Application Note

Porting Z-Wave Appl. SW from ZW0301 to 400 Series

Document No.:	APL10979
Version:	3
Description:	The purpose of this document is to give guidelines for the Z Wave application developer, when porting software applications from ZW0301 to 400 Series
Written By:	JFR;SSE;EFH;MVO;TJC;TRO
Date:	2011-02-25
Reviewed By:	SSE;TRO
Restrictions:	Partners Only

Approved by:

Date	CET	Initials	Name	Justification
2011-02-25	11:23:19	JFR	Jørgen Franck	on behalf of NTJ

CONFIDENTIAL

REVISION RECORD

Doc. Rev	Date	By	Pages affected	Brief description of changes
1	20100821	JFR EFH MVO SSE	ALL	Initial draft
2	20110127	EFH JFR	Section 1.1.1	Additional porting details regarding Makefile.Common and Makefile.
3	20110225	JFR	Section 1.2.6	Added limitations with respect to the standard 8051 timers Timer0 and Timer1.

Table of Contents

1 ABBREVIATIONS

Abbreviation	Explanation
AES	The Advanced Encryption Standard is a symmetric block cipher algorithm. The AES is a NIST-standard cryptographic cipher that uses a block length of 128 bits and key lengths of 128, 192 or 256 bits. Officially replacing the Triple DES method in 2001, AES uses the Rijndael algorithm developed by Joan Daemen and Vincent Rijmen of Belgium.
API	Application Programming Interface
PWM	Pulse Width Modulator
RF	Radio Frequency
WUT	Wake Up Timer

2 INTRODUCTION

2.1 Purpose

The purpose of this document is to give guidelines for the Z-Wave application developer, when porting SDK 4.51 based software applications from ZW0301 to 400 Series based hardware platforms.

2.2 Audience and prerequisites

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

1 400 SERIES PORTING ISSUES

The software porting process contains a number of logical steps to achieve a ported product on the 400 Series based hardware platform:

- Select an appropriate embedded sample application distributed on the Developer's Kit. Use typically a sample application using the wanted library.
- Rename sample application to the application in question.
- Move source code into application taking development mode (patch system) into account.
- Update API calls to 400 Series, especially API calls related to peripherals have changed significantly. Secure applications can also use the new AES API and thereby avoid royalty payment to SIC's AES implementation.
- Build application.

1.1 SDK 4.51 based LED Dimmer

This example shows how to port a SDK 4.51 based LED Dimmer to SDK 6.0x based on a 400 Series single chip.

1.1.1 Porting the sample application from SDK 4.5x to SDK 6.0x

Create a new application source directory LED_Dimmer_Port in the installed SDK 6.01.01 directory C:\DevKit_6_01_01\Product. Copy LED_Dimmer application from SDK 4.51 to LED_Dimmer_Port directory.

Adapt Makefile to SDK 6.0x common makefiles. Add definitions for $(APP) and $(LIB) on every real target MAKELINE. Change all ZW020x targets to ZW040x targets. Rename the application common makefile to "Makefile.common".

Adapt project source files to SDK 6.0x API calls (see section 1.2). Remove all "#ifdef ZW020x" and "#ifdef ZW030x" directives in the source code files.

Now the LED_Dimmer is ready to be compiled and executable in the ZW040x in normal mode.

The porting process is illustrated in the file LED_Dimmer_Porting_SDK4_5x_to_SDK6_0x.zip situated in the same directory as this document. Use a tool to see the changes made in each step:

- Make a copy of the old application directory to a new application directory:
  C:\Product\LED_Dimmer_Port_r18848
- Makefile rename:
  C:\Product\LED_Dimmer_Port_r18849
    - Makefile.Testdefines change to Makefile.TestDefines to comply with a case-sensitive makefile system.
- Adapt Makefiles to SDK 6.0x common Makefiles:
  C:\Product\LED_Dimmer_Port_r18850

    - Makefile.leddimmer_common change to Makefile.common

- Adapt Source files to SDK 6.0x API for normal mode OTP code:
  C:\Product\LED_Dimmer_Port_r18852

    - Changes in LEDdim.c

- Change makefile method of specifying what to build:
  C:\Product\LED_Dimmer_Port_r19889

    - Changes in Makefile.common now situated in directory Z-Wave\Common.

