Instruction

Z-Wave Programmer Communication Protocol

Document No.:	INS11072
Version:	5
Description:	Description of the Z-Wave Programmer communication interface.
Written By:	JFR;SSE;VVI;MVO;DDA
Date:	2011-01-26
Reviewed By:	JSI;JFR
Restrictions:	Partners Only

Approved by:

Date	CET	Initials	Name	Justification
2011-01-26	10:24:08	JFR	Jørgen Franck	on behalf of NTJ

CONFIDENTIAL

REVISION RECORD

Doc. Rev	Date	By	Pages affected	Brief description of changes
1	20080118	VVI JFR SSE	ALL	Initial draft
2	20091103	VVI	Section 3.8.1	Added description of FUNC_ID_BUTTON_PRESSED
2	20091126	VVI	Section 3.7	Added description of MTP Interface and updated description of the SRAM interface.
3	20100218	VVI	All	Added description of FUNC_ID_ZW0x0x_WRITE_OTP_STATS_READ. Added new description of FUNC_ID_TOGGLE_EEPROM_IF with support of detection an external NVM chip type. Added description of the lock bits support for ZW010x, ZW020x, ZW030x. Updated descriptions of all commands with additional information like on what ZW0x0x chip it is supported, etc. Removed not implemented commands.
4	20101021	DDA	Section 3.6	Added “Programming calibrated devices”.
5	20101102	DDA	Section 3.7.44	Added FUNC_ID_CALIBRATION_START

Table of Contents

Table of Figures

1 ABBREVIATIONS

Abbreviation	Explanation
COM	Serial port interface on IBM PC-compatible computers
GUI	Graphical User Interface
LSB	Least significant bit is the bit position in a binary number giving the units value, that is, determining whether the number is even or odd
MSB	Most significant bit is the bit position in a binary number having the greatest value
NAK	Negative-acknowledge character that is used to indicate that an error was detected in the previously received frame and that the receiver is ready to accept retransmission of that frame
OTP	One Time Programmable, a type of programmable read-only memory
NVM	Non-Volatile Memory
MTP	Multiple Time Programmable
PC	Personal computer
SOF	Start Of Frame
SPI	Serial Peripheral Interface Bus
SW	Software
UART	Universal Asynchronous Receiver/Transmitter is a piece of computer hardware that translates data between parallel and serial forms, is usually an individual (or part of an) integrated circuit used for serial communications over a computer or peripheral device serial port

2 INTRODUCTION

1.1 Purpose

The purpose of this document is to describe the Z-Wave Programmer communication interface.

1.2 Audience and prerequisites

The audience is Z-Wave partners.

3 COMMUNICATION INTERFACE

3.1 Introduction

The Z-Wave programmer includes two parts:

1. The firmware, which runs on Atmel ATMEGA128 processor.
2. The host Windows application, which runs on a PC.

These two components communicate using UART port. The UART runs with the following parameters: the transfer rate 115200 kbps, 1 start bit, 8 data bits, one stop bit and no parity bit.
The purpose of this document is to describe the communication protocol between the Z-Wave programmer's GUI and the firmware.

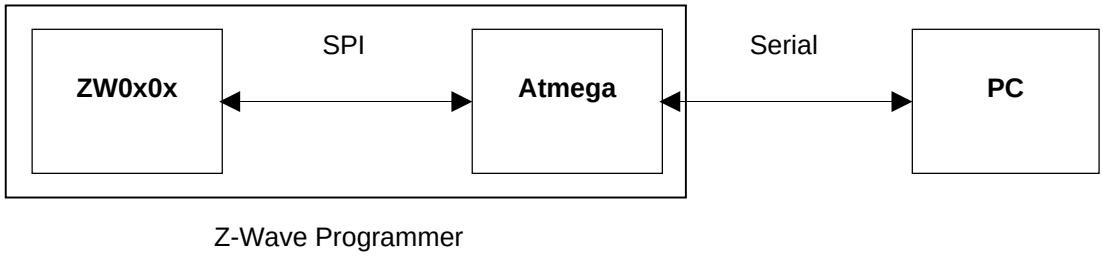


Figure 1. Overview of Z-Wave Programmer and PC including interfaces

3.2 Implementation

The Z-Wave programmers' firmware source code is provided on the Z-Wave Developer's Kit. Be aware that altering the function ID's and frame formats in the source code can result in interoperability problems with the Z-Wave Programmer's PC SW supplied on the Developer's Kit as well as commercially available GUI applications. Regarding how to determine the current version of the Z-Wave programmers' firmware, refer to the function ID FUNC_ID_ZW_GET_VERSION.

The Atmel ATMega128 firmware consists of the boot loader and Z-Wave programmer application. The boot loader updates the ATMega with a new version of the firmware when needed. The Z-Wave programmer application is responsible of programming the ZW0x0x single chips. After reset the ATMega always starts by executing the Z-Wave programmer application. The only way to execute the boot loader is by a command from the host SW. The boot loader is located on the upper 4k of the ATMega code space.

The following sections describe the communication protocol implementation and how a host can communicate with the firmware.

3.3 Frame Layout

The protocol between the PC (host) and the ATMega (ZW) consists of three frame types: ACK frame, NAK frame and Data frame. Each Data frame is prefixed with SOF byte and Length word and suffixed with a Checksum byte.

