Optimizing CC2530 Z-Stack 3.0.2 Flash and RAM

This document is intended to help developers optimize the Flash and RAM usage of Z-Stack 3.0.2 when developing their own application using the CC2530 platform. It will also describe some limitations of using the CC2530 device as a Zigbee Coordinator device configuration due to RAM dependencies. The SampleSwitch Router project from Z-Stack 3.0.2 will be used throughout as an example.

1 Initial Out-of-Box Code Size

Z-Stack 3.0.2 can be downloaded from the Texas Instruments™ Z-STACK tool page and afterwards will be located by default in C:\Texas Instruments. The first step is to load a CC2530 project workspace inside IAR EW 8051 v10.20. This is accomplished through selecting *File* > *Open Workspace* and finding the correct workspace file, in this case **SampleSwitch.eww** is located here:

C:\Texas Instruments\Z-Stack 3.0.2\Projects\zstack\HomeAutomation\SampleSwitch\CC2530DB

Then select *Project -> Configuration* and choose the Router option, followed by *Project* > *Make* (or F7) to build the SampleSwitch Router project from Z-Stack 3.0.2 without any modifications. After building the code, refer to the bottom of the .map file (located in the Output folder) to see the memory footprint starting point:

```
239 555 bytes of CODE  memory
     32 bytes of DATA  memory (+ 68 absolute )
  7 425 bytes of XDATA memory
    192 bytes of IDATA memory
      8 bits  of BIT   memory
  1 867 bytes of CONST memory
```

The important numbers on this list are CODE and XDATA memory, which correspond to Flash and RAM, respectively.

2 Flash Optimizations

2.1 Removing the Example User Interface

The User Interface (UI) that is provided with the Z-Stack 3.0.2 sample applications provide a means to easily get a network up and running, view the network status on an LCD, and modify network commissioning parameters at runtime. However it also takes up a lot of code space and is unnecessary for developers to keep in their own custom applications.

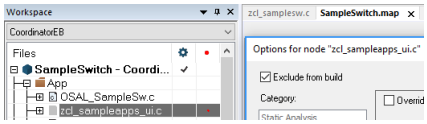
Three things must be done to remove this interface:

- Exclude the UI files from compilation
- Remove the API calls to the UI functions in the sample app file(s)
- Modify the project compile flags to exclude LCD-related drivers

2.1.1 Exclude UI Files

First we must remove **App/zcl_sampleapps_ui.c** from compilation. You can do that by right clicking on the file and going to *Options...*, then excluding it from the build by clicking the check box at the top left of the prompt.

Figure 1. Exclude zcl_sampleapps_ui.c From Build



2.1.2 Remove API Calls to UI Functions

In `zcl_samplesw.c`, there are various calls to UI APIs that we must remove. Search this file for "UI_" and either comment out or remove this code entirely. An example of this is:

```
void zclSampleSw_Init( byte task_id )
{
...

  // UI_Init(zclSampleSw_TaskID, SAMPLEAPP_LCD_AUTO_UPDATE_EVT, SAMPLEAPP_KEY_AUTO_REPEAT_EVT,

  // &zclSampleSw_IdentifyTime, APP_TITLE, &zclSampleSw_UiUpdateLcd, zclSampleSw_UiStatesMain);

  // UI_UpdateLcd();

}
```

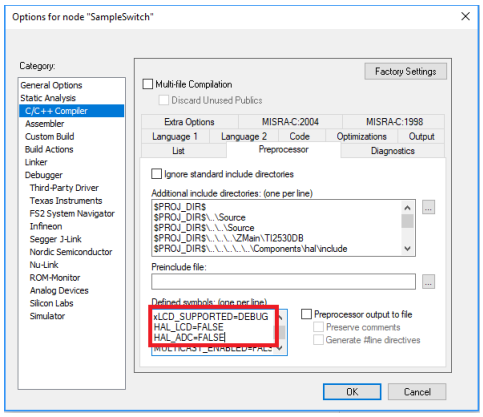
All "UI_" functions must be removed or else build errors will occur since they are located in the excluded `zcl_sampleapps_ui.c` file.

2.1.3 Modify the Compile Flags to Exclude LCD-Related Drivers

The compile flags for the project are accessed by going to *Project Options* and then navigating to *C/C++ Compiler* > *Preprocessor* > *Defined symbols*. From here, the following LCD-related changes can be made:

- Remove **LCD_SUPPORTED=DEBUG**
- Add **HAL_LCD=FALSE**

Figure 2. Accessing Preprocessor Defined Symbols



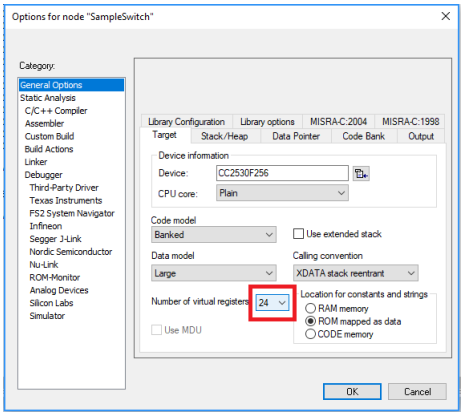
2.2 Removing/Modifying Driver Files

There are some drivers that are included by default in the Z-Stack sample applications that you may not be needed for a custom application. For example, to remove the ADC driver go to *Project Options* and navigate to *C/C++ Compiler* > *Preprocessor* > *Defined symbols* and then add the **HAL_ADC=FALSE** compile flag.

