AN5492
Application note

Persistent data management ZigBee® and non-volatile memory in STM32WB Series

Introduction

This application note describes the implementation of the persistent data management ZigBee® and the non-volatile memory in the STM32WB Series. The stack context data and the application data are stored in a non-volatile memory (NVM). For the STM32WB Series the NVM is the internal Flash memory.

The NVM presented in this document is an example. The EEPROM emulator (EE) and Flash driver are given as an example of implementation. The integration of a proprietary EE and Flash driver implementation can be easily inserted to the project.

Contents

List of tables

List of figures

1 General information

This document applies to the STM32WB Series Arm®[a] Cortex®-M4/M0 core-based microcontrollers.

2 Terms and abbreviations

Table 1. Terms and abbreviations

Acronyms	Definitions
HAL	Hardware abstraction layer
RAM	Random-access memory
EEPROM	Electrically erasable programmable read only memory
NVM	Non-volatile memory
API	Application programming interface
EE	EEPROM emulator
Persistent data	Data to save in NVM
Flash	Cortex®- M4 Internal Flash
Cold start	Refers to a first time start or re-start without or invalid persistent data in memory held
Power cycle	Refers to cold start
Warm start	Refers to a re-start with valid persistent data in memory held.
Power off	Refers to power lost where RAM memory lost
MSC	Message sequence chart
Secured	Memory cannot be accessed from M4
Unsecured	Memory can be accessed from M4
ZC	ZigBee coordinator
ZR	ZigBee router
ZED	ZigBee end device

a. Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

3 Reference documents

- AN5500: ZSDK API for ZigBee® on STM32WB Series, available on *www.st.com.*

4 Overview

When data needed by a network node are stored only in RAM, they are maintained only while the node is powered and is lost if an interruption occurred such as a power shortage or a battery replacement. These data might be stack context needed by the network and application data such as cluster attributes. Persistence data is dynamic information such as frame counters and dynamic keys and is independent of static configuration data such as the IEEE address.

In order for the node to recover from a power shutdown and to resume its role in the network, data must be stored in NVM. Restarting from persistence does not involve leaving and joining again or rejoin the network, that is the node has remained on the network.

The STM32WB storage in Flash memory is done by the mean of dedicated persistent data APIs describes in this document.

The NVM implementation is emulating an EEPROM operations to finally write/read or erase the Flash memory in order to save Flash memory lifetime and to offer the end-user the possibility to save data in a real EEPROM or a Flash. Data and implementation are in the M4 and unsecured.

5 Persistent data management

5.1 Introduction

In order to perform a warm start any time after a power off, the needed data can be saved in NVM each time they change.

To achieve this in his application implementation the end-user can either:

- Manages by himself the moment he calls the save data in NVM
- Relies on the stack notification.

5.2 Register the stack notification

5.2.1 Enable the stack notification

The STM32WB ZigBee stack notifies the application each time persistent data change. In order to perform this the application must enable notification and to implement the associated callback, the following API must be called.

```
/* Register Persistent data change notification */
ZbPersistNotifyRegister(zigbee_app_info.zb, APP_ZIGBEE_persist_notify_cb,
NULL);
```

And the associated callback:

```
/**
  * @brief notify to save persistent data callback
  * @param zb: zigbee context pointer, cbarg: callback arg pointer
  * @retval: None
  */

  static void APP_ZIGBEE_persist_notify_cb(struct ZigbeeT * zb, void
*cbarg)
  {
APP_DBG("Notification to save persistent data requested from stack");
/* Save the persistent data */
   APP_ZIGBEE_persist_save();
  }
```

5.2.2 Disable the stack notification

In order to disable the notification, the application calls the stack API with NULL parameters:

```
/* Disable Notification */
ZbPersistNotifyRegister(zigbee_app_info.zb, NULL, NULL);
```

Refer to *Section 3: Reference documents* and read about ZbPersistNotifyRegister API.

5.3 Memory mapping and data path

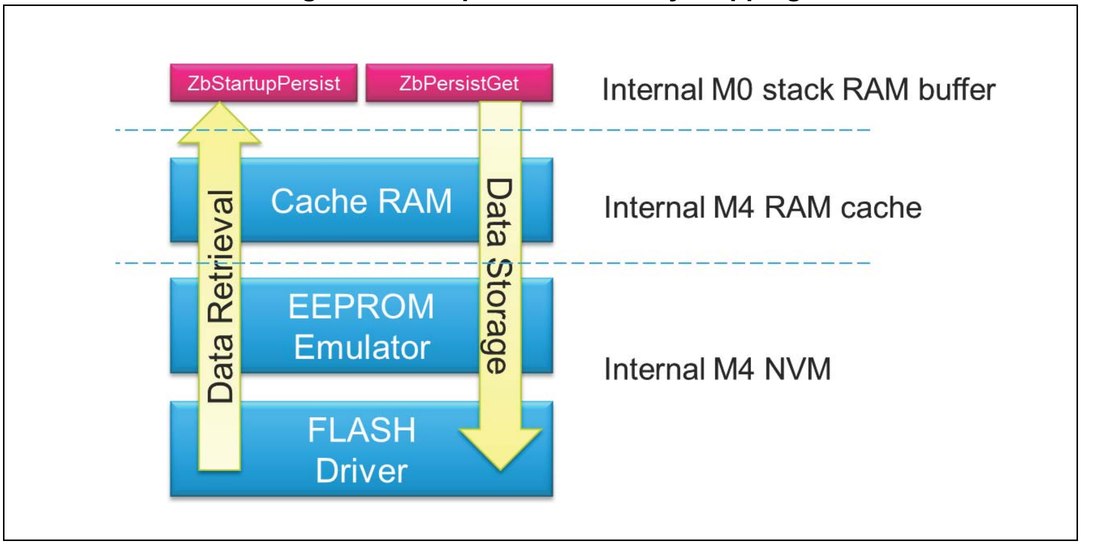
The stack provides the ZbPersistGet API to retrieve data and its length from an internal stack buffer in RAM. Refer to *Section 3: Reference documents* to read about ZbPersistGet API.

The data and length are copied in a RAM cache and through the EEPROM emulator are written in Flash. The revert path is done to read back persistent data.

In the read case the cache feeds with Flash data read after warm start is given to the stack API ZbStartupPersist. Refer to *Section 3: Reference documents* to read about ZbStartupPersist API.

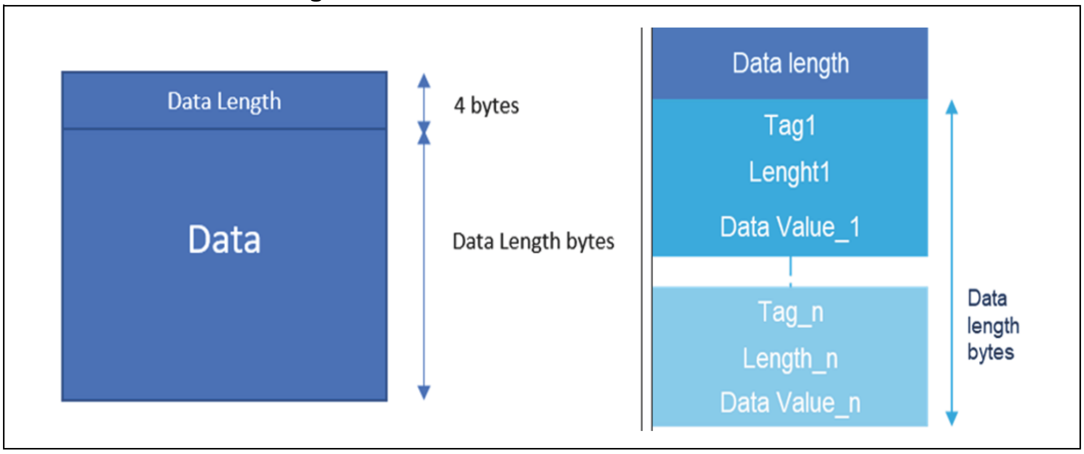
The following figure shows memory mapping and data path.

Figure 1. Data path and memory mapping



The persistent data have the following buffer structure as shown in the figure below. The data persistent data format is stored as TLV structures (tag, length, value) stack and cluster attribute data are stored all together.

Figure 2. Persistent data buffer structure



5.4 Cluster attributes configuration

Whereas the network stack data are always persisted, the cluster attributes are to be told so.

To indicate the cluster attributes to be persisted, this must declared with the flag ZCL_ATTR_FLAG_PERSISTABLE.

Here is an example of on off cluster attribute declared to be in persistent data.

```
/* Attributes */
Static const struct ZbZclAttrT zcl_onoff_server_attr_list[] = {
{
ZCL_ONOFF_ATTR_ONOFF, ZCL_DATATYPE_BOOLEAN,
ZCL_ATTR_FLAG_REPORTABLE|ZCL_ATTR_FLAG_PERSISTABLE,0,NULL,{0,0},{0,0}
},
};
```

6 NVM configuration and sizing

6.1 Introduction

The first question for the end-user is the Flash memory amount needed to save the persistent data. From this number he deducts the maximum persistent data the application can save at once while keeping the NVM management functional.

The STM32WB internal Flash memory unit is the page, a page size is 4 Kbytes and its width is 64 bits.

6.2 Flash memory size allocation (scatter file)

- The M4 code is flashed starting at 0x08000000.
- The Cortex®-M0 code is flashed starting at 0x08080000.

Note: M0 starting address is an example, the real address in a product depends on the M0 firmware size.

Therefore the developer must define in the scatter file the allocated Flash memory dedicated for NVM between this range and must be a multiple of 2 pages.

Below an example of scatter file for IAR that allocates 64 Kbytes of NVM in the M4 internal Flash memory.

Figure 3. Scatter file example

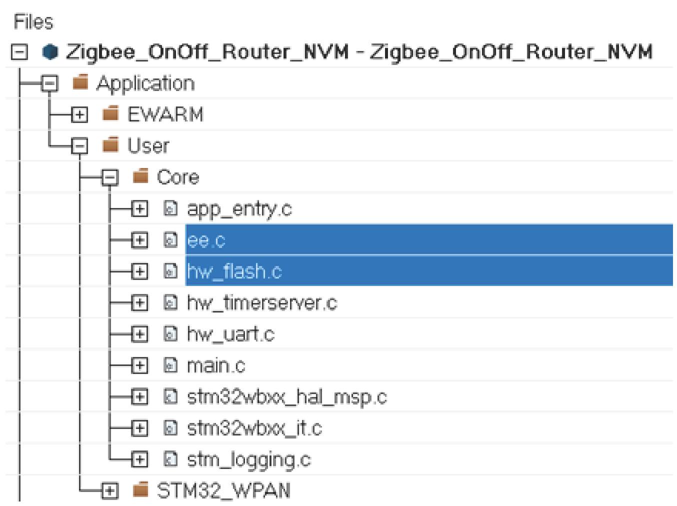
```
define symbol __ICFEDIT_intvec_start__ = 0x08000000;
/*-Memory Regions-*/
/***** FLASH Part dedicated to M4 *****/
define symbol __ICFEDIT_region_ROM_start__ = 0x08000000;
define symbol __ICFEDIT_region_ROM_end__   = 0x0806FFFF; // end of allocated flash for M4 code, 0x08070000 start of the NVM
define symbol __ICFEDIT_region_RAM_start__ = 0x20000004;
define symbol __ICFEDIT_region_RAM_end__   = 0x2002F000;
```

The NVM starts at 0x08070000 to end at 0x08080000 (64 Kbytes,16 pages).

6.3 Files location

The EEPROM emulator (EE) and the Flash driver files are located in the middleware as shown in the below figure.

Figure 4. NVM file location



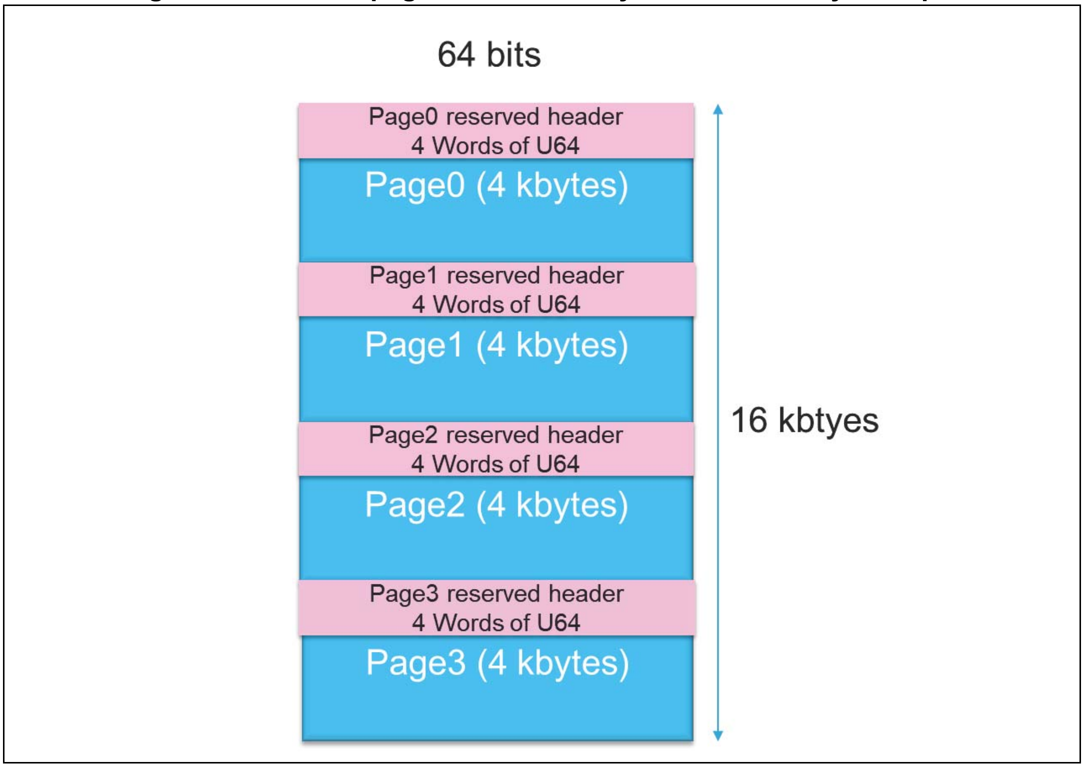
6.4 Persistent data size configuration

6.4.1 EEPROM emulation Flash memory page headers

Basically, the NVM implementation emulate an EEPROM meaning that the total size of the Flash memory defined in the scatter file is not fully available to store persistent data.

