RemoTI
Host Processor Sample Application and Porting Guide

Document Number: SWRA259

Table of Contents

1 REFERENCES 4
2 INTRODUCTION 5
3 ARCHITECTURE 5
4 HOST PROCESSOR SAMPLE APPLICATION 5
4.1 HARDWARE 5
4.2 SAMPLE APPLICATION SOFTWARE INSTALLATION 6
4.3 BUILDING AND DOWNLOADING 6
4.4 OPERATION 13
5 PORTING RTI SURROGATE 15
5.1 SAMPLE CODE 15
5.2 IMPLEMENTING NPI FUNCTIONS 16
5.2.1 NPI_SendAsynchData 16
5.2.2 NPI_SendSynchData 16
5.2.3 NPI_AsynchMsgCback 17
5.2.4 npiUartDisableSleep 17
5.2.5 Network processor interface frame structure 17
5.3 IMPLEMENTING NETWORK PROCESSOR SYNCHRONIZATION FUNCTIONS 18
5.4 IMPLEMENTING UTILITY FUNCTIONS 19
5.5 PORTING RTI_INIT() FUNCTION 19
6 NETWORK PROCESSOR CONNECTIONS 19
7 GENERAL INFORMATION 21
7.1 DOCUMENT HISTORY 21
8 ADDRESS INFORMATION 21
9 TI WORLDWIDE TECHNICAL SUPPORT 21

Acronyms and Definitions

CERC	Consumer Electronics Remote Control
EM	Evaluation module
IO	Input Output
NP	Network Processor
NPI	Network Processor Interface
RPC	Remote Procedure Call
RTI	RemoTI application framework layer
RTIS	RTI Surrogate
SOC	System On Chip
SOF	Start Of Frame byte
SPI	Serial Peripheral Interface
UART	Universal Asynchronous Receiver-Transmitter

1 References

[1] RemoTI API, SWRA268

[2] RemoTI Network Processor Interface Specification, SWRA271

[3] Flash Programmer User Manual, Rev 1.4 (User Manual Flash Programmer.pdf)

[4] RemoTI-CC2530DK Quick Start Guide, SWRA277

[5] RemoTI Target Emulator User's Guide, SWRU202

[6] http://focus.ti.com/docs/toolsw/folders/print/msp-exp430f5438.html

2 Introduction

RemoTI development kit provides network processor, which interfaces with a host processor. In such a configuration, host processor application controls the network processor to perform pairing, sending and receiving data over the air.
This document explains how to use a sample application code that runs on MSP430F5438 experimenter board, using MSP430F5438 as a host processor connected to a CC2530 as mounted CC2530EM module.
The document also explains what needs to be done to port the host processor code that interface with network processor. Sample code includes a particular module called RTI surrogate which provides the same C function based application programming interface for host processor applications as if it is provided for a standalone application embedded within the RemoTI radio processor. This RTI surrogate module should remain mostly unchanged and the document adds explanation on what need to be done to make this RTI surrogate module run on a different host processor and/or on a different software framework.

3 Architecture

RemoTI network processor provides interface for host processor through UART or SPI as described in [2]. Within network processor, a network processor RTI surrogate module (RTIS_NP) decodes the command frames sent over thru UART or SPI and calls appropriate C function of RemoTI application framework. The C function based application programming interface is defined in [1].
In sample application project, application processor RTI surrogate (RTIS_AP) module abstracts the same C function interface on host processor, enabling application to call C functions remotely instead of building network processor interface directly. Figure 1 shows such architecture.

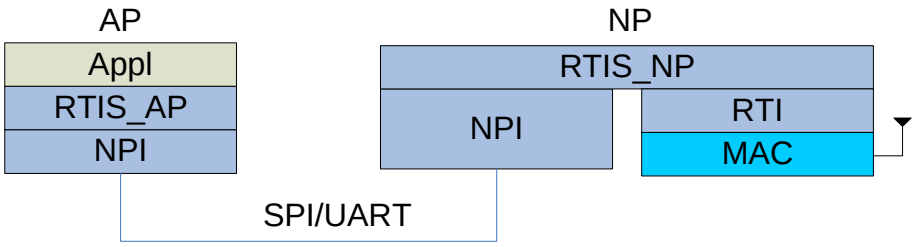


Figure 1. RTIS architecture

Thus, RemoTI API functions are implemented in a surrogate module (RTIS_AP) of the host processor, which calls the remote C function running in network processor through serial interface using network processor interface.
In order to use the application processor RTI surrogate module provided in the sample application for a new host processor or a new host processor platform, functions used by the RTI surrogate, such as network processor interface (NPI) module functions, have to be implemented to build and execute the RTI surrogate module on a new platform.
Subsequent sections describe the functions used by RTI surrogate on host processor.

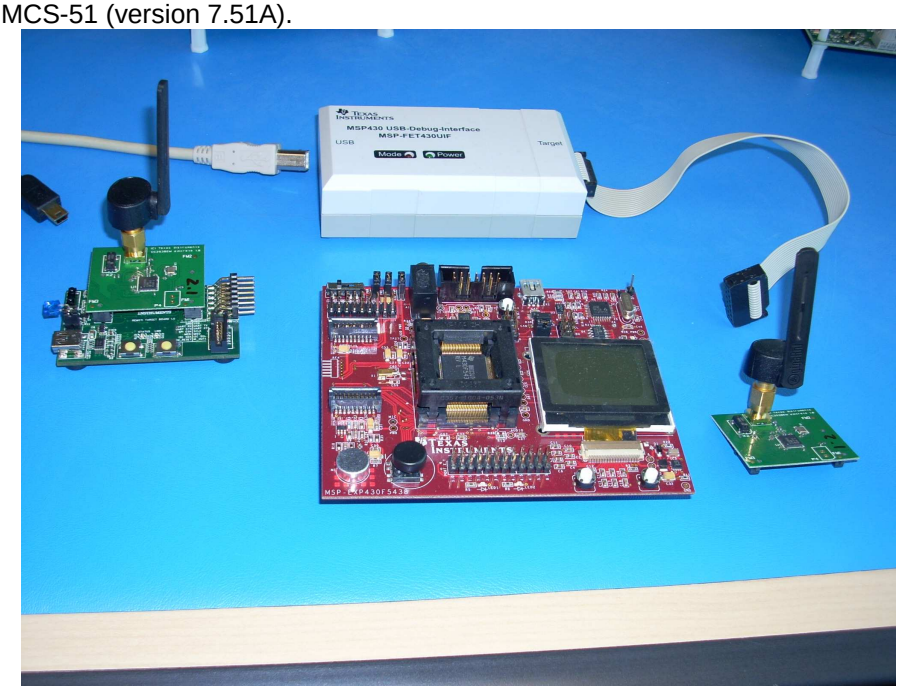
4 Host processor sample application

A sample host application runs on MSP430F5438 experimenter board to demonstrate SPI interface of a RemoTI network processor. The sample application controls network processor as a controller node, like a remote controller. Note that the same network processor can be used either as a controller node or a target node.
This chapter includes information on how to run the sample application.

