# 500 Series Z-Wave Chip Programming Mode

<table>
<thead>
<tr>
<th><strong>Document No.:</strong></th>
<th>INS11681</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Version:</strong></td>
<td>13</td>
</tr>
<tr>
<td><strong>Description:</strong></td>
<td>This document describes the operations necessary in order to program the Flash memory and lock bits and how to use the &quot;Execute out of SRAM&quot; mode in the 500 Series Z Wave chip/modules.</td>
</tr>
<tr>
<td><strong>Written By:</strong></td>
<td>MVO;ANI;MHANSEN;JFR;OPP;BBR</td>
</tr>
<tr>
<td><strong>Date:</strong></td>
<td>2018-03-05</td>
</tr>
<tr>
<td><strong>Reviewed By:</strong></td>
<td>ANI;OPP;JFR</td>
</tr>
<tr>
<td><strong>Restrictions:</strong></td>
<td>Public</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Approved by:</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
</tr>
<tr>
<td>2018-03-05</td>
</tr>
</tbody>
</table>
## REVISION RECORD

<table>
<thead>
<tr>
<th>Doc. Rev</th>
<th>Date</th>
<th>By</th>
<th>Pages affected</th>
<th>Brief description of changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1A</td>
<td>20120627</td>
<td>MVO</td>
<td>All</td>
<td>Initial Draft</td>
</tr>
<tr>
<td>1B</td>
<td>20120627</td>
<td>MVO</td>
<td>All</td>
<td>Corrected and elaborated</td>
</tr>
<tr>
<td>1C</td>
<td>20120711</td>
<td>MVO</td>
<td>Sections 4, 5, 7.4 and 7.14</td>
<td>Elaborated</td>
</tr>
<tr>
<td>2A</td>
<td>20130506</td>
<td>MVO</td>
<td>Corrected Erase Sector command</td>
<td></td>
</tr>
<tr>
<td>2C</td>
<td>20130513</td>
<td>MVO</td>
<td>Section 7.3</td>
<td>Changed the Flash programming procedure</td>
</tr>
<tr>
<td>2D</td>
<td>20130513</td>
<td>MVO</td>
<td>Section 7.3</td>
<td>Fixed missing picture and elaborated text</td>
</tr>
<tr>
<td>3A</td>
<td>20130719</td>
<td>MVO</td>
<td>Section 7.6 and section 9</td>
<td>Changed and simplified the Flash sector programming algorithm</td>
</tr>
<tr>
<td>3B</td>
<td>20130815</td>
<td>MVO</td>
<td>Section 4.4</td>
<td>Added note about the XRAM areas that is used by the USB circuit</td>
</tr>
<tr>
<td>3C</td>
<td>20130823</td>
<td>MVO</td>
<td>Section 3.1</td>
<td>Corrected read back byte value</td>
</tr>
<tr>
<td>4A</td>
<td>20130828</td>
<td>MVO</td>
<td>Section 7.9.1</td>
<td>Added notes that stresses that &quot;Check State &quot; must be used after</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>All</td>
<td>programming/erase command</td>
</tr>
<tr>
<td>4B</td>
<td>20130927</td>
<td>MVO</td>
<td>Section 5.2.2</td>
<td>Added the need for contact to the RESET_N</td>
</tr>
<tr>
<td>5A</td>
<td>20130930</td>
<td>MVO</td>
<td>Section 3.4.1</td>
<td>Added missing doc reference</td>
</tr>
<tr>
<td>6A</td>
<td>20131016</td>
<td>MVO</td>
<td>Section 6.3</td>
<td>Changed USB programming protocol</td>
</tr>
<tr>
<td>6B</td>
<td>20131101</td>
<td>MVO</td>
<td>Section 6.3</td>
<td>Changed requirements for the high period of the RESET_N signal</td>
</tr>
<tr>
<td>6C</td>
<td>20131210</td>
<td>MVO</td>
<td>Section 6.2</td>
<td>Elaborated note about omitted stop bits</td>
</tr>
<tr>
<td>6D</td>
<td>20131212</td>
<td>MVO</td>
<td>Section 6.2</td>
<td>Changed REST_N low period description</td>
</tr>
<tr>
<td>7A</td>
<td>20140311</td>
<td>ANI</td>
<td>Section 6.3 and 9</td>
<td>Added USB HWID check to description in section 6.3 and the flowchart in section 9. Added recommendation of max. 3 commands at a time to section 6.3 Now recommends to use UART programming interface for SD3503 and ZM5304 Updated timing values</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MVO</td>
<td>Table 8</td>
<td></td>
</tr>
<tr>
<td>7B</td>
<td>20140321</td>
<td>MVO</td>
<td>Section 5.2.2</td>
<td>UART programming interface for SD3503 and ZM5304 is now mandatory</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Table 3</td>
<td>Added missing lines</td>
</tr>
<tr>
<td>8A/8B</td>
<td>20140425</td>
<td>MVO</td>
<td>Figure 23</td>
<td>Added a path that covers the case where APM mode is enabled and RESET_N is asserted. Moved &quot;Program NVR &quot; to an earlier stage in the flow. Added check of NVR CRC16. Corrected text &quot;clear AutoProg0&quot; →&quot;Autoprogr1&quot; &quot;Handle Error&quot; flow added Elaborated New calibration section added</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Figure 25</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Section 8</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Section 4.6</td>
<td></td>
</tr>
<tr>
<td>8C</td>
<td>20140430</td>
<td>MVO</td>
<td>Section 4.6</td>
<td>Changed wording and fixed typos</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Section 8</td>
<td>Changed wording and fixed typos</td>
</tr>
</tbody>
</table>
## REVISION RECORD

<table>
<thead>
<tr>
<th>Doc. Rev</th>
<th>Date</th>
<th>By</th>
<th>Pages affected</th>
<th>Brief description of changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>8D</td>
<td>20140502</td>
<td>MVO</td>
<td>Sections 7.3.2 and 7.6</td>
<td>Emphasized that it is only the lower 2 kB part of the SRAM that is used as buffer when programming a flash sector. Added figure of flash sectors Deleted the incorrect statement that said that the 12kB XRAM couldn't be accessed through the programming interface. Added note on how to handle the case when RESET_N is set low and APM is enabled.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Section 3.4.1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Section 3.4.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Section 5.2</td>
<td></td>
</tr>
<tr>
<td>8E</td>
<td>20140505</td>
<td>MVO</td>
<td>Sections 3.4.2 and 5.2</td>
<td>Fixed typo/poor wording</td>
</tr>
<tr>
<td>9A</td>
<td>20140701</td>
<td>MVO</td>
<td>Section 7.14</td>
<td>Corrected 2nd calculation example</td>
</tr>
<tr>
<td>10A</td>
<td>20140918</td>
<td>MVO</td>
<td>Section 7.3.2</td>
<td>Changed sector programming algorithm and moved description to section 9 Changed CRC check duration Now depicts the modified sector programming algorithm Modified command strings</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Table 9</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Figure 28</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Table 3</td>
<td></td>
</tr>
<tr>
<td>10B</td>
<td>20141020</td>
<td>MVO</td>
<td>Section 4</td>
<td>Added note about wire lengths and connectors</td>
</tr>
<tr>
<td>11A</td>
<td>20150114</td>
<td>MVO</td>
<td>Table 4</td>
<td>Changed &quot;passes&quot; to &quot;completes&quot; Changed &quot;NVR&quot; to &quot;NVM&quot; Added process step to &quot;Main programming flow chart&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Section 11</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Section 9 and Figure 23</td>
<td></td>
</tr>
<tr>
<td>11B</td>
<td>20150408</td>
<td>MVO</td>
<td>Table 4</td>
<td>Corrected description of &quot;FlashFSM busy&quot; Elaborated text in various flow boxes Elaborated on the description of the RESET_N signal.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Figure 23</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Section 11</td>
<td></td>
</tr>
<tr>
<td>11C</td>
<td>20150408</td>
<td>MVO</td>
<td>Table 3</td>
<td>Removed red marking</td>
</tr>
<tr>
<td>12A</td>
<td>20150414</td>
<td>MVO</td>
<td>Section 5.2</td>
<td>Clarified the approaches to enable/disable APM mode. Fixed reference. Added process step to &quot;Main programming flow chart&quot; Elaborated about problems going to APM mode. Added info about optional series resistor Fixed doc references, added SD3503 datasheet to list</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OPP</td>
<td>Section 9</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Section 4.3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>References</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>20180305</td>
<td>BBR</td>
<td>All</td>
<td>Added Silicon Labs template</td>
</tr>
</tbody>
</table>
# Table of Contents

