Wireless Sensor Networks

PRACTICAL TRAINING WITH CC430F6137

OTH Regensburg | E-lab
MATTHIAS WAGNER | ALBERT MARTINEZ
PROF. DR. MARTIN SCHUBERT

Contents

List of Figures

List of tables

1 Introduction to CC430 target board

The WSN-target boards are created out of Texas Instruments Chronos Watch, which is a development kit for wireless sensor communication. The WSN-target board is based on a CC430F6137 SoC. It integrates a MSP430 microcontroller and a CC1101 Radio module with AES-128 encryption technology.

This development kit also provides a debugger in order to program the microcontroller. For programming the device, TI provides an IDE (integrated development environment) which is the CCS (Code composer Studio)[1]. (It's also possible to use IAR[2] or the GCC compiler[3])

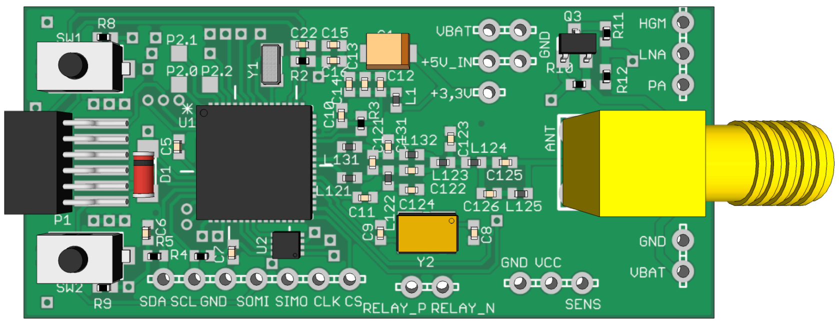


Figure 1- CC430 target board

In this practical training you will use:

- 2 FET (Flash emulation tool) Debuggers
- 1 WSN-target board on a training adapter
- 1 WSN-target board (with battery holder)

As this is a development tool thought to learn wireless communication mechanisms every group is provided with 2 boards. Every board can act as a receiver and transmitter.

[1] CCS: http://www.ti.com/tool/ccstudio
[2] IAR: https://www.iar.com/iar-embedded-workbench/
[3] GCC: http://www.ti.com/tool/msp430-gcc-opensource

CC430F6137

The Texas Instruments CC430 Family of ultra-low-power microcontroller system-on-chip (SoC) with integrated RF transceiver cores consists of several devices featuring different sets of peripherals targeted for a wide range of applications. The architecture, combined with five low-power modes, is optimized to achieve extended battery life in portable measurement applications. The device features the powerful MSP430 16-bit RISC CPU, 16-bit registers, and constant generators that contribute to maximum code efficiency.

The CC430 family provides a tight integration between the microcontroller core, its peripherals, software, and the RF transceiver, making these true SoC solutions easy to use as well as improving performance.

Capabilities:

- 20 MHz system frequency (up to)
- 32 kB of Flash memory
- 4 kB RAM
- 32 GPIO
- 2 16 Bit Timers
- 12 Bit ADC channels
- 128 Bit AES Security Encryption Coprocessor
- 32 Bit Hardware Multiplier
- Hardware Real-Time Clock (RTC)
- Communication per UART, SPI and I^{2}C.

CC1101 Radio Chip

The radio chip is a sub-1 GHz transceiver designed for very low-power wireless applications, providing extensive hardware support for packet handling, data buffering, burst transmissions, clear channel assessment, link quality indication and wake-on-radio.

The main operating parameters and the 64-byte transmit/receive FIFOs are controlled via the internal Bus.

Capabilities:

- High Sensitivity
- Low power consumption (14,7mA in RX mode)
- Programmable data rate from 0,6 to 600 kbps
- Frequency bands: 300 – 348 MHz, 387 – 464 MHz, 779 – 928MHz

2 Introduction to MSP430

In this chapter all the capabilities of the CC430F6137 will be shown**.** The CC430 uses an internal von-Neuman communication. The Internal architecture of the CC430 looks like following:

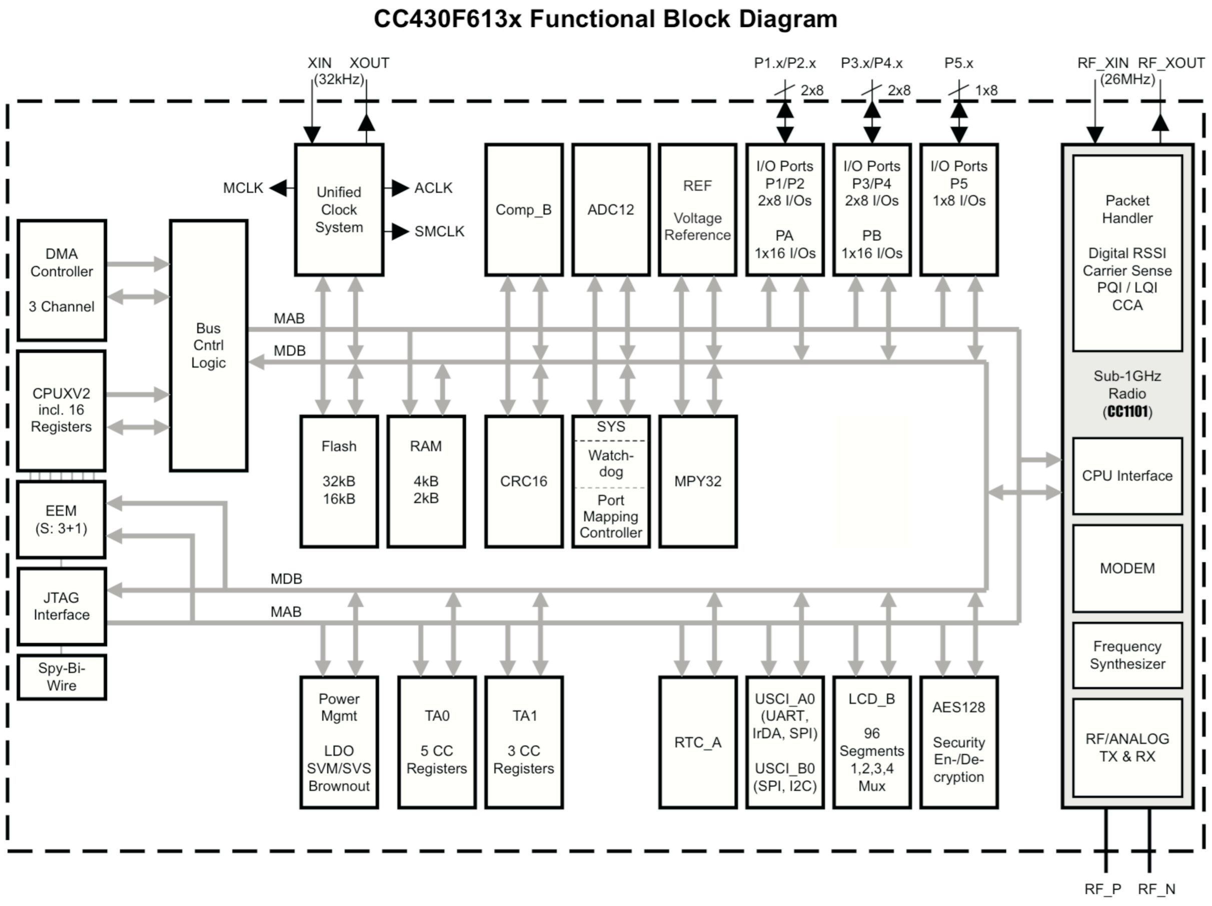


Figure 2 - Block Diagram

2.1 Clocks

The universal clock system (UCS) is the heart of the MSP430 SoC as it garenties a synchronous operation of all connected modules. The UCS sets the frequency of the CPU, which means the speed at which the CPU will operate. In CC430 family there are 3 ways to supply a clock source:

- Internal oscillators
- External oscillators
- External crystals

CC430 has three internal clock signals, the user can select the best balance between performance and low power consumption. The basic clock module can be configured to operate without any external components, with one or two external crystals, or with resonators, under full software control.

