Application Note

Porting Z-Wave Appl. SW from 400 to 500 Series

Document No.:	APL12445
Version:	4
Description:	The purpose of this document is to give guidelines for the Z Wave application developer, when porting software applications from 400 Series to 500 Series
Written By:	ABR;JFR;MVO;BBR
Date:	2018-03-06
Reviewed By:	EFH;MKIDMOSE;CHL;MVO;JFR;PSH;TRO;SSE;BBR
Restrictions:	Public

Approved by:

Date	CET	Initials	Name	Justification
2018-03-06	09:25:02	NTJ	Niels Thybo Johansen	

REVISION RECORD

Doc. Rev	Date	By	Pages affected	Brief description of changes
1	20140130	ABR	ALL	Initial draft; based on APL12444
2	20140307	JFR	3	Note about 400 Series binary compatibility
3	20140611	MVO	Section 3.3.5.2	Added comment about the RELOAD_BIT bit mask setting of ZW_PWM_init
4	20180306	BBR	All	Added Silicon Labs template

Table of Contents

1 ABBREVIATIONS

Abbreviation	Explanation
ADC	Digital to Analog Converter
AES	A symmetric block cipher algorithm. AES uses a block length of 128 bits and key lengths of 128, 192 or 256 bits.
API	Application Programming Interface
GPIO	General Purpose I/O
IDE	Integrated Design Environment
LED	Light Emitting Diode
NVM	Non-Volatile Memory
OTA	Over The Air (firmware update)
PWM	Pulse Width Modulator
RF	Radio Frequency
SDK	System Development Kit
WUT	Wake Up Timer

2 INTRODUCTION

2.1 Purpose

The purpose of this document is to give guidelines to Z-Wave application developers for porting applications from 400 series SDKs to 500 series SDKs.

2.2 Audience and prerequisites

The audience of this document is Z-Wave partners and Silicon Labs.

3 500 SERIES PORTING ISSUES

The following memory resources are available for application development:

- One 32kB bank of flash memory for executable code
- 4kB of RAM for temporary data
- 64 Bytes of internal NVM for persistent settings
- Additional external NVM may be available (depends on actual development module or product)

The above limitations MUST NOT be exceeded. Violating the limitations may impair compatibility with future SDK versions.

The 400 Series firmware are not binary compatible with the 500 Series SoC despite many similarities. Therefore, must a 400 Series based application be ported to the 500 Series SoC. The application porting process comprises a number of steps:

- Make sure you have KEIL compiler version PK51 v9.51a. The version MUST NOT be PK51 v9.52. Critical linker operations will fail if not using v9.51a. The PK51 installer asks if previous settings should be retained. Answer “No” to this question.

- Install SDK 6.51 in the folder “C:\DevKit_6_51\Product”. It is possible to install the SDK in another folder structure but the following examples refer to this folder structure.

- Select an appropriate embedded sample application distributed on the SDK. It is RECOMMENDED to use a sample application based on the same library as is used by the existing 400 series application.

- Make a copy of the selected sample application folder. Rename the folder to match the application in question.

- Delete all source files (.c, .h) in the new folder and copy the source files (.c, .h) of the 400 Series application into the new folder.

- Rename the application in the Makefile (look for “APP_NAME:=”) and update the object file list accordingly (look for “RELFILES:=”).

- Add a special prototype declaration to all call-back, interrupt and timer functions. SDK 6.5x introduces code space banking to utilize the full code space of the 500 series SoC. This again necessitates the use of a special prototype declaration like this:
  ```
  code const void (code * functionName_p)(BYTE b) = &functionName;
  ```

- Add source code for OTA firmware update support. Refer to [2]. (Optional)

- Update all references to NVM variables to follow the new scheme used in SDK 6.5x. Refer to [1].

- Modify API calls. A few API calls have changed.

- In secure applications, use the new AES API to avoid paying royalty fees.
- Build the application via the console. This step is also necessary if using uVision IDE for software design.
    - Issue the command “mk clean” to prepare a clean build
    - Issue the command “mk” with the proper command line arguments. uVision project files have to be auto-generated from the console. Specify “UVISION=1” in the command line to auto-generate the uVision project files.
- In uVision, do the following
    - Open the bootloader_... project file. Select “Clean Target” and then “Rebuild All Files”. This step is necessary to prepare the bootloader specific module which is necessary for building a complete firmware image for in-system programming via the ZDP03A programming hardware.
    - Open the real project file. Select “Clean Target” and then “Rebuild All Files”
- When having cleared all compilation errors, open the Z-Wave Programmer PC application and select the project hex file that has the string “BOOTLOADER” in it for device programming.