First a header of 4 U64 words is taken for the page state handling as shown in the following figure.

Figure 5. Reserved page headers 16-kbyte Flash memory example

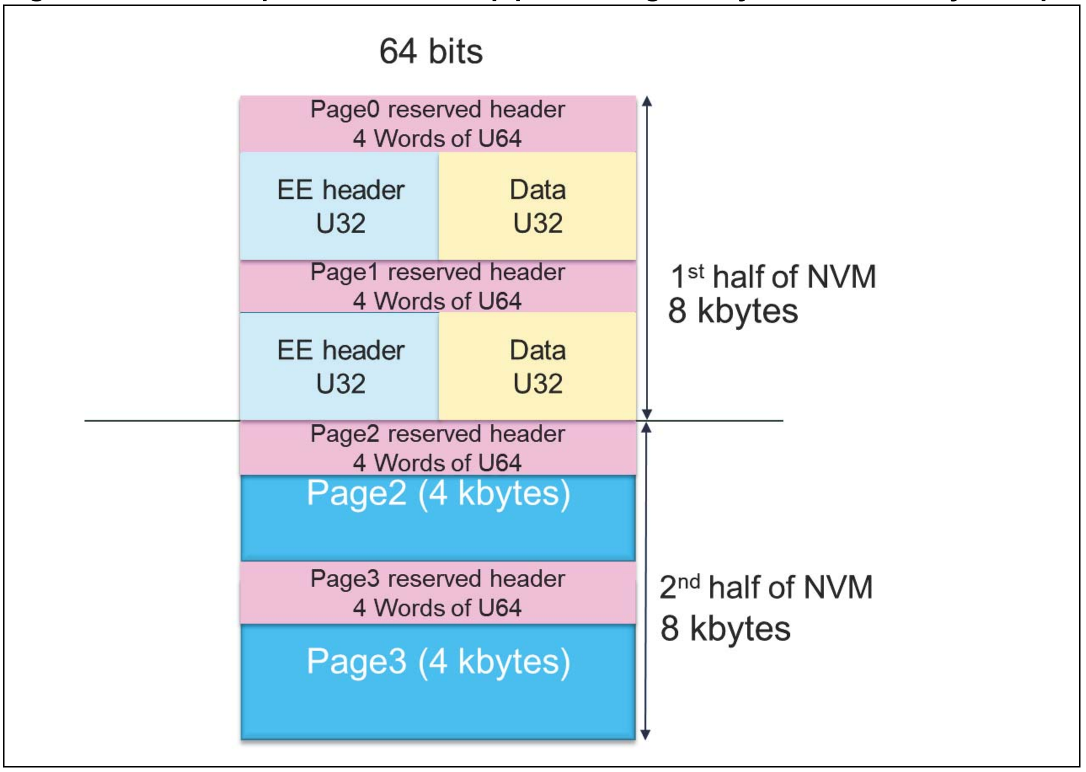


6.4.2 EEPROM emulation Flash memory bank partitioning

In addition, each record requests an EE header of the data (tag, CRC, EEPROM address) that takes a U32 words. The second part of the record is the data sized on a U32 words. As shown in the below figure, half of the page taken for EE headers. 2032 data bytes remains per page. Finally, the Flash memory is partitioned in half.

First half is used to store data while second half of the bank size is reserved to swap data when the limit of the storage is reached. The below figure shows the configuration.

Figure 6. EE header plus data and swap partitioning 16-kbyte Flash memory example

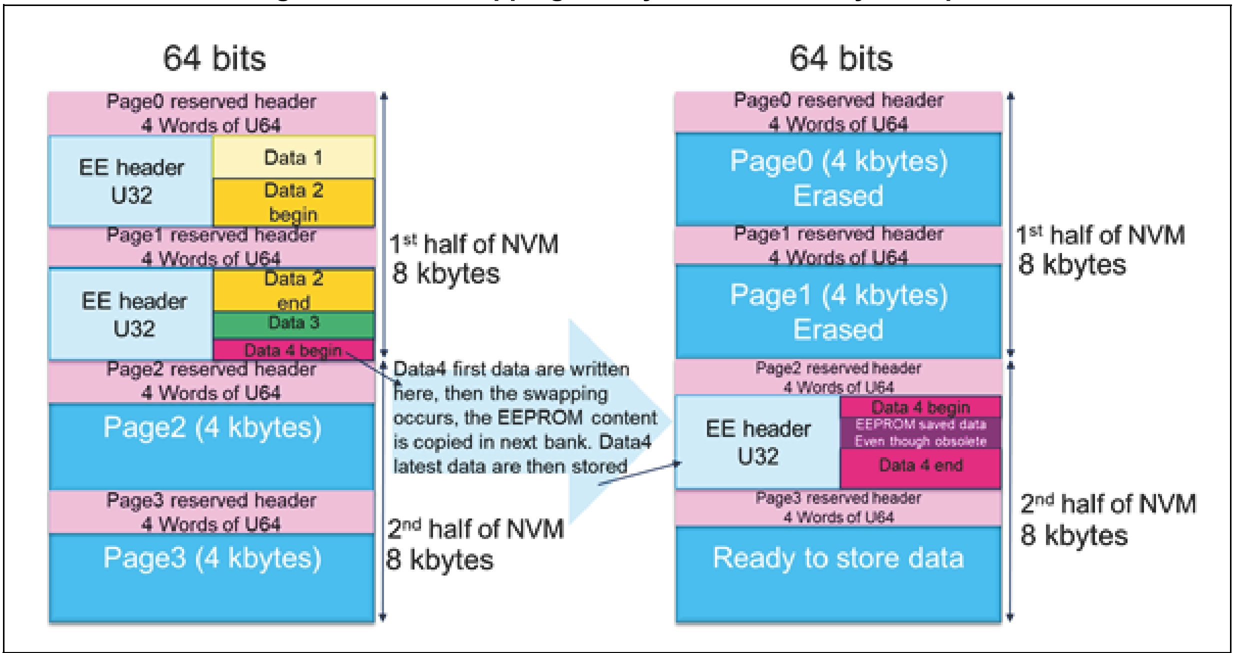


6.4.3 EEPROM emulation Flash memory bank swapping

The swap requesting a minimum of 2 pages by half bank to work. Once the limit of the half bank is reached, current EEPROM data is saved on the beginning of the second half bank and continue to save the new data. The first half of the bank is then erased and ready for next swap turnover. Finally, the Flash memory is partitioned in half.

First half is used to store data while second half of the bank size is reserved to swap data when the limit of the storage is reached. The below figure shows the configuration.

Figure 7. Bank swapping 16-kbyte Flash memory example



6.4.4 EEPROM emulation Flash memory final effective data size

This leads to have the (total Flash memory size/8) * 2032 bytes as a maximum size for a persistent data record. The following table shows an example of maximum data size versus Flash memory size and life cycle.

Note: 2032 bytes is the maximum number of data byte per page.

Table 2. Flash memory size versus persistent data maximum size (unit in Kbytes)

Used Flash size	Emulated EEPROM 10000 cycles	Emulated EEPROM 10000 cycles
8	1.9	0.2
16	3.9	0.4
24	5.9	0.6
32	7.9	0.8
40	9.9	1.0
48	11.9	1.2
56	13.8	1.4

Table 2. Flash memory size versus persistent data maximum size (unit in Kbytes) (continued)

Used Flash size	Emulated EEPROM 10000 cycles	Emulated EEPROM 10000 cycles
64	15.8	1.6
72	17.8	1.8
80	19.8	2.0
88	21.8	2.2
96	23.8	2.4
104	25.7	2.6
112	27.7	2.8
120	29.7	3.3

6.4.5 NVM configuration and sizing in the code

The configuration of the NVM constants is declared in the app_conf.h header file.

This file is in ProjectName/Core/Inc

```
#define CFG_NB_OF_PAGE                  (16U)
#define CFG_EE_BANK0_SIZE               (CFG_NB_OF_PAGE * HW_FLASH_PAGE_SIZE)
#define CFG_NVM_BASE_ADDRESS            (0x70000U)
#define CFG_EE_BANK0_MAX_NB             (1000U)                 // in U32 words
#define ST_PERSIST_MAX_ALLOC_SZ         (4U*CFG_EE_BANK0_MAX_NB) //max data in
                                        bytes
#define ST_PERSIST_FLASH_DATA_OFFSET    (4U)
#define ZIGBEE_DB_START_ADDR            (0U)
#define CFG_EE_AUTO_CLEAN               (1U)
```

CFG_NB_OF_PAGE is the number of pages of NVM (in accordance with scatter file).

CFG_EE_BANK0_SIZE is the total Flash size in bytes of the NVM.

CFG_NVM_BASE_ADDRESS is the base offset of the Flash (in accordance with scatter file) so the starting NVM address in Flash is 0x0870 0000.

ZIGBEE_DB_START_ADDR start of the data base in case more DB are using the NVM.

CFG_EE_AUTO_CLEAN is used for internal cleaning is the EE emulator.

ST_PERSIST_FLASH_DATA_OFFSET is the 4 bytes used by the data length storage.

CFG_EE_BANK0_MAX_NB is the maximum size of data in U32 refer to *Table 2*.

ST_PERSIST_MAX_ALLOC_SZ is the maximum size of data in bytes.

The highlighted value must be configured based on scatter file config and max possible data allowed by the bank size. The EE at compile time checks if those values are consistent at compile time.

The RAM cache memory allocation is declared statically in the app_zigbee.c and relies to the above define ST_PERSIST_MAX_ALLOC_SZ. IT can be access either in U8 or U32.

```
/* NVM variables */
/* cache in uninit RAM to store/retrieve persistent data */
union cache
{
uint8_t U8_data[ST_PERSIST_MAX_ALLOC_SZ];      //in bytes
uint8_t U32_data[ST_PERSIST_MAX_ALLOC_SZ/4U]; // in U32 words
};
__no_init union cache cache_persistent_data;
```

6.4.6 Flash driver implementation

The EE at some point calls Flash memory procedure APIs. The HW_FLASH_Write and HW_FLASH_Erase wrapper APIs implementation are given as example. The developer can integrate and call its own Flash driver from the wrapper.

7 NVM initialization

7.1 Introduction

For his application the end-user can choose either the NVM (EE plus Flash memory) or stick with the RAM cache only.

Using only the cache leads to lose persistent data in case of power off. Doing a SW reset keeps the cache data since it is defined in the unit zone though.

Note: This capability to not use the Flash memory prevents the end-user from burning the life cycle of his Flash memory during test or debug phase.

7.2 NVM usage initialization

- To use the Flash declares the following define:

```
#define CFG_NVM (1U) /* Use NVM */
```

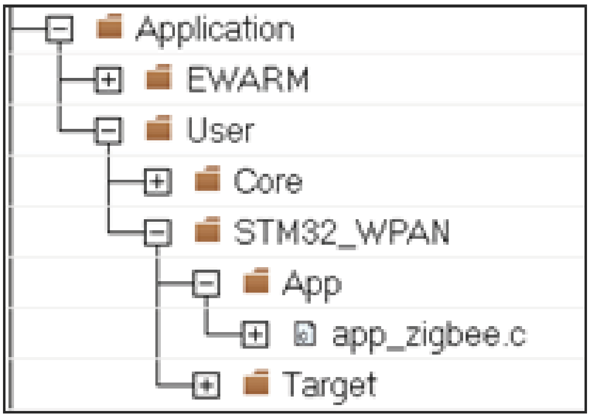
- To use the RAM cache only skip this code line.

Note: In both cases the cache is always used.

7.3 EE initialization

The whole NVM implementation is in the app_zigbee.c files located as shown in the below figure.

Figure 8. Application file location



The EE must be initialized at each cold or warm start.