ACK frame:

The ACK frame is used to acknowledge a successful transmission of a data frame. The format is as follows:

7	6	5	4	3	2	1	0
ACK (0x06)							

NAK frame:

The NAK frame is used to de-acknowledge an unsuccessful transmission of a data frame. The format is as follows:

7	6	5	4	3	2	1	0
NAK (0x15)							

Only a frame with a LRC checksum error is de-acknowledged with a NAK frame.

Data frame:

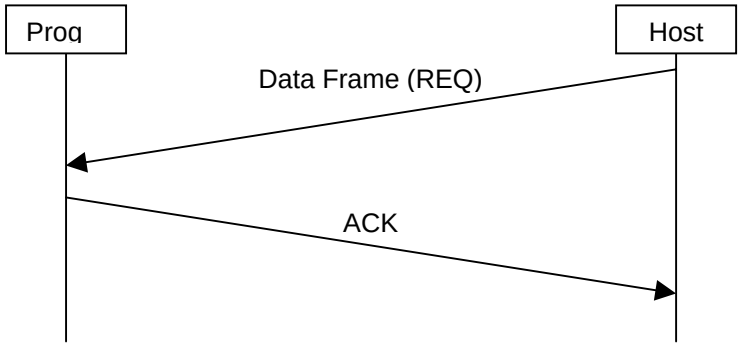
The Data frame contains the Z-Wave programmer's command including parameters for the command in question. The format is as follows:

7	6	5	4	3	2	1	0
SOF							
Length MSB							
Length LSB							
Type							
Z-Wave programmer's Command ID							
Command Specific Data							
…							
SeqNo							
Checksum							

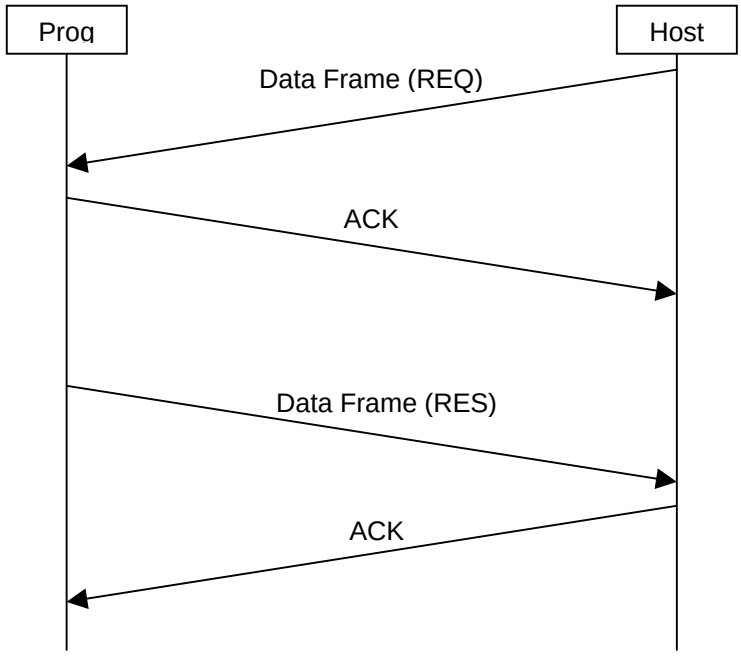
Field	Description
SOF	Start Of Frame. Used for synchronization and is equal to 0x01
Length	Number of bytes in the frame, exclusive SOF and Checksum. The host application is responsible for entering the correct length field. The current firmware does no validation of the length field.
Type	Used to distinguish between unsolicited calls and immediate responses. The request (REQ) is equal to 0x00 and response (RES) is equal to 0x01. The current Z-Wave programmer firmware doesn't make use of unsolicited calls.
Command ID	Unique command ID for the function to be carried out. Any data frames returned by this function will contain the same command ID
Command Specific Data	One or more bytes of command specific data. Possible callback handling is also defined here.
Checksum	LRC checksum used to check for frame integrity. Checksum calculation includes the **Length**, **Type**, **Command IS Data** and **Command Specific Data** fields. The Checksum is a XOR checksum with an initial checksum value of 0xFF. For a checksum implementation refer to the function ConTxFrame in the conhandle.c module
SeqNo	A byte which is incremented for every transmitted frame. If two the host receive 2 frames with the same sequence number the last one will be dropped out.

3.4 Frame Flow

The frame flow between a host and a ATMega (ZW) running the Z-Wave programmer's code depends on the API call. There are two different ways to conduct communication between the host and ZW.



Data frame from host, which is acknowledged by ATMega when successfully received. An example could be the commandI call FUNC_ID_PROGRAMMER_SET_LED.



Data frame from host, which is acknowledged by ATMega when successfully received. A data frame (RES) is returned by ATMega with the result at command completion. The host acknowledges the data frame when successfully received. An example could be the commandI call FUNC_ID_ZW0x0x_READ_PAGE.

3.5 Error handling

A number of scenarios exist, which can impede the normal frame flow between the host and the ATMega CPU running the Z-Wave programmer's firmware.

