Application Note

Porting Z-Wave Appl. SW from ZW0301 to 500 Series

Document No.:	APL12444
Version:	6
Description:	This document provides guidelines for porting software applications from ZW0301 to 500 Series
Written By:	JFR;EFH;SSE;ABR;JSI;PSH;BBR
Date:	2018-03-06
Reviewed By:	EFH;MVO;MKIDMOSE;BBR;PSH;TRO;SSE;JFR
Restrictions:	Public

Approved by:

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

REVISION RECORD

Doc. Rev	Date	By	Pages affected	Brief description of changes
1	20130402	SSE JFR	ALL	Initial draft
2	20130912	SSE	Section 1.3.19	Added a table that show how IOs with specific functions are mapped in 300 and 500 series
2	20130924	JFR	Section 1	Added OTA firmware update porting instructions
3	20131001	EFH	Section 1	Added NVM variable porting instructions
3	20131016	JFR	Section 1.2 Section 1.3.1	Added external NVM porting considerations Clarified ApplicationRFNotify porting considerations
3	20131108	ABR	ALL	Editorial clarifications
3	20131204	JFR	Section 1.3.14	Discontinued ZW_EnableSUC
3	20131212	SEE	Section 1.1.1 Section 1.3.6.2 Section 1.3.19	Restructured paragraph Pin dependence of module used. Added additional SPI interface
3	20131213	JSI	Section 2	Added migration considerations and example from 4.5x to 6.51
4	20140107	JFR	Section 1.3.7	Updated wrt. far variables
4	20140130	ABR	Section 1.2	Removed NVM data offsets as they are not needed.
4	20140219	JFR	Section 1.3.6	API call ZW_TIMER0_ext_gate discontinued
5	20140404	PSH	Section 1	Changed description of interrupt handling and referred to document [2] for more details
6	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
LED	Light Emitting Diode
OTA	Over The Air (firmware update)
PWM	Pulse Width Modulator
RF	Radio Frequency
SDK	System Development Kit
NVM	Non-Volatile Memory
IDE	Integrated Design Environment
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 SDK 4.5x to SDK 6.5x.

2.2 Audience and prerequisites

The audience of this document is Z-Wave partners and Sigma Designs.

1 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 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 ZW0301 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 ZW0301 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 and timer functions. SDK 6.5x introduces code space banking to utilize the full code space of the 500 series Z-Wave chip. This again necessitates the use of a special prototype declaration like this:
  ```
  code const void (code * functionName_p)(BYTE b) = &functionName;
  ```

- Move interrupt functions to a separate .c file and make sure that the linker locates the module in the common code bank. Refer to [2] for a detailed description of interrupt functions in the 500 series.

- 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 are 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.

1.1 Example: Porting the LED Dimmer sample application from SDK 4.5x to SDK 6.5x

This example shows how to port an SDK 4.5x based LED Dimmer to SDK 6.5x based LED Dimmer on a 500 Series single chip.

1.1.1 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_4_50\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_4_50”.
- Adapt source files to SDK 6.5x API calls (see section 1.2).
- Remove all “#ifdef ZW020x” and “#ifdef ZW030x” directives in the source code files.
- The porting process is illustrated in the file LED_Dimmer_Porting_SDK4_5x_to_SDK6_5x.zip situated in the same directory as this document.
- 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

1.1.2 Code space banking support

The 500 series chip uses code space banking, which requires changes to function declarations for call-back 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 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.