3.1 Example: Porting the SDK 6.02 LED Dimmer sample application to SDK 6.5x

This example shows how to port the LED Dimmer sample app to the SDK 6.5x based LED Dimmer on a 500 Series SoC.

3.1.1 Preparation

- In “C:\DevKit_6_51\Product”, make a copy of the folder “LED_Dimmer”.
- Rename the copied folder “LED_Dimmer_Port”.
- Copy the old source files (.c and .h) from “C:\DevKit_6_02\Product\LED_Dimmer” into “C:\DevKit_6_51\Product\LED_Dimmer_Port”, overwriting existing source files. Do NOT copy any other files from “C:\DevKit_6_02”.
- Adapt source files to SDK 6.5x API calls (see section 3.2).
- Remove all “#ifdef ZW020x” and “#ifdef ZW030x” branches until the corresponding #endif statement.
- Use a compare tool to see the source file changes made in each step.
- Adapt source files (.c and .h) to SDK 6.5x API code. Changes can be seen in LED_Dimmer_Port_step_3

3.1.2 Code space banking support

The 500 Series SoC uses code space banking, which requires changes to function declarations for call-back, interrupt and timer functions as well as new function prototypes for these functions. One example is the callback function `cbVoidByte` used by ZW_SendData:

```
/*========================   cbVoidByte   ========================
**
**  Function:  stub for callback
**
**  Side effects: None
**
**--------------------------------------------------------------------*/
static void cbVoidByte(BYTE b)
{
}
```

must be changed to:

```
code const void (code * ZCB_cbVoidByte_p)(BYTE b) = &ZCB_cbVoidByte;
/*======================   ZCB_cbVoidByte   =====================
**
**  Function:  stub for callback
**
**  Side effects: None
**
**--------------------------------------------------------------------*/
void
ZCB_cbVoidByte(BYTE b)
{
}
```

This change causes the linker to add ZCB_cbVoidByte to the interbank call table. The "`static`" attribute MUST NOT be used for call-back, interrupt and timer functions with SDK 6.5x. The linker cannot generate an interbank call table entry for non-public (static) functions. Finally, the string "ZCB_" is prepended for easy identification of the indirectly called functions. Unmodified functions will not be caught by compiler or linker. The result is unpredictable, but may include behaviors like non-responding applications or applications behaving in strange ways.

3.1.3 New NVM addressing method

Persistent settings like light level and dimmer status are stored in NVM storage.

Below is seen a snippet of eeprom.h of SDK 6.02:

```
/*****************************************************************************/
/*                      EXPORTED TYPES and DEFINITIONS                       */
/*****************************************************************************/

/* EEPROM address definitions */

/* EEPROM LED dimmer node layout */
#define EEOFFSET_LEVEL              0x00
#define EEOFFSET_STATUS             EEOFFSET_LEVEL + 1
#define EEOFFSET_IGNORE_ALL_ON_OFF  EEOFFSET_STATUS + 1
...
```

The definitions of eeprom.h must be changed along the following principle:

```
/*****************************************************************************/
/*                      EXPORTED TYPES and DEFINITIONS                       */
/*****************************************************************************/

extern BYTE far EEOFFSET_LEVEL_far;
extern BYTE far EEOFFSET_STATUS_far;
extern BYTE far EEOFFSET_IGNORE_ALL_ON_OFF_far;
...
```

Further, this last change necessitates the creation of a new file “eeprom.c” where the data structures of the NVM are actually defined:

```
/*****************************************************************************/
/*                      EXPORTED TYPES and DEFINITIONS                       */
/*****************************************************************************/

/**********************/
/*      NVM layout    */
/**********************/
BYTE far EEOFFSET_LEVEL_far;
BYTE far EEOFFSET_STATUS_far;
BYTE far EEOFFSET_IGNORE_ALL_ON_OFF_far;
...
```

It is recommended that the file “eeprom.c” is copied from the folder “C:\DevKit_6_51\Product\LED_Dimmer” in order to get the correct file format. The Makefile object file list already contains a line for the module eeprom since this is already a part of SDK 6.51.

3.1.4 Renaming sample application

Renaming the sample application to the product in question can be done by renaming the application folder within the “product” folder.

In “C:\DevKit_6_51\Product\LED_Dimmer_Port\Makefile”, modify the application name from “leddimmer” to “leddimmer_port” (look for “APP_NAME:=”). If the application to be ported also uses other .c files the object file list must be updated accordingly (look for “RELFILES:=”).

3.1.5 Controlling library, frequency and other properties

In SDK 6.51, the library and other properties of the application can be specified at the command line as parameters to the MK command.

Alternatively, the parameters may be specified as environment variables in the Makefile.