4.1 Hardware

You need the following hardware to run sample application:

- RemoTI CC2530 Development Kit
- One MSP430F5438 experimenter board.
- MSP430 USB debug interface box (MSP-FET430UIF) and USB/FET cables for the box.
- One extra CC2530EM module with antenna.
- Two AA batteries for MSP430F5438 experimenter board
- PC installed with IAR embedded workbench for both MSP430 (version 4.20) and MCS-51 (version 7.51A).

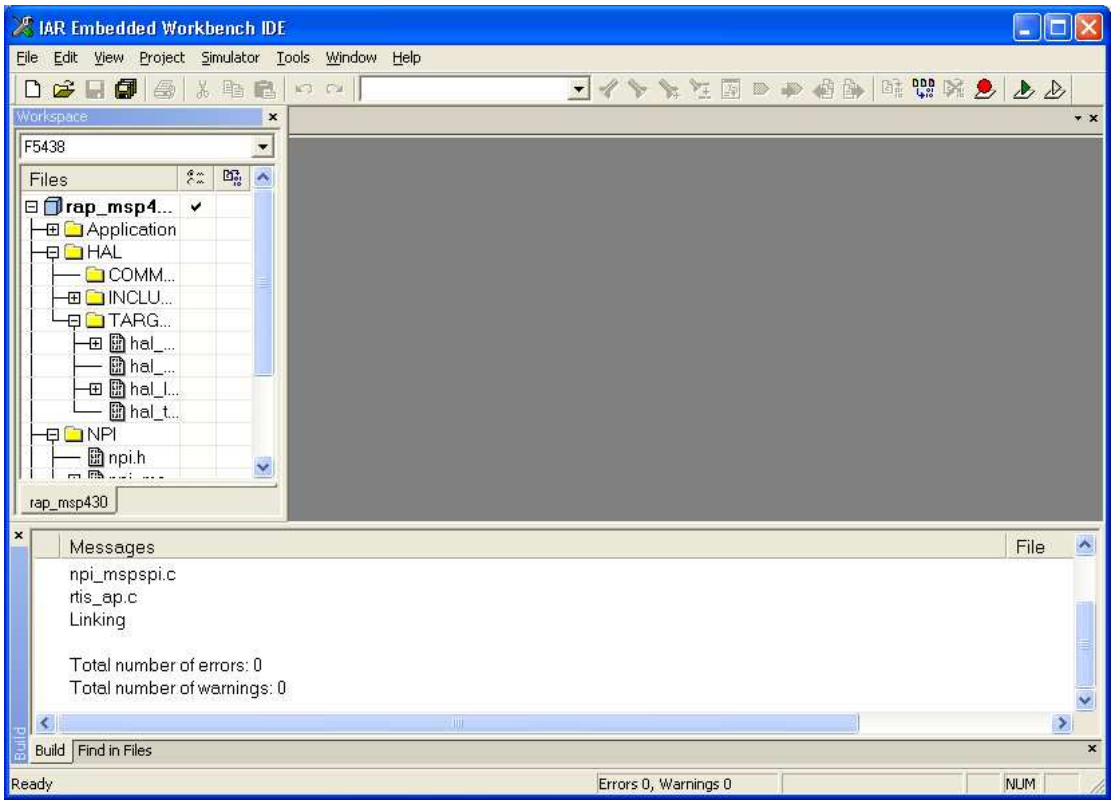


4.2 Sample application software installation

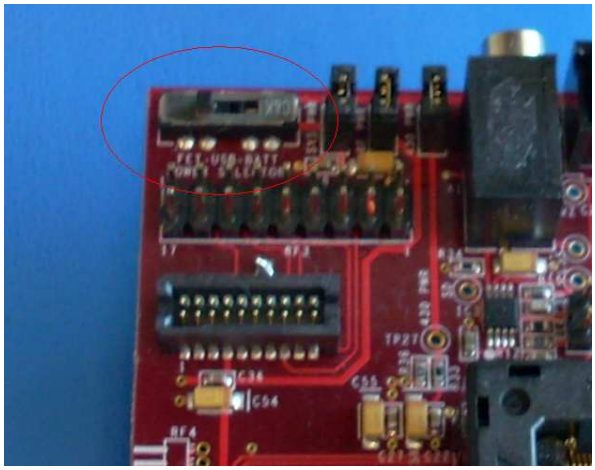
- Install RemoTI software from RemoTI development kit. See [4].
- Run RemoTI-CC2530DK-MSP-1.0.exe file included in the RemoTI_MSP430_sample.zip file and follow the instructions to extract MSP sample application. Note that the sample application software must be installed into a different path than the one where RemoTI development kit software is installed.

4.3 Building and downloading

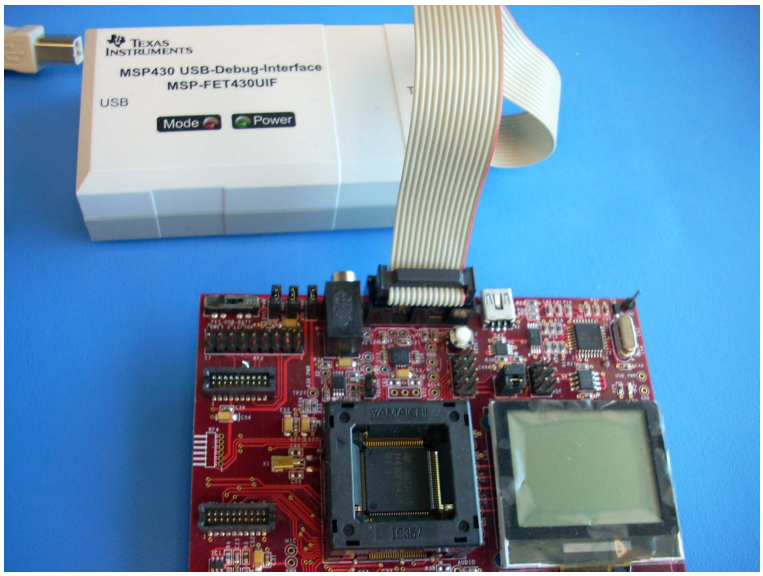
- Run IAR workbench for MSP430.
- Select File -> Open -> Workspace… menu item.
- Browse and select the following file from the sample application installation path: Projects\RemoTI\ApplicationProcessor\MSP430F5438EXP\IAR Project\rap_msp430.eww
- Select Project -> Rebuild All menu item. Note that POWER_SAVING preprocessor definition is included in the default configuration. Do not remove the definition. The option indicates to use power saving mode of MSP and the use of such is necessary for the proper operation of the sample application.
- Wait till build completes. Messages log displays completion of build as follows:



- Place SW1 of MSP430F5438 experimenter board to FET position. This selects power source to JTAG cable.