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 4.50:

```
/****************************************************************************/
/*                      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.

1.1.3 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:=”).

1.1.4 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.

1.2 Mapping of external NVM from SDK 4.5x to SDK 6.5x

SDK6.5x implements one full-fledged API with support for Over-The-Air (OTA) firmware update of 500 series chips. 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.

The NVM initialization file extern_epp.hex found in SDK 4.5x has been obsoleted.

NVM storage is not initialized by SDK 6.5x. The application must not assume that NVM contents are zero. The ApplicationInitSW must check the NVM integrity. If the NVM variable EEOFFSET_MAGIC_far is not equal to MAGIC_VALUE, the application must write all NVM variables including the EEOFFSET_MAGIC_far variable.

1.3 Mapping of SDK 4.5x API calls to SDK 6.5x

This section describes changes in API calls when moving from SDK 4.5x to SDK 6.5x. For details refer to [1].

All 300 series / SDK 4.5x applications that used direct SFR access for SPI, FLASH programming or any other HW peripherals should now use the relevant HW driver API call as direct SFR access is not possible any more. For details refer to [1].

1.3.1 Required Application Functions

SDK 4.5x	SDK 6.5x	Note
ApplicationRFNotify	ApplicationRFNotify	Same API in both SDK's. However, ApplicationRFNotify MUST be present in the application as ApplicationTestPoll. Remove ifdef around ApplicationRFNotify if present.

1.3.2 Z-Wave Basis API

SDK 4.5x	SDK 6.5x	Note
N/A	ZW_RF_above_3v_supply_guaranteed	400 series specific API function. Discontinued in 500 Series.
ZW_SetSleepMode		Moved to Z-Wave Power API in 6.5x.

1.3.3 Z-Wave Transport API

SDK 4.5x	SDK 6.5x	Note
ZW_SendData_Generic	Obsolete	
ZW_SendDataMeta_Generic	Obsolete	
ZW_SendConst	ZW_SendConst	Parameters changed see [1] for more details
N/A	ZW_SetListenBeforeTalkThreshold	400 series specific API function. Discontinued in 500 Series.

1.3.4 Z-Wave TRIAC API

SDK 4.5x	SDK 6.5x	Note
TRIAC_Init	ZW_TRIAC_init	SDK 4.5x supports 2 Zero-x modes, whereas SDK 6.5x supports 3 modes The SDK 6.5x supports a prescaler for the correction timer The SDK 6.5x supports a specific mode for FET's/IGBT's
	ZW_TRIAC_enable(TRUE)	-
TRIAC_SetDimLevel	ZW_TRIAC_dimlevel_set	SDK 4.5x supports 100 dimming steps , whereas SDK 6.5x supports 1000 steps
N.A	ZW_TRIAC_int_enable	SDK 4.5x does not support Triac interrupts.
N.A	ZW_TRIAC_int_get	SDK 4.5x does not support Triac interrupts.
N.A	ZW_TRIAC_int_clear	SDK 4.5x does not support Triac interrupts.
TRIAC_Off	ZW_TRIAC_enable(FALSE)	6.5x supports one API to Enable/Disable the TRIAC controller

1.3.5 Z-Wave Timer API

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

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