The basic clock module includes up to five clock sources:

• **XT1CLK:** Low-frequency oscillator that can be used with low-frequency *32768-Hz watch crystals*

• **VLOCLK:** Internal very low power, low frequency oscillator with 10 kHz typical frequency

• **REFOCLK:** Internal, trimmed, low-frequency oscillator with 32768 Hz typical frequency, with the ability to be used as a clock reference into the FLL

• **DCOCLK:** Internal digitally-controlled oscillator (DCO) that can be stabilized by the FLL

• **XT2CLK:** RF XT2 oscillator required for radio functionality

Three clock signals are available from the UCS:

- **ACLK**: Auxiliary clock. (default: XT1 = 32768 Hz)
- **MCLK**: Master clock. MCLK is used by the CPU (default:1 MHz)
- **SMCLK**: Sub-main clock. (default: 1 MHz)

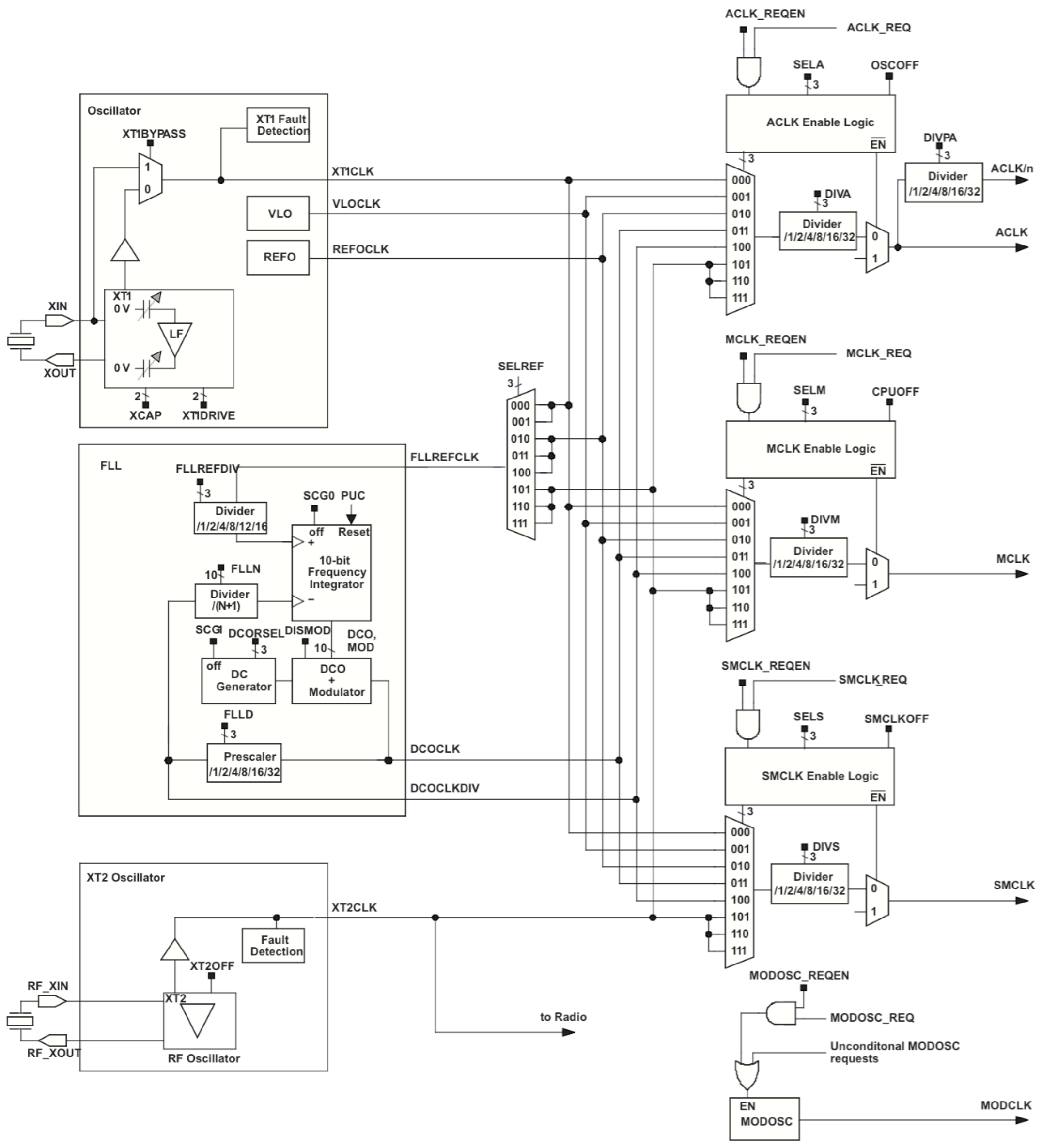


Figure 3 - UCS Block Diagram

The Block Diagram in Figure 3 seems pretty complicated at first, but isn't to hard to understand if you look at the 6 blocks. On the left you can see the 3 Oscillators (XT1, FLL (DCO), XT2). These three blocks output the 5 raw clock speeds. The 3 clock signal modules on the right side scale down the raw clocks to the 3 output clocks.

The DCO is usually the source clock for the main clock which sets the frequency of the CPU.

2.2 General Purpose Input Output (GPIO)

Digital pins are the main communication between a microcontroller and the peripheries inside a system. They have two conditions, high or low. MSP430 devices have up to eight digital I/O ports implemented, P1 to P8 depending on the specific device. Each port has up to eight I/O pins.

Ports P1 and P2 have interrupt capability. Each interrupt for the P1 and P2 I/O lines can be individually enabled and configured to provide an interrupt on a rising or falling edge of an input signal.

The digital I/O features include:

- Independently programmable individual I/Os
- Any combination of input or output
- Individually configurable P1 and P2 interrupts
- Independent input and output data registers
- Individually configurable pullup or pulldown resistors
- Individually configurable pin-oscillator function (some MSP430 devices)

The relation between the different ports in a pin and its notation in hexadecimal is the following:

Port	Hexadecimal	Code Define
P1.0	0x01	BIT0
P1.1	0x02	BIT1
P1.2	0x04	BIT2
P1.3	0x08	BIT3
P1.4	0x10	BIT4
P1.5	0x20	BIT5
P1.6	0x40	BIT6
P1.7	0x80	BIT7

Table 1 - Port number correlation

Each port has several 8 bit registers assigned which control them and provide information about their current status:

- **PxIN:** if a pin is set as an input, this register is read-only
- **PxOUT:** if a pin is set as an output, this register sets the value of that output. (if configured as an input, it sets the pullup or pulldown resistors)
- **PxDIR:** this register selects pin direction, input or output
- **PxREN:** internal pullup/pulldown resistor is enabled. The PxOUT register indicates if it is a pull up (Bit=1) or a pull down (Bit=0)
- **PxSEL and PxSEL2:** I/O function (Bit=0), or peripheral module function (Bit=1)
- **PxIFG:** interrupt flag. It is set when a pin experiences a signal edge transition (from high to low or from low to high).
- **PxIE:** interrupt enable register. Generates an interrupt if a transaction has occurred
- **PxIES:** interrupt transaction selection register. Generates an interrupt indicator when there is a signal edge transaction

2.3 Interrupts

The interrupt priorities are fixed and defined by the arrangement of the modules in the connection chain as shown in Figure 2. The nearer a module is to the CPU/NMIRS, the higher the priority. Interrupt priorities determine what interrupt is taken when more than one interrupt is pending simultaneously.