1 ABBREVIATIONS ................................................................................................................................. 1

2 INTRODUCTION ................................................................................................................................. 1

2.1 Purpose ........................................................................................................................................ 1

2.2 Audience and prerequisites ............................................................................................................ 1

3 OVERVIEW .......................................................................................................................................... 2

3.1 Programming a 500 Series Z-Wave Chip ......................................................................................... 2

3.2 Programming Interfaces .................................................................................................................... 2

3.3 Signature ....................................................................................................................................... 2

3.4 Memories ...................................................................................................................................... 2

3.4.1 Flash Memory ............................................................................................................................. 4

3.4.2 SRAM’s ..................................................................................................................................... 4

3.5 Lock Bits ..................................................................................................................................... 5

3.6 CRC checksum ............................................................................................................................... 5

3.7 Modes ......................................................................................................................................... 5

4 HARDWARE INTERFACES .................................................................................................................. 6

4.1 Power .......................................................................................................................................... 6

4.2 Using SPI as Programming Interface ............................................................................................... 7

4.3 Using UART as Programming Interface ........................................................................................... 8

4.4 Using USB as Programming Interface ............................................................................................. 9

4.5 System Clock ................................................................................................................................. 9

4.6 Required Calibration ....................................................................................................................... 10

5 ENTERING PROGRAMMING MODE ................................................................................................. 12

5.1 Enabling Programming Mode ........................................................................................................ 12

5.2 Auto Programming Mode ............................................................................................................... 12

5.2.1 Enable/disable APM mode ......................................................................................................... 14

5.2.2 Programming SD3503/ZM5304 in production .......................................................................... 14

6 PROTOCOLS ....................................................................................................................................... 16

6.1 SPI Protocol .................................................................................................................................. 16

6.2 UART protocol ............................................................................................................................... 17

6.3 USB protocol ................................................................................................................................ 18

6.3.1 USB Hardware ID (HWID) .......................................................................................................... 19

7 COMMAND SET ............................................................................................................................... 20

7.1 Read Flash ..................................................................................................................................... 22

7.2 Read SRAM/Continue Read ............................................................................................................ 22

7.3 Write SRAM/Continue Write .......................................................................................................... 22

7.3.1 Write SRAM only....................................................................................................................... 22

7.3.2 Use SRAM as intermediate buffer for programming Flash ..................................................... 22

7.4 Erase Chip .................................................................................................................................... 23

7.5 Erase Sector .................................................................................................................................. 23

7.6 Write Flash Sector .......................................................................................................................... 23

7.7 Check State .................................................................................................................................. 23

7.8 Read Signature ............................................................................................................................... 24

7.9 Lock bits ...................................................................................................................................... 24

7.9.1 RBAP ......................................................................................................................................... 25

7.9.2 EPX ......................................................................................................................................... 25

7.10 Set NVR ...................................................................................................................................... 25

7.11 Read NVR ................................................................................................................................... 26

7.12 Enable EooS Mode ......................................................................................................................... 26

silabs.com | Building a more connected world.
7.13 Disable EooS Mode ..........................................................................................................................26
7.14 Run CRC Check ...............................................................................................................................26
7.15 Reset Chip ........................................................................................................................................27

8 NVR STORAGE ..................................................................................................................................28
8.1 NVR considerations when developing a programmer ......................................................................28
   8.1.1 REV field ................................................................................................................................28
   8.1.2 CRC16 field ............................................................................................................................28
   8.1.3 Mandatory fields .....................................................................................................................28

9 PROGRAMMING FLOW .....................................................................................................................29

10 TROUBLE SHOOTER .........................................................................................................................38

11 PROGRAMMING EXTERNAL NVM MEMORY CONNECTED TO SPI1 ...........................................39

12 PROGRAMMING TIMING ...................................................................................................................40

13 REFERENCES ....................................................................................................................................42
Table of Figures

Figure 1, Memory map as seen from the internal MCU ................................................................. 3
Figure 2, Memory map seen from the Programming interface ....................................................... 3
Figure 3, Flash sectors .................................................................................................................. 4
Figure 4, Power Interface to Programming Equipment ................................................................. 7
Figure 5, SPI Interface to Programming Equipment ................................................................. 7
Figure 6, UART Interface to a RS232 Programming Equipment ...................................................... 8
Figure 7, UART Interface to a 3.3V UART Programming Equipment ........................................... 8
Figure 8, External components for USB interface ........................................................................ 9
Figure 9, Components needed for programming when using a crystal ........................................... 9
Figure 10, Setup when using external clock source ..................................................................... 10
Figure 11, Programming Equipment with built-in reference clock generator (UART) ..................... 11
Figure 12, Programming Equipment with built-in reference clock generator (SPI) ......................... 11
Figure 13, Programming Equipment with Silicon Labs Calibration Box (UART) ......................... 11
Figure 14, Programming Equipment with Silicon Labs Calibration Box (SPI) ............................... 11
Figure 15, SPI protocol in Flash programming mode .................................................................... 16
Figure 16, Command timing ......................................................................................................... 16
Figure 17, SPI Four byte timing .................................................................................................... 17
Figure 18, Read Flash/Read SRAM command timing ................................................................... 17
Figure 19, Serial bits pattern format ............................................................................................ 17
Figure 20, UART Rx Tx flow .......................................................................................................... 17
Figure 21 CRC32 calculation code example in Python ..................................................................... 27
Figure 22, Flow chart box description ........................................................................................... 29
Figure 23, Main programming flow chart .................................................................................... 31
Figure 24, Enable and Sync Interface .......................................................................................... 32
Figure 25, Handle Error ............................................................................................................... 33
Figure 26, SPI/UART Program Flash Code Space ......................................................................... 33
Figure 27, USB Program Flash Code Space .................................................................................. 34
Figure 28, Program Flash Sector .................................................................................................. 35
Figure 29, CRC Check .................................................................................................................. 36
Figure 30, Program NVR ............................................................................................................. 37
Figure 31, Program Lock Bits ......................................................................................................... 37
Figure 32, SPI Flash Programming Control Signal Timing ............................................................. 40
Figure 33, SPI Flash Programming Data and Clock Timing ............................................................ 40