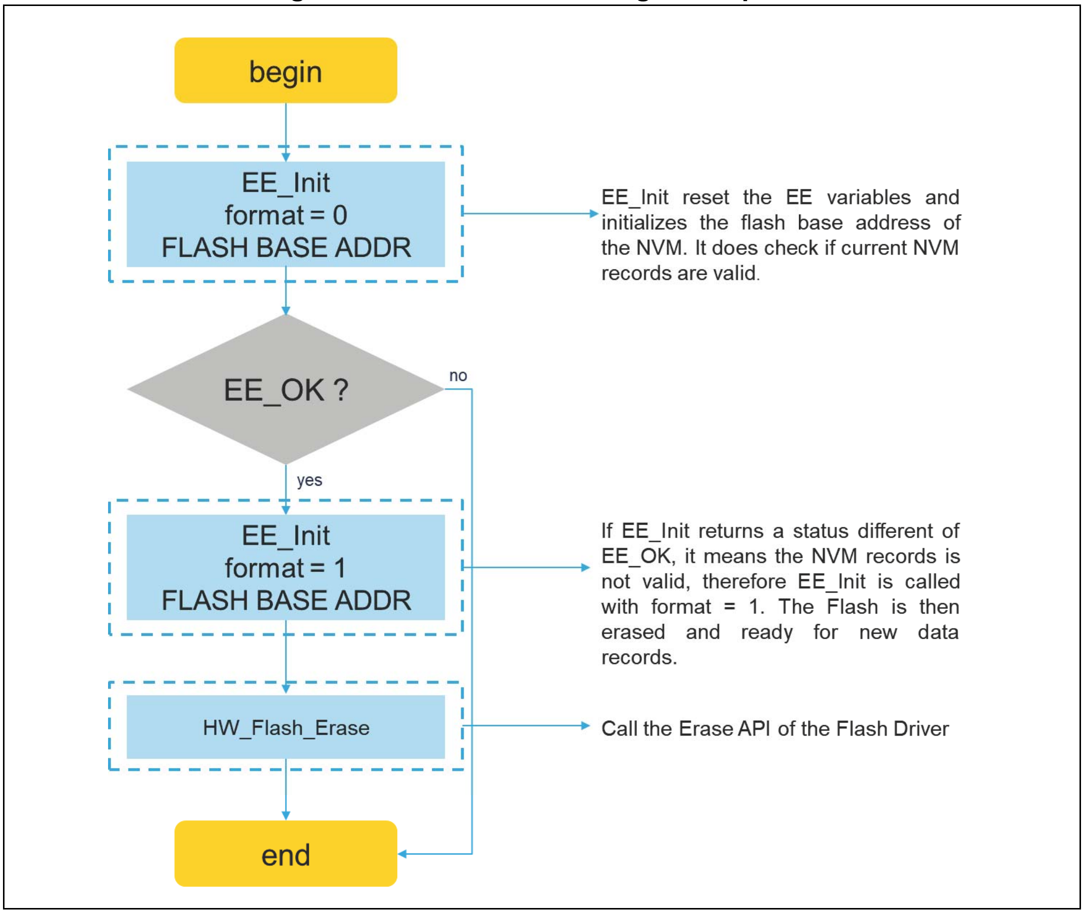
The APP_ZIGBEE_NVM_Init API is therefore called in the APP_ZIGBEE_Init API.

```
void APP_ZIGBEE_Init(void)
{
       SHCI_CmdStatus_t ZigbeeInitStatus;
       APP_DBG("APP_ZIGBEE_Init");

       /* Check the compatibility with the Coprocessor Wireless Firmware
loaded */
       APP_ZIGBEE_CheckWirelessFirmwareInfo();

       /* Register cmdbuffer */
       APP_ZIGBEE_RegisterCmdBuffer(&ZigbeeOtCmdBuffer);

       /* Init config buffer and call TL_ZIGBEE_Init */
       APP_ZIGBEE_TL_INIT();

       /* NVM Init */
   #if CFG_NVM
       APP_ZIGBEE_NVM_Init();
   #endif
}
```

7.4 Algorithm diagram

The following figure shows the algorithm diagram of the (NVM) initialization phase.

Figure 9. NVM initialization algorithm phase



8 NVM data storage algorithm

The following figures show the algorithm of the NVM data storage phase.

Figure 10. NVM data storage algorithm diagram (phase 1)

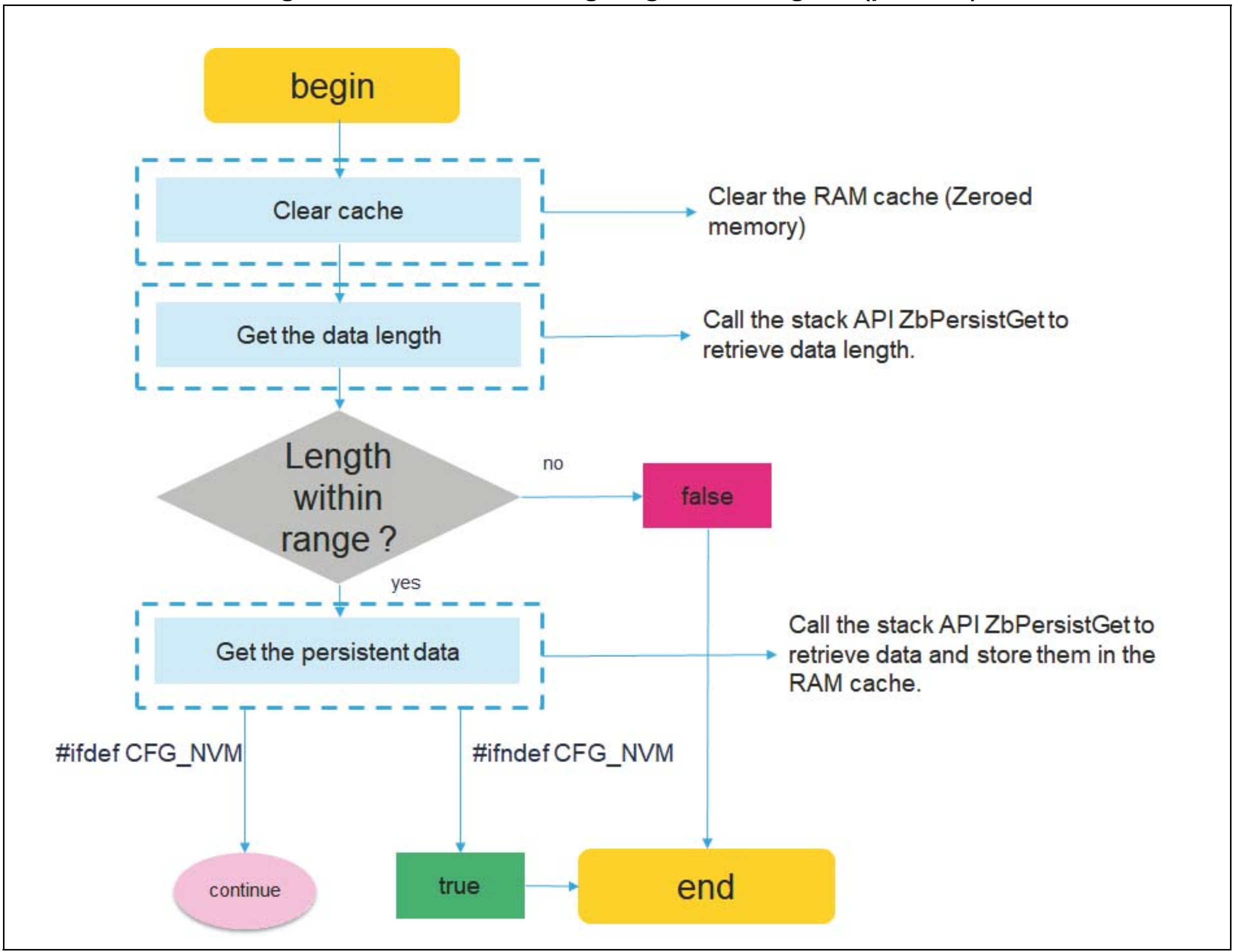
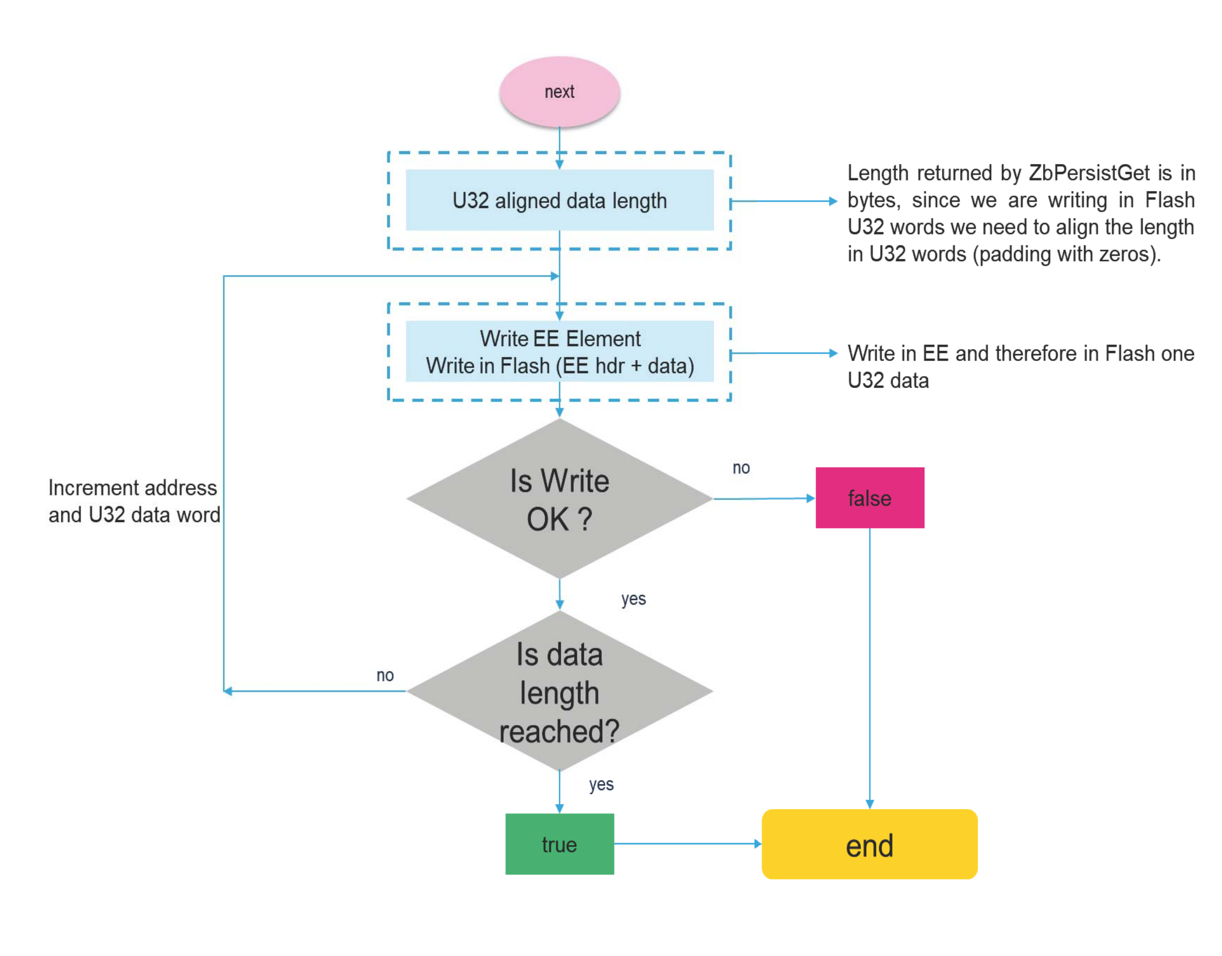


Figure 11. NVM data storage algorithm diagram (phase 2)



9 NVM data retrieval algorithm

The following figures show the algorithm of the NVM data retrieval phase.

Figure 12. NVM data retrieval algorithm diagram (phase 1)

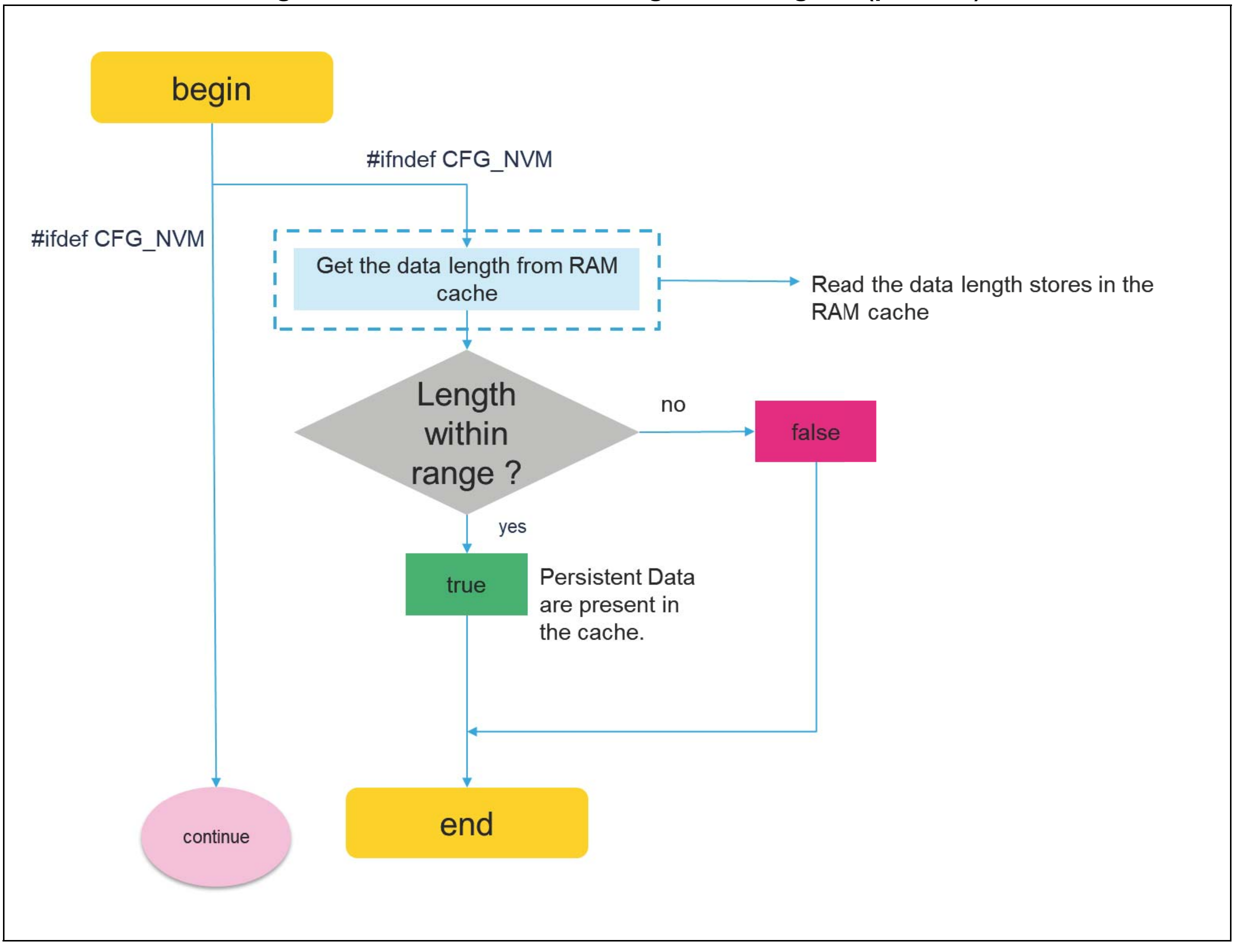
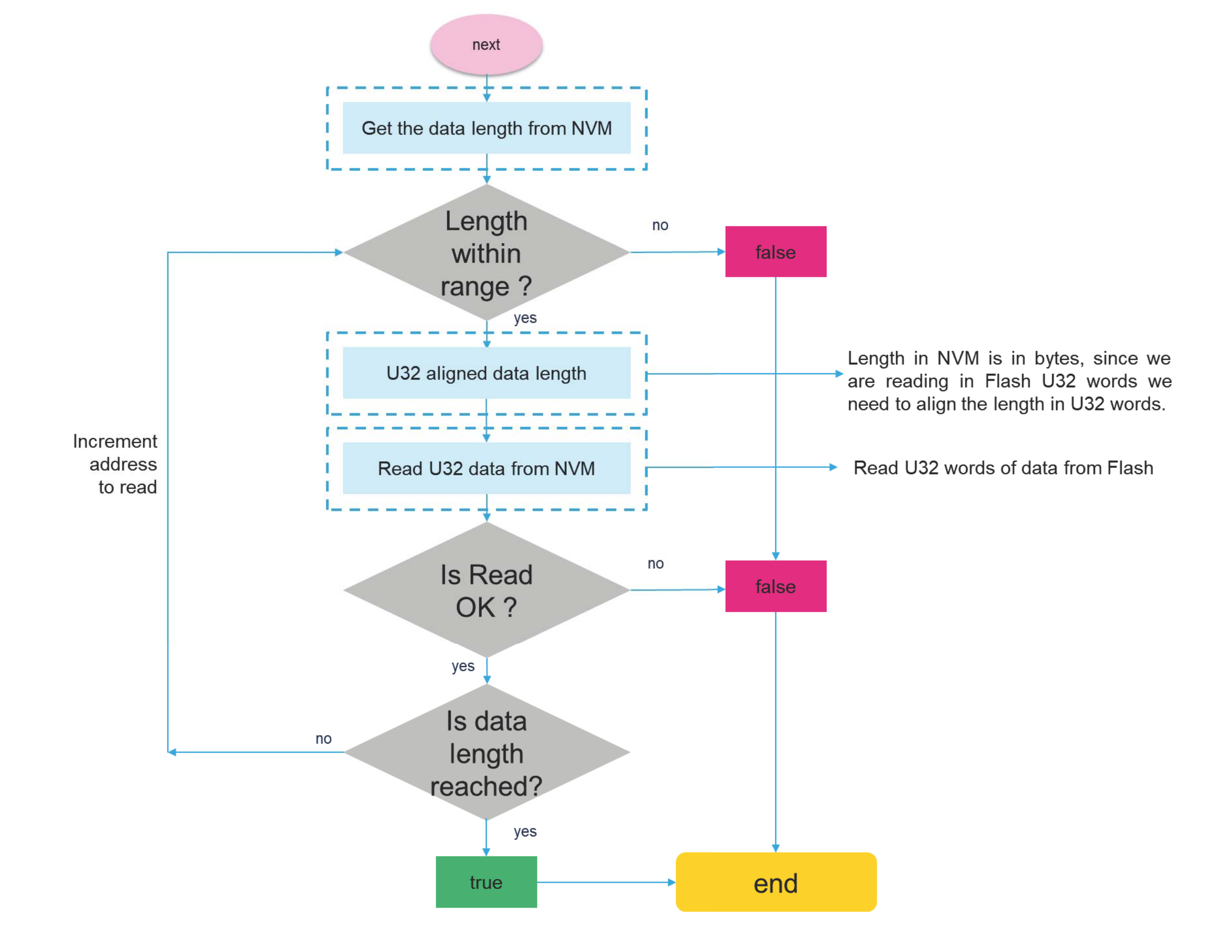