There are three types of interrupts:

- System reset explication
- (Non) -maskable NMI: explication
- Maskable: explication

2.4 Flash

CC430 has several kinds of memories designed for different purposes.

- **RAM**: is used for both code and data. Working memory
- **Flash**: can be used for both code and data. Word or byte tables can also be stored and read by the program from Flash. All code, tables, and hard-coded constants reside in this memory space.
- **Informationflash**: where variables needed for the next power up can be stored during power down
- **Specialfunctionregister (SFR)**: Some peripheral functions are mapped into memory with special dedicated functions. The Special Function Registers (SFRs) are located at memory addresses from 0000h to 000Fh, and are the specific registers for:

  a) Interrupt enables (locations 0000h and 0001h);
  b) Interrupt flags (locations 0002h and 0003h);
  c) Enable flags (locations 0004h and 0005h);

- **Peripheral registers**: Peripheral modules consist of all on-chip peripheral registers that are mapped into the address space. There are all the key words to configurate timers, ADC…

Flash memory is partitioned into segments. Single bits, bytes, or words can be written to flash memory, but the segment is the smallest size of flash memory that can be erased.

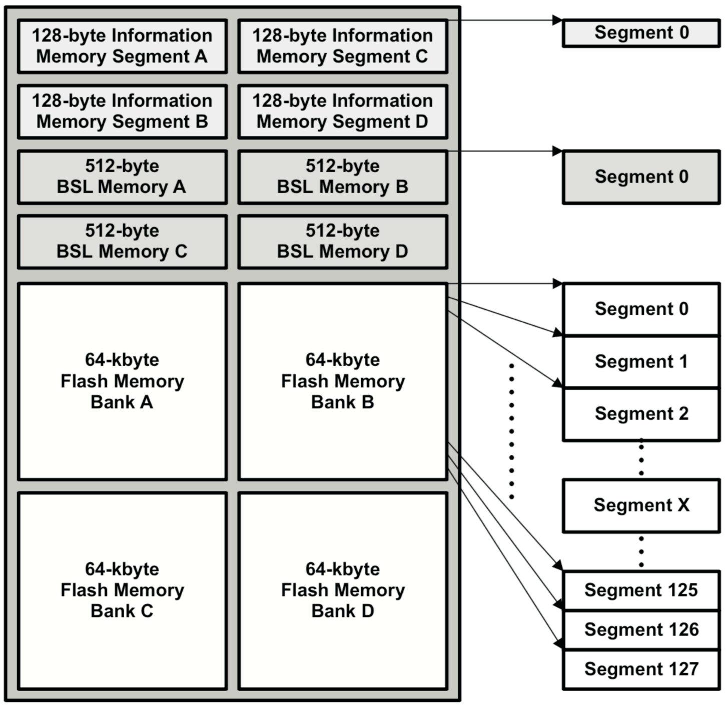

128-byte Information Memory Segment A
128-byte Information Memory Segment C
128-byte Information Memory Segment B
128-byte Information Memory Segment D
512-byte BSL Memory A
512-byte BSL Memory B
512-byte BSL Memory C
512-byte BSL Memory D
64-kbyte Flash Memory Bank A
64-kbyte Flash Memory Bank B
64-kbyte Flash Memory Bank C
64-kbyte Flash Memory Bank D
Segment 0
Segment 0
Segment 0
Segment 1
Segment 2
Segment X
Segment 125
Segment 126
Segment 127

Figure 4 - Flash memory segmentation

In order to write into flash memory there are several settings you have to take into consideration. The timing generator must be a value between 257 kHz to 476 kHz, there should not be any kind of interrupts enabled during the write process. If we want to write into the memory it is necessary to erase the whole segment we are attempting to use. Be aware that flash memory can only be erased 10000 times.

To read from flash memory there is no need to manipulate the frequency or other configurations, you only need a pointer to a flash segment.

2.5 Low Power Modes

Low power mode (**LPM**) is the ability to decrease the power consumed by a microcontroller by shutting progressively different modules and capabilities. Not all the features supplied by a microcontroller are always needed, features such as CPU, clocks, timers etc. can be disabled in order to save energy

MSP430 family includes 5 different low power modes, which shut down several modules:

- **Active**: no power saving, everything is on
- **LPM0**: CPU and MCLK are disabled
- **LPM1**: CPU and MCLK are disabled. FLL loop control is disabled
- **LPM2**: CPU, MCLK, SMCLK and DCO are disabled. ACLK remains active
- **LPM3**: CPU, MCLK, SMCLK, DCO and DC generator are disabled.
- **LPM4**: everything is disabled.

In order to enter and exit on a specific LPM CC430 provides the following keywords:

- `Entering in a xLPM: __bis_SR (LPMx_bits + GIE);`
- `Exiting from a xLPM: __bic_SR_register_on_exit (LPMx_bits);`

2.6 Timers

Timers use a clock source in order to count a specified amount of steps (TAxCCRn). Once this number of steps is reached, the timer usually restarted and triggers an interupt. MSP430 provides one (or more) 16 bit timer/counter with at least 3 capture/compare registers:

- TAxCCR0
- TAxCCR1
- TAxCCR2

Timer features:

- Asynchronous 16-bit timer/counter with four operating modes
- Selectable and configurable clock source
- Two or three configurable capture/compare registers
- Configurable outputs with PWM capability
- Asynchronous input and output latching
- Interrupt vector register for fast decoding of all Timer_A interrupts

Timer modes

Up mode: The timer repeatedly counts up to TACCR0 value, which defines the period.

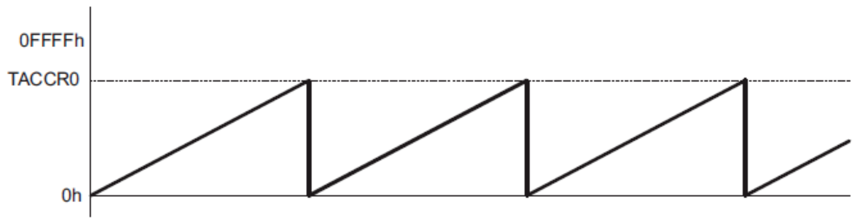


Figure 5 - Timer Up mode

Continuous mode: The timer repeatedly counts up to 0FFFFh and restarts from zero

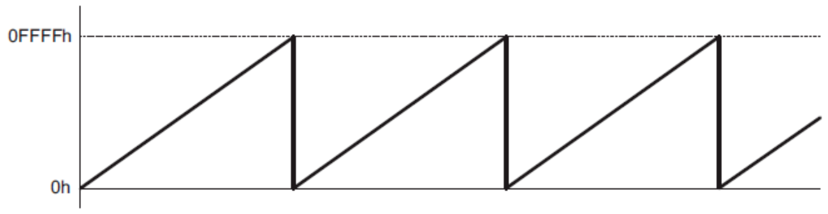


Figure 6 - Timer Continuous mode

Up and down mode: The timer repeatedly counts up to TACCR0 value and back down to zero.

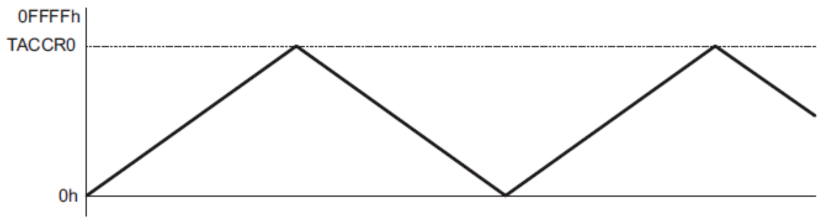


Figure 7 - Timer Up/Down mode

Output Modes

The output unit is used to generate output signals such as PWM signals. Each output unit has eight operating modes that generate different output signals:

- **Output:** The output signal OUTx is defined by the OUT bit.
- **Set:** when the timer reaches the TAxCCRn value the output is set to 1 until the reset of the timer
- **Toggle/Reset:** the output toggles when TAxCCRn is reached. It is reset once the TAxCCR0 is reached.
- **Set/Reset:** the output is set when TAxCCRn is reached. It is reset once the TAxCCR0 is reached.
- **Toggle:** the output toggles when TAxCCRn is reached.
- **Reset:** the output is reset once TAxCCRn is reached. It will remain so unless another output mode is called.
- **Toggle/Set:** the output toggles when TAxCCRn is reached. It is set once the TAxCCR0 is reached.
- **Reset/Set:** the output is reset when TAxCCRn is reached. It is set once the TAxCCR0 is reached.