API calls are also added for the standard 8051 Timer0 and Timer1 and must replace SFR calls used in 300 Series. For details refer to [1].

Moved to Application HW Timers/PWM interface API in SDK 6.5x, where the API calls are divided into two sets, one for the GP Timer and one for the PWM.

1.3.6.1 GP Timer operation

SDK 4.5x	SDK 6.5x	Note
ZW_PWMSetup	ZW_GPTIMER_init/ ZW_GPTIMER_enable	Mode bit 1 in the function parameter set to 0 in SDK 4.5x.
ZW_PWMPrescale	ZW_GPTIMER_reload_set	
ZW_PWMClearInterrupt	ZW_GPTIMER_int_clear	
ZW_PWMEnable	ZW_GPTIMER_int_enable	
N.A.	ZW_GPTIMER_int_get	
N.A.	ZW_GPTIMER_pause	
N.A.	ZW_GPTIMER_reload_get	
N.A.	ZW_GPTIMER_get	

1.3.6.2 PWM operation

SDK 4.5x	SDK 6.5x	Note
ZW_PWMSetup	ZW_PWM_init/ ZW_PWM_enable	Mode bit 1 in the function parameter set to 1 in SDK 4.5x.
ZW_PWMPrescale	ZW_PWM_waveform_set	
ZW_PWMClearInterrupt	ZW_PWM_int_clear	
ZW_PWMEnable	ZW_PWM_int_enable	
N.A.	ZW_PWM_int_get	
N.A.	ZW_PWM_waveform_get	

PWM port is different on ZM5202 compared to SD3502 and ZM5101. Libraries support default SD3502 and ZM5101 with respect to PWM port. Use API call ZW_UART0_zm5202_mode_enable to map UART0 and PWM pins when using ZM5202.

1.3.7 Z-Wave Memory API

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

BYTE far EEOFFSET_SENSOR_LEVEL_far; /* Just an example */

For details refer to [1].

1.3.8 Z-Wave ADC API

SDK 4.5x	SDK 6.5x	Note
ADC_Off	ZW_ADC_power_enable(FALSE)	
ADC_Start	ZW_ADC_enable(TRUE)	
ADC_Stop	ZW_ADC_enable(FALSE)	
ADC_Init	ZW_ADC_init/ ZW_ADC_batt_monitor_enable	
ADC_SelectPin	ZW_ADC_pin_select	
ADC_Buf	ZW_ADC_buffer_enable	
ADC_SetAZPL	ZW_ADC_auto_zero_set	
ADC_SetResolution	ZW_ADC_resolution_set	
ADC_SetThresMode	ZW_ADC_threshold_mode_set	
ADC_SetThres	ZW_ADC_threshold_set	
ADC_Int	ZW_ADC_int_enable	
ADC_IntFlagClr	ZW_ADC_int_clear	
ADC_GetRes	ZW_ADC_result_get	

1.3.9 Z-Wave Power Control API

API calls are the same. However, documentation did not list ZW_SetSleepMode under this API.

1.3.10 Z-Wave UART interface API

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

SDK 4.5x	SDK 6.5x	Note
UART_Init	ZW_UARTx_init	
UART_RecStatus	-	
UART_RecByte	ZW_UARTx_rx_data_wait_get	
UART_SendStatus	ZW_UARTx_tx_active_get	
UART_SendByte	ZW_UARTx_tx_data_wait_set	
UART_SendNum	ZW_UARTx_tx_send_num	
UART_SendStr	ZW_UARTx_ tx_send_str	
UART_SendNL	ZW_UARTx_ tx_send_nl	
UART_Enable	ZW_UARTx_init	Parameter bEnableTx (set to TRUE) in ZW_UARTx_init
UART_Disable	ZW_UARTx_init	Parameter bEnableTx (set to FALSE) in ZW_UARTx_init
UART_ClearTx	ZW_UARTx_tx_int_clear	
UART_ClearRx	ZW_UARTx_rx_int_clear	
UART_Write	ZW_UARTx_tx_data_set	
UART_Read	ZW_UARTx_rx_data_get	
N.A.	ZW_UARTx_tx_int_get	
N.A.	ZW_UARTx_rx_int_get	
N.A.	ZW_UARTx_rx_active_get	

UART0 port is different on ZM5202 compared to SD3502 and ZM5101. Libraries support default SD3502 and ZM5101 with respect to UART0. Use API call ZW_UART0_zm5202_mode_enable to map UART0 and PWM pins when using ZM5202.

