Creating a ZigBee® Smart Energy Device with the MSP430F54xx and the CC2530-ZNP (ZigBee Pro Network Processor)

Physical interface and development platform

Although the CC2530-ZNP supports UART, USB, and SPI interfaces, we’ll be discussing use of the SPI interface. Table 1 shows the recommended pin connection between the host processor and the CC2530-ZNP, which matches the interconnect used by the EXP430F5438 Experimenter Board where the MSP430F5438 USCI B0 is used in 3-wire SPI mode.

<table>
<thead>
<tr>
<th>MSP430F5438</th>
<th>Direction</th>
<th>CC2530-ZNP</th>
<th>Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>P3.0</td>
<td>Out</td>
<td>P1.4 (SS)</td>
<td>In</td>
</tr>
<tr>
<td>P3.1 (UCB0SIMO)</td>
<td>Out</td>
<td>P1.6 (MOSI)</td>
<td>In</td>
</tr>
<tr>
<td>P3.2 (UCB0SOMI)</td>
<td>In</td>
<td>P1.7 (MISO)</td>
<td>Out</td>
</tr>
<tr>
<td>P3.3 (UCB0CLK)</td>
<td>Out</td>
<td>P1.5 (SCLK)</td>
<td>In</td>
</tr>
<tr>
<td>P1.2</td>
<td>Out</td>
<td>RESET_N</td>
<td>In</td>
</tr>
<tr>
<td>P1.4</td>
<td>In</td>
<td>P0.4 (SRDY)</td>
<td>Out</td>
</tr>
<tr>
<td>P1.6</td>
<td>Out</td>
<td>P0.3 (MRDY)</td>
<td>In</td>
</tr>
<tr>
<td>P8.2</td>
<td>Out</td>
<td>P2.0 (CFG1, driven HIGH for SPI transport)</td>
<td>In</td>
</tr>
<tr>
<td>N/C</td>
<td>Configured as input with pull-up</td>
<td>P1.2 (CFG0)</td>
<td>In</td>
</tr>
</tbody>
</table>

Table 1. MSP430F5438 to CC2530-ZNP interconnect.

To test this connection, one can obtain a CC2530EMK (which contains 2× CC2530 evaluation modules (EMs)) and 2× EXP430F5438 Experimenter Boards separately. This setup is shown in Figure 1 on the following page. One will also need the SmartRF05EB board which is used to program the CC2530EM modules. The SmartRF05EB board comes included with either the CC2530DK, CC2530ZDK, or CC2520DK.
The software application framework is built upon the OSAL (Operating System Abstraction Layer) operating system (see reference [2] for more information) from TI. OSAL provides services for task management, power management, non-volatile memory, dynamic memory management, software timers, event generation, inter- and intra-task messaging (or message queues), and a seamless interface to the layer that abstracts the board support package (BSP) called the Hardware Abstraction Layer (HAL). The ZAP framework re-uses the architecture of the ZigBee® application framework on TI’s ZigBee stack (Z-Stack™). Each layer of the application framework is organized as a task within OSAL, and the HAL task is always given top priority, as shown in the system diagram of OSAL ZAP services in Figure 2. The HAL task is the first task that runs and...
after each subsequent task runs to completion, a branch is always taken back to the HAL task to see if it needs to be serviced. The tasks in the diagram shown are typical of a ZAP implementation.

The ZAP layer shown in Figure 3 acts as a translation layer between Z-Stack™ APIs and ZNP APIs. Therefore, a customer application written for the full ZigBee® stack will work as is when used on a ZAP platform that TI supports. This saves the customer from having to learn a different application development philosophy when using the CC2530-ZNP. In the sections ahead, we’ll examine how the ZAP layer takes care of translating application API calls to ZNP API calls for configuration and data exchanges. For this purpose we use the Smart Energy Sample Application Project which can be found in C:\Texas Instruments\ZAP-MSP430-1.0.0\Projects\zstack\ZAP\SE-SampleApp\EXP5438 after installing the ZAP installer package.

**ZNP startup**

The macro HAL_SPI_INIT() takes care of initializing the SPI interface of the MSP430™. The function zapPhyInit() is used to initialize the ZNP hardware control signals and allow the CC2530-ZNP to run by de-asserting the RESET_N pin. Once the ZNP has finished its internal initialization sequence, it signals completion to the host processor by asserting the SRDY signal. The zapPhySpiPoll() process will then retrieve the SYS_RESET_IND command. Subsequently, the zapSysResetInd() function is called to alert the ZAP layer that it has received the reset indication and that it can now go ahead and configure the ZNP. From now on, we’ll call this the “ZNP sync event”.