2.7 ADC

CC430 has two ADCs, one with 10 Bit and the other with 12 Bit. The ADC registers are in charge of handling the conversion of analog signals to digital signals. Its main features are:

- Greater than 200-ksps maximum conversion rate.
- Monotonic 10/12-bit converter with no missing codes.
- Sample-and-hold with programmable sample periods.
- Conversion initiation by software or Timer_A.
- Software selectable reference voltage generation (1.5 V or 2.5 V).
- Software selectable internal or external reference.
- Up to 12 external input channels
- Conversion channels for internal temperature sensor, VCC, and external references.
- Selectable conversion clock source.
- Single-channel, repeated single-channel, sequence, and repeated sequence conversion modes.

- ADC core and reference voltage can be powered down separately.
- Data transfer controller for automatic storage of conversion results.

MSP430 provides a general equation to translate the value of an analog signal into a its digital value:

$$N_{ADC10bit} = 1023 \cdot \frac{V_{in} - V_{R-}}{V_{R+} - V_{R-}} \qquad N_{ADC12bit} = 4095 \cdot \frac{V_{in} - V_{R-}}{V_{R+} - V_{R-}}$$

Equation 1 - ADC conversion

One important setting in ADC registers is the sample frequency. As we are reading from an analog signal it is possible that we get several peaks that we do not want to take into consideration while we are measuring. Therefore it's important to set a suitable sampling frequency and therefor sample multiple times sequentially. Another feature provided by MSP430 family is the possibility to measure different channels at once. The different modes provided are:

- Sample one channel once
- Sample one channel, multiple times sequentially
- Sample multiple channels (a range of channels) once
- Sample multiple channels, multiple times sequentially

The resolution of the conversion is related to the number of bits of the ADC module. Than for a 10 bit ADC we have a resolution of 3mV and for a 12 bit ADC we have a 1mV resolution. That means, ADC converter will take in consideration variations of data larger than 3mV for 10 bit ADC and larger than 1mV for 12 bit ADC.

2.8 UART

Abbreviation for Universal Asynchronous Receiver/Transmitter. It is a simple protocol to connect different devices with 2 wires: Tx (transmit) and Rx (receive). As it works in asynchronous mode, it does not need a clock!

UART features

- 7- or 8-bit data with odd, even, or non-parity
- Independent transmit and receive shift registers
- Separate transmit and receive buffer registers
- LSB-first or MSB-first data transmit and receive
- Built-in idle-line and address-bit communication protocols for multiprocessor systems
- Receiver start-edge detection for auto-wake up from LPMx modes
- Programmable baud rate with modulation for fractional baud rate support
- Status flags for error detection and suppression
- Status flags for address detection
- Independent interrupt capability for receive and transmit

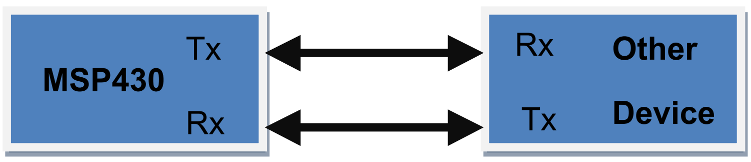


Figure 8 - UART communication

When CC430 tries to communicate with another device using the UART module, it sends a sequence of bits like shown in Table 2. At first a start bit is sent, this warns that some data is coming. Then the payload will be sent, which is software selectable between 7 and 8 bit. CC430 has also the possibility to use parity in order to detect if a message has been corrupted. At the end two stop Bits are sent to end the serial communication.

Bit number	Bit 0	Bit 1 to Bit 8	Bit 9	Bit 10	Bit 11
Definition	Start	Payload	Optional parity	Stop	Stop

Table 2 - UART bits

The baud rate is the number of pulses sent per second. The typical values of baud rates are the following:

Baud Rate	1200	2400	9600	19200	57600	115200	230400

Table 3 - Baud rates

The clock source for the UART is one of the different clocks available in CC430 (UCA0CTL1). In order to set the baud rate of a transmission TI provide the following equation.

$$f_{UART} = f_{UCA0\_CLK}/UCA0BRx$$

Equation 2 Baud rate calculation

2.9 SPI

SPI is the abbreviation of Serial Peripheral Interface and it is a synchronous protocol for communication. Normally at least 4 wires are needed, as it is shown in Figure 9, but sometimes only three are used (e.g. no data coming back). SPI protocol deals with Master and Slaves concepts. The Master initializes the configuration, sends and receive data and generates a clock signal. Slaves only receive commands from the Master and execute them, or answers accordingly. The four signals provided in SPI protocol are:

Signal Name	**Complete Name**
SCLK	Serial Clock
MISO/SOMI	Master Input Slave Output
MOSI/SIMO	Master Output Slave Input
CS	Chip Select

Table 4 - SPI Signals

The clock synchronizes the communication between Master and Slave. MOSI and MISO perform data exchange between the devices. The CS is only used in cases were we have more than one Slave. In case we are using the CS, it indicates which Slave we are talking to.

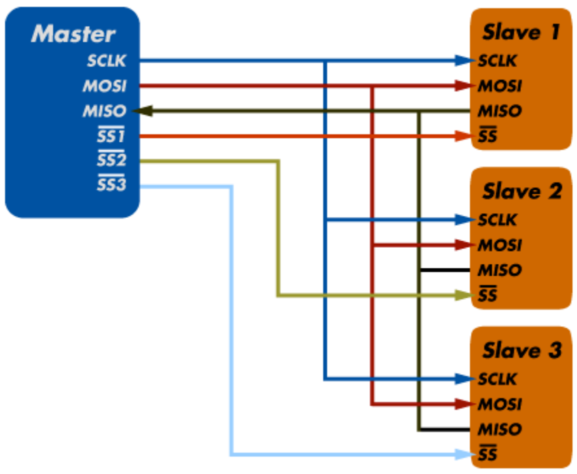


Figure 9 - SPI communication

The main features of SPI module are:

- 7- or 8-bit data length
- LSB-first or MSB-first data transmit and receive
- 3-pin and 4-pin SPI operation
- Master or slave modes
- Independent transmit and receive shift registers
- Separate transmit and receive buffer registers
- Continuous transmit and receive operation
- Selectable clock polarity and phase control
- Programmable clock frequency in master mode
- Independent interrupt capability for receive and transmit
- Slave operation in LPM4

3 SimpliciTi

SimpliciTI is a wireless protocol developed for peer-to-peer communication within Wireless Sensor Networks. It handles all the address handling, authorization, cyclic redundancy check (CRC) as well as encryption. As the CC430 has a integrated RF module with a direct interface to the MDB Bus (Figure 2 - Block Diagram), the controller can go to one of the Low Power modes while the RF Interrupts are still working.