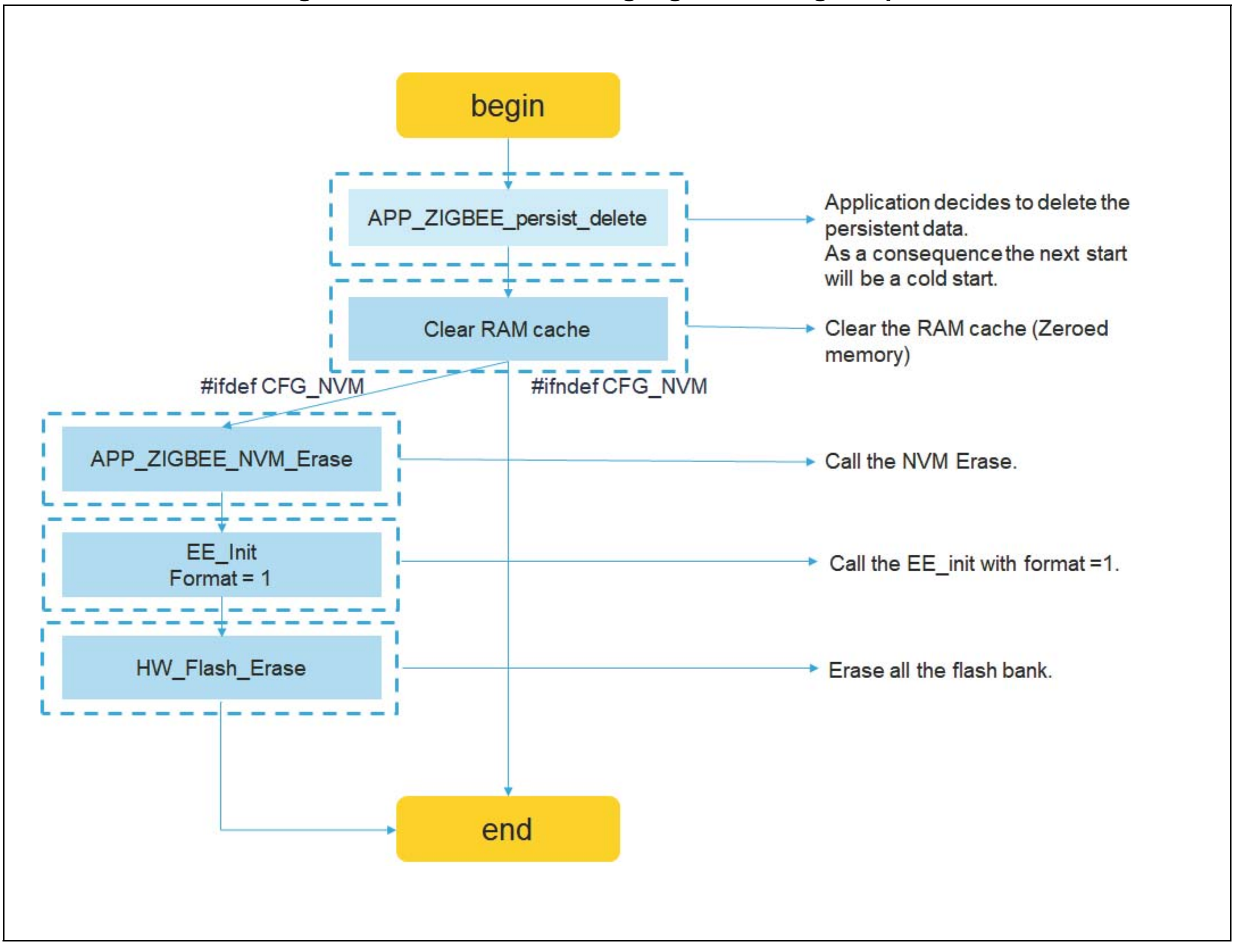
Figure 13. NVM data retrieval algorithm diagram (phase 2)



10 NVM clearing algorithm

The following figure shows the algorithm of the NVM data clearing phase.

Figure 14. NVM data clearing algorithm diagram phase



11 API description

The below table describes the persistent data and NVM APIs.

Table 3. Persistent data and NVM APIs description

API	Description
Persistent data application APIs	
APP_ZIGBEE_peristent_save	Starts the persistent data saving procedure.
APP_ZIGBEE_peristent_load	Starts the persistent data loading procedure.
APP_ZIGBEE_peristent_delete	Starts the persistent data clearing procedure.
NVM application APIs	
APP_ZIGBEE_NVM_Init	Initializes the NVM.
APP_ZIGBEE_NVM_Write	Starts to write the persistent data to the NVM form cache.
APP_ZIGBEE_NVM_Read	Starts to read the persistent data from the NVM and load the cache.
APP_ZIGBEE_NVM_Erase	Starts to erase the NVM and clear the cache.
EE APIs	
EE_Init	Initializes the EE. If format variable is set the NVM Erase is triggered.
EE_WriteEl	Writes the persistent data to NVM adding EE header.
EE_ReadEl	Read the persistent data from NVM stripping EE header off.
Flash driver wrapper APIs	
HW_FLASH_Write	Calls the Flash driver write procedure.
HW_FLASH_Erase	Calls the Flash diver erase procedure.

12 Message sequence charts

The message sequence charts (MSC) shown in the following figures in this section are to be considered with CFG_NVM defined.

Legend for all figures represents:

