TEXAS INSTRUMENTS

Z-Stack Generic Application User's Guide

Document Number: SWRU394

Texas Instruments, Inc.
San Diego, California USA

Revision	Description	Date
1.0	Initial release	09/18/2014

Table of Contents

Table of Figures

Table of Tables

1. Introduction

The Texas Instruments Z-Stack™ product line includes support for popular ZigBee standard profiles, such as ZigBee Home Automation, ZigBee Light Link, and ZigBee Smart Energy. In addition, Z-Stack provides complete support for manufacturer-specific or proprietary wireless implementations employing the full mesh networking capabilities offered by ZigBee technology. Z-Stack is a complete protocol stack and application development solution that conforms to ZigBee Alliance standards (www.zigbee.org).

1.1. Scope

The Z-Stack Mesh product includes certified ZigBee stack software coupled with an easy-to-use sample application, **GenericApp**, to demonstrate a generic wireless mesh solution running on various Texas Instruments development platforms. This document provides basic information for setting up and running **GenericApp** for demonstration and development purposes.

2. Hardware and Software

To begin working with GenericApp, some development hardware and associated software needs to be selected and then installed to work with a Windows PC. All development environments discussed below require at least one standard USB port for programming and debugging.

2.1. Development Kits

GenericApp software supports several Texas Instruments Low-Power RF devices, including the CC2530/31/38 series of wireless MCUs and the CC2520 transceiver. Low-cost development kits are available from TI for use with the Z-Stack Mesh software package to work with GenericApp. ZigBee applications built for these development kits are interoperable since they all employ the certified Texas Instruments ZigBee stack software.

2.1.1. CC2538EM with SmartRF06EB

The CC2538 Development Kit (CC2538DK) provides a pair of SmartRF06EB general purpose development boards, a pair of CC2538EM RF evaluation modules, and a CC2531USB dongle for packet sniffing. The CC2538 is a high performance 32-bit ARM Cortex M3-based, IEEE 802.15.4 compliant Wireless MCU, with up to 512KB of flash and 32KB of RAM memory, ideal for demanding ZigBee applications

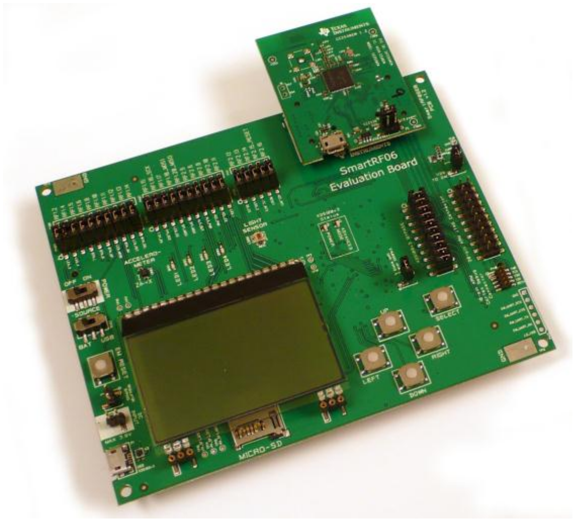


Figure 1: CC2538EM on SmartRF06EB

2.1.2. CC2530EM with SmartRF05EB

The CC2530 Development Kit (CC2530DK) provides a pair of SmartRF05EB general purpose development boards, two CC2530EM RF evaluation modules, and a CC2531USB dongle for packet sniffing. The CC2530 is an 8-bit 8051-based, IEEE 802.15.4 compliant wireless MCU, with up to 256KB of flash and 8KB of RAM, ideal for general purpose ZigBee applications.

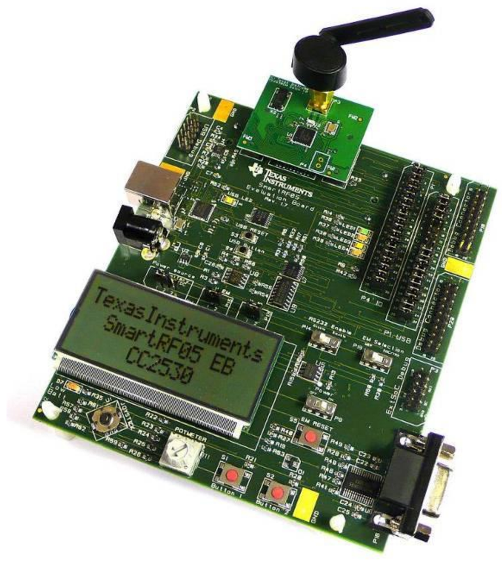


Figure 2: CC2530EM on SmartRF05EB

2.1.3. CC2520EM with EXP430F5438

The CC2520 Evaluation Module Kit (CC2520EMK) provides a pair of CC2520EM modules that can be used with EXP5438 Experimenter's Boards (MSP-EXP430F5438). The CC2520 is an intelligent, 2.4 GHz, IEEE 802.15.4 compliant transceiver. The EXP5438 is a 16-bit low-power MCU with 256KB of flash and 16KB of RAM, ideal for general purpose ZigBee applications.

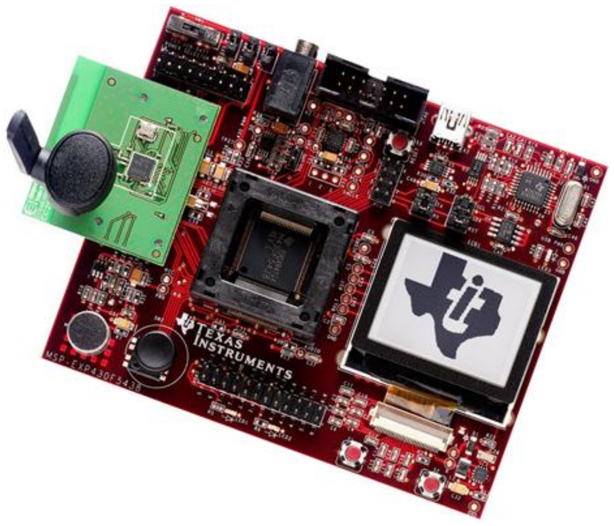


Figure 3: CC2520EM on EXP430F5438

2.1.4. CC2531DK USB Dongle

The CC2531 USB Evaluation Module Kit (CC2531EMK) provides a CC2531EM USB Dongle which can be used to prototype USB-based ZigBee devices and to evaluate the RF performance of CC2531 with a small PCB antenna. The CC2531 is an 8-bit 8051-based, IEEE 802.15.4 compliant wireless MCU (same core as CC2530), with up to 256KB of flash and 8KB of RAM, ideal for PC or Linux hosted ZigBee applications.

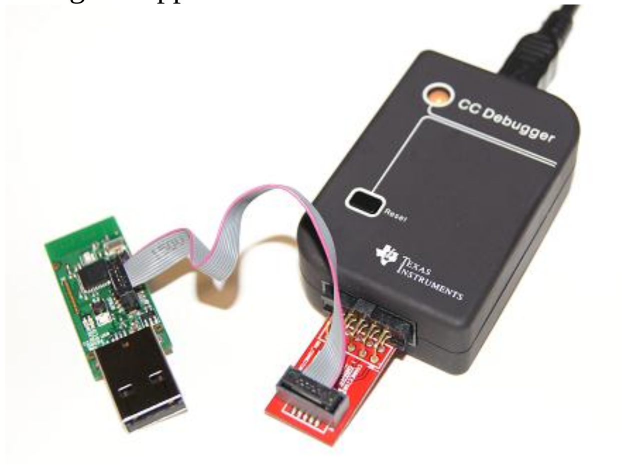


Figure 4: CC2531DK USB Dongle with CC Debugger

2.2. Software Tools

To begin working with the Z-Stack Mesh software package, some software development tools are required, depending on the selected development kit. Other tools are available that can make the development and debugging process more efficient.