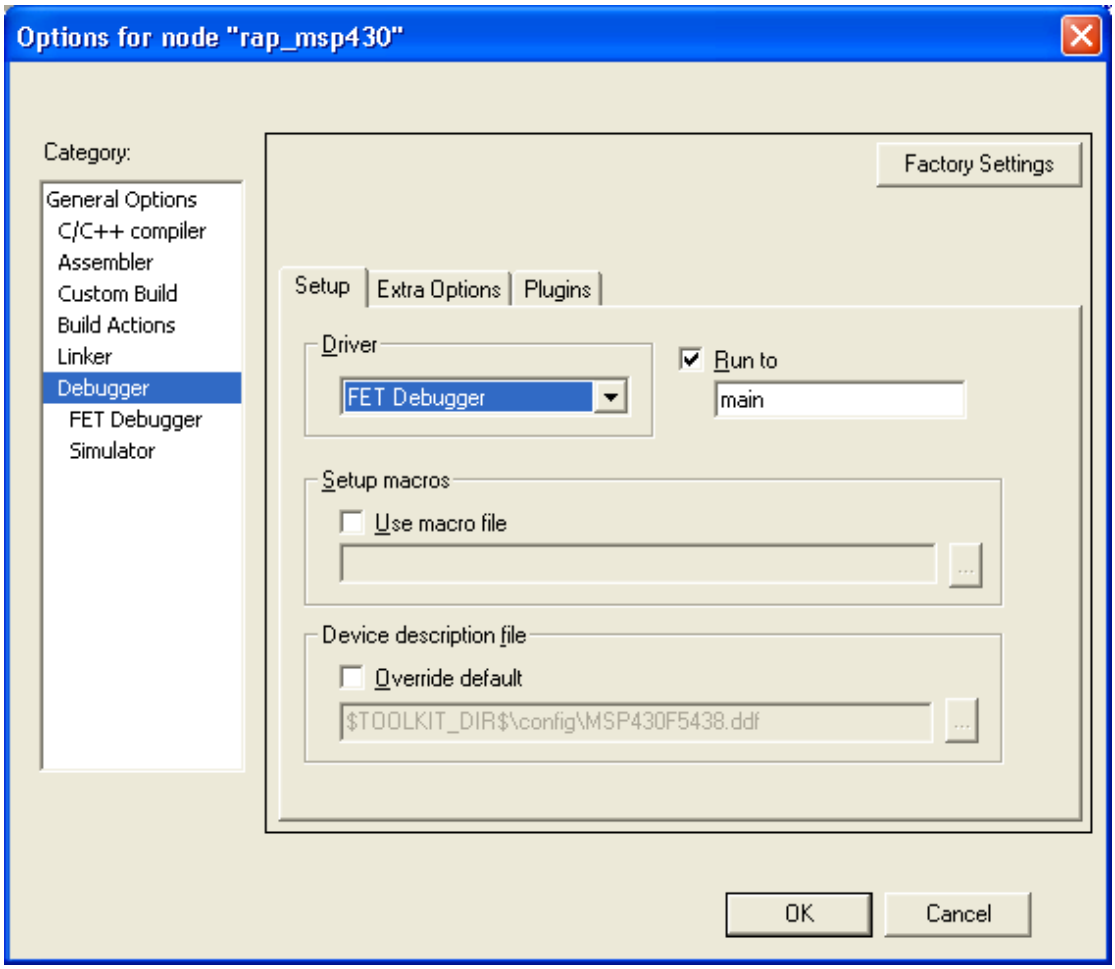


- Connect MSP430 USB debug interface box to the MSP430F5438 experimenter board with a JTAG ribbon cable.

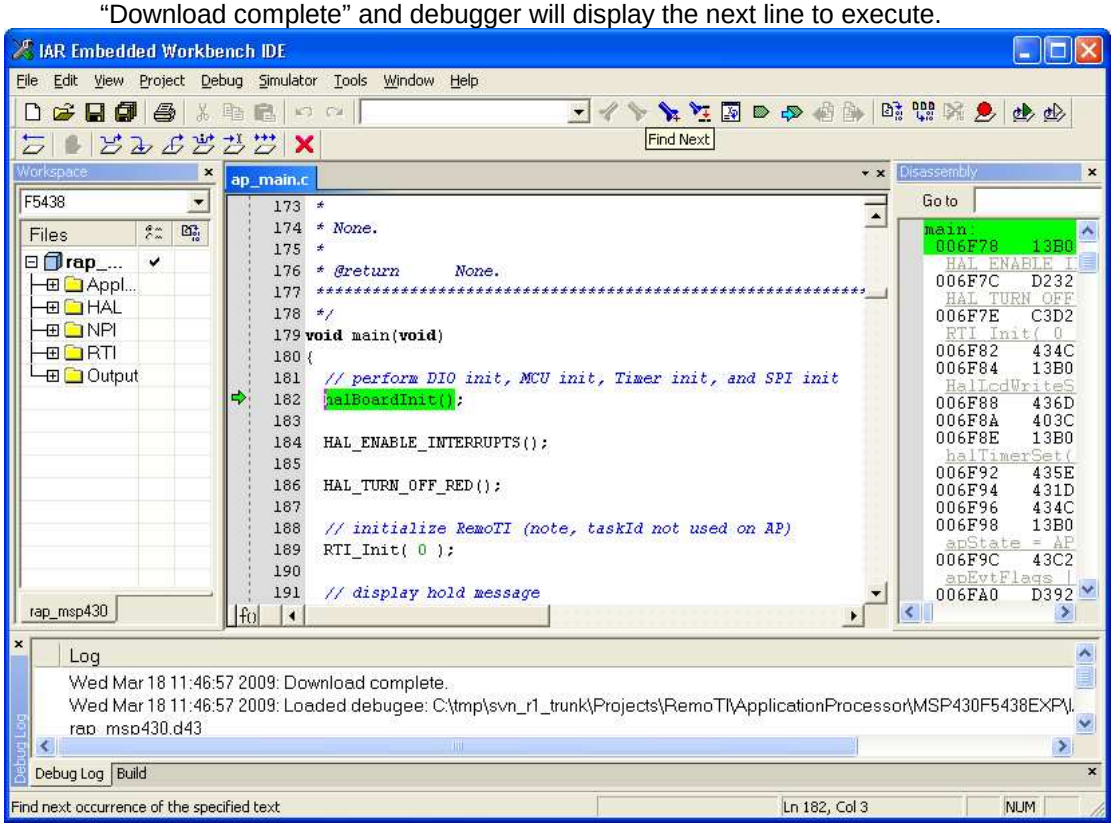


- Connect MSP430 USB debug interface box to the PC with a USB cable. Note that mode LED (red) turns on for a short while and then it turns off and power LED (green) turns on, on the MSP430 USB debug interface box. If power LED is not turned on, disconnect USB and connect it again and repeat it till power LED is turned on correctly.

