Application Note: Application-Level Tuning of Z-Stack

Document Number: F8W-2006-0005

Texas Instruments, Inc.
San Diego, California USA
(619) 497-3845

Version	Description	Date
1.0	Initial release.	12/08/2006
1.1	Changed document name. Updated title page.	05/21/2007

Table of Contents

1. Purpose

This document discusses the various global variables, compile options, and compiler directives that affect Z-Stack usage and configuration. This document applies to version 1.4.0 of the Z-Stack. The most common compiler command-line options used to define a Z-Stack device can now be set by editing a configuration file called f8wConfig.cfg. This file can be found in \Projects\zstack\Tools\CC2430DB or \Projects\zstack\Tools\CC2420DB, depending on the Z-Stack platform used.

2. Definitions

The following terms are used in this document:

APS – Application support sub-layer

MT – Monitor Test

NV – Non-volatile

AES – Advanced Encryption Engine

3. General Considerations

In general, the default settings should be used. Unless one wants to tune the stack for their application requirements, the default settings are adequate for most applications.

4. Global Variables

This section discusses the global variables in nwk_globals.c that the user can modify.

Application Layer

CONST byte apscMaxFrameRetries = APSC_MAX_FRAME_RETRIES; (APSC_MAX_FRAME_RETRIES in f8wConfig.cfg)

This variable controls the number of retries that will be attempted by the APS layer if APS acknowledgement service is used for the transmitted message. In order to use APS acknowledgement service, pass into the tx_options parameter the value of AF_ACK_REQUEST when calling the afDataRequest() function.

Security Options

In nwk_globals.c, there is the following section dedicated to security global variables.

```
/****************************************************************************
 * SECURITY GLOBAL VARIABLES
 */
// This is the default pre-configured key,
// change this to make a unique key
// SEC_KEY_LEN is defined in ssp.h.
CONST byte defaultKey[SEC_KEY_LEN] =
{
```



```
  // Key for In-House Testing
  0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
  0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F
}; (DEFAULT_KEY in f8wConfig.cfg)


// If true, preConfigKey should be configured on all devices on the network
// If false, it is configured only on the coordinator and sent to other devices upon joining
byte gPreConfigKeys = FALSE;          //  TRUE;


// Choose the aes engine
// HARDWARE_AES uses the hardware aes engine ( fastest setting )
// SOFTWARE_AES uses software aes  ( slowest setting )
// SW_AES_AND_KEY_EXP enables software aes with key expansion ( improves speed at the cost of 176 bytes of data (RAM) )
byte aesEngineSetting = HARDWARE_AES;
```

If gPreConfigKeys is set to TRUE, all devices should use the same pre-configured security key. If gPreConfigKeys is set to FALSE, the pre-configured key is set only on the coordinator device, and is handed to joining devices. The key is sent in the clear over the last hop. Upon reset, the device will retrieve the pre-configured key from NV memory if the NV_RESTORE and NV_INIT compile options are defined.

The variable aesEngineSetting controls whether AES encryption is done in hardware or software. By default it is set to HARDWARE_AES, which delivers the highest throughput when security is used.

5. Compile Options

The following compile options are used for debug trace and monitor test purposes.

MT_TASK

This compile option enables the device to be able to talk to the Z-Tool PC application, and to be able to output minimal debug information using the debug_str() function. Removing this saves some code and RAM usage.

MT_ZDO_FUNC and other MT compile options

MT_ZDO_FUNC is an example of a compile option that enables the ZDO commands to be used with Z-Tool. Most devices would not need all of the monitor test commands to be enabled. Each ZDO command can be enabled individually in ZDConfig.h. Selectively enabling requests that need to be in the application minimizes code usage. See the Z-Stack Compile options document for more information on the different MT compile options.

LCD_SUPPORTED

This turns on support for displaying debug trace information on development boards that support an LCD. On the CC2420DB, setting LCD_SUPPORTED enables sending information to the Serial Port (since it does not have an LCD display). Enabling this compile option uses a significant portion of RAM.

LCD_SUPPORTED=DEBUG

On the CC2430EB, setting LCD_SUPPORTED=DEBUG enables sending information to both the LCD and Serial Port, so code is actually larger. On the CC2430DB, since there is no LCD, debug information is only sent to the Serial Port, therefore using less code than the CC2430EB with the same setting.

BLINK_LEDS

The BLINK_LEDS compile option enables extended LED functionality at the expense of extra code and RAM space.

See the Z-Stack compile options document for more information.

6. Compiler Directives

The following compiler directives allow one to minimize RAM usage where applicable, or change different user settings.

Nwk_globals.h:

NWK_MAX_DEVICE_LIST

This is used to pre-allocate RAM for the child devices. Changing NWK_MAX_DEVICE_LIST by 1 changes the size of the association table by 18 bytes. This setting should match the MAX_CHILD_NODES #define to ensure that the device has enough RAM for the maximum number of children.

STACK_PROFILE_ID, MAX_CHILD_NODES, MAX_ROUTER_NODES, MAX_NODE_DEPTH

By default the **STACK_PROFILE_ID** is set to the HOME_CONTROLS stack profile. The stack profile ID determines the settings for MAX_CHILD_NODES, MAX_ROUTER_NODES, MAX_NODE DEPTH, NWK_MODE, and other network parameters. Therefore, changing the stack profile ID will typically allow the user to change the network topology using one #define. If one needs to define their own topology, one can use the NETWORK_SPECIFIC stack profile ID.

MAX_CHILD_NODES refers to how many child nodes a device can have. Of that **MAX_ROUTER_NODES** is allocated to router-child nodes (so the difference is allotted to end-device child nodes).

MAX_DEPTH refers to how many successive parent-child links are possible and thus imposes a maximum range of the network.