2.3 Changing the Number of Virtual Registers Available

This is a minor CC2530 device-specific optimization that can be done to save a few hundred bytes of Flash. Go to *Project Options* and navigate to *General Options* > *Target* > *Number of virtual registers*. By default this value is set to 16 but a more optimal value found for Z-Stack projects was 24. Attempts can be made to modify this value to various different sizes and observe which builds the smallest code size.

Figure 3. Changing Number of Virtual Registers



2.4 Supported ZCL Cluster Considerations

Make sure ZCL clusters that are not necessary for either device type or Zigbee certification do not exist in the application. For instance, in the SampleSwitch Router project, the **ZCL_GROUPS** cluster is included by default in the project compile flags. By the Zigbee Lighting & Occupancy Device Specification, this cluster is an optional cluster and can thus be removed from *C/C++ Compiler* > *Preprocessor* > *Defined symbols* of the *Project Options*. Use the Zigbee Cluster Library v6 Specification, the Zigbee Lighting & Occupancy Device Specification, and the Zigbee Home Automation Profile specification (all available from the Zigbee alliance website) to determine which clusters the chosen Zigbee device must support.

2.5 BDB_Reporting Considerations

Attribute report sending functionality in the Zigbee 3.0 specification is a mandatory feature for certain device types that support specific clusters. For instance, a Zigbee On/Off Light device must support the Client Side of the On/Off Cluster. This must support the "OnOff" attribute, and by the ZCL v6 specification this attribute must be reportable. If an application is going to support any cluster attributes that have a reportable access type (by the ZCL v6 specification), the compile flag **BDB_REPORTING** must be included in the application.

In Z-Stack, there are 4 separate macros that are related to ZCL reporting functionality:

1. **BDB_REPORTING:** defines the BDB reporting state machine that adds report sending functionality.
2. **ZCL_REPORTING_DEVICE:** defines the device that is sending reports.
3. **ZCL_REPORT_DESTINATION_DEVICE:** enables report receiving/processing functionality in your application code.
4. **ZCL_REPORT_CONFIGURING_DEVICE:** enables configuration of reporting parameters on remote devices.

The **BDB_REPORTING** feature adds about 9 kB to the Flash size. This state machine uses functions that are defined in `zcl.c` under the **ZCL_REPORTING_DEVICE** compile flag, so when **BDB_REPORTING** is defined then **ZCL_REPORTING_DEVICE** is automatically defined as well. **BDB_REPORTING**, however, does not automatically define the other two compile flags, **ZCL_REPORT_DESTINATION_DEVICE** and **ZCL_REPORT_CONFIGURING_DEVICE**, which must be added separately if their functionality is required. Although these are small features, less than 1 kB of Flash each, if a device does not require report sending functionality (the device does not support any clusters that have mandatorily reportable attributes by the ZCL specification) there is no need to waste the 9 kB of Flash needed for the BDB report sending functionality in order to support report processing and/or reporting configuration.

3 RAM Optimizations

3.1 UART Driver

If UART is needed by the application, it is add through the **HAL_UART** compile flag. By default, the UART driver uses 256-byte TX and RX buffers which takes up a combined 1 kB of RAM. Using UART with smaller buffer sizes is possible by modifying the compile flag **HAL_UART_DMA_RX_MAX=128**. Adding this to the *Defined symbols* will change the size of both the RX and the TX buffers from 256 to 128 bytes, and result in a RAM savings of ~500 bytes. But one drawback that must be considered is whether communication contains large data packets, for example the Monitor and Test (MT) interface for which frames are allowed to be at most 253 bytes in length. Under these circumstances the buffer size must remain at the default value.

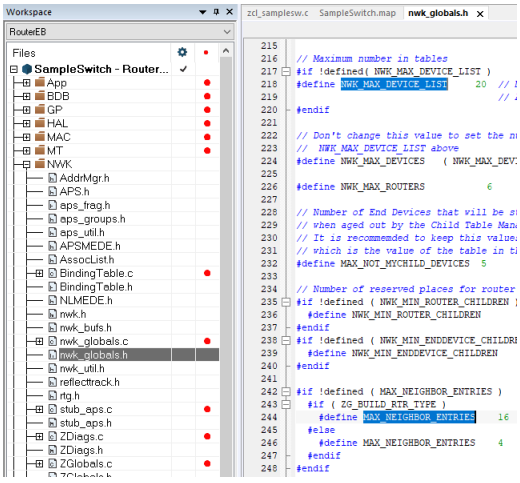
3.2 Device List Sizes

There are two device lists present on Zigbee routing devices (coordinators and routers) that take up a statically-allocated amount of RAM at compile time:

- `NWK_MAX_DEVICE_LIST`
  - Defines the number of directly-connected child devices supported
  - Each entry in this table is 28 bytes of RAM
  - Default size is 20
- `MAX_NEIGHBOR_ENTRIES`
  - Defines the number of "neighbor" devices, which is used in part of the Zigbee mesh-routing procedure. More neighbors <==> better mesh networking.
  - Each entry in this table is 23 bytes of RAM
  - Default value is 16

To redefine the size of one or both of these lists, the default values can be changed in **NWK/nwk_globals.h** or added to *Defined symbols* (for example, **NWK_MAX_DEVICE_LIST=15**).