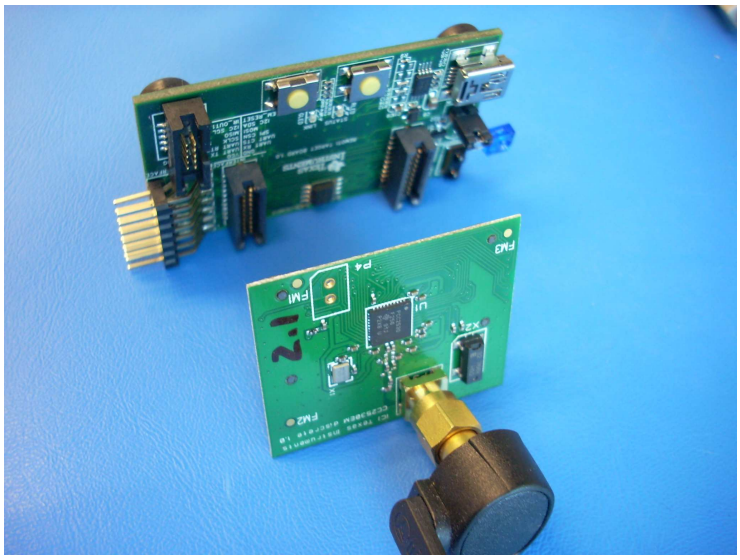
- Select Project->Options menu item. Then, select Debug category. Check if the Driver pull down selection selects FET debugger.



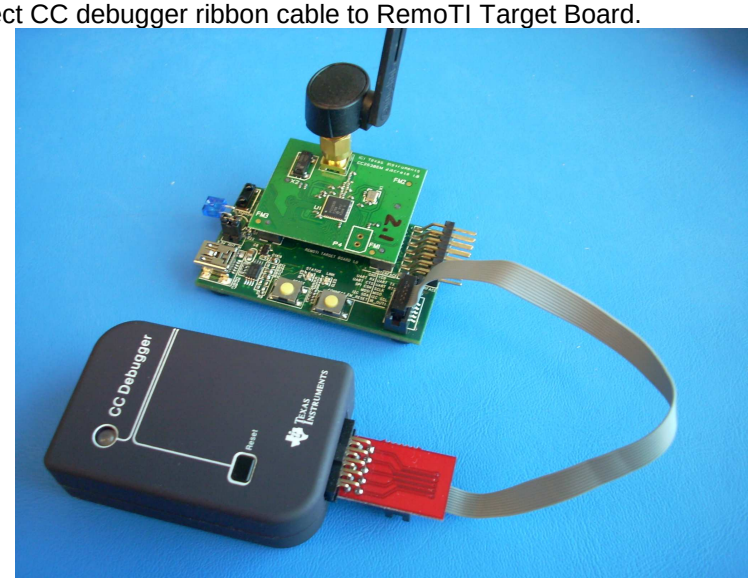
- From IAR embedded workbench, select Project -> Download and Debug menu item. Wait until downloading completes. When download completes, log window will show “Download complete” and debugger will display the next line to execute.



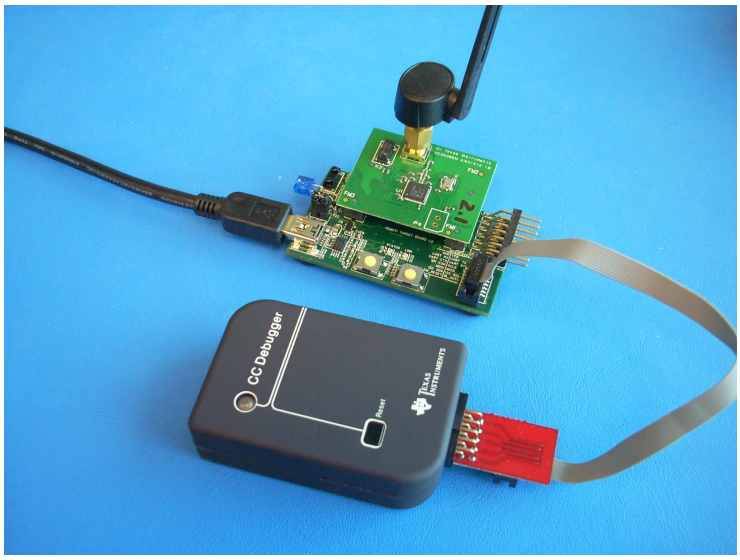
- Select Debug -> Stop Debugging menu item.
- Disconnect MSP430 USB Debug interface from PC.
- Disconnect JTAG ribbon cable of MSP430 USB Debug interface from MSP430F5438 experimenter board. You are complete with programming MSP with sample host application. Close IAR workbench.
- Now you have to program the RemoTI network processor image to the extra CC2530EM module. Remove USB connector from RemoTI Target Board to shut down power. Remove CC debugger ribbon cable connector from RemoTI Target Board as well if it is connected.
- Remove default CC2530EM module from the RemoTI Target Board that came with the RemoTI development kit. We need to use this default CC2530EM module, as is, later.



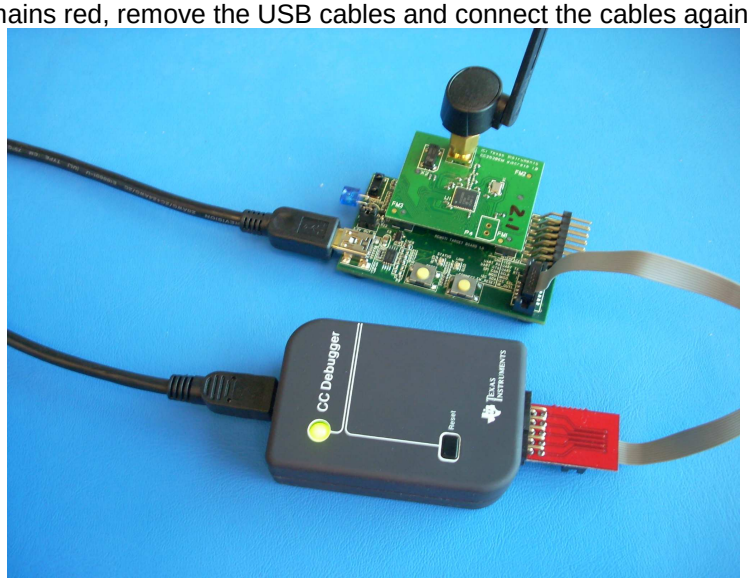
- Install the extra CC2530EM module into the RemoTI Target Board.
- Connect CC debugger ribbon cable to RemoTI Target Board.