3.2 Mapping of external NVM from SDK 6.02 to SDK 6.5x

SDK6.5x implements one full-fledged API with support for Over-The-Air (OTA) firmware update of 500 Series SoC's. When planning a new product, the designer should decide which functionality level is required. This has impact on the required external NVM. The following options may be considered:

Category	OTA Support	Slave	Controller	Description
Very low cost		✓	✓	16kB EEPROM May not support future protocol versions. **(Not recommended)**
Low cost		✓	✓	32kB EEPROM (new) Support for future protocol versions guaranteed.
OTA Basic	✓	✓		128kB FLASH Support for OTA and future protocol versions. (**Slaves only)**
OTA Full	✓	✓	✓	256kB FLASH (new) Support for OTA and all future protocol versions.

Initialization of the external NVM is completely handled by the Z-Wave protocol and thereby obsoleting the NVM initialization file extern_epp.hex found in SDK 6.02.

3.2.1 Removing Patch system declations

The Patch system used for debugging purposes in 400 Series SoC's is not needed in the 500 Series environment.

Remove the following declarations to restore normal c code constructs:

- Remove all #ifdef PATCH_ENABLE branches until the corresponding #endif statement
- Remove the PATCH_FUNCTION_NAME_STARTER statement from function headers
- Remove the PATCH_VARIABLE statement from variable declarations

Generally, conduct an "all-files" search for the term "PATCH" to make sure the project has been completely cleaned.

3.3 Mapping of SDK 6.02 API calls to SDK 6.5x

This section describes changes in API calls when moving from SDK 6.02 to SDK 6.5x. For details, refer to [1].

3.3.1 Z-Wave Basis API

SDK 6.02	SDK 6.5x	Note
ZW_RF_above_3v_supply_guaranteed	Discontinued	400 series specific API function. Discontinued in 500 Series.

3.3.2 Z-Wave Transport API

The Z-Wave Transport API calls are the same.

3.3.3 Z-Wave TRIAC API

The Z-Wave TRIAC API calls are the same.

3.3.4 Z-Wave Timer API

The Z-Wave software timer API calls are the same.

3.3.5 Z-Wave Timer / GP Timer / GP PWM API

3.3.5.1 GP Timer operation

The Z-Wave GP Timer API calls are the same.

3.3.5.2 PWM operation

The Z-Wave GP PWM API calls are the same.

For the 400 series SoC you had to include RELOAD_BIT in the bit mask argument of the function ZW_PWM_init. This is no longer needed with the corresponding 500 Series SoC function call.

PWM pin positions are different on ZM5202 compared to SD3502 and ZM5101. Libraries support PWM operation by default in SD3502 and ZM5101. The API call ZW_UART0_zm5202_mode_enable should be used to map UART0 and PWM pins when using ZM5202.

3.3.6 Z-Wave Memory API

API calls are the same. However, NVM variables must now be declared and defined just like any other variable, apart from the needed use of the "far" keyword:

BYTE far EEOFFSET_SENSOR_LEVEL_far; /* Just an example */

For details refer to [1].

3.3.7 Z-Wave ADC API

The Z-Wave ADC API calls are the same.

3.3.8 Z-Wave Power Control API

The Z-Wave Power Control API calls are the same.

3.3.9 Z-Wave UART interface API

500 series has up to 2 UART's depending on the module/SoC used. In the UART API the x in ZW_UARTx_ refers to 0 for UART0 and 1 for UART1.

UART0 pin positions are different on ZM5202 compared to SD3502 and ZM5101. Libraries support UART0 operation by default in SD3502 and ZM5101. The API call ZW_UART0_zm5202_mode_enable should be used to map UART0 pins when using ZM5202.

3.3.10 Z-Wave Node Mask API

API calls are the same.

3.3.11 Z-Wave AES API

The new AES API provides royalty-free AES encryption for secure applications.

3.3.12 Z-Wave Controller API

API calls are the same.

3.3.13 Z-Wave Static Controller API

API calls are the same.

3.3.14 Z-Wave Bridge Controller API

API calls are the same.

3.3.15 Z-Wave Routing and Enhanced 232 Slave API

API calls are the same.

3.3.16 Serial Command Line Debugger

API calls are the same.

3.3.17 Hardware Pin Definitions

API calls are the same.

IO functions have the same IO port (Px.y) mapping in 500 Series SoC's.

4 500 SERIES MIGRATION CONSIDERATIONS

Migration from a 400 Series to a 500 Series based product can be done by simply replacing the module with a 400 Series connector compatible module such as ZDB5101 in case the hardware design supports this operation.

