



## Instruction

### Z-Wave ZW0201/ZW0301 Series Developer's Kit v4.54.02 Contents

<b>Document No.:</b>	INS12351
<b>Version:</b>	2
<b>Description:</b>	Describes the contents and sample applications user guides of the Z-Wave 400 Series Developer's Kit v4.5x
<b>Written By:</b>	JFR
<b>Date:</b>	2012-12-04
<b>Reviewed By:</b>	BBR;ABR;EFH
<b>Restrictions:</b>	Public

#### Approved by:

Date	CET	Initials	Name	Justification
2012-12-04	14:26:43	NTJ	Niels Thybo Johansen	

This document is the property of Sigma Designs Inc. The data contained herein, in whole or in part, may not be duplicated, used or disclosed outside the recipient for any purpose. This restriction does not limit the recipient's right to use information contained in the data if it is obtained from another source without restriction.



## REVISION RECORD

Doc. Ver.	Date	By	Pages affected	Brief description of changes
1	20090924	JFR	ALL	Initial draft
1	20090928	JFR JSI	3 & 4 3 & 4 3.1 & 3.2.4.9 3.1 & 3.2.4.15 5	Discontinued slave library. India (IN) frequency added for both applications and tools. Added ZW_slave_enhanced_noflirs_nomr_ZW020xs.lib Added ZW_slave_routing_noflirs_nomr_ZW020xs.lib Tool code added
1	20091229	VVI	5.1.1	Added description of Z-Wave programmer firmw are source files
1	20110902	DDA	4.8	Added explanation of LEDs functioning on ZDP03A
1	20111011	JSI	3.3.1, 3.3.2, 3.3.6, 3.3.7, 4.2, 4.3, 4.7, 4.8	Both None-Secure and Secure versions of Bin_Sensor(Bin_Sensor_Battery) are build using same sourcecode. It is similar for DoorLock and LED_Dimmer having none-secure and secure versions.
1	20111101	JFR	3.4	Removed Eeprom Loader and Equinox
1	20111108	JFR	4.6 & 4.7	Updated wakeup intervals used
1	20111205	JFR	All	Added RU frequency
1	20120326	JFR	3.3.1	Filename changed showing supported serial API functions
1	20120329	JFR	4.3 4.8	Updated description of Binary Sensor Battery code. Updated description of Led Dimmer code.
1	20121028	JFR	4.12	Moved serial API embedded code to document INS12350
2	20121130	JFR	4.8	Configuration CC removed from Secure LED Dimmer due to code space shortage.

# Table of Contents

<b>1</b>	<b>ABBREVIATIONS .....</b>	<b>1</b>
<b>2</b>	<b>INTRODUCTION .....</b>	<b>2</b>
2.1	Purpose.....	2
2.2	Audience and prerequisites .....	2
<b>3</b>	<b>SOFTWARE COMPONENTS .....</b>	<b>3</b>
3.1	Directory Structure .....	3
3.2	Z-Wave .....	7
3.2.1	Common .....	7
3.2.2	Include .....	7
3.2.3	I/O Defines .....	8
3.2.4	Libraries .....	8
3.2.4.1	Bridge Controller without SUC, repeater and FLiRS functionality.....	8
3.2.4.2	Installer Controller .....	9
3.2.4.3	Portable Controller .....	10
3.2.4.4	Static Controller without repeater, FLiRS and manual routing functionality .....	10
3.2.4.5	Static Controller without SUC and repeater functionality .....	10
3.2.4.6	Static Controller without SUC and repeater functionality but includes PA .....	11
3.2.4.7	Static Controller without SUC and FLiRS functionality .....	11
3.2.4.8	Enhanced Slave .....	11
3.2.4.9	Enhanced Slave without FLiRS and Manual Routing functionality .....	12
3.2.4.10	Enhanced 232 Slave .....	12
3.2.4.11	Enhanced 232 Slave without FLiRS and Manual Routing functionality .....	12
3.2.4.12	Production Test DUT .....	13
3.2.4.13	Production Test Generator .....	13
3.2.4.14	Routing Slave .....	13
3.2.4.15	Routing Slave without FLiRS and Manual Routing functionality .....	13
3.2.5	RF Frequency .....	13
3.3	Product.....	14
3.3.1	Bin.....	15
3.3.1.1	Bin_Sensor .....	15
3.3.1.2	Bin_Sensor_Sec .....	15
3.3.1.3	Bin_Sensor_Battery .....	15
3.3.1.4	Bin_Sensor_Battery_Sec .....	15
3.3.1.5	Dev_Ctrl .....	15
3.3.1.6	Dev_Ctrl_AVR_Sec .....	16
3.3.1.1	DoorBell .....	16
3.3.1.2	DoorLock .....	16
3.3.1.3	DoorLock_Sec .....	16
3.3.1.4	LED_Dimmer.....	16
3.3.1.5	LED_Dimmer_Sec .....	17
3.3.1.6	MyProduct.....	17
3.3.1.7	Prod_Test_DUT.....	17
3.3.1.8	Prod_Test_Gen .....	17
3.3.1.9	SerialAPI_Controller_Bridge_Nosuc_Norep_Noflirs .....	18
3.3.1.10	SerialAPI_Controller_Bridge_Nosuc_Norep_Noflirs_Mr .....	18
3.3.1.11	SerialAPI_Controller_Installer .....	19
3.3.1.12	SerialAPI_Controller_Installer_Mr .....	19
3.3.1.13	SerialAPI_Controller_Portable .....	20
3.3.1.14	SerialAPI_Controller_Portable_Mr .....	20
3.3.1.15	SerialAPI_Controller_Static_Norep_Noflirs_Nomr .....	21

3.3.1.16	SerialAPI_Controller_Static_Nosuc_Noflirs .....	21
3.3.1.17	SerialAPI_Controller_Static_Nosuc_Noflirs_Mr .....	22
3.3.1.18	SerialAPI_Controller_Static_Nosuc_Norep .....	22
3.3.1.19	SerialAPI_Controller_Static_Nosuc_Norep_PA .....	23
3.3.1.20	SerialAPI_Slave_Enhanced .....	23
3.3.1.21	SerialAPI_Slave_Enhanced_232 .....	24
3.3.1.22	SerialAPI_Slave_Enhanced_232_Mr .....	24
3.3.1.23	SerialAPI_Slave_Enhanced_232_Noflirs .....	25
3.3.1.24	SerialAPI_Slave_Enhanced_Mr .....	25
3.3.1.25	SerialAPI_Slave_Enhanced_Noflirs .....	26
3.3.1.26	SerialAPI_Slave_Routing .....	26
3.3.1.27	SerialAPI_Slave_Routing_Mr .....	27
3.3.1.28	SerialAPI_Slave_Routing_Noflirs .....	27
3.3.2	Binary Sensor .....	27
3.3.3	Development Controller .....	27
3.3.4	Secure Development Controller based on serial API and using an AVR as host .....	28
3.3.5	Doorbell .....	28
3.3.6	Door Lock .....	28
3.3.7	LED Dimmer .....	28
3.3.8	MyProduct .....	28
3.3.9	Production Test DUT .....	28
3.3.10	Production Test Generator .....	28
3.3.11	Serial API .....	28
3.3.12	Utilities .....	28
3.4	Tools .....	30
3.4.1	ERTT .....	31
3.4.2	IncDep .....	31
3.4.3	Intel UPnP .....	31
3.4.4	Make .....	31
3.4.5	Mergehex .....	31
3.4.6	Programmer .....	32
3.4.7	PVT and RF Regulatory .....	33
3.4.8	Python .....	33
3.4.9	TextTools .....	33
3.4.10	XML Editor .....	34
3.4.11	Zniffer .....	34
3.5	PC .....	35
3.5.1	Bin .....	35
3.5.2	Source .....	35
3.5.2.1	Libraries .....	35
3.5.2.2	Sample Application .....	37
<b>4</b>	<b>APPLICATION CODE .....</b>	<b>38</b>
4.1	Building ZW0x0x Code .....	38
4.1.1	MK.BAT .....	39
4.1.2	Makefiles .....	42
4.2	Binary Sensor Code .....	44
4.2.1	Network Wide Inclusion .....	45
4.2.2	Interface .....	47
4.2.3	Bin_Sensor Files .....	47
4.2.3.1	Macros for accessing the LED's .....	48
4.3	Binary Sensor Battery Code .....	49
4.3.1	Network Wide Inclusion .....	50
4.3.2	Interface .....	50
4.3.3	Bin_Sensor_Battery Files .....	51
4.4	Development Controller Code .....	52
4.4.1	Network Wide Inclusion .....	52

4.4.2	Dev_Ctrl Files.....	53
4.5	Secure Development Controller (ATmega) Code .....	55
4.5.1	Dev_Ctrl_AVR_Sec Files. ....	55
4.6	Door Bell Code.....	57
4.6.1	Network Wide Inclusion .....	57
4.6.2	User interface .....	57
4.6.3	Door Bell Files .....	58
4.7	Door Lock Code .....	59
4.7.1	Network Wide Inclusion .....	60
4.7.2	User Interface.....	60
4.7.3	Secure Door Lock Files .....	60
4.7.3.1	Macros for accessing the Lock/Unlock .....	61
4.8	LED Dimmer Code .....	63
4.8.1	Network Wide Inclusion .....	64
4.8.2	Interface.....	64
4.8.3	LED_Dimmer Files.....	64
4.8.3.1	Macros for accessing the LED's .....	65
4.9	MyProduct Code .....	67
4.9.1	MyProduct Files .....	67
4.10	Production Test DUT .....	68
4.10.1	Production Test DUT Files.....	69
4.11	Production Test Generator .....	71
4.11.1	Production Test Generator Files .....	73
4.12	Serial API Embedded Code .....	74
4.13	PC based Controller Sample Application .....	74
4.14	PC based Installer Tool Sample Application .....	74
4.15	PC based Z-Wave Bridge Sample Application .....	74
<b>5</b>	<b>TOOL CODE.....</b>	<b>75</b>
5.1	Z-Wave Programmer Firmware .....	76
5.1.1	ATmega_ZWaveProgFW Files .....	76
<b>6</b>	<b>REQUIRED DEVELOPMENT COMPONENTS .....</b>	<b>78</b>
6.1	Software development components .....	78
6.2	ZW0102/ZW0201/ZW0301 single chip programmer.....	78
6.3	Hardware development components for ZW0102 .....	79
6.4	Hardware development components for ZW0201 .....	80
6.5	Hardware development components for ZW0301 .....	80
6.6	ZW0102/ZW0201/ZW0301 lock bit settings.....	81
6.7	External EEPROM initialization .....	82
	<b>REFERENCES .....</b>	<b>84</b>
	<b>INDEX .....</b>	<b>85</b>

## Table of Tables

Table 1. Lock bits settings during development .....	81
Table 2. Lock bits settings in end products .....	81

# 1 ABBREVIATIONS

Abbreviation	Explanation
ACK	Acknowledge
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.
ANZ	Australia/New Zealand
API	Application Programming Interface
ASIC	Application Specific Integrated Circuit
DLL	Dynamic Link Library
DUT	Device Under Test
ERTT	Enhanced Reliability Test tool
EU	Europe
GNU	An organization devoted to the creation and support of Open Source software
HK	Hong Kong
HW	Hardware
IN	India
ISR	Interrupt Service Routines
LRC	Longitudinal Redundancy Check
MY	Malaysia
NAK	Not Acknowledged
NWI	Network Wide Inclusion
PA	Power Amplifier
POR	Power On Reset
PRNG	Pseudo-Random Number Generator
PWM	Pulse Width Modulator
RF	Radio Frequency
RU	Russian Federation
SFR	Special Function Registers
SIS	SUC ID Server
SOF	Start Of Frame
SPI	Serial Peripheral Interface
SUC	Static Update Controller
UPnP	Universal Plug and Play
US	United States
WUT	Wake Up Timer
XML	eXtensible Markup Language

## **2 INTRODUCTION**

### **2.1 Purpose**

The purpose of this document is to describe the contents on the Z-Wave Developer's Kit. Document contains also a description of all embedded sample applications including user guide or reference to relevant document.

### **2.2 Audience and prerequisites**

The audience is Z-Wave Partners.

## 3 SOFTWARE COMPONENTS

The Z-Wave development software packet consists of a protocol part, sample applications and a number of tools used for developing and building the Code.

### 3.1 Directory Structure

The development software is organized in the following directory structure:

```
/
  - PC
    - Bin
      - ZwaveDll
      - ZWaveInstaller
      - ZWavePCController
        - Non-secure
        - Secure
      - ZWaveUPnPBridge
    - Source
      - Libraries
        - WinFormsUI
        - ZensysFramework
        - ZensysFrameworkUI
        - ZensysFrameworkUIControls
        - ZWaveCommandClasses
        - ZWaveDll
        - ZWaveHAL
      - SampleApplications
        - ZWaveInstaller
        - ZWavePCController
        - ZWaveUPnPBridge
```



- Product
  - Bin
    - Bin\_Sensor
    - Bin\_Sensor\_Sec
    - Bin\_Sensor\_Battery
    - Bin\_Sensor\_Battery\_Sec
    - Dev\_Ctrl
    - Dev\_Ctrl\_AVR\_Sec
    - DoorBell
    - DoorLock
    - DoorLock\_Sec
    - LED\_Dimmer
    - LED\_Dimmer\_Sec
    - Prod\_Test\_DUT
    - Prod\_Test\_Gen
    - SerialAPI\_Controller\_Bridge\_Nosuc\_Norep\_Noflirs
    - SerialAPI\_Controller\_Bridge\_Nosuc\_Norep\_Noflirs\_Mr
    - SerialAPI\_Controller\_Installer
    - SerialAPI\_Controller\_Installer\_Mr
    - SerialAPI\_Controller\_Portable
    - SerialAPI\_Controller\_Portable\_Mr
    - SerialAPI\_Controller\_Static\_Norep\_Noflirs\_Nomr
    - SerialAPI\_Controller\_Static\_Nosuc\_Noflirs
    - SerialAPI\_Controller\_Static\_Nosuc\_Noflirs\_Mr
    - SerialAPI\_Controller\_Static\_Nosuc\_Norep\_PA
    - SerialAPI\_Controller\_Static\_Nosuc\_Norep
    - SerialAPI\_Slave\_Enhanced
    - SerialAPI\_Slave\_Enhanced\_232
    - SerialAPI\_Slave\_Enhanced\_232\_Mr
    - SerialAPI\_Slave\_Enhanced\_232\_Noflirs
    - SerialAPI\_Slave\_Enhanced\_Mr
    - SerialAPI\_Slave\_Enhanced\_Noflirs
    - SerialAPI\_Slave\_Routing
    - SerialAPI\_Slave\_Routing\_Mr
    - SerialAPI\_Slave\_Routing\_Noflirs
  - Bin\_Sensor
  - Bin\_Sensor\_Sec
  - Dev\_Ctrl\_AVR\_Sec
  - Dev\_Ctrl
  - DoorBell
  - DoorLock
  - LED\_Dimmer
  - LED\_Dimmer\_Sec
  - MyProduct
  - Prod\_Test\_DUT
  - Prod\_Test\_Gen
  - SerialAPI
  - Util\_Func

- Tools
  - ERTT
    - PC
    - Z-Wave\_Firmware
  - IncDep
  - Intel\_UPnP
  - Make
  - Mergehex
  - Programmer
    - PC
      - Source
      - SD3402\_Calibration
      - ZDP0xA\_Firmware
      - Source
  - PVT\_and\_RF\_regulatory
  - Python
  - TextTools
  - XML\_Editor
    - PC
  - Zniffer
    - PC
      - FileConverter
    - Z-Wave\_Firmware

- Z-Wave
  - Common
  - include
  - IO\_defines
  - lib
    - controller\_bridge\_nosuc\_norep\_noflirs\_ZW020x
    - controller\_bridge\_nosuc\_norep\_noflirs\_ZW030x
    - controller\_installer\_ZW020x
    - controller\_installer\_ZW030x
    - controller\_portable\_ZW020x
    - controller\_portable\_ZW030x
    - controller\_static\_norep\_noflirs\_nomr\_ZW020x
    - controller\_static\_norep\_noflirs\_nomr\_ZW030x
    - controller\_static\_nosuc\_noflirs\_ZW020x
    - controller\_static\_nosuc\_noflirs\_ZW030x
    - controller\_static\_nosuc\_norep\_pa\_ZW030x
    - controller\_static\_nosuc\_norep\_ZW020x
    - controller\_static\_nosuc\_norep\_ZW030x
    - slave\_enhanced\_232\_noflirs\_nomr\_ZW020x
    - slave\_enhanced\_232\_noflirs\_nomr\_ZW030x
    - slave\_enhanced\_232\_ZW020x
    - slave\_enhanced\_232\_ZW030x
    - slave\_enhanced\_ZW020x
    - slave\_enhanced\_ZW030x
    - slave\_enhanced\_noflirs\_nomr\_ZW020x
    - slave\_enhanced\_noflirs\_nomr\_ZW030x
    - slave\_prodtest\_dut\_ZW020x
    - slave\_prodtest\_dut\_ZW030x
    - slave\_prodtest\_gen\_ZW020x
    - slave\_prodtest\_gen\_ZW030x
    - slave\_routing\_noflirs\_nomr\_ZW020x
    - slave\_routing\_noflirs\_nomr\_ZW030x
    - slave\_routing\_ZW020x
    - slave\_routing\_ZW030x
- rf\_freq

This directory structure contains all the tools and sample applications needed, except the recommended Keil software, which must be purchased separately. More information about where and how to buy the Keil software development components are described in paragraph 6.1.

**Note!** Recommending leaving the directory structure as is due to compiler and linker issues.

The majority of the above mentioned Z-Wave specific tools and sample application are briefly described in the following sections.

## 3.2 Z-Wave

The Z-Wave header files and libraries are the software files needed for building a Z-Wave enabled product. The files are organized in directories used for building Z-Wave controllers and slaves respectively.

### 3.2.1 Common

The Common directory contains a set of standard make files needed for building the sample applications. The make files define the compiler options, linker options and defines for the different library types.

### 3.2.2 Include

The include directory contain all the header files ZW\_XXX\_api.h with declarations of API calls etc. The header files are the same for ZW0201 and ZW0301. Refer to chapter **Error! Reference source not found.** regarding a detailed description.

Warning: Disabled linker warning L25 'DATA TYPES DIFFERENT' to allow ZW\_classcmd.h updates as device and command class development progress.  
Refer to Makefile.common\_ZW0x0x\_appl files in Common directory regarding linker parameters.

### 3.2.3 I/O Defines

The Product\IO\_defines directory contains hardware definition files needed for building an application e.g. the development controller sample application.

**AppRFSetup.a51**

This file is an assembler file that should be used to define which RF setup is to be used (ANZ/EU/HK/IN/MY/RU/US) in the transmissions. Also if special values for capacity match array RX/TX or Normal/Low power transmission levels are needed than these can be defined here.

**ZW\_evaldefs.h**

This file contains definitions of the connector pins on the controller board.

**ZW\_pindefs.h**

This file contains definitions of the connector pins on the ZM12xxRE/ZM21xxE/ZM31xxC-E module, and macros for accessing the I/O pins. Refer to paragraph **Error! Reference source not found.** regarding a detail description.

**ZW\_portdefs.h**

This file contains I/O port initialization vectors on the single chips.

### 3.2.4 Libraries

The lib directory structure contains all the supported libraries and single chip series.