- Pink: memory
- Light blue: modules
- Dark blue: API

Figure 15. NVM data retrieval procedure MSC

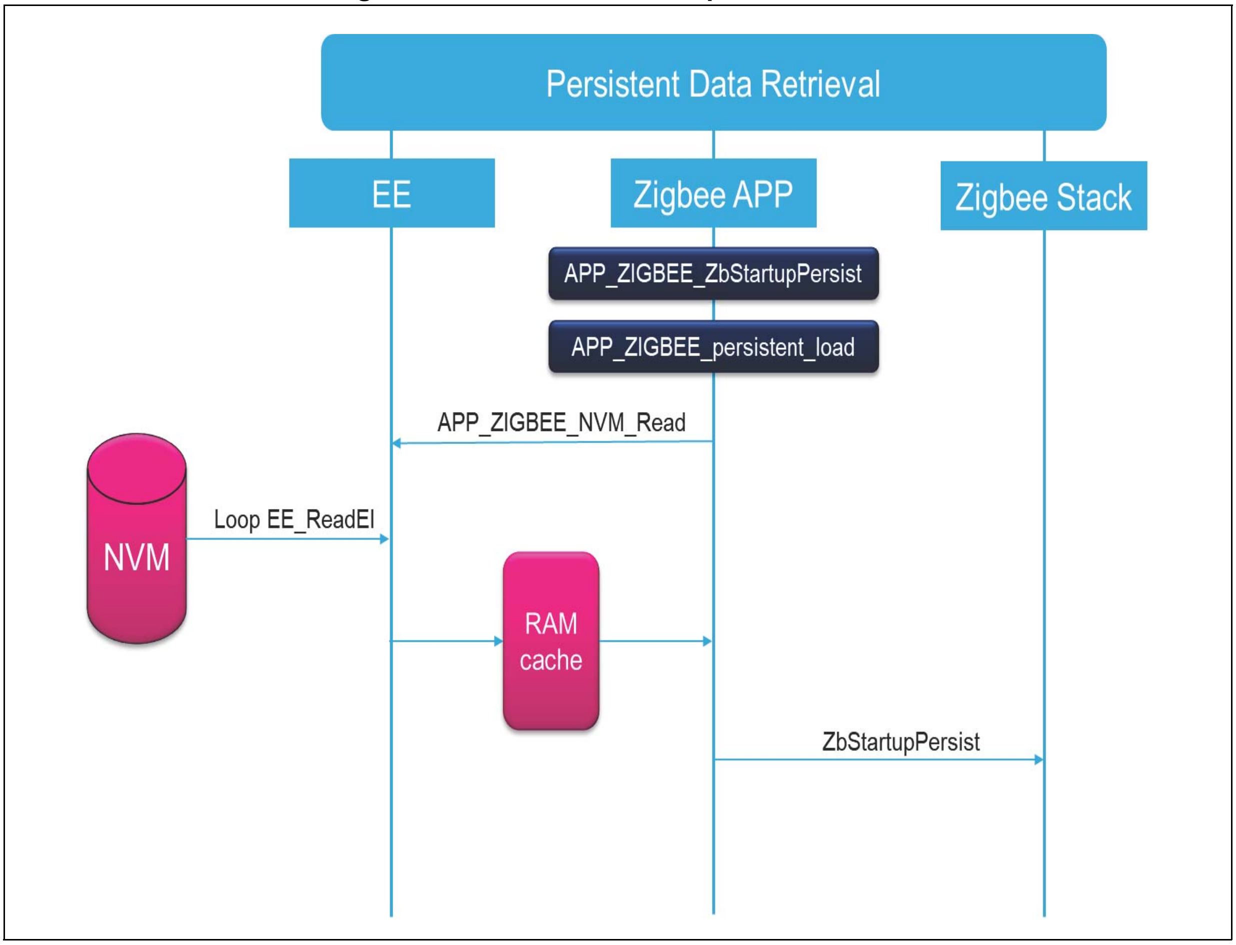


Figure 16. NVM data storage procedure MSC

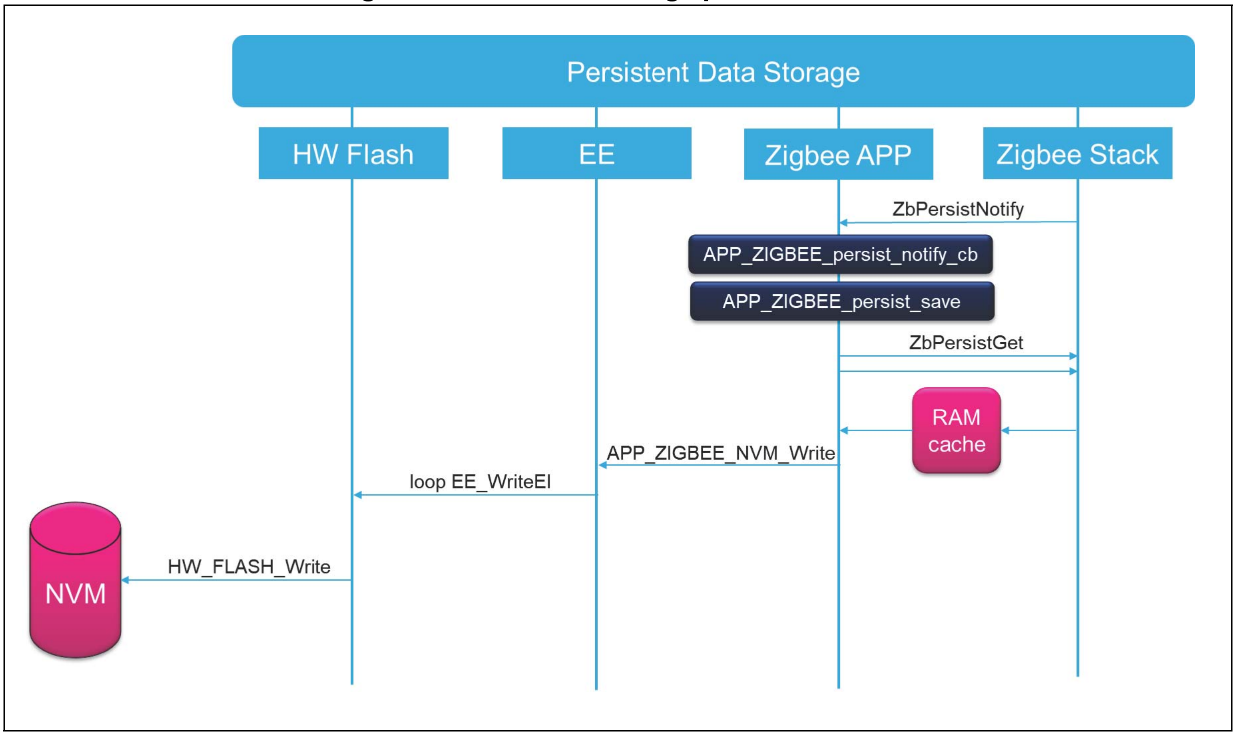


Figure 17. NVM data clearing procedure MSC

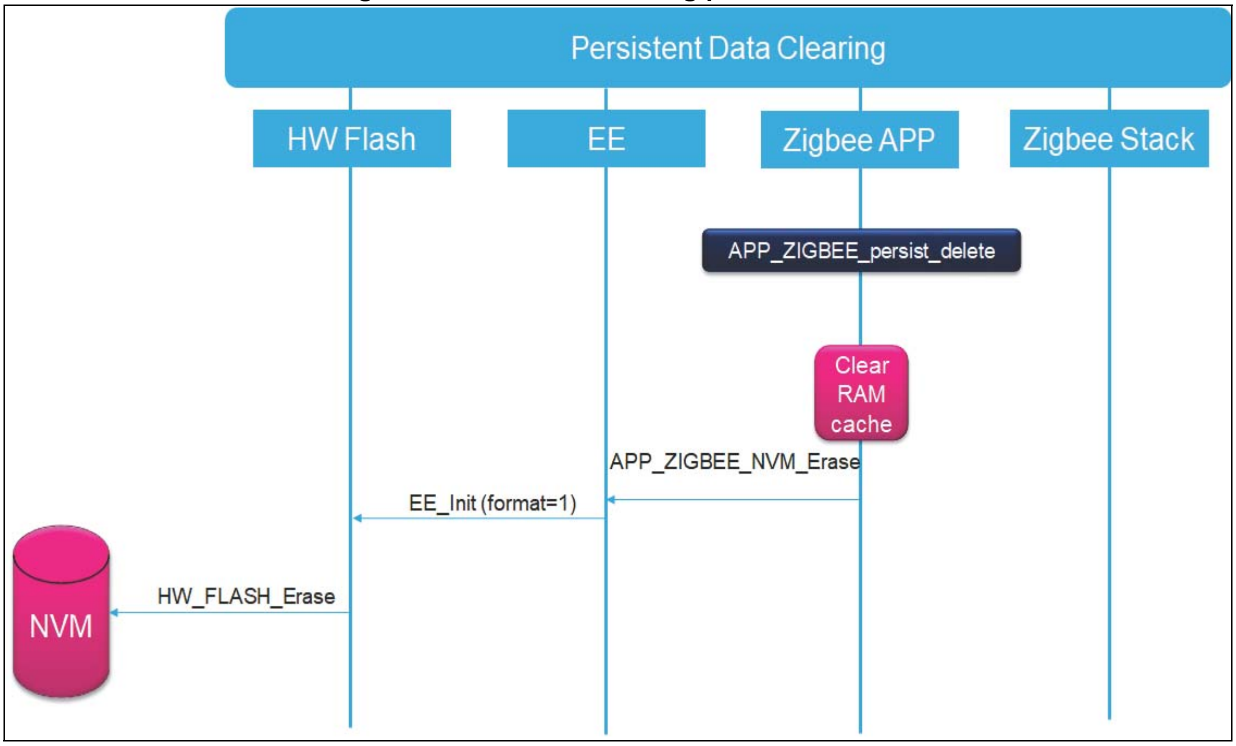


Figure 18. NVM cold start MSC

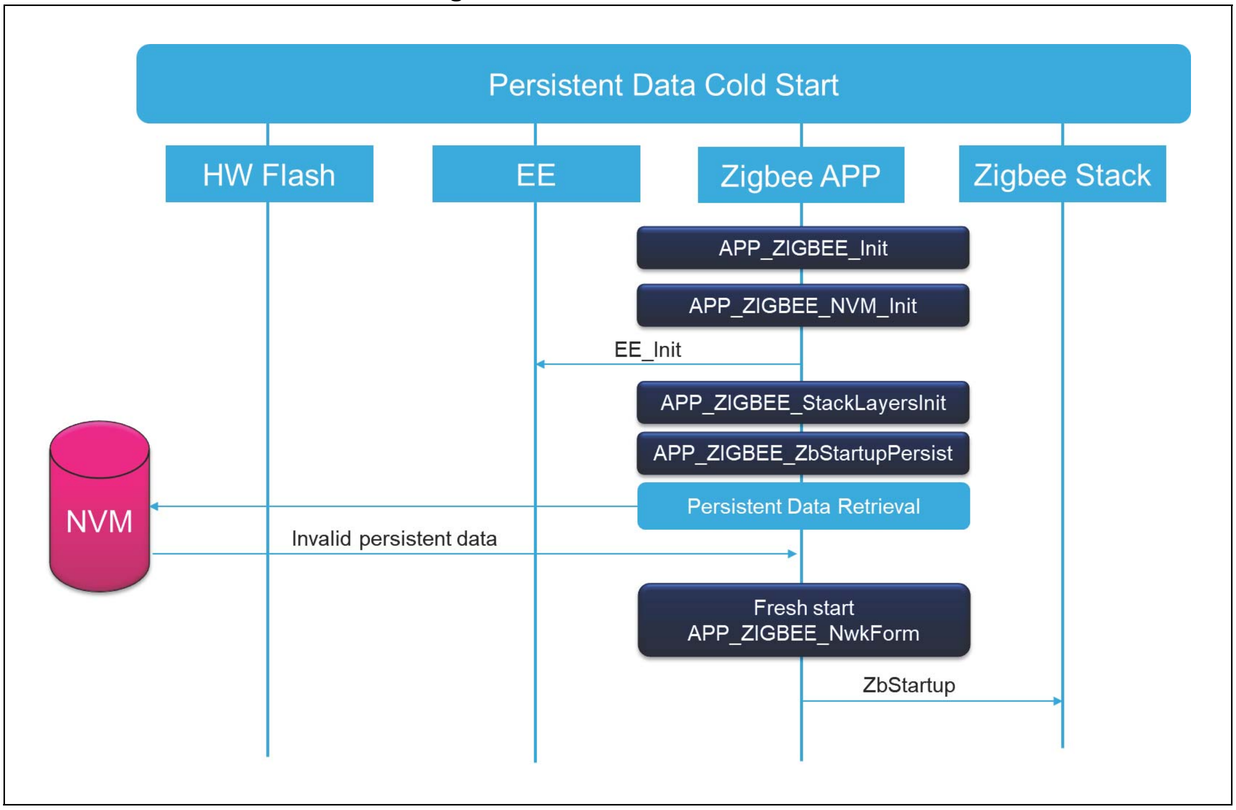


Figure 19. NVM warm start success MSC

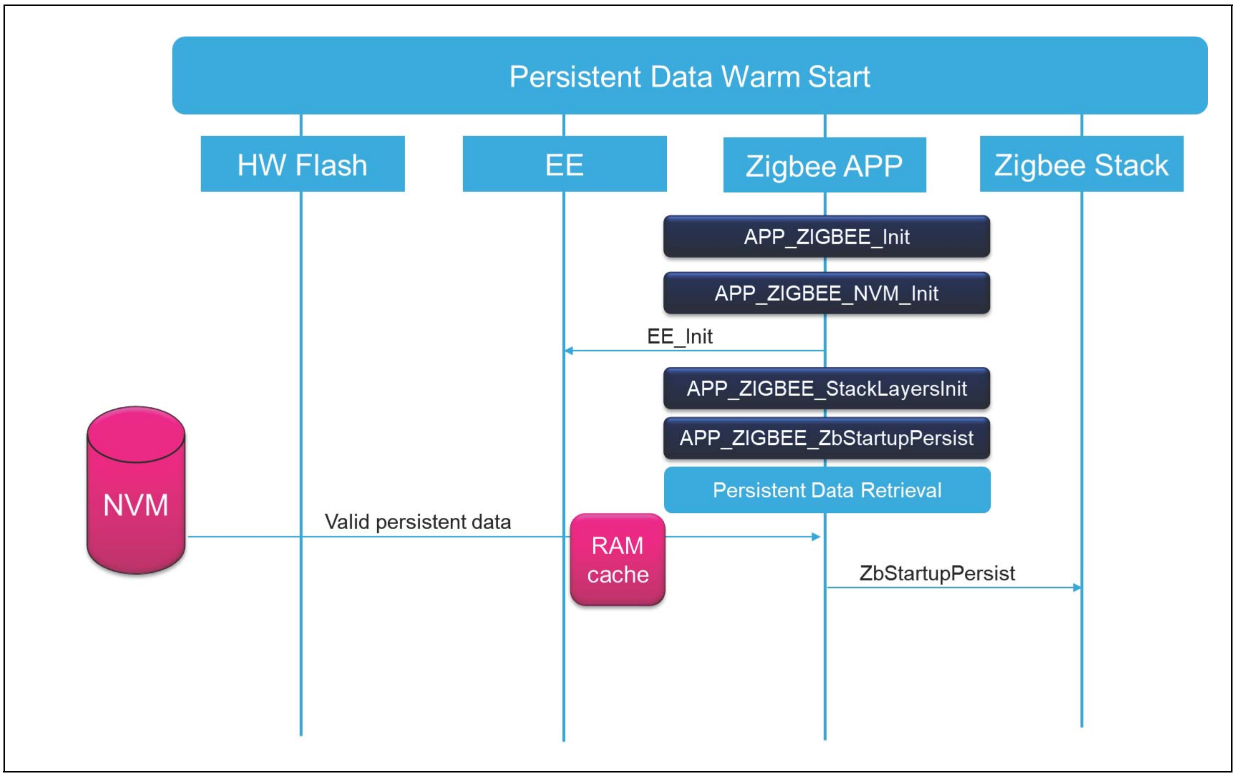
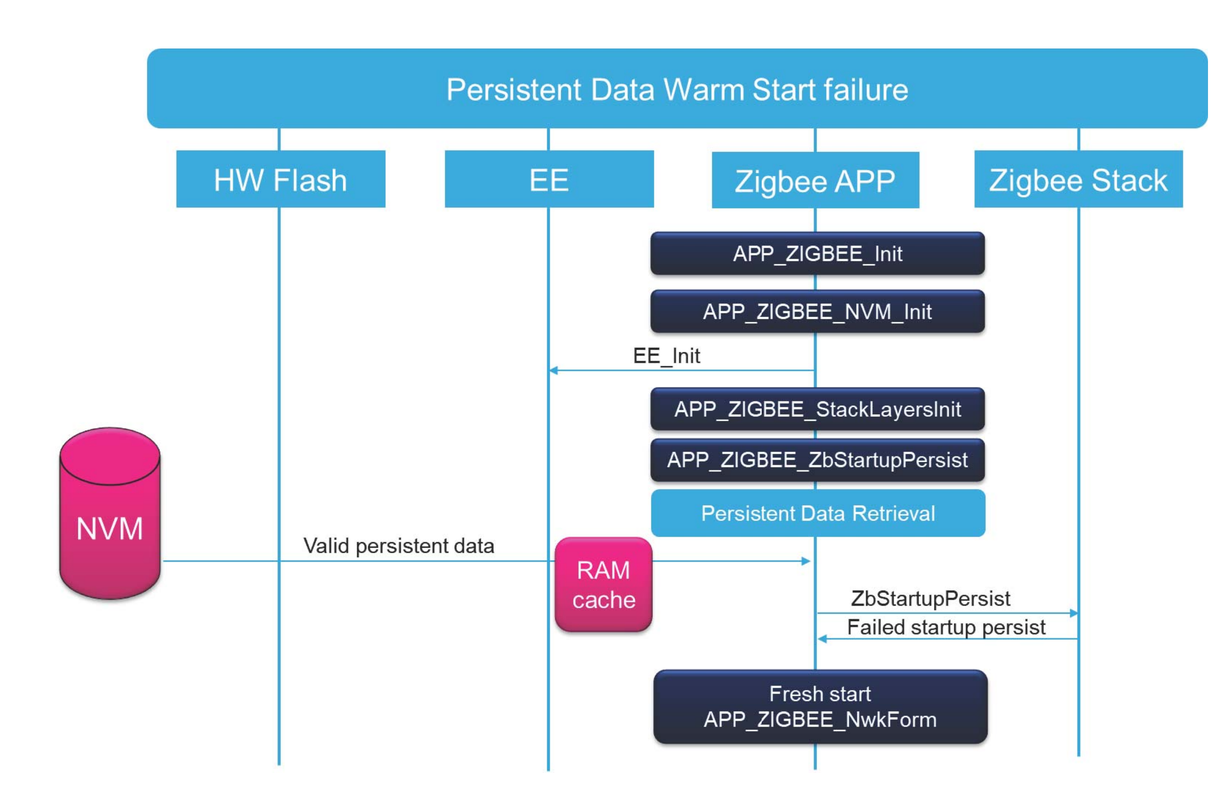


Figure 20. NVM warm start failure MSC



Note: APP_ZIGBEE_Init must configure endpoint and cluster server before a startup from persist.