A LRC checksum failure is the only case there is de-acknowledged by a NAK frame. When a host receives a NAK frame can it either retry transmission of the frame or abandon the task. A task is defined as the whole frame flow associated with the execution of a specific Z-Wave programmer's command call. If a NAK frame is received by the ATMega in response to a just transmitted frame, then the frame in question is retransmitted (max 2 retries).

Frames with an illegal length are ignored without any notification. Frames with an illegal type (only REQ and RES exists) are ignored without any notification

The Z-Wave programmer's firmware can only perform one host-initiated task at a time. A data frame will be dropped without any notification (no ACK/NAK frame transmitted) by the ZW if it is not ready to execute a new host-initiated task.

3.6 Programming calibrated devices

ZM4101 and ZM4012 modules are per default preprogrammed with a calibration value in the OTP. This means that the OTP by default is not empty. The calibration value is stored in OTP address 0x06. The programming and verification process when handling calibrated devices should thus be as described below:

Blank testing: Waive for OTP content in addr. 0x06

Programming: Ensure that data value 0x00 is written to OTP addr. 0x06 (this leaves the calibration value in the OTP untouched)

Verifying: Waive for difference between OTP content and hex-file content for addr. 0x06.

3.7 Z-Wave programmer command Calls

3.7.1 FUNC_ID_ZW_GET_VERSION

The Z-Wave programmer host SW calls this command to get the firmware's version number

Defined in: commands.h

Frames Flow:

HOST>PROG: REQ | 0x06

PROG->HOST: RES| 0x06 | 0x00 | VERSION | REVISION

VERSION: the left part of firmware version number x.xx

REVISION: the right part of firmware version number x.xx

3.7.2 FUNC_ID_CONNECT_PROGRAMMER

Enter the programmer mode. When this command is executed, programmer could be used only to detect the presence of the hardware.

Defined in: commands.h

Frames flow:

HOST->PROG: REQ | 0x25

PROG->HOST: RES | 0x25 | Signature (MSB) | Signature (LSB)

Signature – Signature for device recognition (0xF002).

3.7.3 FUNC_ID_PROGRAMMER_SET_LED

Turn on / off the programmer status LEDs

Defined in: commands.h

Frames flow:

HOST->PROG: REQ | 0x53 | LedType | LedONOFF

LedType – type of LED (PASS_LED, BUSY_LED, ERR_LED)
LedONOFF – 1 to turn led ON, 0 – to turn LED OFF (ignored for PASS_LED and ERR_LED – this led always turns ON)

PROG->HOST: nothing

3.7.4 FUNC_ID_ZW0x0x_PROG_ENABLE

Enable the programming mode in the targeted ZW0x0x single chip. In this mode all memories of the target can be programmed: Program Memory (Flash in ZW030x, OTP in ZW040x), external NVM, MTP memory, etc. For more details about the Flash programming process, refer to [2]. For more details about the OTP programming process, refer to [3].

This function enables the Program Memory interface on the development board and performs a synchronization. The number of syncs performed is returned as a byte.

This function should be called first, before execution of any command, which working with the target ZW0x0x chip, external Non-Volatile Memory, MTP, etc. At the end of programming process the FUNC_ID_ZW0x0x_PROG_RELEASE function should be called to release the Reset signal of the target ZW0x0x chip and run its firmware.

Defined in: commands.h

Frames flow:

HOST->PROG: REQ | 0x40

PROG->HOST: RES | 0x40 | SYNC

SYNC: number of synchronizations trials. If SYNC is 0xFF, then synchronization has failed.

3.7.5 FUNC_ID_ZW0x0x_PROG_RELEASE

Release the ZW0x0x Flash programming interface.

This function release the Reset signal of the target ZW0x0x chip and run its firmware.

Defined in: commands.h

Frames flow:

HOST->PROG: REQ | 0x48 | ChipType

ChipType: Type of the targeted ZW0x0x single chip.

PROG->HOST: RES | 0x48 | DONE

DONE: DONE constant = 0x0d – operation done.

3.7.6 FUNC_ID_ZW0x0x_READ_SIG_BYTE

The command reads the ZW0x0x single chip seven signature bytes.

Defined in: commands.h

Frames flow:

HOST->PROG: REQ | 0x47 | ChipType

ChipType: Type of the targeted ZW0x0x single chip.

PROG->HOST: RES | 0x47 | Sig Bytes[7]

Sig Bytes: unique 7 bytes string

3.7.7 FUNC_ID_ZW0x0x_SET_WRITE_CYCLE

Set the ZW010x, ZW020x, ZW030x single chip FLASH write cycle time.

This function must not be called for ZW040x single chip targets.

Defined in: commands.h

Frames flow:

HOST->PROG: REQ | 0x46 | CTime

CTime: The ZW010x, ZW020x, ZW030x single chip FLASH's write cycle time. On how the cycle time is calculated, refer to [2]

PROG->HOST: RES | 0x46 | DONE

DONE: Constant value indicates (0x0D)

3.7.8 FUNC_ID_ZW0x0x_CHIP_ERASE

Erase the ZW010x, ZW020x, ZW030x single chip FLASH.

This function is ignored if called for ZW040x single chip targets. It returns FAIL.