#### 3.2.4.1 Bridge Controller without SUC, repeater and FLIRS functionality

The lib\controller\_bridge\_nosuc\_norep\_noflirs\_ZW0x0x directory contains all files needed for building a Z-Wave bridge controller application. The directory contains the following files:

<b>ZW_controller_bridge_nosuc_norep_noflirs_zw020xs.lib</b> <b>ZW_controller_bridge_nosuc_norep_noflirs_zw030xs.lib</b>	These files are the compiled Z-Wave protocol and API library for the ZW0201/ZW0301 based modules that a Z-Wave controller bridge application should be linked together with. The library requires a ZM12xxRE/ZM21xxE/ZM31xxC-E module.
<b>extern_eep.hex</b>	This file contains the external EEPROM data on the ZM12xxRE/ZM21xxE/ZM31xxC-E module. The external EEPROM must only be initialized once.

### 3.2.4.2 Installer Controller

The lib\controller\_installer\_ZW0x0x directory contains all files needed for building a Z-Wave installer controller application. The directory contains the following files:

<b>ZW_controller_installer_ZW020xs.lib</b> <b>ZW_controller_installer_ZW030xs.lib</b>	These files are the compiled Z-Wave protocol and API library for the ZW0201/ZW0301 based modules that a Z-Wave installer application should be linked together with. The library requires a ZM12xxRE/ZM21xxE/ZM31xxC-E module.
<b>extern_eep.hex</b>	This file contains the external EEPROM data on the ZM12xxRE/ZM21xxE/ZM31xxC-E module. The external EEPROM must only be initialized once.

### 3.2.4.3 Portable Controller

The lib\controller\_ZW0x0x directory contains all files needed for building a Z-Wave controller application. The directory contains the following files:

**ZW\_controller\_portable\_zw020xs.lib**  
**ZW\_controller\_portable\_zw030xs.lib**

These files are the compiled Z-Wave protocol and API library for the ZW0201/ZW0301 based modules that a Z-Wave portable controller application should be linked together with. The library requires a ZM12xxRE/ZM21xxE/ZM31xxC-E module.

**extern\_eep.hex**

This file contains the external EEPROM data on the ZM12xxRE/ZM21xxE/ZM31xxC-E module. The external EEPROM must only be initialized once.

### 3.2.4.4 Static Controller without repeater, FLIRS and manual routing functionality

The lib\controller\_static\_norep\_noflirs\_nomr\_ZW0x0x directory contains all files needed for building a Z-Wave static controller application. The directory contains the following files:

**ZW\_controller\_static\_norep\_noflirs\_nomr\_zw020xs.lib**  
**ZW\_controller\_static\_norep\_noflirs\_nomr\_zw030xs.lib**

These files are the compiled Z-Wave protocol and API library for the ZW0201/ZW0301 based modules that a Z-Wave static controller application should be linked together with. The library requires a ZM12xxRE/ZM21xxE/ZM31xxC-E module.

**extern\_eep.hex**

This file contains the external EEPROM data on the ZM12xxRE/ZM21xxE/ZM31xxC-E module. The external EEPROM must only be initialized once.

### 3.2.4.5 Static Controller without SUC and repeater functionality

The lib\controller\_static\_nosuc\_norep\_ZW0x0x directory contains all files needed for building a Z-Wave static controller application without SUC and repeater functionality. The directory contains the following files:

**ZW\_controller\_static\_nosuc\_norep\_zw020xs.lib**  
**ZW\_controller\_static\_nosuc\_norep\_zw030xs.lib**

These files are the compiled Z-Wave protocol and API library for the ZW0201/ZW0301 based modules that a Z-Wave static controller application should be linked together with. The library requires a ZM12xxRE/ZM21xxE/ZM31xxC-E module.

**extern\_eep.hex**

This file contains the external EEPROM data on the ZM12xxRE/ZM21xxE/ZM31xxC-E module. The external EEPROM must only be initialized once.

### 3.2.4.6 Static Controller without SUC and repeater functionality but includes PA

The lib\controller\_static\_nosuc\_norep\_pa\_ZW030x directory contains all files needed for building a Z-Wave static controller application without SUC and repeater functionality. In addition, this library refers to the function ApplicationRFNotify enabling control of an external PA. The directory contains the following files:

<b>ZW_controller_static_nosuc_norep_pa_zw030xs.lib</b>	These files are the compiled Z-Wave protocol and API library for the ZW0301 based modules that a Z-Wave static controller application should be linked together with. The library requires a ZM31xxC-E module.
<b>extern_eep.hex</b>	This file contains the external EEPROM data on the ZM31xxC-E module. The external EEPROM must only be initialized once.

### 3.2.4.7 Static Controller without SUC and FLiRS functionality

The lib\controller\_static\_nosuc\_noflirs\_ZW0x0x directory contains all files needed for building a Z-Wave static controller application. The directory contains the following files:

<b>ZW_controller_static_nosuc_noflirs_zw020xs.lib</b> <b>ZW_controller_static_nosuc_noflirs_zw030xs.lib</b>	These files are the compiled Z-Wave protocol and API library for the ZW0201/ZW0301 based modules that a Z-Wave static controller application should be linked together with. The library requires a ZM12xxRE/ZM21xxE/ZM31xxC-E module.
<b>extern_eep.hex</b>	This file contains the external EEPROM data on the ZM12xxRE/ZM21xxE/ZM31xxC-E module. The external EEPROM must only be initialized once.

### 3.2.4.8 Enhanced Slave

The lib\slave\_enhanced\_ZW0x0x directory contains all files needed for building a Z-Wave enhanced slave node application. The directory contains the following files:

<b>ZW_slave_enhanced_ZW020xs.lib</b> <b>ZW_slave_enhanced_ZW030xs.lib</b>	These files are the compiled Z-Wave protocol and API library for the ZW0201/ZW0301 based modules that a Z-Wave enhanced slave application should be linked together with. The library requires a ZM12xxRE/ZM21xxE/ZM31xxC-E module.
<b>extern_eep.hex</b>	This file contains the external EEPROM data without home ID on the ZM12xxRE/ZM21xxE/ZM31xxC-E module. The external EEPROM must only be initialized once.



### 3.2.4.9 Enhanced Slave without FLiRS and Manual Routing functionality

The lib\slave\_enhanced\_noflirs\_nomr\_ZW0x0x directory contains all files needed for building a Z-Wave enhanced slave node application without FLiRS and Manual Routing functionality. The directory contains the following files:

**ZW\_slave\_enhanced\_noflirs\_nomr\_ZW020xs.lib**  
**ZW\_slave\_enhanced\_noflirs\_nomr\_ZW030xs.lib**

These files are the compiled Z-Wave protocol and API library for the ZW0201/ZW0301 based modules that a Z-Wave enhanced slave application should be linked together with. The library requires a ZM12xxRE/ZM21xxE/ZM31xxC-E module.

**extern\_eep.hex**

This file contains the external EEPROM data without home ID on the ZM12xxRE/ZM21xxE/ZM31xxC-E module. The external EEPROM must only be initialized once.

### 3.2.4.10 Enhanced 232 Slave

The lib\slave\_enhanced\_232\_ZW0x0x directory contains all files needed for building a Z-Wave enhanced 232 slave node application. The directory contains the following files:

**ZW\_slave\_enhanced\_232\_ZW020xs.lib**  
**ZW\_slave\_enhanced\_232\_ZW030xs.lib**

These files are the compiled Z-Wave protocol and API library for the ZW0201/ZW0301 based modules that a Z-Wave enhanced 232 slave application should be linked together with. The library requires a ZM12xxRE/ZM21xxE/ZM31xxC-E module.

**extern\_eep.hex**

This file contains the external EEPROM data without home ID on the ZM12xxRE/ZM21xxE/ZM31xxC-E module. The external EEPROM must only be initialized once.

### 3.2.4.11 Enhanced 232 Slave without FLiRS and Manual Routing functionality

The lib\slave\_enhanced\_232\_noflirs\_nomr\_ZW0x0x directory contains all files needed for building a Z-Wave enhanced 232 slave node application without FLiRS and Manual Routing functionality. The directory contains the following files:

**ZW\_slave\_enhanced\_232\_noflirs\_nomr\_ZW020xs.lib**  
**ZW\_slave\_enhanced\_232\_noflirs\_nomr\_ZW030xs.lib**

These files are the compiled Z-Wave protocol and API library for the ZW0201/ZW0301 based modules that a Z-Wave enhanced 232 slave application should be linked together with. The library requires a ZM12xxRE/ZM21xxE/ZM31xxC-E module.

**extern\_eep.hex**

This file contains the external EEPROM data without home ID on the ZM12xxRE/ZM21xxE/ZM31xxC-E module.

The external EEPROM must only be initialized once.

#### 3.2.4.12 Production Test DUT

The lib\slave\_prodtest\_dut\_ZW0x0x directory contains all files needed for building a production test DUT application on a Z-Wave module. The directory contains the following files:

**ZW\_slave\_prodtest\_dut\_ZW020xs.lib**  
**ZW\_slave\_prodtest\_dut\_ZW030xs.lib**

These files are the compiled Z-Wave protocol and API library for the ZW0201/ZW0301 based modules that a Z-Wave production test DUT application should be linked together with.

#### 3.2.4.13 Production Test Generator

The lib\slave\_prodtest\_ZW0x0x directory contains all files needed for building a production test generator application on a Z-Wave module. The directory contains the following files:

**ZW\_slave\_prodtest\_gen\_ZW020xs.lib**  
**ZW\_slave\_prodtest\_gen\_ZW030xs.lib**

These files are the compiled Z-Wave protocol and API library for the ZW0201/ZW0301 based modules that a Z-Wave production test generator application should be linked together with.

#### 3.2.4.14 Routing Slave

The lib\slave\_routing\_ZW0x0x directory contains all files needed for building a Z-Wave routing slave node application on a Z-Wave module. The directory contains the following files:

**ZW\_slave\_routing\_ZW020xs.lib**  
**ZW\_slave\_routing\_ZW030xs.lib**

These files are the compiled Z-Wave protocol and API library for the ZW0201/ZW0301 based modules that a Z-Wave routing slave application should be linked together with.

#### 3.2.4.15 Routing Slave without FLiRS and Manual Routing functionality

The lib\slave\_routing\_ZW0x0x directory contains all files needed for building a Z-Wave routing slave node application without FLiRS and Manual Routing functionality. The directory contains the following files:

**ZW\_slave\_routing\_noflirs\_nomr\_ZW020xs.lib**  
**ZW\_slave\_routing\_noflirs\_nomr\_ZW030xs.lib**

These files are the compiled Z-Wave protocol and API library for the ZW0201/ZW0301 based modules that a Z-Wave routing slave application should be linked together with.

### 3.2.5 RF Frequency

The rf\_freq directory contains all the object files ZW\_rf\_xxxx\_xx.obj with RF initialization settings for each single chip series and frequency.

### **3.3 Product**

The Product directory contains Z-Wave sample applications for a number of different product examples. Both source code and precompiled files ready for download are supplied.

Each directory contains the necessary files for creating ANZ (921.42MHz), EU (868.42MHz), HK (919.82MHz), IN (865.22MHz), MY (868.10MHz), RU (869.0MHz) and US (908.42MHz), products.

### 3.3.1 Bin

The Product\Bin directory structure contains the precompiled code of the Z-Wave sample applications and the hex files needed to download to the Z-Wave ZW0x0x single chip via the Z-Wave Programmer.

#### 3.3.1.1 Bin\_Sensor

The Product\Bin\Bin\_Sensor directory contains all files needed for running a binary sensor sample application on a Z-Wave module. The directory contains the following files:

<b>binsensor_ZW020x_y.hex</b>	The compiled and linked binary sensor sample application for y = ANZ, EU, HK, IN, MY, RU and US frequency versions of the ZW0201 based module.
<b>binsensor_ZW030x_y.hex</b>	The compiled and linked binary sensor sample application for y = ANZ, EU, HK, IN, MY, RU and US frequency versions of the ZW0301 based module.

#### 3.3.1.2 Bin\_Sensor\_Sec

Secure binary sensor sample application binaries not distributed due to export restrictions. Contact support via [zensys\\_support@sigmadesigns.com](mailto:zensys_support@sigmadesigns.com) for further information.

#### 3.3.1.3 Bin\_Sensor\_Battery

The Product\Bin\Bin\_Sensor\_Battery directory contains all files needed for running a battery operated binary sensor sample application on a Z-Wave module. The directory contains the following files:

<b>binsensor_Battery_ZW020x_y.hex</b>	The compiled and linked battery operated binary sensor sample application for y = ANZ, EU, HK, IN, MY, RU and US frequency versions of the ZW0201 based module.
<b>binsensor_Battery_ZW030x_y.hex</b>	The compiled and linked battery operated binary sensor sample application for y = ANZ, EU, HK, IN, MY, RU and US frequency versions of the ZW0301 based module.

#### 3.3.1.4 Bin\_Sensor\_Battery\_Sec

Secure battery operated binary sensor sample application binaries not distributed due to export restrictions. Contact support via [zensys\\_support@sigmadesigns.com](mailto:zensys_support@sigmadesigns.com) for further information.

#### 3.3.1.5 Dev\_Ctrl

The Product\Bin\Dev\_Ctrl directory contains all files needed for running a development controller sample application on a Z-Wave module. The directory contains the following files:

<b>extern_eep.hex</b>	This file contains the external EEPROM data with home ID on the ZM12xxRE/ZM21xxE/ZM31xxC-E module. The
-----------------------	--------------------------------------------------------------------------------------------------------

external EEPROM must only be initialized once.

**dev\_ctrl\_ZW020x\_y.hex**

The compiled and linked development controller sample application for y = ANZ, EU, HK, IN, MY, RU and US frequency versions of the ZW0201 based module.

**dev\_ctrl\_ZW030x\_y.hex**

The compiled and linked development controller sample application for y = ANZ, EU, HK, IN, MY, RU and US frequency versions of the ZW0301 based module.

**3.3.1.6 Dev\_Ctrl\_AVR\_Sec**

Secure development controller sample application binaries not distributed due to export restrictions. The sample application uses an AVR ATmega128 as host on a ZDP02A/ZDP03A Development module. Configure the Z-Wave module on the ZDP02A/ZDP03A Development module with a serial API based portable controller sample application. Contact support via [zensys\\_support@sigmadesigns.com](mailto:zensys_support@sigmadesigns.com) for further information.

**3.3.1.1 DoorBell**

The Product\Bin\DoorBell directory contains all files needed for running a bell sample application on a Z-Wave module. The development controller application is used as button in the doorbell application. The directory contains the following files:

**doorbell\_bell\_ZW020x\_y.hex**

The compiled and linked bell sample application for y = ANZ, EU, HK, IN, MY, RU and US frequency versions of the ZW0201 based module.

**doorbell\_bell\_ZW030x\_y.hex**

The compiled and linked bell sample application for y = ANZ, EU, HK, IN, MY, RU and US frequency versions of the ZW0301 based module.

**3.3.1.2 DoorLock**

The Product\Bin\DoorLock directory contains all files needed for running a doorlock sample application on a Z-Wave module. The directory contains the following files:

**doorlock\_ZW020x\_y.hex**

The compiled and linked bell sample application for y = ANZ, EU, HK, IN, MY, RU and US frequency versions of the ZW0201 based module.

**doorlock\_ZW030x\_y.hex**

The compiled and linked bell sample application for y = ANZ, EU, HK, IN, MY, RU and US frequency versions of the ZW0301 based module.

**3.3.1.3 DoorLock\_Sec**

Secure Secure door lock sample application binaries not distributed due to export restrictions. Contact support via [zensys\\_support@sigmadesigns.com](mailto:zensys_support@sigmadesigns.com) for further information.

**3.3.1.4 LED\_Dimmer**

The Product\Bin\LED\_Dimmer directory contains all files needed for running a LED dimmer sample application on a Z-Wave module. The directory contains the following files:

**leddimmer\_ZW020x\_y.hex**

The compiled and linked LED dimmer sample application for y = ANZ, EU, HK, IN, MY, RU and US frequency versions of the ZW0201 based module.

**leddimmer\_ZW030x\_y.hex**

The compiled and linked LED dimmer sample application for y = ANZ, EU, HK, IN, MY, RU and US frequency versions of the ZW0301 based module.

**3.3.1.5 LED\_Dimmer\_Sec**

Secure LED dimmer sample application binaries not distributed due to export restrictions. Contact support via [zensys\\_support@sigmadesigns.com](mailto:zensys_support@sigmadesigns.com) for further information.

**3.3.1.6 MyProduct**

No hexadecimal files available.

**3.3.1.7 Prod\_Test\_DUT**

The Product\Bin\Prod\_Test\_DUT directory contains all files needed for running a production test DUT sample application on a Z-Wave module. The directory contains the following files:

**prod\_test\_dut\_ZW020x\_y.hex**

The compiled and linked production test DUT sample application for y = ANZ, EU, HK, IN, MY, RU and US frequency versions of the ZW0201 based module.

**prod\_test\_dut\_ZW030x\_y.hex**

The compiled and linked production test DUT sample application for y = ANZ, EU, HK, IN, MY, RU and US frequency versions of the ZW0301 based module.

**3.3.1.8 Prod\_Test\_Gen**

The Product\Bin\Prod\_Test\_Gen directory contains all files needed for running a production test generator sample application on a Z-Wave module. The directory contains the following files:

**prod\_test\_gen\_ZW020x\_y.hex**

The compiled and linked production test generator sample application for y = ANZ, EU, HK, IN, MY, RU and US frequency versions of the ZW0201 based module.

**prod\_test\_gen\_ZW030x\_y.hex**

The compiled and linked production test generator sample application for y = ANZ, EU, HK, IN, MY, RU and US frequency versions of the ZW0301 based module.

### 3.3.1.9 SerialAPI\_Controller\_Bridge\_Nosuc\_Norep\_Noflirs

The Product\Bin\SerialAPI\_Controller\_Bridge\_Nosuc\_Norep\_Noflirs directory contains all files needed for running a serial API based bridge controller sample application without SUC/SIS, repeater and FLiRS BEAM functionality on a Z-Wave module. The directory contains the following files:

<b>extern_eep.hex</b>	This file contains the external EEPROM data with home ID on the ZM12xxRE/ZM21xxE/ZM31xxC-E module. The external EEPROM must only be initialized once.
<b>serialapi_controller_bridge_nosuc_norep_noflirs_ZW020x_y.hex</b>	The compiled and linked serial API based static controller sample application for y = ANZ, EU, HK, IN, MY, RU and US frequency versions of the ZW0201 based module.
<b>serialapi_controller_bridge_nosuc_norep_noflirs_ZW030x_y.hex</b>	The compiled and linked serial API based static controller sample application for y = ANZ, EU, HK, IN, MY, RU and US frequency versions of the ZW0301 based module.
<b>SupportedFunc_SerialAPI_bridge_nosuc_norep_noflirs.txt</b>	Show enabled (1) and disabled (0) serial API calls of released sample application.