2.2.1. Required Tools

Some software tools are required to evaluate Z-Stack and work with GenericApp:

- TI Z-Stack Mesh: complete ZigBee development package for non-public profile devices: http://www.ti.com/tool/z-stack

- IAR Embedded Workbench: a full-featured tool suite for editing, compiling, linking, loading, and debugging applications on a target device:

  EWARM (for CC2538):
  http://www.iar.com/Products/IAR-Embedded-Workbench/ARM/

  EW8051 (for CC2530 and CC2531):
  http://www.iar.com/Products/IAR-Embedded-Workbench/8051/

  EW430 (for CC2520 + MSP430):
  http://www.iar.com/Products/IAR-Embedded-Workbench/TI-MSP430/

- TI CC-Debugger: a small programmer/debugger for TI RF System-on-Chip is required for use with CC2531 USB Dongle:
  http://www.ti.com/tool/cc-debugger

2.2.2. Additional Tools

Additional software tools are available to enhance working with Z-Stack and GenericApp:

- SmartRF Protocol Packet Sniffer: a low-cost tool to decode, and display RF over-the-air packets, with options for filtering and storage to a binary file format: http://www.ti.com/tool/packet-sniffer

- SmartRF Flash Programmer: can be used to program the flash memory in TI low-power RF wireless MCUs and for upgrading the firmware and bootloader on evaluation boards and debugger modules. Flash-Programmer supports a long list of devices and boards except for the SmartRF06EB, which is supported by Flash-Programmer-2: http://www.ti.com/tool/flash-programmer

- Ubiqua Protocol Analyzer or another type of network analyzer that supports full-featured ZigBee over-the-air packet decoding. Note that Ubiqua requires a CC2531 USB Dongle to capture the wireless network traffic: http://www.ubilogix.com/products/ubiqua

3. Getting Started

This section provides step-by-step instructions to get devices programmed with GenericApp. Since device programming involves connecting and disconnecting boards from the development PC, power controls for each development kit are discussed before looking at the software build and download process. To avoid damage, programmable devices should be powered OFF before connecting to or removing from the USB port of the PC.

3.1. Development Kit Power Sources

Each of the development kits with general purpose evaluation boards (section 2.1) has several options for connecting/selecting power sources. These options are typically controlled by switches and/or jumpers on the evaluation board. In addition, each board has a jumper to supply power to the RF evaluation module, providing a site to measure the current drawn by the module. Refer to the evaluation board User’s Guide for instructions for measurement of the RF module power consumption.

3.1.1. SmartRF06EB

The SmartRF06EB (with CC2538EM) provides four power supply options: USB cable from the development PC, two AAA batteries, CR2032 coin cell battery, and external DC power supply (3.6V max). Power to the board, from all sources, is controlled by the OFF/ON power switch shown below. The power source is selected by switch S502. For normal development/debugging, connect a USB cable between the SmartRF06EB and PC, and set the SOURCE switch to USB:

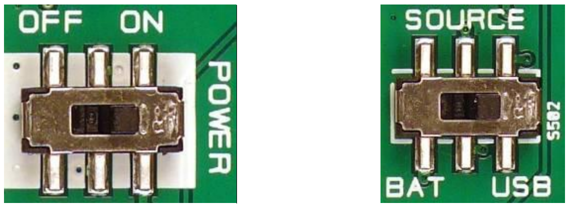


Figure 5: SmartRF06EB Power and Source Switches

3.1.1.1. CC2538EM

For normal operation of a CC2538EM board, a jumper should be placed on the VDD-EB Power pins (P5) as shown below. To measure the power consumption of the board, replace the jumper with a low value resistor, such as 2.2 ohms, and measure the voltage drop across the resistor.

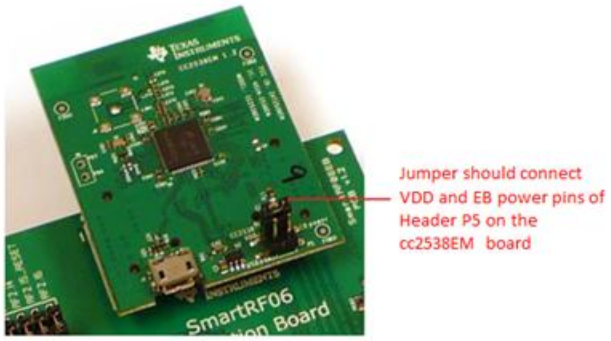

Jumper should connect VDD and EB power pins of Header P5 on the cc2538EM board
SmartRF06
Board

Figure 6: CC2538EM Power Jumper

3.1.2. SmartRF05EB

The SmartRF05EB (with CC2530EM) provides three power supply options: USB cable from the development PC, two AA batteries, and external DC power supply (4-10V). Power to the board, from each source, is controlled by the Power OFF/ON switch (P8) shown below. The power source is be selected by jumper P11. For normal development/debugging, connect a USB cable between connector P12 and the PC and set the SOURCE jumper to USB/DC (pins 2-3):

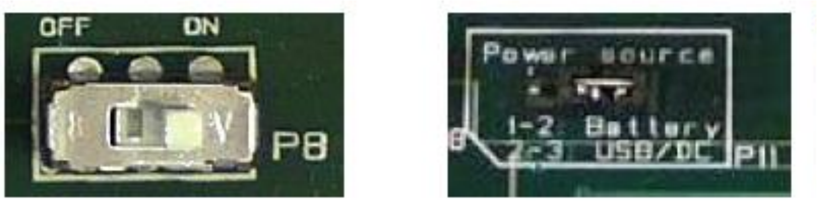


Figure 7: SmartRF05EB Power Switch and Source Jumper

3.1.3. EXP430F5438

The MSP-EXP430F5438 (with CC2520EM) provides three power supply options: FET debug module connected to the development PC, two AA batteries, and USB. Power to the board, from each of these sources, is controlled by a 3-position slider switch (SW1) as shown below. For normal development/debugging, connect an FET430UIF module between the JTAG connector on the evaluation board and the PC, and set the power switch to FET:

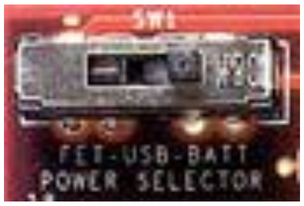


Figure 8: EXP5438 Power Selector Switch

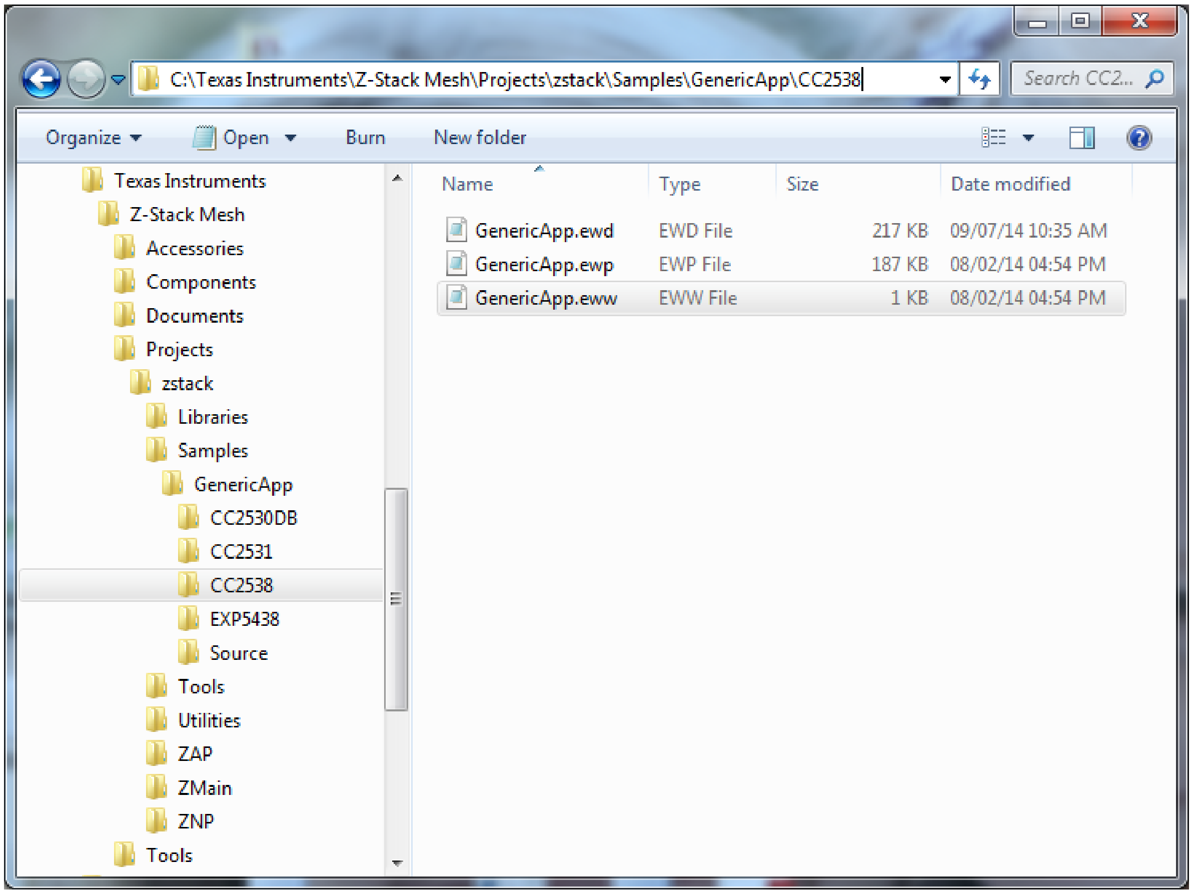
3.1.4. CC2531DK

The CC2531 USB dongle receives power from the USB port that it is plugged into. When it is programmed using a CC-Debugger (as shown in Figure 4), it must also be powered from the USB connector – plugged into a PC port directly or via USB cable.

