will need to set the protocol extension flag in the Request Flags byte, and an additional byte of data will be included in the Memory Size field (24 bits instead of 16 bits).

The number of blocks and block size in bytes returned will be one less than the actual number. For example, if the block size in bytes is 3 there are 4 bytes per memory block. If the number of blocks returned is 63, there are 64 blocks of 4 bytes.

MSB		LSB
Bits 16-14	Bits 13-9	Bits 8-1
Reserved	Block size in bytes	Number of blocks

IC Reference: The IC reference field may be included in the Get System Information command.

Communicating Via SPI

The CR95HF has both a UART and SPI interface. Previously in this document we've covered some of the basics of the CR95HF serial protocol. We've also touched on some aspects of ISO/IEC 15693 RF tag protocol. Using the SPI interface adds an additional level of requirements.

Some SPI basics…
MOSI = master out slave in, or an input on the CR95HF which is the slave
MISO = master in slave out, or an output on the CR95HF which is the slave
Data is sent and received MSB first with the CR95HF
The SPI operates using Mode 0: clock is idle low and data shifted on the falling edge.

The SPI protocol requires a three step process.

Step 1. Send the command: The command data is preceded by a control byte. When sending a command the control byte is 0. The example below shows how to send an ISO/IEC 15693 Inventory command.

MOSI	0x00	0x04	0x03	0x26	0x01	0x00
	SPI Control Byte	CR95HF Send Receive Command	CR95HF length	ISO 15693 Request Flags	ISO 15693 Inventory command code	ISO 15693 Mask length
MISO	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
	Don't care	Don't care	Don't care	Don't care	Don't care	Don't care

Step 2. Poll to see if a response is ready: When the CR95HF is ready to respond to a command in SPI mode it will do two things. It will set the 3rd bit of the byte it returns when the control byte is 0x03. It will pulse the DOUT pin low. If you have an interrupt input you can connect it to DOUT and dispense with the polling. Otherwise you should poll to see when you can read data.

MOSI	0x03	0x03	0x03	0x03	0x03	0x03
	SPI Control Byte	SPI Control Byte	SPI Control Byte	SPI Control Byte	SPI Control Byte	SPI Control Byte
MISO	xxxxxxxx	00000xxx	00000xxx	00000xxx	00000xxx	00001xxx
	Don't care	Repeat sending poll control byte until bit 3 is set				Data ready

Step 3: Read the data by sending the SPI control byte 0x02 and reading back the data.

MOSI	0x02	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
	SPI Control Byte	Don't care	Don't care	Don't care	Don't care	Don't care
MISO	xxxxxxxx	0x80	0x0D	Data0	Data1	DataN
	Don't care	Result code	Length	Data returned		