13 Persistent data application example

13.1 Example project presentation

This example is based on the on off cluster. Both boards can toggle the red LED of the other thanks to on off cluster, the router when starting from persist restore the red LED latest state before the power off demonstrating the on off attribute is well saved and load in NVM.

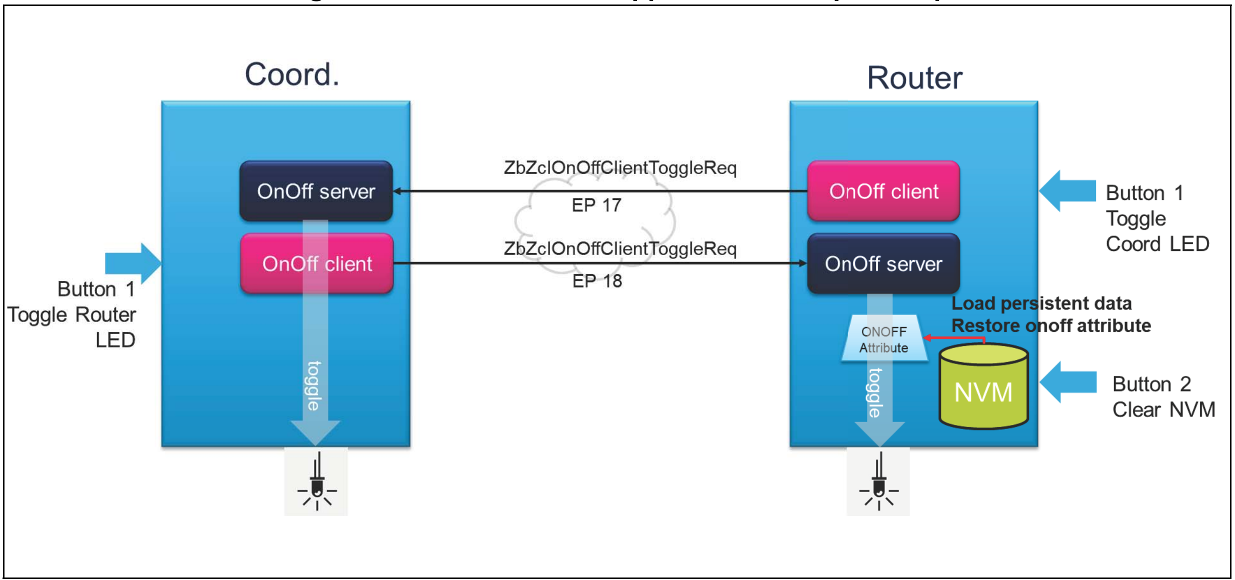
There are two projects, one for a coordinator and one for the router:

- Coordinator project binaries
  - Located in firmware/project/zigbee//Zigbee_OnOff_Coord_NVM
  - One STM32WB55xx board loaded with:
    a) Wireless coprocessor:M0 stm32wb5x_Zigbee_FFD_fw.bin
    b) Application M4: Zigbee_OnOff_Coord_NVM
- Router project binaries
  - Located in firmware/project/zigbee//Zigbee_OnOff_Router_NVM
  - One or more STM32WB55xx board loaded with:
    a) Wireless coprocessor: stm32wb5x_Zigbee_FFD_fw.bin
    b) Application: Zigbee_OnOff_Router_NVM

13.2 Setup

The below figure gives the example setup.

Figure 21. Persistent data application example setup



13.3 Testing description

Cold start:

1. Start the coordinator → blue LED must light on.
2. Start the router, no persistent data stored, so normal network form is performed → Blue LED should light on. The network is up and ready.
3. Push the coordinator button1 → the red LED of the router should toggle. After few seconds the stack should notify the router and persistent data must be saved in NVM.
4. Push the router button1 → the red LED of the coordinator should toggle.

Warm start:

1. Power off the router.
2. Power on the router → NVM load the persistent data and start from persist → green LED should light on.
3. The router red LED must be restored to the state it has before the power off.
4. On off cluster should still work toggling both board red LED when pushing the button1.

Factory setting reinitialization:

1. Push the button2 of the router board to erase the Flash memory.
2. Cold start is performed the next time both boards are plugged on.

14 Conclusion

The persistent data management ZigBee® and non-volatile memory in the STM32WB Series provide an implementation to save or load persistent data to/from a Flash memory NVM.

The emulation of an EEPROM is used to save the Flash memory life cycle and offers the opportunity to go with an external EEPROM only (small persistent data). The code provided in this application note is given as an example only as well as examples of application.

The persistent data size depends of the network size and number of clusters used. Refer to *Section 15: Annexes*.

The integrity of the NVM is not guaranteed if a power off is performed during a write to the Flash memory. To guarantee the data integrity an external hardware setup is needed.

15 Annexes

This section covers three aspects of the ZigBee stack use of persistent memory:

- Impact:
  - What is the number and size of attributes that need to be persisted.
- Footprint:
  - How much memory is required to store the persistent attributes on an active network.
- Frequency:
  - How often is the state of these attributes to be saved to memory.

Information regarding these three areas is described in *Section 15.1: Impact* and *Section 15.2: Footprint and frequency*.

15.1 Impact

15.1.1 Persisted items

This section contains the details of each persisted item. The description for each entry is formatted as shown in *Table 4*.

Table 4. Entry description format

Name	ID	Format	Length
The designated name for each item	The identifier for each item	The format of the data stored in the item	The number of bytes needed to store the current state of the item

Table 5. BDB items

Name	ID	Format	Length
ZB_BDB_CommissioningMode	0x1001	*ZB_PERSIST_TYPE_UINT8*	1
ZB_BDB_PrimaryChannelSet	0x1008	*ZB_PERSIST_TYPE_UINT32*	4
ZB_BDB_ScanDuration	0x1009	*ZB_PERSIST_TYPE_UINT8*	1
ZB_BDB_SecondaryChannelSet	0x100a	*ZB_PERSIST_TYPE_UINT32*	4
ZB_BDB_TCLK_ExchangeAttemptsMax	0x100c	*ZB_PERSIST_TYPE_UINT8*	1
ZB_BDB_TCLinkKeyExchangeMethod	0x100d	*ZB_PERSIST_TYPE_UINT8*	1
ZB_BDB_TrustCenterNodeJoinTimeout	0x100e	*ZB_PERSIST_TYPE_UINT8*	1
ZB_BDB_TrustCenterRequiresKeyExchange	0x100f	*ZB_PERSIST_TYPE_UINT8*	1
ZB_BDB_AcceptNewUnsolicitedTCLinkKey	0x1010	*ZB_PERSIST_TYPE_UINT8*	1
ZB_BDB_AcceptNewUnsolicitedApplicationLinkKey	0x1011	*ZB_PERSIST_TYPE_UINT8*	1
ZB_BDB_TLRssiMin	0x1107	*ZB_PERSIST_TYPE_INT8*	1
ZB_BDB_UpdateDeviceKeyId	0x1109	*ZB_PERSIST_TYPE_UINT8*	1

Table 5. BDB items (continued)

Name	ID	Format	Length
ZB_BDB_JoinScanType	0x110a	*ZB_PERSIST_TYPE_UINT8*	1
ZB_BDB_JoinIgnoreLqi,	0x110b	*ZB_PERSIST_TYPE_BOOL*	1
ZB_BDB_NlmeSyncFailNumBeforeError	0x110c	*ZB_PERSIST_TYPE_UINT8*	1
ZB_BDB_ZdoTimeout	0x110d	*ZB_PERSIST_TYPE_UINT32*	4
ZB_BDB_TLStealFlags	0x110e	*ZB_PERSIST_TYPE_UINT8*	1
ZB_BDB_JoinTclkNodeDescReqDelay	0x110f	*ZB_PERSIST_TYPE_UINT16*	2
ZB_BDB_JoinTclkRequestKeyDelay	0x1110	*ZB_PERSIST_TYPE_UINT16*	2
ZB_BDB_TLDenyFactoryNew	0x1111	*ZB_PERSIST_TYPE_UINT8*	1
ZB_BDB_TLKeyIndex	0x1113	*ZB_PERSIST_TYPE_UINT8*	1
ZB_BDB_ZdoPermitJoinAfterJoin	0x1114	*ZB_PERSIST_TYPE_BOOL*	1
ZB_BDB_ZdoZigbeeProtocolRevision	0x1115	*ZB_PERSIST_TYPE_UINT8*	1
ZB_BDB_NwkAllowRouterLeaveRejoin	0x1116	*ZB_PERSIST_TYPE_BOOL*	1
ZB_BDB_PersistTimeoutMs	0x1117	*ZB_PERSIST_TYPE_UINT32*	4
ZB_BDB_JoinAttemptsMax	0x1118	*ZB_PERSIST_TYPE_UINT8*	1
ZB_BDB_MaxConcurrentJoiners	0x1119	*ZB_PERSIST_TYPE_UINT8*	1
ZB_BDB_DisablePersistRejoin	0x111a	*ZB_PERSIST_TYPE_BOOL*	1
ZB_BDB_TLKey	0x1112	*ZB_SEC_KEYSIZE*	16

Table 6. APS items

Name	ID	Format	Length
ZB_APS_IB_ID_USE_INSECURE_JOIN	0x00c8	*ZB_PERSIST_TYPE_UINT8*	1
ZB_APS_IB_ID_SCAN_COUNT	0x0500	*ZB_PERSIST_TYPE_UINT8*	1
ZB_APS_IB_ID_LEAVE_REMOVE_CHILDREN	0x0501	*ZB_PERSIST_TYPE_UINT8*	1
ZB_APS_IB_ID_TRUST_CENTER_ADDRESS	0x00ab	*ZB_PERSIST_TYPE_UINT64*	8

ZB_APS_IB_ID_CHANNEL_MASK	0x00c3	*1 channel mask (9 bytes), of format:*
		`struct ZbChannelListT {`
		`/* Number of channel masks in list */`
		`uint8_t count;`
		`struct {`
		`/* 802.15.4 Channel Page */`
		`uint8_t page;`
		`uint32_t channelMask;`
		`}`
		`list[MAX_CHANNEL_LIST_ENTRIES];`
		`};`
			9

ZB_APS_IB_ID_PRECONFIGURED_LINK_KEY	0x0502	*ZB_SEC_KEYSIZE*	16

Table 6. APS items (continued)

Name	ID	Format	Length
ZB_APS_IB_ID_DISTRIBUTED_GLOBAL_KEY	0x0503	*ZB_SEC_KEYSIZE*	16
ZB_APS_IB_ID_BINDING_TABLE	0x00c1	Binding Table Value Format: *uint8_t numEntries;* *{* *uint64_t srcAddr;* *uint8_t srcEndpt;* *uint16_t clusterId;* *uint8_t dstAddrMode;* *uint64_t dstExtAddr;* *uint16_t dstGroupAddr;* *uint8_t dstEndpt;* *} entry[numEntries];*	23 bytes per entry
ZB_APS_IB_ID_GROUP_TABLE	0x00c5	Group Table Value Format: *uint8_t numEntries;* *{* *uint16_t groupAddr;* *uint8_t endpoint;* *} entry[numEntries];*	3 bytes per entry
ZB_APS_IB_ID_DEVICE_KEY_PAIR_SET	0x00aa	Link Keys Value Format: uint8_t numEntries; *{* *uint64_t deviceAddress;* *uint8_t linkKey[16];* *uint32_t outCounter;* *uint32_t incomingFrameCounter;* *} entry[numEntries];*	33 bytes per entry

Table 7. NWK items

Name	ID	Format	Length
ZB_NWK_NIB_ID_IsConcentrator	0x009d	*ZB_PERSIST_TYPE_UINT8*	1
ZB_NWK_NIB_ID_ConcentratorRadius	0x009e	*ZB_PERSIST_TYPE_UINT8*	1
ZB_NWK_NIB_ID_ConcentratorDiscoveryTime	0x009f	*ZB_PERSIST_TYPE_UINT8*	1
ZB_NWK_NIB_ID_LinkStatusPeriod	0x00a6	*ZB_PERSIST_TYPE_UINT8*	1
ZB_NWK_NIB_ID_MaxChildren	0x0084	*ZB_PERSIST_TYPE_UINT8*	1
ZB_NWK_NIB_ID_MaxSourceRoute	0x0093	*ZB_PERSIST_TYPE_UINT8*	1
ZB_NWK_NIB_ID_ExtendedPanId	0x009a	*ZB_PERSIST_TYPE_UINT64*	8
ZB_NWK_NIB_ID_StackProfile	0x0097	*ZB_PERSIST_TYPE_UINT8*	1
ZB_NWK_NIB_ID_PassiveAckTimeout	0x0082	*ZB_PERSIST_TYPE_UINT16*	2
ZB_NWK_NIB_ID_MaxBroadcastRetries	0x0083	*ZB_PERSIST_TYPE_UINT8*	1
ZB_NWK_NIB_ID_MaxDepth	0x0085	*ZB_PERSIST_TYPE_UINT8*	1
ZB_NWK_NIB_ID_TransactionPersistenceTime	0x0095	*ZB_PERSIST_TYPE_UINT16*	2
ZB_NWK_NIB_ID_RouterAgeLimit	0x00a7	*ZB_PERSIST_TYPE_UINT8*	1
ZB_NWK_NIB_ID_SecurityLevel	0x00a0	*ZB_PERSIST_TYPE_UINT8*	1
ZB_NWK_NIB_ID_NetworkAddress	0x0096	*ZB_PERSIST_TYPE_UINT16*	2
ZB_NWK_NIB_ID_PanId	0x0080	`ZB_PERSIST_TYPE_UINT16`	2
ZB_NWK_NIB_ID_Depth	0x0400	`ZB_PERSIST_TYPE_UINT8`	1
ZB_NWK_NIB_ID_CapabilityInformation	0x008f	`ZB_PERSIST_TYPE_UINT8`	1
ZB_NWK_NIB_ID_UpdateId	0x0094	`ZB_PERSIST_TYPE_UINT8`	1
ZB_NWK_NIB_ID_ParentInformation	0x0409	`ZB_PERSIST_TYPE_UINT16`	2
ZB_NWK_NIB_ID_EndDeviceTimeoutDefault	0x040a	`ZB_PERSIST_TYPE_UINT8`	1
ZB_NWK_NIB_ID_TxPowerMgmtSupported	0x040e	`ZB_PERSIST_TYPE_UINT8`	1
ZB_NWK_NIB_ID_LinkPowerDeltaPeriod	0x040f	`ZB_PERSIST_TYPE_UINT16`	2
ZB_NWK_NIB_ID_JoiningListUpdateId	0x0410	`ZB_PERSIST_TYPE_UINT8`	1
ZB_NWK_NIB_ID_JoiningPolicy	0x0411	`ZB_PERSIST_TYPE_UINT8`	1
ZB_NWK_NIB_ID_JoiningListTotal	0x0412	`ZB_PERSIST_TYPE_UINT8`	1