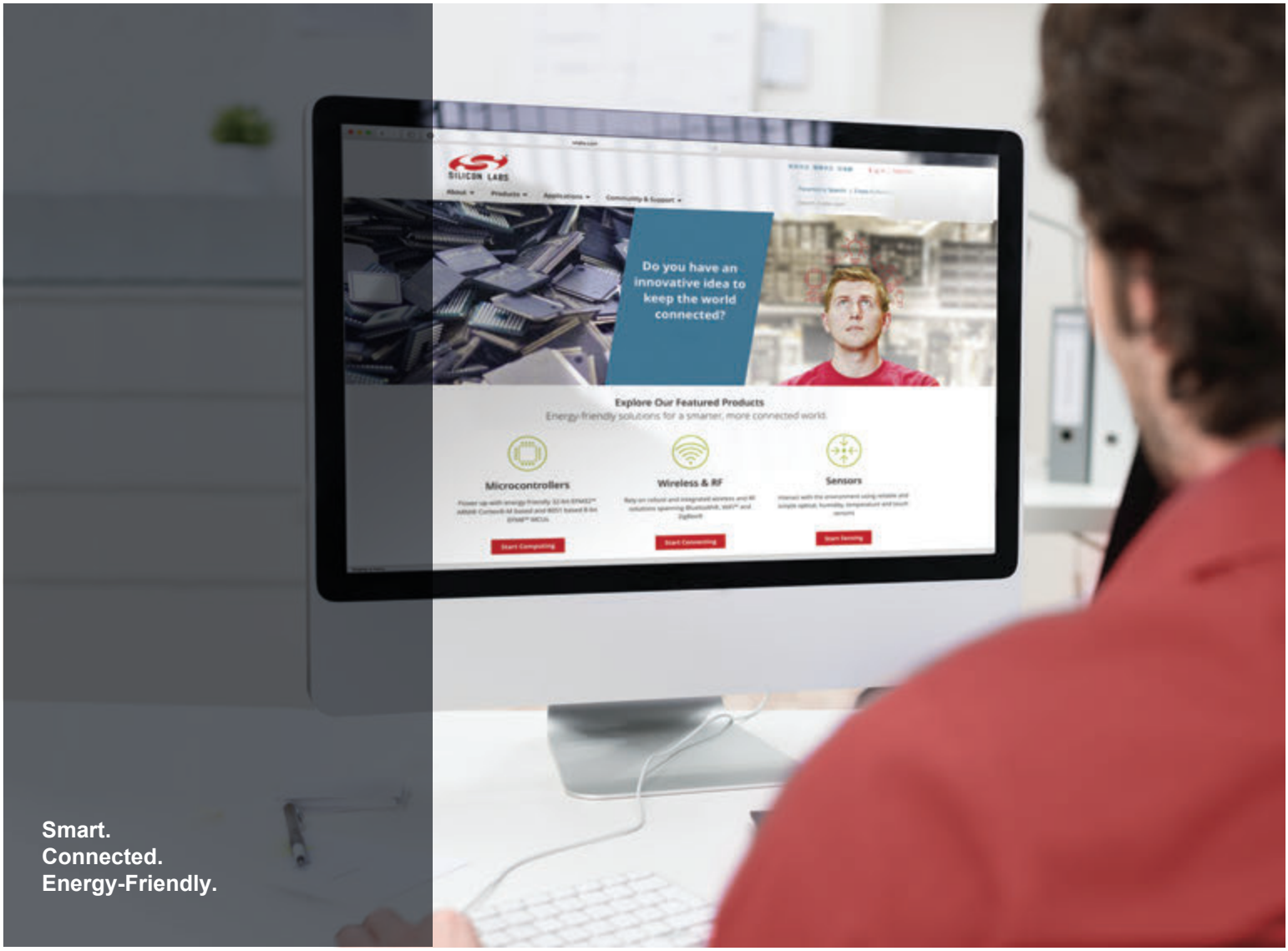
In case the product is already a part of an installation it must be included to the network. Associations, return routes and other state information may have to be restored. Alternatively, the content of the external NVM may be transferred to the new 500 Series based module; thus avoiding reinstallation.

In case other combinations of SDK versions and libraries need to be ported to SDK 6.5x, please contact Z-WaveSupport@SigmaDesigns.com

REFERENCES

[1] Silicon Labs, INS12308, Instruction, Z-Wave 500 Series Application Programming Guide.
[2] Silicon Labs, INS12366, Instruction, Working in 500 Series Environment User Guide.

INDEX



Products
www.silabs.com/products

Quality
www.silabs.com/quality

Support and Community
community.silabs.com

SILICON LABS®

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

http://www.silabs.com