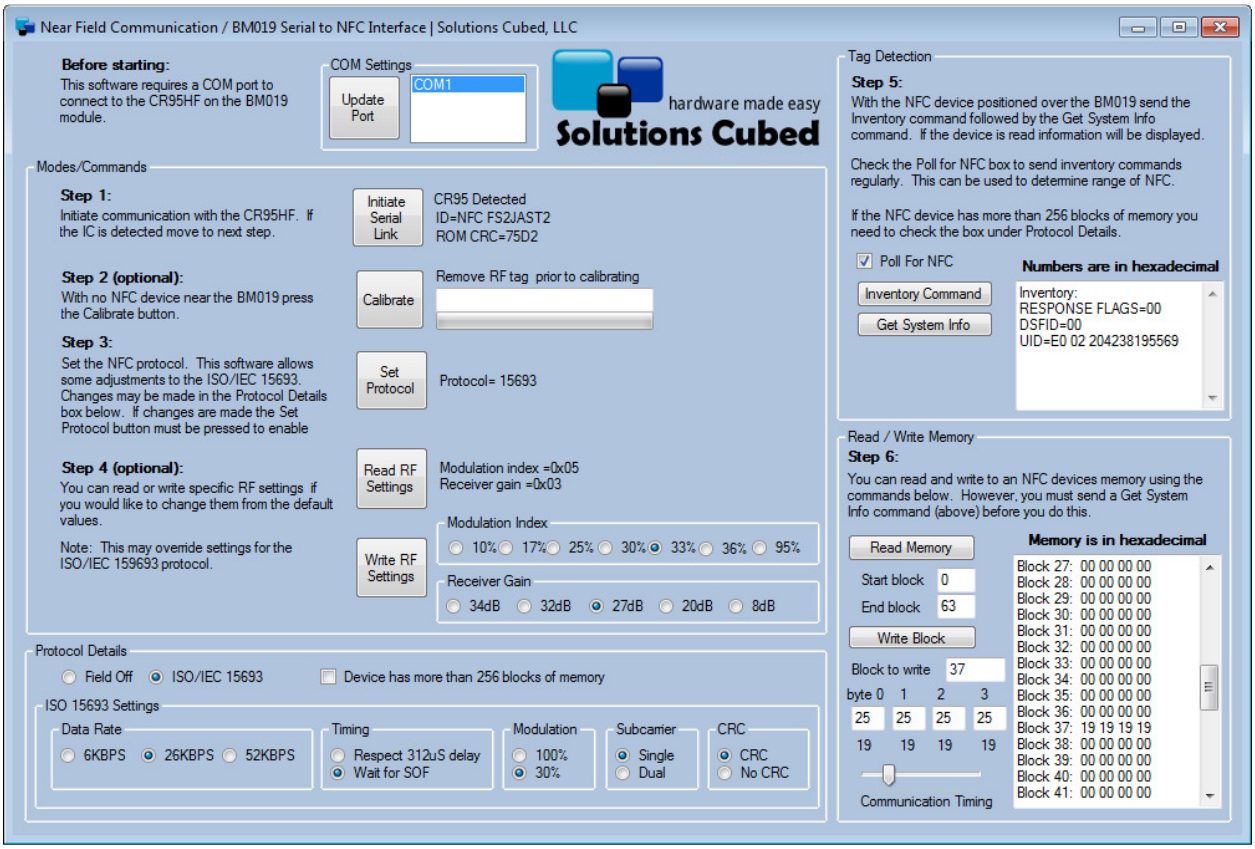
SPI Code Example in C:

The code below implements an ISO/IEC 15693 Inventory Command by wrapping it in the CR95HF Send Receive Command format using the 3 step SPI communication method.