Figure 4. Default Device List Sizes



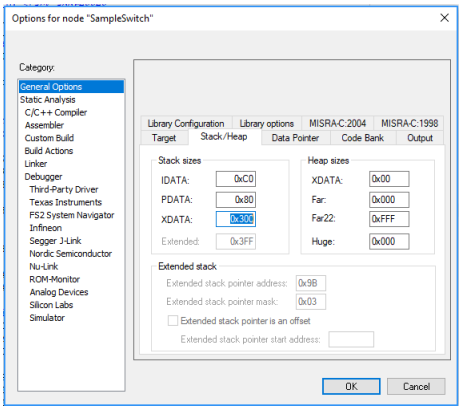
3.3 Heap Size

The heap size can be tuned by changing the value of `INT_HEAP_LEN`, which by default is 3072 bytes of RAM for routing devices in **ZMain/OnBoard.h**. Change the value here or add it to *Defined symbols* (for example, **INT_HEAP_LEN=2688**).

3.4 Stack Size

The stack size can be tuned by changing the value of the XDATA stack, which is defined in the IAR project options. Open *Project Options* and navigate to *General Options* > *Stack/Heap* > *Stack Sizes* > *XDATA*.

Figure 5. Altering Stack Size



By default this value is 0x400 or 1024 bytes of RAM, but 0x300 is found to be suitable for most basic Z-Stack applications.

4 Optimized Code Size

As an example of saving code space, the following changes were made to the SampleSwitch Router project:

- Removed the UI
- Removed ADC driver
- Removed **ZCL_GROUPS** cluster
- Optimized number of Virtual Registers
- Decreased heap to 2688 bytes
- Decreased `NWK_MAX_DEVICE_LIST` to 15 devices
- Decreased `MAX_NEIGHBOR_ENTRIES` to 10 devices

The numbers below reflect the new project build size:

```
225 182 bytes of CODE  memory
     40 bytes of DATA  memory (+ 59 absolute )
  6 435 bytes of XDATA memory
    192 bytes of IDATA memory
      8 bits  of BIT   memory
    660 bytes of CONST memory
```

In total, over 14 kB of Flash and around 1 kB of RAM was saved when compared to its out-of-box counterpart. This leaves about 37 kB of Flash and 1,750 bytes of RAM available for application expansion.

5 CC2530 Zigbee Coordinator Configuration Limitations

By implementing the same optimizations as before, the SampleSwitch Coordinator configuration uses 6762 bytes of XDATA memory on the CC2530 device, which leaves 1430 available bytes of RAM. This extra increase in memory allocation as compared to the Router device counterpart is primarily due to **ZDSECMGR_TC_DEVICE_MAX** (default of 40 from **ZDO/ZDSecMgr.h**), which allocates 8 non-volatile (NV) bytes per APS key. As a Zigbee 3.0 Trust Center (TC), the coordinator must store these keys to manage the network security and as such ZDSECMGR_TC_DEVICE_MAX determines the number of devices allowed to join the network.

Furthermore, if a coordinator is desired to act as a Many-to-One (MTO) data concentrator that records network route discovery (by setting **CONCENTRATOR_ENABLE** and **CONCENTRATOR_ROUTE_CACHE** to true in **NWK/ZGlobals.h**) then XDATA will be further populated by **MAX_RTG_SRC_ENTRIES** from **NWK/nwk_globals.h**, which is a default value of 12 and requires 6 bytes of RAM per entry. It must also be considered that the heap would have to be increased to accommodate a relay list pointer that is stored as the last element of each source route table entry. Transmitting this information through the MT interface as a Zigbee Network Processor (ZNP) would require additional RAM for the UART buffer.

Given these restrictions, along with the network topology and design criteria for an end application, it is understandable that the 8 kB of Flash residing on a CC2530 device will not be capable of supporting networks containing a large number of nodes. Under these circumstances, it is advised that the CC2652R or CC1352P SimpleLink™ Wireless MCUs be considered as an alternative. In addition to sub-µA sleep current and up to 80 kB of RAM retention, these devices support the SimpleLink CC26x2/CC13x2 SDKs that combines the TI-RTOS framework and Z-Stack 3.x for a reliable Zigbee solution, which is tested and maintained on a quarterly cadence. Visit the Zigbee Overview page for more device and solution offerings.

6 Trademarks

Texas Instruments, SimpleLink are trademarks of Texas Instruments.
All other trademarks are the property of their respective owners.

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale (www.ti.com/legal/termsofsale.html) or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

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