Defined in: commands.h

Frames flow:

HOST->PROG: REQ | 0x46 | ChipType | CTime

ChipType: Type of the targeted ZW0x0x single chip.
CTime: The ZW0x0x single chip FLASH's write cycle time. On how the cycle time is calculated, refer to [2]

PROG->HOST: RES | 0x46 | Status

Status: result of operation – SUCCESS constant (0x31) if operation was done successful or FAIL constant (0x30) if operation was failed.

3.7.9 FUNC_ID_ZW0x0x_ERASE_PAGE

Erase a ZW020x, ZW030x single chip's FLASH page.

This function is ignored if called for ZW010x or ZW040x single chip targets. It returns FAIL.

Defined in: commands.h

Frames flow:

HOST->PROG: REQ | 0x52 | ChipType | PageNo

ChipType: Type of the targeted ZW0x0x single chip.
PageNo: The FLASH page number to erase.

PROG->HOST: RES | 0x52 | Status

Status: result of operation – SUCCESS constant (0x31) if operation was done successful or FAIL constant (0x30) if operation was failed.

Note: the Command FUNC_ID_ZW0x0x_SET_WRITE_CYCLE should be called before once FUNC_ID_ZW0x0x_ERASE_PAGE.

3.7.10 FUNC_ID_ZW0x0x_WRITE_PAGE

The command writes data to a page in the FLASH of ZW010x, ZW020x, ZW030x single chips.

The programmer sends the status of the operation after the end of the writing process.

For ZW040x: command writes data to the ZW040x OTP, by logical pages of 256 bytes.

Defined in: commands.h

Frames flow:

HOST->PROG: REQ | 0x43 | ChipType | PageNo | Verfiy | flash data[ ]

ChipType: Type of the targeted ZW0x0x single chip.
PageNo: the FLASH / OTP page number to write data in.
Verify: A Boolean. If true compare the data written to the FLASH page to the original data. If false no comparison is done.
Flash data: the content of the FLASH page to be written to it. For ZW040x: contents of the 256 bytes page to be written to OTP.

PROG->HOST: RES | 0x43 | Status

Status: the status of the write operation.

FAIL constant (0x30) - write operation failed.

SUCCESS constant (0x31) - write operation succeeded

Note for the ZW010x, ZW020x, ZW030x: the Command FUNC_ID_ZW0x0x_SET_WRITE_CYCLE should be called once before FUNC_ID_ZW0x0x_WRITE_PAGE.

3.7.11 FUNC_ID_ZW0x0x_READ_PAGE

The command read the content of a ZW010x, ZW020x, ZW030x single chips FLASH page. For ZW040x – it reads the OTP logical page of 256 bytes.

Defined in: commands.h

Frames flow:

HOST->PROG: REQ | 0x42 | ChipType | PageNo |

ChipType: Type of the targeted ZW0x0x single chip.
PageNo: the FLASH / OTP page number to read data from.

PROG->HOST: RES | 0x43 | flash data[]

Status: the status of the write operation.
Flash data: the content of the FLASH / OTP page.

3.7.12 FUNC_ID_ZW0x0x_SRAM_WRITE_PAGE

The command writes data to the SRAM area of ZW040x, by logical pages of 256 bytes.

This function is ignored if called for ZW010x, ZW020x or ZW030x single chip targets. It returns FAIL.

Defined in: commands.h

Frames flow:

HOST->PROG: REQ | 0x61 | ChipType | PageNo | IsVerify | SRAM data[]

ChipType: Type of the targeted ZW0x0x single chip.
PageNo: the SRAM page number to write data in.
IsVerify: A Boolean. If true (0x01) - compare the data written to the SRAM page to the original data. If false (0x00) - no comparison is done.
SRAM data: the content of the SRAM page to be written to it.

PROG->HOST: RES | 0x61 | Status

Status: result of operation:
SUCCESS constant - if operation was done successful: all data are written to the SRAM page, and, if IsVerify = true, same data read back from SRAM Page.
FAIL constant - if write operation was failed, or, if IsVerify = true, read back data is not the same.

3.7.13 FUNC_ID_ZW0x0x_SRAM_READ_PAGE

The command read data from the SRAM area of ZW040x, by logical pages of 256 bytes.

This function is ignored if called for ZW010x, ZW020x or ZW030x single chip targets. It returns FAIL.

Defined in: commands.h

Frames flow:

HOST->PROG: REQ | 0x60 | ChipType | PageNo

ChipType: Type of the targeted ZW0x0x single chip.
PageNo: the SRAM page number to read data from.

PROG->HOST: RES | 0x60 | SRAM data[]

SRAM data: the content of the readed SRAM page.

3.7.14 FUNC_ID_ZW0x0x_CRC_CHECK

Run CRC Check of OTP memory of ZW040x single chip.

This function is ignored if called for ZW010x, ZW020x or ZW030x single chip targets. It returns FAIL.

Executing the 'Run CRC Check' will start a built-in CRC-32 generator in the chip that will read the whole OTP and calculate the CRC-32 checksum.

To be able to utilize the built-in CRC-32 check the upper four bytes of the OTP memory must be programmed with a pre-calculated CRC-32 checksum. That is, after generating the OTP code a CRC-32 checksum must be calculated of the resulting OTP memory (locations that isn't programmed is 00h) OTP code and the result must be placed at the last 4 byte positions in the code space.

Defined in: commands.h

Frames flow:

HOST->PROG: REQ | 0x66 | ChipType

ChipType: Type of the targeted ZW0x0x single chip.