- Change MyProduct name to starter in Makefile:
  C:\Product\LED_Dimmer_Port_r19897

    - MyProduct_devmode -> starter_devmod,
      MyProduct.obj -> starter.obj &
      MyProduct_patch.obj -> starter_patch.obj

1.1.2 Renaming sample application

Renaming the sample application to the product in question can be done by renaming the base directory of the application. The application name is also defined by definition and injection of the $(APP) variable for the targets in the projects main Makefile.

1.1.3 Changing library

Changing the library used for the application is now controlled by the definition and the injection of the $(LIB) variable for the targets in the projects main Makefile.

1.1.4 Incorporating patch system

Add development mode targets to the applications Makefiles.

Insert standard patch system macro defines and includes in the beginning of every C-source file in your application.

Change all the global variables in every C-source file in your application. If the variable shall be shared between the patchable OTP code and the patch RAM code, then use the PATCH_VARIABLE macro. Else you can make the variable static (not public, and thus local to the module).

Change all public functions in every C-source file in your application. Use the PATCH_FUNCTION_NAME macro for handling the function name. Make the function reentrant to locate the functions frame on the reentrant pseudo stack. Insert the PATCH_TABLE_ENTRY macro with the function name as the very first code in every function. Local variables for the function shall be declared before the PATCH_TABLE_ENTRY, and initialization of local variables shall be done in code after the PATCH_TABLE_ENTRY. A special variant of the two macros, PATCH_FUNCTION_NAME_STARTER and PATCH_TABLE_ENTRY_STARTER exists for use where we need a wrapper function for handling differences in the functions frame format (different parameters and/or different local variables). This is the case for the starter targets, where the patchable empty application and the real patch application contains standard wrappers for all required application functions.

Function definition:

Static void ApplicationPoll(void)

```
{
  BYTE  lastAction = OneButtonLastAction();
….
```

New patch function definition:

```
Void PATCH_FUNCTION_NAME_STARTER(ApplicationPoll)(void)
#ifdef PATCH_ENABLE
reentrant
#endif
{
  BYTE lastAction;
#ifdef PATCH_ENABLE
#pragma asm
PATCH_TABLE_ENTRY_STARTER(ApplicationPoll)
#pragma endasm
#endif
  lastAction = OneButtonLastAction();
….
```

Function declaration:

```
 Extern void ApplicationPoll( void ) ;
```

New patch function declaration:

```
 Extern void ApplicationPoll( void )
 #ifdef PATCH_ENABLE
 reentrant
 #endif
 ;
```

Change all your IDATA and DATA variables to XDATA variables.

The porting process is illustrated in the file LED_Dimmer_Porting_SDK4_5x_to_SDK6_0x.zip situated in the same directory as this document. Use a tool to see the changes made in each step:

Adding development targets to the Makefiles:
C:\Product\LED_Dimmer_Port_r18853

Incorporation of patch-macros into application source files to be able to build both patchable OTP code and patch code for development RAM:
C:\Product\LED_Dimmer_Port_r18855

1.2 Mapping of SDK 4.51 API calls to SDK 6.0x

This section describes changes in API calls when moving from SDK 5.51 to SDK 6.0x. The SDK 6.0x contains also additional API calls, for details refer to [1].