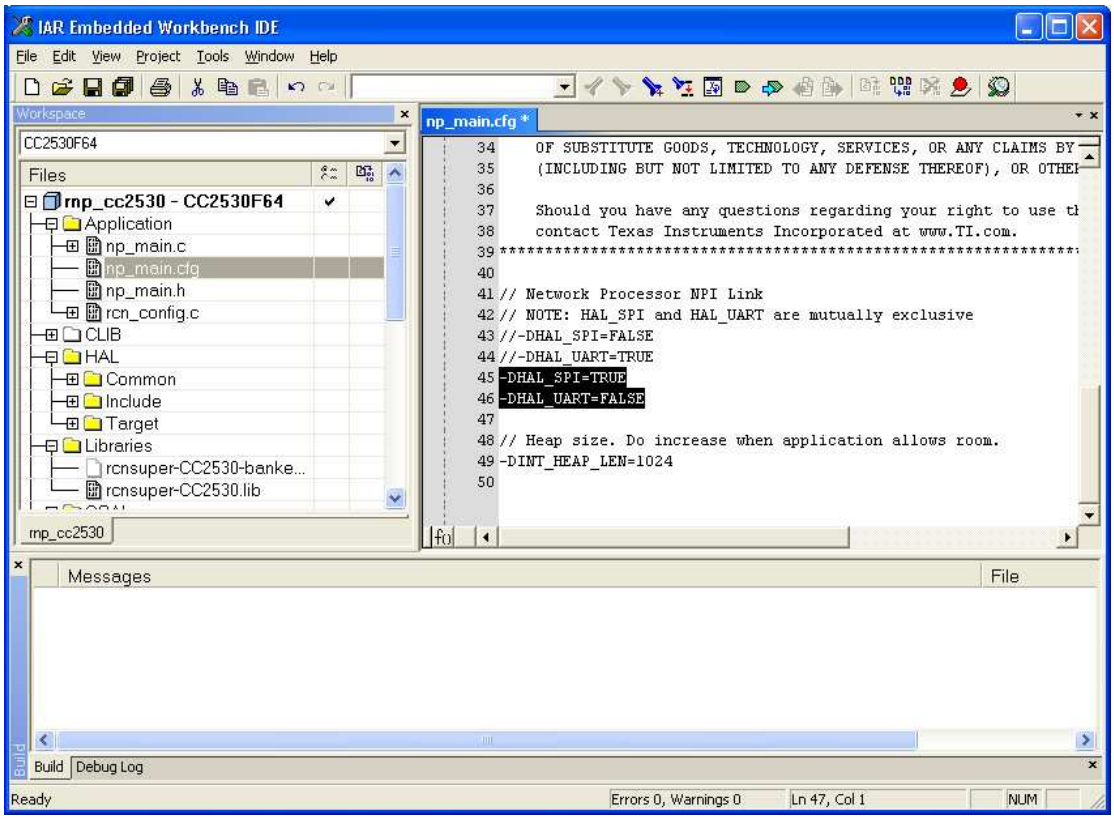
- Connect USB cable from PC to the RemoTI Target Board.



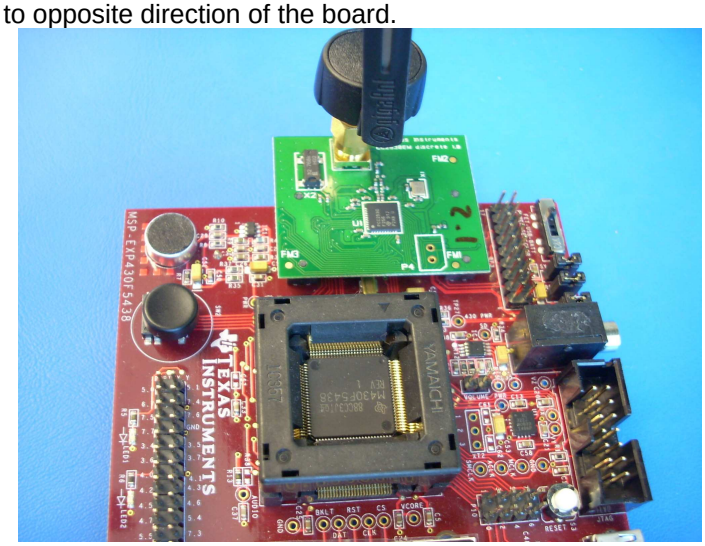
- Connect USB cable from PC to CC debugger. CC debugger should turn on green LED. If it turned on RED instead, preset RESET button on CC debugger. If the LED still remains red, remove the USB cables and connect the cables again.



- Open IAR workbench for MCS-51.
- Select File -> Open -> Workspace… menu item.
- Browse and select the following file from RemoTI development kit installed path: <RemoTI kit installation path>\Projects\RemoTI\RNP\CC2530EB\rnp_cc2530.eww
- From workspace window, double click on Application\np_main.cfg to open the file for editing.



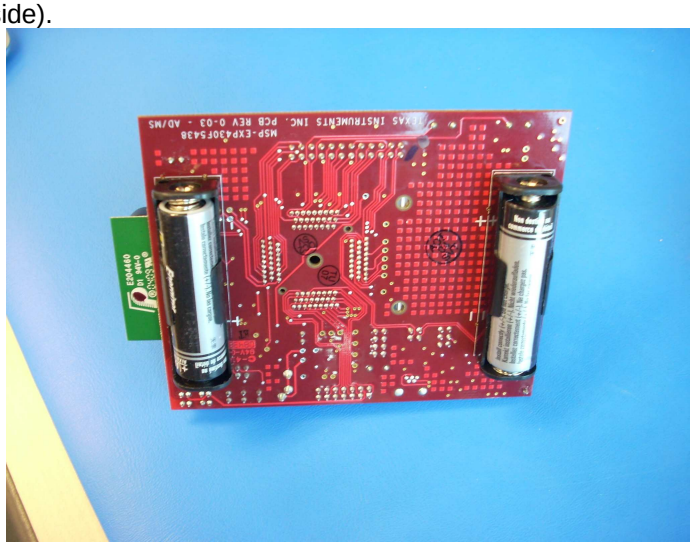
- Within the file, comment out –DHAL_SPI=FALSE and –DHAL_UART=TRUE lines and instead uncomment –DHAL_SPI=TRUE and –DHAL_UART=FALSE lines. This selects SPI as compile configuration.
- Select Project -> Rebuild All menu item and wait till build completes.
- Select Project -> Download and Debug menu item. Wait until downloading completes.
- Select Debug -> Stop Debugging menu item.
- Close IAR workbench.
- Disconnect USB cable from CC debugger.
- Disconnect USB cable from RemoTI target board.
- Remove the extra CC2530EM module that you just programmed from RemoTI Target Board. This module is now programmed with network processor with SPI configuration.
- Install the original CC2530EM module back on the RemoTI Target Board.
- Mount newly programmed extra CC2530EM module onto RF1, RF2 SMD connector sockets of MSP430F5438 experimenter board. EM module antenna socket should be heading to opposite direction of the board.



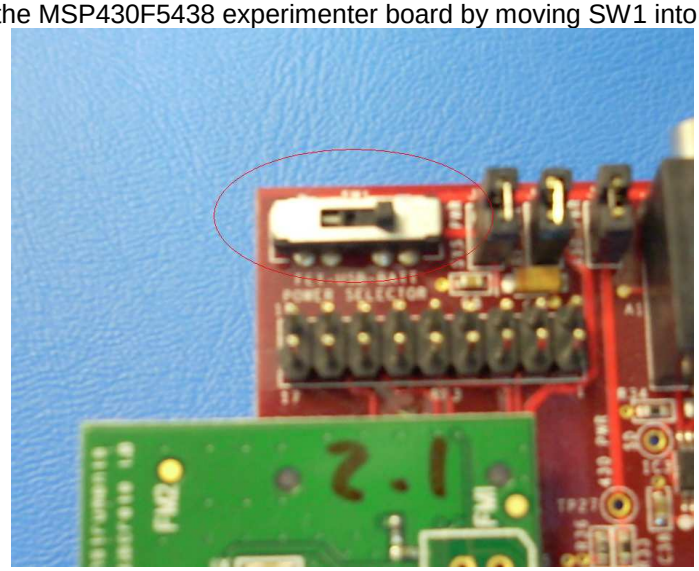
- Now you have completely programmed host processor and network processor on MSP430F5438 experimenter board.