### 3.3.1.10 SerialAPI\_Controller\_Bridge\_Nosuc\_Norep\_Noflirs\_Mr

The Product\Bin\SerialAPI\_Controller\_Bridge\_Nosuc\_Norep\_Noflirs\_Mr directory contains all files needed for running a serial API based bridge controller sample application without SUC/SIS, repeater and FLiRS BEAM functionality on a Z-Wave module using manual routing. The directory contains the following files:

<b>extern_eep.hex</b>	This file contains the external EEPROM data with home ID on the ZM12xxRE/ZM21xxE/ZM31xxC-E module. The external EEPROM must only be initialized once.
<b>serialapi_controller_bridge_nosuc_norep_noflirs_mr_ZW020x_y.hex</b>	The compiled and linked serial API based static controller sample application for y = ANZ, EU, HK, IN, MY, RU and US versions of the ZW0201 based module.
<b>serialapi_controller_bridge_nosuc_norep_noflirs_mr_ZW030x_y.hex</b>	The compiled and linked serial API based static controller sample application for y = ANZ, EU, HK, IN, MY, RU and US versions of the ZW0301 based module.
<b>SupportedFunc_SerialAPI_bridge_</b>	Show enabled (1) and disabled (0) serial API calls of released sample

**nosuc\_norep\_noflirs\_mr.txt**

application.

**3.3.1.11 SerialAPI\_Controller\_Installer**

The Product\Bin\SerialAPI\_Controller\_Installer directory contains all files needed for running a serial API based installer controller sample application on a Z-Wave module. The directory contains the following files:

<b>extern_eep.hex</b>	This file contains the external EEPROM data with home ID on the ZM12xxRE/ZM21xxE/ZM31xxC-E module. The external EEPROM must only be initialized once.
<b>serialapi_controller_installer_ZW020x_y.hex</b>	The compiled and linked serial API based static controller sample application for y = ANZ, EU, HK, IN, MY, RU and US versions of the ZW0201 based module.
<b>serialapi_controller_installer_ZW030x_y.hex</b>	The compiled and linked serial API based static controller sample application for y = ANZ, EU, HK, IN, MY, RU and US versions of the ZW0301 based module.
<b>SupportedFunc_SerialAPI_controller_installer.txt</b>	Show enabled (1) and disabled (0) serial API calls of released sample application.

**3.3.1.12 SerialAPI\_Controller\_Installer\_Mr**

The Product\Bin\SerialAPI\_Controller\_Installer\_Mr directory contains all files needed for running a serial API based installer controller sample application on a Z-Wave module using manual routing. The directory contains the following files:

<b>extern_eep.hex</b>	This file contains the external EEPROM data with home ID on the ZM12xxRE/ZM21xxE/ZM31xxC-E module. The external EEPROM must only be initialized once.
<b>serialapi_controller_installer_mr_ZW020x_y.hex</b>	The compiled and linked serial API based static controller sample application for y = ANZ, EU, HK, IN, MY, RU and US versions of the ZW0201 based module.
<b>serialapi_controller_installer_mr_ZW030x_y.hex</b>	The compiled and linked serial API based static controller sample application for y = ANZ, EU, HK, IN, MY, RU and US versions of the ZW0301 based module.
<b>SupportedFunc_SerialAPI_controller_installer_mr.txt</b>	Show enabled (1) and disabled (0) serial API calls of released sample application.



### 3.3.1.13 SerialAPI\_Controller\_Portable

The Product\Bin\SerialAPI\_Controller\_Portable directory contains all files needed for running a serial API based portable controller sample application on a Z-Wave module. The directory contains the following files:

<b>extern_eep.hex</b>	This file contains the external EEPROM data with home ID on the ZM12xxRE/ZM21xxE/ZM31xxC-E module. The external EEPROM must only be initialized once.
<b>serialapi_controller_portable_ZW020x_y.hex</b>	The compiled and linked serial API based static controller sample application for y = ANZ, EU, HK, IN, MY, RU and US versions of the ZW0201 based module.
<b>serialapi_controller_portable_ZW030x_y.hex</b>	The compiled and linked serial API based static controller sample application for y = ANZ, EU, HK, IN, MY, RU and US versions of the ZW0301 based module.
<b>SupportedFunc_SerialAPI_controller_portable.txt</b>	Show enabled (1) and disabled (0) serial API calls of released sample application.

### 3.3.1.14 SerialAPI\_Controller\_Portable\_Mr

The Product\Bin\SerialAPI\_Controller\_Portable\_Mr directory contains all files needed for running a serial API based portable controller sample application on a Z-Wave module using manual routing. The directory contains the following files:

<b>extern_eep.hex</b>	This file contains the external EEPROM data with home ID on the ZM12xxRE/ZM21xxE/ZM31xxC-E module. The external EEPROM must only be initialized once.
<b>serialapi_controller_portable_mr_ZW020x_y.hex</b>	The compiled and linked serial API based static controller sample application for y = ANZ, EU, HK, IN, MY, RU and US versions of the ZW0201 based module.
<b>serialapi_controller_portable_mr_ZW030x_y.hex</b>	The compiled and linked serial API based static controller sample application for y = ANZ, EU, HK, IN, MY, RU and US versions of the ZW0301 based module.
<b>SupportedFunc_SerialAPI_controller_portable_mr.txt</b>	Show enabled (1) and disabled (0) serial API calls of released sample application.

### 3.3.1.15 SerialAPI\_Controller\_Static\_Norep\_Noflirs\_Nomr

The Product\Bin\SerialAPI\_Controller\_Static\_Norep\_Noflirs\_Nomr directory contains all files needed for running a serial API based static controller sample application without repeater, FLiRS BEAM and manual routing functionality on a Z-Wave module. The directory contains the following files:

<b>extern_eep.hex</b>	This file contains the external EEPROM data with home ID on the ZM12xxRE/ZM21xxE/ZM31xxC-E module. The external EEPROM must only be initialized once.
<b>serialapi_controller_static_norep_noflirs_nomr_ZW020x_y.hex</b>	The compiled and linked serial API based static controller sample application for y = ANZ, EU, HK, IN, MY, RU and US versions of the ZW0201 based module.
<b>serialapi_controller_static_norep_noflirs_nomr_ZW030x_y.hex</b>	The compiled and linked serial API based static controller sample application for y = ANZ, EU, HK, IN, MY, RU and US versions of the ZW0301 based module.
<b>SupportedFunc_SerialAPI_controller_static_norep_noflirs_nomr.txt</b>	Show enabled (1) and disabled (0) serial API calls of released sample application.

### 3.3.1.16 SerialAPI\_Controller\_Static\_Nosuc\_Noflirs

The Product\Bin\SerialAPI\_Controller\_Static\_Nosuc\_Noflirs directory contains all files needed for running a serial API based static controller sample application without SUC/SIS and FLiRS BEAM functionality on a Z-Wave module. The directory contains the following files:

<b>extern_eep.hex</b>	This file contains the external EEPROM data with home ID on the ZM12xxRE/ZM21xxE/ZM31xxC-E module. The external EEPROM must only be initialized once.
<b>serialapi_controller_static_nosuc_noflirs_ZW020x_y.hex</b>	The compiled and linked serial API based static controller sample application for y = ANZ, EU, HK, IN, MY, RU and US versions of the ZW0201 based module.
<b>serialapi_controller_static_nosuc_noflirs_ZW030x_y.hex</b>	The compiled and linked serial API based static controller sample application for y = ANZ, EU, HK, IN, MY, RU and US versions of the ZW0301 based module.
<b>SupportedFunc_SerialAPI_controller_static_nosuc_noflirs.txt</b>	Show enabled (1) and disabled (0) serial API calls of released sample application.

### 3.3.1.17 SerialAPI\_Controller\_Static\_Nosuc\_Noflirs\_Mr

The Product\Bin\SerialAPI\_Controller\_Static\_Nosuc\_Mr directory contains all files needed for running a serial API based static controller sample application without SUC/SIS and FLIRS BEAM functionality on a Z-Wave module using manual routing. The directory contains the following files:

<b>extern_eep.hex</b>	This file contains the external EEPROM data with home ID on the ZM12xxRE/ZM21xxE/ZM31xxC-E module. The external EEPROM must only be initialized once.
<b>serialapi_controller_static_nosuc_noflirs_mr_ZW020x_y.hex</b>	The compiled and linked serial API based static controller sample application for y = ANZ, EU, HK, IN, MY, RU and US versions of the ZW0201 based module.
<b>serialapi_controller_static_nosuc_noflirs_mr_ZW030x_y.hex</b>	The compiled and linked serial API based static controller sample application for y = ANZ, EU, HK, IN, MY, RU and US versions of the ZW0301 based module.
<b>SupportedFunc_SerialAPI_controller_static_nosuc_noflirs_mr.txt</b>	Show enabled (1) and disabled (0) serial API calls of released sample application.

### 3.3.1.18 SerialAPI\_Controller\_Static\_Nosuc\_Norep

The Product\Bin\SerialAPI\_Controller\_Static\_Nosuc\_Norep directory contains all files needed for running a serial API based static controller sample application without SUC/SIS and repeater functionality on a Z-Wave module. The directory contains the following files:

<b>extern_eep.hex</b>	This file contains the external EEPROM data with home ID on the ZM12xxRE/ZM21xxE/ZM31xxC-E module. The external EEPROM must only be initialized once.
<b>serialapi_controller_static_nosuc_norep_ZW020x_y.hex</b>	The compiled and linked serial API based static controller sample application for y = ANZ, EU, HK, IN, MY, RU and US versions of the ZW0201 based module.
<b>serialapi_controller_static_nosuc_norep_ZW030x_y.hex</b>	The compiled and linked serial API based static controller sample application for y = ANZ, EU, HK, IN, MY, RU and US versions of the ZW0301 based module.
<b>SupportedFunc_SerialAPI_controller_static_nosuc_norep.txt</b>	Show enabled (1) and disabled (0) serial API calls of released sample application.

### 3.3.1.19 SerialAPI\_Controller\_Static\_Nosuc\_Norep\_PA

The Product\Bin\SerialAPI\_Controller\_Static\_Nosuc\_Norep\_PA directory contains all files needed for running a 300 Series serial API based static controller sample application without SUC/SIS and repeater functionality on a Z-Wave module. In addition, it supports the function ApplicationRFNotify enabling control of an external PA. The directory contains the following files:

<b>extern_eep.hex</b>	This file contains the external EEPROM data with home ID on the ZM12xxRE/ZM21xxE/ZM31xxC-E module. The external EEPROM must only be initialized once.
<b>serialapi_controller_static_nosuc_norep_pa_ZW030x_y.hex</b>	The compiled and linked serial API based static controller sample application for y = ANZ, EU, HK, IN, MY, RU and US versions of the ZW0301 based module.
<b>SupportedFunc_SerialAPI_controller_static_nosuc_norep.txt</b>	Show enabled (1) and disabled (0) serial API calls of released sample application.

### 3.3.1.20 SerialAPI\_Slave\_Enhanced

The Product\Bin\SerialAPI\_Slave\_Enhanced directory contains all files needed for running a serial API based enhanced slave sample application on a Z-Wave module. The directory contains the following files:

<b>extern_eep.hex</b>	This file contains the external EEPROM data with home ID on the ZM12xxRE/ZM21xxE/ZM31xxC-E module. The external EEPROM must only be initialized once.
<b>serialapi_slave_enhanced_ZW020x_y.hex</b>	The compiled and linked serial API based slave sample application for y = ANZ, EU, HK, IN, MY, RU and US versions of the ZW0201 based module.
<b>serialapi_slave_enhanced_ZW030x_y.hex</b>	The compiled and linked serial API based slave sample application for y = ANZ, EU, HK, IN, MY, RU and US versions of the ZW0301 based module.
<b>SupportedFunc_SerialAPI_slave_enhanced.txt</b>	Show enabled (1) and disabled (0) serial API calls of released sample application.

### 3.3.1.21 SerialAPI\_Slave\_Enhanced\_232

The Product\Bin\SerialAPI\_Slave\_Enhanced\_232 directory contains all files needed for running a serial API based enhanced 232 slave sample application on a Z-Wave module. The directory contains the following files:

<b>extern_eep.hex</b>	This file contains the external EEPROM data with home ID on the ZM12xxRE/ZM21xxE/ZM31xxC-E module. The external EEPROM must only be initialized once.
<b>serialapi_slave_enhanced_232_ZW020x_y.hex</b>	The compiled and linked serial API based enhanced 232 slave sample application for y = ANZ, EU, HK, IN, MY, RU and US versions of the ZW0201 based module.
<b>serialapi_slave_enhanced_232_ZW030x_y.hex</b>	The compiled and linked serial API based enhanced 232 slave sample application for y = ANZ, EU, HK, IN, MY, RU and US versions of the ZW0301 based module.
<b>SupportedFunc_SerialAPI_slave_enhanced_232.txt</b>	Show enabled (1) and disabled (0) serial API calls of released sample application.

### 3.3.1.22 SerialAPI\_Slave\_Enhanced\_232\_Mr

The Product\Bin\SerialAPI\_Slave\_Enhanced\_232\_Mr directory contains all files needed for running a serial API based enhanced 232 slave sample application on a Z-Wave module using manual routing. The directory contains the following files:

<b>extern_eep.hex</b>	This file contains the external EEPROM data with home ID on the ZM12xxRE/ZM21xxE/ZM31xxC-E module. The external EEPROM must only be initialized once.
<b>serialapi_slave_enhanced_232_mr_ZW020x_y.hex</b>	The compiled and linked serial API based slave sample application for y = ANZ, EU, HK, IN, MY, RU and US versions of the ZW0201 based module.
<b>serialapi_slave_enhanced_232_mr_ZW030x_y.hex</b>	The compiled and linked serial API based slave sample application for y = ANZ, EU, HK, IN, MY, RU and US versions of the ZW0301 based module.
<b>SupportedFunc_SerialAPI_slave_enhanced_232_mr.txt</b>	Show enabled (1) and disabled (0) serial API calls of released sample application.

### 3.3.1.23 SerialAPI\_Slave\_Enhanced\_232\_Noflirs

The Product\Bin\SerialAPI\_Slave\_Enhanced\_232\_Noflirs directory contains all files needed for running a serial API based enhanced 232 slave sample application on a Z-Wave module without FLiRS functionality. The directory contains the following files:

<b>extern_eep.hex</b>	This file contains the external EEPROM data with home ID on the ZM12xxRE/ZM21xxE/ZM31xxC-E module. The external EEPROM must only be initialized once.
<b>serialapi_slave_enhanced_232_noflirs_ZW020x_y.hex</b>	The compiled and linked serial API based slave sample application for y = ANZ, EU, HK, IN, MY, RU and US versions of the ZW0201 based module.
<b>serialapi_slave_enhanced_232_noflirs_ZW030x_y.hex</b>	The compiled and linked serial API based slave sample application for y = ANZ, EU, HK, IN, MY, RU and US versions of the ZW0301 based module.
<b>SupportedFunc_SerialAPI_slave_enhanced_232_noflirs.txt</b>	Show enabled (1) and disabled (0) serial API calls of released sample application.

### 3.3.1.24 SerialAPI\_Slave\_Enhanced\_Mr

The Product\Bin\SerialAPI\_Slave\_Enhanced\_Mr directory contains all files needed for running a serial API based enhanced slave sample application on a Z-Wave module using manual routing. The directory contains the following files:

<b>extern_eep.hex</b>	This file contains the external EEPROM data with home ID on the ZM12xxRE/ZM21xxE/ZM31xxC-E module. The external EEPROM must only be initialized once.
<b>serialapi_slave_enhanced_mr_ZW020x_y.hex</b>	The compiled and linked serial API based slave sample application for y = ANZ, EU, HK, IN, MY, RU and US versions of the ZW0201 based module.
<b>serialapi_slave_enhanced_mr_ZW030x_y.hex</b>	The compiled and linked serial API based slave sample application for y = ANZ, EU, HK, IN, MY, RU and US versions of the ZW0301 based module.
<b>SupportedFunc_SerialAPI_slave_enhanced_mr.txt</b>	Show enabled (1) and disabled (0) serial API calls of released sample application.

### 3.3.1.25 SerialAPI\_Slave\_Enhanced\_Noflirs

The Product\Bin\SerialAPI\_Slave\_Enhanced\_Noflirs directory contains all files needed for running a serial API based enhanced slave sample application on a Z-Wave module without FLiRS functionality. The directory contains the following files:

<b>extern_eep.hex</b>	This file contains the external EEPROM data with home ID on the ZM12xxRE/ZM21xxE/ZM31xxC-E module. The external EEPROM must only be initialized once.
<b>serialapi_slave_enhanced_noflirs_ZW020x_y.hex</b>	The compiled and linked serial API based slave sample application for y = ANZ, EU, HK, IN, MY, RU and US versions of the ZW0201 based module.
<b>serialapi_slave_enhanced_noflirs_ZW030x_y.hex</b>	The compiled and linked serial API based slave sample application for y = ANZ, EU, HK, IN, MY, RU and US versions of the ZW0301 based module.
<b>SupportedFunc_SerialAPI_slave_enhanced_noflirs.txt</b>	Show enabled (1) and disabled (0) serial API calls of released sample application.

### 3.3.1.26 SerialAPI\_Slave\_Routing

The Product\Bin\SerialAPI\_Slave\_Routing directory contains all files needed for running a serial API based routing slave sample application on a Z-Wave module. The directory contains the following files:

<b>serialapi_slave_routing_ZW020x_y.hex</b>	The compiled and linked serial API based slave sample application for y = ANZ, EU, HK, IN, MY, RU and US versions of the ZW0201 based module.
<b>serialapi_slave_routing_ZW030x_y.hex</b>	The compiled and linked serial API based slave sample application for y = ANZ, EU, HK, IN, MY, RU and US versions of the ZW0301 based module.
<b>SupportedFunc_SerialAPI_slave_routing_noflirs.txt</b>	Show enabled (1) and disabled (0) serial API calls of released sample application.

### 3.3.1.27 SerialAPI\_Slave\_Routing\_Mr

The Product\Bin\SerialAPI\_Slave\_Routing\_Mr directory contains all files needed for running a serial API based routing slave sample application on a Z-Wave module using manual routing. The directory contains the following files:

<b>serialapi_slave_routing_mr_ZW020x_y.hex</b>	The compiled and linked serial API based slave sample application for y = ANZ, EU, HK, IN, MY, RU and US versions of the ZW0201 based module.
<b>serialapi_slave_routing_mr_ZW030x_y.hex</b>	The compiled and linked serial API based slave sample application for y = ANZ, EU, HK, IN, MY, RU and US versions of the ZW0301 based module.
<b>SupportedFunc_SerialAPI_slave_routing_mr.txt</b>	Show enabled (1) and disabled (0) serial API calls of released sample application.

### 3.3.1.28 SerialAPI\_Slave\_Routing\_Noflirs

The Product\Bin\SerialAPI\_Slave\_Routing\_Noflirs directory contains all files needed for running a serial API based routing slave sample application on a Z-Wave module without FLiRS functionality. The directory contains the following files:

<b>serialapi_slave_routing_noflirs_ZW020x_y.hex</b>	The compiled and linked serial API based slave sample application for y = ANZ, EU, HK, IN, MY, RU and US versions of the ZW0201 based module.
<b>serialapi_slave_routing_noflirs_ZW030x_y.hex</b>	The compiled and linked serial API based slave sample application for y = ANZ, EU, HK, IN, MY, RU and US versions of the ZW0301 based module.
<b>SupportedFunc_SerialAPI_slave_routing_noflirs.txt</b>	Show enabled (1) and disabled (0) serial API calls of released sample application.

## 3.3.2 Binary Sensor

The Product\Bin\_Sensor directory contains sample source code for a non-secure/secure binary sensor and non-secure/secure battery operated binary sensor application (see section 4.2 and 4.3).