Table of Tables

Table 1, Operating modes .............................................................................................................. 5
Table 2, Wait periods ..................................................................................................................... 17
Table 3, Programming command set (recent changes are in red) ................................................... 21
Table 4, State Byte ....................................................................................................................... 24
Table 5, Signature Bytes ............................................................................................................. 24
Table 6, Lock bit bytes .................................................................................................................. 25
Table 7, RBAP byte ..................................................................................................................... 25
Table 8, SPI timing ....................................................................................................................... 41
Table 9, Reset/boot, programming command and power-up timing ............................................. 41
1 ABBREVIATIONS

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>APM</td>
<td>Auto Programming Mode</td>
</tr>
<tr>
<td>CRC</td>
<td>Cyclic Redundancy Check</td>
</tr>
<tr>
<td>EooS</td>
<td>Execute out of SRAM</td>
</tr>
<tr>
<td>EPx</td>
<td>Erase and Program Protection lock bits</td>
</tr>
<tr>
<td>FSM</td>
<td>Finite State Machine</td>
</tr>
<tr>
<td>IC</td>
<td>Integrated Circuit</td>
</tr>
<tr>
<td>IRAM</td>
<td>8051 Internal Random Access Memory</td>
</tr>
<tr>
<td>ISP</td>
<td>In-System Programming</td>
</tr>
<tr>
<td>JEDEC</td>
<td>Joint Electron Devices Engineering Council</td>
</tr>
<tr>
<td>MCU</td>
<td>Micro Controller Unit</td>
</tr>
<tr>
<td>MISO</td>
<td>Master In Slave Out</td>
</tr>
<tr>
<td>MOSI</td>
<td>Master Out Slave In</td>
</tr>
<tr>
<td>MSB</td>
<td>Most Significant Bit</td>
</tr>
<tr>
<td>MSL</td>
<td>Least Significant Bit</td>
</tr>
<tr>
<td>MTP</td>
<td>Many Time Programmable</td>
</tr>
<tr>
<td>NVR</td>
<td>Non Volatile Register</td>
</tr>
<tr>
<td>PVT</td>
<td>Product Verification Test</td>
</tr>
<tr>
<td>QFN</td>
<td>Quad Flat Non-lead</td>
</tr>
<tr>
<td>RBAP</td>
<td>Read Back Auto Programming mode lock bits</td>
</tr>
<tr>
<td>SCK</td>
<td>Serial Clock</td>
</tr>
<tr>
<td>SPI</td>
<td>Serial Peripheral Interface</td>
</tr>
<tr>
<td>SPIRE</td>
<td>SPI read Enable</td>
</tr>
<tr>
<td>TBD</td>
<td>To Be Defined</td>
</tr>
<tr>
<td>UART</td>
<td>Universal Asynchronous Receiver/Transmitter</td>
</tr>
<tr>
<td>USB</td>
<td>Universal Serial Bus</td>
</tr>
<tr>
<td>XOSC</td>
<td>System clock oscillator</td>
</tr>
<tr>
<td>XRAM</td>
<td>8051 External Random Access Memory</td>
</tr>
</tbody>
</table>

2 INTRODUCTION

2.1 Purpose

The purpose of this document is to enable readers to build Hardware and Software for a 500 Series Z-Wave Chip Flash memory programmer, development environment, and/or production test (Execute out of SRAM) environment.

Please refer to [1], [2], and [7], for a detailed description of the 500 Series Z-Wave Single Modules and Chips.

2.2 Audience and prerequisites

This document is targeted engineers that are well acquainted with SPI and with ISP programming in general.
3 OVERVIEW

3.1 Programming a 500 Series Z-Wave Chip

The code area and NVR (Non Volatile Register) areas of a 500 series Z-Wave chip must be programming before the chip can execute a Z-Wave program. An exception is some 500 Series Z-Wave Chips variants which has been pre-programmed with a SerialAPI image.

The NVR area contains fields with calibration data. This calibration data for a certain chip is found by calibration processes. These calibration processes must be performed at least once for each chip during the production/development stage and the calibration data MUST be written to the NVR for proper performance of the chip. Refer to [4] and [5] for descriptions of the NVR fields and for calibration procedures.

3.2 Programming Interfaces

The 500 Series Z-Wave Chip can be programmed through three interfaces: SPI, UART and USB. Refer to the datasheet of a The 500 Series Z-Wave Chip/Module for information of the availability of programming interface(s) of that specific 500 Series Z-Wave device.

The SPI and UART interfaces can be used when the programming unit has control over the RESET_N pin of the 500 Series Z-Wave Chip and it can be used even though the Code memory hasn't been programmed before.

The USB, SPI and UART interfaces can be used even though the programming unit hasn't control over the RESET_N pin of the 500 Series Z-Wave Chip. But it requires that the Chip has been programmed with code that makes it possible to enable Auto Programming Mode (See section 5).

3.3 Signature

The 500 Series Z-Wave Chip has a unique JEDEC signature (7F7F7F7F1F04xxh) that can be read through the SPI interface.

3.4 Memories

The 500 Series Z-Wave Chip has several memory instances, Flash and SRAM's. Figure 1 depicts how the instances are mapped in memory as seen from the internal MCU. Figure 2 depicts the memory mapping of instances, which can be accessed through a programming interface.
### Figure 1, Memory map as seen from the internal MCU

### Figure 2, Memory map seen from the Programming interface
3.4.1 Flash Memory

The Flash memory in the 500 Series Z-Wave Chip contains three areas:

- A 128kbyte code memory. This is used as code space for the 8051 MCU.
- A 255 byte data memory (Also referenced as MTP). This is used as non-volatile data space for the 8051 MCU. This memory is not memory mapped as it is addressed through registers.
- A NVR (Non Volatile Register) memory. This area is used for lock bit, calibration data. This memory is not memory mapped as it is addressed through registers. Refer to [4] for information of the mandatory NVR contents.

The 128kB code space is divided into 64 sectors as seen in Figure 3

![Figure 3, Flash sectors](image)

3.4.2 SRAM's

The 500 series Z-Wave Signal Chip has one SRAM block that can be accessed from the Programming interface.

The lower 2kB of the 4kB XRAM is used as buffer for programming the Flash and the whole 4kB XRAM is used as code memory in the “Execute out of SRAM” mode.

Data can be written to and read from the 4kB and 12kB XRAM's. This can be used for e.g. debugging purposes.

The IRAM and the Critical Memory SRAM is not accessible through the SPI interface.
3.5 Lock Bits

The function of the Lock Bits is to permanently disable the possibility to:

- Read-back the (entire) Flash data through a programming interface and
- Execute code from the SRAM.
- Erase single sectors

Setting either of these Lock bits will not prevent the internal MCU from accessing the Flash memory. Refer to section 7.9 for a further description of the lock bits.

3.6 CRC checksum

The upper four bytes located at the addresses 1FFFCh-1FFFFh are used to store a CRC-32 checksum of the Flash memory contents. An on-chip CRC-32 generator can be used to check the stored checksum value with a calculated value to verify the programmed Flash contents.

3.7 Modes

Table 1 describes the modes used in this document.