4.4 Operation

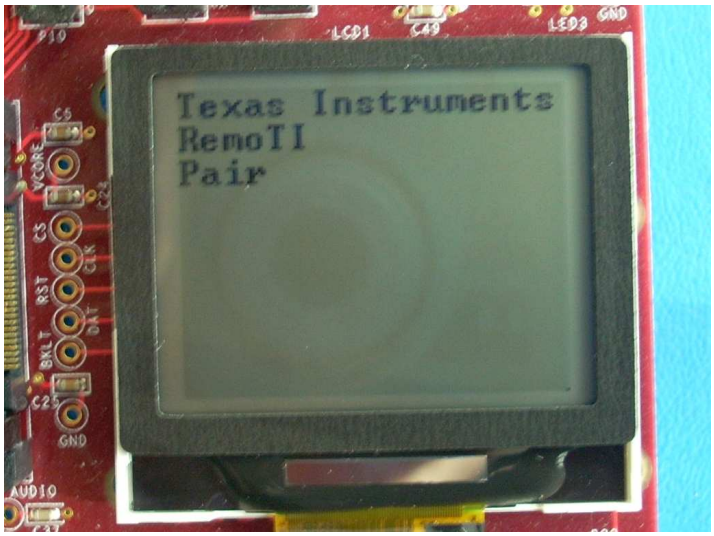
- Insert two AA batteries to the battery bay of MSP430F5438 experimenter board (bottom side).



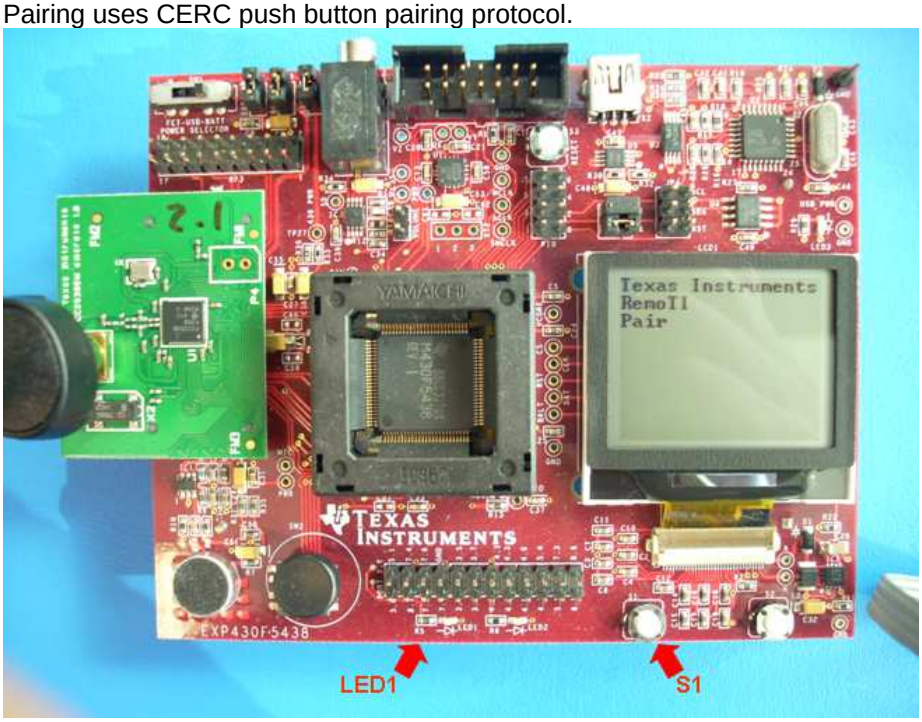
- Connect USB cable to RemoTI Target Board which must have original CC2530EM module on it.
- Run Target Emulator on PC to get RemoTI Target Board ready. See [4] and [5] for instructions on how to operate Target Emulator.
- Turn on the MSP430F5438 experimenter board by moving SW1 into “BATT” position.



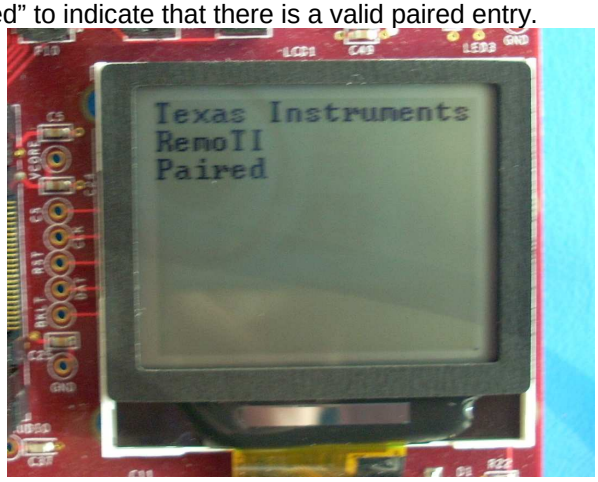
- Unless RemoTI network processor (CC2530) is already paired with a target, the LCD on MSP experimenter board will display “Pair” message to indicate that it is ready to pair.



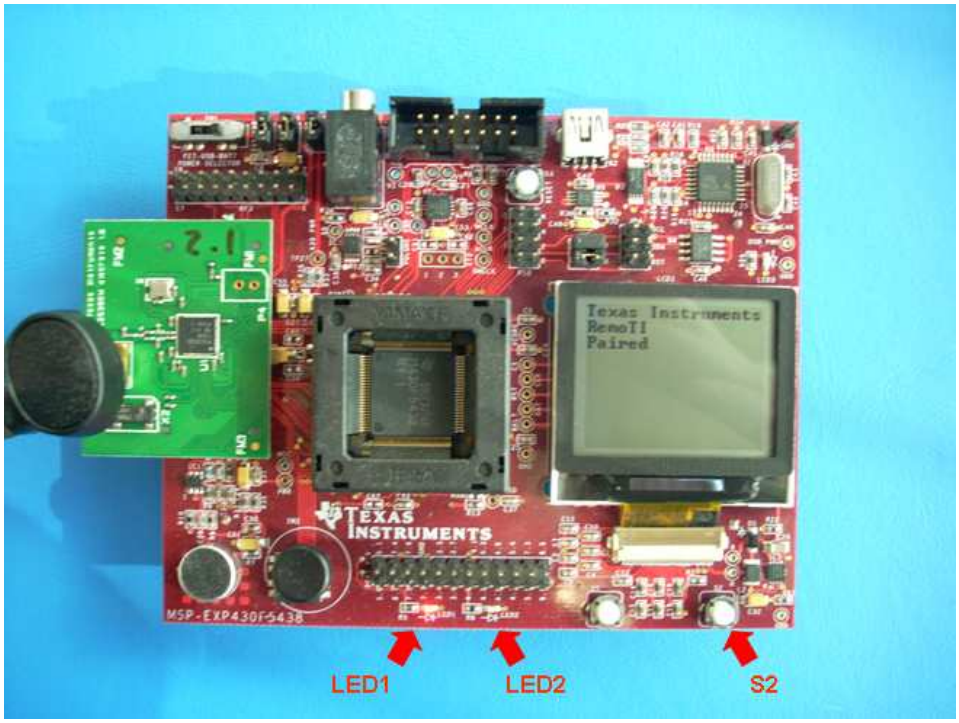
- Perform pairing by pressing “S1” key once. LED1 on MSP experimenter board will blink till the pairing completes or fails. Operate the target emulator to pair the device. Pairing uses CERC push button pairing protocol.