## 3.3.3 Development Controller

The Product\Dev\_Ctrl directory contains sample source code for the development controller application used on the ZM12xxRE Module mounted on the Z-Wave Development module. For further information refer to section 4.4 and reference [15].



### **3.3.4 Secure Development Controller based on serial API and using an AVR as host**

The Product\Dev\_Ctrl\_AVR\_Sec directory contains sample source code for the secure development controller. The sample application uses an AVR ATmega128 as host on a ZDP02A/ZDP03A Development module. Configure the Z-Wave module on the ZDP02A/ZDP03A Development module with a serial API based portable controller sample application. For further information refer to section 4.5 and reference [15].

### **3.3.5 Doorbell**

The Product\DoorBell directory contains sample source code for the doorbell application used on the Z-Wave Interface module. Use the Development Controller application to control the doorbell application. For further information, refer to section 4.6.

### **3.3.6 Door Lock**

The Product\DoorLock directory contains sample source code for the non-secure and secure door lock application used on the Z-Wave Interface module. Use the Secure Development Controller application to control the door lock application. For further information, refer to section 4.7.

### **3.3.7 LED Dimmer**

The Product\LED\_Dimmer directory contains sample source code for the non-secure and secure dimmer application on a Z-Wave module, which uses the LEDs to simulate a light switch with a built in dimmer. For further information, refer to section 4.8.

### **3.3.8 MyProduct**

The Product\MyProduct directory contains sample source code for a routing slave application on a Z-Wave module. For further information, refer to section 4.9.

### **3.3.9 Production Test DUT**

The Product\Prod\_Test\_DUT directory contains sample source code for a production test DUT application on a Z-Wave module. For further information, refer to section 4.10.

### **3.3.10 Production Test Generator**

The Product\Prod\_Test\_Gen directory contains sample source code for a production test generator application on a Z-Wave module. For further information, refer to section 4.11.

### **3.3.11 Serial API**

The Product\SerialAPI directory contains sample source code for the Serial API sample applications. For further information about the Serial API, refer to section 4.12.

### **3.3.12 Utilities**

The Product\util\_func directory contains some helpful functions that are used by several of the sample applications.

### **8051\_AES\_common.a51**

These files contain shared data and functions for AES128 and functions

**8051\_AES\_core.a51**

for AES128 encryption/decryption. Files are not distributed on the Developer's Kit CD due to export restrictions.

The sample applications uses a 3<sup>rd</sup> party product subjected to intellectual property (IP) rights and licensing issues. The files can be acquired from the rightful IP owner (Stiftung Secure Information and Communication Technologies (SIC) in Austria) for approximately €2.000:

[http://jce.iaik.tugraz.at/sic/sales/price\\_list/hardware\\_related\\_products](http://jce.iaik.tugraz.at/sic/sales/price_list/hardware_related_products)

Alternatively, implement the functions based on Atmel's Application Note "AVR231: AES Bootloader":

[http://www.atmel.com/dyn/resources/prod\\_documents/doc2589.pdf](http://www.atmel.com/dyn/resources/prod_documents/doc2589.pdf)

**AES\_module.h**

This header file contains definitions for implementing secure communication using AES as encrypting/decrypting engine.

**association.c  
association.h**

The files contain code that shows how association between nodes could be implemented on a Z-Wave module. This code holds all associations in RAM and the number of nodes/groupings possible using this implementation is limited.

Applications using this collection of functions must implement three functions (ApplicationStoreAll, ApplicationInitAll, ApplicationClearAll). These should handle the storage in nonvolatile memory if this is desired.

**battery.c  
battery.h**

The files contain code that shows how battery operated devices may implement power down, wake up notification and network update requests. Applications using this collection must call the following functions at their appropriate location:

*UpdateWakeupCount* – call from ApplicationInitSW to update the wakeup counter which determines the wakeup interval on application level (200-series) – Only called when Wakeupreason is WUT-Kicked.  
*InitRTCActionTimer* – call from ApplicationInitSW, to activate the RTC timer. (100-Series)

*HandleWakeupFrame* – call from ApplicationCommandHandler to handle incoming COMMAND\_CLASS\_WAKE\_UP is received. Handles WAKE\_UP\_INTERVAL\_GET/SET/NO\_MORE\_INFORMATION.

*SetDefaultBatteryConfiguration* – is called from ApplicationInitHW when node is reset, and from SetDefaultConfiguration. Sets the default values for powerdown timeout, sleep time and networkupdate.

*LoadBatteryConfiguration* – call from LoadConfiguration. Loads the battery related information from EEPROM and make them available for the running application.

*SaveBatteryConfiguration* – call from SaveConfiguration. Saves the battery related information to EEPROM.

*StartPowerDownTimer* – call from ApplicationInitSW and set as callback function ZW\_SEND\_DATA methods after which the node should enter sleep mode.

Please refer to the BatterySensor sample application for an example on how this can be implemented.

<b>ctrl_learn.h</b> <b>ctrl_learn.c</b>	The files contain code for how to handle learn mode on controller nodes.
<b>one_button.c</b> <b>one_button.h</b>	Enables easy use of a button. The functions detect whether a button has been pressed shortly or is being held. To initialise the button detection, run <code>OneButtonInit()</code> from <code>ApplicationInitSW</code> . And call <code>OneButtonLastAction</code> when button information is needed (e.g. in <code>ApplicationPoll()</code> ).
<b>self_heal.c</b> <b>self_heal.h</b>	Support functions to implement Lost / Self Heal functionality. This file is mandatory if the battery helper functions are used and <code>ZW_SELF_HEAL</code> is defined. See the <code>battery.c</code> and <code>bin_sensor.c</code> source files for help on using the functions.
<b>slave_learn.h</b> <b>slave_learn.c</b>	The files contain code for how to handle learn mode on slave nodes. These two files are used by all slave based code in the ZDK. The sample application should just call <code>StartLearnModeNow()</code> to enter learnmode and transmit nodeinformation. Inclusion uses normal power. The sample application should then wait for the <code>BOOL</code> <code>learnState</code> to go <code>FALSE</code> before doing transmissions.
<b>ZW_AES128.h</b>	This header file contains definitions for the security solution on application level.
<b>ZW_FLiRS.c</b> <b>ZW_FLiRS.h</b>	The files contain code for how to handle FLiRS nodes.
<b>ZW_Security_AES_module.c</b> <b>ZW_Security_AES_module.h</b>	The files contain code for the functionality supporting secure communication using AES as encryption/decryption mechanism.
<b>ZW_TransportLayer.h</b>	Transport layer type selector
<b>ZW_TransportNative.h</b>	Implements functions for transporting frames over the native Z-Wave Network.
<b>ZW_TransportSecurity.h</b> <b>ZW_TransportSecurity.c</b>	Implements functions for transporting frames over the secure Z-Wave Network.

### 3.4 Tools

The Tools directory contains various tools needed for building and debugging the sample applications. All tools in this directory can freely be used for building Z-Wave applications.

### 3.4.1 ERTT

This directory contains the PC software and the embedded code for the Enhanced Reliability Test Tool (ERTT). For further details, refer to [11].

The ERTT directory contains the following files:

**PC\setup.exe**

PC application.

**Z-Wave\_Firmware\extern\_eep.hex**

This file contains the external EEPROM data on the ZM12xxRE/ZM21xxE/ZM31xxC-E module. Must initialize the external EEPROM once.

**Z-Wave\_Firmware\serialapi\_controller\_single\_ZW020x\_y.hex**

The compiled and linked ERTT application for y = ANZ, EU, HK and US frequency versions of the ZW0201 based module.

**Z-Wave\_Firmware\serialapi\_controller\_single\_ZW030x\_y.hex**

The compiled and linked ERTT application for y = ANZ, EU, HK and US frequency versions of the ZW0301 based module.

### 3.4.2 IncDep

This directory contains a python script that is used for making dependency files when building the sample applications.

### 3.4.3 Intel UPnP

This directory contains Intel's tools for UPnP technology helping software developers during development, testing, and deployment of UPnP-compliant devices. These tools are used in conjunction with the Z-Wave to UPnP bridge sample applications [8].

### 3.4.4 Make

This directory contains a DOS/Windows version of the GNU make utility. The make utility is used for building the sample applications.

### 3.4.5 Mergehex

This directory contains a tool used for merging two files in Intel hex format. The tool is used for building external EEPROM files in the code.

### 3.4.6 Programmer

This Programmer directory contains the PC software and ATmega128 firmware for the Programmer. For further details, refer to [14].

The Programmer directory contains the following files:

<b>PC\setup.exe</b>	Programmer application.
<b>PC\CP210x_VCP_Win_XP_S2K3_Vista_7.exe</b>	The CP210x USB to UART Bridge Virtual COM Port (VCP) driver. This driver supports Windows XP/2003/Vista(32/64)/7(32/64).
<b>PC\Source\...</b>	ZWaveProgrammer PC source code providing Windows GUI and interface to ATmega128 situated on the ZDP02A/ZDP03A Development Platform. For further details regarding communication protocol interface, refer to [34].
<b>ZDP0xA_Firmware\ATmega128_Firmware.hex</b>	The compiled and linked Z-Wave Programmer bootloader for the ATmega128 situated on the ZDP02A/ZDP03A Development Platform.
<b>ZDP0xA_Firmware\ZWaveProgrammer_FW.hex</b>	The compiled and linked Z-Wave Programmer firmware for the ATmega128 situated on the ZDP02A/ZDP03A Development Platform.
<b>ZDP0xA_Firmware\Source\...</b>	Z-Wave Programmer firmware source code for the ATmega128 situated on the ZDP02A/ZDP03A Development Platform. For further details regarding communication protocol interface, refer to [34].
<b>SD3402_Calibration\SD3402_Calibration.hex</b>	SD3402 crystal calibration firmware used by the calibration box, refer to [36]. ZM4101 and ZM4102 are already calibrated during production.

### 3.4.7 PVT and RF Regulatory

This directory contains software used in connection with PVT and RF regulatory measurements on the hardware. For a guideline how to carry out the measurements, refer to [9] and [19].

The PVT\_and\_RF\_regulatory directory contains the following files:

<b>ZW0201_rx_y.hex</b>	Puts the ZW0201 in receive mode for y = ANZ, EU, HK, IN, MY, RU and US frequency versions of the ZW0201 based products.
<b>ZW0201_TXcar_y.hex</b>	ZW0201 constantly transmits a carrier y = 921.42MHz (ANZ) or 868.42MHz (EU) or 919.82MHz (HK) or 865.22MHz (IN) or 868.10MHz (MY) or 869.0MHz (RU) or 908.42MHz (US).
<b>ZW0201_TXmod_y.hex</b> <b>ZW0201_TXmod_40kbps_y.hex</b>	ZW0201 constantly transmits a modulated signal y +/- 25kHz, where y = 921.42MHz (ANZ) or 868.42MHz (EU) or 919.82MHz (HK) or 865.22MHz (IN) or 868.10MHz (MY) or 869.0MHz (RU) or 908.42MHz (US).
<b>ZW0301_rx_y.hex</b>	Puts the ZW0301 in receive mode. y = ANZ, EU, HK, IN, MY, RU and US frequency versions of the ZW0301 based products.
<b>ZW0301_TXcar_y.hex</b>	ZW0301 constantly transmits a carrier y = 921.42MHz (ANZ) or 868.42MHz (EU) or 919.82MHz (HK) or 865.22MHz (IN) or 868.10MHz (MY) 868.42MHz or 908.42MHz (US).
<b>ZW0301_TXmod_y.hex</b> <b>ZW0301_TXmod_40kbps_y.hex</b>	ZW0301 constantly transmits a modulated signal y +/- 25kHz, where y = 921.42MHz (ANZ) or 868.42MHz (EU) or 919.82MHz (HK) or 865.22MHz (IN) or 868.10MHz (MY) 868.42MHz or 908.42MHz (US).

### 3.4.8 Python

This directory contains a python scripting language interpreter. Python is used for various purposes in the code build process.

### 3.4.9 TextTools

This directory contains the sed stream editor used to modify text strings during the make process.

### 3.4.10 XML Editor

This directory contains the XML Editor program; the program can be used to define approved Z-Wave device and command classes used by the application layer of the Z-Wave protocol. The XML file can be used by the Ziffer for interpretation of the device and command classes. The customer can also define device and command classes under development or proprietary command class structures enabling interpretation by the Ziffer.

Beside a XML file containing all the information, it is also possible to generate a C# class file and C header file as foundation for Z-Wave application development. For further details, refer to [28].

The XML Editor directory contains the following files:

<b>PC\setup.exe</b>	XML Editor application.
<b>PC\Setup.msi</b>	

### 3.4.11 Ziffer

This directory contains the Ziffer program; the program is a development tool for capturing Z-Wave RF communication and presenting the frames in a graphical user interface on a PC. The tool shows the node ID of the Source and Destination for the communication, the type of frame being sent, and the application content, i.e. the specific command, which is being sent.

The Ziffer tool is a passive "listener" to the Z-Wave network traffic, and will only display the RF communications taking place within direct RF range. Be also aware that Ziffer can occasionally miss RF communication even from Z-Wave nodes within direct range.

The tool consists of two parts, an embedded part that should be downloaded to a Z-Wave module and a PC application that should run on a PC attached to the Z-Wave module via the serial interface. For further details, refer to [12].

The Ziffer directory contains the following files:

<b>PC\setup.exe</b>	Ziffer application supporting Windows
<b>PC\ZifferSetup.msi</b>	XP/2003/Vista(32/64)/7(32/64)
<b>PC\FileConverter\setup.exe</b>	FileConverter enable Ziffer to automatically
<b>PC\FileConverter\FileConverterSetup.msi</b>	convert old file formats *.znf to latest *.zlf when
	opening file.
<b>Z-Wave_Firmware\sniffer_ZW040x.hex</b>	Ziffer application supporting ANZ, EU, HK, IN,
	JP, MY, RU and US versions on a 400 Series
	based module.
<b>Z-Wave_Firmware\sniffer_ZW030x_y.hex</b>	Ziffer application supporting y = ANZ, EU, HK, IN,
	MY, RU and US frequency versions on a ZW0301
	based module.
<b>Z-Wave_Firmware\sniffer_ZW020x_y.hex</b>	Ziffer application supporting y = ANZ, EU, HK, IN,
	MY, RU and US frequency versions on a ZW0201
	based module.

### 3.5 PC

The PC directory contains three PC sample applications demonstrating the use of the Z-Wave DLL and Serial API.

#### 3.5.1 Bin

The PC\Bin directory contains the program or installation executables of the PC sample applications.

<b>ZWaveDll\setup.exe</b>	Executables of the Z-Wave DLL used by the PC sample applications.
<b>ZWaveDll\ZWaveSetup.msi</b>	
<b>ZWaveInstaller\setup.exe</b>	Executables of the Z-Wave Installer Tool sample application.
<b>ZWaveInstaller\ZWaveInstallerToolSetup.msi</b>	
<b>ZWavePCController\Non-secure\setup.exe</b>	Executables of the Z-Wave Non-secure PC Controller sample application.
<b>ZWavePCController\Non-secure\ZWaveControllerSetup.msi</b>	
<b>ZWavePCController\Secure\setup.exe</b>	Executables of the Z-Wave Secure PC Controller sample application. Binaries not distributed due to export restrictions. Contact support via <a href="mailto:zensys_support@sigmadesigns.com">zensys_support@sigmadesigns.com</a> for further information.
<b>ZWavePCController\Secure\ZWaveControllerSetup.msi</b>	
<b>ZWaveUPnPBridge\setup.exe</b>	Executables of the Z-Wave to UPnP Bridge sample application.
<b>ZWaveUPnPBridge\ZWaveUPnPBridgeSetup.msi</b>	

#### 3.5.2 Source

The PC\Source directory contains the C# source code of the PC sample applications using the Microsoft Visual Studio 2008 environment.

##### 3.5.2.1 Libraries

The PC\Source\Libraries directory contains various libraries used by the PC sample applications.

###### 3.5.2.1.1 WinForms UI

The PC\Source\Libraries\WinFormsUI directory contains C# source code of the windows docking library.

<b>WinFormsUI.csproj</b>	Microsoft Visual Studio 2008 project file containing information at the project level and used to build the project.
--------------------------	----------------------------------------------------------------------------------------------------------------------

###### 3.5.2.1.2 Zensys Framework

The PC\Source\Libraries\ZensysFramework directory contains C# source code of the additional functions, formatters, helpers.

<b>ZensysFramework.csproj</b>	Microsoft Visual Studio 2008 project file containing information at the project level and used to build the project.
-------------------------------	----------------------------------------------------------------------------------------------------------------------



### 3.5.2.1.3 Zensys Framework UI

The PC\Source\Libraries\ZensysFrameworkUI directory contains C# source code of the completed Z-Wave UI elements that can be reused in applications:

- Associations View Control;
- Bridged UPnP Device View Control;
- Controller View Control;
- Node View Control;
- UPnP Binary Light Device View Control;
- UPnP Device Scanner View Control;
- UPnP Media Renderer View Control.

**ZensysFrameworkUI.csproj** Microsoft Visual Studio 2008 project file containing information at the project level and used to build the project.

### 3.5.2.1.4 Zensys Framework UI Controls

The PC\Source\Libraries\ZensysFrameworkUIControls directory contains C# source code of the additional UI elements such as:

- ListDataView;
- TreeDataView;
- BitBox;
- ThreadSafeLabel.

**ZensysFrameworkUIControls.csproj** Microsoft Visual Studio 2008 project file containing information at the project level and used to build the project.

### 3.5.2.1.5 Z-Wave Command Class

The PC\Source\Libraries\ZWaveCommandClasses directory contains C# source code for the XML parser, which enables parsing of Z-Wave frames by the Zniffer and generating frames by the PC based applications.

**ZWaveCommandClasses.csproj** Microsoft Visual Studio 2008 project file containing information at the project level and used to build the project.

### 3.5.2.1.6 Z-Wave DLL

The PC\Source\Libraries\ZWaveDll directory contains C# source code of the dynamic link library used by the PC application to communicate with the ZW0102/ZW0201/ZW0301 based module via the serial API interface. Refer to [13] for further details.

**SerialZWaveDll.sln** Microsoft Visual Studio 2008 solutions file containing information at the project level and used to build the project.

### 3.5.2.1.7 Z-Wave HAL

The PC\Source\Libraries\ZWaveHAL directory contains C# source code of the ZWave High-level Application Layer in terms of ZWave DLL architecture. It contains common functions that are used in Z-Wave enabled PC applications: ZWavePCController, ZWaveProgrammer, ZWaveUPnPBridge etc. Refer to [13] for further details.

**SerialZWaveHAL.sln**

Microsoft Visual Studio 2008 solutions file containing information at the project level and used to build the project.

### 3.5.2.2 Sample Application

The PC\Source\SampleApplications contains the various the PC applications

#### 3.5.2.2.1 Z-Wave Installer

The PC\Source\SampleApplications\ZWaveInstaller directory contains C# sample source code for a PC based installer tool using the Z-Wave DLL etc. Further reading on how to use the PC based Installer Tool see [7].

**ZWaveInstallerTool.sln**

Microsoft Visual Studio 2008 solutions file containing information at the project level and used to build the project.

#### 3.5.2.2.2 Z-Wave PC Controller

The PC\Source\SampleApplications\ZWavePCController directory contains C# sample source code for a PC based controller using the Z-Wave DLL etc. Further reading on how to use the PC based Controller see [6].

**ZWaveController.sln**

Microsoft Visual Studio 2008 solutions file containing information at the project level and used to build the project.