1.3.11 Z-Wave Node Mask API

API calls are the same.

1.3.12 Z-Wave AES API

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

1.3.13 Z-Wave Controller API

API calls are the same.

1.3.14 Z-Wave Static Controller API

SDK 4.5x	SDK 6.5x	Note
ZW_EnableSUC	Discontinued	Assignment of SUC/SIS functionality are now default enabled.

1.3.15 Z-Wave Bridge Controller API

API calls are the same.

1.3.16 Z-Wave Slave API

SDK 4.5x	SDK 6.5x	Note
ZW_Support9600Only	Discontinued	

1.3.17 Z-Wave Routing and Enhanced 232 Slave API

API calls are the same.

1.3.18 Serial Command Line Debugger

API calls are the same.

1.3.19 Hardware Pin Definitions

Moved to GPIO helper macros in SDK 6.5x.

SDK 4.5x	SDK 6.5x	Note
PIN_ON	PIN_HIGH	
PIN_OFF	PIN_LOW	
N.A.	ZW_io_set	Used in ApplicationInitHW instead of PIN_HIGH/PIN_LOW

Bellow table shows how IOs with specific functions are mapped in 300 series and 500 series chips:

Specific function IO	IO pin in 300 series	IO pin in 500 series
INT1	P1.7	P1.1
INT0	P1.6	P1.0
MISO as a master[1]	P1.2	P2.3 (SPI1) P2.6 (SPI0)
SCK as a master[1]	P1.4	P2.4 (SPI1) P2.7 (SPI0)
MOSI as a master[1]	P1.3	P2.2 (SPI1) P2.5 (SPI0)
MISO as a slave	P1.2	P2.6 (SPI0)
SCK as a slave	P1.4	P2.7 (SPI0)
MOSI as a salve	P1.3	P2.5 (SPI0)
ADC3	P1.1	P3.7
ADC1	P0.1	P3.5
ADC0	P0.0	P3.4
ADC2	P1.0	P3.6
RXD	P1.1	P3.4 – ZM5202 P2.0 – other 500 Series chips/modules

[1] You cannot mix SPI interfaces GPIOs. For example, you cannot use SPI1_MOSI, SPI0_MISO, and SPI0_SCK. They all should be from the same interface.

Specific function IO	IO pin in 300 series	IO pin in 500 series
TXD	P1.0	P3.5 – ZM5202 P2.1 – other 500 Series chips/modules
TRIAC	P0.1	P3.6
ZEROX	P0.0	P3.7
PWM	P1.6	P1.0 – ZM5202 P3.7 – other 500 Series chips/modules
SS_N (SPI0)	P1.5	P3.0

2 500 SERIES MIGRATION CONSIDERATIONS

Migration from a 300 Series to a 500 Series based product can be done by simply replacing the module with a 300 Series connector compatible module ZDB5202 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 and restore any associations.

Another possibility is to transfer the content of the external NVM to the new 500 Series based module and thereby avoiding reinstallation.

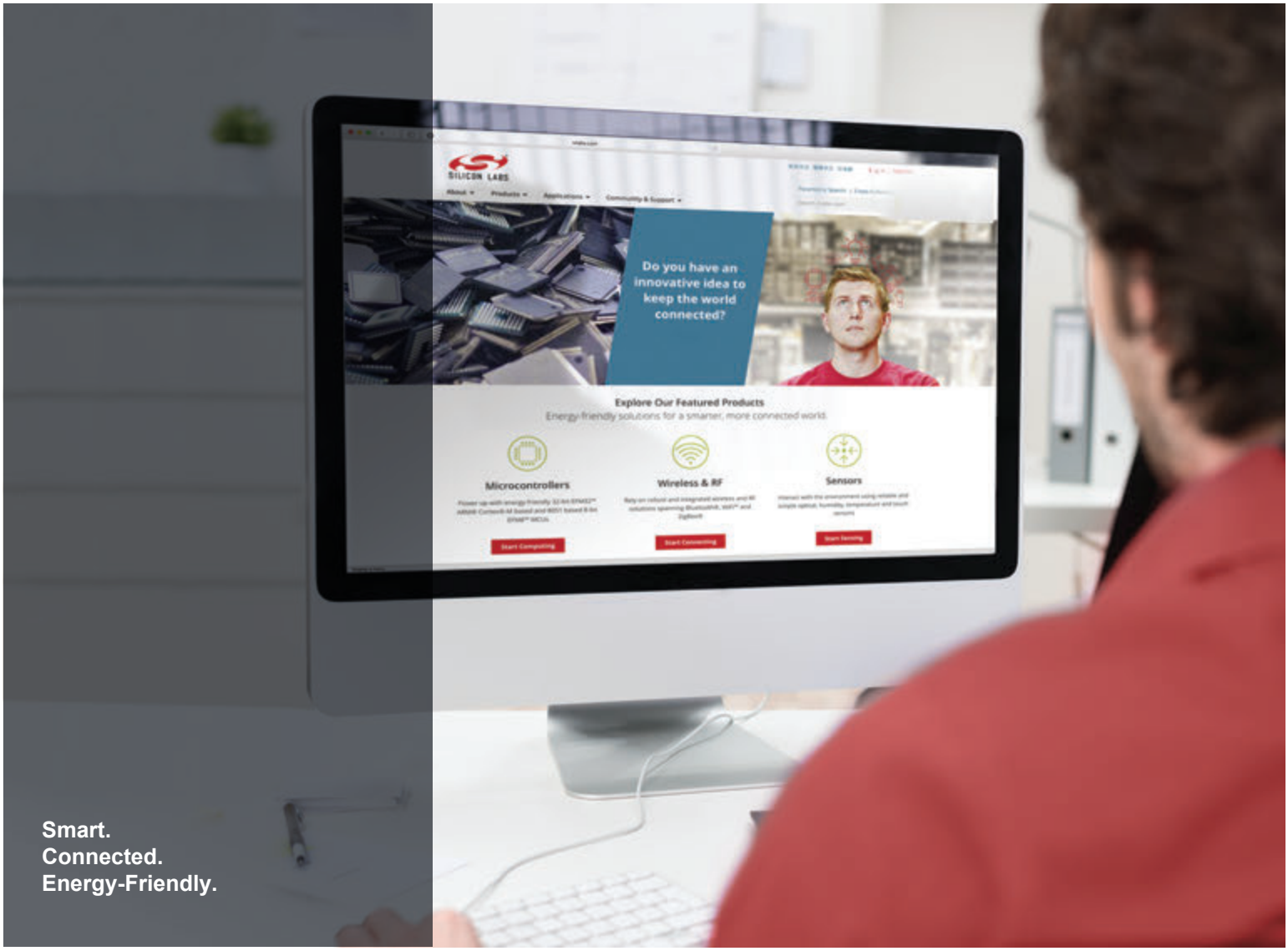
As an example: If an installation contains a 4.5x controller_static_nosuc_norep_ZW030x based application and the destination is an 6.51.00 controller_static_ZW050x based application the Z-Wave protocol contents in NVM (0x0000-0x2BFF) can be directly copied 0x0000-0x2BFF in the new module NVM. Two bytes in the new image must then be changed. Byte at address 0x0015 must be set to 0x54 and the byte at address 0x0016 must be set to 0xA5. The Application data in NVM must then be copied from 0x2C00- to 0x6000- in the new ZW05x image.

In case other combinations of SDK versions and libraries must be transfer to SDK 6.51.00 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