<table>
<thead>
<tr>
<th>Mode name</th>
<th>Description</th>
<th>Enabled by</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal mode</td>
<td>The normal functional chip mode</td>
<td>Power-n reset or after a 'Reset Chip' Command</td>
</tr>
<tr>
<td>Programming mode</td>
<td>The mode where the 4 kbyte and 12kB SRAM's, the lockbits, and the Flash can be accessed through the SPI or UART interface</td>
<td>Asserting the RESET_N pin for 5.1 ms</td>
</tr>
<tr>
<td>Auto Programming mode</td>
<td>The mode where the 4 kbyte and 12kB SRAM's, the lockbits, and the Flash can be accessed through the SPI, UART and USB interfaces</td>
<td>Instructing MCU to set the AUTOPROG bit or clearing lock bit</td>
</tr>
<tr>
<td>Execute out of SRAM mode (EooS)</td>
<td>In this mode the MCU can start-up and execute code from the 4 kbyte SRAM. This mode is intended for production test and PVT tests.</td>
<td>By setting a bit in programming Mode</td>
</tr>
</tbody>
</table>
4 HARDWARE INTERFACES

This section describes the hardware interfaces used during programming of the Flash and when running "Execute out of SRAM" mode.

The unit that controls the 500 series Z-Wave Chip in Programming mode is from now on called the "programming unit".

A 500 Series Z-Wave Chip/Module can be programmed through one or more of these interfaces:

- SPI
- UART
- USB

The state where either the SPI interface or the UART interface can be selected as programming interface can be entered by holding RESET_N asserted for 5.1ms and then applying an "Interface Enable" string on one of the interfaces.

A state, called Auto Programming Mode (APM), can be entered by first setting a register bit, AUTOPROG, in an internal SFR register or by clearing a lock bit. Then any reset pulse (internal or external generated) will boot the chip to state where any of the programming interfaces, SPI, UART or USB can be used as programming interface without having to assert the RESET_N pin. Except for the option to use the USB as programming interface the state is the same as after asserting RESET_N for 5.1ms.

The SFR register bit, AUTOPROG, is cleared by power-cycling or by a programming interface command. The mentioned lock bit can only be set by erasing the entire flash memory ('Erase Chip')

**Note:** keep the length of the wires between the programming unit and the 500 Series Z-Wave Chip/Module as short as possible. Especially the SPI signals are sensitive; hence the SPI wires should not be longer than 15cm. Also be careful with connectors in the signal path. Limit the number of connectors and make sure there is good contact.

4.1 Power

The section describes how the 500 Series Z-Wave Chip has to be powered in a programming setup.

For Z-Wave modules, the power pins (DVDD and AVDD) have to be connected to a 3.3V supply. These power supplies are decoupled on the module and there is no need for external decoupling for these supplies. All the ground pins (GND) on the chip should be connected to the ground of the programming unit.

For the SD350x chips, all power pins (*VDD*) and ground pins (GND) must be connected to the power supplies and well decoupled. In addition, the VDDQ has to be decoupled with a 6.8nF capacitor. Refer to section “Supply System” in [1] and [7] for a detailed description of how to decouple these power supplies for that device.

It is good practice that the 3.3V supply and the communication signals to the 500 Series Z-Wave Chip the can be controlled, so that all the power is off and the communication signals are in a High-Z state when the chip is inserted or removed from a programming socket/bed of nails.
4.2 Using SPI as Programming Interface

A 500 Series Z-Wave Chip can have up to two SPI interfaces, SPI0 and SPI1. The SPI1 interface can be used as programming interface.

The programming unit must be implemented as a SPI master and it must take over the control of the SPI bus.

The SPI1 interface consists of two data signals (MOSI1 and MISO1) and a clock signal (SCK1). The reset signal (RESET_N) is used in the process of enabling the programming mode. Another method for entering Programming mode is described in section 5.2. The SPI interconnection and control pins are shown in Figure 5. From now on in the document the signals MOSI1, MISO1, and SCK1 will be referred to as MOSI, MISO, and SCK respectively. All other GPIO ports will be set as inputs with the internal pull-up enabled.

Make sure that the IO's of the chip are not supplied, if the power inputs (DVDD and AVDD) are not powered.
The programming timing is described in Figure 33. Refer to section 6.1 for a description of the SPI interface protocol.

### 4.3 Using UART as Programming Interface

![Figure 6. UART Interface to a RS232 Programming Equipment](image)

![Figure 7. UART Interface to a 3.3V UART Programming Equipment](image)

A 500 Series Z-Wave Chip can have up to two UART interfaces, UART0 and UART1. The UART0 interface can be used as programming interface when available.

**Note:** The UART0 can only be used as programming interface where it is located on the pins P2.0 and P2.1 on the specific chip/module.

A RS232 level converter must be connected to the 2 pins of the UART0 interface to make the chip able to communicate according to the RS232 standard, as depicted in Figure 6.

The UART0 interface consists of two data signals \( UART0\_RX \) and \( UART0\_TX \). The reset signal \( RESET\_N \) is used in the process of enabling the programming mode. Another method for entering Programming mode is described in section 5.2. The UART interconnection and control pins are shown in Figure 6 and Figure 7. From now on in the document the signals \( UART0\_TX \) and \( UART0\_RX \) will be referred to as \( TX \) and \( RX \) respectively. All other GPIO ports will be set as inputs with the internal pull-up enabled. Figure 6 gives a simplified block diagram of the interface to the programming unit.

The 430Ω series resistor in the UART Rx signal to the chip is only required if the chip/module is to be programmed using APM mode. Please refer to section 5.2 for a description of APM mode.

Refer to section 6.2 for a description of the UART interface protocol.

Make sure that the IO's of the chip are not supplied, if the power inputs (DVDD and AVDD) are not powered.
4.4 Using USB as Programming Interface

The 500 Series Z-Wave Chip has an USB interface which can be used as a programming interface.

The USB interface consists of two data signals \( \text{USB\_DP} \) and \( \text{USB\_DM} \). The method for entering Programming mode is described in section 5.2. Figure 8 gives a simplified block diagram of the interface to the programming unit. The series resistors should be 23ohm and the pull-up resistor should be 1.5kOhm±0.075kOhm.

Refer to section 6.3 for a description of the USB interface protocol.

**Note:** The XRAM areas 0E00h to 0FFFh and 500h to 0507h are being used by the internal USB circuit when the USB is selected as programming interface. That is, you cannot rely on the contents of these areas when the USB is selected as programming interface.

4.5 System Clock

The 500 Series Z-Wave Chip must have a system clock (XOSC) running during programming. This system clock can either be applied as an external clock signal (see Figure 10) or by the means of an external crystal. If the crystal solution is used the following components must be connected (as depicted in Figure 9):

A. A crystal \( X_1 \) (32MHz)
B. Two load capacitors \( C_1 \) and \( C_2 \)

The value of the capacitors depend on the crystal type, refer to the datasheet of the crystal.
The external clock signal must run at 32MHz and must be a square wave 0V→1.2V. The external clock signal must be connected to XOSC_Q1 and XOSC_Q2 must be left unconnected as depicted in Figure 6.

![Diagram of 500-Series Z-Wave Single Chip](image)

**Figure 10. Setup when using external clock source**

When using a Z-Wave Module (ZM51xx) the X-TAL is part of the internal components, hence no external X-TAL or clock signal is needed when programming this module.

### 4.6 Required Calibration

The following calibration operations MUST have been performed on a final Z-Wave product that is based on the 500 series Z-Wave chip:

- Xtal calibration – to eliminate initial tolerance of the Xtal and load caps and
- TX calibration – to eliminate TX symbol separation variation

The calibration values MUST be stored in the non-volatile NVR area. A module that is based on a 500 Series Chip can already have one or both of the calibration values stored in the NVR when shipped out by Silicon Labs. Refer to [4] a description of the NVR contents, including the information on what is contained in the NVR of the different chips/modules when shipped out of Silicon Labs.