#### 3.5.2.2.3 Z-Wave UPnP Bridge

The PC\Source\SampleApplications\ZWaveUPnPBridge directory contains C# sample source code for a PC based Z-Wave Bridge using the Z-Wave DLL etc. Further readings on how to use the Z-Wave UPnP Bridge see [8].

**ZWaveUPnPBridge.sln**

Microsoft Visual Studio 2008 solutions file containing information at the project level and used to build the project.

## 4 APPLICATION CODE

The Z-Wave Developer's Kit includes several sample applications: a serial controller application, a LED dimmer application, a binary sensor and a battery operated binary sensor application for the Z-Wave module. The sample application realizes a light control system to help the developer to understand how the various components can interact. In addition the Z-Wave Developer's Kit also comprises of a number of PC centric sample applications for displaying advanced functionalities of the Z-Wave protocol:

- How a Z-Wave Module can be controlled from a PC.
- Installation including display of network topology.
- Bridging to and from other networks.

### 4.1 Building ZW0x0x Code

All the sample applications for the ZW0201/ZW0301 contains source code and make files that allows the developer to modify and compile the applications without a lot of make file configuration. All sample applications are built by calling the MK.BAT file that is located in the sample application directory.

#### 4.1.1 MK.BAT

This batch-file calls the make tool which then, via the makefiles, builds the sample application to the different RF frequencies (ANZ/EU/HK/IN/MY/RU/US) used when transmitting/receiving and Z-Wave module targets. Three environment variables must be defined before it is possible to build the sample applications. The procedure is on a PC using Windows XP as follows:

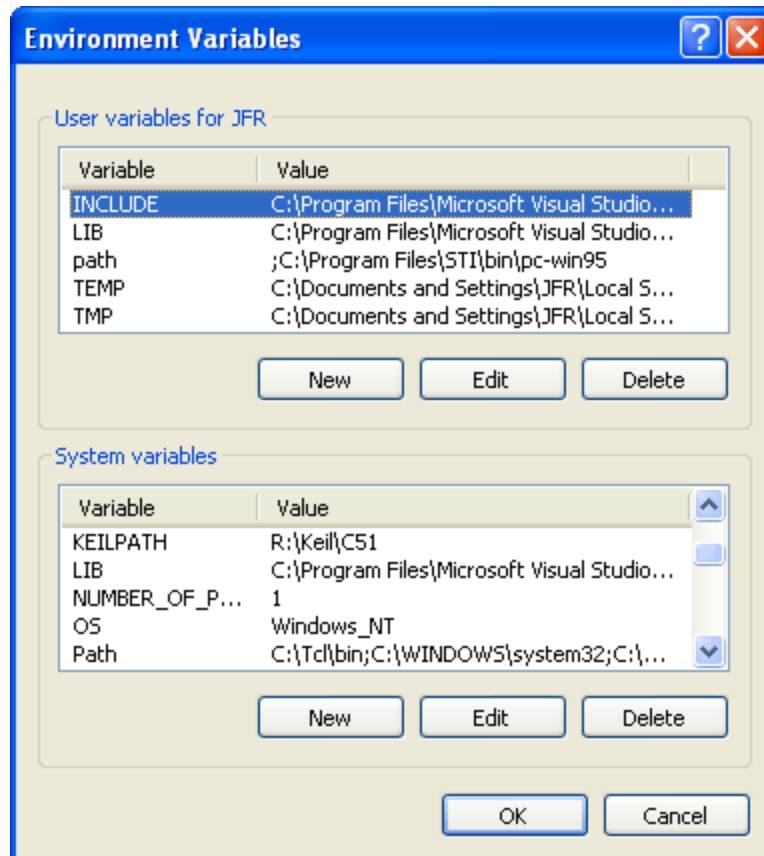


Figure 1. Configuring environment variables

1. Select **Start, Control Panel** and **System**
2. Select **Advanced** tab and activate the **Environment Variables** button
3. Under **System variables** activate the **New** button
4. In the **Variable name** textbox enter **KEILPATH** (use capital letters because Windows XP is case sensitive)
5. In the **Variable value** textbox enter **C:\KEIL\C51** and activate the **OK** button
6. Under **System variables** activate the **New** button
7. In the **Variable name** textbox enter **TOOLSDIR** (use capital letters because Windows XP is case sensitive)
8. In the **Variable value** textbox enter **C:\Devkit\_5\_00\TOOLS** and activate the **OK** button

Afterwards open a command prompt (DOS box) in the relevant sample application directory to build the application.



**Figure 2. Building sample applications**

Remember to use upper case in **KEILPATH**, **KEIL\_LOCAL\_PATH** and **TOOLS DIR** when using Windows XP, because this operating system is case sensitive. If the environment variables are not defined then MK.BAT will prompt the user to define them.

Opening a command prompt to a particular directory from Explorer is enabled in the following way:

1. Start Regedit
2. Go to HKEY\_CLASSES\_ROOT \ Directory \ shell
3. Create a new key called *Command*
4. Give it the value of the name you want to appear in the Explorer. Something like *Open DOS Box*
5. Under this create a new key called *command*
6. Give it a value of *cmd.exe /k "cd %L"*
7. Now when you are in the Explorer, right click on a folder, select *Open DOS Box*, and a command prompt will open to the selected directory.

The batch file MK.BAT builds all versions with respect to targets and RF frequencies. The wanted target can also be entered as a parameter on the command line. The figure below displays the possible targets for a given product.

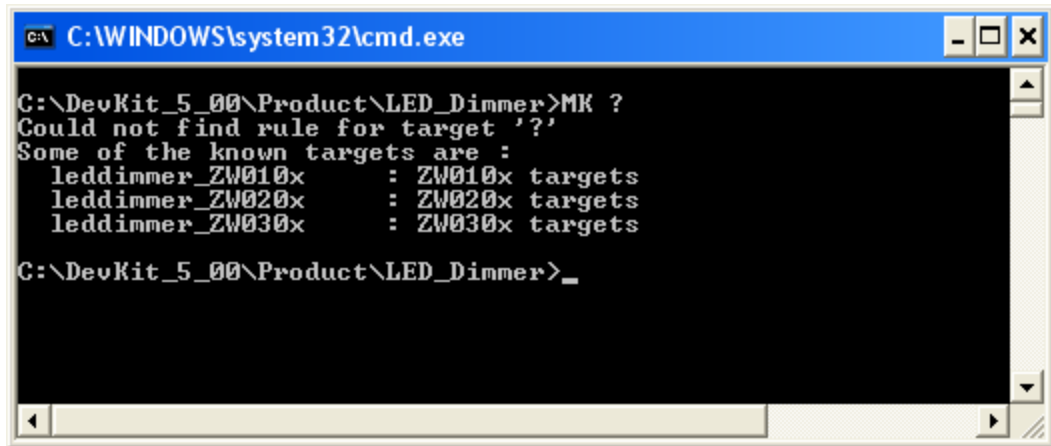


Figure 3. Possible sample application targets

Remember to enter the targets as shown when using Windows XP, because this operating system is case sensitive.

When MK.BAT is executed the following directory structure is created within the source code directory

For ZW0201 targets:

- <application>
  - build
    - <application>\_ZW020x\_ANZ
      - list - contains list files
      - Rels - contains object files and map file
      - <application>\_ZW020x\_ANZ.hex - flash Intel hex file for ANZ ZW0201 based module
    - <application>\_ZW020x\_EU
      - list - contains list files
      - Rels - contains object files and map file
      - <application>\_ZW020x\_EU.hex - flash Intel hex file for EU ZW0201 based module
    - <application>\_ZW020x\_HK
      - list - contains list files
      - Rels - contains object files and map file
      - <application>\_ZW020x\_HK.hex - flash Intel hex file for HK ZW0201 based module
    - <application>\_ZW020x\_IN
      - list - contains list files
      - Rels - contains object files and map file
      - <application>\_ZW020x\_IN.hex - flash Intel hex file for IN ZW0201 based module
    - <application>\_ZW020x\_MY
      - list - contains list files
      - Rels - contains object files and map file
      - <application>\_ZW020x\_MY.hex - flash Intel hex file for MY ZW0201 based module
    - <application>\_ZW020x\_RU
      - list - contains list files
      - Rels - contains object files and map file
      - <application>\_ZW020x\_RU.hex - flash Intel hex file for RU ZW0201 based module
    - <application>\_ZW020x\_US
      - list - contains list files

- Rels - contains object files and map file
- <application>\_ZW020x\_US.hex - flash Intel hex file for US ZW0201 based module

For ZW0301 targets:

- <application>
  - build
    - <application>\_ZW030x\_ANZ
      - list - contains list files
      - Rels - contains object files and map file
      - <application>\_ZW030x\_ANZ.hex - flash Intel hex file for ANZ ZW0301 based module
    - <application>\_ZW030x\_EU
      - list - contains list files
      - Rels - contains object files and map file
      - <application>\_ZW030x\_EU.hex - flash Intel hex file for EU ZW0301 based module
    - <application>\_ZW030x\_HK
      - list - contains list files
      - Rels - contains object files and map file
      - <application>\_ZW030x\_HK.hex - flash Intel hex file for HK ZW0301 based module
    - <application>\_ZW030x\_IN
      - list - contains list files
      - Rels - contains object files and map file
      - <application>\_ZW030x\_IN.hex - flash Intel hex file for IN ZW0301 based module
    - <application>\_ZW030x\_MY
      - list - contains list files
      - Rels - contains object files and map file
      - <application>\_ZW030x\_MY.hex - flash Intel hex file for MY ZW0301 based module
    - <application>\_ZW030x\_RU
      - list - contains list files
      - Rels - contains object files and map file
      - <application>\_ZW030x\_RU.hex - flash Intel hex file for RU ZW0301 based module
    - <application>\_ZW030x\_US
      - list - contains list files
      - Rels - contains object files and map file
      - <application>\_ZW030x\_US.hex - flash Intel hex file for US ZW0301 based module

#### 4.1.2 Makefiles

##### Makefile

This file is part of the make job. It creates the directory structure and defines the build targets and calls the other make files in the build depending on the target.

**NOTE:** The Makefile might contain test or debug targets that are not build in the default build.

##### Product\Common directory

The common directory contains a set of standard make files that are used to build all the sample application. The make files define the compiler and linker options used to build all Z-Wave applications and the correct defines for all the different library types. The following compiler control line defines are used by the common makefiles:

ZW\_CONTROLLER Basic controller

ZW_CONTROLLER_32	Adding 32KB flash and controller functionality
ZW_CONTROLLER_STATIC	Adding static controller functionality
ZW_INSTALLER	Adding installer controller functionality
ZW_CONTROLLER_BRIDGE	Adding bridge controller functionality
ZW_SLAVE	Basic slave
ZW_SLAVE_32	Adding 32KB flash and enhanced slave functionality
ZW_SLAVE_ROUTING	Adding routing slave functionality
ZW010x	Basic 100 Series ASIC functionality
ZW0102	ZW0102 ASIC specific functionality
ZW020x	Basic 200 Series ASIC functionality
ZW0201	ZW0201 ASIC specific functionality
ZW030x	Basic 300 Series ASIC functionality
ZW0301	ZW0301 ASIC specific functionality
NEW_NODEINFO	Supports all Node Information Frame formats



## 4.2 Binary Sensor Code

The Developer's Kit contains code for a non-secure and secure binary sensor. This device is in effect a binary sensor where the sensor input is the pin also used as a button input on the device module. The Bin\_Sensor will on every button release transmit a basic set frame to any associated devices. Pressing the button a little while instead a node Info frame will be transmitted. A static controller such as the one described in [6] can control, configure and assign routes to the Bin\_Sensor.

The Bin\_Sensor is a binary sensor that supports the association command class described in the device class specification (see ref [1]). This device complies with the specific device class named routing binary sensor device class (4.1).

The none-secure Binary Sensor application lists the following supported command classes in the Node Information Frame:

- Binary Sensor command class
- Association command class
- Version command class
- Manufacturer Specific command class

When included non-secure the Secure Binary Sensor application lists the following supported command classes in the Node Information Frame:

### Non-Secure Included

- Binary Sensor command class
- Association command class
- Version command class
- Manufacturer Specific command class
- Security command class

### Secure Included

When included secure the Secure Binary Sensor application lists the following supported command classes in the Node Information Frame:

- Version command class
- Manufacturer Specific command class
- Security command class

The following listed in the Security Commands Supported Report frame:

- Binary Sensor command class
- Association command class
- Manufacturer Specific command class
- Version command class

The Basic command class is secure because application does not list it in Node Information Frame.

The Bin\_Sensor is a slave device based on the enhanced slave API. During initialization, the Bin\_Sensor will initialize the mounted button, the 4 LED's and a timer function that handles the button input and sensor input (in this example the same as the button input). It will also get stored data from the non-volatile memory. After the initialization, the Z-Wave basis software will continually call the **ApplicationPoll** function, which contains the Bin\_Sensor main function. The **ApplicationPoll** function checks if the button or the sensor input has changed state and then acts accordingly to the current state the Bin\_Sensor is in. The other main function is the **ApplicationCommandHandler** function that is called every time a command has been received, destined for the Bin\_Sensor. This function checks the command and acts according to the command. When transmitting the Bin\_Sensor will, if routes have been assigned use these.

The Bin\_Sensor implements Lost functionality and network topology maintenance by using a series of methods. If the device is unsuccessful in sending a message a predefined count it will enter lost state, and attempt to find a SUC in the network, and if successful ask the SUC for routes to the failing devices. At regular intervals, the Bin\_Sensor will transmit a Static Route Request, which asks the SUC for any updates done to the network.

#### 4.2.1 Network Wide Inclusion

By default the Bin\_Sensor will enter network wide inclusion (NWI) to be added to a network when it is powered up and have not already been included Bin\_Sensor. The Bin\_Sensor will stay in NWI mode for 30 minutes or until it has been included into the network. Any key press will terminate the NWI mode and the only way to make the Bin\_Sensor enter NWI mode again is by doing hardware reset either by remove and reapply the power or press the reset button on the side of the board.

The figure below show the NWI flow diagram implemented on application level for a slave. The slave\_learn.c file contains the implementation situated in the util\_func directory.

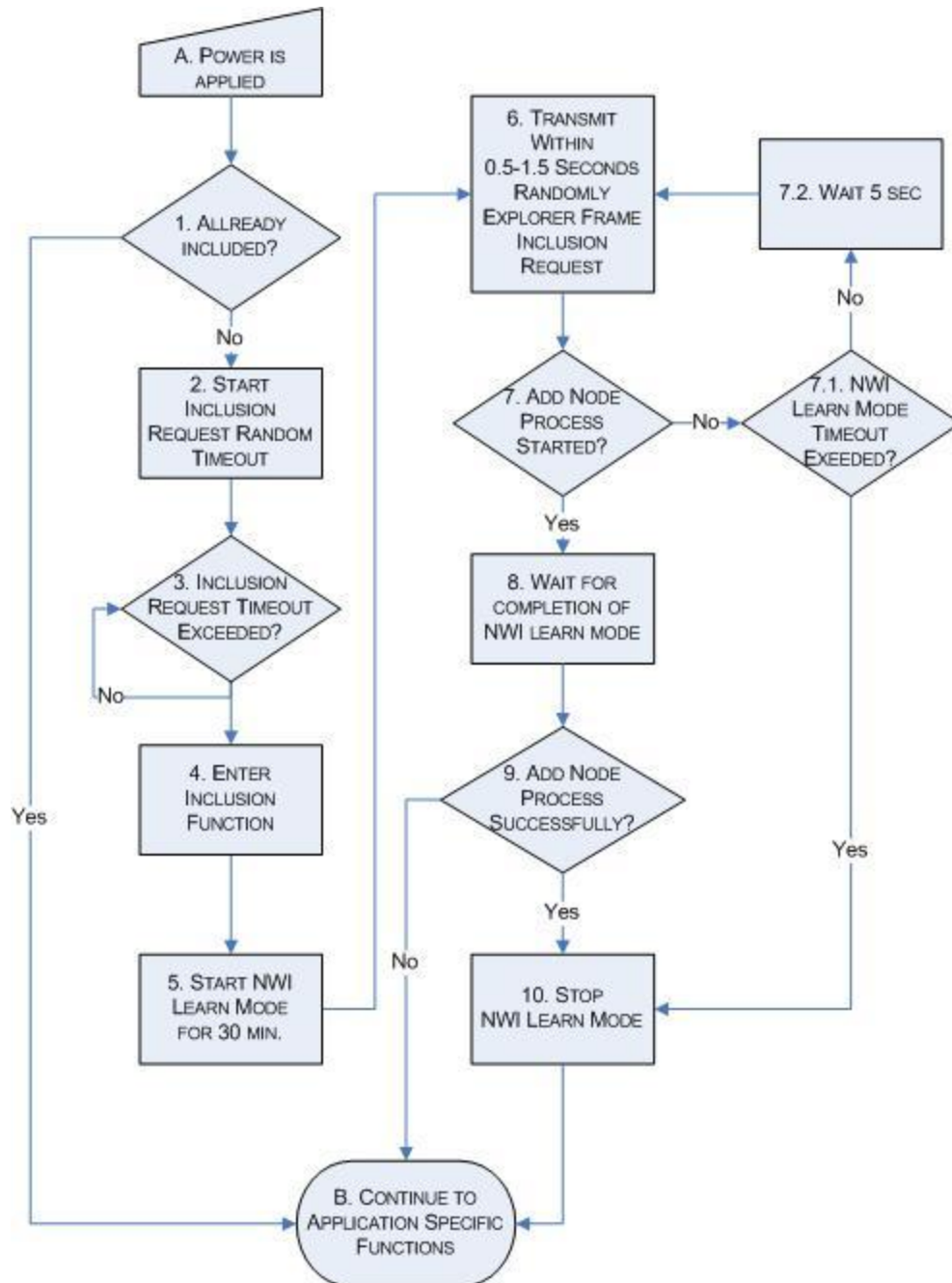


Figure 4. NWI flow diagram for a slave

### 4.2.2 Interface

The following table defines the functionality of the button on the Z-Wave module.

	Button Triple Clicked	Button Clicked
<b>In Network</b>	Node Info Frame / Enter learn mode	Basic Set (Broadcast)
<b>Not in Network</b>	Node Info Frame / Enter learn mode	None
<b>Associated</b>	Node Info Frame / Enter learn mode	Basic Set (to associated nodes)

Learn mode is now activated by pressing the button three times within 1.5 seconds to avoid unintentional inclusion/exclusion of the node.

### 4.2.3 Bin\_Sensor Files

The Product\Bin\_Sensor directory contains sample source code for a non-secure/secure binary sensor and a non-secure/secure battery powered binary sensor slave application on a Z-Wave module. The application uses also a number of utility functions described in section 3.3.12.

#### MK.BAT

Batch file used to build ZW0201/ZW0301 based sample applications in ANZ, EU, HK, IN, MY, RU and US versions respectively. Refer to chapter 4.1 regarding details about the build procedure.

#### Makefile

This file is part of the make job. It creates the directory structure and defines targets. The following compiler control line defines are used in the makefiles:

```

ANZ          : Build ANZ frequency target.
EU           : Build EU frequency target.
HK           : Build HK frequency target.
IN           : Build IN frequency target.
MY           : Build MY frequency target.
RU           : Build RU frequency target.
US           : Build US frequency target.
```

#### Makefile.binsensor\_common

This makefile is a common file defining all dependencies etc.