1.2.1 Required Application Functions

SDK 4.51	SDK 6.0x	Note
ApplicationRFNotify	Discontinued	-

1.2.2 Z-Wave Basis API

SDK 4.51	SDK 6.0x	Note
	ZW_RF_above_3v_supply_guaranteed	
ZW_SetSleepMode	ZW_SetSleepMode	Moved to Z-Wave Power API

1.2.3 Z-Wave Transport API

SDK 4.51	SDK 6.0x	Note
ZW_SendData_Generic	Obsolete	
ZW_SendDataMeta_Generic	Obsolete	
ZW_SendConst	ZW_SendConst	Parameters changed see [1] for more details
	ZW_SetListenBeforeTalkThreshold	

1.2.4 Z-Wave TRIAC API

SDK 4.51	SDK 6.0x	Note
TRIAC_Init	ZW_TRIAC_init	SDK 4.51 supports 2 Zero-x modes, whereas SDK 6.0x supports 3 modes The SDK 6.0x supports a prescaler for the correction timer The SDK 6.0x supports a specific mode for FET's/IGBT's
	ZW_TRIAC_enable(TRUE)	-
TRIAC_SetDimLevel	ZW_TRIAC_dimlevel_set	SDK 4.51 supports 100 dimming steps , whereas SDK 6.0x supports 1000 steps
	ZW_TRIAC_int_enable	SDK 4.51 does not support Triac interrupts.
	ZW_TRIAC_int_get	SDK 4.51 does not support Triac interrupts.
	ZW_TRIAC_int_clear	SDK 4.51 does not support Triac interrupts.
TRIAC_Off	ZW_TRIAC_enable(FALSE)	-

1.2.5 Z-Wave Timer API

API calls are the same.

1.2.6 Z-Wave PWM API (GP Timer)

Moved to Application HW Timers/PWM interface API in SDK 6.0x, where the API calls are divided into two sets, one for the GP Timer and one for the PWM.

API calls are also added for the standard 8051 Timer0. However,

1.2.6.1 GP Timer operation

SDK 4.51	SDK 6.0x	Note
ZW_PWMSetup	ZW_GPTIMER_init/ ZW_GPTIMER_enable	Bit 1 in the function parameter set to 0 in SDK 4.51.
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	

1.2.6.2 PWM operation

SDK 4.51	SDK 6.0x	Note
ZW_PWMSetup	ZW_PWM_init/ ZW_PWM_enable	Bit 1 in the function parameter set to 1 in SDK 4.51.
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	

1.2.7 Z-Wave Memory API

API calls are the same.

1.2.8 Z-Wave ADC API

SDK 4.51	SDK 6.0x	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	

1.2.9 Z-Wave Power API

SDK 4.51	SDK 6.0x	Note
ZW_SetWutTimeout	ZW_SetWutTimeout	

1.2.10 Z-Wave UART interface API

SDK 4.51	SDK 6.0x	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 in ZW_UARTx_init
UART_Disable	ZW_UARTx_init	Parameter bEnableTx 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 pin positions are different on ZM4102 compared to SD3402 and ZM4101. Libraries support default SD3402 and ZM4101 with respect to UART0. Use API call ZW_UART0_zm4102_mode_enable to map UART0 pins when using ZM4102.

1.2.11 Z-Wave Node Mask API

API calls are the same.

1.2.12 Z-Wave Controller API

API calls are the same.

1.2.13 Z-Wave Static Controller API

API calls are the same.

1.2.14 Z-Wave Bridge Controller API

API calls are the same.

1.2.15 Z-Wave Installer Controller API

API calls are the same.

1.2.16 Z-Wave Slave API

SDK 4.51	SDK 6.0x	Note
ZW_Support9600Only	Discontinued	

1.2.17 Z-Wave Routing and Enhanced Slave API

API calls are the same.

1.2.18 Serial Command Line Debugger

API calls are the same.

1.2.19 Hardware Pin Definitions

Moved to GPIO helper macros in SDK 6.0x.

SDK 4.51	SDK 6.0x	Note
PIN_ON	PIN_HIGH	
PIN_OFF	PIN_LOW	
N.A.	ZW_io_set	Used in ApplicationInitHW instead of PIN_HIGH/PIN_LOW

REFERENCES

[1] Sigma Designs, INS10682, Instruction, Z-Wave 400 Series Application Programming Guide.

INDEX