SimpliciTI is a connection-based peer-to-peer protocol and supports 2 basic topologies:

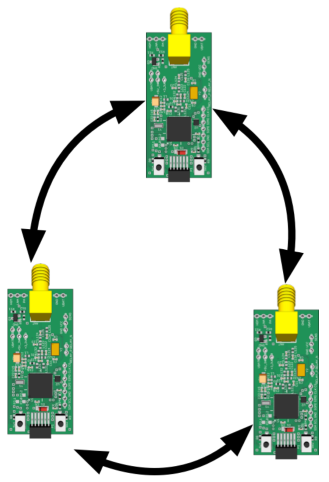


Figure 10 – Peer to peer topology

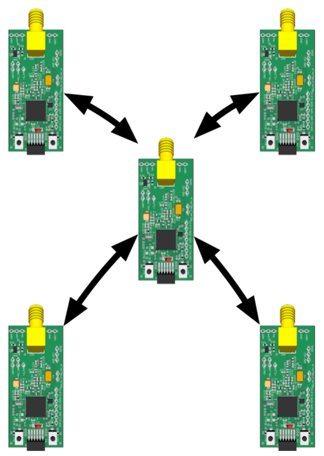


Figure 11 - Star topology

It endures three different types of devices:

a) **Access Point**: used in the star topology, AP act as the hub of the network. They are always powered on.

b) **Range Extender**: used as a repeater, extending the radio signal range on the network. They are limited to 4 range extenders and due to their function they are always on.

c) **End Device**: normally contain the sensors and actuators from the network and host the application peers. A strictly peer to peer topology is only composed of End Devices.

SimpliciTI Architecture

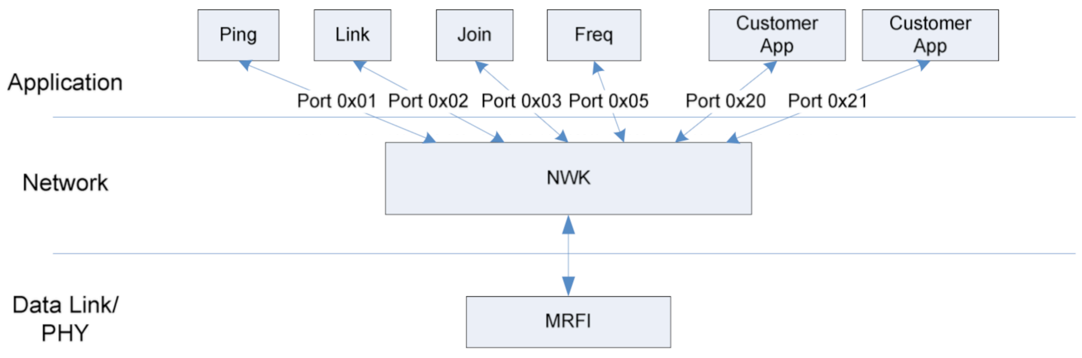


Figure 12 - SimpliciTI layers

a) **Data Link/Physical Layer**: SimpliciTI uses a combination of 2 layers in one. Data link layer which is in charge of providing a reliable link between two nodes and Physical layer which defines the electrical and physical specification of the data connection.

b) **Network Layer**: The network layer provides the functional and procedural means of transferring variable length data sequences from one node to another connected to the same network.

c) **Application Layer**: This layer interacts with software applications that implement a communicating component.

3.1 Data Link/Physical layer

This layer is divided in two categories as they are referring to two OSI layers[4]: The Physical and the Link layers, which are in charge of sending a message between peers and delivering this message without errors/interruptions. This layer is then separated into the following entities:

a) Board Support Package

b) Minimal RF Interface

[4] OSI model: http://en.wikipedia.org/wiki/Physical_layer

3.1.1 Board Support Package (BSP)

This Layer provides a basic support to a particular microcontroller and its RF modul. The BSP functions know the used hardware and guide the communication from the main CPU to the RF module as well as some basic configuration functions.

BSP basic API

a) `BSP_INIT(void)` calls two different functions:

`BSP_INIT_BOARD(void)`

set a CPU frequency, reset the Timer A and set the Timer clock to be SubMain Clock.

`BSP_INIT_DRIVERS(void)`

Initialize the buttons and Led´s on the specific board.

b) `BSP_Delay(uint16_t usec)` set the Timer A to sleep a specific amount of time.

Parameters	**Description**
uint16_t usec	Delay time in microseconds

Table 5 - Delay function parameters

c) `BSP_EARLY_INIT(void)` stop watchdog Timer.

3.1.2 Minimal RF Interface (MRFI)

The MRFI layer interfaces directly with the radio chip. This interface is made to be able to work in different radio families:

- Family 1: CC2500, CC1100, CC1101
- Family 2: CC2510, CC2511, CC1110, CC1111
- Family 3: CC2520
- Family 4: CC2430
- ***Family 5: CC430***
- Family 6: CC2530

This layer is basically an interface to support everything that is necessary to talk to the radio and to read and write data.

MRFI basic API

a) `MRFI_Init(void)`

Calls different functions in order to reset and initialize the communication to the RF module. At the end the registers of the radio get configured with default settings (default frequencies, power…).

b) `uint8_t MRFI_Transmit(mrfiPacket_t *pPacket, uint8_t txType)`

It is in charge of transmiting and delivering a packet using the CCA algorithm or a type forced transmission.

Parameters	**Description**
pPacket	Pointer to a packet location
txType	MRFI_TX_TYPE_CCA MRFI_TX_TYPE_FORCED

Table 6 - MRFI transmit function parameters

CCA: Clear channel assessment (part of listen-before-talk discipline)[5]

[5] ignoring the CCA is used by DoS (Denial-of-service) attacks

Return	Description
MRFI_TX_RESULT_SUCCESS	Transmission succeed
MRFI_TX_RESULT_FAILED	Transmission failed

Table 7 - MRFI transmit function returns

c) `MRFI_RxCompleteISR_new(void)`

This Interrupt service routine gets called when a packet was received. When this ISR is triggerd all of the CRC checks and address filters have already been completed.

d) `MRFI_Receive(mrfiPacket_t * pPacket)`

This function shows the specific location of the last packet received. This function should be called within/out of the ISR.

Parameters	Description
pPacket	Pointer to location of the packet received

Table 8 - MRFI receive function parameters

e) `Mrfi_RxModeOn(void)`

Set the radio in receiving mode.

f) `MRFI_RxOn(void)`

Turns on the receiver.

g) `Mrfi_RxModeOff(void)`

Turns off the receiver.

h) `MRFI_RxIdle(void)`

Set the radio in idle mode, the receiver is off.

i) `MRFI_Sleep(void)`

Request the radio to go to sleep. The radio is turned off.

j) `MRFI_WakeUp(void)`

Used to wake up the radio after a sleep state. Set the radio into Idle mode.

k) `int8_t MRFI_Rssi(void)`

Measures the power present in a received radio signal. It does not check the noise level in a received signal. (How to measure the noise of an incoming signal will be shown in a further chapter)

Return	Description
RSSI	Power of the received signal in dBm

Table 9 - MRFI RSSI function return

l) `uint8_t MRFI_RandomByte(void)`

Generates a pseudo-random number. The generated sequences will be repeated every 256 values.

Return	Description
Random Byte	Random ASCII value

Table 10 - MRFI randombyte function return

m) `Mrfi_DelayUsec(uint16_t howLong)`

It executes a delay loop using a timer.

Parameters	Description
Delay time	Expressed in µs

Table 11 - MRFI delayusec function parameter

There are several functions implemented which can create a delays depending on the time, from µs to ms. They are all in the “mrfi_radio.c” file.

n) `uint8_t MRFI_GetRadioState(void)`

This function gets the state of the radio. It is used by other subroutines and is useful to avoid obsolete settings.

Return	Description
Radio State	Return the actual state of the radio (Rx, idle, off)

Table 12 - MRFI - get radio state function return

RF Transmission Frequencies

To set and change the default values of the registers of the radio module, it is necessary to use an extra function provided by the MRFI API.

`MRFI_RADIO_REG_WRITE(uint8_t register, uint8_t value)` write a value directly to a radio register.