#### eeeprom.h

This header file defines the addresses where application data are stored in the external EEPROM.

## **Bin\_Sensor.h / Bin\_Sensor.c**

These files contain the source code for the binary sensor application state machine. The common API functions such as **ApplicationInitHW**, **ApplicationInitSW**, **ApplicationNodeInformation**, **ApplicationPoll**, **ApplicationSlaveUpdate** and **ApplicationCommandHandler** are defined here.

## **Battery\_Sensor.mpw / Battery\_Sensor\_....Uv2**

uVision3 multiproject workspace file (\*.mpw) that unifies all the \*.Uv2 project files. There is a project file for every particular target with respect to single chip series and frequency.

### **4.2.3.1        Macros for accessing the LED's**

#### **LED\_ON(led)**

Turn LED on.

Parameter:

led - LED number

Example:

```
PIN_OUT(LED1); /* define LED1 as an output pin */  
LED_ON(1);     /* turn LED 1 on */
```

#### **LED\_OFF(led)**

Turn LED off.

Parameter:

led - LED number

Example:

```
LED_OFF(1);    /* turn LED 1 off */
```

#### **LED\_TOGGLE(led)**

Toggle the LED OFF if the LED was ON and ON if the LED was OFF.

Parameter:

led - LED number

Example:

```
LED_TOGGLE(1); /* toggle LED 1 */
```

### 4.3 Binary Sensor Battery Code

The Developer's Kit contains code for a non-secure and secure battery powered binary sensor. This device is in effect a binary sensor where the sensor input is the pin also used as a button input on the device module. When the Binary Sensor is inactive, the ASIC will be powered down. The Binary sensor will power up when the button is pressed or the WUT is fired. Upon wakeup, a button press or WUT a Wakeup Notification Frame is sent either as broadcast or as singlecast to the device that configured the wakeup settings. If the device has any associations it will transmit, a basic set to the associated devices. If the button is held for a longer time, a Node Information Frame is transmitted. A static controller such as the one described in [6] can control, configure and assign routes to the Binary Sensor Battery.

The Binary Sensor Battery will start sending Wake Up Notifications once included into a network. The default wake up time is 1 minute, according to the sample application code. However, the actual outcome value can differ due to ASIC implementation. Once a Binary Sensor Battery wakes up, it stays awake for 34 seconds or until it receives a Wake Up No More frame from the controller.

The Binary Sensor Battery is a binary sensor that supports the association command class and the Wake Up command class described in the device class specification (see ref [1]). This device complies with the specific device class named routing binary sensor device class (4.1). When included non-secure the Secure Binary Sensor application lists the following supported command classes in the Node Information Frame:

#### Non-Secure Included

- Binary Sensor command class
- Wake Up command class
- Association command class
- Version command class
- Manufacturer Specific command class
- Security command class

#### Secure Included

When included secure the secure battery-operated Binary Sensor application lists the following supported command classes in the Node Information Frame:

- Version command class
- Manufacturer Specific command class
- Security command class

The following listed in the Security Commands Supported Report frame:

- Binary Sensor command class
- Wake Up command class
- Association command class
- Version command class
- Manufacturer Specific command class

The Basic command class is secure because application does not list it in Node Information Frame.

The Bin\_Sensor\_Battery is a slave device based on the enhanced slave API. During initialization, the Bin\_Sensor\_Battery will initialize the mounted button, the 4 LED's and a timer function that handles the button input and sensor input (in this example the same as the button input). It will also get stored data from the non-volatile memory. After the initialization it will go in power down mode and will wakeup again either when the button is pressed or when the WUT is fired. While the Bin\_sensor\_Battery is awake the Z-Wave basis software will continually call the **ApplicationPoll** function, which contains the Bin\_Sensor\_Battery main function. The **ApplicationPoll** function checks if the button or the sensor input has changed state and then acts accordingly to the current state the Bin\_Sensor\_Battery is in. The other main function is the **ApplicationCommandHandler** function that is called every time a command has been received, destined for the Bin\_Sensor\_Battery. This function checks the command and acts according to the command. When transmitting the Bin\_Sensor\_Battery will, if routes have been assigned use these. If the Bin\_sensor\_Battery wake up caused by sensor input or button activity, then it will power down again when it is done executing any event, which have been produced, by either the sensor input or the button activity. If the binary sensor is awakened by WUT and the wakeup time interval is expired then it will send wake notification frame and wait for 5 second before powering down again.

The Bin\_Sensor\_Battery implements Lost functionality and network topology maintenance by using a series of methods. If the device is unsuccessful in sending a message a predefined count it will enter lost state, and attempt to find a SUC in the network, and if successful ask the SUC for routes to the failing devices. At regular intervals the Bin\_Sensor\_Battery will transmit a Static Route Request, which asks the SUC for any updates done to the network.

Note that the wakeup notification frame will only be sent when the Bin\_sensor\_Battery has been assigned a node ID.

On the ZW0201/ZW0301 some of the uninitialized RAM bytes are used to keep track of the WUT timer. See also section **Error! Reference source not found.**

The Bin\_Sensor and Bin\_Sensor\_Battery share the same code base. They are distinguished between by defining BATTERY when compiling which will also enable use of the utility function file battery.c/h.

#### 4.3.1 Network Wide Inclusion

By default the Bin\_Sensor\_Battery will enter network wide inclusion (NWI) to be added to a network when it is powered up and have not already been included. The Bin\_Sensor\_Battery will stay in NWI mode for 30 minutes or until it has been included into the network. Any key press will terminate the NWI mode and the only way to make the Bin\_Sensor\_Battery enter NWI mode again is by doing hardware reset either by remove and reapply the power or press the reset button on the side of the board. Refer to section 4.2.1 regarding implementation details.

#### 4.3.2 Interface

The following table defines the functionality of the button on the Z-Wave module.

	<b>Button Triple Pressed</b>	<b>Button Clicked</b>
<b>In Network</b>	Node Info Frame / Enter learn mode	Basic Set (Broadcast)
<b>Not in Network</b>	Node Info Frame / Enter learn mode	None
<b>Associated</b>	Node Info Frame / Enter learn mode	Basic Set (to associated nodes)
<b>Wakeup Node set</b>	Node Info Frame / Enter learn mode	Wake Up Notifications (to Wake up node)
<b>Wakeup Node not set</b>	Node Info Frame / Enter learn mode	Wake Up Notifications (Broadcast)

#### 4.3.3 Bin\_Sensor\_Battery Files

Refer to chapter 4.2.3.



## 4.4 Development Controller Code

The Developer's Kit contains code that demonstrates how the basic tasks of adding, removing and controlling devices in a Z-Wave network using the Z-Wave portable controller API.

The Application complies with the Generic Controller command class [1]. When included the Development Controller application lists the following supported command classes in the Node Information Frame:

- Controller Replication command class
- Version command class

The Development Controller controls the following command classes:

- Controller Replication command class
- Basic command class
- Association command class

Controlled command classes not listed in the Node Information Frame in this sample application because it is optional to list.

For details regarding functionality supported by the development controller and user interface, refer to [15].

The Z-Wave basis software continually calls the **ApplicationPoll** function. The **ApplicationPoll** function contains a state machine, which initiates actions from user input. The **ApplicationCommandHandler** function is called when the Z-Wave basis software receives a frame. This could be a Basic Get Command to obtain the dim level of a multilevel switch.

### 4.4.1 Network Wide Inclusion

By default the controller will enter network wide inclusion (NWI) to be added to a network when it is powered up and have not already been included or have included other nodes itself. The controller will stay in NWI mode for 30 minutes or until it has been included into the network. Any key press will terminate the NWI mode and the only way to make the controller enter NWI mode again is by doing hardware reset either by remove and reapply the power or press the reset button on the side of the board.

The figure below show the NWI flow diagram implemented on application level for a controller. The ctrl\_learn.c file contains the implementation situated in the util\_func directory.

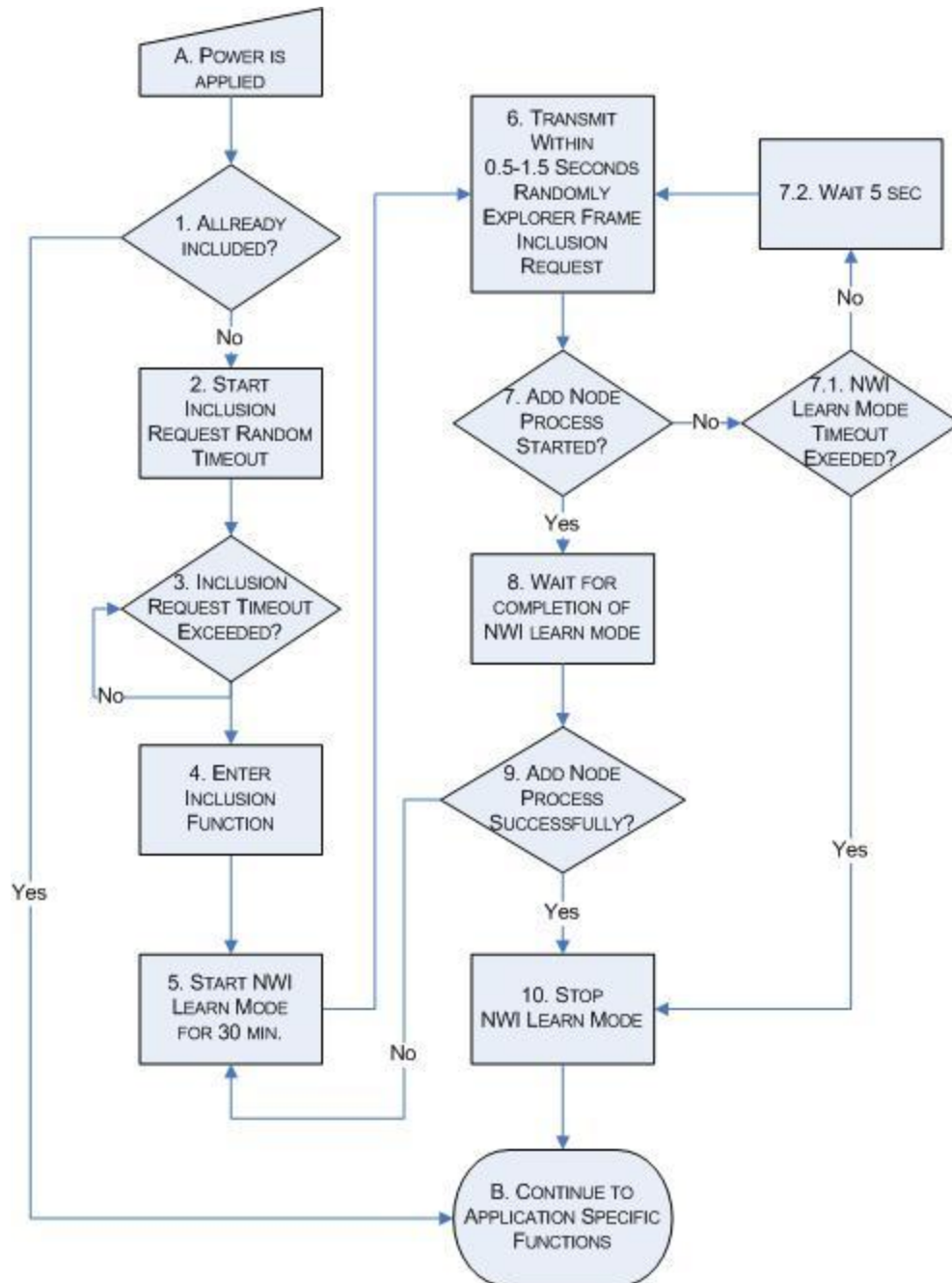


Figure 5. NWI flow diagram for a controller

#### 4.4.2 Dev\_Ctrl Files.

The Product\Dev\_ctrl directory contains the source code for the controller application. The application uses also a number of utility functions described in section 3.3.12.

#### MK.BAT

Batch file used to build ZW0201/ZW0301 based sample applications in ANZ, EU, HK, IN, MY, RU and US versions respectively. Refer to chapter 4.1 regarding details about the build procedure.

## Makefile

This file is part of the make job. It creates the directory structure and defines the targets that can be selected during make. The following compiler control line defines are used in the makefiles:

ANZ	: Build frequency target.
EU	: Build EU frequency target.
HK	: Build HK frequency target.
IN	: Build IN frequency target.
MY	: Build MY frequency target.
RU	: Build RU frequency target.
US	: Build US frequency target.
ZW_DEBUG	: Enable text output via UART.
ZW_DEBUG_CMD	: Enable command line debug via UART.
ZW_ID_SERVER	: Enable development controller as SIS.

## Makefile.dev\_ctrl\_common

This makefile is a common file defining all dependencies etc.

## eeeprom.c / eeeprom.h

These files contain functions and define for accessing the application data in the external EEPROM.

## dev\_ctrl\_if.h

This file defines how the IO connections on the Z-Wave module are connected to the Development Module.

## dev\_ctrl.c

This file contains the source code for the development controller application state machine. The common API functions such as **ApplicationInitHW**, **ApplicationInitSW**, **ApplicationNodeInformation**, **ApplicationPoll**, **ApplicationSlaveUpdate** and **ApplicationCommandHandler** are defined here.

## p\_button.c / p\_button.h

These files contain functions and define for detecting Push button presses. This includes Debounce checking.

## dev\_ctrl.mpw / dev\_ctrl\_....Uv2

uVision3 multiproject workspace file (\*.mpw) that unifies all the \*.Uv2 project files. There is a project file for every particular target with respect to single chip series and frequency.

## 4.5 Secure Development Controller (ATmega) Code

The ZDK contains code that demonstrates how the basic tasks of adding, removing and controlling devices in a Z-Wave network can be accomplished using a host processor to control a Serial API based portable controller application. The application is a security updated Development Controller application. The Z-Wave Development Platform ZDP02A [31] or ZDP03A [32] is used for this purpose. The host processor is an AVR ATmega128 and software is build by environment below:

- IAR Embedded Workbench for Atmel AVR (v. 4.30A)
- IAR C/C++ Compiler for AVR 4.30A/W32 (4.30.1.5)

The AVR ISP In-System Programmer programs the AVR ATmega128.

When included non-secure the Development Controller application lists the following supported command classes in the Node Information Frame:

### Non-Secure Included

- Controller Replication command class
- Version command class
- Security command class

### Secure Included

When included secure the Development Controller lists the following supported command classes in the Node Information Frame:

- Version command class
- Security command class

The following listed in the Security Commands Supported Report frame:

- Controller Replication command class
- Version command class

The Basic command class is secure because application does not list it in Node Information Frame. The Development Controller controls the following command classes:

- Controller Replication command class
- Basic command class
- Association command class

Controlled command classes not listed in the Node Information Frame in this sample application because it is optional to list.

For further information about the features of the Secure Development Controller using an AVR as host, see [30].

### 4.5.1 Dev\_Ctrl\_AVR\_Sec Files.

The Product\dev\_ctrl\_AVR\_Sec directory contains the source code for the controller application. Only selected files in the directory structure is described below.

#### Portable.dep/.ewd/.ewp/.eww

Project files used to build AVR based sample application.

**include\ZW\_Security\_AES\_module.h**

Header file used to implement security on application level.

**include\AES\_module.h**

This header file contains definitions for implementing secure communication using AES as encrypting/decrypting mechanism.

**src\ZW\_Security\_AES.c**

These files contain shared data and functions for AES128 and functions for AES128 encryption/decryption. Files are not distributed on the Developer's Kit CD due to export restrictions. Contact support via [zensys\\_support@sigmadesigns.com](mailto:zensys_support@sigmadesigns.com) for further information.

Alternatively, implement the functions based on an Atmel's Application Note "AVR231: AES Bootloader":

[http://www.atmel.com/dyn/resources/prod\\_documents/doc2589.pdf](http://www.atmel.com/dyn/resources/prod_documents/doc2589.pdf)

## 4.6 Door Bell Code

The developer's Kit contains code for a Door Bell sample application. This device an example of how a battery operated chime in a doorbell system could be build. The Door Bell uses the frequently listening mode where it powers up the radio for a short period every 1000ms@ZW0201 / 250ms@ZW0301 and if it receives a command it will power up entirely and turn on the LED's.

The Door Bell based on the routing slave library and it has its generic device class set to Binary Switch and the specific device class set to none. The Door Bell supports the following command classes:

- Binary Switch command class
- Version command class

**NOTE:** This node will fail certification because when its level is set to on with a binary set command it will toggle its state back to off again after a timeout to emulate the behavior of a doorbell.

### 4.6.1 Network Wide Inclusion

By default the Door Bell will enter network wide inclusion (NWI) to be added to a network when it is powered up and have not already been included. The Door Bell will stay in NWI mode for 30 minutes or until it has been included into the network. Any key press will terminate the NWI mode and the only way to make the Door Bell enter NWI mode again is by doing hardware reset either by remove and reapply the power or press the reset button on the side of the board. Refer to section 4.2.1 regarding implementation details.

### 4.6.2 User interface

The following list defines the functionality of the button on the Z-Wave module.

Press shortly

Wake up for 2 sec.

Press 3 times within 1.5 sec.

Enter learn mode and timeout after 3 sec.

The LEDs on the Z-Wave module has the following meaning:

LED 0	LED 1	LED 2	Description
Off	Off	Off	The door bell is in powerdown mode (Frequently listening mode)
On	Off	Off	The node was awakened by button press or reset
Off	On	Off	The node was awakened by an RF beam
Off	Off	On	The node is in learn mode
On	On	On	Bell was turned on by Binary or Basic set command

### 4.6.3 Door Bell Files

The Product\DoorBell directory contains the source code and makefiles for the application. The application uses also a number of utility functions described in section 3.3.12.

#### **Mk.bat**

Batch file to start compiling the sample application

#### **Makefile**

Theist file defines the targets that can be built in this directory.

#### **Makefile.doorbell\_common**

This makefile defines what source files that should be compiled for a specific target and it calls the appropriate makefiles in the Product\Common directory.

#### **Bell.c + Bell.h**

This file contains the source code for the Door Bell sample application

#### **DoorBell.mpw / DoorBell\_....Uv2**

uVision3 multiproject workspace file (\*.mpw) that unifies all the \*.Uv2 project files. There is a project file for every particular target with respect to single chip series and frequency.

## 4.7 Door Lock Code

The ZDK contains code for a non-secure and secure Door Lock sample application. This device shows an example of how a door lock system could be build.

A controller such as the Development Controller and secure Development Controller (Atmega) can control the Door Lock. The Door Lock uses the frequently listening mode (FLiRS) where it powers up the radio for a short period 1000ms@ZW0201 / 1000ms@ZW0301 and in case a wakeup beam for this particular node is detected then it stay awake to receive a command. It is now possible to turn the LED on/off indicating lock/unlock status. After receiving one command, it returns to frequently listening mode again to conserve battery consumption.

The Door Lock is based on the enhanced slave library and it has its generic device class set to Entry Control and the specific device class set to Door Lock.

The none-secure Door Lock lists the following supported command classes in the Node Information Frame:

- Lock command class
- Powerlevel command class
- Version command class
- Manufacturer Specific command class

When included non-secure the secure Door Lock application lists the following supported command classes in the Node Information Frame:

### Non-Secure Included

- Lock command class
- Powerlevel command class
- Version command class
- Manufacturer Specific command class
- Security command class

### Secure Included

When included secure the secure Door Lock lists the following supported command classes in the Node Information Frame:

- Version command class
- Manufacturer Specific command class
- Security command class

The following listed in the Security Commands Supported Report frame:

- Lock command class
- Powerlevel command class
- Version command class
- Manufacturer Specific command class

The Basic command class is secure because application does not list it in Node Information Frame.