- If pairing succeeds or there is already a valid pairing entry, LED1 on MSP experimenter board will be turned on and LCD on MSP experimenter board will display “Paired” to indicate that there is a valid paired entry.



- If pairing failed, LCD on MSP experimenter board will display “Failed” for a short while and then display status whether there is existing pairing or not. If no device was ever paired, LCD will display “Pair” and LED1 will be turned off. If any device was ever paired before this particular pairing attempt, LCD will display “Paired” and LED1 will be turned on.
- Once, there is a valid pairing entry, press “S2” key once to send data to a paired target. LED2 on MSP experimenter board will be turned on while the data is being transmitted. Data will alternate among CERC key pressed command for numeric key 1 thru 9 and 0.



When the board is powered up, the MSP sample application will select the last paired entry for default destination. To erase all pairing entry, turn off the MSP board by moving SW1 to “USB” position (assuming USB cable is detached) and keep “S1” key pressed down while moving SW1 to “BATT” position again.

If LCD displays “Failed”, it indicates that network processor interface failed. LCD may display “Init” forever in certain occasions when network processor interface failed in the initialization phase. If such incidents occur, please check whether CC2530 was mounted correctly on MSP430F5438 board and retry programming CC2530EM.

Joystick keys are also used in MSP sample application. Moving joystick to left, right, up, down position or pressing down the joystick will display text on LCD corresponding to such move. It does not perform any RemoTI packet transmission but the sample code was created just to demonstrate joystick.

5 Porting RTI surrogate

5.1 Sample code

The MSP sample code comprises of the files explained in the following table.

File name	Description
hal_board.c	Hardware abstraction layer
hal_lcd.c, hal_lcd.h	LCD driver
hal_types.h, _hal_types.h	Data type definitions

File name	Description
hal_defs.h	Generic macro definitions
hal_board.h, hal_board_cfg.h	Board configuration header file
hal_rpc.h	Remote procedure call enumerations
rcn_attribs.h	RemoTI network layer attribute enumerations and structure definition
rti.h	RemoTI application framework interface header file
rti_constants.h	RemoTI constant definition
saddr.h	Extended address data structure and constants
ap_main.c, ap_main.h	Sample application main code
ap_main.cfg	Sample application configuration
rap_msp430.ewp	IAR project file for the sample application
rap_msp430.eww	IAR workspace file for the sample application
npi_mspspi.c	NPI module implementation using SPI
npi.h	NPI module function interface definition
rcns_ap.c	Network layer surrogate. This module is not used by this sample application. It is provided as a reference in case network layer interface is to be used from host processor.
Rtis.h	RTI surrogate interface header file
rtis_ap.c	RTI surrogate module

5.2 Implementing NPI functions

RTI surrogate module uses a set of platform-dependent functions to implement its functionality. Ideally, RTI surrogate module itself (rtis_ap.c) does not have to be modified across different platform.
Among the platform dependent functions that RTI surrogate module uses are the NPI functions which abstract underlying transport layer of serial communication between the application processor and the network processor.
NPI functions are summarized in the following subsections. Note that the C functions listed below signify only the interface between RTI surrogate module and NPI module but in fact the entire UART or SPI NPI transport protocol described in [2] has to be ported to the new platform besides the interface alone.
The sample implementation of NPI (npi_mspspi.c) does not use any operating system. The sample NPI module and hardware abstraction layer it is using was implemented in monolithic single thread context from main routine, with no operating system. It is not a typical host processor environment but it is done so in order to be independent of a variety of operating systems, software frameworks and platforms available for host processor.

5.2.1 NPI_SendAsynchData

```
void NPI_SendAsynchData( npiMsgData_t *pMsg )
```

The above function abstracts building and sending an asynchronous message to the network processor. pMsg is a pointer to a message structure which includes subSys (subsystem identifier), cmdId (command identifier), pData (pointer to payload data buffer) and len (length in bytes of the payload data). RTI surrogate module fills in all parameters of npiMsgData_t structure pointed by pMsg before calling this function. The function implementation should copy the content of this structure if it needs to access the content after the function returns.

5.2.2 NPI_SendSynchData

```
void NPI_SendSynchData( npiMsgData_t *pMsg )
```

The above function abstracts building and sending a synchronous message towards network processor and receiving a synchronous response message from network processor. The function is a blocking function till it receives the synchronous response message from network processor or till implementation specific timeout corresponding to round trip of UART plus network processor execution time. npiMsgData_t structure pointed by pMsg argument is filled

in by the RTI surrogate module before calling this function but also will have to be overwritten by this function before the function returns. Content of the received synchronous response message from network processor has to overwrite the structure and the payload buffer pointed by the structure. If an error such as timeout occurs, the first byte of the payload buffer, which is used as status indicator, has to be set to 0xFF.

5.2.3 NPI_AsynchMsgCback

```
void NPI_AsynchMsgCback( npiMsgData_t *pMsg )
```

The above function is implemented in RTI surrogate module. The NPI implementation of the host application has to call the above function to forward a received NPI asynchronous request message from the network processor. Hence, npiMsgData_t structure has to be filled in with the content of the received asynchronous message and argument pMsg has to be set to point to such structure. Memory management of npiMsgData_t structure used for the call is completely up to the NPI implementation and it can de-allocate the structure after this function returns. Note that NPI_AsynchMsgCback() function may call NPI_SendAsynchData() function or NPI_SendSynchData() function from the same thread and hence, NPI implementation should be designed to allow such calls.

5.2.4 npiUartDisableSleep

```
void npiUartDisableSleep( void )
```

The above function abstracts network processor UART receiver wakeup protocol. The function is required if UART transport protocol is chosen as interface between the host processor and the network processor.
The function must implement sending a wakeup null character (0x00) and blocking till the host processor receives a response null character (0x00) from network processor, hence completing the wakeup handshake protocol.

5.2.5 Network processor interface frame structure

The above three functions have to translate between C function API and network processor interface, by encoding or decoding network processor interface command frames and handling or issuing corresponding C function calls.

Network processor interface frame format is described in [2]. Described below is how the frame fields map to C function arguments.