Parameter	Description
register	Address of the Register
value	Value to write to the Register

Table 13 - MRFI write Reg function parameters

The CC430 and its CC1101 RF module can operate in 3 different frequency bands. 300 – 348 MHz, 389 – 464 MHz (ISM Band) and 779 – 928 MHz (SRD Band). In this bands you can set a base frequency with the FREQ register and then be more specific by choosing a channel CHAN. This channel number will then be multiplied with the channel spacing CHANSPC and added to the base frequency.

$$f_{carrier} = \frac{f_{XOSC}}{2^{16}} \times \left(FREQ + CHAN \times \left(\left(256 + CHANSPC\_M\right) \times 2^{CHANSPC\_E-2}\right)\right)$$

Equation 3 frequency calculation

(see *freq_cal.py* for a example calculation)

To change the frequency of the RF module you have to change several registers.

FREQ

22 Bit Register for setting up the base frequency. Normaly set to the middle of the lowest channel.

Register	Description	Default
FREQ2 [23:16]	Bit 21 to 16 of FREQ	0x22
FREQ1 [15:8]	Bit 15 to 8 of FREQ	0xB1
FREQ0 [7:0]	Bit 7 to 0 of FREQ	0x3B

Table 14 - FREQ register

CHANSPC

With the CHANSPC registers you can set the gap between the channels.

Register	Description	Default
MDMCFG1 [7:2]	*Dont change*	001000b
MDMCFG1 [1:0]	Two Bit exponent of cannel spacing **CHANSPC_E**	10b

Table 15 - CHANSPC_E register

Register	Description	Default
MDMCFG0 [7:0]	8-Bit mantissa of cannel spacing **CHANSPC_M**	0xF8

Table 16 - CHANSPC_M register

CHAN

Register	Description	Default
CHANNR [7:0]	The 8-bit unsigned cannel number	0x14

Table 17 - CHAN register

With default settings the channels have the following frequencys:

Channel	Frequency
0x00	902.000 MHz
0x01	902.200 MHz
…	…
0xAA	935.992 MHz
0x11	936.192 MHz
…	…
0xFF	952.988 MHz

Table 18 - Frequencies channels

RF Power settings

In order to provide more irradiation of the electromagnetic field, to achieve more distance and a better resolution of the transmission frames it is possible to adjust the power transmission of the radio chip through its integrated amplifier. For that it is necessary to write to the specific radio register. This radio register is called PATABLE[ ] and its values range is between 0x00 and 0xFF. The PATABLE[ ] register can hold up to eight user selected output power settings. Secondly, the 3-bit FREND0.PA_POWER value selects the PATABLE entry to use. The function should be written as

```
MRFI_RADIO_REG_WRITE(PATABLE[register], value)
```

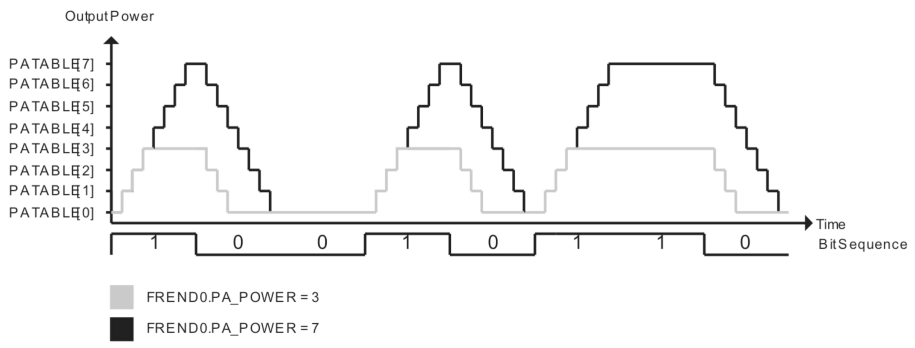


Figure 13 PTABLE ASK

Current consumption at different frequencys

Current Consumption, Transmit Mode

T_A = 25°C, V_{CC} = 3 V (unless otherwise noted)[(1)] [(2)]

PARAMETER	FREQUENCY [MHz}	PATABLE Setting	OUTPUT POWER (dBm)	MIN	TYP	MAX	UNIT
Current consumption, TX	315	0xC0	max.		26		mA
		0xC4	+10		25		mA
		0x51	0		15		mA
		0x29	-6		15		mA
	433	0xC0	max.		33		mA
		0xC6	+10		29		mA
		0x50	0		17		mA
		0x2D	-6		17		mA
	868	0xC0	max.		36		mA
		0xC3	+10		33		mA
		0x8D	0		18		mA
		0x2D	-6		18		mA
	915	0xC0	max.		35		mA
		0xC3	+10		32		mA
		0x8D	0		18		mA
		0x2D	-6		18		mA

Table 19 - power transmission table

3.1.3 Cyclic Redundancy Check (CRC)

The CRC is an error detection code. It is used to detect accidental change of data during transmission.

The CRC follows a mathematical expression based on the remainder of a polynomial division. This remainder comes from a mathematical operation based on the sent and received frame. Then the remainder is added to the frame. Once the frame is received, the receiver makes the same mathematical operation and get a remainder of the received frame. Both remainders are compared (the one calculated by the transceiver and the one calculated by the receiver), if there is a mismatch between them, then the frame has been corrupted and it is deleted.

The **PKTCTRL0** register is in charge of CRC enableing or disenableing.

Bit	Field Name	Description
7	Reserved	-
6	WHITE_DATA	Turn data whitening on / off
		0: Whitening off
		1: Whitening on
5:4	PKT_FORMAT[1:0]	Format of RX and TX data
		00 FIFO for Rx and Tx
		01 Synchronous serial mode
		10 Random TX mode
		11 Asynchronous serial mode
3	Reseved	
2	CRC_EN	0 CRC disabled for TX and RX
		1 CRC calculation in TX and CRC check in RX enabled
1:0	LENGTH_CONFIG[1:0]	Configure Packet Length
		00 Fixed packet length mode
		01 Variable packet length mode
		10 Infinite packet length mode

Table 20 - CRC Register

By default, no CRC or data whitening is enabled. When a packet is sent with enabled CRC, 2 bytes will be added to the payload.

There are also different ways to detect an error in a transmission:

a) Error detection: like CRC
b) Positive confirmations: the receptor returns a confirmation of a correctly received frame.
c) Retransmission: after a while the transceiver gets a positive confirmation, if not it sends the frame again.
d) Negative confirmation and retransmission: the receiver sends only the frames, which did not receive well, then the transceiver has to send the frame again.

3.1.4 Received Signal Strength Indicator (RSSI)

In order to read the power of a received signal, the MRFI layer has a function called MRFI_Rssi, which has been introduced in MRFI basic API. This function does not show the noise or interferences that could be in a specific communication channel, but shows the power of a frame plus the possible interferences. In order to know how much noise there is in a channel we can use this same function.

The process to detect the noise will be the following:

a) Activation of Access Point or node. There should not be any kind of transmission.
b) Choose a determinate channel.
c) Activate the MRFI_Rssi function.

The value in dBm coming from the MRFI_Rssi function is the noise that there is in a determinate channel, as there is no transmission of data at this moment, the power signal received comes only from the interferences and the noise at this frequency.

Another aspect to have in consideration is the channel spacing. As shown before, the default channel spacing provided for the MRFI layer is 200 kHz. This spacing value provides to the user about 200 different channels to use. This can provoke some losses in multiple channels, because of the proximity of the bands. In reality it is important to expand this spacing value, to avoid possible losses. As good practice a channel spacing of 5Mhz is used.

3.2 Network

In SimpliciTI protocol the network layer acts as it does in the OSI model. The network layer is in charge of connecting two different peers and selecting a proper route between these two systems, even if these two are not connected directly. The network layer basically uses the functions of the Data link layer for its applications.

The NWK layer allows run time adjustments. Some of these may be accessible from the application layer via an input output control interface.