![Figure 3. Layers of the ZAP framework](image-url)
The first thing the user has to do is decide whether to restore the network state of the ZNP device every time a ZNP sync event occurs. This is selectable by the compile option ZAP_NV_RESTORE. If ZAP_NV_RESTORE is set to FALSE, the network state of the ZNP will be cleared, and default configuration used. zapAfSync() and zapZdoSync() are used to register application endpoints and ZDO callbacks, respectively. The host application must do this upon every ZNP sync event since application endpoint and callback information are not saved in ZNP non-volatile memory.

The application then makes a series of calls using znp_nv_write for the items ZCD_NV_LOGICAL_TYPE, ZDAPP_CONFIG_PAN_ID, DEFAULT_CHANLIST to configure the logical device type (coordinator, router, or end device), PAN ID, and scan channel mask, respectively. The default values from the zap.cfg configuration file are taken as values. When the compile option TC_LINKKEY_JOIN is defined, zapCertSync() is called to configure test certificate data for ZigBee® CBKE (Certificate Based Key Establishment) security. Note that since these configuration items are declared as global variables within the ZNP, a reset of the ZNP is not required to change these items at runtime. Note that this configuration of ZNP has been automated if the ZAP_AUTO_CFG compile option is set to TRUE in zap.cfg.

Once all the configuration changes are made, the host processor can then start the device by calling ZDOinitDevice which in turn calls the ZDO_STARTUP_FROM_APP ZNP API function. The compile option ZAP_ZDO_STARTUP_AREQ is used to allow the ZDO_STARTUP_FROM_APP command to be sent as an asynchronous request (AREQ) because the device startup process (which includes scanning for networks) could take > 400 msec. Therefore, this allows the application to service other tasks instead of having to wait for the synchronous response (SRSP). The status of this request will be queued for the host processor as a ZDO_STATE_CHANGE_IND command. This indication is then passed up to the application task to notify it that the device has a valid ZigBee network address. The application then processes this event as:

```c
    case ZDO_STATE_CHANGE:
        if ((DEV_END_DEVICE == (devStates_t)(MSGpkt->hdr.status)) ||
            (DEV_ROUTER == (devStates_t)(MSGpkt->hdr.status)))
        {
            // device is on the ZigBee network
        }
```

Note that this network startup of ZNP has been automated if the ZAP_AUTO_START compile option is set to TRUE in zap.cfg.

Part of the application’s initialization function is to register for the SE endpoint and desired ZDO callbacks. For example, the function zclSE_Init registers the SE endpoint description with the ZNP.
This translates into a ZAP proxy function called znp_afRegisterExtended() which uses the AF_REGISTER ZNP API to register this endpoint with the ZNP.

The application can also register for certain ZDO callback notifications. The Energy Service Portal (ESP) example registers for the following ZDO callbacks:

```c
// register for end device announce and simple descriptor responses
ZDO_RegisterForZDOMsg( espTaskID, Device_annce );
ZDO_RegisterForZDOMsg( espTaskID, Simple_Desc_rsp );
```

The ZDO_RegisterForZDOMsg() function then calls the ZAP proxy function znp_ZDO_RegisterForZDOMsg(), which sends the ZDO_MSG_CB_REGISTER command to the ZNP. When the ZAP layer receives an incoming ZDO command, the function zapZdoProcessIncoming is called. The command is then processed in the case handler for MT_ZDO_MSG_CB_INCOMING. This command handler then calls ZDO_SendMsgCBs in order to parse and determine which application task should receive the callback.

The ESP application uses the end device announce to detect when other devices join the SE network. The ESP then uses the ZDO simple descriptor request to query the device for what type of device it is. Both callbacks are processed as follows:

```c
static void esp_ProcessZDOMsg( zdoIncomingMsg_t *inMsg )
{
    ZDO_DeviceAnnce_t devAnnce;

    switch ( inMsg->clusterID )
    {
        case Device_annce:
        {
            ZDO_ParseDeviceAnnce( inMsg, &devAnnce );
            simpleDescReqAddr.addrMode = (afAddrMode_t)Addr16Bit;
            simpleDescReqAddr.addr.shortAddr = devAnnce.nwkAddr;

            // set simple descriptor query event
            osal_set_event( espTaskID, SIMPLE_DESC_QUERY_EVT );
        }
    }
}
```
When full ZigBee® CBKE security is used with the Certicom Elliptic Curve Cryptography (ECC) library, in order to establish an application link key with the ESP, the function zclGeneral_KeyEstablish_InitiateKeyEstablishment is called. This in turn sends the UTIL_ZCL_KEY_EST_INIT_EST command to the ZNP. As the entire ECC process could take up to 20 seconds, zapPhyWait(zapAppPort, ZCL_KEY_EST_INIT_EST_WAIT) is called to tell the low-level SPI driver to wait at least 30 seconds before timing out on the SRSP. When the key establishment process completes, the zapUtilProcessIncoming function will notify the application via the MT_UTIL_ZCL_KEY_ESTABLISH_IND handler. The application can then decide to use this indication to determine that it can start to communicate with the ESP. For example, in the simple meter example we have:

```c
case ZCL_KEY_ESTABLISH_IND:
    if ((MSGpkt->hdr.status) == TermKeyStatus_Success)
    {
        osal_start_timerEx( simpleMeterTaskID, SIMPLEMETER_REPORT_ATTRIBUTE_EVT, SIMPLEMETER_REPORT_PERIOD );
    }
```