PROG->HOST: RES | 0x66 | Status

Status: result of operation – SUCCESS constant (0x31) if operation was done successful or FAIL constant (0x30) if operation was failed.

3.7.15 FUNC_ID_ZW0x0x_SRAM_EXECUTE

The command initiates an "Execute Out Of SRAM" mode of the ZW040x single chip targets.

This function is ignored if called for ZW010x, ZW020x or ZW030x single chip targets. It returns FAIL.

4k of SRAM from base address 0x0000 is used for execution.

Defined in: commands.h

Frames flow:

HOST->PROG: REQ | 0x62 | ChipType

ChipType: Type of the targeted ZW0x0x single chip.

PROG->HOST: RES | 0x62 | Status

Status: result of operation – SUCCESS constant if operation was done successful or FAIL constant if operation was failed.

3.7.16 FUNC_ID_ZW0x0x_DEV_MODE_ENABLE

The command initiates the “Development Mode” of ZW040x single chip targets.

This function is ignored if called for ZW010x, ZW020x or ZW030x single chip targets. It returns FAIL.

In this mode the contents of upper 12k (from 52k to 64k) of OTP memory is replaced by contents of of upper 12k (from 4k to 16k) of SRAM.

Defined in: commands.h

Frames flow:

HOST->PROG: REQ | 0x65 | ChipType

ChipType: Type of the targeted ZW0x0x single chip.

PROG->HOST: RES | 0x65 | Status

Status: result of operation – SUCCESS constant if operation was done successful or FAIL constant if operation was failed.

3.7.17 FUNC_ID_ZW0x0x_STATUS_READ

The command reads and returns the OTP status byte of ZW040x single chip targets.

This function is ignored if called for ZW010x, ZW020x or ZW030x single chip targets. It returns 0 (zero).

Defined in: commands.h

Frames flow:

HOST->PROG: REQ | 0x64 | ChipType

ChipType: Type of the targeted ZW0x0x single chip.

PROG->HOST: RES | 0x64 | StatusByte

StatusByte: byte with combination of bits from table below:

Bit	***Constant***	***Description***
0	STATE_CRC_BUSY	This bit will go high when a ‘Run CRC Check’ command has been sent to the Single Chip. It will return to low when the CRC check procedure is done

Bit	*Constant*	*Description*
1	STATE_CRC_DONE	This bit is cleared when a 'Run CRC Check' command is issued and it will be set if the CRC check procedure passes.
2	STATE_CRC_FAILED	This bit is cleared when a 'Run CRC Check' command is issued and it will be set if the CRC check procedure fails.
3	STATE_WRITE_BUSY	This bit is high if the OTP programming logic is busy programming the OTP
4	STATE_WRITE_FAILED	This bit is cleared when a 'Write OTP' command is issued and it will be set if the OTP write operation fails.
5	STATE_CONTINUE_FEFUSED	This bit will be set if either a 'Continue Write Operation' or a Continue Read Operation' are refused. These operations will be refused if: A 'Continue Write Operation' is not succeeding a 'Write SRAM' or a 'Continue Write Operation' command A 'Continue Read Operation' is not succeeding a 'Read OTP', a 'Read SRAM' or a 'Continue Read Operation' command
6	STATE_DEV_MODE_ENABLED	This bit is set if the 'Development Mode' has been enabled
7	STATE_EXEC_SRAM_MODE_ENABLED	This bit is set if the 'Execute out of SRAM' Mode has been enabled

3.7.18 FUNC_ID_ZW0x0x_WRITE_OTP_STATS_READ

The command reads and returns the OTP Write Statistics of ZW040x single chip targets.

This function is ignored if called for ZW010x, ZW020x or ZW030x single chip targets. It returns 0 (zero).

After all code data has been programmed into the OTP it is possible to read the programming statistics. The statistics can be read as a number that tells to how many locations the OTP state machine were forced to use excessive writes. A write operation to a certain OTP memory location is noted as an "Excessive write" when byte write operation first fails 3 times then is following by more than 9 times of bit write operations before the data byte finally is verified.

This number can be used to check the quality of the OTP memory cells. It tells you how often the programming process was close to fail and thereby to determine if the wafer production somehow is degraded.

Defined in: commands.h

Frames flow:

HOST->PROG: REQ | 0x6C | ChipType

ChipType: Type of the targeted ZW0x0x single chip.

PROG->HOST: RES | 0x6C | WriteOTPStats(MSB) | WriteOTPStats(LSB)

WriteOTPStats: count of excessive writes.

3.7.19 FUNC_ID_ZW0x0x_READ_LOCKBITS

The command reads the Flash / OTP lock bits of the ZW0x0x single chip targets.

Defined in: commands.h

Frames flow:

HOST->PROG: REQ | 0x44 | ChipType

ChipType: Type of the targeted ZW0x0x single chip.

PROG->HOST: RES | 0x44 | LockBits

LockBits: for ZW010x, ZW020x andZW030x this command is returns state of 5 lock bits;

for ZW040x it returns the state of two lock bits.