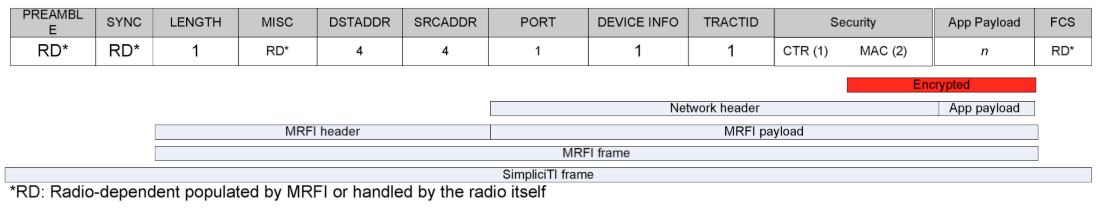
Network parameters can include:

a) Base frequency and frequency spacing
b) Number of frequencies supported
c) Modulation method and data rate and other general radio parameters
d) Default and generated network encryption keys
e) Number of store-and-forward messages to hold
f) Device address
g) Repeat rates on Tx-only devices
h) Join and link tokens

3.2.1 Frame Structure

In SimpliciTI when a frame is sent, it is split up in 3 logical portions:

a) Physical layer
b) Network management supported by the NWK
c) The payload supported in the Application



PREAMBLE	SYNC	LENGTH	MISC	DSTADDR	SRCADDR	PORT	DEVICE INFO	TRACTID	Security	App Payload	FCS
RD*	RD*	1	RD*	4	4	1	1	1	CTR (1) MAC (2)	*n*	RD*

*RD: Radio-dependent populated by MRFI or handled by the radio itself

Table 21 - SimplciTI frame

Detailed description of every field sends in a frame.

Field	Description
Preamble	Radio synchronization
Sync	Radio synchronization
Length	Length of remaining packet in bytes
Misc	Miscellaneousframefields
DSTAddr	Destinationaddress
SRCAddr	Sourceaddress
Port	Forwardedframe (7), Encryptioncontext (6) Applicationportnumber (5-0).
Device Info	Sender/receiver and platform capabilities
TRACTID	TransactionID
Payload	Application data
FCS	FrameCheckSequence

Table 22 - Frame description

A frame can also add some security fields. When the security field is active the payload is encrypted and has to be interpreted by the receiver.

3.2.2 SMPL API

All the functions supplied by the NWK layer of the SimpliciTI protocol will return information about the realized transaction. The information provided is the status of a task. The possible states are:

Status	**Description**
SMPL_SUCCESS	Action successful.
SMPL_TIMEOUT	No link frame received during listen interval. Link ID not valid.
SMPL_BAD_PARAM	No valid Connection Table entry for Link ID; data in Connection Table entry bad; no message or message too long.
SMPL_NOMEM	No room to allocate local Rx port, no more room in Connection Table, or noroom in output frame queue.
SMPL_NO_FRAME	No frame available in input frame queue.
SMPL_NO_LINK	No Link reply received during wait window.
SMPL_NO_JOIN	No Join reply. Access Point possibly not yet up. Not an error if no Access Point in topology
SMPL_NO_CHANNEL	Only if Frequency Agility enabled. Channel scan failed. Access Point possibly not yetup.
SMPL_NO_PEER_UNLINK	Peer did not have a Connection Table entry for specified connection
SMPL_TX_CCA_FAIL	Could not send Link frame. CCA failure. Message not sent.

SMPL_NO_PAYLOAD	Frame received with no payload. Not necessarily an error and could be deduced by application because the returned length will be 0.
SMPL_NO_AP_ADDRESS	Access Point address not known.
SMPL_NO_ACK	Noacknowledgmentreceived.

Table 23 - Status of communication

These states are part of the Smpl_Statusenumstructure located in the nwk_types library.

- **SMPL_Init**

Initialize the radio (mrfi_Init, set a frequency channel, network applications...) to default parameters. This function has to be called once the system is started and before any other API function is called. During the initialization, the device tries to join the network of an AP (if configured as ED). The argument has to be a pointer to a function that takes a Link ID argument and returns a uint8. This allows the application to provide a callback function pointer to the frame handling logic for received frames.

Prototype

```
smplStatus_t   SMPL_Init(uint8 (*pCB)(linkID))
```

Input parameters:

It is a pointer to a callback function, which will be used by the user application once it is received. The key purpose of providing a callback function pointer in the call to SMPL_Init is to allow the user application code access to ISR context management of packets being received on user ports. Every time a packet is received which is associated with a link ID, it is passed to this callback function in the ISR context. Depending on what the callback function does, the packet may be either discarded there or stored in the incoming queue for the associated link ID.

Returned parameters:

- SMPL_SUCCESS
- SMPL_NO_JOIN
- SMPL_NO_CHANNEL

- **SMPL_Link**

Provide a link to an Access Point or a listening device. This function gets an acknowledge once the listener device had received the link request. If the petition does not succeed, it should be repeated until there is a possible link. Upon receiving a reply a connection is established between the two peers and a Link ID is assigned to be used by the application as a handle to the connection.

Prototype

```
smplStatus_t   SMPL_Link(linkID_t *lid)
```

Input parameters:

It is a pointer to a link ID. If the link request succeeds, this ID will be used for further communications.

Returned parameters:

- SMPL_SUCCESS
- SMPL_NO_LINK
- SMPL_NOMEM
- SMPL_TX_CCA_FAIL

- **SMPL_Unlink**

This function disables the connection between two peers deleting the ID, which links them. The local peer sends a message to the linked peer. Once the linked peer receives the termination it deletes also the ID and any call to this ID will always return SMPL_BAD_PARAM.

Prototype

```
smplStatus_t    SMPL_Unlink(linkID_t linkID)
```

Input parameters:

The parameter is the link ID to the connection which should be disabled.

Returned parameters:

- SMPL_SUCCESS
- SMPL_BAD_PARAM
- SMPL_TIMEOUT
- SMPL_NO_PEER_UNLINK

- ## SMPL_LinkListen

This function listens for a call from the link frame (SMPL_LINK). Once it has received a Link ID it will answer with the same ID to the sender.

This is a blocking call, which means, it blocks the program during a while (default 5s) waiting for Link ID´s.

Prototype

```
smplStatus_t    SMPL_LinkListen(linkID_t *linkID)
```

Input parameters:

Pointer to a Link ID. If the link is valid the ID will be used for further communications between the peers.

Returned parameters:

- SMPL_SUCCESS
- SMPL_TIMEOUT

- **SMPL_Send**

This function is used to send application data to a peer. During a transmission the network layer sets a proper configuration of the radio to a successful transaction. By default the transmission always uses CCA.

Prototype

```
smplStatus_t SMPL_Send(linkID_t lid, uint8 *msg, uint8 len)
```

Input parameters:

- Link ID of peer
- Pointer to message buffer
- Length of message

Returned parameters:

- SMPL_SUCCESS
- SMPL_BAD_PARAM
- SMPL_NOMEM
- SMPL_TX_CCA_FAIL

- **SMPL_SendOpt**

This function does the same as the SMPL_Send function, but with the capability of sending a transmit option parameter. This option parameter allows to request an acknowledge to know if the message has been received correctly. By default the SMPL_Send provides no acknowledge.

Prototype

```
    smplStatus_t SMPL_SendOpt(linkID_t lid, uint8 *msg, uint8
len, uint8 opt)
```

Input parameters:

- Link ID of peer

- Pointer to message buffer
- Length of message
- Optional acknowledge: SMPL_TXOPTION_NONE, SMPL_TXOPTION_ACKREQ

Returned parameters:

- SMPL_SUCCESS
- SMPL_BAD_PARAM
- SMPL_NOMEM
- SMPL_TX_CCA_FAIL.
- SMPL_NO_ACK

- ## SMPL_Receive

This function depends on SMPL_SEND. It checks the frame buffers coming from a determinate ID and returns its message and length. Normally this function is used for devices, which are always on.