Silicon Labs provides a code image hex file, which when programmed into the 500 Series chip and being executed, will perform the calibration calculations and store the result in the XRAM in the chip. The Xtal calibration operation requires an external precision reference clock running at 10MHz/256~39.06kHz. Silicon Labs provided a calibration box that generates this reference clock. If you are going to manufacturer a production grade programmer including the calibration reference clock circuit, contact your Silicon Labs representative to get the required information on how to build this circuit.

The option to perform calibration is only required when using the UART and SPI interfaces. The calibration operation is expected to have been performed on a device that can be programmed through the USB interface.

The reference clock signal must either be connected to the SPI1 MISO pin or the UART0 TX pin on the 500 Series Chip. Since these signals are used by the programming too, the programmer MUST be able to tri-state the reference clock signal. The calibration box provided by Silicon Labs has an input signal that is used to tri-state the reference clock output.
A programmer that can is capable of doing calibration must execute the following sequence:

1. Read NVR and store contents in programmer
2. If the values of the calibration fields in the NVR are valid and the CRC16 checksum is correct
   goto step 6.
3. Enter programming mode, erase the chip and program the calibration hex image to the 500 Series device
4. Execute the calibration code in the chip.
5. Enter programming mode and read the calibration values from the XRAM
6. Program the final application image to the chip including the optionally modified NVR contents.
7. Exit programming mode

Refer to [5] for a detailed description of the 500 Series Silicon Labs calibration box and of how to perform the calibration operations.
5 ENTERING PROGRAMMING MODE

When using SPI or UART as programming interface, Programming Mode can be entered by holding RESET_N asserted for 5.1ms and then applying an “Enable Interface” string on one of the interfaces. Refer to sections 5.1 for further information about how to enter programming mode by asserting the RESET_N pin.

When using SPI, UART or USB as programming interface, Programming Mode can be entered by first entering Auto Programming Mode. Refer to section 5.2 for a further description of Auto Programming Mode.

Once programming has been enabled using a certain interface the chip must be power-cycled or RESET_N has be asserted before another interface can be used as programming interface.

5.1 Enabling Programming Mode

The following procedure has to be performed to enable the programming mode:

1. Power cycle the Chip
2. Set RESET_N low for more than \( t_{PE} = 5.1 \text{ms} \). This will set the Chip in a state where it listens to data on the SPI1 and UART0 interfaces, but doesn’t drive the MISO nor UART_TX signals.
3. Send an “Enable Interface” string to the Chip through the SPI1 or the UART0 interface. The MISO IO will be set as output and driven after the first two bytes of the “Enable Interface” string have been received by the Chip on the SPI1 interface (MOSI). Likewise, the UART_TX IO will be set as output and driven after the first two bytes of the “Enable Interface” string have been received by the Chip on the UART0 interface (UART_RX).

The ‘Enable Interface’ command is used to enable the programming interface for communication and to do byte-synchronization. The 500 Series Z-Wave Chip must be powered-on and it must have a stable system clock running, i.e. wait for \( t_{power-up} \) (see Table 9) after power is applied. See section 9 for a full programming sequence description.

Figure 24 shows the procedure to enable programming mode and to synchronize. The synchronization has taken place when the 500 Series Z-Wave Chip echoes 53h and AAh when the 3rd and 4th byte is transferred respectively. For SPI the synchronization normally takes 1 attempt, but may take up to 31 attempts.

5.2 Auto Programming Mode

Another method for entering Programming Mode is to use Auto Programming Mode (APM). The chip MUST have been programmed (through the SPI or the UART interface) with firmware that enables the ability to communicate with the internal MCU and instruct it to call a specific API command which sets a register bit, AUTOPROG. If the AUTOPROG bit has been set, then any reset pulse (internal or external) will bring the chip in a state where the chip’s USB product ID (0280h) will report the device as a “Silicon Labs ZWave Programming Interface”. Otherwise the chip will be same state as the state after asserting RESET_N for 5.1ms.

SPI, UART or USB can be used as programming interface, when APM has been enabled, the chip has been reset and has rebooted. All interfaces will then listen for the “Enable Interface” string. Once this

---

1 This is the same procedure used to enter programming mode for the 3- and 400 Series Z-Wave Chip.
string has been received on an interface, this interface will exclusively be selected as the programming interface.

**Note:** Before using the USB as a programming interface, it is required that the USB host programmer software can instruct the chip to go into APM state and therefore be able to "find" the device and identify it as a "Silicon Labs ZWave Programming Interface".

The USB host can instruct the chip to go to APM mode by having the internal MCU calling an API call (ZW_FLASH_auto_prog_set()). The communication to the MCU can be either through a native driver (known to the programmer software) or through the Z-Wave SerialAPI.

The Z-Wave application developers can also implement other methods for enabling APM. This could be like having the user to press a button, signal on a certain pin from a host CPU, etc.

If the 500 Series Z-Wave chip has been pre-programmed with the Z-Wave SerialAPI a SerialAPI command can be sent to the chip to enable APM mode. The SerialAPI uses either the USB interface or UART0 interface as communication interface. The "EnableAPM" command consists of six bytes:

```
01h, 03h, 00h, 27h, DBh
```

where 01h is send first, then 03h, etc.

The device will acknowledge the command by sending

```
06h
```

Also refer to [8] for a description of the SerialAPI command.

The Auto Programming Mode (APM) is added to the 500 Series chip to be able to enter programming mode without any manual interaction or by having any other electrical connectivity to the chip than the wires needed for the SPI, UART, or USB interfaces.

When the APM has been enabled and the chip is starting after a reset cycle, a boot code will be executed by the MCU. This boot code will configure the USB, so that the USB device type (PID) becomes a "Silicon Labs ZWave Programming Interface".
5.2.1 Enable/disable APM mode

There are two sets approaches to enable/disable APM mode:

- **SFR bit: AUTOPROG**
  - APM mode can be enabled by setting this bit. The bit is set by the means of firmware running in the device.
  - When calling the API call `ZW_FLASH_auto_prog_set()` this bit is set.
  - If APM mode was enabled by setting the AUTOPROG SFR bit then:
    - APM mode can be disabled by executing the "Reset Chip" programming command. Refer to section 7.15.
    - APM mode can also be disabled by power-cycling the device.
    - APM mode can NOT be disabled by asserting RESET_N pin the device.

- **Lock bits: AutoProg0 and AutoProg1**
  - APM mode can be enabled by clearing the lock bit AutoProg0 through a programming interface.
  - If APM mode was enabled by clearing AutoProg0 then:
    - APM mode is disabled by clearing the lock bit AutoProg1 through a programming interface.
    - APM mode can NOT be disabled by power-cycling the device, asserting RESET_N pin the device or by executing the "Reset Chip" programming command.

If APM mode was enabled by setting the AUTOPROG SFR bit AND at the same time the lock bit AutoProg0 is cleared and the lock bit AutoProg1 is set, then APM mode is disabled by first clearing the lock bit AutoProg1 through a programming interface and then either executing the "Reset Chip" programming command or power-cycling the device.

APM mode is disabled by first clearing the lock bit AutoProg1 through a programming interface and then executing the "Reset Chip" programming command. Refer to section 7.15.