3.2. Programming Devices

This section illustrates programming some devices with GenericApp to set up a simple ZigBee network with 2 or more nodes - a Coordinator and one or more Routers. The instructions and screenshots shown below are specifically from IAR Embedded Workbench for ARM (EWARM) working with a CC2538EM and SmartRF06EB. The process is functionally identical when using the EW8051 or EW430 toolsets with CC2530/31 or EXP5438 platforms, respectively.

- Make sure all required development tools have been installed (section 2.2.1)
- Connect a CC2538EM module to a SmartRF06EB development board
- Connect the SmartRF06EB to the development PC with a USB cable
- Power up the SmartRF06EB and CC2538EM (section 3.1.1)
- Using Windows, navigate to the GenericApp project directory:



Name	Type	Size	Date modified
GenericApp.ewd	EWD File	217 KB	09/07/14 10:35 AM
GenericApp.ewp	EWP File	187 KB	08/02/14 04:54 PM
GenericApp.eww	EWW File	1 KB	08/02/14 04:54 PM

Figure 9: Locating the Generic Application Project

- Launch IAR Embedded Workbench, then open the GenericApp project workspace by dragging and dropping the ***GenericApp.eww*** file into the IAR IDE window:

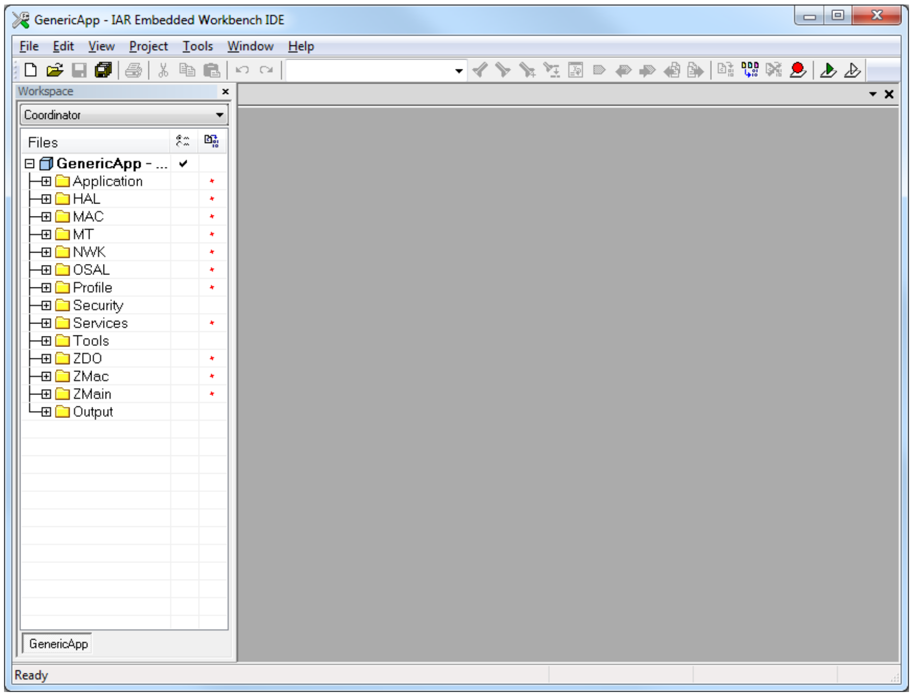
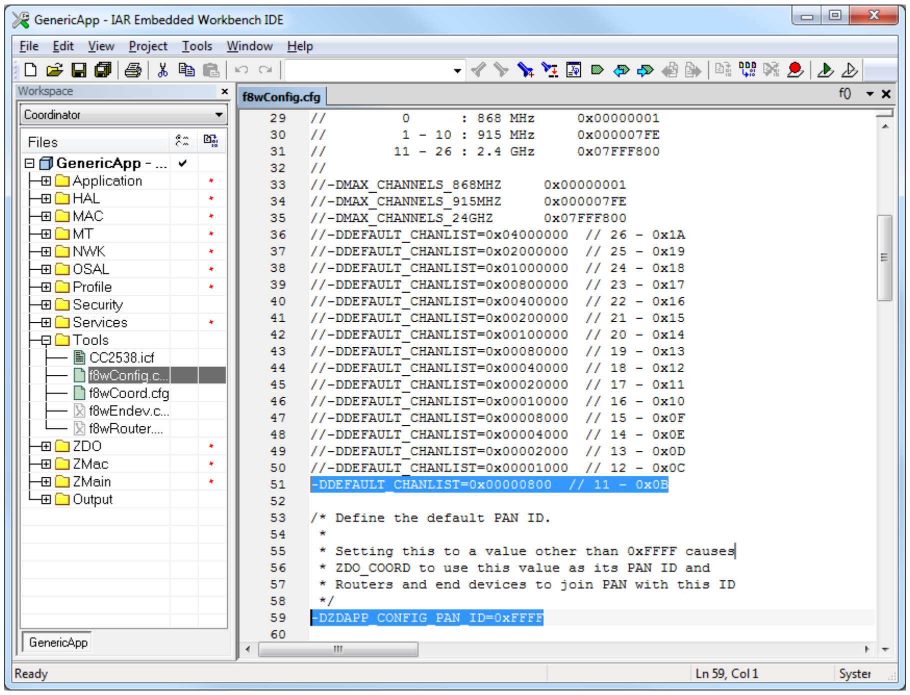


Figure 10: Opening the Generic Application Project

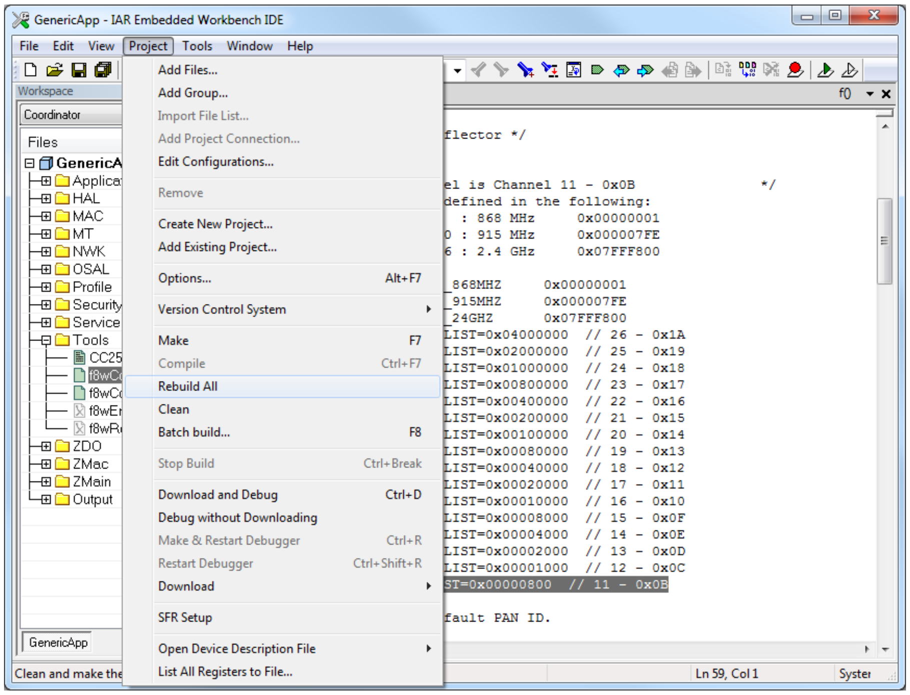
- If desired, modify the ZigBee Channel and/or PanID by editing ***fw8Config.cfg***:



```
//          0       : 868 MHz      0x00000001
//          1 - 10 : 915 MHz      0x000007FE
//         11 - 26 : 2.4 GHz      0x07FFF800
//
//-DMAX_CHANNELS_868MHZ     0x00000001
//-DMAX_CHANNELS_915MHZ     0x000007FE
//-DMAX_CHANNELS_24GHZ      0x07FFF800
//-DDEFAULT_CHANLIST=0x04000000  // 26 - 0x1A
//-DDEFAULT_CHANLIST=0x02000000  // 25 - 0x19
//-DDEFAULT_CHANLIST=0x01000000  // 24 - 0x18
//-DDEFAULT_CHANLIST=0x00800000  // 23 - 0x17
//-DDEFAULT_CHANLIST=0x00400000  // 22 - 0x16
//-DDEFAULT_CHANLIST=0x00200000  // 21 - 0x15
//-DDEFAULT_CHANLIST=0x00100000  // 20 - 0x14
//-DDEFAULT_CHANLIST=0x00080000  // 19 - 0x13
//-DDEFAULT_CHANLIST=0x00040000  // 18 - 0x12
//-DDEFAULT_CHANLIST=0x00020000  // 17 - 0x11
//-DDEFAULT_CHANLIST=0x00010000  // 16 - 0x10
//-DDEFAULT_CHANLIST=0x00008000  // 15 - 0x0F
//-DDEFAULT_CHANLIST=0x00004000  // 14 - 0x0E
//-DDEFAULT_CHANLIST=0x00002000  // 13 - 0x0D
//-DDEFAULT_CHANLIST=0x00001000  // 12 - 0x0C
-DDEFAULT_CHANLIST=0x00000800  // 11 - 0x0B

/* Define the default PAN ID.
 *
 * Setting this to a value other than 0xFFFF causes
 * ZDO_COORD to use this value as its PAN ID and
 * Routers and end devices to join PAN with this ID
 */
-DZDAPP_CONFIG_PAN_ID=0xFFFF

```

Figure 11: Setting ZigBee Channel and PanID

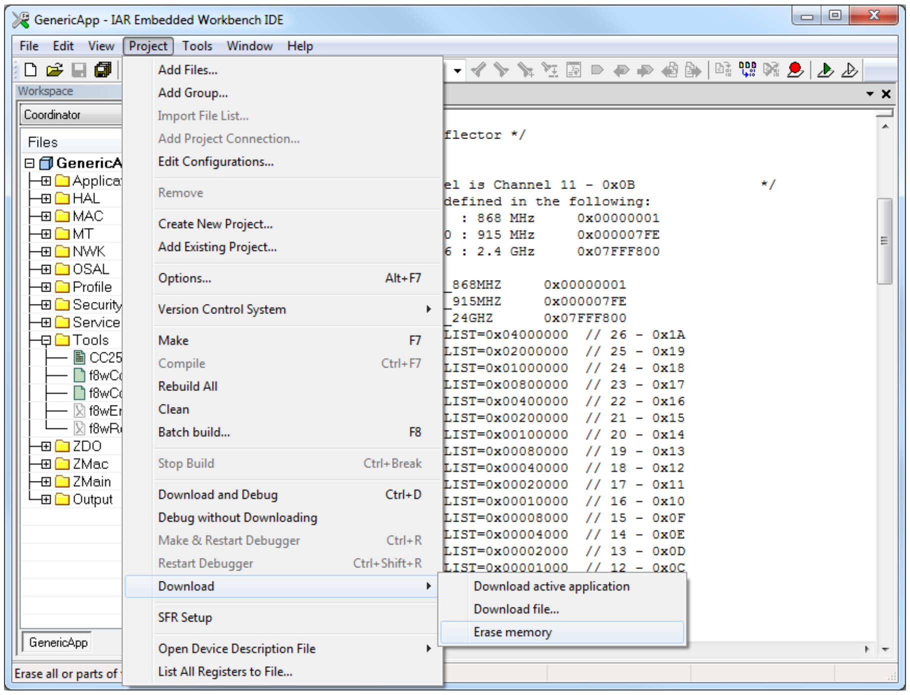
- Build the device by pulling down the *Project* menu and clicking on **Rebuild All**:



```
flector */

el is Channel 11 - 0x0B                        */
defined in the following:
  : 868 MHz      0x00000001
0 : 915 MHz      0x000007FE
6 : 2.4 GHz      0x07FFF800

 868MHZ     0x00000001
 915MHZ     0x000007FE
 24GHZ      0x07FFF800
LIST=0x04000000  // 26 - 0x1A
LIST=0x02000000  // 25 - 0x19
LIST=0x01000000  // 24 - 0x18
LIST=0x00800000  // 23 - 0x17
LIST=0x00400000  // 22 - 0x16
LIST=0x00200000  // 21 - 0x15
LIST=0x00100000  // 20 - 0x14
LIST=0x00080000  // 19 - 0x13
LIST=0x00040000  // 18 - 0x12
LIST=0x00020000  // 17 - 0x11
LIST=0x00010000  // 16 - 0x10
LIST=0x00008000  // 15 - 0x0F
LIST=0x00004000  // 14 - 0x0E
LIST=0x00002000  // 13 - 0x0D
LIST=0x00001000  // 12 - 0x0C
ST=0x00000800  // 11 - 0x0B

fault PAN ID.
```

Figure 12: Building the Generic Application

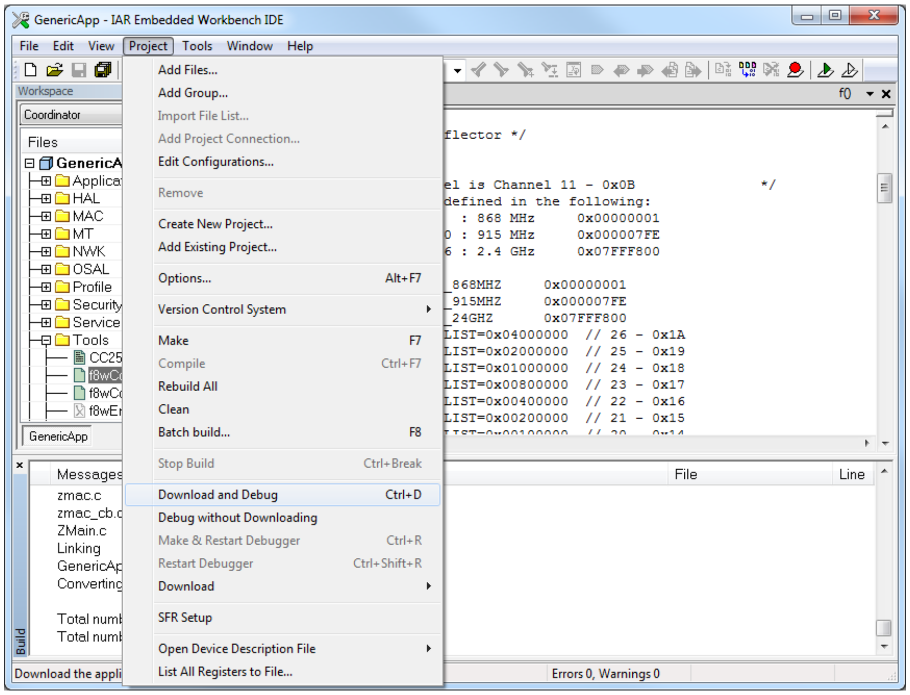
- Before programming this CC2538, erase the flash memory by pulling down the *Project* menu and selecting **Download->Erase memory**:



```
flector */

el is Channel 11 - 0x0B                        */
defined in the following:
  : 868 MHz      0x00000001
0 : 915 MHz      0x000007FE
6 : 2.4 GHz      0x07FFF800

 868MHZ     0x00000001
 915MHZ     0x000007FE
 24GHZ      0x07FFF800
LIST=0x04000000  // 26 - 0x1A
LIST=0x02000000  // 25 - 0x19
LIST=0x01000000  // 24 - 0x18
LIST=0x00800000  // 23 - 0x17
LIST=0x00400000  // 22 - 0x16
LIST=0x00200000  // 21 - 0x15
LIST=0x00100000  // 20 - 0x14
LIST=0x00080000  // 19 - 0x13
LIST=0x00040000  // 18 - 0x12
LIST=0x00020000  // 17 - 0x11
LIST=0x00010000  // 16 - 0x10
LIST=0x00008000  // 15 - 0x0F
LIST=0x00004000  // 14 - 0x0E
LIST=0x00002000  // 13 - 0x0D
LIST=0x00001000  // 12 - 0x0C
```

Figure 13: Erase CC2538 Program Memory

- Download by pulling down the *Project* menu and clicking on **Download and Debug**:

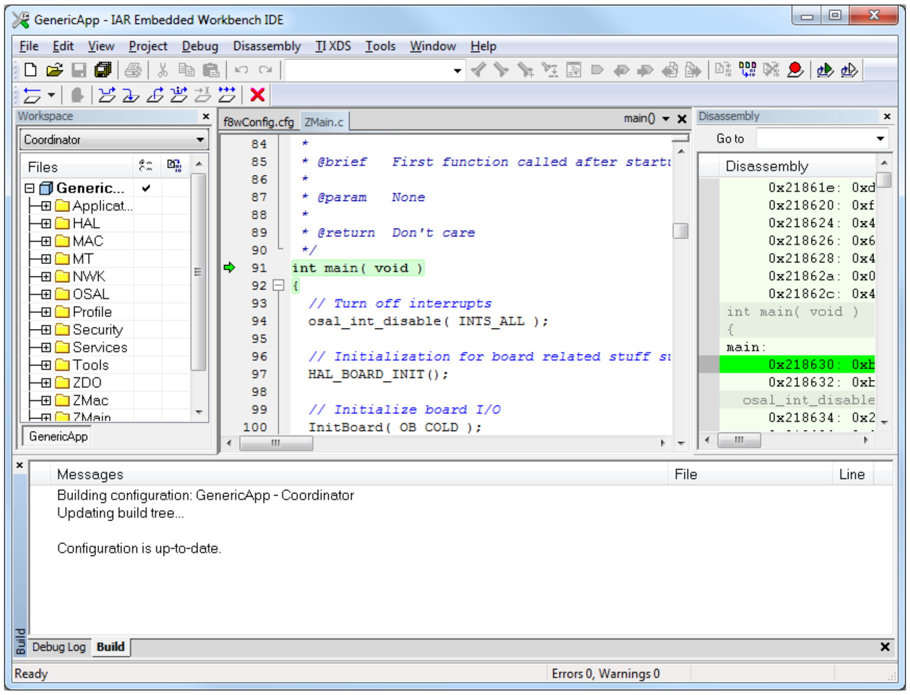


```
flector */

el is Channel 11 - 0x0B                          */
defined in the following:
 : 868 MHz      0x00000001
0 : 915 MHz      0x000007FE
6 : 2.4 GHz      0x07FFF800

_868MHZ      0x00000001
_915MHZ      0x000007FE
_24GHZ       0x07FFF800
LIST=0x04000000  // 26 - 0x1A
LIST=0x02000000  // 25 - 0x19
LIST=0x01000000  // 24 - 0x18
LIST=0x00800000  // 23 - 0x17
LIST=0x00400000  // 22 - 0x16
LIST=0x00200000  // 21 - 0x15
```

```
zmac.c
zmac_cb.c
ZMain.c
Linking
GenericAp
Converting

Total numb
Total numb
```

Figure 14: Downloading the Generic Application

- When downloading is complete, the program halts at *main()*, ready to run or debug:



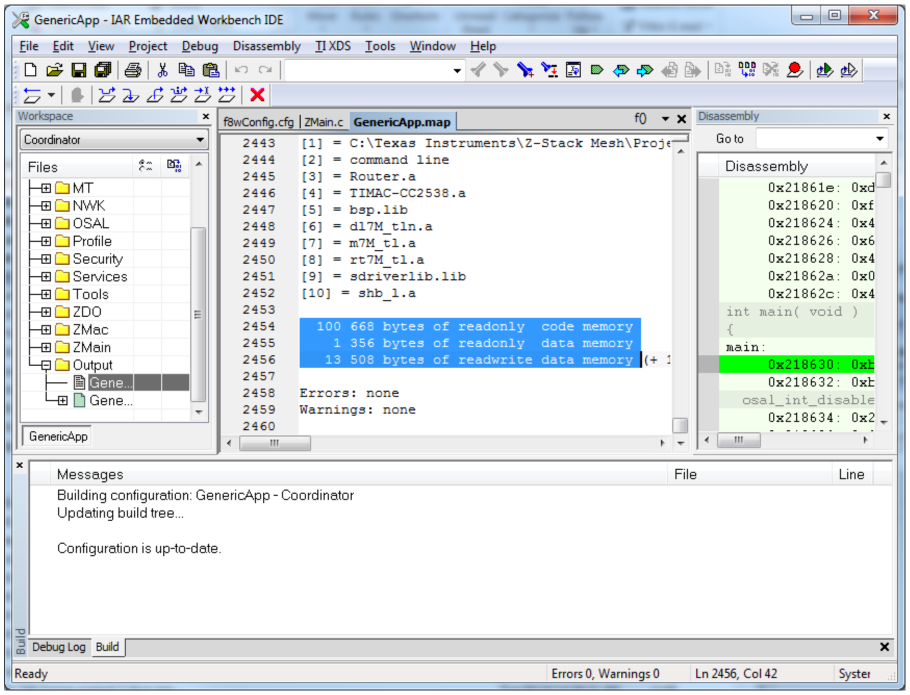
```
 *
 * @brief   First function called after start
 *
 * @param   None
 *
 * @return  Don't care
 */
int main( void )
{
  // Turn off interrupts
  osal_int_disable( INTS_ALL );

  // Initialization for board related stuff su
  HAL_BOARD_INIT();

  // Initialize board I/O
  InitBoard( OB_COLD );
```

```
Disassembly
    0x21861e: 0xd
    0x218620: 0xf
    0x218624: 0x4
    0x218626: 0x6
    0x218628: 0x4
    0x21862a: 0x0
    0x21862c: 0x4
 int main( void )
 {
main:
    0x218630: 0xb
    0x218632: 0xb
  osal_int_disable
    0x218634: 0x2
```

```
Building configuration: GenericApp - Coordinator
Updating build tree...

Configuration is up-to-date.
```

Figure 15: Program Download Is Complete

- To examine memory utilization, open *GenericApp.map* and scroll to the end of the file:



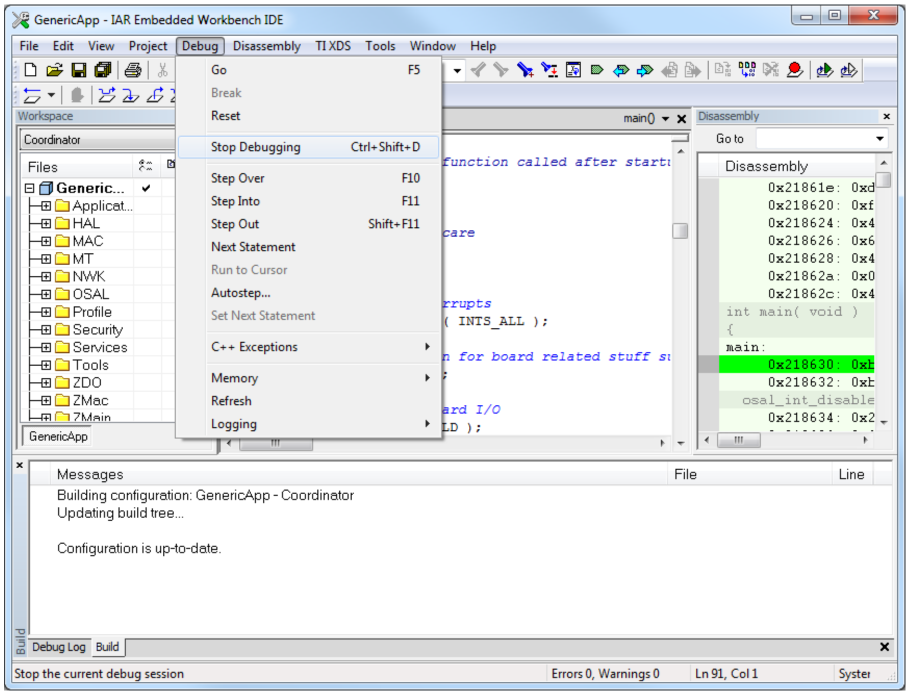
```
[1] = C:\Texas Instruments\Z-Stack Mesh\Proj
[2] = command line
[3] = Router.a
[4] = TIMAC-CC2538.a
[5] = bsp.lib
[6] = dl7M_tln.a
[7] = m7M_tl.a
[8] = rt7M_tl.a
[9] = sdriverlib.lib
[10] = shb_l.a

 100 668 bytes of readonly  code memory
   1 356 bytes of readonly  data memory
  13 508 bytes of readwrite data memory  (+ 1

Errors: none
Warnings: none
```

```
0x21861e: 0xd
0x218620: 0xf
0x218624: 0x4
0x218626: 0x6
0x218628: 0x4
0x21862a: 0x0
0x21862c: 0x4
int main( void )
{
main:
0x218630: 0xb
0x218632: 0xb
osal_int_disable
0x218634: 0x2
```

Messages	File	Line
Building configuration: GenericApp - Coordinator		
Updating build tree...		
Configuration is up-to-date.		

Figure 16: Examine Device Flash/RAM Utilization

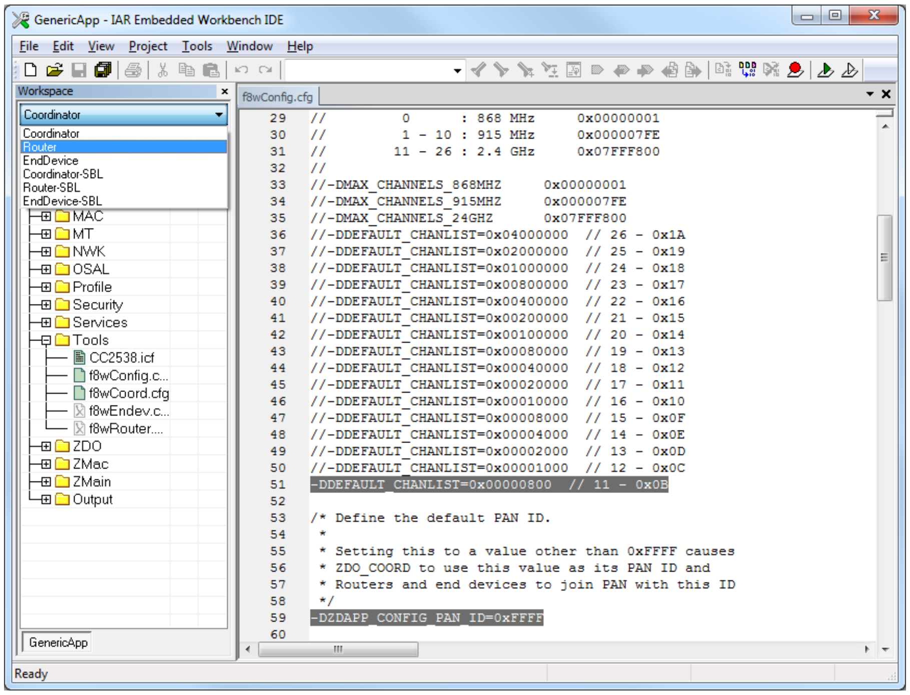
- To prepare for programming another device, exit the debugger on this device by pulling down the ***Debug*** menu and clicking on **Stop Debugging**:



Messages	File	Line
Building configuration: GenericApp - Coordinator		
Updating build tree...		
Configuration is up-to-date.		

Figure 17: Quit Debugging the Generic Application

- Remove power from this SmartRF06EB board by moving its POWER switch to OFF. Disconnect the SmartRF06EB from the USB cable. Attach a unique, descriptive label (such as Coordinator, Router1, etc.), and set it aside.
- The Generic Sample Application requires at least two devices to form a ZigBee network, one Coordinator and one or more non-Coordinator devices (Router or End-Device). To build a non-Coordinator device continue to the next step, otherwise exit the IAR IDE and proceed to Section 4 to run the Generic Application..
- Select a non-Coordinator device type configuration from the *Workspace* pull-down menu. The example below shows a Router device selection. For this tutorial, please do not select the -SBL (*S*erial *B*oot *L*oader) configurations. Now repeat all steps in this section to program another device:



```
//          0       : 868 MHz      0x00000001
//          1 - 10  : 915 MHz      0x000007FE
//         11 - 26  : 2.4 GHz      0x07FFF800
//
//-DMAX_CHANNELS_868MHZ     0x00000001
//-DMAX_CHANNELS_915MHZ     0x000007FE
//-DMAX_CHANNELS_24GHZ      0x07FFF800
//-DDEFAULT_CHANLIST=0x04000000  // 26 - 0x1A
//-DDEFAULT_CHANLIST=0x02000000  // 25 - 0x19
//-DDEFAULT_CHANLIST=0x01000000  // 24 - 0x18
//-DDEFAULT_CHANLIST=0x00800000  // 23 - 0x17
//-DDEFAULT_CHANLIST=0x00400000  // 22 - 0x16
//-DDEFAULT_CHANLIST=0x00200000  // 21 - 0x15
//-DDEFAULT_CHANLIST=0x00100000  // 20 - 0x14
//-DDEFAULT_CHANLIST=0x00080000  // 19 - 0x13
//-DDEFAULT_CHANLIST=0x00040000  // 18 - 0x12
//-DDEFAULT_CHANLIST=0x00020000  // 17 - 0x11
//-DDEFAULT_CHANLIST=0x00010000  // 16 - 0x10
//-DDEFAULT_CHANLIST=0x00008000  // 15 - 0x0F
//-DDEFAULT_CHANLIST=0x00004000  // 14 - 0x0E
//-DDEFAULT_CHANLIST=0x00002000  // 13 - 0x0D
//-DDEFAULT_CHANLIST=0x00001000  // 12 - 0x0C
-DDEFAULT_CHANLIST=0x00000800  // 11 - 0x0B

/* Define the default PAN ID.
 *
 * Setting this to a value other than 0xFFFF causes
 * ZDO_COORD to use this value as its PAN ID and
 * Routers and end devices to join PAN with this ID
 */
-DZDAPP_CONFIG_PAN_ID=0xFFFF

```

Figure 18: Change Project Device Configuration

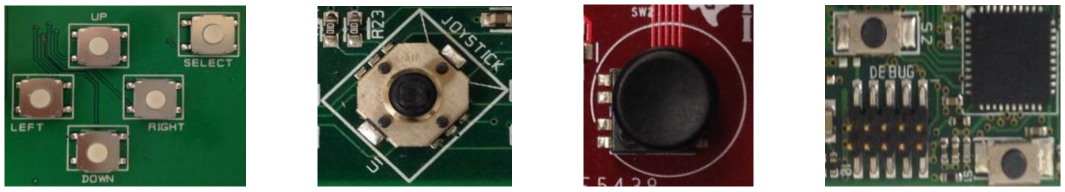
4. Working with GenericApp

Once the development boards have been programmed, the GenericApp program can be run by following the simple instructions below. Since GenericApp needs some user input to get things started and shows some basic status on LEDs, the next section discusses the "switch" input and LED output characteristics for each of the development boards.

4.1. Switches and LEDs

Z-Stack sample applications, including GenericApp, make references to "switches" and LEDs to initiate various device functions and to display application/device status. Since Z-Stack runs on many different platforms, "switches" and LEDs are abstracted in software to "logical" devices, referred to as SW1 through SW5, and LED1 through LED4, respectively.

Each development board has some number of pushbuttons or a mini-joystick to generate the "switch" inputs. As indicated below, boards discussed in Section 2.1 use quite different physical implementations for their "switch" inputs. Table 1 shows mapping of "logical" switches to the actual hardware input devices. For joysticks, the 'Up' position is toward the top of this page.



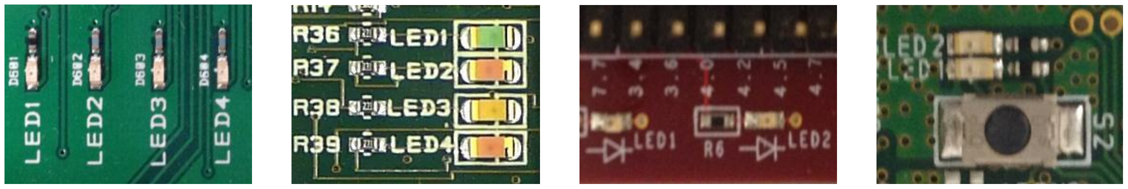
SmartRF06 SmartRF05 EXP5438 CC2531DK

Figure 19: Development Board Buttons and Joysticks

Logical Switch	SmartRF06 Buttons	SmartRF05 Joystick	EXP5438 Joystick	CC2531DK Buttons
SW1	UP	Up	Up	S1
SW2	RIGHT	Right	Right	S2
SW3	DOWN	Down	Down	-
SW4	LEFT	Left	Left	-
SW5	SELECT	Center Press	Center Press	-

Table 1: Development Board Logical Switch Mapping

Each development board has some number of LEDs for status display. As shown in below, the boards discussed in Section 2.1 have differing layout and color of their LEDs. Table 2 shows the "logical" LEDs on each development board – board labels match the "logical" LED number.



SmartRF06 SmartRF05 EXP5438 CC2531DK

Figure 20: Development Board LEDs

Logical LED	SmartRF06	SmartRF05	EXP5438	CC2531DK
LED1	Red	Green	-	-
LED2	Yellow	Red	-	-
LED3	Green	Yellow	Red	Red
LED4	Orange	Red	Yellow	Green

Table 2: Development Board Logical LED Colors

4.2. Running the Generic Application

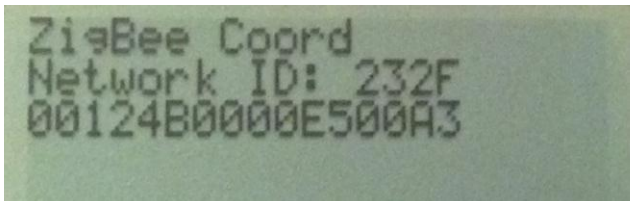
Initially, place all of the programmed devices on the same table or work area, preferably in a location without other ZigBee networks close by. Initially, you will establish a network with the devices in close proximity of each other. Later, you can experiment with moving devices to different distances from each other and try other power-up and "pairing" sequences.

GenericApp responds to user control via switch functionality listed in the table below. Since the CC2531DK device only has 2 buttons, it can't process any SW3, SW4, or SW5 functionality.

Logical Switch	Function	Description
SW1	TIMER	Changes time delay between messages being sent
SW2	BIND	Starts device "pairing" using ZigBee Binding
SW3	-	Not used
SW4	MATCH	Starts device "pairing" using ZigBee Match Descriptor
SW5	-	Not used

Table 3: GenericApp Functional Switch Assignments

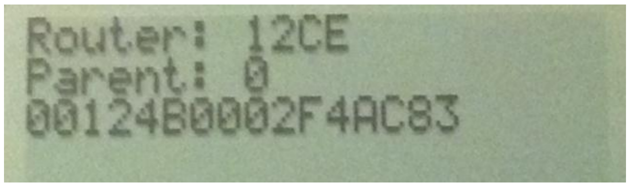
1. Switch on power to the device that was labeled "Coordinator". It will perform a scan of the designated ZigBee channel, looking for other networks already on that channel. After it successfully starts up a new network, LED3 will turn on, indicating that it is ready for other devices to join. The network PanID and the device's 64-bit IEEE address will be displayed on the LCD, similar to:



```
ZigBee Coord
Network ID: 232F
00124B0000E500A3
```

Figure 21: Initial Coordinator LCD Display

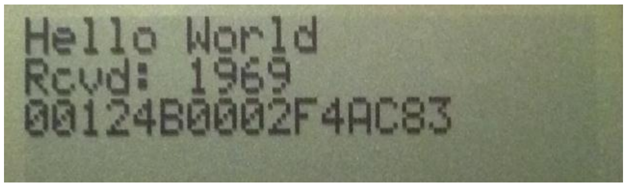
2. Apply power to the device that was labeled "Router". It will scan the designated ZigBee channel, looking for a network to join. In a few seconds it should join the network started by the "Coordinator", indicated by turning on LED3. The LCD display will show the Router's assigned 16-bit network address, its parent's 16-bit network address , and the Router's 64-bit IEEE address, similar to:



```
Router: 12CE
Parent: 0
00124B0002F4AC83
```

Figure 22: Initial Router LCD Display

3. Now that a simple network has been formed, the devices will be "paired" through use of the ZigBee Binding mechanism. This is accomplished by pressing SW2 on both devices within a few seconds of each other - the device binding process will terminate if not completed within a timeout period. When the binding process is successful, LED4 on both development boards will turn on, and a message with the text string "Hello World" will be periodically sent to the other device. When a device receives a message, it will blink LED4 and increment a count of received messages. The LCD display shows the received text string, the number of messages received, and the device's 64-bit IEEE address, similar to:



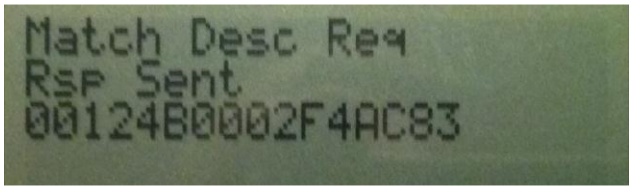
```
Hello World
Rcvd: 1969
00124B0002F4AC83
```

Figure 23: Received Message LCD Display

4. GenericApp comes “out-of-the-box” with a time delay of 5 seconds between messages. That delay can be changed by pressing SW1, which cuts the current timer setting in half. Press SW1 once on the Coordinator and notice that messages are received by the Router twice as often as before. Press SW1 on the Router three times and notice that messages are received by the Coordinator eight times as often as before. Pressing SW1 more times will continue to halve the message delay time until a lower limit is reached – then the timer will be reset to the original “out-of-the-box” value.

5. GenericApp also supports the “pairing” method known as Match Descriptor where a device queries the network for other devices that are capable of communicating with it. To evaluate this feature, turn off both Coordinator and Router devices, then perform steps 1 and 2 above.

6. After the network is back up and running (LED3 lit on both devices), press SW4 on the Coordinator, to start a “pairing” process - the Coordinator will broadcast a Match Descriptor Request message. When received by the Router, it will send back a Match Descriptor Response message to the Coordinator, indicating that it supports the “Hello World” scheme defined in the GenericApp Simple Descriptor. The Router LCD displays its status, similar to:



```
Match Desc Req
Rsp Sent
00124B0002F4AC83
```

Figure 24: Match Descriptor LCD Display

When the Coordinator receives the Match Descriptor Response, it turns on LED4 and starts sending “Hello World” messages to the Router, which displays its received message status as shown in Figure 23. Note that unlike the Binding scenario which enabled message traffic in both directions, Match Descriptor is uni-directional. To start sending messages in both directions, press SW4 on the Router and see that the Coordinator responds as in Figure 24 and later displays message reception status as in Figure 23.

5. Additional Considerations

This section discusses a few topics which affect how GenericApp and other Z-Stack applications operate in simple demonstration examples and later, in larger real-world networks. For more detailed information on Z-Stack application development, refer to the "Z-Stack Developer's Guide" that accompanies all Z-Stack product installations.

5.1. Channel and PanID Selection

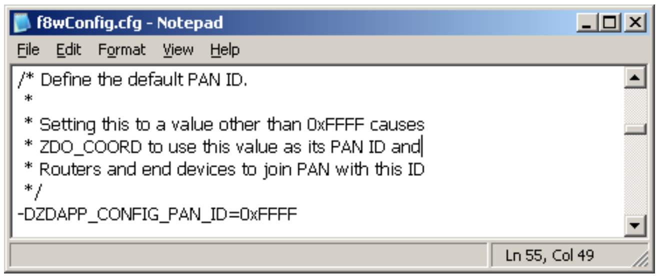
The ZigBee specification defines the use of a 16-bit Personal Area Network Identifier (PanID) to uniquely identify a network. Z-Stack provides the user with two methods of selecting a PanID when starting or joining a network by setting the value of *ZDAPP_CONFIG_PAN_ID*. For a Coordinator device, setting this value to 0xFFFF causes it to start a network with a randomly generated PanID. For a Router device, setting this parameter to 0xFFFF causes the device to join the "best" network it can discover within the specified channel list, any other value causes it to use the exact value specified. The "best" network is defined as the beacon response to scan commands that has the highest received signal strength (RSSI).

The IEEE 802.15.4 specification defines 16 channels in the 2.4 GHz frequency range. These channels are assigned numbers 11 through 26. Z-Stack initially defaults to channel 11, but the user can select a different channel by changing *DEFAULT_CHANLIST*. This parameter is a bit map field, with each bit representing a single channel.

Channel Number	Bit Map Field
11	0x00000800
12	0x00001000
13	0x00002000
14	0x00004000
15	0x00008000
16	0x00010000
17	0x00020000
18	0x00040000
19	0x00080000
20	0x00100000
21	0x00200000
22	0x00400000
23	0x00800000
24	0x01000000
25	0x02000000
26	0x04000000

Table 4: Default ZigBee Channel Select Bit Map

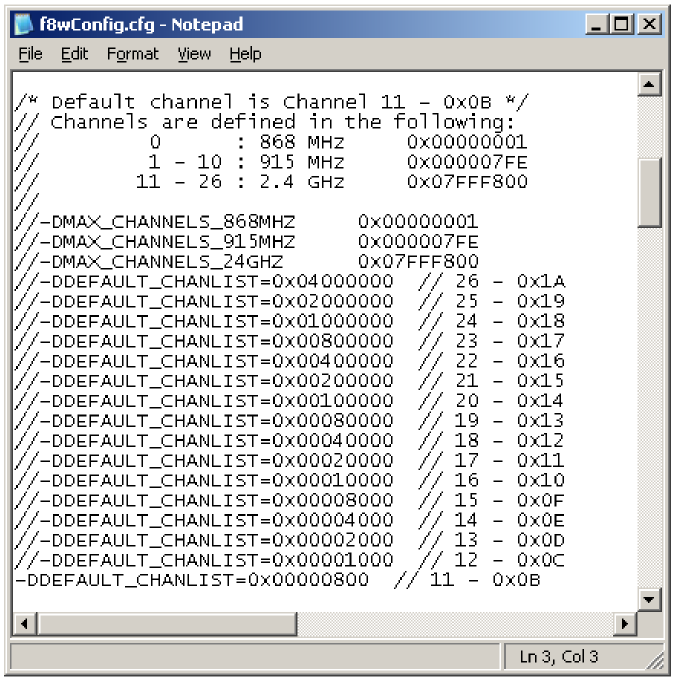
ZDAPP_CONFIG_PAN_ID and *DEFAULT_CHANLIST* may be defined as compile options in the IAR IDE, as well as in a project's configuration command file. Configuration command files are located in the applicable …\Projects\zstack\Tools\<CCxxxx> folder. As shown below, the *f8wConfig.cfg* file specifies the PanID that will be used when the Z-Stack devices start up. This feature allows developers to set up a "personal" PanID to avoid conflicts with others using the same ZigBee channel. The *f8wConfig.cfg* file is the recommended location for developers to establish shared settings for their projects.



```
/* Define the default PAN ID.
 *
 * Setting this to a value other than 0xFFFF causes
 * ZDO_COORD to use this value as its PAN ID and
 * Routers and end devices to join PAN with this ID
 */
-DZDAPP_CONFIG_PAN_ID=0xFFFF
```

Figure 25: PanID Configuration in *f8wconfig.cfg*

As shown below, entries in the *f8wConfig.cfg* file also specify the channel(s) that will be used when the Z-Stack devices start up. This feature allows developers set up a “personal” channel and to avoid conflict with others in the same RF vicinity. Multiple channels can be specified by including the appropriate bits in the *DEFAULT_CHANLIST* definition. As shown below, the default channel 11 is represented by 0x00000800 (11th bit in the field, starting from bit 0).



```
/* Default channel is Channel 11 - 0x0B */
// Channels are defined in the following:
//         0      : 868 MHz     0x00000001
//         1 - 10 : 915 MHz     0x000007FE
//        11 - 26 : 2.4 GHz     0x07FFF800
//
//-DMAX_CHANNELS_868MHZ     0x00000001
//-DMAX_CHANNELS_915MHZ     0x000007FE
//-DMAX_CHANNELS_24GHZ      0x07FFF800
//-DDEFAULT_CHANLIST=0x04000000  // 26 - 0x1A
//-DDEFAULT_CHANLIST=0x02000000  // 25 - 0x19
//-DDEFAULT_CHANLIST=0x01000000  // 24 - 0x18
//-DDEFAULT_CHANLIST=0x00800000  // 23 - 0x17
//-DDEFAULT_CHANLIST=0x00400000  // 22 - 0x16
//-DDEFAULT_CHANLIST=0x00200000  // 21 - 0x15
//-DDEFAULT_CHANLIST=0x00100000  // 20 - 0x14
//-DDEFAULT_CHANLIST=0x00080000  // 19 - 0x13
//-DDEFAULT_CHANLIST=0x00040000  // 18 - 0x12
//-DDEFAULT_CHANLIST=0x00020000  // 17 - 0x11
//-DDEFAULT_CHANLIST=0x00010000  // 16 - 0x10
//-DDEFAULT_CHANLIST=0x00008000  // 15 - 0x0F
//-DDEFAULT_CHANLIST=0x00004000  // 14 - 0x0E
//-DDEFAULT_CHANLIST=0x00002000  // 13 - 0x0D
//-DDEFAULT_CHANLIST=0x00001000  // 12 - 0x0C
-DDEFAULT_CHANLIST=0x00000800  // 11 - 0x0B
```

Figure 26: Channel Configuration in *f8wconfig.cfg*

5.2. Energy Level Detection

The Coordinator will try to start a network on a selected channel with the lowest detected energy level. Energy scans will be attempted until at least one channel is found with energy level below a threshold that increases by ENERGY_SCAN_INCREMENT for each iteration. This constant can be adjusted to alter the granularity of the iterative energy scans – larger numbers get scans done more quickly while potentially increasing the number of less desirable channels. To ensure that the Coordinator finds a suitable channel to start its network on, *DEFAULT_CHANLIST* is recommended to specify more than one channel.

5.3. Preserving Network Configuration

Initialization and preservation of network parameters into Non-Volatile memory (NV) that are established when devices join a network is controlled by a pair of Z-Stack compile options, NV_INIT and NV_RESTORE. To simplify working with sample applications, these options are disabled by default, so if any GenericApp device is power cycled, it will need to perform the network joining process again. In production devices, NV_INIT and NV_RESTORE need to be enabled, so that the configuration will be saved in NV and the user will not need to repeat the initial setup after each power cycle.

5.4. IEEE Address Selection

Every ZigBee device requires a unique 64-bit IEEE address. Z-Stack uses the following four-level hierarchy to determine the IEEE address that will be used for the device when it operates:

1. Read from Z-Stack non-volatile memory
2. Look-up from Secondary IEEE location
3. Look-up from Primary IEEE location (for CC253x devices only)
4. Create temporary using random number generation

Under normal circumstances, when a Z-Stack device boots up, it reads the IEEE address from non-volatile memory (NV) that was stored during a previous “run” of the device. NV memory retains parameters, including the IEEE address, for occasions when the device resets, typically after a power failure. The IEEE address in NV memory gets saved under 3 possible scenarios – initially from steps 2-3 of the hierarchy listed above, or later by delivery from an external source (such as via serial I/O from a PC-hosted program like Z-Tool).

During a device reset process, if the “read from NV memory” operation fails, Z-Stack will first attempt to find an IEEE address at the Secondary IEEE address location (step 2). If that fails, it will then attempt to find an address in the Primary IEEE address location (step 3). Finally, if that fails, Z-Stack will generate a “temporary” address using random numbers (step 4). In steps 2 and 3, the IEEE address gets written to NV memory – on the next device reset this address will be read from NV memory (step 1). In a development environment (NV_RESTORE not used), the temporary IEEE address is not written to NV memory, so each time the device gets reset it will have a different IEEE address than before.

For CC253x devices, the Secondary IEEE address is located on the last page of flash memory. For MSP430F5438 devices, the Secondary IEEE address is located in the last 8 bytes of INFOA flash memory. Secondary IEEE address locations are provided for the user to override the pre-programmed TI Primary address with manufacturer-specific IEEE addresses “at the factory”. Device programming tools that work with Z-Stack should be preserve IEEE addresses located at:

CC2530 (256K)	--	0x3FFE8-0x3FFEF
CC2531 (256K)	--	0x3FFE8-0x3FFEF
CC2538 (512K)	--	0x27FFCC-0x27FFD3
MSP430 (256K)	--	0x19F8-0x19FF

During a device reset process on CC253x devices, if an IEEE address cannot be found in NV memory or in the Secondary IEEE address location, Z-Stack will attempt to read a pre-programmed TI Primary IEEE address from the device "information memory" located at:

CC2530 (256K)	--	0x00C-0x013
CC2531 (256K)	--	0x00C-0x013
CC2538 (512K)	--	0x028-0x02F

Finally, in the rare event that the CC253x device does not have a pre-programmed Primary IEEE address (pre-RTM devices), a temporary 64-bit address beginning with 0xF8 will be generated using the system's random number generator.

Z-Stack permits the IEEE address to be updated in NV memory via the standard *NV_Write* API. This allows developers to modify the address at their discretion, using the serial MT interface via Z-Tool™ or another equivalent mechanism. In the production environment, it's advisable to disable this capability since deployed ZigBee devices should not be permitted to change their factory programmed IEEE address.

5.5. Wireless Product Support

Texas Instruments provides a broad selection of SimpleLink™ Solutions - cost-effective, low-power wireless products for short range, long range, mesh and IP networks and more. Please visit the Wireless Connectivity web page (http://www.ti.com/lsds/ti/wireless_connectivity/overview.page).

TI is a leading supplier of certified ZigBee™ solutions, including Z-Stack™ and TIMAC™ which implement the latest revisions of the ZigBee Pro stack and application profiles. Please visit the ZigBee web page (http://www.ti.com/lsds/ti/wireless_connectivity/zigbee/overview.page).

The TI E2E Community (http://e2e.ti.com/support/wireless_connectivity/default.aspx) provides numerous forums for developers to ask questions, share knowledge, explore ideas, and help solve problems with fellow engineers.

The TI ZigBee E2E Forum (http://e2e.ti.com/support/wireless_connectivity/f/158.aspx) is an active group with many discussions on ZigBee, 802.15.4, and 6LoWPAN software and hardware.

TI E2E™ Community
engineer to engineer, solving problems