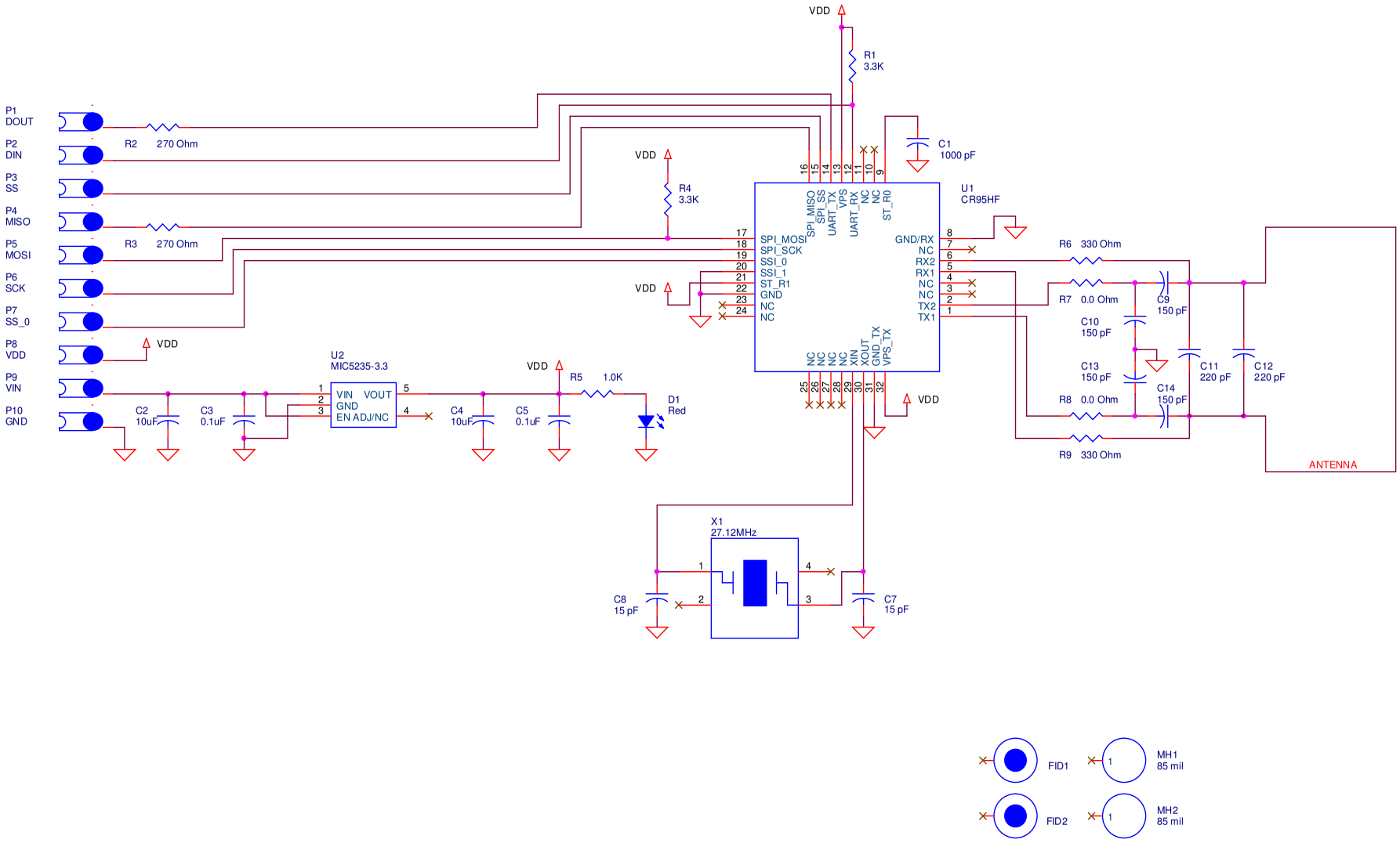
```
void Inventory_Command()
 {
 byte i = 0;

// step 1 send the command
  digitalWrite(SSPin, LOW);
  SPI.transfer(0x00);  // SPI control byte to send command to CR95HF
  SPI.transfer(0x04);  // Send Receive CR95HF command
  SPI.transfer(0x03);  // length of data that follows is 0
  SPI.transfer(0x26);  // request Flags byte
  SPI.transfer(0x01);  // Inventory Command for ISO/IEC 15693
  SPI.transfer(0x00);  // mask length for inventory command
  digitalWrite(SSPin, HIGH);
  delay(1);

// step 2, poll for data ready
// data is ready when a read byte
// has bit 3 set (ex:  B'0000 1000')

  digitalWrite(SSPin, LOW);
  while(RXBuffer[0] != 8)
    {
    RXBuffer[0] = SPI.transfer(0x03);  // Write 3 until
    RXBuffer[0] = RXBuffer[0] & 0x08;  // bit 3 is set
    }
  digitalWrite(SSPin, HIGH);
  delay(1);


// step 3, read the data
  digitalWrite(SSPin, LOW);
  SPI.transfer(0x02);   // SPI control byte for read
  RXBuffer[0] = SPI.transfer(0);  // response code
  RXBuffer[1] = SPI.transfer(0);  // length of data
  for (i=0;i<RXBuffer[1];i++)
      RXBuffer[i+2]=SPI.transfer(0);  // data
  digitalWrite(SSPin, HIGH);
  delay(1);
```

UART Test Software: Open source Visual Studio 2010 software is available for testing the BM019 and to provide a roadmap for developing your own applications.