Once the key establishment procedure is complete, sending application commands is straightforward. The ZCL layer provides API functions to send correctly formatted SE commands, such as the following, which is used in the simple meter example to send a report attribute command to the ESP:

```c
zcl_SendReportCmd( SIMPLEMETER_ENDPOINT, &ESPAddr,
        ZCL_CLUSTER_ID_SE_SIMPLE_METERING, pReportCmd,
        ZCL_FRAME_SERVER_CLIENT_DIR, 1, 0 );
```
These commands are translated into AF_DATA_REQUEST commands that are sent to the ZNP.

In the Smart Energy Sample Application Project, there are also other device configurations such as the In-Premise Display and Load Control Device that show examples of how to use commands related to Pricing and Demand Response and Load Control, respectively.

Any incoming over-the-air message is extracted by the host processor (as an AF_INCOMING_CMD) and then passed up to the ZCL layer. The application registers callbacks for each command that it wishes to process on the receive side. As the ESP has to support all SE commands, all SE callbacks are registered by the ESP application:

```c
static zclSE_AppCallbacks_t esp_SECmdCallbacks =
{
    esp_GetProfileCmdCB,                      // Get Profile Command
    esp_GetProfileRspCB,                      // Get Profile Response
    esp_ReqMirrorCmdCB,                      // Request Mirror Command
    esp_ReqMirrorRspCB,                       // Request Mirror Response
    esp_MirrorRemCmdCB,                      // Mirror Remove Command
    esp_MirrorRemRspCB,                       // Mirror Remove Response
    esp_GetScheduledPriceCB,                  // Get Scheduled Price
    esp_PublishPriceCB,                       // Publish Price
    esp_DisplayMessageCB,                     // Display Message Command
    esp_CancelMessageCB,                      // Cancel Message Command
    esp_GetLastMessageCB,                     // Get Last Message Command
    esp_MessageConfirmationCB,                // Message Confirmation
    esp_LoadControlEventCB,                  // Load Control Event
    esp_CancelLoadControlEventCB,             // Cancel Load Control Event
    esp_CancelAllLoadControlEventsCB,         // Cancel All Load Control Events
    esp_ReportEventStatusCB,                  // Report Event Status
    esp_GetScheduledEventCB,                  // Get Scheduled Event
};
```

For example when the ESP receives a Get Current Price command from an In-Premise Display, the esp_GetCurrentPriceCB callback function is called and the Publish Price command is sent back:
Conclusion

This paper has described how to create a ZigBee® Smart Energy device using the MSP430F54xx as the host application processor to the CC2530-ZNP (ZigBee Pro Network Processor). A typical smart meter application that would benefit most from this type of architecture would be a gas or water meter since extended battery life of 10 years or greater is a key requirement for these devices. The MSP430F54xx and CC2530-ZNP combination provides a low-power and low-cost solution that meets this requirement. The ZAP software application framework provides an easy-to-use development environment that accelerates ZigBee development for smart meter applications.
References

[2] OSAL API document, SWRA194
[4] ZCL API document, SWRA197

For more information on Smart Metering devices from TI, visit www.ti.com/metering, or e-mail: metering@ti.com
IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of TI.

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

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

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or “enhanced plastic.” Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products

- Amplifiers: amplifier.ti.com
- Data Converters: dataconverter.ti.com
- DLP® Products: www.dlp.com
- DSP: dsp.ti.com
- Clocks and Timers: www.ti.com/clocks
- Interface: interface.ti.com
- Logic: logic.ti.com
- Power Mgmt: power.ti.com
- Microcontrollers: microcontroller.ti.com
- RFID: www.ti-rfid.com
- RF/IF and ZigBee® Solutions: www.ti.com/lprf

Applications

- Audio: www.ti.com/audio
- Automotive: www.ti.com/automotive
- Communications and Telecom: www.ti.com/communications
- Computers and Peripherals: www.ti.com/computers
- Consumer Electronics: www.ti.com/consumer-apps
- Energy: www.ti.com/energy
- Industrial: www.ti.com/industrial
- Medical: www.ti.com/medical
- Security: www.ti.com/security
- Space, Avionics & Defense: www.ti.com/space-hypersonics-defense
- Wireless: www.ti.com/wireless-apps
- Video and Imaging: www.ti.com/video

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2010, Texas Instruments Incorporated