During initialization, the Door Lock will initialize the mounted button and one LED. It will also get stored data from the non-volatile memory. After the initialization the Z-Wave basis software will continually call the **ApplicationPoll** function, which contains the Door Lock main function. The **ApplicationPoll** function checks button activation and act according to the state the Door Lock is in. The other main function is the **ApplicationCommandHandler** function that is called every time a command has been received, destined for the Door Lock. This function checks the command and acts according to the command.

#### 4.7.1 Network Wide Inclusion

By default the Door Lock will enter network wide inclusion (NWI) to be added to a network when it is powered up and have not already been included. The Door Lock will stay in NWI mode for 30 minutes or until it has been included into the network. Any key press will terminate the NWI mode and the only way to make the Door Lock enter NWI mode again is by doing hardware reset either by remove and reapply the power or press the reset button on the side of the board. Refer to section 4.2.1 regarding implementation details.

#### 4.7.2 User Interface

The following table defines the functionality of the button on the Z-Wave module.

	Button Triple Pressed	Button Clicked
In Network	Node Info Frame / Enter learn mode	Toggle on/off status
Not in Network	Node Info Frame / Enter learn mode	Toggle on/off status

Learn mode is now activated by pressing the button three times within 1.5 seconds to avoid unintentional inclusion/exclusion of the node.

#### 4.7.3 Secure Door Lock Files

The Product\DoorLock directory contains the source code and makefiles for the application. The application uses also a number of utility functions described in section 3.3.12.

##### Mk.bat

This batch file controls the build process of the sample applications by calling the appropriate makefiles and parameter settings.

##### Makefile / Makefile.SecureTargets

These files define the possible targets to build in this directory. It creates the directory structure and defines targets. The following compiler control line defines are used in the makefiles:

ANZ : Build Australia/New Zealand frequency targets

EU : Build European frequency targets

HK : Build Hong Kong frequency targets  
IN : Build India frequency targets  
MY : Build Malaysia frequency targets  
RU : Build Russian Federation frequency targets  
US : Build US frequency targets

### **Makefile.common**

This makefile defines what source files that should be compiled for a specific target and it calls the appropriate makefiles in the Product\Common directory.

### **eeeprom.h**

This header file contains the address definitions in the external EEPROM used to store application data.

### **DoorLock.c + DoorLock.h**

This file contains the source code for the non-secure and secure Door Lock sample application

### **DoorLock....mpw / DoorLock\_Secure\_....Uv2**

uVision3 multiproject workspace file (\*.mpw) that unifies all the \*.Uv2 project files. There is a project file for every particular target with respect to non-secure/secure support, single chip series and frequency.

## **4.7.3.1 Macros for accessing the Lock/Unlock**

### **PIN\_ON(pin)**

Set output pin to 1.

Parameter:

pin - Z-Wave pin name

Example:

```
PIN_ON(TRIACpin); /* turn TRIACpin on */
```

### **PIN\_OFF(pin)**

Set output pin to 0.

Parameter:

pin - Z-Wave pin name

Example:

```
PIN_OFF(TRIACpin); /* turn TRIACpin off */
```

**PIN\_GET(pin)**

Read pin value.

Parameter:

pin - Z-Wave pin name

Example:

```
PIN_GET(SSN);      /* Read pin SSN value*/
```

**PIN\_IN(pin, pullup)**

Set I/O pin as input.

Parameter:

pin - Z-Wave pin name

pullup - if not zero activate the internal pullup resistor

Example:

```
PIN_IN(SSN, 0);    /* Set I/O pin SSN as input and activate the internal pullup resistor */
```

## 4.8 LED Dimmer Code

The Developer's Kit contains code for a non-secure and secure LED Dimmer. This device is in effect a light switch with a built in dimmer where the light bulb is substituted with 3 LED's when using ZW0201/ZW0301. A controller such as the secure or non-secure Development Controller code can control the LED Dimmer.

The LED Dimmer is a multilevel switch that complies with the specific device class named multilevel power switch device class (4.1). It supports the all switch command class, the protection command class and the powerlevel command class described in the device class specification (see ref [1]). The LED Dimmer also supports Meter Command Class enabling it to report current power consumption of connected load by configuration of Association Command Class. The rate of Meter Report commands are determined by the Configuration command class. Some of the rates supported are only for test purposes and should be removed in the final product. The LED Dimmer does not support the optional basic Clock command class. When included the non-secure LED Dimmer application lists the following supported command classes in the Node Information Frame:

### Non-Secure Included

- Multilevel Switch command class
- All Switch command class
- Protection command class
- Powerlevel command class
- Version command class
- Manufacturer Specific command class
- Meter command class
- Configuration command class (Used to configure unsolicited Meter Report frequency)
- Association command class (Used to send unsolicited Meter Reports)
- Security command class

### Secure Included

When included secure the secure LED Dimmer lists the following supported command classes in the Node Information Frame:

- Version command class
- Manufacturer Specific command class
- Security command class

The following listed in the Security Commands Supported Report frame:

- Multilevel Switch command class
- All Switch command class
- Protection command class
- Powerlevel command class
- Version command class
- Manufacturer Specific command class
- Meter command class (Fixed 300 seconds interval for unsolicited reports)
- Association command class (Used to send unsolicited Meter Reports)

The Basic command class is secure because application does not list it in Node Information Frame.

The none-secure LED Dimmer is a routing slave device based on the slave/routing slave API. The secure LED Dimmer is a slave device based on the slave/enhanced slave API. During initialization, the LED Dimmer will initialize the mounted button and the 3 LED's. It will also get stored data from the non-volatile memory. After the initialization, the Z-Wave basis software will continually call the **ApplicationPoll** function, which contains the LED Dimmer main function. The **ApplicationPoll** function checks if the button has been pressed and act according to the state the LED Dimmer is in. The other main function is the **ApplicationCommandHandler** function that is called every time a command has been received, destined for the LED Dimmer. This function checks the command and acts according to the command.

When using ZW0201/ZW0301 as the device containing the LED Dimmer application, it is primarily supposed to be mounted onto a slave interface module. It can also be mounted on a ZDP03A board, however, in this case only two of the three LEDs will be used by the application. This is determined by the architecture of the ZDP03A board.

#### 4.8.1 Network Wide Inclusion

By default the LED Dimmer will enter network wide inclusion (NWI) to be added to a network when it is powered up and have not already been included. The LED Dimmer will stay in NWI mode for 30 minutes or until it has been included into the network. Any key press will terminate the NWI mode and the only way to make the LED Dimmer enter NWI mode again is by doing hardware reset either by remove and reapply the power or press the reset button on the side of the board. Refer to section 4.2.1 regarding implementation details.

#### 4.8.2 Interface

The following table defines the functionality of the button on the Z-Wave module.

	Button Triple Pressed	Button Clicked	Button is held
<b>In Network</b>	Node Info Frame / Enter learn mode	Toggle on/off status	Dim up/down
<b>Not in Network</b>	Node Info Frame / Enter learn mode	Toggle on/off status	Dim up/down

Learn mode is now activated by pressing the button three times within 1.5 seconds to avoid unintentional inclusion/exclusion of the node.

#### 4.8.3 LED\_Dimmer Files

The Product\LED\_Dimmer directory contains sample source code for a none-secure/secure routing/enhanced slave application on a Z-Wave module. The application uses also a number of utility functions described in section 3.3.12.

#### MK.BAT

Batch file used to build ZW0201/ZW0301 based sample applications in ANZ, EU, HK, IN, MY, RU and US versions respectively. Refer to chapter 4.1 regarding details about the build procedure.

## Makefile

This file is part of the make job. It creates the directory structure and defines targets. The following compiler control line defines are used in the makefiles:

```

ANZ           : Build ANZ frequency target.
EU            : Build EU frequency target.
HK            : Build HK frequency target.
IN            : Build IN frequency target.
MY            : Build MY frequency target.
RU            : Build RU frequency target.
US            : Build US frequency target.
NOOFFONDIM   : When enabled not all the LED's will turn off when dimming.
APPL_PROD_TEST: Enable the production test.

```

## Makefile.leddimmer\_common

This makefile is a common file defining all dependencies etc.

## eeeprom.h

This header file defines the addresses where application data are stored in the external EEPROM.

## LEDdim.h / LEDdim.c

This file contains the source code for the LED dimmer application state machine. The common API functions such as **ApplicationInitHW**, **ApplicationInitSW**, **ApplicationNodeInformation**, **ApplicationPoll**, **ApplicationSlaveUpdate** and **ApplicationCommandHandler** are defined here.

## LED\_Dimmer.mpw / LED\_Dimmer\_....Uv2

uVision3 multiproject workspace file (\*.mpw) that unifies all the \*.Uv2 project files. There is a project file for every particular target with respect to single chip series and frequency.

### 4.8.3.1 Macros for accessing the LED's

#### LED\_ON(led)

Turn LED on.

Parameter:

led - LED number

Example:

```

PIN_OUT(LED1); /* define LED1 as an output pin */
LED_ON(1);     /* turn LED 1 on */

```

**LED\_OFF(led)**

Turn LED off.

Parameter:

led - LED number

Example:

```
LED_OFF(1);    /* turn LED 1 off */
```

**LED\_TOGGLE(led)**

Toggle the LED OFF if the LED was ON and ON if the LED was OFF.

Parameter:

led - LED number

Example:

```
LED_TOGGLE(1);    /* toggle LED 1 */
```

## 4.9 MyProduct Code

The My Product contains the minimum framework to begin developing a slave application. To realize the application in question it's often easier to modify the existing code applications than build one from scratch based on MyProduct.

### 4.9.1 MyProduct Files

The Product\MyProduct directory contains sample source code for a routing slave application on a Z-Wave module. The application uses also a number of utility functions described in section 3.3.12.

#### **MK.BAT**

Batch file used to build ZW0201/ZW0301 based sample applications in ANZ, EU, HK, IN, MY, RU and US versions respectively. Refer to chapter 4.1 regarding details about the build procedure.

#### **Makefile**

This file is part of the make job. It creates the directory structure and defines targets. The following compiler control line defines are used in the makefiles:

ANZ	: Build ANZ frequency target.
EU	: Build EU frequency target.
HK	: Build HK frequency target.
IN	: Build IN frequency target.
MY	: Build MY frequency target.
RU	: Build RU frequency target.
US	: Build US frequency target.

#### **Makefile.MyProduct\_common**

This makefile is a common file defining all dependencies etc.

#### **MyProduct.h / MyProduct.c**

This file contains the source code for the MYProduct. The common API functions such as **ApplicationInitHW**, **ApplicationInitSW**, **ApplicationNodeInformation**, **ApplicationPoll**, **ApplicationSlaveUpdate** and **ApplicationCommandHandler** are defined here.

#### **MyProduct.mpw / MyProduct\_....Uv2**

uVision3 multiproject workspace file (\*.mpw) that unifies all the \*.Uv2 project files. There is a project file for every particular target with respect to single chip series and frequency.



#### 4.10 Production Test DUT

The Developer's Kit contains code that demonstrates how the basic tasks of testing devices in a Z-Wave network can be accomplished using the Z-Wave API.

The Z-Wave basis software continually calls the **ApplicationPoll** function. The **ApplicationPoll** function contains a state machine, which initiates actions from user input.

The Production Test DUT sample application is based on the `ZW_slave_prodtest_dut` library.

This sample application has two functions that can be used during production test

- The radio will start to transmit continuously if the P1.5 (SS\_N) pin on the ZM2102 is pulled low during power up.
- If the pin isn't pulled low the radio will go into receive mode and send acknowledge to all frames send to the module.

The program execution flow is as follows:

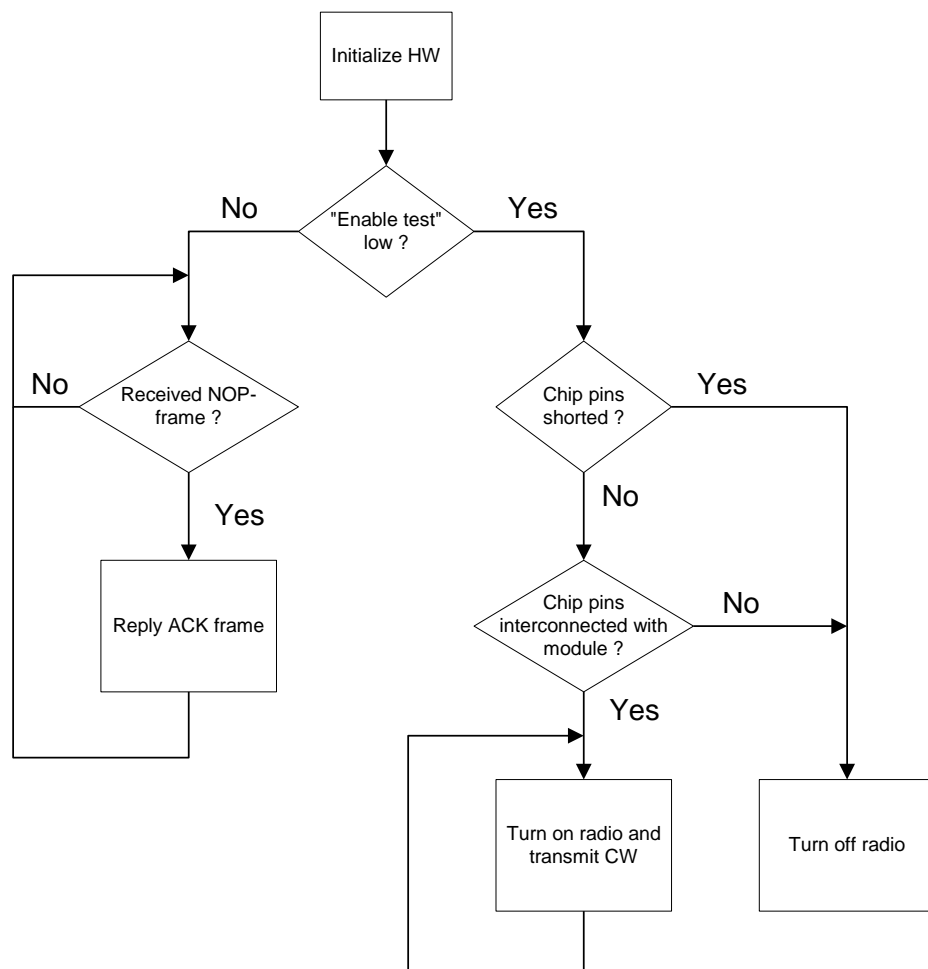


Figure 6. Prod\_Test\_DUT test program flow

Immediately after program execution start, the state of the "Enable test" pin (pin P1.5 on the ZW0201) is tested. If the pin is high, "normal" Z-Wave slave code is started, and the DUT will reply to a NOP frame with an ACK frame. This behavior is used during the Link test, where 10 NOP's are transmitted from the Z-Wave test box and 10 ACK's are expected from the DUT.

If the "Enable test" pin is low, a test program flow is started.

All pins not used during chip programming and test enabling are tested for shorts. The CPU of the ZW0201 writes a "0101..." pattern to its pins and then reads back the state of the pins. If no shorts, the pattern read will be "0101...". Then a "1010..." pattern is written, and "1010..." is expected when reading back.

The interconnections from the ZW0201 chip to the castellation notched of the ZM2102 module are tested. The CPU enables the internal pull-up resistors on the ZW0201 chip and sets the pins high. The read back of the pins should then be high. All pins are then set to low, and because of the external 10 kOhm pull down resistors, all pins should be read back as being low. If an interconnection fails, the internal pull up will lead the CPU to read the pin as being high instead of low.

If one of the above tests fails, the radio will be turned off.

If both of the above tests pass, the radio will be turned on to transmit a CW, and the spectrum analyzer is then able to measure the RF frequency and RF output power.

#### 4.10.1 Production Test DUT Files

The Product\Prod\_Test\_DUT directory contains the source code for the Production Test DUT sample application.

##### **MK.BAT**

Batch file used to build ZW0201/ZW0301 based sample applications in ANZ, EU, HK, IN, MY, RU and US versions respectively. Refer to chapter 4.1 regarding details about the build procedure.

##### **Makefile**

This file is part of the make job. It creates the directory structure and defines the targets that can be selected during make. The following compiler control line defines are used in the makefiles:

ANZ	: Build ANZ frequency target.
EU	: Build EU frequency target.
HK	: Build HK frequency target.
IN	: Build IN frequency target.
MY	: Build MY frequency target.
RU	: Build RU frequency target.
US	: Build US frequency target.

##### **prodtestdut.c**

This file contains the main source code for the sample application. Both **ApplicationPoll** and **ApplicationCommandHandler** are defined in this file.

##### **prodtestdut.h**

This file contains definitions for the prod\_test\_dut sample application.

##### **Prod\_Test\_DUT.mpw / Prod\_Test\_DUT\_....Uv2**

uVision3 multiproject workspace file (\*.mpw) that unifies all the \*.Uv2 project files. There is a project file for every particular target with respect to single chip series and frequency.

#### 4.11 Production Test Generator

The Developer's Kit contains code that demonstrates how the basic tasks of testing devices in a Z-Wave network can be accomplished using the Z-Wave API. The Z-Wave generator is used to verify the TX / RX circuits on Z-Wave enabled products.

A simple generator consists of a ZW010x Interface Module and a ZMxx20 Z-Wave Module.

On the Interface module there are 6 LED diodes, which have these assignments in the Prod\_Test\_Gen sample application:

LED #	Colour	Description
D6	Green	Power on
D1	Red	Error
D2	Red	Success
D3	Red	Send (flashes during transmission)
D4	Red	-
D5	Red	Indication of Push button

The push button on the ZMxx20 Z-Wave Module is the "Test" button.

After connection to power, the red "Error" LED 'D1' on the Interface module will be on.

When the push button is pressed, 10 NOP's will be transmitted. A device under test (a Prod\_Test\_DUT) is expected to verify the reception of each NOP with an ACK. During transmission, the red LED 'D3' will flash.

If all NOP's are replied correctly, the red "Error" LED 'D1' will turn off and the red "Success" LED 'D2' will turn on and stay on until the next test is conducted. If the DUT does not reply correctly, the red "Error" LED 'D1' will turn on and stay on until the next test is conducted.

The Z-Wave basis software continually calls the **ApplicationPoll** function. The **ApplicationPoll** function contains a state machine, which initiates actions from user input. The **ApplicationCommandHandler** function is only called when the Z-Wave basis software receives information for the application.

The Production Test Generator sample application is based on the ZW\_slave\_proctest\_gen library.

The application is controlled via RS232 (115200,8,N,1) or button with fixed timings:  
Device will respond to any char received with an ASCII SPACE followed by a command answer or error  
'!' followed by error information:  
Following ASCII commands are implemented.  
Received:

'U':  
Frequency US is selected  
Response is: ' ' 'U' 'S'

'E':  
Frequency EU is selected  
Response is: ' ' 'E' 'U'

'S':  
Start test  
Response is ' ' 'S' 'T'

'C':  
Set the number of NOPs to send  
Response: ' ' 'C' 'O'

'N':  
Set the destination node ID.  
Response: ' ' 'N' 'I'

'R':  
Reset the hardware  
Response: ' ' 'R' 'S'

For ZW020x series and ZW030x series

'B':  
'4' - Use 40KBit. Response "B:40k"  
'9' - Use 9.6kbit. Response "B:9.6k"

On Unknown:  
'!' 'received Char'

#### 4.11.1 Production Test Generator Files

The Product\Prod\_Test\_Gen directory contains the source code for the Production Test Generator sample application.