ZB_NWK_NIB_ID_AddressMap	0x00a9	Address Map Table Value Format:
		`uint16_t numEntries;`
		`{`
		`    uint64_t extAddr;`
		`    uint16_t nwkAddr;`
		`} entry[numEntries];`
			10 bytes per entry

Table 7. NWK items (continued)

Name	ID	Format	Length

ZB_NWK_NIB_ID_NeighborTable	0x0087	Neighbor Table Value Format:
		`uint8_t numEntries;`
		`{`
		`    uint8_t relationship;`
		`    uint64_t extAddr;`
		`    uint16_t nwkAddr;`
		`    uint8_t capability;`
		`} entry[numEntries];`
			12 bytes per entry

ZB_NWK_NIB_ID_RouteTable	0x008b	Routing Table Value Format:
		`uint8_t numEntries;`
		`{`
		`    uint8_t noCache;`
		`    uint8_t isManyToOne;`
		`    uint8_t isMcast;`
		`    uint16_t destAddr;`
		`    uint16_t nextAddr;`
		`} entry[numEntries];`
			7 bytes per entry

ZB_NWK_NIB_ID_RouteRecordTable	0x009c	Routing Record Table Value Format:
		`uint8_t numEntries;`
		`{`
		`    uint16_t destAddr;`
		`    uint16_t nextAddr;`
		`} entry[numEntries];`
			4 bytes per entry

ZB_NWK_NIB_ID_SecurityMaterialSet	0x00a1	Value Format:
		`uint8_t keySeqNumber;`
		`uint32_t outFrameCounter;`
		`uint8_t key[16];`
		`uint8_t keyType;`
			22 bytes per entry

ZB_NWK_NIB_ID_FrameCounterSet	0x0401	Incoming Network Frame Counters Value Format:
		`uint8_t numEntries;`
		`{`
		`    uint8_t keySeqNumber;`
		`    uint64_t senderAddr;`
		`    uint32_t counter;`
		`} entry[numEntries];`
			13 bytes per entry

ZB_NWK_NIB_ID_OutgoingCounter	0x0406	`ZB_PERSIST_TYPE_UINT32`	4 bytes per entry

Table 8. Miscellaneous items

Name	ID	Format	Length
MAC interface channel mask list	0x0001U	Up to eight entries of type: `ZB_PERSIST_TYPE_UINT32`	4 bytes per entry
Extended address	0x0002U	`ZB_PERSIST_TYPE_UINT64`	8 bytes per entry

15.2 Footprint and frequency

The following section contains the results of simulations run on four networks, each configured with a different central node, number of routers, and number of end devices.

The purpose of the simulations was to gather information regarding the memory footprint of persisted items, and frequency with which the state of these items was saved.

The data gathered during the 24-hour runtime of each network is described in the format detailed in the following section.

15.2.1 Network report format

Network configuration

The first section describes the configuration used when simulating a network, including:

- The type of central node: a ZigBee coordinator, or a ZigBee router without trust center capability
- The number of routers in the network
- The number of end devices in the network

Network architecture

This section details how the nodes in the simulated networks are connected.

Cluster configuration

This section describes the configuration of clusters being used with the devices on the simulated network, including:

- The types of cluster used
- The simulated use of each cluster over a 24-hour period

Note: The information listed in the next three sections is provided for both the central node of the network, and level two routers.

Writes to persistent memory

This section lists:

- The total number of writes to persistent memory over a 24-hour period
- The number of writes to persistent memory that occurred during the initialization of the network
- The total number of writes that occurred while the network was running.

Some of this information is provided for both the central node of the network, and level two routers

Persistent memory required

This section lists, in bytes, the total amount of persistent memory used by the configuration being tested.

Memory use table

This section consists of a table listing the amount of memory used by ZigBee stack for reasons detailed in the following table.

Table 9. Memory use table format for ZigBee stack

Name	Definition	Name	Definition
Persistence table version	Persistence table version	APS channel mask	Application support sublayer channel mask
bdb_ib	Base device behaviour information base	APS pre-conf link key	Application support sublayer pre-confirmation link key
aps_ib	Application support Sublayer Information Base	APS binding table	Application support sublayer binding table
nwk_ib	Network information base	APS group table	Application support sublayer group table
bdb_tl_key	Touchlink key	APS link key table	Application support sublayer link key table
address_map	Address map	MAC channels	Media access control channels
NNT	Network neighbour table	MAC power Table	Media access control power table
NRT	Network routing table	EUI address	Extended unique identifier address
NWK RREC	Table of outstanding route requests	ZCL persist server	Zigbee cluster library persist server
NWK key table	Network key table	-	

15.2.2 Network report

Network configuration ZigBee router 50 nodes network

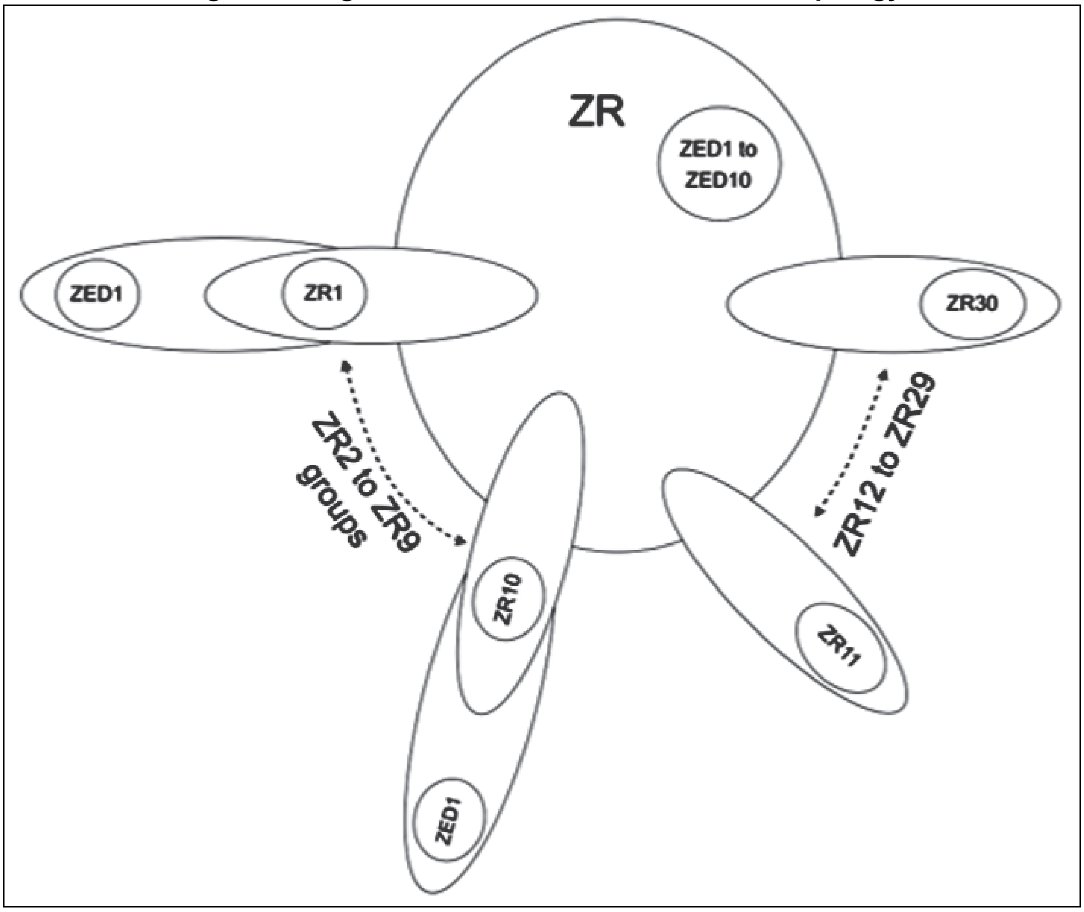
This network is configured to run with a total of 51 nodes: 31 ZigBee routers and 20 ZigBee end devices.

Network architecture

- 1 central router without trust center capability
- 10 end devices connected to the central router
- 30 level two routers
- 10 end devices connected to level two routers

Network diagram

Figure 22. ZigBee router with 50 nodes network topology



Cluster configuration

On/Off cluster:

- On/Off cluster clients are mapped to each end device connected to level two routers.
- On/Off servers are mapped to all level two routers.
- User presses On/Off button 20 times over 24 hours.

Temperature sensors:

- Temperature servers are on each standard end device connected to the central router.
- Reports to the client on the central router every 10 minutes.

Data: central router

Writes to persistent memory

- Total writes after 24 hours of testing: 85
- Flash writes during network initialization: 41
- Network initialization time: approximately 0.5 hours
- Runtime writes: 44

Persistent memory required

- 2636 bytes

Memory use table: central router

Table 10. Memory use ZigBee central router for 50 nodes network

Name	Bytes	Name	Bytes
Persistence Table Version	6	APS channel mask	9
bdb_ib	182	APS pre-conf link key	42
aps_ib	31	APS binding table	0
nwk_ib	169	APS group table	0
bdb_tl_key	21	APS link key table	1327
address_map	107	MAC channels	9
NNT	127	MAC power table	0
NRT	0	EUI address	13
NWK RREC	0	ZCL persist server	30
NWK Key table	563	-	

Note: The results listed above are for this specific network topology. Different network topologies yield different data. Even a small change to the network can result in a large change in the data gathered.

Data: level two routers

Writes to persistent memory

- Total writes after 24 hours of testing: 103
- Flash writes during network initialization: 4
- Network initialization time: approximately 3.5 hours
- Runtime writes: 99

Persistent memory required

- 979 bytes

Memory use table: level two router

Table 11. Memory use ZigBee level two router for 50 nodes network

Name	Bytes	Name	Bytes
Persistence table version	6	APS channel mask	9
bdb_ib	182	APS pre-conf link key	42
aps_ib	31	APS binding table	0
nwk_ib	169	APS group table	0
bdb_tl_key	21	APS link key table	1327
address_map	327	MAC channels	9
NNT	31	MAC power table	0
NRT	0	EUI address	13
NWK RREC	0	ZCL persist server	30
NWK key table	69	-	

Note: The results listed above are for this specific network topology. Different network topologies yield different data. Even a small change to the network can result in a large change in the data gathered.

Network configuration ZigBee router 400 nodes network

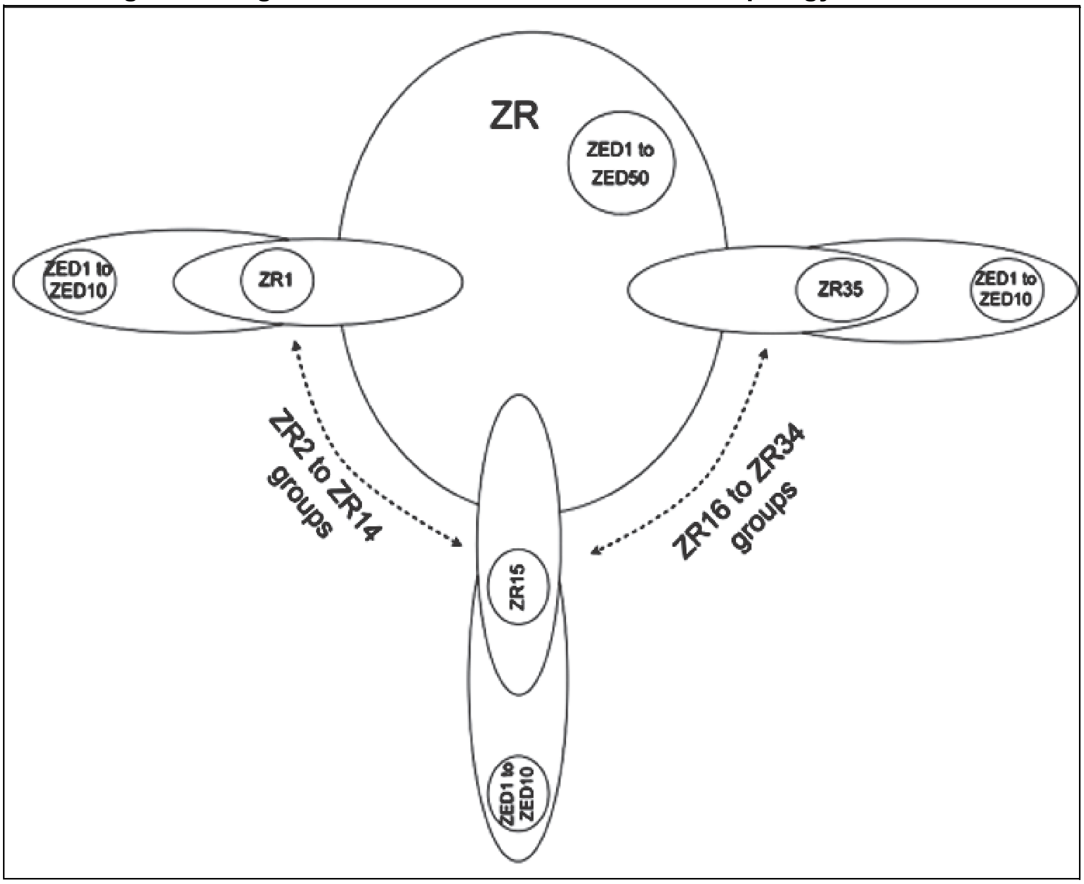
This network was configured to run with a total of 436 nodes: 36 ZigBee routers and 400 ZigBee end devices.