These should be set based on how dense your network is expected to be and what ratio of router to end-device devices you expect.

MAX_UNRESERVED_RTG_ENTRIES and MAX_RTG_ENTRIES

MAX_UNRESERVED_RTG_ENTRIES defines the actual number of routing table entries available for general use. The difference between this parameter and MAX_RTG_ENTRIES are entries for use only in the case of route repair. In the case where route requests are being done too often, the user may want to increase the number of unreserved entries. The routing entries refer to the number of routes that go through a particular node. If all devices are sending data to a single device, there will be only route (because data has to reach only a single destination). Otherwise there can be as many route as there are destinations. But it is unlikely all routes will not go through a given device.

MAX_BCAST_ENTRIES (MAX_BCAST in f8wConfig.cfg)

This is the number of broadcast entries (for data packets) that can be held for the period of _NIB.BroadcastDeliveryTime (in nwk_globals.c). Together, these two settings enforce the settling time for broadcast data packets traversing through the network. For example, if MAX_BCAST_ENTRIES is set to 9, and _NIB.BroadcastDeliveryTime is set to 10 (seconds), a device could send out 9 sequential broadcast data packets, but would not be able to send out a 10^{th} one until after 10 seconds.

_NIB.RouteExpiryTime (ROUTE_EXPIRY_TIME in f8wConfig.cfg)

This setting determines the number of seconds before a route expires if no data is transmitted on it (by default set to 30 which means that established routes will expire in 30 seconds from the time they are recorded). If routes change often (due to router devices moving locations or end devices moving around within the network), one will want the old routes to expire. The expiry time is reset to this value, each time a packet goes over a particular route. Therefore, routes only expire if there is no traffic along it. After a route expires, the routing table entry will be deleted. One can set this value to 0 to never allow routes to expire.

Nwk_globals.c:

POLL_RATE, QUEUED_POLL_RATE, and RESPONSE_POLL_RATE

These control the different poll intervals for end devices. See AN017 or AN030 for a detailed explanation of these #defines.

gMAX_POLL_FAILURE_RETRIES (MAX_POLL_FAILURE_RETRIES in f8wConfig.cfg)

Number of times an end device will poll its parent before indicating a loss of synchronization. Once this threshold has been exceeded, the end device will attempt a Network Rejoin Request (per the Zigbee 2006 Specification). Note that a larger value will cause longer delays for the child to rejoin the network.

gREJOIN_POLL_RATE (REJOIN_POLL_RATE in f8wConfig.cfg)

This is used as an alternate response poll rate only for the network rejoin request. This rate is determined by the response time of the parent that the device is trying to join.

gNWK_MAX_DATA_RETRIES (NWK_MAX_DATA_RETRIES in f8wConfig.cfg)

This controls the number of retries that is perform by the network layer for the next hop message once the MAC layer retries are exhausted for that message (note that MAC layer will retry 3 times before returning a failure status to the network layer).

Cskip Array

The Cskip values are automatically determined by the parameters MAX_CHILD_NODES, MAX_ROUTER_NODES, and MAX_NODE DEPTH in version 1.4.0 of the stack. The user does not need to modify this array by themselves. One does need to make sure that the values in CskipRtrs and CskipChldrn match the settings MAX_ROUTER_NODES and MAX_CHILD_NODES, respectively.

For example if one uses:

```
#if ( STACK_PROFILE_ID == HOME_CONTROLS )
  byte CskipRtrs[MAX_NODE_DEPTH+1] = {6,6,6,6,6,0};
```


byte CskipChldrn[MAX_NODE_DEPTH+1] = {20,20,20,20,20,0};

MAX_ROUTER_NODES = 6
MAX_CHILD_NODES = 20

MAX_NEIGHBOR_ENTRIES

MAX_NEIGHBOR_ENTRIES is for neighbor nodes that are not child/parent devices. This setting is unrelated to MAX_ROUTER_NODES. A value of between 4 and 8 is needed here because it is only used to ensure sufficient connectivity in the network. There is no benefit to having a large value.

BindingTable.h:

NWK_MAX_BINDING_ENTRIES

This controls the maximum number of entries in the binding table. If binding is not used, set this to 1 to be minimize RAM usage. Adjusting the number of entries by 1 changes the size of the binding table by 12 bytes.

MAX_BINDING_CLUSTER_IDS

This controls the maximum number of cluster IDs for each binding table entry. Set this to 1 if binding is not used in order to minimize RAM usage.

ZDApp.c:

DEF_PROTO_VERS

Starting with version 1.4.0 of the stack, versions 1.0 and 1.1 of the Zigbee specification are supported. The new preprocessor macro **DEF_PROTO_VERS** may be used to control the default protocol version of the network to be joined, or started. Please see AN026 for more details regarding the protocol versioning.

NWK_START_DELAY (NWK_START_DELAY in f8wConfig.cfg)

This #define gives the user the option of waiting NWK_START_DELAY number of milliseconds before starting up the device on the network.

ZMain.c:

gZDO_CONFIG_PAN_ID (ZDAPP_CONFIG_PAN_ID in f8wConfig.cfg)

In the zMain_NvInit() function the PAN ID global variable gZDO_CONFIG_PAN_ID is initialized. If it is something other than 0xFFFF, the coordinator will start with that PAN ID, and routers and end devices will join the PAN set to that ID. If the NV_INIT compile option is defined, the user can recall this value from NV memory. The user can configure this value in NV memory using Z-Tool.

nwkDefaultChannelList (DEFAULT_CHANLIST in f8wConfig.cfg)

In the zMain_NvInit() function the channel list global variable nwkDefaultChannelList is initialized with the setting from DEFAULT_CHANLIST. This is where the user can configure the channel list supported by their device.