Refer to the description of the programming flow (section 9) of where the SFR and the Lock bits are used to enable/disable APM mode in a programming operation.

Note: If the 500 Series Chip is in APM mode, the RESET_N signal must be set high to be able to program the flash. The "FlashFSM busy" flag is constant set, if APM is enabled and RESET_N is set low. In order to detect whether APM is enabled or not when you enter programming mode by asserting RESET_N, check the flash state right after having enabled the interface (and optionally read the signature) If it turns out that APM is enabled, set RESET_N high and re-start the process by re-enabling the interface.

Note: In the case where the Auto Programming mode SFR register bit is set and the EooS bit is set, the EooS bit will take precedence and the chip will enter EooS mode after a reset. Refer

Note: In the case where the APM lock bits are set to APM enabled state and the EooS bit is set, the EooS bit will take precedence and the chip will enter EooS mode after a reset.

Note: When using APM mode for programming a device through the UART interface a series resistor is required on the UAT Rx signal to the device. Please refer to section 4.3 for a description of this resistor.

5.2.2 Programming SD3503/ZM5304 in production

The SD3503/ZM5304 only supports the USB or the UART interface as programming interface. Only the UART programming interface can be used for programming the SD3503 and the ZM5304 in production. This is because you have to first put the chip/module in APM mode to be able to use the USB as programming interface and blank devices can't be put in APM mode.
So for SD3503 applications where the chip is programmed in the application PCB, it is required that the UART and the RESET_N signals can be reached through test points or a connector, etc.
6 PROTOCOLS

The following sections describe the protocols used for the three programming interfaces, SPI, UART and USB.

6.1 SPI Protocol

The SPI interface must run in “mode 0” as defined in the SPI standard. That is, in programming mode the SPI clock SCK must be low in the idle state, the SPI data (MISO and MOSI) is read on the rising edge of SCK, and the SPI data is clocked out on the falling edge of SCK, as depicted in Figure 15.

![Figure 15: SPI protocol in Flash programming mode](image)

Table 3 shows the programming command formats. Each command consists of four bytes. The command bytes are sent in the order byte 0 to byte 3 with most significant bit (MSB) first. All 4 bytes must be sent.

The period from the rising edge of the SCK signal of the last bit in a byte (LSB) to the first bit of the following byte (MSB) must be a minimum of 5 system clock periods, as depicted in Figure 16. When reading data through the programming interface of the 500 Series Z-Wave Chip this period is longer at some points, as described in Table 2.

![Figure 16. Command timing](image)

**Note:** that a Flash memory location contains FFh before it is programmed, whereas SRAM locations can contain any value when entering programming mode.

When communicating with the chip through the SPI interface the bytes send to the chip will normally be echoed on the following byte transfer, as depicted in Figure 17. When data is returned the received data will be exchanged with the returned data, e.g. byte 3 for the “Read Flash” command.

---

Table 3: Programming command formats

<table>
<thead>
<tr>
<th>Command byte</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x00</td>
</tr>
<tr>
<td>1</td>
<td>0x01</td>
</tr>
<tr>
<td>2</td>
<td>0x02</td>
</tr>
<tr>
<td>3</td>
<td>0x03</td>
</tr>
</tbody>
</table>

---

2 The system clock is the clock derived from the X-tal freq: 32MHz
When a read operation is performed ('Read Flash', 'Read SRAM', 'Continue Read') it will take up to 45 system clock cycles before valid read data are available on the SPI data output. The double-border, ||, in Table 3 shows where the programming unit (SPI master) must wait, before sending the first positive edge on SCK after sending the last negative edge of SCK for the previous byte. Figure 18 depicts the inter-byte delays. Table 2 shows the wait period for the different command and the location of the wait period.

![Figure 18. Read Flash/Read SRAM command timing](image)

Table 2, Wait periods

<table>
<thead>
<tr>
<th>Command</th>
<th>Space location</th>
<th>System clock periods</th>
</tr>
</thead>
<tbody>
<tr>
<td>'Read Flash'</td>
<td>Between byte 2 and 3</td>
<td>45</td>
</tr>
<tr>
<td>'Read SRAM'</td>
<td>Between byte 2 and 3</td>
<td>8</td>
</tr>
<tr>
<td>'Continue Read'</td>
<td>Between byte 0 and byte 1</td>
<td>45 (after 'Read Flash')</td>
</tr>
<tr>
<td>Other</td>
<td>Between all bytes</td>
<td>5</td>
</tr>
</tbody>
</table>

6.2 UART protocol

The UART will be configured as 115.2kbit/s, 1 start bit, 8bit data, no parity and 2 stop bits. Data is transmitted with least significant bit (LSB) first, as depicted in Figure 19. No handshake.

![Figure 19. Serial bits pattern format](image)

When communicating with the chip through the UART interface the bytes send to the chip will normally be echoed a few system clock periods after the incoming data has been received, as depicted in Figure 20. When data is returned the chip will exchange the received data with the data to be returned to the programming unit, e.g. byte 3 for the “Read Flash” command.

![Figure 20. UART Rx Tx flow](image)

Note: One of the two transmitted stop bits can occasionally be omitted by the 500 Series Z-Wave Chip in scenarios where the speed of the programmers UART data is higher than the one of the 500 Series Z-Wave Chip. So the programmer MUST ignore framing errors.
6.3 USB protocol

After booting the chip when Auto Programming Mode has been enabled, the USB interface will be configured as a "Silicon Labs ZWave Programming Interface" device with Vendor ID 0658h and Device ID 0280h. The device contains one control endpoint for USB enumeration and two bulk endpoints for programming commands, one IN endpoint and one OUT endpoint. The programming commands can be seen in section 7. The commands consist of 4 bytes the host must be sent to the bulk OUT interface, and then the chip will respond with 4 bytes that are read from the bulk IN endpoint. The USB programming interface operates like a stream interface, so that it is possible to submit the bytes one at a time, or queue up several commands before reading the responses. Internal IN and OUT stream buffers are 255 bytes each, so no more than 63 commands can be buffered before the results must be read. Due to internal limitations no more than 3 commands (12 bytes) must be sent in one USB command.

The USB interface is USB ver 2.0 running 12Mbit/s and will act as a virtual COM port device in APM mode. It requires a proprietary Sigma driver (Available for Linux and Windows XP/7/8).

A full programming flow is shown in Figure 23. A part of this flow is the process of programming the Flash code space area. This "USB Program Flash Code Space" process must be handled in a special manner as shown in Figure 27. The process is divided into three steps:

1. Program usb_loader firmware into sector 0 (In two parts as shown below)
2. Program sectors 1-63 with target firmware
3. Program sector 0 with target firmware

The hardware ID for the USB interface is 0 for the hardware based programming mode, and 1 when in the programming mode supplied by the usb_loader firmware. The hardware ID should therefore be checked, and it should be 1 in step 2, and 0 in step 1 and 3. (See section 6.3.1)

The flow chart of these steps combined is shown in Figure 27. The Z-Wave chip USB interface will operate as a "Silicon Labs ZWave Programming Interface" device in all three steps, but it will request re-enumeration before step 2 and step 3.

In step 1 and 3 XRAM area 500h to 0507h are being used by the internal USB circuit, so this area MUST be FFh's in both the usb_loader firmware and in sector 0 of the target firmware.