This software works well with our BM010 USB to serial converter module.

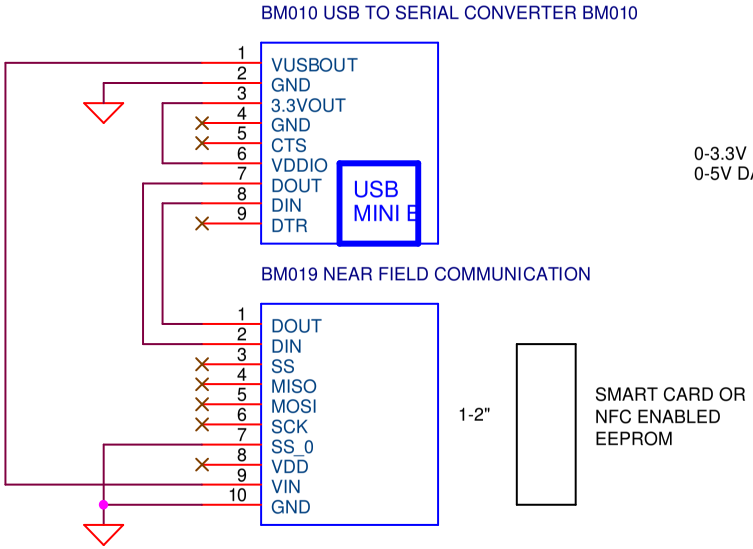


Schematics:

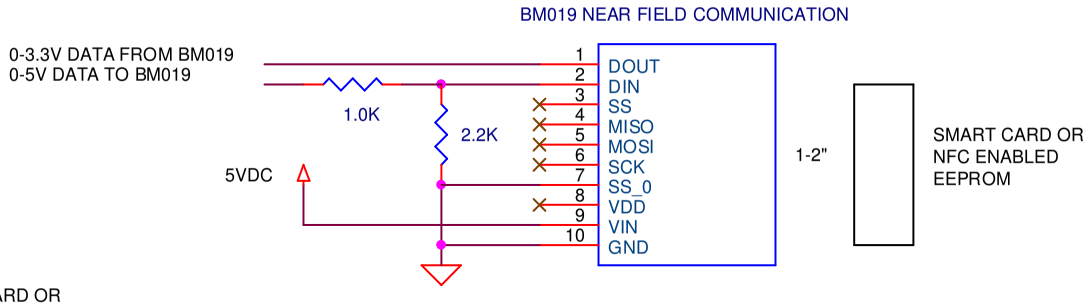

P1
DOUT
P2
DIN
P3
SS
P4
MISO
P5
MOSI
P6
SCK
P7
SS_0
P8
VDD
P9
VIN
P10
GND
R2 270 Ohm
R3 270 Ohm
VDD
R1
3.3K
R4
3.3K
C1
1000 pF
U1
CR95HF
SPI_MOSI
SPI_SCK
SSI_0
SSI_1
ST_R1
GND
NC
NC
GND/RX
NC
RX2
RX1
NC
NC
TX2
TX1
R6 330 Ohm
R7 0.0 Ohm
R8 0.0 Ohm
R9 330 Ohm
C9
150 pF
C10
150 pF
C13
150 pF
C14
150 pF
C11
220 pF
C12
220 pF
ANTENNA
U2
MIC5235-3.3
VIN VOUT
GND
EN ADJ/NC
C2
10uF
C3
0.1uF
C4
10uF
C5
0.1uF
R5 1.0K
D1
Red
X1
27.12MHz
C8
15 pF
C7
15 pF
FID1
FID2
MH1
85 mil
MH2
85 mil

Application Schematics:

USB TO NFC INTERFACE



USING WITH 5V SYSTEMS



Application Schematics: