Smart Energy Sample Application User's Guide

Document Number: SWRU215

Texas Instruments, Inc.
San Diego, California USA

Revision	Description	Date
1.0	Initial release.	04/06/2009
1.1	Updated for Z-Stack 2.3 release. Document modifications include INTER-PAN support for ESP and IPD, use of End Device Annce message to trigger device discovery instead of Match Descriptor Request.	12/23/2009
1.2	Updated for Z-stack 2.3.1 release. Document modifications include support for installing Certicom keys in the CC2530 lock bits flash page and MSP430 information memory segments. The APP_MSG command is no longer used to retrieve the established link key. Instead, ZDO_GET_LINK_KEY is used.	07/29/2010
1.3	Added “In Premise Display – End Device – OTA” configuration which provides a sample OTA client application. Changed router and end device startup behavior to initiate a match descriptor request to discover the key establishment endpoint instead of assuming a fixed endpoint number.	11/16/2010
1.4	Added Certicom Key memory layout for LM9B96 (section 3.3)	12/21/2010
1.5	Updated Pricing test with Price Acknowledgement message. Added Fast Polling Test section. Updated ESP section with fast poll and prepayment Added Prepayment Test section.	07/12/2011

Table of Contents

Table of Figures

1. Introduction

ZigBee Smart Energy (ZSE) is one of the public application profiles released for the ZigBee 2007 specification. It enables utility companies and their customers to directly communicate with thermostats and other smart appliances; see www.zigbee.org for more information.

The Smart Energy Sample application (part of the Z-Stack installer for ZigBee 2007; covering ZigBee and ZigBee PRO) is the optimal starting point to build your own SE application on top of Texas Instruments' Z-Stack (www.ti.com/z-stack).

1.1. Scope

This document describes how to use the Smart Energy Sample Application and discusses its theory of operation. For a more general description of Smart Energy, the reader is referred to the Zigbee Smart Energy specification available from www.zigbee.org. The reader should also review the Z-Stack Smart Energy Developer's Guide prior to using this document.

There are seven defined application instances within the IAR project:

a. Energy Service Portal (ESP) as a Coordinator
b. Metering Device as a Router and also as an End Device
c. In Premise Display as an End Device
d. In Premise Display as an End Device with OTA cluster client support
e. Programmable Communicating Thermostat (PCT) as an End Device
f. Load Control Device as a Router
g. Range Extender as a Router

Figure 1 shows the usage model of how these sample application instances interact with the ESP.

1.2. Definitions, Abbreviations, Acronyms

Term	Definition
AF	Application Framework
CBKE	Certificate-based Key Establishment
DCO	Digitally Controlled Oscillator
DUT	Device Under Test
ECC	Elliptic Curve Cryptography
ESP	Energy Service Portal. An alternate name is Energy Service Interface (ESI)
HAN	Home Area Network
IPD	In Premise Display
OSAL	Operating System Abstraction Layer
OTA	Over The Air firmware upgrade

PCT	Programmable Communicating Thermostat
SE	Smart Energy
ZCL	ZigBee Cluster Library
ZDO	Zigbee Device Object

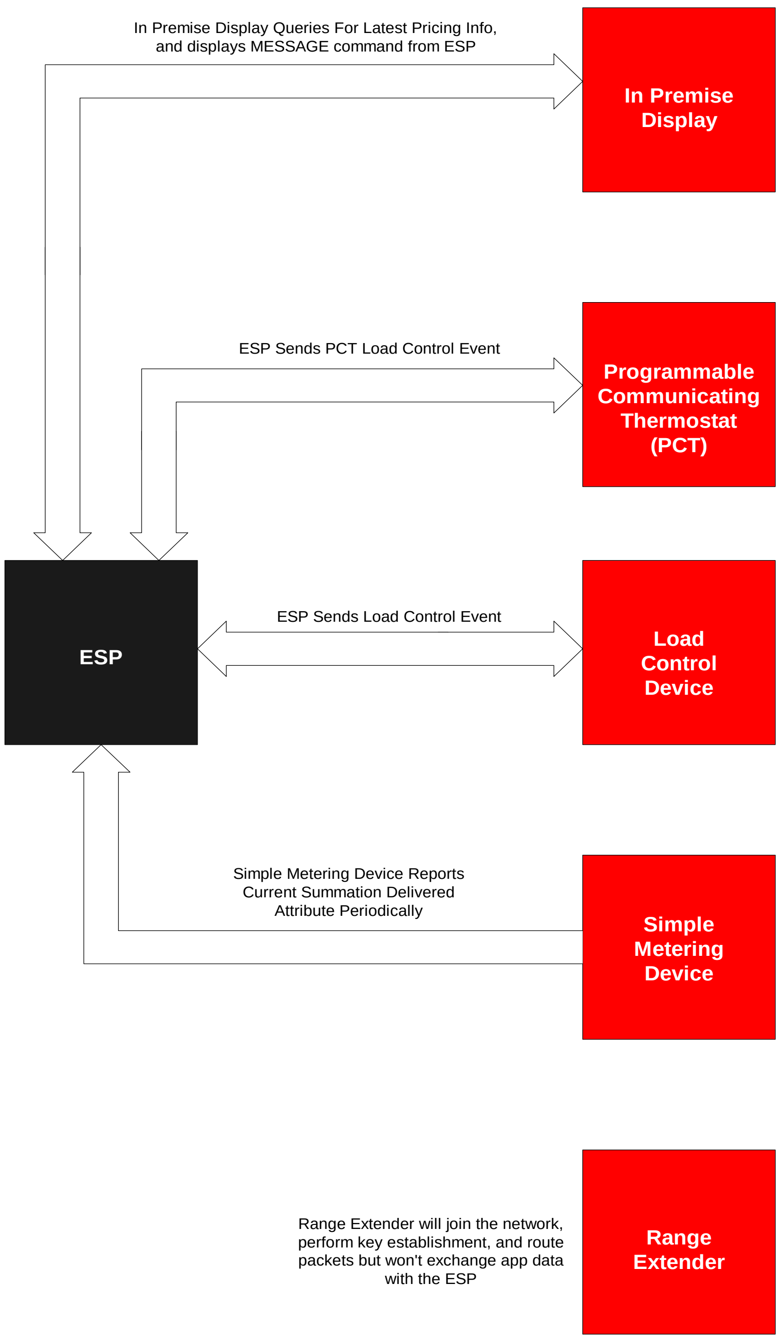

In Premise Display Queries For Latest Pricing Info, and displays MESSAGE command from ESP
In Premise Display
ESP Sends PCT Load Control Event
Programmable Communicating Thermostat (PCT)
ESP
ESP Sends Load Control Event
Load Control Device
Simple Metering Device Reports Current Summation Delivered Attribute Periodically
Simple Metering Device
Range Extender will join the network, perform key establishment, and route packets but won't exchange app data with the ESP
Range Extender

Figure 1. System Context Diagram

2. Setup

2.1. Required Tools

The tools that will be needed to evaluate this sample app and build your own application based on it are the following:

a. IAR Embedded Workbench EW8051 7.60 (CC2530) or EW430 5.10.6 (MSP430)

b. SmartRF Flash Programmer Tool (includes USB drivers for the SmartRF05EB board)

c. Ubiqua Protocol Analyzer from Ubilogix (*www.ubilogix.com*) or other type of network analyzer that can support Smart Energy profile decodes

d. Z-Tool 2.0 (provided as part of the Z-Stack install)

e. Z-Converter – A tool used to transform Certicom certificates data into arrays that can easily be imported into a Z-Tool script (Provided as part of the Z-Stack install)

f. Certicom ECC library if using security. Fill out their SDK license registration form at this URL: http://www.certicom.com/index.php/component/chronocontact/?chronoformname=certicom_zigbee_sdk_registration_form. Alternatively, you may contact TI directly to obtain a special installer that has the Certicom ECC library included.

2.2. Using Z-Converter to Transform Certicom Certificates

In order to build a ZigBee Smart Energy certified device, one must support Certificate Based Key Establishment (CBKE). CBKE requires the use of an ECC library from Certicom and a method to install certificate data into the device. One of the methods used to install Certicom certificate data is by using Z-Tool, a PC tool that can communicate with the device using the Monitor and Test API over a UART port. Test scripts are provided to inject test certificates into the device for development purposes only.

The test certificate data comes in a format that a Z-Tool script does not understand, and therefore a tool called Z-Converter has been provided to take Certicom certificate data as input (the actual input requires no carriage returns in order for Z-Converter to process the data correctly), and then generate arrays that can copied and pasted into a Z-Tool script.

For example, a typical test certificate from Certicom looks like the following:

IEEE Address: 00124b0000000001

CA Pub Key: 0200fde8a7f3d1084224962a4e7c54e69ac3f04da6b8

Device Implicit Cert:
0204ac2c2656f1eea4ff5dac4edda176bfe4fa70d95600124b0000000001544553545345434101090001000001091003

Device Private Key: 00f035a9f731f265530ad5c1202562d56d1b822543

Device Public Key: 0202f71c27abfd28eb39e0b4a718ace4cf374559a6f6

This data must be entered line by line as shown above with no carriage returns into a text file, and the user inputs this data using the “Load” button. Z-Converter then transforms this data into an array output as follows:

IEEE Address: 0x01,0x00,0x00,0x00,0x00,0x4b,0x12,0x00

CA Pub Key: 0x02,0x00,0xfd,0xe8,0xa7,0xf3,0xd1,0x08,0x42,0x24,0x96,0x2a,0x4e,0x7c,0x54,0xe6,0x9a, 0xc3,0xf0,0x4d,0xa6,0xb8

Device Implicit Cert: 0x02,0x04,0xac,0x2c,0x26,0x56,0xf1,0xee,0xa4,0xff,0x5d,0xac,0x4e,0xdd,0xa1,0x76,0xbf, 0xe4,0xfa,0x70,0xd9,0x56,0x00,0x12,0x4b,0x00,0x00,0x00,0x00,0x01,0x54,0x45,0x53,0x54, 0x53,0x45,0x43,0x41,0x01,0x09,0x00,0x01,0x00,0x00,0x01,0x09,0x10,0x03

Device Private Key: 0x00,0xf0,0x35,0xa9,0xf7,0x31,0xf2,0x65,0x53,0x0a,0xd5,0xc1,0x20,0x25,0x62,0xd5,0x6d, 0x1b,0x82,0x25,0x43

Device Public Key:
0x02,0x02,0xf7,0x1c,0x27,0xab,0xfd,0x28,0xeb,0x39,0xe0,0xb4,0xa7,0x18,0xac,0xe4,0xcf,0x37,0x45,0x59,0xa6,0xf6

Note that the Device Public Key is not used as part of the input into the Certicom library but is provided for completeness.

The user then copies these values into a Z-Tool script that uses arrays such as the following:

var ieee = [**0x01,0x00,0x00,0x00,0x00,0x4b,0x12,0x00**];

var data0x69 = [**0x02,0x04,0xac,0x2c,0x26,0x56,0xf1,0xee,0xa4,0xff,0x5d,0xac,0x4e,0xdd,0xa1,0x76,0xbf, 0xe4,0xfa,0x70,0xd9,0x56,0x00,0x12,0x4b,0x00,0x00,0x00,0x00,0x01,0x54,0x45,0x53,0x54, 0x53,0x45,0x43,0x41,0x01,0x09,0x00,0x01,0x00,0x00,0x01,0x09,0x10,0x03**];

var data0x6a = [**0x00,0xf0,0x35,0xa9,0xf7,0x31,0xf2,0x65,0x53,0x0a,0xd5,0xc1,0x20,0x25,0x62,0xd5,0x6d, 0x1b,0x82,0x25,0x43**];

var data0x6b = [**0x02,0x00,0xfd,0xe8,0xa7,0xf3,0xd1,0x08,0x42,0x24,0x96,0x2a,0x4e,0x7c,0x54,0xe6,0x9a, 0xc3,0xf0,0x4d,0xa6,0xb8**];

The mapping of the labels from the Certicom certificate to the Z-Tool script is:

IEEE -> var ieee
Device Implicit Cert -> var data 0x69
Device Private Key -> var data 0x6a
CA Pub Key -> var data0x6b

Figure 2 provides a screenshot of the Z-Converter graphical interface. The user can also save the output into a text file for later use using the “Save As” button.

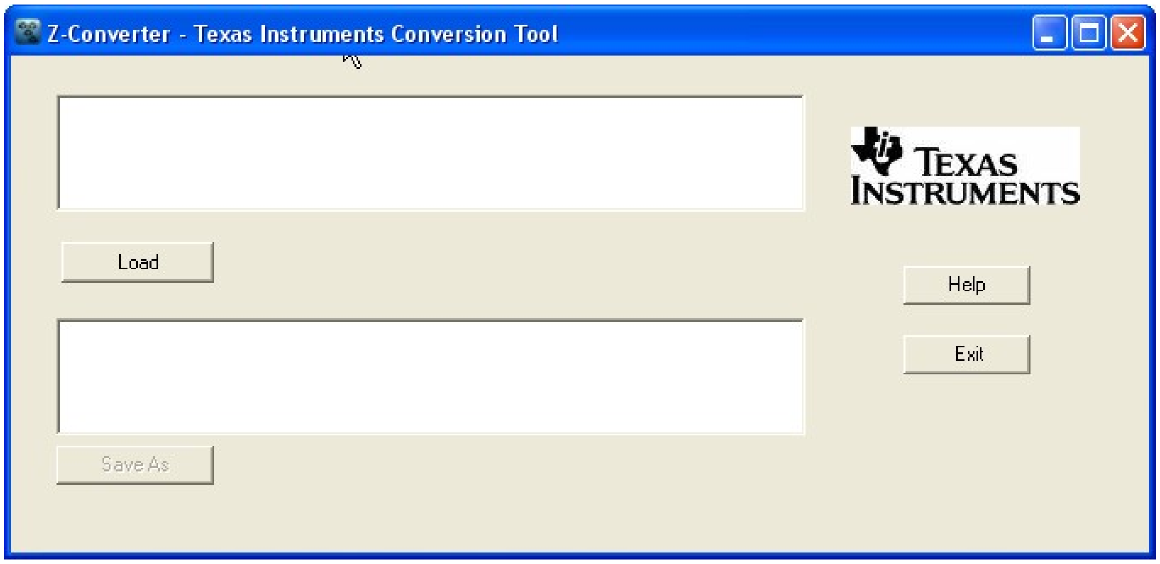


Figure 2. Z-Converter Graphical Interface

Z-Tool is used to program the Certicom Certificate information into Z-stack Non-Volatile memory. A serial cable is required to allow Z-Tool to communicate with the stack via the DB-9 connector on the SmartRF05EB board. The MSP5438 Experimenter's board uses the mini-USB connector with the supported virtual COM port driver available from the TI website.

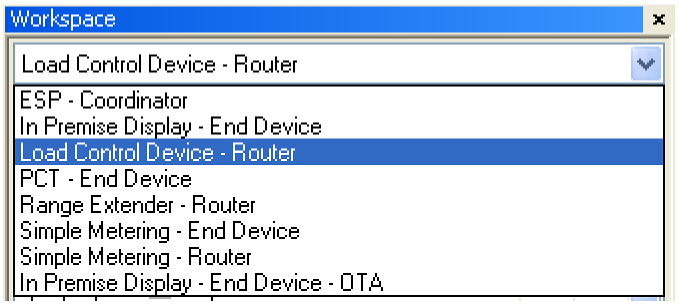
2.3. Supported Hardware Platforms

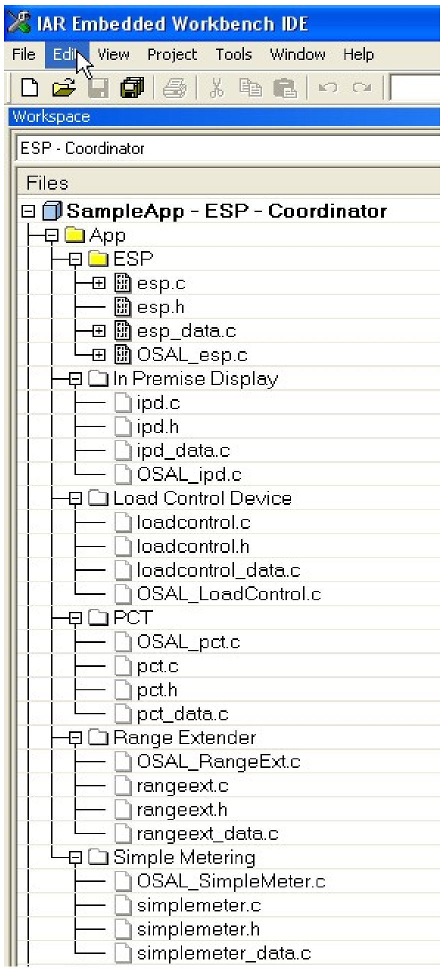
a. SmartRF05EB/BB boards with CC2530EM
b. SmartRF05EB boards with CCMSP2618 + CC2520EM
c. MSP5438 Experimenter boards with CC2520EM

3. Getting Started

3.1. Building the ESP, PCT, and IPD Application Instances

The Smart Energy sample application project is located in C:\Texas Instruments\ZStack-2.4.0-1.4.0\Projects\zstack\SE\SampleApp\<Target> (e.g. <Target> = CC2530DB). Open the SampleApp.eww project file. The figures below show how each configuration is organized. For each device configuration, only the relevant application files are brought in for each device configuration.

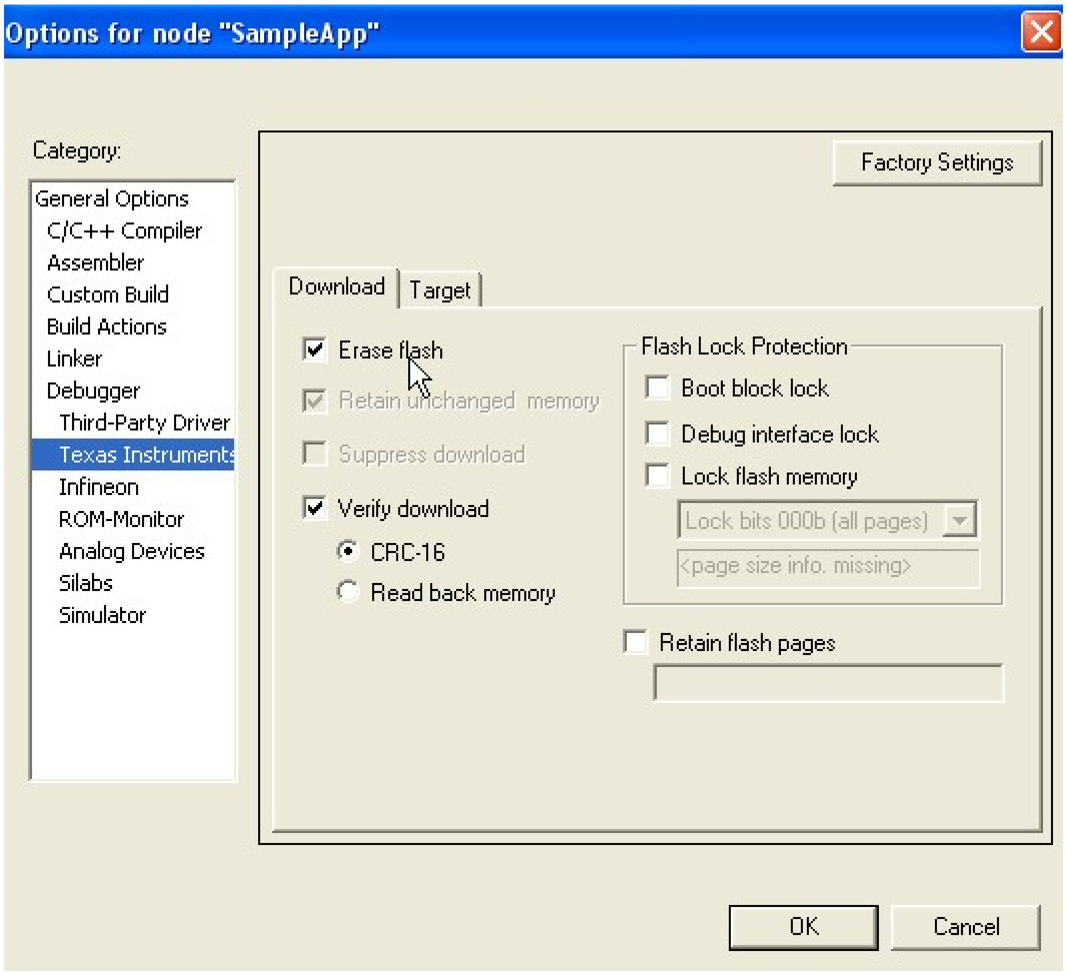

Workspace
Load Control Device - Router
ESP - Coordinator
In Premise Display - End Device
Load Control Device - Router
PCT - End Device
Range Extender - Router
Simple Metering - End Device
Simple Metering - Router
In Premise Display - End Device - OTA



Build the “ESP – Coordinator”, “PCT – End Device”, and “In Premise Display – End Device” configurations.

1. If the SECURE=1 compile option is used, before rebuilding the project, take the Certicom library and rename it to ecc.r51 (for 8051 builds) or ecc.r43 (for MSP430 builds) and drop it into C:\Texas Instruments\ZStack-2.4.0-1.4.0\Projects\zstack\Libraries\<Target>\bin (e.g. <Target> = CC2530DB).
2. The NWK_INDIRECT_MSG_TIMEOUT parameter in the f8wConfig.cfg file must be increased to a suggested value of 10 in order to buffer the message long enough to accommodate the 8 second poll period by the end device. The MAX_POLL_FAILURE_RETRIES parameter should also be set to 4 as during the CBKE procedure the ESP will be somewhat unresponsive to data requests during this time.
3. To gain access to the link key that is established via CBKE using Z-Tool, the compile options MT_ZDO_FUNC and MT_SYS_KEY_MANAGEMENT must be defined. For production, it is not recommended to turn on the MT_SYS_KEY_MANAGEMENT compile option so that the established link key cannot be read via the serial port.
4. Select Project -> Rebuild All.

5. Highlight the configuration desired and go to Project -> Options and select the "Erase flash" option as shown for the CC2530 project.



Options for node "SampleApp"

Category:

- General Options
  - C/C++ Compiler
  - Assembler
  - Custom Build
  - Build Actions
  - Linker
  - Debugger
    - Third-Party Driver
    - Texas Instruments
    - Infineon
    - ROM-Monitor
    - Analog Devices
    - Silabs
    - Simulator

Factory Settings

Download | Target

- [✓] Erase flash
- [✓] Retain unchanged memory
- [ ] Suppress download
- [✓] Verify download
  - (●) CRC-16
  - ( ) Read back memory

Flash Lock Protection

- [ ] Boot block lock
- [ ] Debug interface lock
- [ ] Lock flash memory
  - Lock bits 000b (all pages)
  - <page size info. missing>

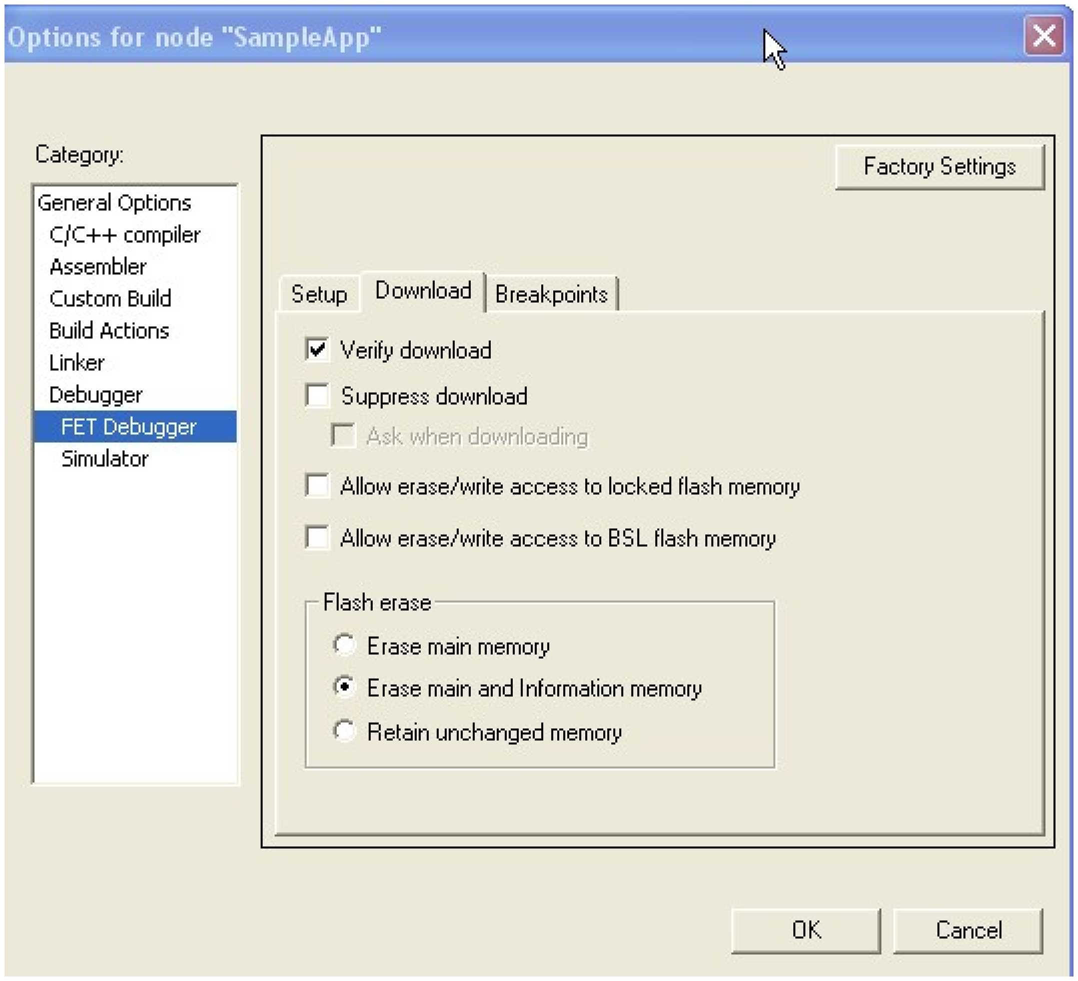
- [ ] Retain flash pages

OK | Cancel

This allows one to start with a clean slate before programming Certicom certs. After the device has been configured, unselect this box so you can retain the IEEE address and key information when downloading new code each time.

For MSP5438 projects, select "Erase main and information memory" as shown below.

IMPORTANT NOTE: For MSP430 users that have ADC calibration info stored in INFOA information memory (such as MSP430F2618 users), **DO NOT** select this option unless you are sure that the INFOA information memory segment is locked. Otherwise you will erase the ADC calibration info and DCO constants and may have to restore it. From the factory, the INFO information memory segment is typically locked, so unless the user has unlocked this segment, there is no danger of overwriting this region of information memory.



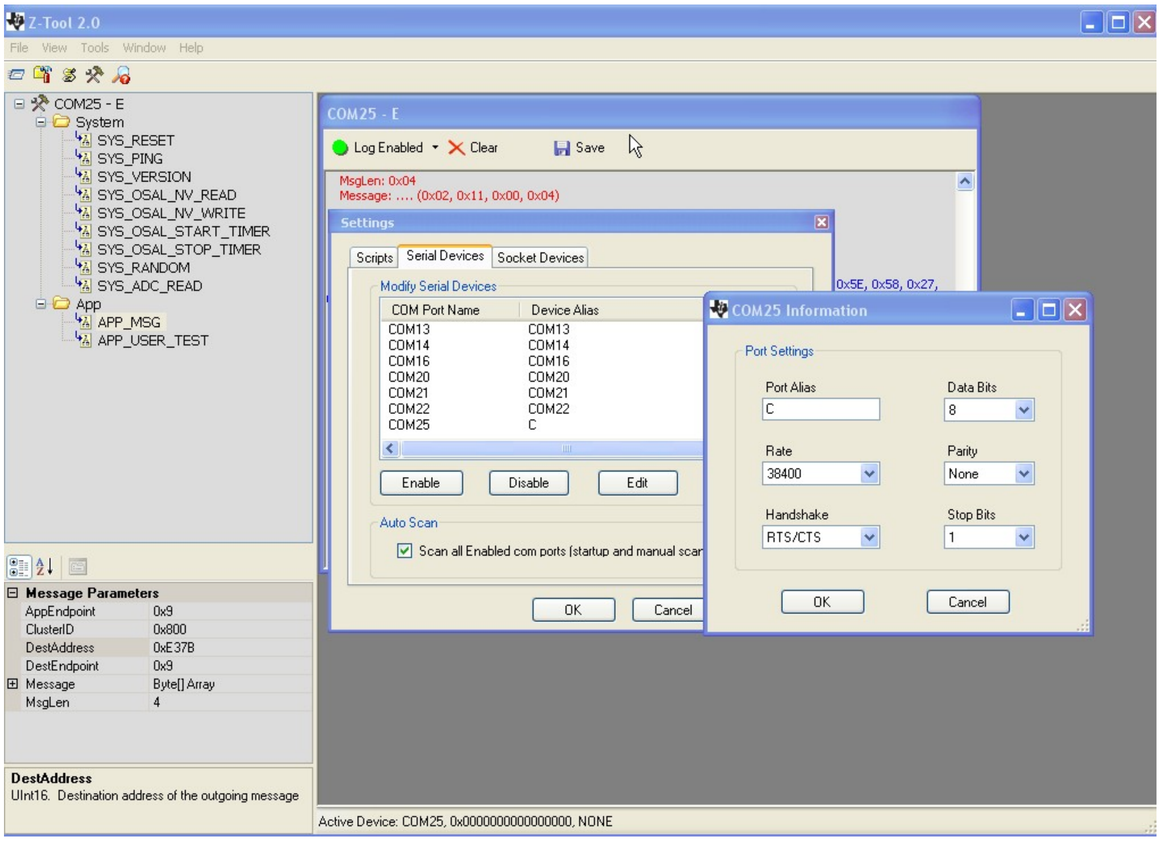
[✓] Verify download
[ ] Suppress download
[ ] Ask when downloading
[ ] Allow erase/write access to locked flash memory
[ ] Allow erase/write access to BSL flash memory

Flash erase
○ Erase main memory
● Erase main and Information memory
○ Retain unchanged memory

6. Download the program by clicking on the debug icon or by going to Project -> Debug. Hit the Run button, and watch the coordinator start up, or if it's a device, it will blink its LED waiting for you to press SW1 to join an existing network.

3.2. Configuring Certicom Keys Using Z-Tool

In order to configure the IEEE address and Certicom keys for each device, use the provided Z-Tools scripts and Z-Tool as shown in the following screen captures:



MsgLen: 0x04
Message: .... (0x02, 0x11, 0x00, 0x04)
0x5E, 0x58, 0x27,

COM Port Name	Device Alias
COM13	COM13
COM14	COM14
COM16	COM16
COM20	COM20
COM21	COM21
COM22	COM22
COM25	C

☑ Scan all Enabled com ports (startup and manual scan

Port Alias	C
Data Bits	8
Rate	38400
Parity	None
Handshake	RTS/CTS
Stop Bits	1

Message Parameters	
AppEndpoint	0x9
ClusterID	0x800
DestAddress	0xE37B
DestEndpoint	0x9
Message	Byte[] Array
MsgLen	4

DestAddress
UInt16. Destination address of the outgoing message

Active Device: COM25, 0x0000000000000000, NONE

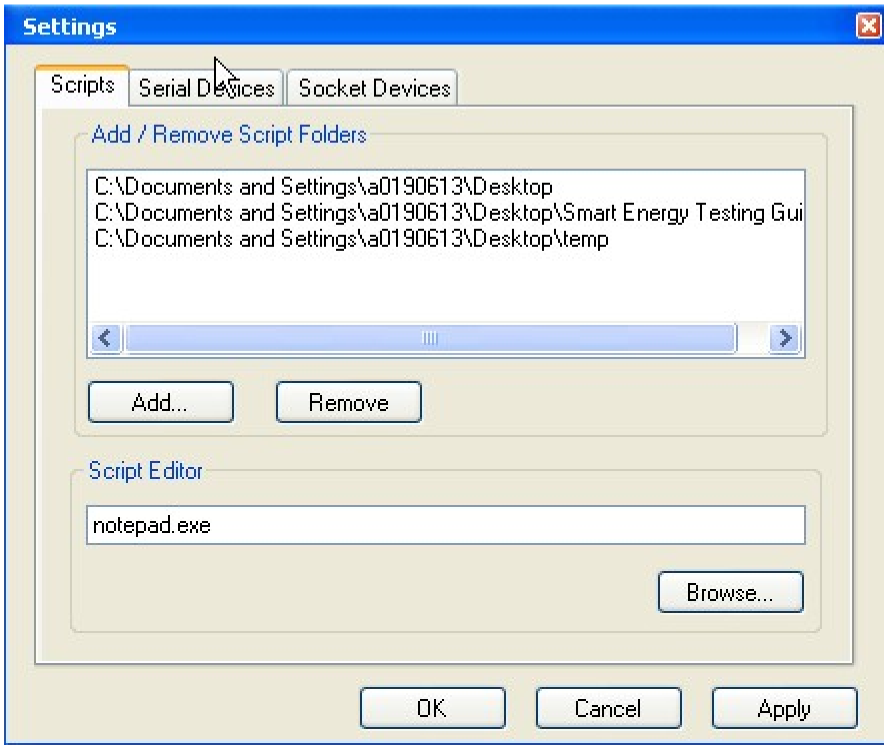
Open Z-Tool from the Texas Instruments start menu links and select Tools -> Settings to configure the baud rate, Port Alias, and Handshake protocol as shown above. It is crucial that the Port Alias matches what is used in the script.

Then as shown in the figure below, go to the Settings -> Scripts tab to import the scripts. Point it to the folder where the scripts are located by clicking on add.

There are already pre-made Z-Tool scripts which you can use in the folder "Z-Tool scripts" in ZStack-2.4.0-1.4.0\Tools\Z-Converter. There is one for each type of device, coordinator, router, and end device.

Coordinator = IEEE address ending with 01 (Port Alias = “C”)
Router = IEEE address ending with 02 (Port Alias = “R”)
End Device = IEEE address ending with 03 (Port Alias = “E”)

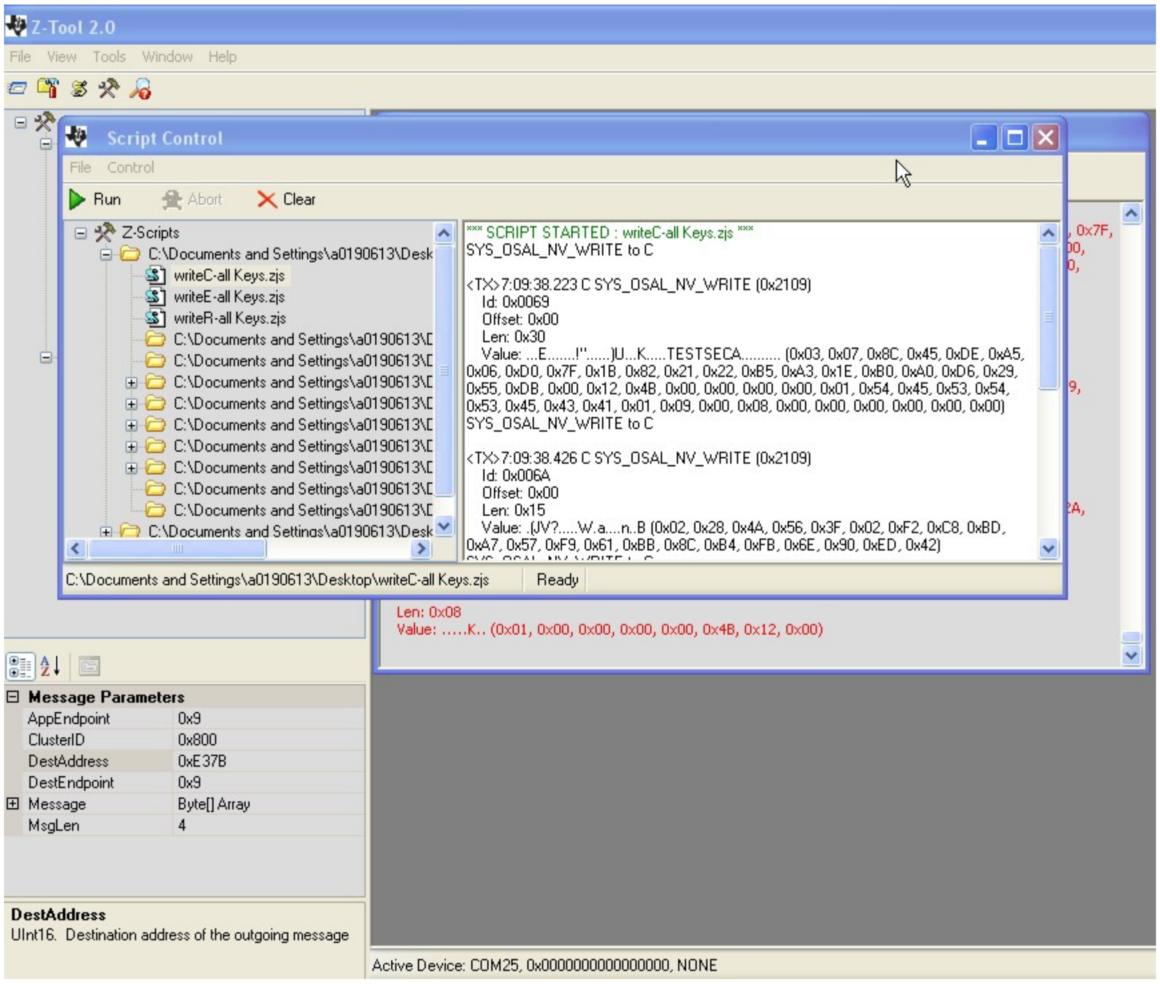
Of course, there is no reason that you couldn't interchangeably use the scripts, but using it in this order makes it easier to keep track of configuration of certs.



C:\Documents and Settings\a0190613\Desktop
C:\Documents and Settings\a0190613\Desktop\Smart Energy Testing Gui
C:\Documents and Settings\a0190613\Desktop\temp

notepad.exe

Then, go to Tools -> Script Control and execute the script called “writeC-all Keys.zjs” by highlighting it and clicking on the RUN button.



```
*** SCRIPT STARTED : writeC-all Keys.zjs ***
SYS_OSAL_NV_WRITE to C

<TX>7:09:38.223 C SYS_OSAL_NV_WRITE (0x2109)
   Id: 0x0069
   Offset: 0x00
   Len: 0x30
   Value: ...E.......!"......)U...K.....TESTSECA.......... (0x03, 0x07, 0x8C, 0x45, 0xDE, 0xA5,
0x06, 0xD0, 0x7F, 0x1B, 0x82, 0x21, 0x22, 0xB5, 0xA3, 0x1E, 0xB0, 0xA0, 0xD6, 0x29,
0x55, 0xDB, 0x00, 0x12, 0x4B, 0x00, 0x00, 0x00, 0x00, 0x01, 0x54, 0x45, 0x53, 0x54,
0x53, 0x45, 0x43, 0x41, 0x01, 0x09, 0x00, 0x08, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00)
SYS_OSAL_NV_WRITE to C

<TX>7:09:38.426 C SYS_OSAL_NV_WRITE (0x2109)
   Id: 0x006A
   Offset: 0x00
   Len: 0x15
   Value: .(JV?.....W.a....n..B (0x02, 0x28, 0x4A, 0x56, 0x3F, 0x02, 0xF2, 0xC8, 0xBD,
0xA7, 0x57, 0xF9, 0x61, 0xBB, 0x8C, 0xB4, 0xFB, 0x6E, 0x90, 0xED, 0x42)
```

C:\Documents and Settings\a0190613\Desktop\writeC-all Keys.zjs   Ready

```
Len: 0x08
Value: .....K.. (0x01, 0x00, 0x00, 0x00, 0x00, 0x4B, 0x12, 0x00)
```

Message Parameters	
AppEndpoint	0x9
ClusterID	0x800
DestAddress	0xE37B
DestEndpoint	0x9
Message	Byte[] Array
MsgLen	4

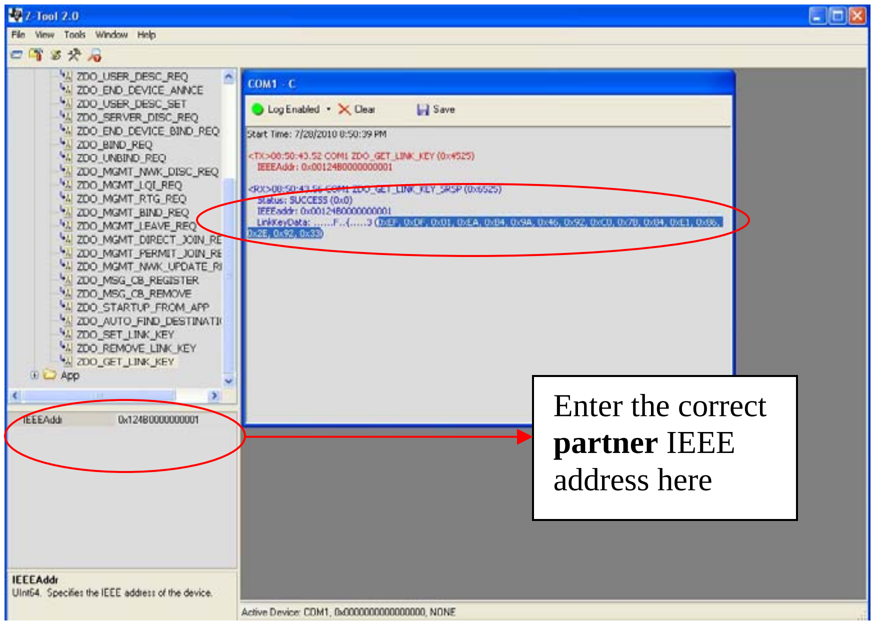
DestAddress
UInt16. Destination address of the outgoing message

Active Device: COM25, 0x0000000000000000, NONE

Repeat this procedure for each device you wish to configure with the IEEE address and certificate data.

Note: *For the CC2530 platform, if Z-Tool is not used to populate the IEEE address and certificate data, then the pre-defined data in zcl_cert_data.c will be used to initialize the corresponding NV items. This is the out-of-box behavior that is enabled via the TC_LINKKEY_JOIN compile option.*

Once key establishment is successful, use the ZDO_GET_LINK_KEY command to extract the link key established for the device. This can be run on any of the pair of devices in order to get the key (needed for the packet sniffer as described below). Configure the ZDO_GET_LINK_KEY command as shown below. The only value that changes is the IEEEAddress of the partner device.



IEEEAddr 0x12480000000001

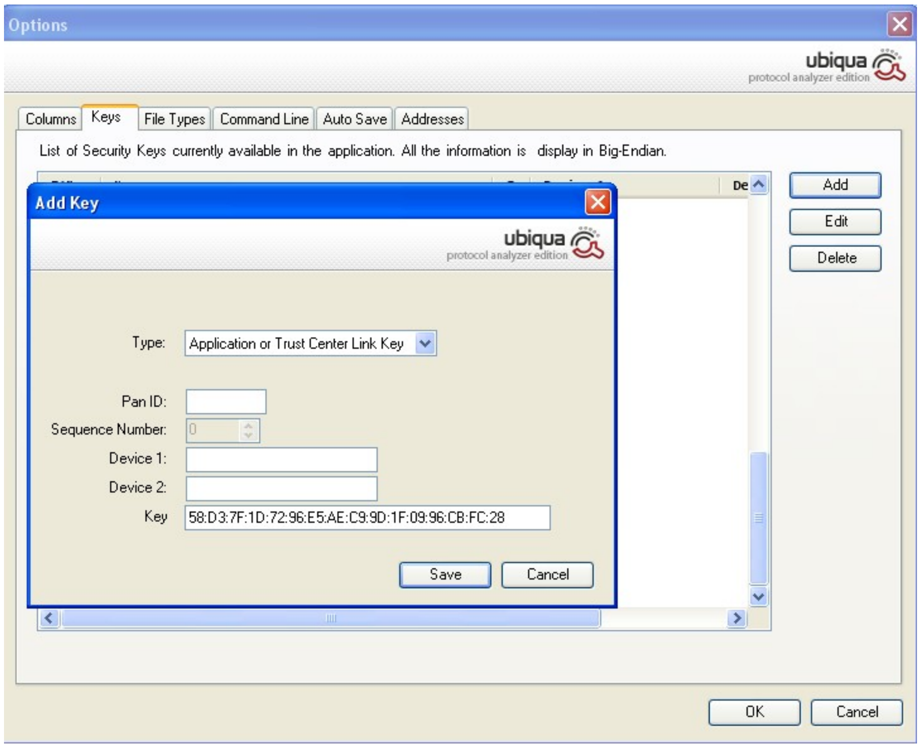
```
Start Time: 7/28/2010 0:50:39 PM

<TX>00:50:43.52 COM1 ZDO_GET_LINK_KEY (0x4525)
   IEEEAddr: 0x00124B0000000001

<RX>00:50:43.56 COM1 ZDO_GET_LINK_KEY_SRSP (0x6525)
   Status: SUCCESS (0x0)
   IEEEaddr: 0x00124B0000000001
   LinkKeyData: ......F..{.....3 (0xEF, 0xDF, 0x01, 0xEA, 0xD4, 0x9A, 0x46, 0x92, 0xC0, 0x70, 0xD4, 0xE1, 0x86,
0x2E, 0x92, 0x33)
```

IEEEAddr
UInt64. Specifies the IEEE address of the device.

Active Device: COM1, 0x0000000000000000, NONE

In order to decode APS secured packets in Ubiqua, one must enter in this key. The best way to format this key is to copy and paste it into notepad, and just replace “0x” with blank space, then “,” with blank space. If the link key extraction is not successful, the string of bytes (16 total) will not be displayed.

Copy and paste the key into Ubiqua:



List of Security Keys currently available in the application. All the information is display in Big-Endian.

Type:	Application or Trust Center Link Key
Pan ID:	
Sequence Number:	0
Device 1:	
Device 2:	
Key	58:D3:7F:1D:72:96:E5:AE:C9:9D:1F:09:96:CB:FC:28

3.3. Configuring Certicom Keys for Production Devices

Section 3.2 discussed a method for configuring Certicom keys. This works well for the development phase, but for production, a different solution is often desired to help streamline the manufacturing process. For this reason, and also for enhanced security, Z-stack provides an option to store the Certicom keys in the lock bits page of the CC2530, information memory of the MSP430, and the last flash page of the Stellaris LM9B96. This is also the recommended option as each family of chips has a locking mechanism to prevent direct access of code space from their respective debug ports (such as JTAG).

For the CC2530, the following locations in the last available page of flash memory are used:

```
// RESERVED for future Z-Stack use:               0x3F800 - 0x3FF8B
#define HAL_NV_IMPLICIT_CERTIFICATE_ADDR          0x3FF8C
#define HAL_NV_CA_PUBLIC_KEY_ADDR                 0x3FFBC
#define HAL_NV_DEVICE_PRIVATE_KEY_ADDR            0x3FFD2
#define HAL_NV_IEEE_ADDR                          0x3FFE8
#define HAL_FLASH_LOCK_BITS                       0x3FFF0
```

For the LM9B96, the following locations in the last available page of flash memory are used:

```
// RESERVED for future Z-Stack use:               0x3FC00 - 0x3FF97
#define HAL_NV_IMPLICIT_CERTIFICATE_ADDR          0x3FF98
#define HAL_NV_CA_PUBLIC_KEY_ADDR                 0x3FFC8
#define HAL_NV_DEVICE_PRIVATE_KEY_ADDR            0x3FFE0
#define HAL_NV_IEEE_ADDR                          0x3FFF8
```

For the 54xx, the following locations in the INFOA memory are used:

```
// RESERVED for future Z-Stack use:               0x1980 - 0x199C
#define HAL_NV_DEVICE_PRIVATE_KEY_ADDR            0x199D
#define HAL_NV_CA_PUBLIC_KEY_ADDR                 0x19B2
#define HAL_NV_IMPLICIT_CERTIFICATE_ADDR          0x19C8
#define HAL_NV_IEEE_ADDR                          0x19F8
```

For the 2618, the following locations in the info memory are used:

```
#define HAL_NV_IEEE_ADDR                          0x1000
#define HAL_NV_IMPLICIT_CERTIFICATE_ADDR          0x1008
#define HAL_NV_CA_PUBLIC_KEY_ADDR                 0x1038
#define HAL_NV_DEVICE_PRIVATE_KEY_ADDR            0x104E
// RESERVED for future Z-Stack use:               0x1063 - 0x10D9
// MSP Calibration Data (ADC/DCO):                0x10DA - 0x10FF
```

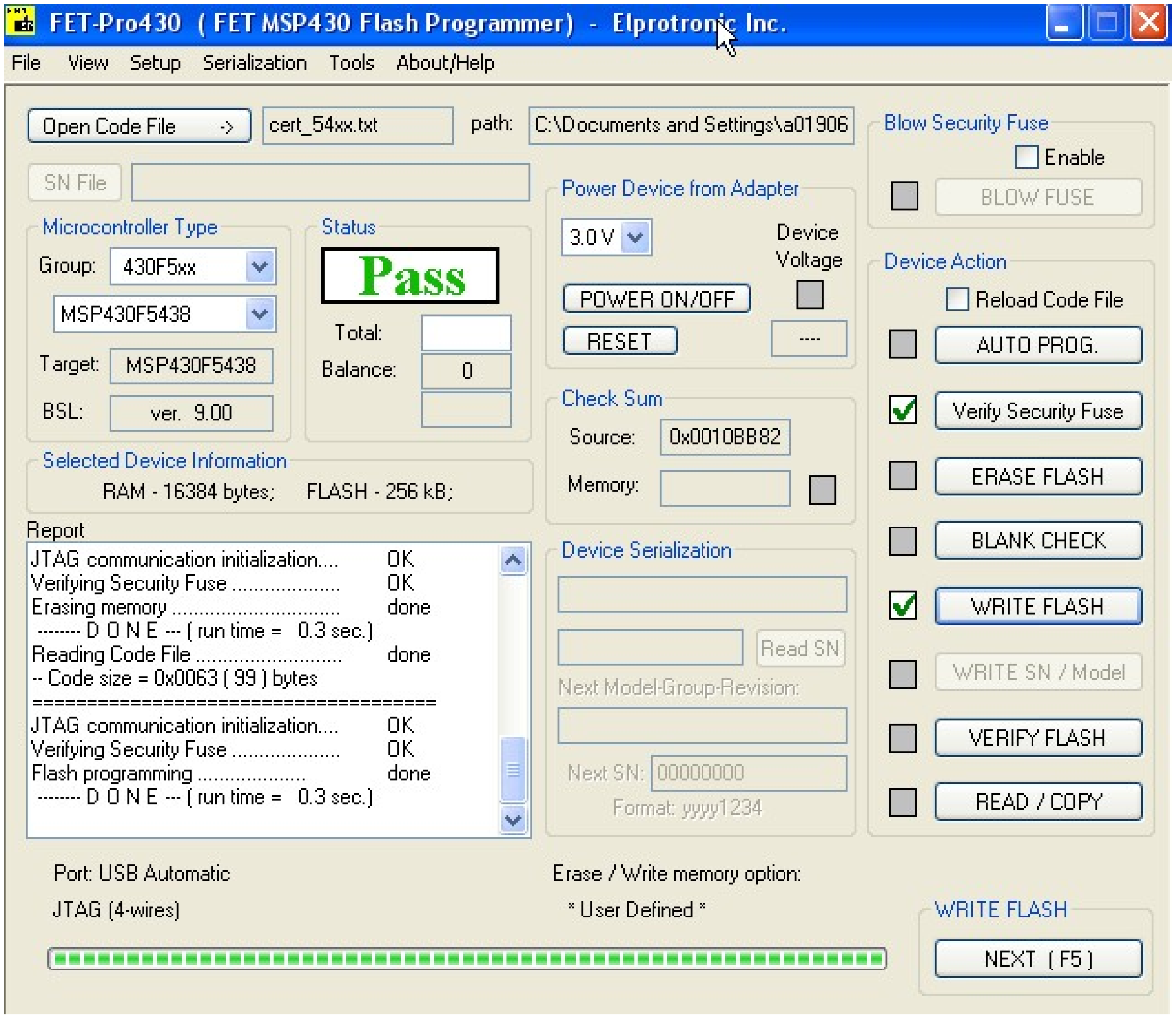
Populating the Certicom key data in the MSP430 info memory can be done via a 3rd party programmer tool like the FET-Pro430 from Elprotronic Inc (www.elprotronic.com). Consult section 6.1 of the FET-Pro430 user manual for more details on how to generate a code file to update the corresponding sections of info memory only. The following sections below provide example code files and screenshots of the FET-Pro430 setup for the 2618 and 54xx devices.

IMPORTANT NOTE: For best results, one should download the program first using IAR (ensuring to erase flash), then run the FET-Pro430 to update the various information memory segments. This is because when Z-stack first starts, NV items are created and populated with the values read from the information memory segments.

54xx Fet-Pro430 sample code file

```
@19F8
01 00 00 00 00 4b 12 00
@19C8
03 07 8c 45 de a5 06 d0 7f 1b 82 21 22 b5 a3 1e b0 a0 d6 29 55 db 00 12 4b 00 00 00 00 01 54
45 53 54 53 45 43 41 01 09 00 08 00 00 00 00 00 00
@19B2
02 00 fd e8 a7 f3 d1 08 42 24 96 2a 4e 7c 54 e6 9a c3 f0 4d a6 b8
@199D
02 28 4a 56 3f 02 f2 c8 bd a7 57 f9 61 bb 8c b4 fb 6e 90 ed 42
q
```

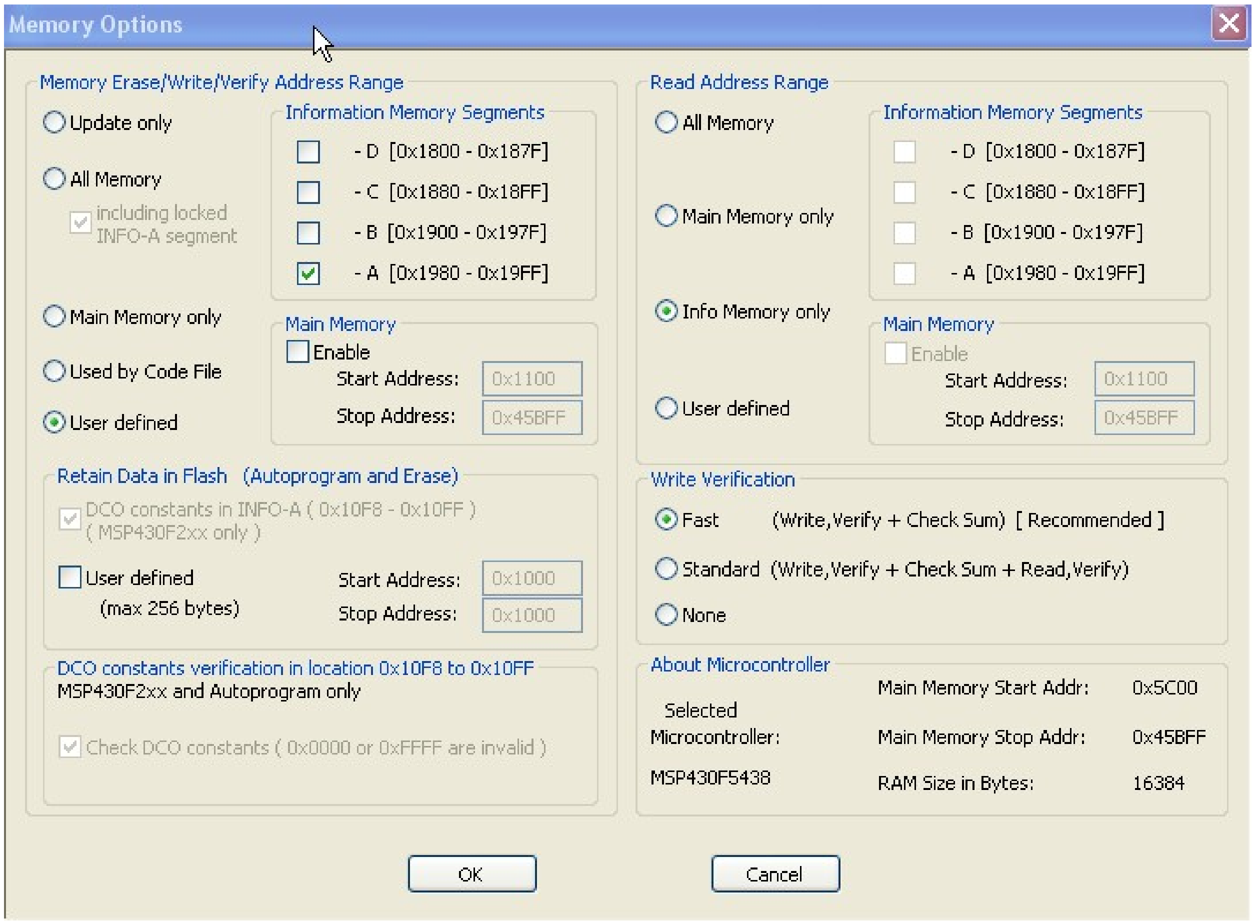
54xx Fet-Pro430 configuration screenshots



cert_54xx.txt
path: C:\Documents and Settings\a01906
Enable: [ ]
Microcontroller Type
Group: 430F5xx
MSP430F5438
Target: MSP430F5438
BSL: ver. 9.00
Status: Pass
Total:
Balance: 0
3.0 V
Device Voltage
Reload Code File: [ ]
Verify Security Fuse: [✓]
WRITE FLASH: [✓]
Check Sum
Source: 0x0010BB82
Memory:
Selected Device Information
RAM - 16384 bytes; FLASH - 256 kB;
Device Serialization
Next Model-Group-Revision:
Next SN: 00000000
Format: yyyy1234

Report
```
JTAG communication initialization....    OK
Verifying Security Fuse ...................   OK
Erasing memory ..............................   done
 ------- D O N E --- ( run time =   0.3 sec.)
Reading Code File ..........................   done
-- Code size = 0x0063 ( 99 ) bytes
=====================================
JTAG communication initialization....    OK
Verifying Security Fuse ...................   OK
Flash programming ...................   done
 ------- D O N E --- ( run time =   0.3 sec.)
```

Port: USB Automatic
JTAG (4-wires)
Erase / Write memory option:
* User Defined *
WRITE FLASH


Memory Options

Memory Erase/Write/Verify Address Range
- Update only
- All Memory
  - including locked INFO-A segment ✓
- Main Memory only
- Used by Code File
- [✓ User defined]

Information Memory Segments
- ☐ - D [0x1800 - 0x187F]
- ☐ - C [0x1880 - 0x18FF]
- ☐ - B [0x1900 - 0x197F]
- ✓ - A [0x1980 - 0x19FF]

Main Memory
- ☐ Enable
- Start Address: 0x1100
- Stop Address: 0x45BFF

Retain Data in Flash (Autoprogram and Erase)
- ✓ DCO constants in INFO-A (0x10F8 - 0x10FF) (MSP430F2xx only)
- ☐ User defined (max 256 bytes) Start Address: 0x1000 Stop Address: 0x1000

DCO constants verification in location 0x10F8 to 0x10FF
MSP430F2xx and Autoprogram only
- ✓ Check DCO constants (0x0000 or 0xFFFF are invalid)

Read Address Range
- All Memory
- Main Memory only
- [✓ Info Memory only]
- User defined

Information Memory Segments
- ☐ - D [0x1800 - 0x187F]
- ☐ - C [0x1880 - 0x18FF]
- ☐ - B [0x1900 - 0x197F]
- ☐ - A [0x1980 - 0x19FF]

Main Memory
- ☐ Enable
- Start Address: 0x1100
- Stop Address: 0x45BFF

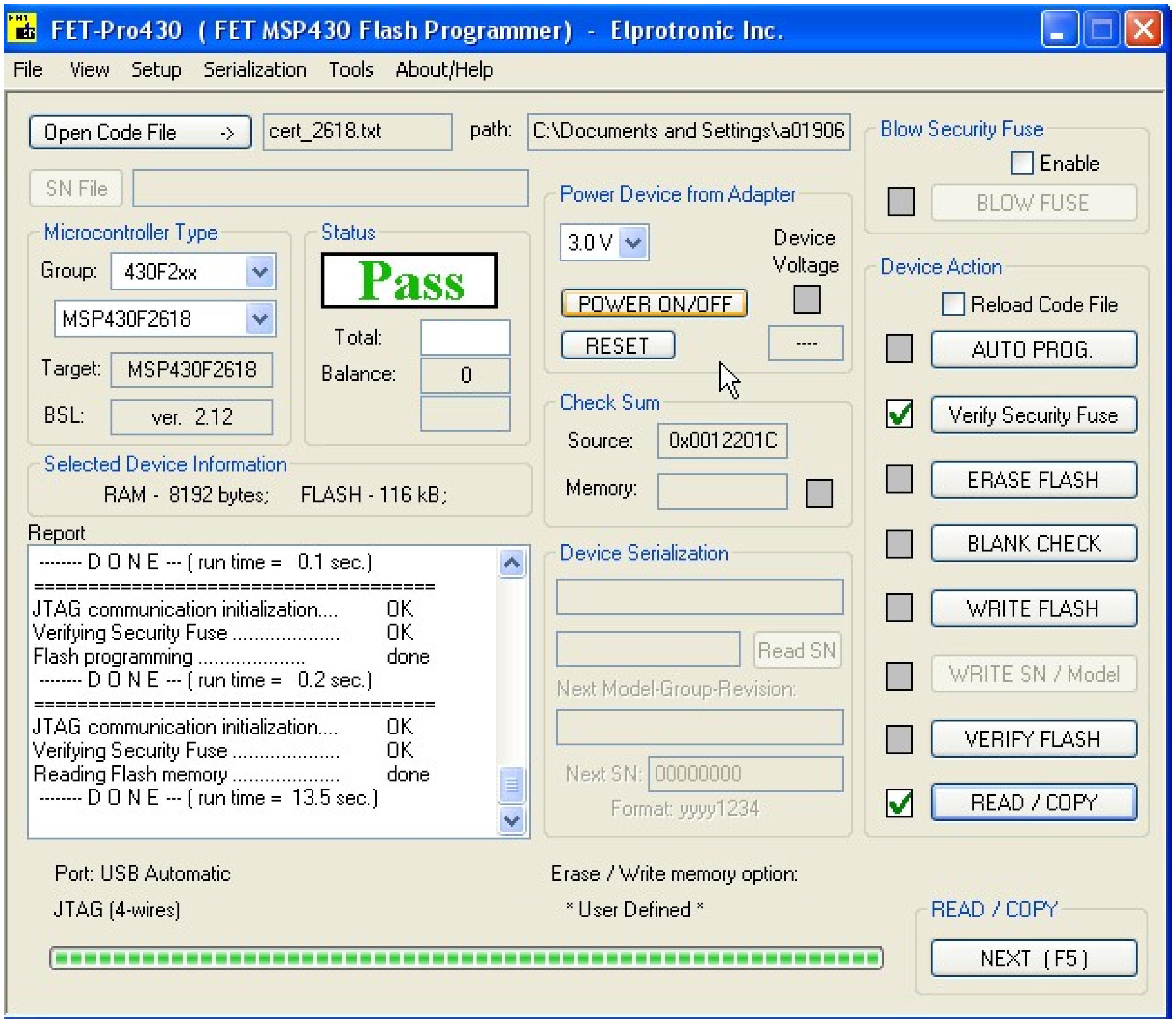
Write Verification
- [✓ Fast] (Write,Verify + Check Sum) [Recommended]
- Standard (Write,Verify + Check Sum + Read,Verify)
- None

About Microcontroller
- Selected Microcontroller: MSP430F5438
- Main Memory Start Addr: 0x5C00
- Main Memory Stop Addr: 0x45BFF
- RAM Size in Bytes: 16384

2618 Fet-Pro430 sample code file

```
@1000
01 00 00 00 00 4b 12 00
@1008
03 07 8c 45 de a5 06 d0 7f 1b 82 21 22 b5 a3 1e b0 a0 d6 29 55 db 00 12 4b 00 00 00 00 01 54
45 53 54 53 45 43 41 01 09 00 08 00 00 00 00 00 00
@1038
02 00 fd e8 a7 f3 d1 08 42 24 96 2a 4e 7c 54 e6 9a c3 f0 4d a6 b8
@104E
02 28 4a 56 3f 02 f2 c8 bd a7 57 f9 61 bb 8c b4 fb 6e 90 ed 42
q
```

2618 Fet-Pro430 configuration screenshots



Open Code File: cert_2618.txt path: C:\Documents and Settings\a01906

Microcontroller Type
Group: 430F2xx
MSP430F2618
Target: MSP430F2618
BSL: ver. 2.12

Status: Pass
Total:
Balance: 0

Power Device from Adapter: 3.0 V
Device Voltage: ----

Check Sum
Source: 0x0012201C
Memory:

Blow Security Fuse: Enable [ ]

Device Action: Reload Code File [ ]; Verify Security Fuse [✓]; READ / COPY [✓]

Selected Device Information
RAM - 8192 bytes; FLASH - 116 kB;

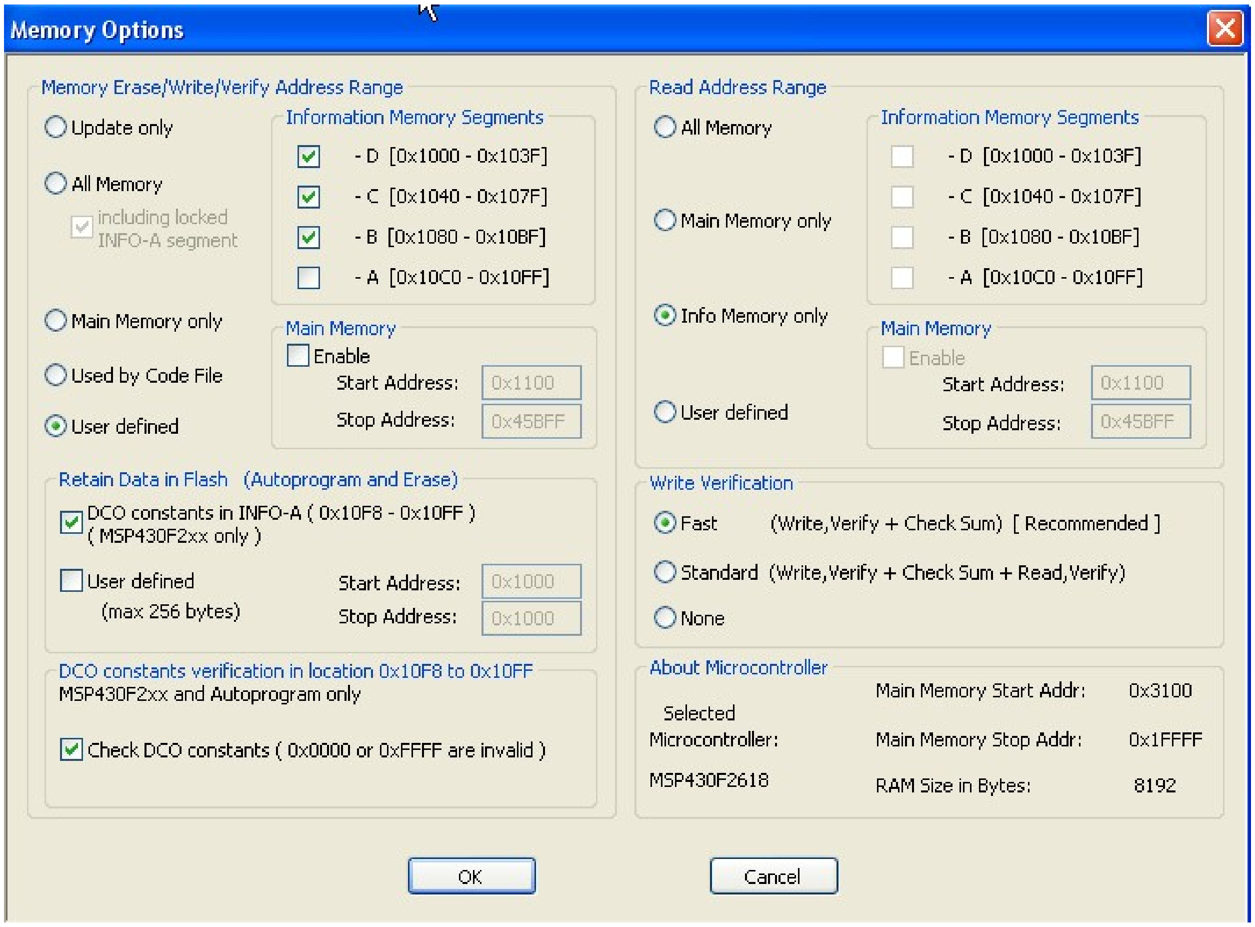
Device Serialization
Next Model-Group-Revision:
Next SN: 00000000
Format: yyyy1234

Report
```
 -------- D O N E --- ( run time =   0.1 sec.)
======================================
JTAG communication initialization....     OK
Verifying Security Fuse ...................     OK
Flash programming ...................     done
 -------- D O N E --- ( run time =   0.2 sec.)
======================================
JTAG communication initialization....     OK
Verifying Security Fuse ...................     OK
Reading Flash memory ...................     done
 -------- D O N E --- ( run time =  13.5 sec.)
```

Port: USB Automatic
JTAG (4-wires)

Erase / Write memory option:
* User Defined *

READ / COPY



Memory Erase/Write/Verify Address Range

- ○ Update only
- ○ All Memory
  - ✓ including locked INFO-A segment
- ○ Main Memory only
- ○ Used by Code File
- ● User defined

Information Memory Segments

- ✓ - D [0x1000 - 0x103F]
- ✓ - C [0x1040 - 0x107F]
- ✓ - B [0x1080 - 0x10BF]
- ☐ - A [0x10C0 - 0x10FF]

Main Memory

- ☐ Enable
- Start Address: 0x1100
- Stop Address: 0x45BFF

Retain Data in Flash (Autoprogram and Erase)

- ✓ DCO constants in INFO-A (0x10F8 - 0x10FF) (MSP430F2xx only)
- ☐ User defined (max 256 bytes) Start Address: 0x1000 Stop Address: 0x1000

DCO constants verification in location 0x10F8 to 0x10FF
MSP430F2xx and Autoprogram only

- ✓ Check DCO constants (0x0000 or 0xFFFF are invalid)

Read Address Range

- ○ All Memory
- ○ Main Memory only
- ● Info Memory only
- ○ User defined

Information Memory Segments

- ☐ - D [0x1000 - 0x103F]
- ☐ - C [0x1040 - 0x107F]
- ☐ - B [0x1080 - 0x10BF]
- ☐ - A [0x10C0 - 0x10FF]

Main Memory

- ☐ Enable
- Start Address: 0x1100
- Stop Address: 0x45BFF

Write Verification

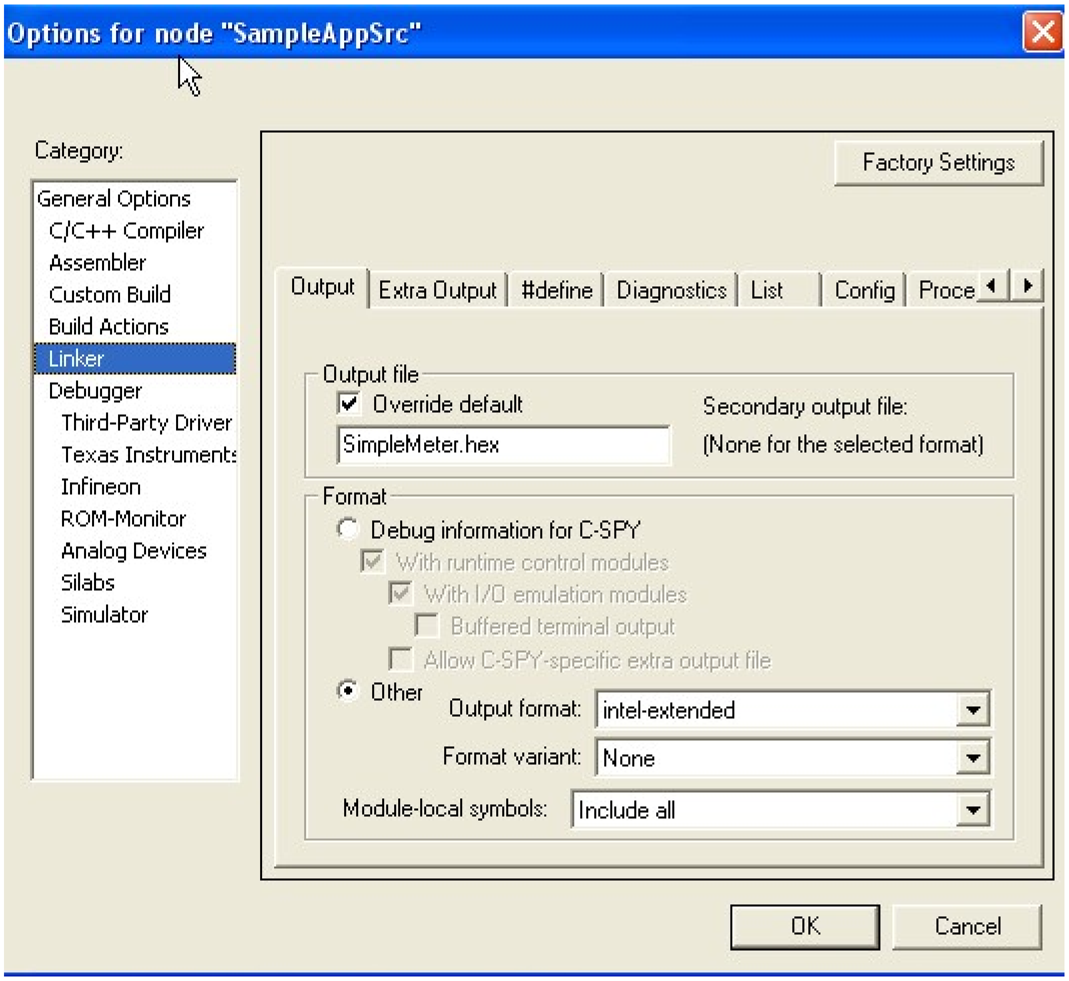
- ● Fast (Write,Verify + Check Sum) [Recommended]
- ○ Standard (Write,Verify + Check Sum + Read,Verify)
- ○ None

About Microcontroller

Selected Microcontroller:	Main Memory Start Addr:	0x3100
MSP430F2618	Main Memory Stop Addr:	0x1FFFF
	RAM Size in Bytes:	8192

For the CC2530, the console version of the SmartRF Flash Programmer has been updated (since ver 1.8) to provide a command line interface that is able to program the lock bits page with the certificate data. The console program accepts a .hex file along with the certificate data in .txt format.

The .hex image can be created using the IAR linker output tab and by un-commenting the "-M" lines in the f8w2530.xcl file as shown in the following screenshots.


Options for node "SampleAppSrc"

Category:
- General Options
- C/C++ Compiler
- Assembler
- Custom Build
- Build Actions
- Linker
- Debugger
- Third-Party Driver
- Texas Instruments
- Infineon
- ROM-Monitor
- Analog Devices
- Silabs
- Simulator

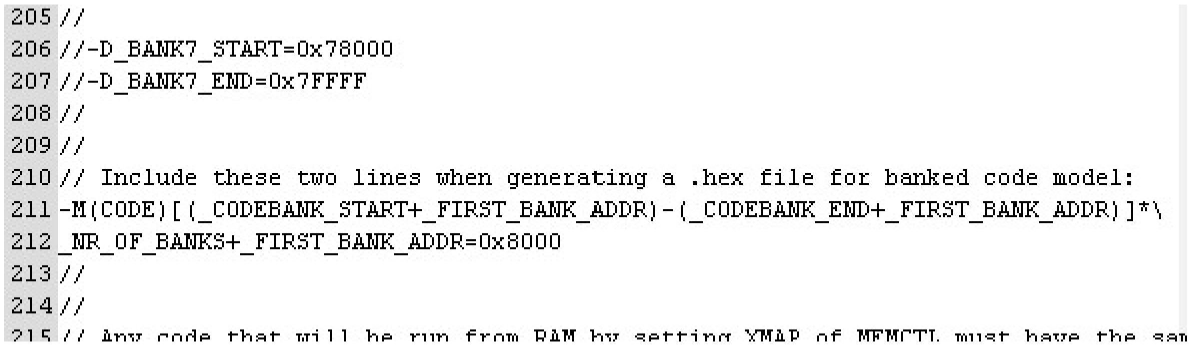
Factory Settings

Output | Extra Output | #define | Diagnostics | List | Config | Proce

Output file
- [✓] Override default: SimpleMeter.hex
- Secondary output file: (None for the selected format)

Format
- ○ Debug information for C-SPY
  - [✓] With runtime control modules
  - [✓] With I/O emulation modules
  - [ ] Buffered terminal output
  - [ ] Allow C-SPY-specific extra output file
- ● Other
  - Output format: intel-extended
  - Format variant: None
- Module-local symbols: Include all

OK | Cancel


```
//
//-D_BANK7_START=0x78000
//-D_BANK7_END=0x7FFFF
//
//
// Include these two lines when generating a .hex file for banked code model:
-M(CODE)[(_CODEBANK_START+_FIRST_BANK_ADDR)-(_CODEBANK_END+_FIRST_BANK_ADDR)]*\
_NR_OF_BANKS+_FIRST_BANK_ADDR=0x8000
//
//
// Any code that will be run from RAM by setting XMAP of MEMCTL must have the sam
```

At the command line prompt, the following syntax should be used:

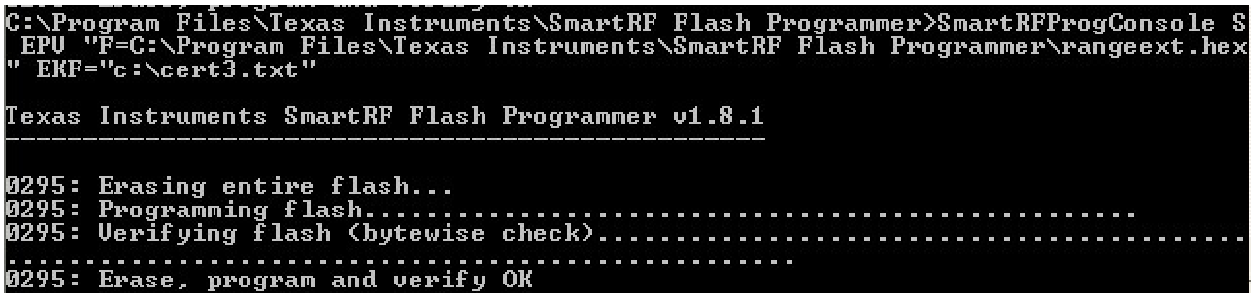
SmartRFProgConsole S EPV “F= <path to hex file>” EKF=”<path to certificate file.txt>”

The format of the certificate should be as follows:

IEEE Address: 00124b0000000003
CA Pub Key: 0200fde8a7f3d1084224962a4e7c54e69ac3f04da6b8
Device Implicit Cert:
0204ac2c2656f1eea4ff5dac4edda176bfe4fa70d95600124b0000000003544553545345434101090
001000001091003
Device Private Key: 034bc37a7210b7407a51dc11e5aebaf2e1503f6955
Device Public Key: 020718021c2ce9c58d2ad8352ab9ff452ff1c3bdadb3

Note: *The data for the device implicit cert should be on one line (i.e. no carriage return separations should be included).*

The .hex image would then be programmed into the CC2530 using the SmartRF Flash Programmer tool at the same time the lock bits page is programmed with the certificate data in the .txt file. The screenshot below shows an example.



```
C:\Program Files\Texas Instruments\SmartRF Flash Programmer>SmartRFProgConsole S
 EPV "F=C:\Program Files\Texas Instruments\SmartRF Flash Programmer\rangeext.hex
" EKF="c:\cert3.txt"

Texas Instruments SmartRF Flash Programmer v1.8.1
-------------------------------------------------

0295: Erasing entire flash...
0295: Programming flash...............................................
0295: Verifying flash (bytewise check)................................
..................................................
0295: Erase, program and verify OK
```

There are also several functions in the console version that can be used to create a very flexible script. First, note the 'x' option, that will list all available EBs. Example:

```
C:\Program Files\Texas Instruments\SmartRF Flash Programmer>SmartRFProgConsole.exe x

Texas Instruments SmartRF Flash Programmer
-----------------------------------------
Device:SmartRF05EB    ID:2769  (fwId:0500, fwRev:0013)  Chip:CC2530
Device:SmartRF05EB    ID:7335  (fwId:0500, fwRev:0013)  Chip:CC2530
```

One can then write a batch script (or similar) with the following lines:

```
SmartRFProgConsole.exe S(2769) EPV F="hex_image.hex" EKF="certificate_data_1.txt"
SmartRFProgConsole.exe S(7335) EPV F="hex_image.hex" EKF="certificate_data_2.txt"
```

This example assumes that the path of the .hex image and the certificate data .txt file are in the same directory as the SmartRFProgConsole.exe.

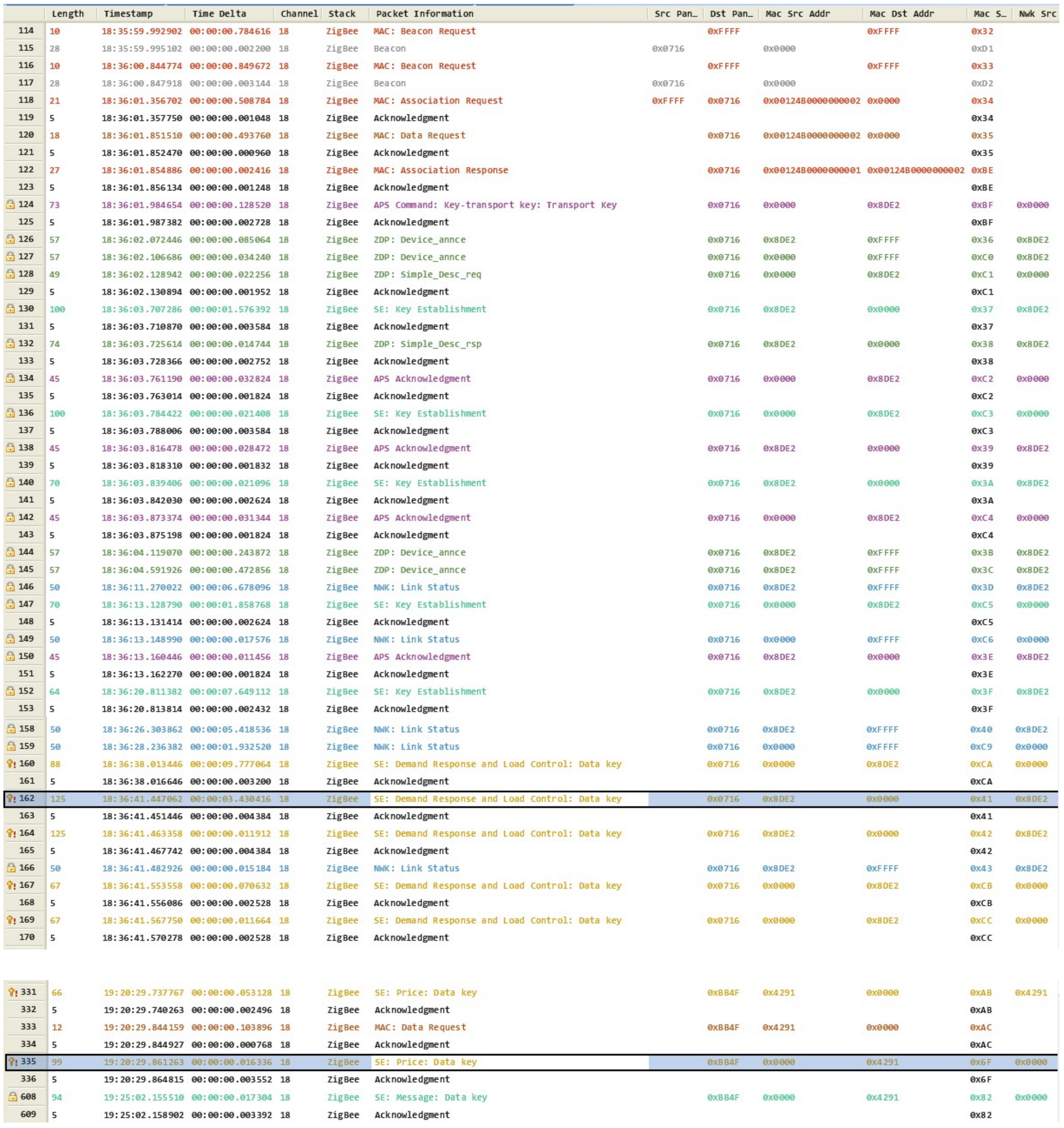
3.4. Running the ESP, PCT, and IPD Applications

After compiling these sample app instances and programming the devices one can start these applications as follows:

1. Power the coordinator, wait for "Zigbee Coord" to show up on the LCD display. The 2nd line of the LCD will display the PAN ID.
2. Then power the PCT device. The LED will flash, waiting for you to press SW1 which is joystick UP. This will cause the device to join the network. Wait about 20 seconds for the key establishment procedure to complete. Verify with the sniffer that the end device received the confirm key response. Then, press SW2 (joystick to the right) to send a PCT event to the PCT device. The PCT device will flash its LED and display "PCT Evt Started". When it completes the PCT event (which is for 1 min), then it will stop flashing its LED and display "PCT Evt Complete".
3. Power up the IPD end device. Hit SW1 to let it join the network. Wait about 20 seconds for the key establishment procedure to complete. Verify with the sniffer that the end device received the confirm key response. Then, every 5 seconds it will get the pricing info and display the provider id. On the coord, hit SW3 (joystick down) to send a display message to the IPD. "Rcvd MESSAGE Cmd" should be displayed. The Get Current Price command can also be sent from the IPD via INTER-PAN by press SW2 on the IPD.

Note: NV_RESTORE is **not** turned on by default, so if you power cycle any of the devices, it needs to go through the key establishment procedure again, and a different link key would be used. NV_RESTORE is turned off by default to allow rapid development. Also, any device joining the ESP wouldn't be able to join more than once since the new link key is used for authenticating the device instead of the default TC link key. Power cycle the ESP and then the joining device so that it is able to join using the default TC link key.

The following shows what happens in the Ubiqua Protocol Analyzer for these application instances.



	Length	Timestamp	Time Delta	Channel	Stack	Packet Information	Src Pan…	Dst Pan…	Mac Src Addr	Mac Dst Addr	Mac S…	Nwk Src
114	10	18:35:59.992902	00:00:00.784616	18	ZigBee	MAC: Beacon Request		0xFFFF		0xFFFF	0x32	
115	28	18:35:59.995102	00:00:00.002200	18	ZigBee	Beacon	0x0716		0x0000		0xD1	
116	10	18:36:00.844774	00:00:00.849672	18	ZigBee	MAC: Beacon Request		0xFFFF		0xFFFF	0x33	
117	28	18:36:00.847918	00:00:00.003144	18	ZigBee	Beacon	0x0716		0x0000		0xD2	
118	21	18:36:01.356702	00:00:00.508784	18	ZigBee	MAC: Association Request	0xFFFF	0x0716	0x00124B0000000002	0x0000	0x34	
119	5	18:36:01.357750	00:00:00.001048	18	ZigBee	Acknowledgment					0x34	
120	18	18:36:01.851510	00:00:00.493760	18	ZigBee	MAC: Data Request		0x0716	0x00124B0000000002	0x0000	0x35	
121	5	18:36:01.852470	00:00:00.000960	18	ZigBee	Acknowledgment					0x35	
122	27	18:36:01.854886	00:00:00.002416	18	ZigBee	MAC: Association Response		0x0716	0x00124B0000000001	0x00124B0000000002	0xBE	
123	5	18:36:01.856134	00:00:00.001248	18	ZigBee	Acknowledgment					0xBE	
🔒 124	73	18:36:01.984654	00:00:00.128520	18	ZigBee	APS Command: Key-transport key: Transport Key		0x0716	0x0000	0x8DE2	0xBF	0x0000
125	5	18:36:01.987382	00:00:00.002728	18	ZigBee	Acknowledgment					0xBF	
🔒 126	57	18:36:02.072446	00:00:00.085064	18	ZigBee	ZDP: Device_annce		0x0716	0x8DE2	0xFFFF	0x36	0x8DE2
🔒 127	57	18:36:02.106686	00:00:00.034240	18	ZigBee	ZDP: Device_annce		0x0716	0x0000	0xFFFF	0xC0	0x8DE2
🔒 128	49	18:36:02.128942	00:00:00.022256	18	ZigBee	ZDP: Simple_Desc_req		0x0716	0x0000	0x8DE2	0xC1	0x0000
129	5	18:36:02.130894	00:00:00.001952	18	ZigBee	Acknowledgment					0xC1	
🔒 130	100	18:36:03.707286	00:00:01.576392	18	ZigBee	SE: Key Establishment		0x0716	0x8DE2	0x0000	0x37	0x8DE2
131	5	18:36:03.710870	00:00:00.003584	18	ZigBee	Acknowledgment					0x37	
🔒 132	74	18:36:03.725614	00:00:00.014744	18	ZigBee	ZDP: Simple_Desc_rsp		0x0716	0x8DE2	0x0000	0x38	0x8DE2
133	5	18:36:03.728366	00:00:00.002752	18	ZigBee	Acknowledgment					0x38	
🔒 134	45	18:36:03.761190	00:00:00.032824	18	ZigBee	APS Acknowledgment		0x0716	0x0000	0x8DE2	0xC2	0x0000
135	5	18:36:03.763014	00:00:00.001824	18	ZigBee	Acknowledgment					0xC2	
🔒 136	100	18:36:03.784422	00:00:00.021408	18	ZigBee	SE: Key Establishment		0x0716	0x0000	0x8DE2	0xC3	0x0000
137	5	18:36:03.788006	00:00:00.003584	18	ZigBee	Acknowledgment					0xC3	
🔒 138	45	18:36:03.816478	00:00:00.028472	18	ZigBee	APS Acknowledgment		0x0716	0x8DE2	0x0000	0x39	0x8DE2
139	5	18:36:03.818310	00:00:00.001832	18	ZigBee	Acknowledgment					0x39	
🔒 140	70	18:36:03.839406	00:00:00.021096	18	ZigBee	SE: Key Establishment		0x0716	0x8DE2	0x0000	0x3A	0x8DE2
141	5	18:36:03.842030	00:00:00.002624	18	ZigBee	Acknowledgment					0x3A	
🔒 142	45	18:36:03.873374	00:00:00.031344	18	ZigBee	APS Acknowledgment		0x0716	0x0000	0x8DE2	0xC4	0x0000
143	5	18:36:03.875198	00:00:00.001824	18	ZigBee	Acknowledgment					0xC4	
🔒 144	57	18:36:04.119070	00:00:00.243872	18	ZigBee	ZDP: Device_annce		0x0716	0x8DE2	0xFFFF	0x3B	0x8DE2
🔒 145	57	18:36:04.591926	00:00:00.472856	18	ZigBee	ZDP: Device_annce		0x0716	0x8DE2	0xFFFF	0x3C	0x8DE2
🔒 146	50	18:36:11.270022	00:00:06.678096	18	ZigBee	NWK: Link Status		0x0716	0x8DE2	0xFFFF	0x3D	0x8DE2
🔒 147	70	18:36:13.128790	00:00:01.858768	18	ZigBee	SE: Key Establishment		0x0716	0x0000	0x8DE2	0xC5	0x0000
148	5	18:36:13.131414	00:00:00.002624	18	ZigBee	Acknowledgment					0xC5	
🔒 149	50	18:36:13.148990	00:00:00.017576	18	ZigBee	NWK: Link Status		0x0716	0x0000	0xFFFF	0xC6	0x0000
🔒 150	45	18:36:13.160446	00:00:00.011456	18	ZigBee	APS Acknowledgment		0x0716	0x8DE2	0x0000	0x3E	0x8DE2
151	5	18:36:13.162270	00:00:00.001824	18	ZigBee	Acknowledgment					0x3E	
🔒 152	64	18:36:20.811382	00:00:07.649112	18	ZigBee	SE: Key Establishment		0x0716	0x8DE2	0x0000	0x3F	0x8DE2
153	5	18:36:20.813814	00:00:00.002432	18	ZigBee	Acknowledgment					0x3F	
🔒 158	50	18:36:26.303862	00:00:05.418536	18	ZigBee	NWK: Link Status		0x0716	0x8DE2	0xFFFF	0x40	0x8DE2
🔒 159	50	18:36:28.236382	00:00:01.932520	18	ZigBee	NWK: Link Status		0x0716	0x0000	0xFFFF	0xC9	0x0000
🔑 160	88	18:36:38.013446	00:00:09.777064	18	ZigBee	SE: Demand Response and Load Control: Data key		0x0716	0x0000	0x8DE2	0xCA	0x0000
161	5	18:36:38.016646	00:00:00.003200	18	ZigBee	Acknowledgment					0xCA	
🔑 162	125	18:36:41.447062	00:00:03.430416	18	ZigBee	SE: Demand Response and Load Control: Data key		0x0716	0x8DE2	0x0000	0x41	0x8DE2
163	5	18:36:41.451446	00:00:00.004384	18	ZigBee	Acknowledgment					0x41	
🔑 164	125	18:36:41.463358	00:00:00.011912	18	ZigBee	SE: Demand Response and Load Control: Data key		0x0716	0x8DE2	0x0000	0x42	0x8DE2
165	5	18:36:41.467742	00:00:00.004384	18	ZigBee	Acknowledgment					0x42	
🔒 166	50	18:36:41.482926	00:00:00.015184	18	ZigBee	NWK: Link Status		0x0716	0x8DE2	0xFFFF	0x43	0x8DE2
🔑 167	67	18:36:41.553558	00:00:00.070632	18	ZigBee	SE: Demand Response and Load Control: Data key		0x0716	0x0000	0x8DE2	0xCB	0x0000
168	5	18:36:41.556086	00:00:00.002528	18	ZigBee	Acknowledgment					0xCB	
🔑 169	67	18:36:41.567750	00:00:00.011664	18	ZigBee	SE: Demand Response and Load Control: Data key		0x0716	0x0000	0x8DE2	0xCC	0x0000
170	5	18:36:41.570278	00:00:00.002528	18	ZigBee	Acknowledgment					0xCC	
🔑 331	66	19:20:29.737767	00:00:00.053128	18	ZigBee	SE: Price: Data key		0xBB4F	0x4291	0x0000	0xAB	0x4291
332	5	19:20:29.740263	00:00:00.002496	18	ZigBee	Acknowledgment					0xAB	
333	12	19:20:29.844159	00:00:00.103896	18	ZigBee	MAC: Data Request		0xBB4F	0x4291	0x0000	0xAC	
334	5	19:20:29.844927	00:00:00.000768	18	ZigBee	Acknowledgment					0xAC	
🔑 335	99	19:20:29.861263	00:00:00.016336	18	ZigBee	SE: Price: Data key		0xBB4F	0x0000	0x4291	0x6F	0x0000
336	5	19:20:29.864815	00:00:00.003552	18	ZigBee	Acknowledgment					0x6F	
🔒 608	94	19:25:02.155510	00:00:00.017304	18	ZigBee	SE: Message: Data key		0xBB4F	0x0000	0x4291	0x82	0x0000
609	5	19:25:02.158902	00:00:00.003392	18	ZigBee	Acknowledgment					0x82	

4. Theory of Operation

This section explains how each application instance behaves with respect to the ESP device and also discusses the design of each application instance.

4.1. SE Secure Joining

The default trust center link key (TC link key) is used to commission each device. Each device uses the SE secure joining process. See the Z-stack Smart Energy Developer's Guide for more details.

4.2. Key Establishment

Upon successfully joining the network using security, each device will initiate key establishment with the ESP. Certificate information required to perform the key establishment is entered in ahead of time using Z-Tool. Figure 3 shows a flowchart of the device startup logic. The ZDO_STATE_CHANGE event is an OSAL system message that is provided to the application to indicate that the device has successfully started or joined the network.

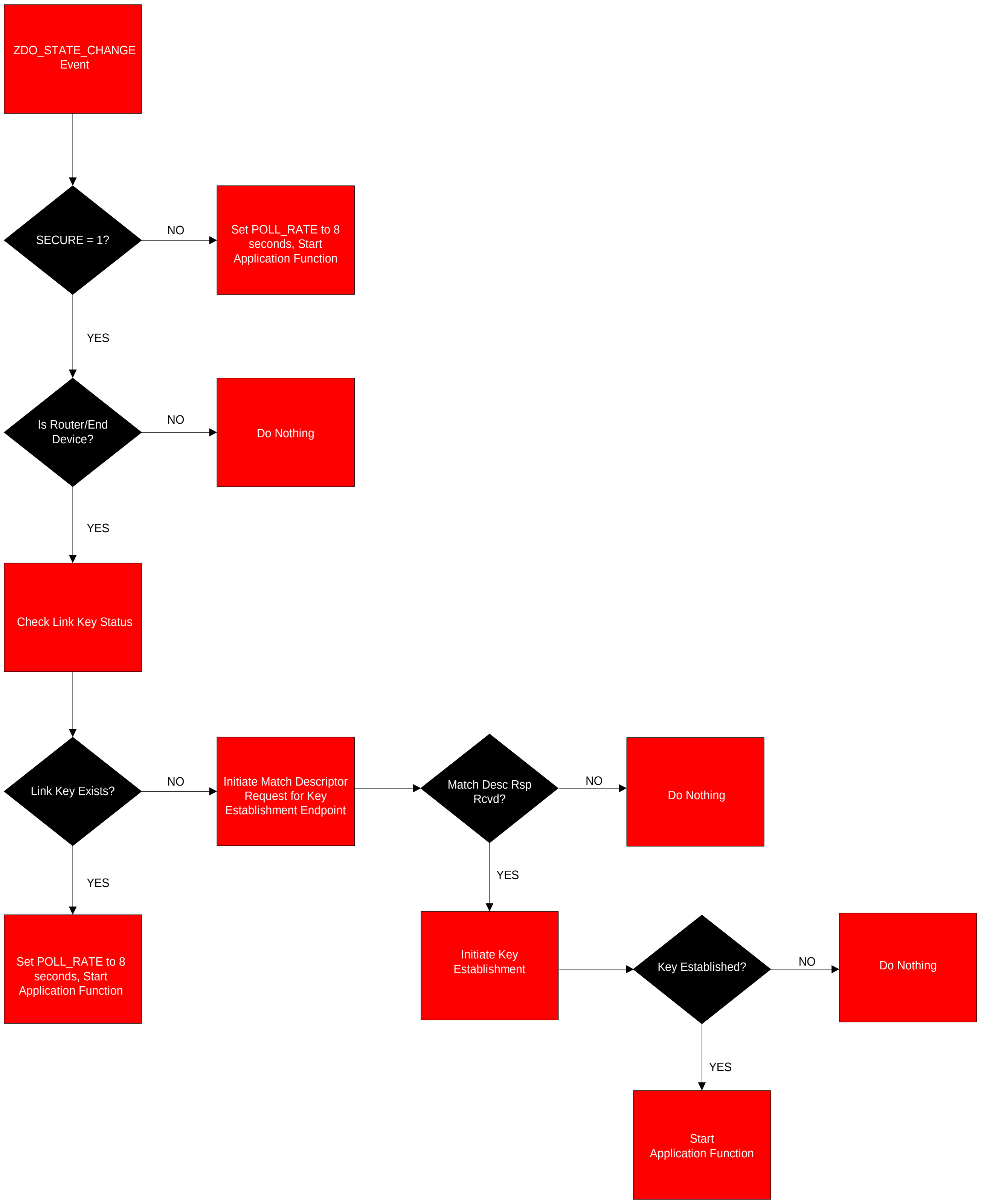

ZDO_STATE_CHANGE Event
SECURE = 1?
NO
Set POLL_RATE to 8 seconds, Start Application Function
YES
Is Router/End Device?
NO
Do Nothing
YES
Check Link Key Status
Link Key Exists?
NO
Initiate Match Descriptor Request for Key Establishment Endpoint
Match Desc Rsp Rcvd?
NO
Do Nothing
YES
Set POLL_RATE to 8 seconds, Start Application Function
YES
Initiate Key Establishment
Key Established?
NO
Do Nothing
YES
Start Application Function

Figure 3. Flowchart of Device Startup Logic

4.3. Device and Service Discovery

Each device communicates with the ESP. Since the ESP is the coordinator and will always have a short address of 0x0000, devices that join the network can assume this and communicate with the ESP directly. Therefore, no device discovery or service discovery is required for joining devices.
Once the device joins the network and performs a successful key establishment, it starts its communications with the ESP based on the application behavior.

The following section defines the application behavior for each device after it has successfully joined the network using SE secure join, and has an application link key established with the ESP.

4.4. ESP

The ESP is assumed to be the coordinator, trust center, and network manager of the network. Every other device communicates with the ESP.

The ESP application instance consists of the following modules:

OSAL_esp.c - functions and tables for task initialization
esp.c – main application function that has init and event loop function
esp.h – header file for application module
esp_data.c – container for declaration of attributes, clusters, simple descriptor

The ESP application makes function calls to the following ZCL SE functions and ZDO API functions:

esp_GetCurrentPriceCB – callback executed when get current price message is received.

zclSE_Pricing_Send_PublishPrice – send publish price command to IPD.

esp_ProcessInReportCmd - print out value of current summation delivered when attribute report is received.

zclSE_LoadControl_Send_LoadControlEvent – send a load control event message to a load control device.

zclSE_Message_Send_DisplayMessage – send a message command to an IPD

esp_ReqFastPollModeCmdCB – callback executed when request fast poll mode command is received.

zclSE_SimpleMetering_Send_ReqFastPollModeRsp – send request fast poll mode response command to IPD.

esp_ChangeSupplyCmdCB – callback executed when change supply message is received.

zclSE_Prepayment_Send_SupplyStatusResponse – send supply status response message to IPD.

Prepayment cluster functionality is disabled by default, the ZCL_PREPAYMENT compiler flag must be enabled in f8wZCL.cfg.

esp_ProcessAppMsg – MT_SYS_APP_MSG can be used to tunnel any application specific messages into the device via UART (e.g. by using Z-Tool). It is left as a hook so the application developer can fill in the necessary functionality.

esp_ProcessZDOMsg – responses for end device annce and simple descriptor requests are processed here. When an end device annce message is received, the SIMPLE_DESC_QUERY_EVT event is set, and this causes the simple descriptor request to be triggered for this source address.

esp_HandleKeys – user switch events are processed here. SW1 sends out a load control event to the PCT, SW2 sends out a load control event to the load control device, SW3 sends out a display message command to the IPD.
The ESP interacts with other application instances in different ways. At first, the ESP is configured with Certicom key information using MT OSAL_NV_READ and OSAL_NV_WRITE commands. A Z-Tool script is used to configure each device. Endpoints, command structures, ZDO callbacks, and SE callbacks are initialized by calling esp_Init. System and user events are handled in esp_event_loop.

When a Simple Metering device joins the network, once the key establishment procedure is successful (the ESP application doesn't have to do anything here since key establishment is handled via the zcl_key_establishment.c module), it will receive attribute reports from the Simple Metering device every 5 seconds. Therefore, every 5 seconds the function esp_ProcessInReportCmd is called to display the received CurrentSummationDelivered attribute on the ESP's LCD screen.

When an IPD device joins the network, once the key establishment procedure is successful (the ESP application doesn't have to do anything here since key establishment is handled via the zcl_key_establishment.c module), it will send a get pricing info message. On the ESP, the esp_GetCurrentPriceCB is called when the get pricing info message is received. This then calls zclSE_Pricing_Send_PublishPrice() to send out the publish price command to the IPD. If SW3 is pressed, the function zclSE_Message_Send_DisplayMessage() is called to send a display MESSAGE to the IPD device.

When a PCT or Load Control device joins the network and establishes a link key with the ESP, the ESP will discover these devices by using the end device annce to trigger a simple descriptor ZDO request. Each simple descriptor response is parsed in the ZDO callback handler *esp_ProcessZDOMsg*, and the device ID field is checked to determine whether this response came from the PCT or Load Control Device. Depending on the device ID, the destination address for the PCT or Load Control Device is then populated. This is how the ESP then knows which type of load control event to send to which device. SW1 is used to send a PCT event to the PCT, and SW2 is used to send an event to the load control device.

4.5. Simple Metering Device

The Simple Metering Device will periodically send reports for the CurrentSummationDelivered Simple Metering attribute to the ESP. The ESP will display the CurrentSummationDelivered value on the LCD. See Figure 4 for a sequence diagram.

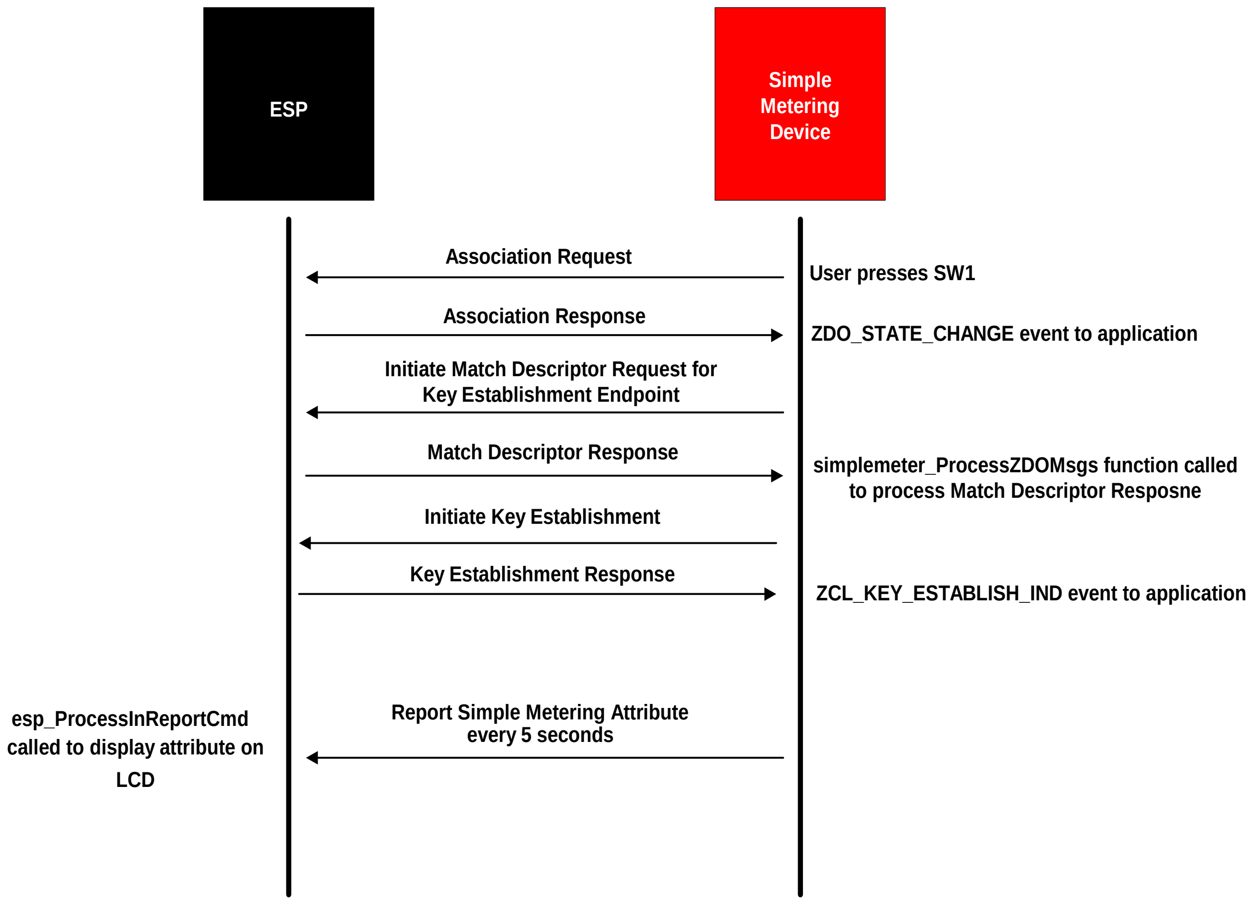

ESP
Simple Metering Device
Association Request
User presses SW1
Association Response
ZDO_STATE_CHANGE event to application
Initiate Match Descriptor Request for Key Establishment Endpoint
Match Descriptor Response
simplemeter_ProcessZDOMsgs function called to process Match Descriptor Resposne
Initiate Key Establishment
Key Establishment Response
ZCL_KEY_ESTABLISH_IND event to application
esp_ProcessInReportCmd called to display attribute on LCD
Report Simple Metering Attribute every 5 seconds

Figure 4. Sequence Diagram for a Simple Meter Device

The Simple Meter application instance consists of the following modules:

OSAL_simplemeter.c - functions and tables for task initialization
simplemeter.c – main application function that has init and event loop function
simplemeter.h – header file for application module
simplemeter_data.c – container for declaration of attributes, clusters, simple descriptor

The Simple Meter application makes the following function calls:

zcl_SendReportCmd – this function sends the CurrentSummationDelivered report attribute to the ESP.

simplemeter_HandleKeys – user switch events are processed here. HOLD_AUTO_START is defined so the device will normally not join the network upon startup. The user has to press SW1, which then calls ZDOInitDevice() to start the device. The rationale for doing this is so that the user can have time to configure the Certicom keys prior to joining the network.

Once the simple meter device joins the network, it goes through the state machine explained in Figure 3. simplemeter_KeyEstablish_ReturnLinkKey() is called to check if a link key has already been established with the ESP. If one hasn't been established, it will send out a match descriptor request in search of the endpoint on the ESP that has the key establishment cluster. Upon processing the match descriptor response in the simplemeter_ProcessZDOMsgs() function, the osal event SIMPLEMETER_KEY_ESTABLISHMENT_REQUEST_EVT is set. The event handler for this in the process event loop will then call zclGeneral_KeyEstablish_InitiateKeyEstablishment() to do the CBKE procedure. Upon its success, the application will receive a ZCL_KEY_ESTABLISH_IND system message. An osal timer event called SIMPLEMETER_REPORT_ATTRIBUTE_EVT is then started to send attribute reports every 5 seconds of the CurrentSummationDelivered attribute. The structure for the report command is created and initialization of the attribute is done in the simplemeter_Init() function.

4.6. Load Control Device

The ESP will send a load control event to the Load Control Device via a user switch button press. In the Load Control Event payload, the Device Class Field Bitmap will indicate that Bit 7 is set (representing Residential ON/OFF Load). When the Load Control Device receives the load control event, it sends a ReportEventStatus command that it received it, and sends another one when it starts it. The Start Time field will indicate to "Start Now". The event duration will last 1 minute. While the load control event is in process, the Load Control Device will flash its LED. When the load control event is finished, another ReportEventStatus command will be sent to the ESP to indicate the completion of the load control event. See Figure 5 for a sequence diagram.

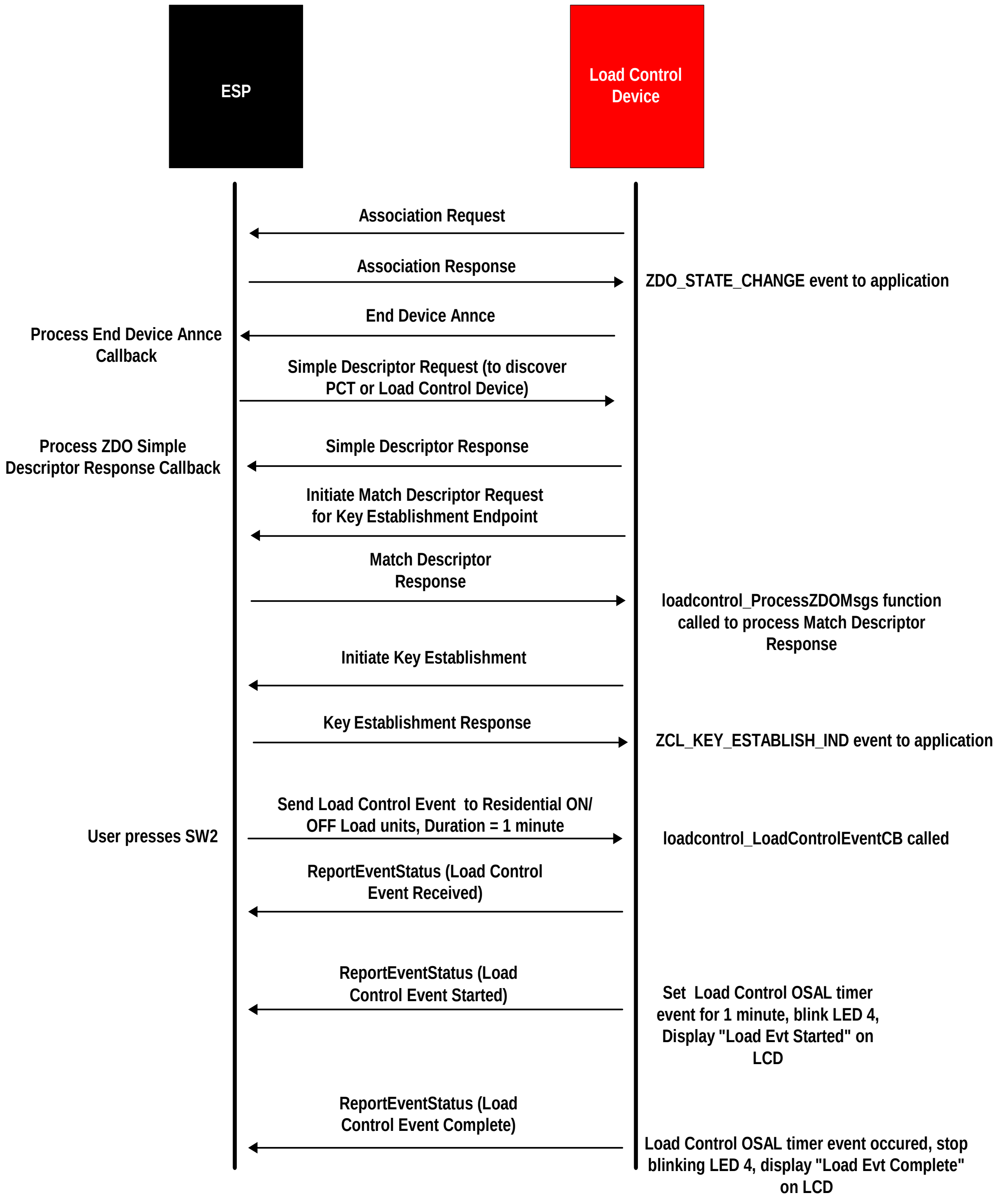

ESP
Load Control Device
Association Request
Association Response
ZDO_STATE_CHANGE event to application
End Device Annce
Process End Device Annce Callback
Simple Descriptor Request (to discover PCT or Load Control Device)
Process ZDO Simple Descriptor Response Callback
Simple Descriptor Response
Initiate Match Descriptor Request for Key Establishment Endpoint
Match Descriptor Response
loadcontrol_ProcessZDOMsgs function called to process Match Descriptor Response
Initiate Key Establishment
Key Establishment Response
ZCL_KEY_ESTABLISH_IND event to application
Send Load Control Event to Residential ON/ OFF Load units, Duration = 1 minute
User presses SW2
loadcontrol_LoadControlEventCB called
ReportEventStatus (Load Control Event Received)
ReportEventStatus (Load Control Event Started)
Set Load Control OSAL timer event for 1 minute, blink LED 4, Display "Load Evt Started" on LCD
ReportEventStatus (Load Control Event Complete)
Load Control OSAL timer event occured, stop blinking LED 4, display "Load Evt Complete" on LCD

Figure 5. Sequence Diagram for a Load Control Device

OSAL_loadcontrol.c - functions and tables for task initialization
loadcontrol.c – main application function that has init and event loop function
loadcontrol.h – header file for application module
loadcontrol_data.c – container for declaration of attributes, clusters, simple descriptor

The Load Control application makes function calls to the following ZCL SE functions and ZDO API functions:

loadcontrol_LoadControlEventCB – this function is called when the load control device received a load control command.

zclSE_LoadControl_Send_ReportEventStatus – When a load control command is received, the device sends a report event status command back to the ESP. Possible values for the event type for this particular sample app could be event received, event started, and event completed. The response structure (rsp) is made a global variable so that the process event loop can manipulate the eventStatus field once the load control event is complete.

loadcontrol_HandleKeys – user switch events are processed here. HOLD_AUTO_START is defined so the device will normally not join the network upon startup. The user has to press SW1, which then calls ZDOInitDevice() to start the device. The rationale for doing this is so that the user can have time to configure the Certicom keys prior to joining the network.

Once the Load Control device joins the network, it goes through the state machine explained in Figure 3. loadcontrol_KeyEstablish_ReturnLinkKey() is called to check if a link key has already been established with the ESP. If one hasn't been established, it will send out a match descriptor request in search of the endpoint on the ESP that has the key establishment cluster. Upon processing the match descriptor response in the loadcontrol_ProcessZDOMsgs() function, the osal event LOADCONTROL_KEY_ESTABLISHMENT_REQUEST_EVT is set. The event handler for this in the process event loop will then call zclGeneral_KeyEstablish_InitiateKeyEstablishment() to do the CBKE procedure. Upon its success, the application will receive a ZCL_KEY_ESTABLISH_IND system message. Nothing is done at this point, and the load control device is ready to accept load control messages from the ESP. When a load control command is received, the callback loadcontrol_LoadControlEventCB is executed. This callback function checks the issuer event id (0x12345678), the start time (0x00000000 = NOW) in order to make sure that it is not just blindly responding to a random load control event. Furthermore, it checks the deviceGroupClass to determine whether this load control event was for the PCT or load control device. The values within this load control command originate from the ESP. The correct string on the LCD is then displayed to indicate whether this is a load control device event or PCT event. It is assumed that the load control device belongs to the residential on/off load device class, and that the PCT belongs to the HVAC compressor/furnace device class.
An osal timer event called LOADCONTROL_LOAD_CTRL_EVT is then started to commence the load control event and flash the LED for the duration specified, which is 1 minute. When this timer event expires, the status response of event completed is sent back to the ESP, and the LED stops flashing. The user also sees a display on the LCD indicating that the load control event is complete.

4.7. PCT

The PCT will have a very similar behavior to the Load Control Device. However, in the Load Control Event payload, the Device Class Field BitMap will indicate that Bit 0 is set (HVAC compressor or furnace). The Load Control Event for the PCT will be regarded as a “PCT Event”, and the message “PCT Event Started” will be displayed if the device has LCD support. See Figure 6 for a sequence diagram.

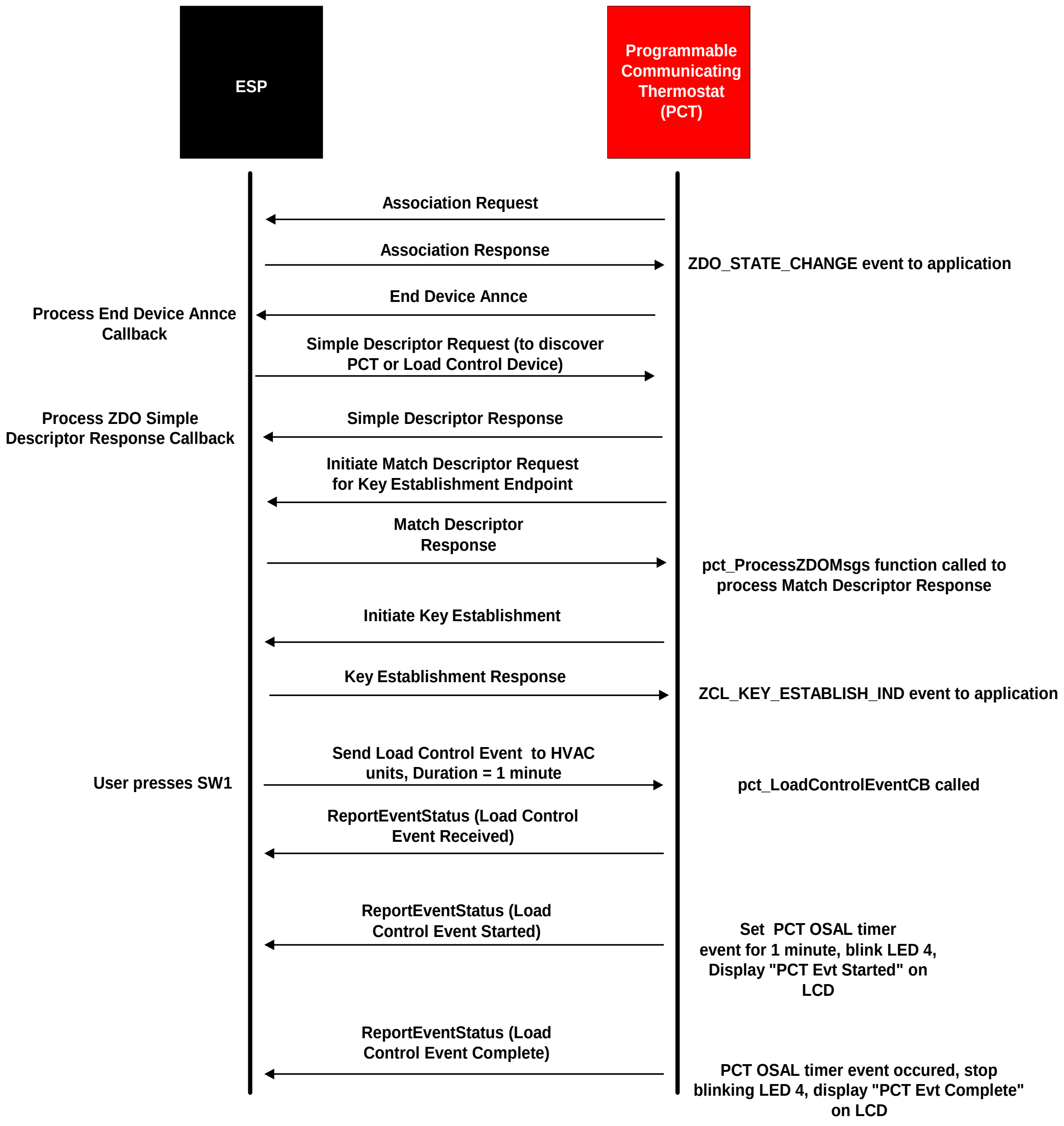

ESP
Programmable Communicating Thermostat (PCT)
Association Request
Association Response
ZDO_STATE_CHANGE event to application
End Device Annce
Process End Device Annce Callback
Simple Descriptor Request (to discover PCT or Load Control Device)
Process ZDO Simple Descriptor Response Callback
Simple Descriptor Response
Initiate Match Descriptor Request for Key Establishment Endpoint
Match Descriptor Response
pct_ProcessZDOMsgs function called to process Match Descriptor Response
Initiate Key Establishment
Key Establishment Response
ZCL_KEY_ESTABLISH_IND event to application
User presses SW1
Send Load Control Event to HVAC units, Duration = 1 minute
pct_LoadControlEventCB called
ReportEventStatus (Load Control Event Received)
ReportEventStatus (Load Control Event Started)
Set PCT OSAL timer event for 1 minute, blink LED 4, Display "PCT Evt Started" on LCD
ReportEventStatus (Load Control Event Complete)
PCT OSAL timer event occured, stop blinking LED 4, display "PCT Evt Complete" on LCD

Figure 6. Sequence Diagram for a PCT

OSAL_pct.c – functions and tables for task initialization
pct.c – main application function that has init and event loop function
pct.h – header file for application module
pct_data.c – container for declaration of attributes, clusters, simple descriptor

The PCT application makes function calls to the following ZCL SE functions and ZDO API functions:

pct_LoadControlEventCB – this function is called when the PCT device received a load control command.

zclSE_LoadControl_Send_ReportEventStatus – When a load control command is received, the device sends a report event status command back to the ESP. Possible values for the event type for this particular sample app could be event received, event started, and event completed. The response structure (rsp) is made a global variable so that the process event loop can manipulate the eventStatus field once the PCT event is complete.

pct_HandleKeys – user switch events are processed here. HOLD_AUTO_START is defined so the device will normally not join the network upon startup. The user has to press SW1, which then calls ZDOInitDevice() to start the device. The rationale for doing this is so that the user can have time to configure the Certicom keys prior to joining the network.

Once the PCT device joins the network, it goes through the state machine explained in Figure 3. pct_KeyEstablish_ReturnLinkKey() is called to check if a link key has already been established with the ESP. If one hasn't been established, it will send out a match descriptor request in search of the endpoint on the ESP that has the key establishment cluster. Upon processing the match descriptor response in the pct_ProcessZDOMsgs() function, the osal event PCT_KEY_ESTABLISHMENT_REQUEST_EVT is set. The event handler for this in the process event loop will then call zclGeneral_KeyEstablish_InitiateKeyEstablishment() to do the CBKE procedure. Upon its success, the application will receive a ZCL_KEY_ESTABLISH_IND system message. Nothing is done at this point, and the PCT device is ready to accept load control messages from the ESP. When a load control command is received, the callback pct_LoadControlEventCB is executed. This callback function checks the issuer event id (0x12345678), the start time (0x00000000 = NOW) in order to make sure that it is not just blindly responding to a random load control event. Furthermore, it checks the deviceGroupClass to determine whether this load control event was for the PCT or load control device. The values within this load control command originate from the ESP. The correct string on the LCD is then displayed to indicate whether this is a load control device event or PCT event. It is assumed that the load control device belongs to the residential on/off load device class, and that the PCT belongs to the HVAC compressor/furnace device class. An osal timer event called PCT_LOAD_CTRL_EVT is then started to commence the load control event and flash the LED for the duration specified, which is 1 minute. When this timer event expires, the status response of event completed is sent back to the ESP, and the LED stops flashing. The user also sees a display on the LCD indicating that the PCT event is complete.

Note: The mechanics of the PCT and Load Control Device are the same. There are just slight differences in variable names and the type of message displayed on the LCD.

4.8. In Premise Display

OSAL_ipd.c - functions and tables for task initialization
ipd.c – main application function that has init and event loop function
ipd.h – header file for application module
ipd_data.c – container for declaration of attributes, clusters, simple descriptor

If IPD is built with ZCL_PREPAYMENT enabled, the pricing and fast poll test are disabled.

There is a mutually exclusive compiler flag, FAST_POLL_TEST, used to set the device Fast Polling test mode. This flag is not enabled in the project by default, so Pricing Test is enabled.

4.8.1. Pricing Test

The In Premise Display will use the Get Current Price command to obtain the current pricing info from the ESP. The user can press SW2 to send an INTER-PAN version of the Get Current Price command to the ESP. The pricing information will be displayed on its LCD. The ESP will be able to use the SW3 button press to send a display MESSAGE command to the In Premise Display. This message will then be displayed on the LCD. See Figure 7 for a sequence diagram.

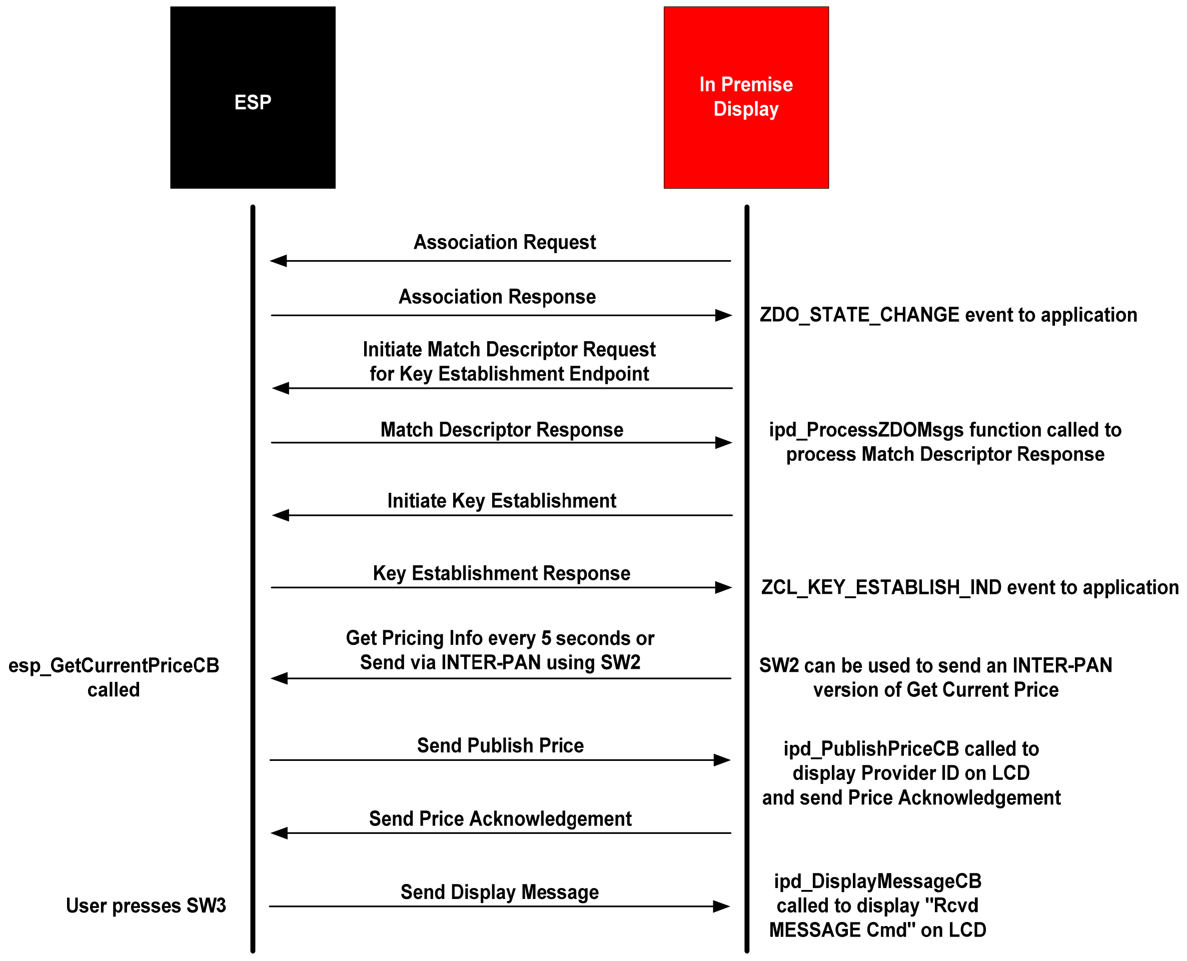

ESP
In Premise Display
Association Request
Association Response
ZDO_STATE_CHANGE event to application
Initiate Match Descriptor Request for Key Establishment Endpoint
Match Descriptor Response
ipd_ProcessZDOMsgs function called to process Match Descriptor Response
Initiate Key Establishment
Key Establishment Response
ZCL_KEY_ESTABLISH_IND event to application
Get Pricing Info every 5 seconds or Send via INTER-PAN using SW2
esp_GetCurrentPriceCB called
SW2 can be used to send an INTER-PAN version of Get Current Price
Send Publish Price
ipd_PublishPriceCB called to display Provider ID on LCD and send Price Acknowledgement
Send Price Acknowledgement
User presses SW3
Send Display Message
ipd_DisplayMessageCB called to display "Rcvd MESSAGE Cmd" on LCD

Figure 7. Sequence Diagram for an In Premise Display Pricing Test

The IPD application makes function calls to the following ZCL SE functions and ZDO API functions:

zclSE_Pricing_Send_GetCurrentPrice – this function sends the get current price request to the ESP.

ipd_PublishPriceCB – this function is called when the IPD device receives the publish price command from the ESP, and displays the provider ID on the LCD.

zclSE_Pricing_Send_PriceAcknowledgement – this function is called to send the price acknowledgement command to the ESP in response to the publish price command.

ipd_HandleKeys – user switch events are processed here. HOLD_AUTO_START is defined so the device will normally not join the network upon startup. The user has to press SW1, which then calls ZDOInitDevice() to start the device. The rationale for doing this is so that the user can have time to configure the Certicom keys prior to joining the network.

Once the IPD device joins the network, it goes through the state machine explained in Figure 3. ipd_KeyEstablish_ReturnLinkKey() is called to check if a link key has already been established with the ESP. If one hasn't been established, it will send out a match descriptor request in search of the endpoint on the ESP that has the key establishment cluster. Upon processing the match descriptor response in the ipd_ProcessZDOMsgs() function, the osal event IPD_KEY_ESTABLISHMENT_REQUEST_EVT is set. The event handler for this in the process event loop will then call zclGeneral_KeyEstablish_InitiateKeyEstablishment() to do the CBKE procedure. Upon its success, the application will receive a ZCL_KEY_ESTABLISH_IND system message. An osal timer event called IPD_GET_PRICING_INFO_EVT is then started to send the get pricing info request every 5 seconds. The result of the pricing request is received via ipd_PublishPriceCB and is displayed on the LCD. Only the provider ID field of the published price payload is displayed. . If the price control options on the publish price command require it, a price acknowledgment command is sent to the ESP.

4.8.2. Fast Polling Test

The user can press SW3 in the In Premise Display to send the Request Fast Poll Mode command to the ESP to initiate a fast polling period. Once the IPD receives Request Fast Poll Mode Response the fast polling information will be displayed on its LCD. The ESP will display the current time and the end time of the fast poll period (in HEX format).

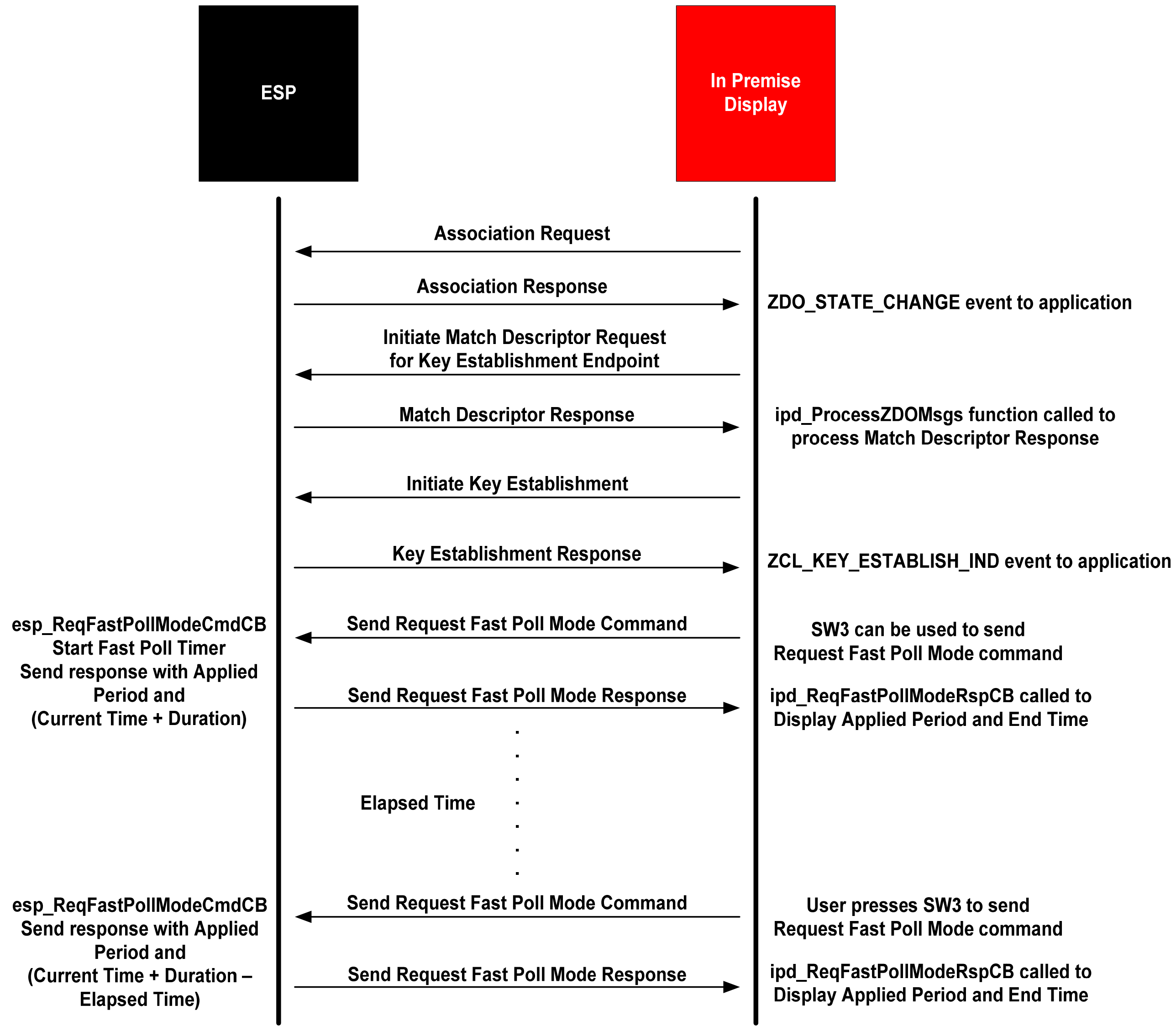

ESP
In Premise Display
Association Request
Association Response
ZDO_STATE_CHANGE event to application
Initiate Match Descriptor Request for Key Establishment Endpoint
Match Descriptor Response
ipd_ProcessZDOMsgs function called to process Match Descriptor Response
Initiate Key Establishment
Key Establishment Response
ZCL_KEY_ESTABLISH_IND event to application
esp_ReqFastPollModeCmdCB Start Fast Poll Timer Send response with Applied Period and (Current Time + Duration)
Send Request Fast Poll Mode Command
SW3 can be used to send Request Fast Poll Mode command
Send Request Fast Poll Mode Response
ipd_ReqFastPollModeRspCB called to Display Applied Period and End Time
Elapsed Time
esp_ReqFastPollModeCmdCB Send response with Applied Period and (Current Time + Duration – Elapsed Time)
Send Request Fast Poll Mode Command
User presses SW3 to send Request Fast Poll Mode command
Send Request Fast Poll Mode Response
ipd_ReqFastPollModeRspCB called to Display Applied Period and End Time

Figure 8. Sequence Diagram for an In Premise Display Fast Polling Test

The IPD application makes function calls to the following ZCL SE functions and ZDO API functions:

zclSE_SimpleMetering_Send_ReqFastPollModeCmd – this function sends the request fast poll mode command to the ESP.

ipd_ReqFastPollModeRspCB – this function is called when the IPD device receives the request fast poll mode response from the ESP, and displays the applied update period and the end time of the fast poll period on the LCD.

ipd_HandleKeys – user switch events are processed here. HOLD_AUTO_START is defined so the device will normally not join the network upon startup. The user has to press SW1, which

then calls ZDOInitDevice() to start the device. The rationale for doing this is so that the user can have time to configure the Certicom keys prior to joining the network.
The user has to press SW3 to send zclSE_SimpleMetering_Send_ReqFastPollModeCmd() to ESP.

Once the IPD device joins the network, it goes through the state machine explained in Figure 3. ipd_KeyEstablish_ReturnLinkKey() is called to check if a link key has already been established with the ESP. If one hasn't been established, it will send out a match descriptor request in search of the endpoint on the ESP that has the key establishment cluster. Upon processing the match descriptor response in the ipd_ProcessZDOMsgs() function, the osal event IPD_KEY_ESTABLISHMENT_REQUEST_EVT is set. The event handler for this in the process event loop will then call zclGeneral_KeyEstablish_InitiateKeyEstablishment() to do the CBKE procedure. Upon its success, the application will receive a ZCL_KEY_ESTABLISH_IND system message. The user can press SW3 to send Resquest Fast Poll Mode Command to ESP; ESP will send a Request Fast Poll Mode Response to IPD. If the user presses SW3 within the fast poll period, the IPD will send a Request Fast Poll Mode Command and ESP will return Request Fast Poll Mode Response with applied update period and original poll period end time, the end time is not extended with the new request.

4.8.3. Prepayment Test

ZCL_PREPAYMENT must be enabled in f8wZCL.cfg to perform prepayment test and enable all related code.

The user can press SW3 in the In Premise Display to send the Select Available Emergency Credit command to the ESP. The ESP will display the site ID and the meter serial number used for the purpose of this sample application.

The user can press SW4 in the In Premise Display to send the Change Supply command to the ESP. Once the IPD receives Supply Status Response the information will be displayed on its LCD.

See Figure 7 for a sequence diagram.

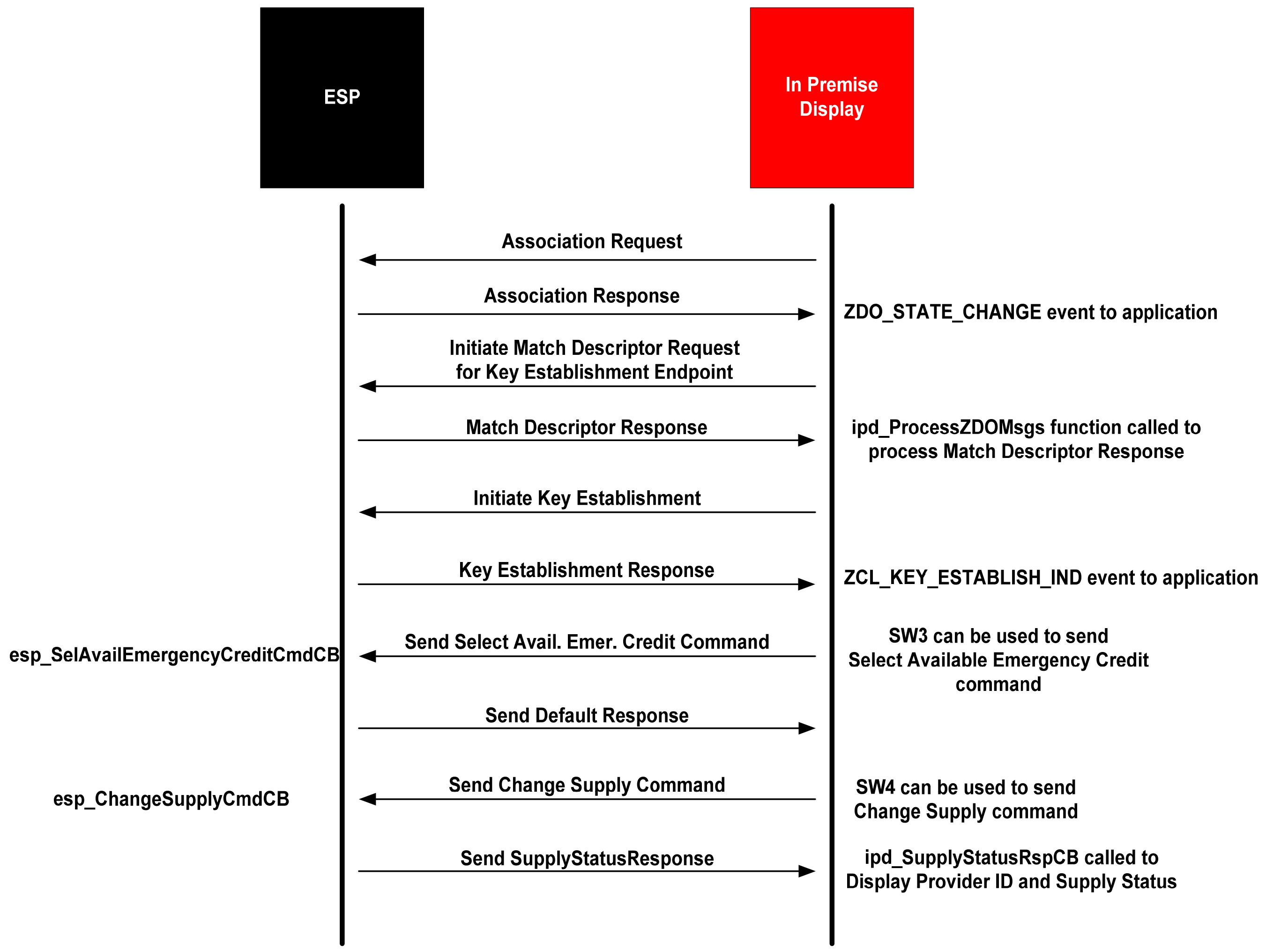

ESP
In Premise Display
Association Request
Association Response
ZDO_STATE_CHANGE event to application
Initiate Match Descriptor Request for Key Establishment Endpoint
Match Descriptor Response
ipd_ProcessZDOMsgs function called to process Match Descriptor Response
Initiate Key Establishment
Key Establishment Response
ZCL_KEY_ESTABLISH_IND event to application
esp_SelAvailEmergencyCreditCmdCB
Send Select Avail. Emer. Credit Command
SW3 can be used to send Select Available Emergency Credit command
Send Default Response
esp_ChangeSupplyCmdCB
Send Change Supply Command
SW4 can be used to send Change Supply command
Send SupplyStatusResponse
ipd_SupplyStatusRspCB called to Display Provider ID and Supply Status

Figure 9. Sequence Diagram for an In Premise Display Prepayment Test

The IPD application makes function calls to the following ZCL SE functions and ZDO API functions:

zclSE_Prepayment_Send_SelAvailEmergencyCredit – this function sends the select available emergency credit command to the ESP.

zclSE_Prepayment_Send_ChangeSupply – this function sends the change supply command to the ESP.

ipd_SupplyStatusRspCB – this function is called when the IPD device receives the supply status response command from the ESP, and displays the provider ID and supply status on the LCD.

ipd_HandleKeys – user switch events are processed here. HOLD_AUTO_START is defined so the device will normally not join the network upon startup. The user has to press SW1, which then calls ZDOInitDevice() to start the device. The rationale for doing this is so that the user can have time to configure the Certicom keys prior to joining the network.

Once the IPD device joins the network, it goes through the state machine explained in Figure 3. ipd_KeyEstablish_ReturnLinkKey() is called to check if a link key has already been established with the ESP. If one hasn't been established, it will send out a match descriptor request in search of the endpoint on the ESP that has the key establishment cluster. Upon processing the match descriptor response in the ipd_ProcessZDOMsgs() function, the osal event IPD_KEY_ESTABLISHMENT_REQUEST_EVT is set. The event handler for this in the process event loop will then call zclGeneral_KeyEstablish_InitiateKeyEstablishment() to do the CBKE procedure. The user can press SW3 to send Select Available Emergency Credit Command to ESP; ESP will send a Default Response to IPD. The user can press SW4 to send Change Supply Command to ESP; ESP will return Supply Status Response with provider ID, implementation date and time, and proposed supply status.

4.9. In Premise Display – OTA

The In Premise Display – OTA configuration provides the same functionality as the In Premise Display configuration, except that it provides client support for the OTA upgrade cluster. This sample app configuration should be used with the OTA Dongle Coordinator application as described in [6], where more information can be found regarding how to exercise the OTA upgrade cluster functionality. Figure 10 shows the high-level interactions between the In Premise Display – OTA upgrade client and the OTA upgrade server.

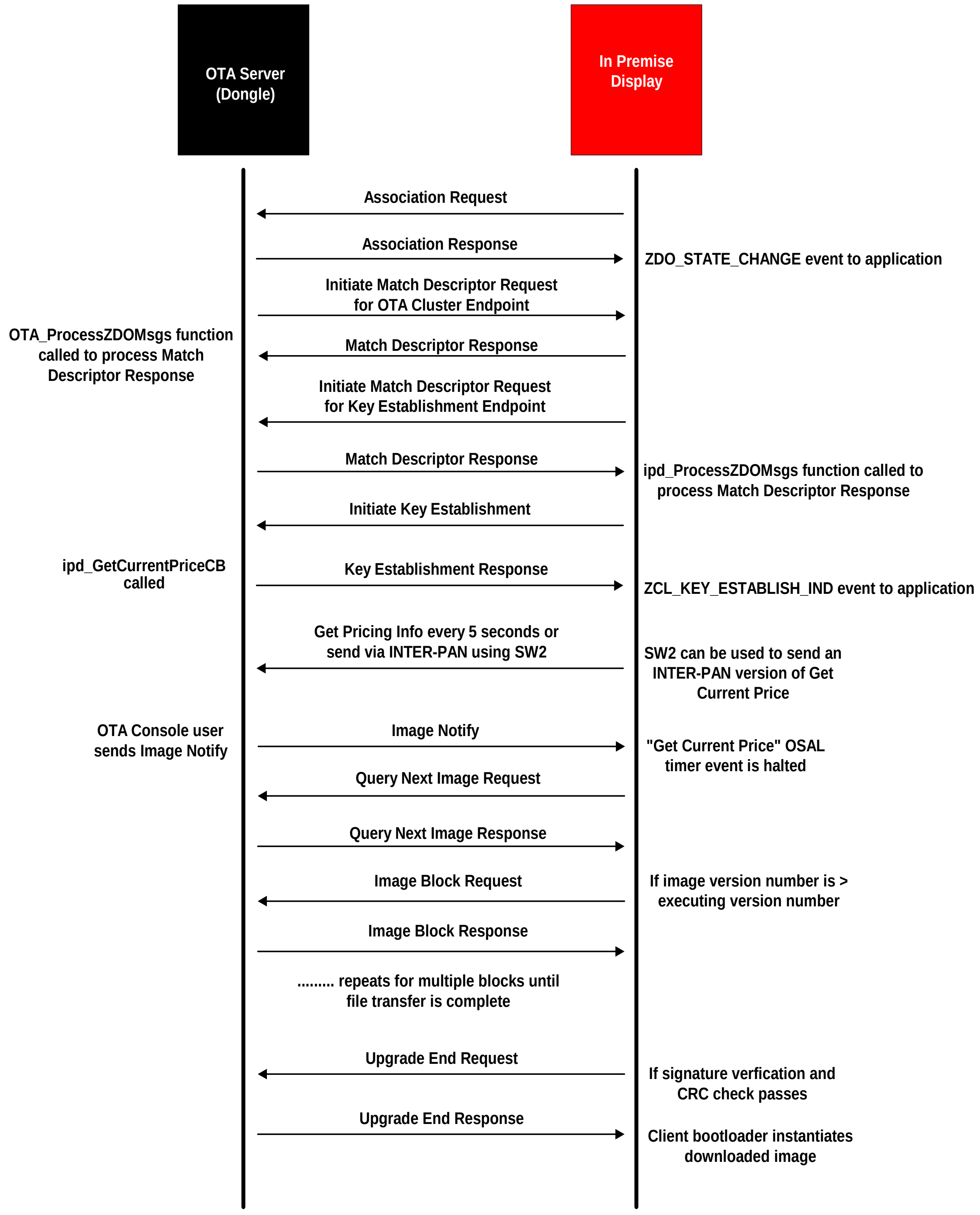

OTA Server (Dongle)
In Premise Display
Association Request
Association Response
ZDO_STATE_CHANGE event to application
Initiate Match Descriptor Request for OTA Cluster Endpoint
OTA_ProcessZDOMsgs function called to process Match Descriptor Response
Match Descriptor Response
Initiate Match Descriptor Request for Key Establishment Endpoint
Match Descriptor Response
ipd_ProcessZDOMsgs function called to process Match Descriptor Response
Initiate Key Establishment
ipd_GetCurrentPriceCB called
Key Establishment Response
ZCL_KEY_ESTABLISH_IND event to application
Get Pricing Info every 5 seconds or send via INTER-PAN using SW2
SW2 can be used to send an INTER-PAN version of Get Current Price
OTA Console user sends Image Notify
Image Notify
"Get Current Price" OSAL timer event is halted
Query Next Image Request
Query Next Image Response
Image Block Request
If image version number is > executing version number
Image Block Response
.... repeats for multiple blocks until file transfer is complete
Upgrade End Request
If signature verfication and CRC check passes
Upgrade End Response
Client bootloader instantiates downloaded image

Figure 10. Sequence Diagram for an In Premise Display with OTA cluster client support

OSAL_ipd.c - functions and tables for task initialization
ipd.c – main application function that has init and event loop function
ipd.h – header file for application module
ipd_data.c – container for declaration of attributes, clusters, simple descriptor

The IPD application makes function calls to the following ZCL SE functions and ZDO API functions:

zclSE_Pricing_Send_GetCurrentPrice – this function sends the get current price request to the ESP.

ipd_PublishPriceCB – this function is called when the IPD device receives the publish price command from the ESP, and displays the provider ID on the LCD.

ipd_HandleKeys – user switch events are processed here. HOLD_AUTO_START is defined so the device will normally not join the network upon startup. The user has to press SW1, which then calls ZDOInitDevice() to start the device. The rationale for doing this is so that the user can have time to configure the Certicom keys prior to joining the network.

ipd_ProcessOTAMsgs – this function is called whenever a callback message from the ZCL OTA application task is processed, such as when the download has commenced or completed.

Once the IPD device joins the network, it goes through the state machine explained in Figure 3. ipd_KeyEstablish_ReturnLinkKey() is called to check if a link key has already been established with the ESP. If one hasn't been established, it will send out a match descriptor request in search of the endpoint on the ESP that has the key establishment cluster. Upon processing the match descriptor response in the ipd_ProcessZDOMsgs() function, the osal event IPD_KEY_ESTABLISHMENT_REQUEST_EVT is set. The event handler for this in the process event loop will then call zclGeneral_KeyEstablish_InitiateKeyEstablishment() to do the CBKE procedure. Upon its success, the application will receive a ZCL_KEY_ESTABLISH_IND system message. An osal timer event called IPD_GET_PRICING_INFO_EVT is then started to send the get pricing info request every 5 seconds. The result of the pricing request is received via ipd_PublishPriceCB and is displayed on the LCD. Only the provider ID field of the published price payload is displayed.

4.10. Range Extender

The Range Extender device will not exchange application data with the ESP. It will be able to join the SE network, perform key establishment, and route packets. Figure 11 shows a sequence diagram for the Range Extender.

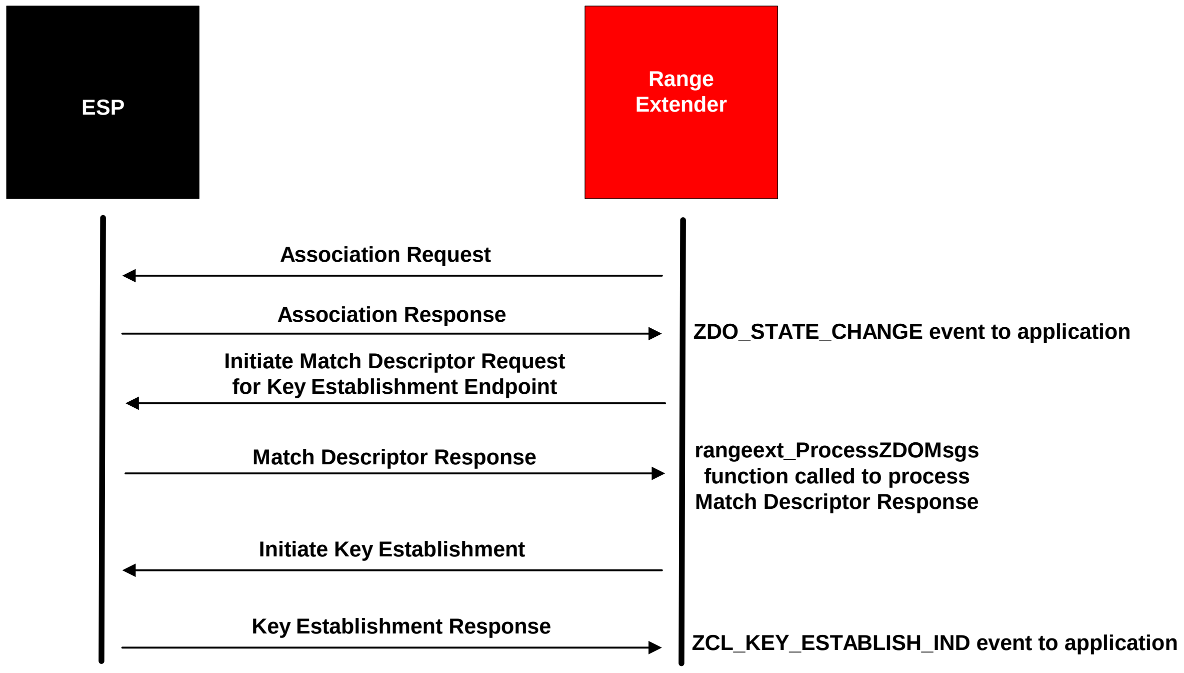

ESP
Range Extender
Association Request
Association Response
ZDO_STATE_CHANGE event to application
Initiate Match Descriptor Request for Key Establishment Endpoint
Match Descriptor Response
rangeext_ProcessZDOMsgs function called to process Match Descriptor Response
Initiate Key Establishment
Key Establishment Response
ZCL_KEY_ESTABLISH_IND event to application

Figure 11. Sequence Diagram for a Range Extender

OSAL_rangeext.c - functions and tables for task initialization
rangeext.c – main application function that has init and event loop function
rangeext.h – header file for application module
rangeext_data.c – container for declaration of attributes, clusters, simple descriptor

The Range Extender application makes function calls to the following functions:

rangeext_HandleKeys – user switch events are processed here. HOLD_AUTO_START is defined so the device will normally not join the network upon startup. The user has to press SW1, which then calls ZDOInitDevice() to start the device. The rationale for doing this is so that the user can have time to configure the Certicom keys prior to joining the network.

The Range Extender application does not send any application level SE messages, but it does join the network and perform key establishment. It functions as a standard Zigbee Pro router device.

5. Limitations

5.1. Trust Center Operation

- The trust center does not demonstrate how to refresh network keys.
- The trust center does not demonstrate how to manage an access control list
- The trust center assumes that joining devices have the same pre-configured trust center link key. It is assumed that the trust center link key is already available. Out of band means of deriving the trust center link key is out of the scope of this sample application.

5.2. Network Manager Operation

- The ESP is the only device that has the Network Manager functionality.
- The ESP application instance should be compiled with the NWK_MANAGER compile option in order to enable Network Manager functionality hooks.
- The default network manager application ZDNwkMgr.c is provided “as is” and there is no application level functionality incorporated to demonstrate this Network Manager application.

5.3. Secure Joining Operation

All devices must have the pre-configured Trust Center Link Key defined at compile time, or entered via Z-Tool.

5.4. Key Establishment Operation

SE Key Establishment is always initiated by the joining device and its partner will always be the ESP.

5.5. Device Startup Behavior

The Smart Energy Profile spec mentions application best practices of controlling join/rejoin duty cycles. This sample app does not make any attempt at changing the join/rejoin mechanism as implemented in ZDApp.c. The Smart Energy Profile spec says that end devices should not be more than 7.5 seconds. This sample application therefore supports a nominal poll rate of 8 seconds. The NWK_INDIRECT_MSG_TIMEOUT parameter in the f8wConfig.cfg file must be increased to a suggested value of 10 in order to buffer the message long enough to accommodate the 8 second poll period by the end device. The MAX_POLL_FAILURE_RETRIES parameter should also be set to 4 as during the CBKE procedure the ESP will be somewhat unresponsive to data requests during this time.

5.6. Load Control Device Behavior

There is no example of how to cancel a load control event from the ESP or how to supersede an ongoing load control event.

5.7. ESP Behavior

The ESP service discovery implementation is limited to supporting up to two load control devices at any given time, but could be easily modified to support more.

6. Applicable Documents

6.1. Z-Stack Documents (part of the Z-Stack installer)

1. OSAL API, Texas Instruments Document SWRA194
2. Z-Stack API, Texas Instruments Document SWRA195
3. Z-Stack ZCL API, Texas Instruments Document SWRA197
4. Z-Stack Monitor and Test API, Texas Instruments Document SWRA198
5. Z-Stack Smart Energy Developer's Guide, Texas Instruments Document SWRA216
6. Z-Stack OTA Upgrade User's Guide, Texas Instruments Document SWRA353

6.2. Other Documents (www.zigbee.org)

1. ZigBee Alliance – Smart Energy Profile Specification
2. ZigBee Alliance – ZigBee Smart Energy Test Specification
3. ZigBee Alliance – ZigBee OTA Upgrade Cluster Specification