Step 1:

In the first step, the chip must be set in APM mode by either calling the Z-Wave SerialAPI function, (ZW_FLASH_auto_prog_set()), tie a pin low or other (depending on the specific functionality of the firmware currently programmed into the chip to be reprogrammed). As mentioned above it must be checked that the USB hardware ID (HWID) is 0. If it is not 0 this step should be restarted, resending the ZW_FLASH_auto_prog_set() command.

The usb_loader firmware is divided into two parts (usb_loader_part1.hex and usb_loader_part2.hex). First, the file usb_loader_part1.hex must be programmed into the chip and then the content MUST be verified. Secondly, the file usb_loader_part2.hex MUST be programmed into the chip WITHOUT erasing anything. The chip must then be reset and start in normal operation mode where the usb_loader will provide a new programming mode that will configure the USB device as a Z-Wave programming USB Device again.

Step 2:

When this "Silicon Labs ZWave Programming Interface" device has been re-enumerated, it must be checked that the USB hardware ID is 1 (otherwise goto step 1). Now the sectors 1-63 of the target firmware can be programmed to the Flash code space. In this state where the usb_loader firmware implements the USB device function only the following programming commands are available:
- Read signature
- Write SRAM
- Continue write
- Write flash sector
- Check state (always returns idle)

See section 7 for a description of the programming commands.

The `usb_loader` firmware parses the Z-Wave SerialAPI call, `ZW_FLASH_auto_prog_set()`, which will set the SFR APM register bit and reset the chip where after the chip will be re-enumerrated again and reappear as a Z-Wave programming USB Device. This SerialAPI command must be called in the end of step 2. Refer to section 5.2 for the description of how to enable APM.

Step 3:

The USB hardware ID must be checked again, and it should be 0 now. Now the chip's programming interface is again handled by the programming hardware circuit, where all programming commands are available. In this step first the lock bit AutoProg0 must be cleared and then sector 0 must be erased. Sector 0 of the target firmware is then programmed in to the flash code space in two separate operations, first programming the data belonging to the addresses from 00000h to 004FFh and then the data belonging to the addresses from 00508h to 007FFh. After this the normal programming flow is followed in Figure 23.

6.3.1 USB Hardware ID (HWID)

During the USB programming interface two different modes are used (here called HWID). The two modes can be distinguished using `bcdDevice` field in the USB device descriptor. It can be read in the Windows registry and in Linux `/sys` file structure. The value is either 0 (0000h) or 1 (0001h).
7 COMMAND SET

Table 3 gives an overview of all the 4-byte programming commands. Be aware of the different ways of returning data (TX\textsubscript{X} rows) for the different interfaces. The rows TX\textsubscript{U} shows the data returned from USB and UART interfaces and the rows TX\textsubscript{S} shows the data returned from the SPI interface, where \textit{prev} means the last byte transferred to the device in the previous command.
### Table 3, Programming command set

<table>
<thead>
<tr>
<th>Command</th>
<th>Byte 0</th>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable Interface</td>
<td>RX: ACh&lt;sup&gt;1&lt;/sup&gt;</td>
<td>TX&lt;sub&gt;U&lt;/sub&gt;: 53h&lt;sup&gt;1&lt;/sup&gt;</td>
<td>TX&lt;sub&gt;S&lt;/sub&gt;: AAh</td>
<td>TX&lt;sub&gt;S&lt;/sub&gt;: 55h</td>
<td>Enable serial programming after reset has been low for 5s, see section 5.</td>
</tr>
<tr>
<td>Read Flash</td>
<td>RX: 10h</td>
<td>TX&lt;sub&gt;U&lt;/sub&gt;: 0Fh&lt;sup&gt;1&lt;/sup&gt;</td>
<td>TX&lt;sub&gt;S&lt;/sub&gt;: 0Fh&lt;sup&gt;1&lt;/sup&gt;</td>
<td>TX&lt;sub&gt;S&lt;/sub&gt;: FFh</td>
<td>Read at first byte of Flash sector number Sector. Valid range of Sector is 00h-3Fh</td>
</tr>
<tr>
<td>Read SRAM</td>
<td>RX: 06h</td>
<td>TX&lt;sub&gt;U&lt;/sub&gt;: Addr1</td>
<td>TX&lt;sub&gt;S&lt;/sub&gt;: Addr0</td>
<td>TX&lt;sub&gt;S&lt;/sub&gt;: FFh</td>
<td>Read data at SRAM address Addr1:Addr0. Valid range of Addr1:Addr0 is 0000h-3FFFh</td>
</tr>
<tr>
<td>Continue Read</td>
<td>RX: A0h</td>
<td>TX&lt;sub&gt;U&lt;/sub&gt;: Data0</td>
<td>TX&lt;sub&gt;S&lt;/sub&gt;: Data1</td>
<td>TX&lt;sub&gt;S&lt;/sub&gt;: Data2</td>
<td>This command can follow a &quot;Read SRAM&quot; or a &quot;Read Flash&quot; command. This command will read the next three memory locations from the SRAM or Flash, depending on the preceding command.</td>
</tr>
<tr>
<td>Write SRAM</td>
<td>RX: 04h</td>
<td>TX&lt;sub&gt;U&lt;/sub&gt;: Addr1</td>
<td>TX&lt;sub&gt;S&lt;/sub&gt;: Addr0</td>
<td>TX&lt;sub&gt;S&lt;/sub&gt;: Data</td>
<td>Write data Data to SRAM page buffer at address Addr1:Addr0. Valid range of Addr1:Addr0 is 0000h-3FFFh</td>
</tr>
<tr>
<td>Continue Write</td>
<td>RX: 80h</td>
<td>TX&lt;sub&gt;U&lt;/sub&gt;: Data0</td>
<td>TX&lt;sub&gt;S&lt;/sub&gt;: Data1</td>
<td>TX&lt;sub&gt;S&lt;/sub&gt;: Data2</td>
<td>The command can follow a &quot;Write SRAM&quot; command and it writes three bytes, Data0, Data1, Data2 to the next memory locations in SRAM.</td>
</tr>
<tr>
<td>Erase Chip</td>
<td>RX: 0Ah</td>
<td>TX&lt;sub&gt;U&lt;/sub&gt;: 00h</td>
<td>TX&lt;sub&gt;S&lt;/sub&gt;: 00h</td>
<td>TX&lt;sub&gt;S&lt;/sub&gt;: 00h</td>
<td>This will erase the code space and the NVR space including lock bits</td>
</tr>
<tr>
<td>Erase Sector</td>
<td>RX: 08h</td>
<td>TX&lt;sub&gt;U&lt;/sub&gt;: Sector</td>
<td>TX&lt;sub&gt;S&lt;/sub&gt;: FFh</td>
<td>TX&lt;sub&gt;S&lt;/sub&gt;: FFh</td>
<td>This will erase the code sector number Sector. Valid range of Sector is 00h-3Fh</td>
</tr>
<tr>
<td>Write Flash Sector</td>
<td>RX: 20h</td>
<td>TX&lt;sub&gt;U&lt;/sub&gt;: Sector</td>
<td>TX&lt;sub&gt;S&lt;/sub&gt;: FFh</td>
<td>TX&lt;sub&gt;S&lt;/sub&gt;: FFh</td>
<td>Write previously loaded SRAM data to the Flash Memory. The Flash start address will be the first byte of the sector set by the Sector value. Valid range of Sector is 00h-3Fh</td>
</tr>
<tr>
<td>Check State</td>
<td>RX: 7Fh</td>
<td>TX&lt;sub&gt;U&lt;/sub&gt;: FEh</td>
<td>TX&lt;sub&gt;S&lt;/sub&gt;: 00h</td>
<td>TX&lt;sub&gt;S&lt;/sub&gt;: 00h</td>
<td>Read the status of the programming logic</td>
</tr>
<tr>
<td>Read Signature</td>
<td>RX: 30h</td>
<td>TX&lt;sub&gt;U&lt;/sub&gt;: Num</td>
<td>TX&lt;sub&gt;S&lt;/sub&gt;: FFh</td>
<td>TX&lt;sub&gt;S&lt;/sub&gt;: FFh</td>
<td>Read signature byte number Num</td>
</tr>
<tr>
<td>Disable EooS Mode</td>
<td>RX: 0Fh</td>
<td>TX&lt;sub&gt;U&lt;/sub&gt;: FFh</td>
<td>TX&lt;sub&gt;S&lt;/sub&gt;: FFh</td>
<td>TX&lt;sub&gt;S&lt;/sub&gt;: FFh</td>
<td>Disable EooS mode. The chip will start in normal mode when reset is released</td>
</tr>
<tr>
<td>Enable EooS Mode</td>
<td>RX: COh</td>
<td>TX&lt;sub&gt;U&lt;/sub&gt;: 00h</td>
<td>TX&lt;sub&gt;S&lt;/sub&gt;: 00h</td>
<td>TX&lt;sub&gt;S&lt;/sub&gt;: 00h</td>
<td>Running this command will set the mode in EooS Mode after reset is released</td>
</tr>
<tr>
<td>Set Lock Bits</td>
<td>RX: F0h</td>
<td>TX&lt;sub&gt;U&lt;/sub&gt;: Num</td>
<td>TX&lt;sub&gt;S&lt;/sub&gt;: FFh</td>
<td>TX&lt;sub&gt;S&lt;/sub&gt;: FFh</td>
<td>See section 7.9 for a description of the LockData contents</td>
</tr>
<tr>
<td>Read Lock Bits</td>
<td>RX: F1h</td>
<td>TX&lt;sub&gt;U&lt;/sub&gt;: Num</td>
<td>TX&lt;sub&gt;S&lt;/sub&gt;: FFh</td>
<td>TX&lt;sub&gt;S&lt;/sub&gt;: FFh</td>
<td>Read the lock bits, LockData, refer to section 7.9</td>
</tr>
<tr>
<td>Set NVR</td>
<td>RX: FEh</td>
<td>TX&lt;sub&gt;U&lt;/sub&gt;: Addr</td>
<td>TX&lt;sub&gt;S&lt;/sub&gt;: Addr</td>
<td>TX&lt;sub&gt;S&lt;/sub&gt;: Data</td>
<td>Set a byte in NVR space. Valid range for Addr is 09h - FFh</td>
</tr>
<tr>
<td>Read NVR</td>
<td>RX: F2h</td>
<td>TX&lt;sub&gt;U&lt;/sub&gt;: Addr</td>
<td>TX&lt;sub&gt;S&lt;/sub&gt;: Addr</td>
<td>TX&lt;sub&gt;S&lt;/sub&gt;: Data</td>
<td>Read from NVR space. Valid range for Addr is 0FFh - FFh</td>
</tr>
<tr>
<td>Run CRC Check</td>
<td>RX: C3h</td>
<td>TX&lt;sub&gt;U&lt;/sub&gt;: 00h</td>
<td>TX&lt;sub&gt;S&lt;/sub&gt;: 00h</td>
<td>TX&lt;sub&gt;S&lt;/sub&gt;: 00h</td>
<td>Run the CRC check procedure. Used to verify that the correct data has been written to the Flash.</td>
</tr>
<tr>
<td>Reset Chip</td>
<td>RX: FFh</td>
<td>TX&lt;sub&gt;U&lt;/sub&gt;: FFh</td>
<td>TX&lt;sub&gt;S&lt;/sub&gt;: FFh</td>
<td>TX&lt;sub&gt;S&lt;/sub&gt;: FFh</td>
<td>If the Auto Prog mode register bit is set then the command clears the Auto Prog mode register bit and resets the chip.</td>
</tr>
</tbody>
</table>