Bytes: 1	2	0-123
Length	Command	Data

Figure 2 – General frame format (See [2])

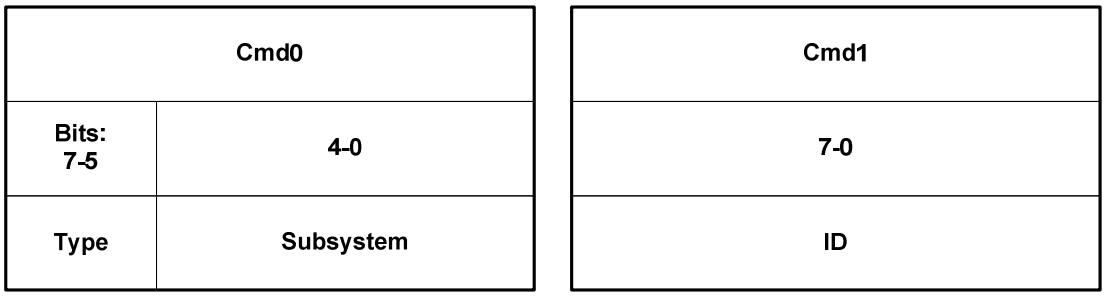


Figure 3 – Command field format (See [2])

- Length

Length field should be copied from 'len' field of npiMsgData_t structure when building a frame to send over to network processor and this field value of received synchronous response command packet or asynchronous request command packet from network processor should be copied to the structure.

- Command Type and Subsystem Identifier (Cmd0 field)

Command Type and Subsystem Identifier field comprises of 5 least significant bits of subsystem identifier and 3 most significant bits of command field type as follows:

```
#define RPC_CMD_POLL        0x00
#define RPC_CMD_SREQ        0x20
#define RPC_CMD_AREQ        0x40
#define RPC_CMD_SRSP        0x60
```

See [2] for details on command type field and subsystem identifier field.

NPI_SendAsynchData() function must be implemented to build a frame with command type RPC_CMD_AREQ.
NPI_SendSynchData() function must build a frame with command type RPC_CMD_SREQ and expect a response frame with command type RPC_CMD_SRSP.

NPI implementation should not care which actual network processor interface commands are mapped to synchronous or asynchronous requests. Such is determined by RTI surrogate module and proper function between NPI_SendAsynchData() and NPI_SendSynchData is selected and called.

- Command Identifier (Cmd1 field)

Command identifier field of a network processor interface command packet corresponds to cmdId field of npiMsgData_t structure.

- Payload (Data field)

Payload should be filled in with the byte array pointed by pData field of the passed structure to build a request frame to send to network processor and the payload of received response message or asynchronous message from network processor should be copied to the pData field of the C structure.

5.3 Implementing network processor synchronization functions

RTI surrogate module uses two platform dependent functions to synchronize network processor upon boot up.

```
Void halResetSlave(void)
```


The above function abstracts resetting the network processor chip, through RESET_N line assertion. The function is called when RTI surrogate module initializes.

```
Void npiSynchSlave(void)
```

The above function abstracts synchronizing with network processor wakeup. By implementation specific means, the function is supposed to block till network processor is up and running, ready for message transaction with host application. Implementation specific means could be simply waiting for a certain known duration of time or it could be using IO pins to handshake the readiness.

5.4 Implementing utility functions

Other platform dependent functions are also used to implement RTI surrogate module.

```
Void *msg_memcpy( void *dst, const void *src, uint16 len )
```

The above function is abstraction of memory block copy function, the same as C library memcpy function.

```
Void rtisFatalError( uint8 status )
```

The above function is called when a fatal error occurs in RTI surrogate module.

```
Void NPI_Init( void )
```

The above function is called when RTI surrogate module initializes as a placeholder for host processor framework specific NPI initialization such as creating new threads to handle NPI transport layer message transmission and reception.

```
Void halDelay( uint8 msecs, uint8 sleep )
```

The above function is called to block the caller (RTI surrogate module) for certain duration of time. Msecs parameter indicates time duration in milliseconds and sleep indicate whether to block (1) or not to block (0).

```
Uint8 halDelayDone( void )
```

The above function is called to poll whether the time duration set up in halDelay() call without blocking has expired or not. The non-zero return value shall indicate that the time has expired.

5.5 Porting RTI_Init() function

RTI_Init() function in surrogate module performs initialization of the surrogate in a platform specific way. The function calls NPI_Init(), halResetSlave() and npiSynchSlave() functions from within and it also generates a TEST_PING_REQ command at the end. Host application or host application framework has to call this function to initialize RTI surrogate module.

6 Network processor connections

Table 1 explains how the CC2530 is connected to the MSP430F5438 when you mounted CC2530EM module on MSP430F5438 experimenter board.

Table 1 – SPI connection between network processor and MSP430F5438

SPI I/F and extra	CC2530 port	MSP430F5438 port
MOSI	P1.6	P3.1
MISO	P1.7	P3.2
SS	P1.4	P3.0
SCLK	P1.5	P3.3

MRDY (Master Ready)	P0.3	P1.6
SRDY (Slave Ready)	P0.4	P1.4
Reset network processor	RESETN	P1.2

7 General Information

7.1 Document History

Revision	Date	Description/Changes
1.0	2009.03.31	Initial release.
1.1	2009.05.07	IAR versions are mentioned. Preprocessor definition is explained. Installation instruction and pictures are updated to match the updated sample code.

8 Address Information

Texas Instruments Norway AS
Gaustadalléen 21
N-0349 Oslo
NORWAY
Tel: +47 22 95 85 44
Fax: +47 22 95 85 46
Web site: http://www.ti.com/lpw

9 TI Worldwide Technical Support

Internet

TI Semiconductor Product Information Center Home Page: support.ti.com
TI Semiconductor KnowledgeBase Home Page: support.ti.com/sc/knowledgebase

Product Information Centers

Americas

Phone: +1(972) 644-5580
Fax: +1(972) 927-6377
Internet/Email: support.ti.com/sc/pic/americas.htm

Europe, Middle East and Africa

Phone:

Belgium (English)	+32 (0) 27 45 54 32
Finland (English)	+358 (0) 9 25173948
France	+33 (0) 1 30 70 11 64
Germany	+49 (0) 8161 80 33 11
Israel (English)	180 949 0107
Italy	800 79 11 37
Netherlands (English)	+31 (0) 546 87 95 45
Russia	+7 (0) 95 363 4824
Spain	+34 902 35 40 28
Sweden (English)	+46 (0) 8587 555 22
United Kingdom	+44 (0) 1604 66 33 99

Fax: +49 (0) 8161 80 2045
Internet: support.ti.com/sc/pic/euro.htm

Japan

Fax	International	+81-3-3344-5317
	Domestic	0120-81-0036
Internet/Email	International	support.ti.com/sc/pic/japan.htm
	Domestic	www.tij.co.jp/pic

Asia

Phone	International	+886-2-23786800
	Domestic	Toll-Free Number
	Australia	1-800-999-084
	China	800-820-8682
	Hong Kon	800-96-5941
	India	+91-80-51381665 (Toll)
	Indonesia	001-803-8861-1006
	Korea	080-551-2804
	Malaysia	1-800-80-3973
	New Zealand	0800-446-934
	Philippines	1-800-765-7404
	Singapore	800-886-1028
	Taiwan	0800-006800
	Thailand	001-800-886-0010
Fax		+886-2-2378-6808
Email		tiasia@ti.com or ti-china@ti.com
Internet		support.ti.com/sc/pic/asia.htm

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve 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 (also referred to herein as “components”) 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 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.

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right 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.

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

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

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 which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm 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 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 a special agreement specifically governing such use.

Only those TI components which 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 which have ***not*** been so designated is solely at the Buyer's risk, and that 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.

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		

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