##### **MK.BAT**

Batch file used to build ZW0201/ZW0301 based sample applications in ANZ, EU, HK, IN, MY, RU and US versions respectively. Refer to chapter 4.1 regarding details about the build procedure.

##### **Makefile**

This file is part of the make job. It creates the directory structure and defines the targets that can be selected during make. The following compiler control line defines are used in the makefiles:

ANZ	: Build ANZ frequency target.
EU	: Build EU frequency target.
HK	: Build HK frequency target.
IN	: Build IN frequency target.
MY	: Build MY frequency target.
RU	: Build RU frequency target.
US	: Build US frequency target.

##### **prod\_test\_gen.c**

This file contains the main source code for the sample application. Both **ApplicationPoll** and **ApplicationCommandHandler** are defined in this file.

##### **Prod\_Test\_Gen.mpw / Prod\_Test\_Gen.....Uv2**

uVision3 multiproject workspace file (\*.mpw) that unifies all the \*.Uv2 project files. There is a project file for every particular target with respect to single chip series and frequency.

#### **4.12 Serial API Embedded Code**

For a description of how to develop serial API based host applications refer to [37].

#### **4.13 PC based Controller Sample Application**

The PC\Source\SampleApplications\ZWavePCController directory contains sample application source code in C# that implements a PC based Controller using the development tool Visual Studio 2008.

For further information about the features of the PC based Controller, see [6].

#### **4.14 PC based Installer Tool Sample Application**

The PC\Source\SampleApplications\ZWaveInstaller directory contains sample application source code in C# that implements a PC based Installer Tool using the development tool Visual Studio 2008.

For further information about the features of the PC based Installer Tool, see [7].

#### **4.15 PC based Z-Wave Bridge Sample Application**

The PC\Source\SampleApplications\ZWaveUPnPBridge directory contains sample application source code in C# that implements a PC based Z-Wave to UPnP Bridge using the development tool Visual Studio 2008.

For further information about the features of the PC based Z-Wave to UPnP Bridge, see [8].

## 5 TOOL CODE

The Z-Wave Developer's Kit includes tool code to enable customization of production environment.



## 5.1 Z-Wave Programmer Firmware

The ZDK contains code that demonstrates how to program the 100/200/300/400 Series ASIC. The ZDP02A Z-Wave Development Platform [31] or ZDP03A Z-Wave Development Platform [32] supports this purpose. The Z-Wave Programmer firmware resides on the AVR ATmega128 chip on ZDP03A and controlled by the PC based Z-Wave Programmer application [14]. For a detailed description of the communication protocol between the AVR and PC based Z-Wave Programmer application, refer to [34].

Source code developed in the following environment:

- WinAVR v20071221:
  - o GNU Binutils 2.18 (including assembler, linker, etc.)
  - o Compiler Collection (GCC) 4.2.2
  - o avr-libc 1.6.0
- Z-Wave Library v2.91
- Keil uVision PK51 v9

Project environment:

- Eclipse Platform v3.5 with plugins:
  - o AVR Eclipse Plugin
  - o (optional) Polarion Subversive SVN Connectors
  - o (optional) Eclipse Subversive - SVN Team Provider Project

The AVR ISP In-System Programmer programs the AVR Atmega128.

### 5.1.1 ATmega\_ZWaveProgFW Files

The Tools\Programmer\ATmega\_ZWaveProgFW directory contains the source code for the 400 Series low level programming application.

#### **MK.BAT**

Batch file used to build AVR based sample applications in versions for the firmware update (via Z-Wave Programmer) and complete ATmega128 firmware (via AVR ISP In-System Programmer). Refer to chapter 4.1 regarding details about the build procedure.

#### **MAKE\_FIRMWARE.BAT**

Batch file used to make complete ATmega128 firmware from bootloader firmware and firmware update. Called by MK.BAT.

#### **MAKE\_MTP.BAT**

Batch file used to build the ZW040x Execute Out of SRAM application, that give the ability to the ATmega128 firmware to access the MTP memory of the ZW040x chip. Called by MK.BAT.

#### **.cproject; .project; .settings**

Project files of the Eclipse IDE used to edit AVR based sample application source code.

#### **src\ATmega\_spi.c; .h**

Source code of the implementation of the software SPI, which is connected to the Z-Wave Module.

**src\commands.h**

This header file contains definitions of the commands of the Z-Wave Programmer Communication Protocol [34].

**src\conhandle.c; .h**

Source files, contains the functions for handling the Programmer frames via the UART.

**src\leeprom\_if.c; .h**

Source code of the Z-Wave Module External EEPROM interface. Reading / writing of the Z-Wave Module External EEPROM via the software SPI was implemented.

**src\mtp.c; .h**

Source code of the ZW040x Execute out of SRAM application, which implements the ZW040x MTP memory interface.

**src\ports.h**

Header file with definitions of port names of the ATmega128 in ZDP02 (ZDP03) board.

**src\UART\_buf\_io.c; .h**

Source code of buffered transmit/receive of data through the UART.

**src\ZWWaveFlash.c; .h**

Main source code of the Z-Wave Programmer Firmware. Contains the implementation of all programmer commands handlers and Z-Wave chips programming algorithms.

## 6 REQUIRED DEVELOPMENT COMPONENTS

### 6.1 Software development components

There is an additional 3<sup>rd</sup> party software tool that is required to develop Z-Wave applications that is not supplied with the Z-Wave Developer's Kit. That is the Keil PK51 v9 Professional Developer's Kit for the 8051 microcontroller:

Z-Wave libraries and sample applications are built and tested on above version 9 but newer versions should also apply according to Keil's recommendations.

The Keil Developer's Kits can be purchased directly from Keil or from one of their local distributors. Please visit [www.keil.com](http://www.keil.com) for details. Alternatively can it be purchased from Sigma Designs.

<b>Keil Software, Inc.</b> 1501 10th Street, Suite 110 Plano, TX 75074 USA		<b>Keil Elektronik GmbH</b> Bretonischer Ring 15 D-85630 Grasbrunn Germany	
Toll Free:	<b>800-348-8051</b>	Toll Free:	-
Phone:	972-312-1107	Phone:	(49) (089) 45 60 40 0
Fax:	972-312-1159	Fax:	(49) (089) 46 81 62
Sales:	<a href="mailto:sales.us@keil.com">sales.us@keil.com</a>	Sales:	<a href="mailto:sales.intl@keil.com">sales.intl@keil.com</a>
Support:	<a href="mailto:support.us@keil.com">support.us@keil.com</a>	Support:	<a href="mailto:support.intl@keil.com">support.intl@keil.com</a>

### 6.2 ZW0102/ZW0201/ZW0301 single chip programmer

This Z-Wave Developer's Kit comes with the Z-Wave Programmer included. The Z-Wave Programmer is used for downloading new firmware to the ZW0x0x Single Chip. For a detailed description refer to [14].

The Z-Wave Programmer is also used when programming the external EEPROM on the Z-Wave module. For further details refer to paragraph 6.7.

### 6.3 Hardware development components for ZW0102

The ZW0102 based static controller serial API and all slave sample applications are designed for the ZW0102 Controller/Slave Unit, which is an assembly of the ZW0x0x Interface Module [2] and the ZM1220 Z-Wave Module [4].

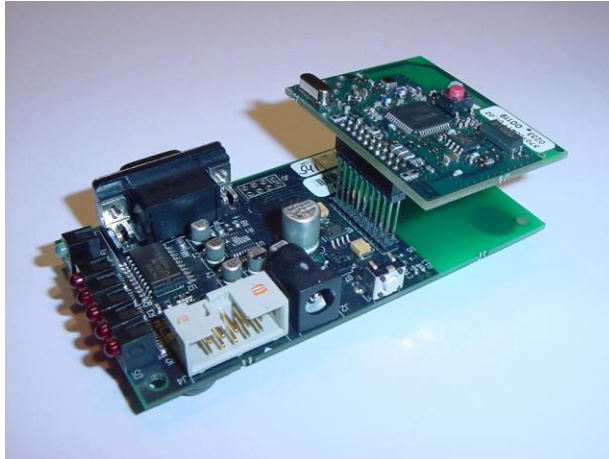


Figure 7. ZW0102 Controller/Slave Unit

The ZW0102 development controller sample application is designed for the ZW0102 Development Controller Unit, which is an assembly of the ZW0x0x Development Module [3] and the ZM1220 Z-Wave Module [4].

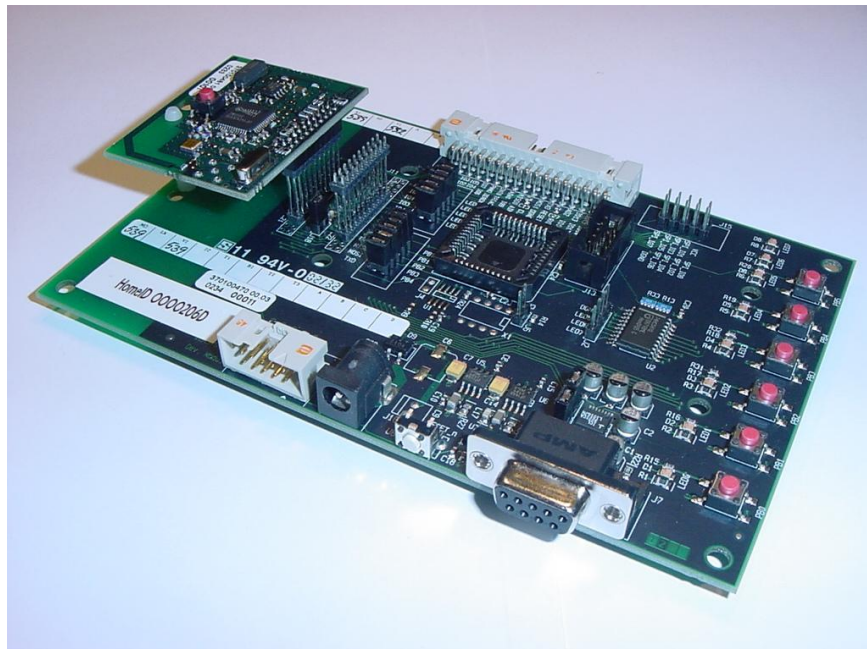


Figure 8. ZW0102 Development Controller Unit

## 6.4 Hardware development components for ZW0201

The ZW0201 based static controller serial API and all slave sample applications are designed for the ZW0201 Controller/Slave Unit, which is an assembly of the ZW0x0x Interface Module [2], ZMxx06 Converter Module [17], and the ZM2106C Module (incl. ZM2102) [18]. Alternatively can it be an assembly of the ZW0x01 Interface Module [2], and the ZM2120C Module (incl. ZM2102) [23].

The ZW0201 development controller sample application is designed for the ZW0201 Development Controller Unit, which is an assembly of the ZW0x0x Development Module [3], ZMxx06 Converter Module [17] and the ZM2106C Module (incl. ZM2102) [18]. Alternatively can it be an assembly of the ZW0x0x Development Module [3], and the ZM2120C Module (incl. ZM2102) [23].

## 6.5 Hardware development components for ZW0301

The ZW0301 based static controller serial API and all slave sample applications are designed for the ZW0301 Controller/Slave Unit, which is an assembly of the ZW0x0x Interface Module [2], ZMxx06 Converter Module [17] and the ZM3106C Module (incl. ZM3102) [22]. Alternatively can it be an assembly of the ZW0x0x Interface Module [2], and the ZM3120C Module (incl. ZM3102) [24].

The ZW0301 development controller sample application is designed for the ZW0301 Development Controller Unit, which is an assembly of the ZW0x0x Development Module [3], ZMxx06 Converter Module [17] and the ZM3106C Module (incl. ZM3102) [22]. Alternatively can it be an assembly of the ZW0x0x Development Module [3], and the ZM3120C Module (incl. ZM3102) [24].

## 6.6 ZW0102/ZW0201/ZW0301 lock bit settings

The ZW0102/ZW0201/ZW0301 lock bits related to protection of the flash contents should during development be set as follows:

**Table 1. Lock bits settings during development**

Lock bits	Value	Description
SPIRE	1	It is allowed to read the flash data via the SPI interface
BSIZE[2..0]	111	Boot sector size set to 0 bytes
BOBLOCK	1	Page 0 is writeable

This allows the developer to read contents of the flash. The possibility to read flash contents should be disabled in the end product to avoid copy production. The ZW0102/ZW0201/ZW0301 lock bits in end products should be set as follows:

**Table 2. Lock bits settings in end products**

Lock bits	Value	Description
SPIRE	0	It is not allowed to read the flash data via the SPI interface
BSIZE[2..0]	111	Boot sector size set to 0 bytes
BOBLOCK	1	Page 0 is writeable

Regarding a detailed description about flash programming and lock bits for ZW0102, ZW0201 and ZW0301, refer to [16], and [10] respectively.

## 6.7 External EEPROM initialization

When creating a controller on a new ZW0102/ZW0201/ZW0301 based Z-Wave module a home ID must be allocated. The home ID is stored in the external EEPROM on the Z-Wave module. Beside the home ID the remaining part of the external EEPROM must be zeroed. The controller requires an initialized external EEPROM as describe above to operate correct. With respect to an enhanced slave then the whole external EEPROM must be zeroed before it can operate correct. The external EEPROM must only be initialized once when creating a controller or enhanced slave on a new Z-Wave module.

In the binary controller directories ...\\Product\\Bin\\ supplied on the Developer's Kit CD are the image files `extern_eep.hex` to be downloaded found. The 32-bit home ID (xxxxxxx) is located in byte 8, 9, 10 and 11 (when counting from 0) in the file. Byte 8 is the most significant byte and byte 11 is the least significant.

[illegible]

The procedure to initialize the external EEPROM on the Z-Wave module is described in [14].

The external EEPROM on the ZM1206 module can **only** be updated by the steps described below:

1. Use the Z-Wave Programmer [14] to download eeploader\_ZW0102.hex file to the module.
2. Connect a RS-232 serial cable directly from the PC to the Z-Wave interface module. Notice that download of the extern\_eep.hex file is not done via the Z-Wave programmer.
3. Go to the directory ...\\Tools\\Eeprom\_loader\\ and open a command prompt (DOS box). The eeploader.exe program in this directory is used to download the extern\_eep.hex file via the COM port.
4. To download the extern\_eep.hex in the Development Controllers binary directory ...\\Product\\Bin\\Dev\_Ctrl run **eeploader ..\\..\\Product\\Bin\\Dev\_Ctrl\\extern\_eep.hex COMx**, where x is equal to the COM port number the cable is attached to.
5. The PC application displays download progress and finally download status.

## APPENDIX A FOO

bla bla

### Appendix A.1 Bar

bla bla



## REFERENCES

- [1] SD, SDS10242, Software Design Specification, Z-Wave Device Class Specification.
- [2] SD, DSH10086, Datasheet, ZW0x0x Z-Wave Interface Module.
- [3] SD, DSH10087, Datasheet, ZW0x0x Z-Wave Development Module.
- [4] SD, DSH10033, Datasheet, ZM1220 Z-Wave Module.
- [5] SD, DSH10034, Datasheet, ZM1206 Z-Wave Module.
- [6] SD, INS10240, Instruction, PC Based Controller User Guide.
- [7] SD, INS10241, Instruction, PC Installer Tool Application User Guide.
- [8] SD, INS10245, Instruction, Z-Wave Bridge User Guide.
- [9] SD, INS10029, Instruction, ZW0102 Single Chip Implementation Guideline.
- [10] SD, APL10312, Application Note, Programming the 200 and 300 Series Z-Wave Single Chip Flash.
- [11] SD, INS10336, Instruction, Z-Wave Reliability Test Guideline.
- [12] SD, INS10249, Instruction, Z-Wave Ziffer User Guide.
- [13] SD, INS10250, Instruction, Z-Wave DLL User's Manual.
- [14] SD, INS10679, Instruction, Z-Wave Programmer User Guide.
- [15] SD, INS10236, Instruction, Development Controller User Guide.
- [16] SD, INS10579, Instruction, Programming the ZW0102 Flash and Lock Bits.
- [17] SD, DSH10088, Datasheet ZMxx06 Converter Module.
- [18] SD, DSH10230, Datasheet, ZM2106C Z-Wave Module.
- [19] SD, INS10326, Instruction, ZW0201 Single Chip Implementation Guidelines.
- [20] SD, SRN11886, Software Release Note, ZW0201/ZW0301 Developer's Kit v4.53.00.
- [21] SD, APL10512, Application Note, Battery Operated Applications Using the ZW0201/ZW0301.
- [22] SD, DSH10856, Datasheet, ZM3106C Z-Wave Module.
- [23] SD, DSH10275, Datasheet, ZM2120C Z-Wave Module.
- [24] SD, DSH10857, Datasheet, ZM3120C Z-Wave Module.
- [25] SD, APL10292, Application Note, ZW0102 Triac Controller Guideline.
- [26] SD, APL10370, Application Note, ZW0201/ZW0301 Triac Controller Guideline.
- [27] SD, APL10514, Application Note, The ZW0201/ZW0301 ADC.
- [28] SD, INS10680, Instruction, Z-Wave XML Editor.
- [29] SD, INS11018, Instruction, Secure PC Based Controller User Guide (OBSOLETE, see INS10240).
- [30] SD, INS10681, Instruction, Secure Development Controller (AVR) User Guide.
- [31] SD, DSH10704, Datasheet, ZDP02A Z-Wave Development Platform.
- [32] SD, DSH11243, Datasheet, ZDP03A Z-Wave Development Platform.
- [33] SD, SDS11060, Software Design Specification, Z-Wave Command Class Specification.
- [34] SD, INS11072, Instruction, Z-Wave Programmer Communication Protocol.
- [35] SD, APL10742, Application Note, ZM3102N with External PA and Switch.
- [36] SD, INS11552, Instruction, 400 Series Crystal Calibration User Guide.
- [37] SD, INS12350, Instruction, Serial API Host Appl. Prg. Guide.

## INDEX

### A

AES128 encryption/decryption .....	28, 56
AVR ATmega128.....	16, 28

### C

Calibration.....	32
Command prompt .....	39
Crystal calibration .....	32

### D

DOS box .....	39
---------------	----

### E

Eeploader.exe .....	82
Extern_eep.hex file .....	82
External EEPROM .....	82

### I

Intellectual property rights .....	28
------------------------------------	----

### K

Keil .....	78
KEILPATH .....	39

### L

Lock bits .....	81
-----------------	----

### M

Make files.....	38
MK.BAT .....	38

### P

PVT and RF regulatory measurements.....	33
-----------------------------------------	----

### S

Stiftung Secure Information and Communication Technologies .....	28
------------------------------------------------------------------	----

### T

TOOLS DIR .....	39
-----------------	----

### U

uninitialized RAM bytes .....	50
-------------------------------	----

### Z

ZDP02A Development module.....	16, 28
ZDP03A Development module.....	16, 28
ZW0102 Controller/Slave Unit.....	79

ZW0102 Development Controller Unit .....	79
ZW0201 Controller/Slave Unit.....	80
ZW0201 Development Controller Unit .....	80
ZW0301 Controller/Slave Unit.....	80
ZW0301 Development Controller Unit .....	80
Z-Wave Programmer .....	78