The table below displays the meaning of lock bits for ZW010x, ZW020x, ZW030x:

Bit	***Name***	***Description***
7:5	-	Reserved, write as '0'
4	BOBLOCK	Boot Block Lock 0: Page 0 is write protected 1: Page 0 is writeable, unless BSIZE = 000
3:1	BSIZE	Boot Sector Size 000: 32768 bytes (all) 001: 16384 bytes 010: 8192 bytes 011: 4096 bytes 100: 2048 bytes 101: 1024 bytes 110: 512 bytes 111: 0 bytes
0	SPIRE	SPI Read Flash Enable 0: SPI interface is not allowed to read flash data 1: SPI interface is allowed to read flash data

The table below displays the meaning of lock bits for ZW040x:

Bit	***Description***
0	Read back disabled. The contents of the OTP cannot be read back through the SPI interface, if this bit is set.
1	Development Mode disabled. The option to use SRAM as code memory in the so called "Development Mode" is permanently disabled if this bit is set.

3.7.20 FUNC_ID_ZW0x0x_WRITE_LOCKBITS

Write the ZW0x0x Flash / OTP lock bits settings.

Defined in: commands.h

Frames flow:

HOST->PROG: REQ | 0x45 | ChipType | LockBits

ChipType: Type of the targeted ZW0x0x single chip.
LockBits: Lock bits - see table in FUNC_ID_ZW0x0x_READ_LOCKBITS command for values.

PROG->HOST: RES | 0x45 | Status

Status: result of operation – SUCCESS constant if operation was done successful or FAIL constant if operation was failed.

3.7.21 FUNC_ID_ZW0x0x_MODEM_BIT_WRITE

The command writes the modem bit to the OTP of the ZW040x single chip targets.

This function is ignored if called for ZW010x, ZW020x or ZW030x single chip targets. It returns FAIL.

Defined in: commands.h

Frames flow:

HOST->PROG: REQ | 0x63 | ChipType

ChipType: Type of the targeted ZW0x0x single chip.

PROG->HOST: RES | 0x63 | Status

Status: result of operation – SUCCESS constant if operation was done successful or FAIL constant if operation was failed.

3.7.22 FUNC_ID_TOGGLE_MTP_IF

Enable / Disable the interface used to write / read the MTP memory of the ZW040x single chip targets.

This function is ignored if called for ZW010x, ZW020x or ZW030x single chip targets. It returns FAIL.

Before executing this or any other functions, the FUNC_ID_ZW0x0x_PROG_ENABLE function should be called to enter the programming mode.

Defined in: commands.h

Frames flow:

HOST->PROG: REQ | 0x68 | IsEnable | ChipType

IsEnable = 1 to enable MTP memory programming; 0 – to exit MTP memory programming.

ChipType: Type of the targeted ZW0x0x single chip.

PROG->HOST: RES | 0x68 | Status

Status: result of operation:
SUCCESS constant - if operation was done successful.
FAIL constant - if operation was failed: Normal working mode of the ZW040x chip can't be set – power cycling needed!

3.7.23 FUNC_ID_MTP_FILL

Fill the MTP memory of the ZW040x single chip targets with specified value (erase).

This function is ignored if called for ZW010x, ZW020x or ZW030x single chip targets. It returns FAIL.

Defined in: commands.h

Frames flow:

HOST->PROG: REQ | 0x69 | ChipType | FillValue

ChipType: Type of the targeted ZW0x0x single chip.
FillValue: Each byte of the ZW040x single chip MTP memory will be set to this value.

PROG->HOST: RES | 0x69 | Status

Status: result of operation:
SUCCESS constant - if operation was done successful: each byte of MTP memory is set to FillValue and also readed back as FillValue.
FAIL constant - if operation was failed.

3.7.24 FUNC_ID_MTP_READ_PAGE

The command read data from the all 64 bytes of the MTP memory area of ZW040x.

This function is ignored if called for ZW010x, ZW020x or ZW030x single chip targets. It returns zero length MTP data[].

Defined in: commands.h

Frames flow:

HOST->PROG: REQ | 0x6A | ChipType | PageNo

ChipType: Type of the targeted ZW0x0x single chip.
PageNo: Currently not used by Z-Wave Programmer firmware but must be sent as 0 (zero).

PROG->HOST: RES | 0x6A | MTP data[]

MTP data: the content of the MTP memory (all 64 bytes).

3.7.25 FUNC_ID_MTP_WRITE_PAGE

The command writes data to the all 64 bytes of the MTP memory area of ZW040x.

This function is ignored if called for ZW010x, ZW020x or ZW030x single chip targets. It returns FAIL.

Defined in: commands.h

Frames flow:

HOST->PROG: REQ | 0x6B | ChipType | PageNo | IsVerify | MTP data[]

ChipType: Type of the targeted ZW0x0x single chip.
PageNo: Currently not used by Z-Wave Programmer firmware but must be sent as 0 (zero).
IsVerify: A Boolean. If true (0x01) - compare the data written to the MTP memory to the original data. If false (0x00) - no comparison is done.
MTP data: the content of the MTP memory (all 64 bytes) to be written to it.

PROG->HOST: RES | 0x6B | Status

Status: result of operation:
SUCCESS constant - if operation was done successful: all data are written to the MTP, and, if IsVerify = true, same data readed back from MTP Memory.
FAIL constant - if write operation was failed, or, if IsVerify = true, readed back data is not the same.