Prototype

```
smplStatus_t SMPL_Receive(linkID_t lid, uint8 *msg, uint8 *len)
```

Input parameters:

- Link ID of a peer from where the messages are coming
- Pointer to message buffer
- Pointer to length of received message

Returned parameters:

- SMPL_SUCCESS
- SMPL_BAD_PARAM
- SMPL_NO_FRAME
- SMPL_NO_PAYLOAD
- SMPL_TIMEOUT

- SMPL_NO_AP_ADDRESS
- SMPL_TX_CCA_FAIL
- SMPL_NOMEM
- SMPL_NO_CHANNEL

SMPL_Ping

Provides a ping to a determinate peer, determining the presence of a specific device. It can be used to see if the hosting device is there. It does not talk directly to the peer, that means it does not verify that the peer is there, but only that the device hosting the peer is there.

Prototype

```
smplStatus_t    SMPL_Ping(linkID_t lid)
```

Input parameters:

The Link ID of the peer whose device should be pinged.

Returned parameters:

- SMPL_SUCCESS
- SMPL_TIMEOUT

SMPL_Ioctl

The Input/output control API (IOCTL) has the capability of changeing the configuration parameters during run time.

Prototype

```
    smplStatus_t SMPL_Ioctl(ioctlObject_t object, ioctlAction_t
action, void *val)
```

Input parameters:

Objects: Object of the action requested.

Object	**Description**
IOCTL_OBJ_FREQ,	The current logical channel can be set and retrieved with this interface. A scan can also be requested. All of these interfaces are used by **NWK** in support of Frequency Agility.
IOCTL_OBJ_CRYPTKEY,	encryption key
IOCTL_OBJ_RAW_IO,	This object permits sending to and receiving from arbitrary destination address/port combinations. Normally applications must have established peer connection using the linking scheme.
IOCTL_OBJ_RADIO,	Some simple radio control features are currently available. At this time this interface does not support direct access to the radio configuration registers.
IOCTL_OBJ_AP_JOIN,	To add some control over the ability of a device to gain access to the SimpliciTI network the protocol uses tokens to both join a network and to create peers by linking.
IOCTL_OBJ_ADDR,	This interface permits the application to override the build-time device address setting. If the application generates a device address at run time this interface is used to set that address
IOCTL_OBJ_CONNOBJ,	Currently the following interface removes the connection entry for the specified Link ID. It does not tear down the connection by alerting the peer that the local connection is destroyed.
IOCTL_OBJ_FWVER,	The firmware version that is running can be retrieved. It is a read-only (Get) object.
IOCTL_OBJ_PROTOVER,	The protocol version can be used to determine interoperability context or to deny access.
IOCTL_OBJ_NVOBJ,	This object provides direct access to the current connection object
IOCTL_OBJ_TOKEN	An interface is provided to get and set the two network access control tokens, the Join token and the Link token. This feature is enabled with **EXTENDED_API** build time macro definition

Table 24 - Run time objects

Action: Action requested for the specified object.

- IOCTL_ACT_SET
- IOCTL_ACT_GET
- IOCTL_ACT_READ
- IOCTL_ACT_WRITE
- IOCTL_ACT_RADIO_SLEEP
- IOCTL_ACT_RADIO_AWAKE
- IOCTL_ACT_RADIO_SIGINFO
- IOCTL_ACT_RADIO_RSSI
- IOCTL_ACT_RADIO_RXON
- IOCTL_ACT_RADIO_RXIDLE
- IOCTL_ACT_RADIO_SETPWR
- IOCTL_ACT_ON
- IOCTL_ACT_OFF
- IOCTL_ACT_SCAN
- IOCTL_ACT_DELETE

Val: Pointer to parameter information. May be input or output depending on the action. May also be null if the object/action combination requires no parametric information.

Returned parameters:

- Depending on the specific object

3.2.3 Configuration Files

In order to build a program using SimpliciTI protocol additional build time configuration items are needed. There are two macros, the first one defines the structure of the network generals suppliying values to all devices in the network, the second one sets the configuration of specific device functions.

The standard configuration supplied by Texas Instruments does not need to be modified, if we are not going to exceed the default parameters.

General network configuration

The configuration items to take into consideration for the network configurations are:

Macro	**Definition**
MAX_HOPS	Maximum number of times a frame is resent before frame dropped.
MAX_HOPS_FROM_AP	Maximum distance and ED can be from the AP
MAX_APP_PAYLOAD	Maximum payload possible. It is based on size of Rx and Tx FIFOs
DEFAULT_JOIN_TOKEN	access security
DEFAULT_LINK_TOKEN	Should be changed by APs on AP networks.
FREQUENCY_AGILITY	When defined enabled support for Frequency Agility.
APP_AUTO_ACK	Adds support for application level auto acknowledgment.
EXTENDED_API	Enables **SMPL_Unlink()**, **SMPL_Ping()** and **SMPL_Commission()**.
NVOBJECT_SUPPORT	Adds support for getting and setting connection context for saving across resets.
SMPL_SECURITY	Enables security infrastructure

Table 25 - Network configurations

Specific device configuration

Set the configuration of a particular device: Range Extender, End Device or Acces Point.

Macro	**Definition**
NUM_CONNECTIONS	Number of connections supported.
SIZE_INFRAME_Q	Incoming frame queue
SIZE_OUTFRAME_Q	Output frame queue
THIS_DEVICE_ADDRESS	
Range Extender	
RANGE_EXTENDER	
End Device	
END_DEVICE	
RX_POLLS	Define RX_POLLS of the device is a polling End Device.
Access Point	
NUM_STORE_AND_FWD_CLIENTS	Number of store-and-forward clients supported.
NUM_STORE_AND_FWD_CLIENTS	If this macro is defined the AP will be notified through the callback each time a device joins.

Table 26 - Specific device Configurations

3.3 Application

The customer develops the application as the implementation of sensor/actuator interactions with the environment. Using the SimpliciTI API the application can send/receive messages to/from an application peer on another device.

Management of the network itself is supported by SimpliciTI network "applications". Each has its own application protocol as would any customer application.

A port (number of 6 bits) is a conceptual abstraction that specifies which application will handle a determinate frame. The ports are separated into two categories:

- Well known Ports: - Include 0x00 -0x1F

  - Specific services

  - Used by the network management

Application	Port	Description
PING	0x01	Search for a packet on the network, that means, search the presence of devices.
LINK	0x02	Associate different devices.
JOIN	0x03	Only when an AP is present and used to get access to the Network.
SECURITY	0x04	Exchange security information (Encryption key and context). Only when AP is present.
FREQ	0x05	Used for frequency migration (change channel, echo request, change request).
MGMT	0x06	General use of network manage application

Table 27 - Application ports

- Mapped for user handlers: - Include 0x20- 0x3F

Port	Description
0x20-0x3E	Reserved for static allocation
0x3F	Reserved as a broadcast port.

Table 28 - Application reserved ports

4 Bibliography

- **Code composer studio**

  http://www.ti.com/lit/ug/slau157ag/slau157ag.pdf
- **MSP430F2274**

  http://www.ti.com/product/cc430f6137

  cc430f6137.pdf

  slau259e.pdf
- **SimpliciTI**

  Application Note on SimpliciTI Frequency Agility.pdf

  Application Note on SimpliciTI Security.pdf

  SimpliciTI API.pdf

  SimpliciTI Change Log.pdf

  SimpliciTI Channel Table Information.pdf

  SimpliciTI Channel Table Information.pdf

  SimpliciTI Developers Notes.pdf

  SimpliciTI Sample Application User's Guide.pdf

  SimpliciTI Specification.pdf

  http://processors.wiki.ti.com/index.php/SimpliciTI_FAQ

  http://processors.wiki.ti.com/index.php/Creating_CCS_Project_using_SimpliciTI

  http://www.ti.com/tool/simpliciti