Network Architecture

- 1 central router without trust center capability
- 50 end devices connected to the central router
- 35 level two routers
- 350 end devices connected to level two routers

Network diagram

Figure 23. ZigBee router with 400 nodes network topology



Cluster configuration

On/Off cluster:

- On/Off cluster clients are mapped to each end device connected to level two routers
- On/Off servers are mapped to all level two routers
- User presses On/Off button 20 times over 24 hours.

Temperature sensors

- Temperature servers are on each standard end device connected to the central router
- Reports to the client on the central router every 10 minutes.

Data: central router

Writes to persistent memory

- Total writes after 24 hours of testing: 131
- Flash writes during network initialization: 86
- Network initialization time: approximately 3.5 hours
- Runtime writes: 45

Persistent memory required

- 8586 bytes

Memory use table: central router

Table 12. Memory use ZigBee central router for 400 nodes network

Name	Bytes	Name	Bytes
Persistence table version	6	APS channel mask	9
bdb_ib	182	APS pre-conf link key	42
aps_ib	31	APS binding table	0
nwk_ib	169	APS group table	0
bdb_tl_key	21	APS link key table	2812
address_map	3507	MAC channels	9
NNT	607	MAC power table	0
NRT	0	EUI address	13
NWK RREC	0	ZCL persist server	30
NWK Key Table	1148	-	

Note: The results listed above are for this specific network topology. Different network topologies yield different data. Even a small change to the network can result in a large change in the data gathered.

Data: level two routers

Writes to persistent memory

- Total writes after 24 hours of testing: 204
- Flash writes during network initialization: 37
- Network initialization time: approximately 3.5 hours
- Runtime writes: 167

Persistent memory required

- 1501 bytes

Memory use table: level two router

Table 13. Memory use ZigBee level two router for 400 nodes network

Name	Bytes	Name	Bytes
Persistence table version	6	APS channel mask	9
bdb_ib	182	APS pre-conf link key	42
aps_ib	31	APS binding table	0
nwk_ib	169	APS group table	0
bdb_tl_key	21	APS link key table	337
address_map	327	MAC channels	9
NNT	139	MAC power table	0
NRT	0	EUI address	13
NWK RREC	0	ZCL persist server	30
NWK key table	186	-	

Note: The results listed above are for this specific network topology. Different network topologies yield different data. Even a small change to the network can result in a large change in the data gathered.

Network configuration ZigBee coordinator 50 nodes network

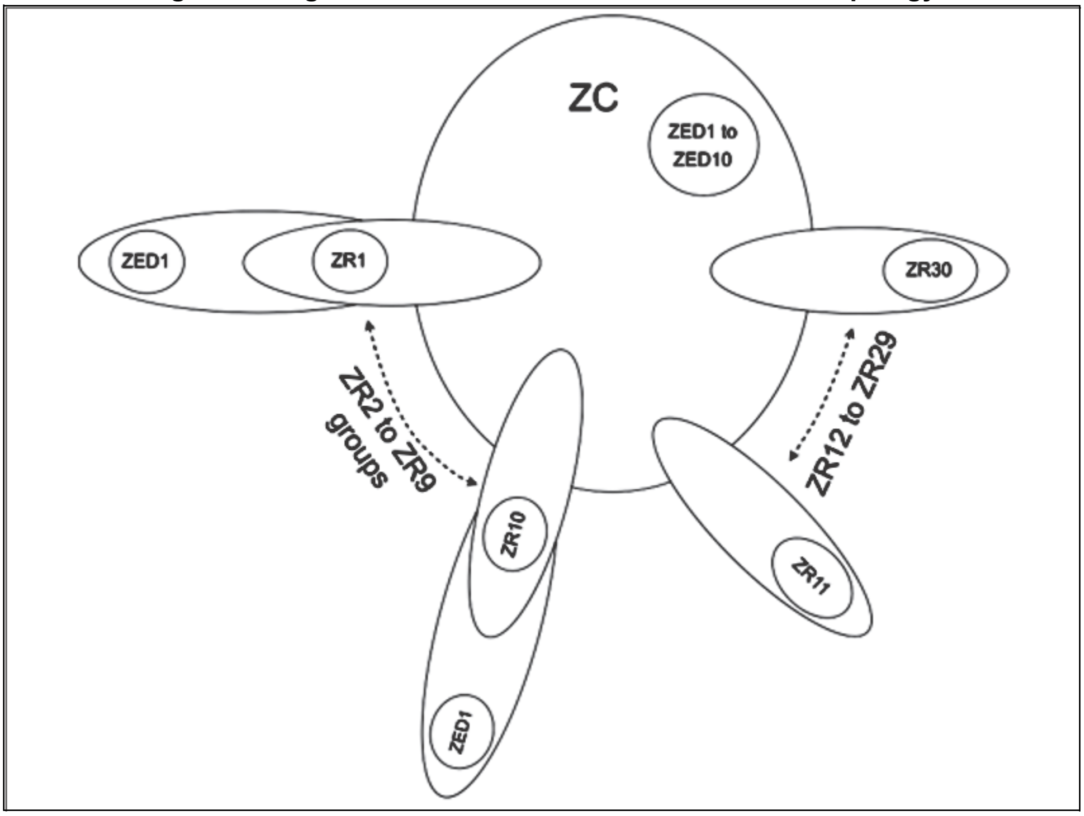
This network was configured to run with a total of 51 nodes: 1 ZigBee coordinator, 30 ZigBee routers and 20 ZigBee end devices.

Network architecture

- 1 ZigBee coordinator
- 10 end devices connected to the coordinator
- 30 level two routers
- 10 end devices connected to level two routers

Network diagram

Figure 24. ZigBee coordinator with 50 nodes network topology



Cluster configuration

On/Off cluster:

- On/Off cluster clients are mapped to each end device connected to level two routers
- On/Off Servers are mapped to routers all routers
- User presses On/Off button 20 times over 24 hours.

Temperature sensors:

- Temperature servers are on each standard end device connected to the coordinator
- Reports to the client on the coordinator every 10 minutes.

Data: ZigBee coordinator

Writes to persistent memory

- Total writes after 24 hours of testing: 109
- Flash writes during network initialization: 51
- Network initialization time: approximately 0.5 hours
- Runtime writes: 58

Persistent memory required

- 3183 bytes

Memory use table: ZigBee coordinator

Table 14. Memory use ZigBee coordinator for 50 nodes network

Name	Bytes	Name	Bytes
Persistence table version	6	APS channel mask	9
bdb_ib	182	APS pre-conf link key	42
aps_ib	31	APS binding table	0
nwk_ib	169	APS group table	0
bdb_tl_key	21	APS link key table	1657
address_map	117	MAC channels	9
NNT	127	MAC power table	0
NRT	0	EUI address	13
NWK RREC	207	ZCL persist server	30
NWK Key Table	563	-	

Note: The results listed above are for this specific network topology. Different network topologies yield different data. Even a small change to the network can result in a large change in the data gathered.

Data level two routers

Writes to persistent memory

- Total writes after 24 hours of testing: 110
- Flash writes during network initialization: 5
- Network initialization time: approximately 0.5 hours
- Runtime writes: 105

Persistent memory required

- 979 bytes

Memory use table: level two router

Table 15. Memory use level two router ZigBee coordinator for 50 nodes network

Name	Bytes	Name	Bytes
Persistence table version	6	APS channel mask	9
bdb_ib	182	APS pre-conf link key	42
aps_ib	31	APS binding table	0
nwk_ib	169	APS group table	0
bdb_tl_key	21	APS link key table	40
address_map	327	MAC channels	9
NNT	31	MAC power table	0
NRT	0	EUI address	13
NWK RREC	0	ZCL persist server	30
NWK Key Table	69	-	

Note: The results listed above are for this specific network topology. Different network topologies yield different data. Even a small change to the network can result in a large change in the data gathered.

Network configuration ZigBee coordinator 400 nodes network

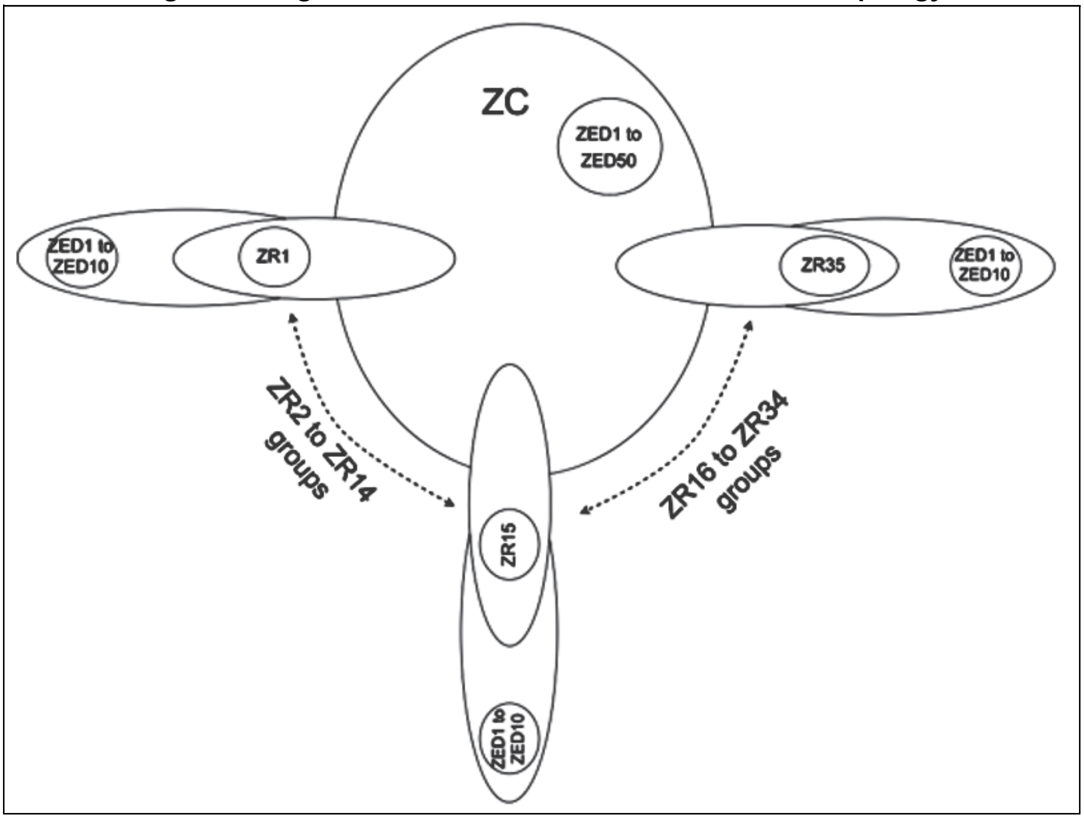
This network was configured to run with a total of 436 nodes: 1 ZigBee coordinator, 35 ZigBee routers and 400 ZigBee end devices.

Network architecture

- 1 ZigBee coordinator
- 50 end devices connected to the coordinator
- 35 level two routers
- 350 end devices connected to level two routers

Network Diagram

Figure 25. ZigBee coordinator with 400 nodes network topology



Cluster configuration

On/Off Cluster:

- On/Off cluster clients are mapped to each end device connected to level two routers
- On/Off servers are mapped to all routers
- User presses On/Off button 20 times over 24 hours.

Temperature sensors:

- Temperature servers are on each standard end device connected to the coordinator
- Reports to the client on the coordinator every 10 minutes.

Data: ZigBee coordinator

Writes to persistent memory

- Total writes after 24 hours of testing: 496
- Flash writes during network initialization: 436
- Network initialization time: approximately 3.5 hours
- Runtime writes: 60

Persistent memory required

- 21903 bytes

Memory use table: ZigBee coordinator

Table 16. Memory use ZigBee coordinator for 400 nodes network

Name	Bytes	Name	Bytes
Persistence table version	6	APS channel mask	9
bdb_ib	182	APS pre-conf link Key	42
aps_ib	31	APS binding table	0
nwk_ib	169	APS group table	0
bdb_tl_key	21	APS link key table	14362
address_map	3517	MAC channels	9
NNT	607	MAC power table	0
NRT	14	EUI address	13
NWK RREC	1743	ZCL persist server	30
NWK key table	1148	-	

Note: The results listed above are for this specific network topology. Different network topologies will yield different data. Even a small change to the network can result in a large change in the data gathered.

Data: level two routers

Writes to persistent memory

- Total writes after 24 hours of testing: 222
- Flash writes during network initialization: 51
- Network initialization time: approximately 3.5 hours
- Runtime writes:171

Persistent memory required

- 1204 bytes

Memory use table: level two router

Table 17. Memory use level two router ZigBee coordinator for 400 nodes network

Name	Bytes	Name	Bytes
Persistence table version	6	APS channel Mask	9
bdb_ib	182	APS pre-conf link Key	42
aps_ib	31	APS binding table	0
nwk_ib	169	APS group table	0
bdb_tl_key	21	APS link key table	40
address_map	327	MAC channels	9
NNT	139	MAC power table	0
NRT	0	EUI address	13
NWK RREC	0	ZCL persist server	30
NWK key table	186	-	

Note: The results listed above are for this specific network topology. Different network topologies will yield different data. Even a small change to the network can result in a large change in the data gathered.

15.3 Comments

Incoming frame counters are saved every 128th frame when originating from a unique sender. This is why updates occur on the Coordinator approximately every 20 minutes, as a result of link status and MTORR messages. The value 128 is used to limit the window for replay attack attempt

The outgoing frame counter window size is set to 2048, as it is not vulnerable to replay attacks. This means when a device resets, the outgoing frame counter jumps by 2048.

16 Revision history

Table 18. Document revision history

Date	Revision	Changes
18-May-2020	1	Initial release