3.7.26 FUNC_ID_TOGGLE_EEPROM_IF

Enable / Disable the interface used to write / read the external NVM.

Before executing this or any other functions, the FUNC_ID_ZW0x0x_PROG_ENABLE function should be called to enter the programming mode.

Defined in: commands.h

Frames flow:

HOST->PROG: REQ | 0x51 | IsEnable | ChipType

IsEnable = 1 to enable external NVM programming mode; 0 – to exit external NVM programming mode.
ChipType: Type of the targeted ZW0x0x single chip.

PROG->HOST: RES | 0x51 | BusType | ManufacturerID | DeviceID | SizeID

BusType, ManufacturerID, DeviceID, SizeID – detected external NVM chip type.
BusType: Bus and protocol type:
0x00 – invalid;
0x01 - Serial EEPROM chip with SPI bus protocol;
0x02 - Serial Flash chip on SPI bus;
ManufacturerID: external NVM chip Manufacturer ID. 0x00 - unknown manufacturer id.
DeviceID: external NVM chip Device ID. 0x00 - unknown device id.

SizeID: external NVM chip Memory size mask. Memory size = SizeID * 8 KBytes. 0x00 - unknown size.

Values of the external NVM chip types (BusType is high byte and so on):

0x00000000 - in case of error (can't detect external NVM chip type, external NVM is not connected, etc.);

0x00010000 - No errors, but chip type is unknown at this stage;

0x01000000 - Serial EEPROM chip with SPI bus protocol like in Atmel AT25128 serial EEPROM and compatible (ST95128, etc.);

0x02000000 - Serial Flash chip on SPI bus;

0x02208010 - Numonyx M25PE10;

0x02208020 - Numonyx M25PE20;

0x02208040 - Numonyx M25PE40;

3.7.27 FUNC_ID_MEMORY_GET_BYTE

Read a byte from the external NVM.

Defined in: commands.h

Frames flow:

HOST->PROG: REQ | 0x08 | offset(MSB) | offset (LSB)

offset: address in external NVM to read byte from.

PROG->HOST: RES | 0x08 | data byte

data byte: readed external NVM byte.

3.7.28 FUNC_ID_MEMORY_PUT_BYTE

Write a byte to the external NVM.

Defined in: commands.h

Frames flow:

HOST->PROG: REQ | 0x09 | old eeprom | offset(MSB) | offset (LSB) | data

old eeprom: value of this byte is not used by Z-Wave programmer;
offset: address in external NVM to write byte to;
data: byte to write;

PROG->HOST: RES | 0x09 | data byte

data byte: byte read back from external NVM after write (must be same as data byte in request; if not – then write failed).

3.7.29 FUNC_ID_MEMORY_GET_BUFFER

Read data bytes from the external NVM.

Defined in: commands.h

Frames flow:

HOST->PROG: REQ | 0x10 | old EEPROM | offset(MSB) | offset (LSB) | length

old EEPROM: value not used;
offset: address of start of block to read from external NVM;
length: length of block of external NVM to read.

PROG->HOST: RES | 0x10 | offset(MSB) | offset (LSB) | data[]

offset: address of start of block in external NVM;
data: data bytes of external NVM block to read.

3.7.30 FUNC_ID_MEMORY_PUT_BUFFER

Writes data bytes to the external NVM.

Defined in: commands.h

Frames flow:

HOST->PROG: REQ | 0x11 | old EEPROM | offset (MSB) | offrest (LSB) | length | buffer[]

old EEPROM: value not used;
offset: address of start of block to write;
length: length of block to write;
buffer[]: data to write;
funcID: value not used.

PROG->HOST: RES | 0x11 | DONE

DONE: DONE constant = 0x0d – operation done.

3.7.31 FUNC_ID_BLOCK_SET_EEP

Write a specific value to block of external NVM address.

Defined in: commands.h

Frames flow:

HOST->PROG: REQ | 0x05 | old EEPROM | startAdr(MSB) | startAdr(LSB) | endAdr(MSB) | endAdr(LSB) | value to set

old eeprom: value is ignored
startAdr: Start address of the block.
endAdr: End address of the block.
value to set: value.

PROG->HOST: RES | cmd | DONE

DONE: DONE constant = 0x0d – operation done.

3.7.32 FUNC_ID_MEMORY_GET_ID

Read home ID from the external NVM.

Defined in: commands.h

Frames flow:

HOST->PROG: REQ | 0x07

PROG->HOST: RES | 0x07 | Home ID 1st byte (highest) | Home ID 2nd byte | Home ID 3rd byte | Home ID 4th byte

Home ID: home ID from external NVM.

3.7.33 FUNC_ID_STORE_HOMEID

Write home ID to the external NVM.

Defined in: commands.h

Frames flow:

HOST->PROG: REQ | 0x13 | old eeprom | Home ID 1st byte (highest) | Home ID 2nd byte | Home ID 3rd byte | Home ID 4th byte

Home ID: new Home ID to write to the external NVM.

PROG->HOST: RES | 0x13 | DONE

DONE: DONE constant = 0x0d – operation done.

3.7.34 FUNC_ID_M128_ENTER_PROG_MODE

Enter the boot loader mode for programming the ATMEL ATMEGA128 FLASH.

Defined in: commands.h

Frames flow:

HOST->PROG: REQ | 0x27 | btSig 1 | btSig 2 | btSig 3 | btSig 4 | btSig 5

btSig – Boot Signature, array that hold 4 bytes random value and a checksum that should be verified before programming the ATMEL

PROG->HOST: RES | 0x27 | DONE

DONE: DONE constant = 0x0d – operation done.

3.7.35 FUNC_ID_EXIT_PROG_MODE

The command exits the boot loader mode.

Defined in: commands.h

Frames flow:

HOST->PROG: REQ | 0x24

PROG->HOST: RES | 0x24 | DONE

3.7.36 FUNC_ID_M128_GET_SW_VER

Get the boot loader version.

Defined in: commands.h

Frames flow:

HOST->PROG: REQ | 0x26

PROG->HOST: RES | 0x26 | VERSION | REVISION

VERSION: the left part of firmware version number x.xx
REVISION: the right part of firmware version number x.xx

3.7.37 FUNC_ID_M128_CHIP_ERASE

Erase the ATMEL ATMEGA128 FLASH (from 0x00000 to 0x1EFFF)

Defined in: commands.h

Frames flow:

HOST->PROG: REQ | 0x16

PROG->HOST: RES | 0x16 | erase status | btSig 1 | btSig 2 | btSig 3 | btSig 4 | btSig 5

erase status = DONE, when erase done successfully; = 0xF0, when the boot loader signature (btSig) is not correct;
btSig – current boot loader signature, see FUNC_ID_M128_ENTER_PROG_MODE command for more info.

3.7.38 FUNC_ID_M128_BLOCK_WRITE

Write block of data bytes to the ATMEL ATMEGA128 FLASH.

Defined in: commands.h

Frames flow:

HOST->PROG: REQ | 0x18 | Page Number (MSB) | Page Number (LSB) | verify | data[]

Page Number: number of ATMEL ATMEGA128 FLASH page to write to;
verify: value should be 1 if written page should be verified;
data[]: data bytes to be written for the ATMEL ATMEGA128 FLASH.

PROG->HOST: RES | 0x18 | Status

Status: result of operation – SUCCESS constant if operation was done successful or FAIL constant if operation was failed.

3.7.39 FUNC_ID_M128_BLOCK_READ

Read block of data bytes from the ATMEL ATMEGA128 FLASH.

Defined in: commands.h

Frames flow:

HOST->PROG: REQ | 0x19 | Page Number (MSB) | Page Number (LSB)

Page Number: number of ATMEL ATMEGA128 FLASH page to read from;

PROG->HOST: RES | 0x19 | data readed[]

data readed[]: data bytes, readed from the ATMEL ATMEGA128 FLASH.

3.7.40 FUNC_ID_M128_GET_LOCKBITS

(Not implemented)

Defined in: commands.h

Frames flow:

HOST->PROG: REQ | 0x20

PROG->HOST: RES | 0x20 | lock bits byte

3.7.41 FUNC_ID_M128_GET_FUSEBITS_LOW

(Not implemented)

Defined in: commands.h

Frames flow:

HOST->PROG: REQ | 0x21

PROG->HOST: RES | 0x21 | fuse bits low byte

3.7.42 FUNC_ID_M128_GET_FUSEBITS_HIGH

(Not implemented)

Defined in: commands.h

Frames flow:

HOST->PROG: REQ | 0x22

PROG->HOST: RES | 0x22 | fuse bits high byte

3.7.43 FUNC_ID_M128_GET_FUSEBITS_EXT

(Not implemented)

Defined in: commands.h

Frames flow:

HOST->PROG: REQ | 0x23

PROG->HOST: RES | 0x23 | fuse bits ext byte

3.7.44 FUNC_ID_CALIBRATION_START

The command initiates “Calibration” mode of the ZW040x single chip targets.

This function is ignored if called for ZW010x, ZW020x or ZW030x single chip targets. It returns FAIL.

4k of SRAM from base address 0x0000 is used for execution of the calibration application pre-loaded into SRAM.

Defined in: commands.h

Frames flow:

HOST->PROG: REQ | 0x71 | ChipType

ChipType: Type of the targeted ZW0x0x single chip.

PROG->HOST: RES | 0x71 | Status

Status: result of operation – SUCCESS constant if operation was done successful or FAIL constant if operation was failed.

3.8 Z-Wave programmer unsolicited commands

This commands Z-Wave programmer send to the host (PC) as unsolicited frames with type REQUEST.

3.8.1 FUNC_ID_BUTTON_PRESSED

This unsolicited command notifies the host, that S1 button of Z-Wave Programmer PCB was pressed.

Defined in: commands.h

Frames flow:

PROG->HOST: REQ | 0x67 | ButtonsState

ButtonsState: State of the buttons: bit 0 = 1 if S1 pushbutton of Z-Wave Programmer PCB is pressed.

REFERENCES

[1] Zensys, INS10034, Instruction, Z-Wave Development
[2] Zensys, APL10312-7, Application Note Programming the 200 and 300 Series Z-Wave Single Chip Flash
[3] Zensys, INS10795, 400 Series Z-Wave Single Chip Programming Mode
[4] Zensys, INS10679, Z-Wave Programmer User Guide

INDEX

F