1 SPI interface: The first time after RESET_N has been asserted the first two bytes will be read as FFh since the output of the Chip is disabled and the signal is pulled up.

UART interface: The first time after RESET_N has been asserted the first two bytes will be ignored.

2 These bytes might not be returned from the device in UART or USB programming mode.

Labeled: prev: Last byte of previous MOSI data transfer

h: Hexadecimal number

---

*silabs.com | Building a more connected world.*
7.1 Read Flash

This command reads a byte at a given address in Flash. Please refer to section 6 regarding the timing of the processing of the command.

The 'Read Flash' command can addresses the Flash at sector boundaries. That is the 'Read Flash' command reads the first byte of a certain sector (low address). Use the command 'Continue Read’ one or more times to read the following bytes.

Note: It is possible to use the command ‘Continue Read’ to read beyond the end of the Flash code space. The locations read beyond 1FFFFh will be return the value read at address 1FFFFh.

7.2 Read SRAM/Continue Read

This command reads a byte at a given address in SRAM. Please refer to section 6 regarding the timing of the processing of the command. The command ‘Continue Read’ can follow the ‘Read SRAM’ command to speed up consecutive reads from the SRAM.

7.3 Write SRAM/Continue Write

The lower 2kB of the SRAM is used as an intermediate buffer when programming the Flash. There are special considerations to take into account when using the SRAM as Flash programming buffer. So the rest of this section is parted in two parts. One that describes the write operations when the transferred data is to be written to SRAM only and one that describes the write operations when the transferred data is to be programming into the Flash

7.3.1 Write SRAM only

Data can be written to the 4kB and the 12kB XRAM memories in the chip. The IRAM and the 128B critical memory cannot be accessed through the programming interface.

Refer to Figure 2 for the details of the addressing of the 4kB and the 12kB XRAM memories.

This command is used to write a data byte to the SRAM. The Command ‘Continue Write’ can follow the 'Write SRAM' command to speed up consecutive writes to the SRAM. Do not write beyond the upper address 3FFFh. Instead, use the ‘Write SRAM’ command for the 1st byte and then continue using 'Continue Write’ until the last 3 byte portion that fits into the SRAM. Finally, use the ‘Write SRAM’ command for the final one or two bytes that remains.

7.3.2 Use SRAM as intermediate buffer for programming Flash

When programming the Flash, a range of data is written to the lower 2kB of the SRAM and thereafter programmed into the Flash. This procedure is repeated until all data has been programmed in to the Flash. Each of the ranges of data MUST be written to the SRAM first by using the 'Write SRAM Byte' then followed by ‘Continue Write’ commands until the SRAM is filled or until there is no more data to be written.

That is, using the ‘Write SRAM’ command will set the base address of the data range in SRAM and the data size to one. Whereas, each of the optionally following ‘Continue Write’ commands, will increase the data size by three.
When writing data to the SRAM for programming the Flash:

- Do not write code data outside the lower 2kB of the XRAM
- The last byte written to the SRAM buffer before executing the "Write Flash Sector" MUST be different from FFh

### 7.4 Erase Chip

The Erase chip command will erase the entire code section and NVR, which contains lock bits and Non-volatile data. A part of this Non-volatile data has been written by Silicon Labs. This data must be kept, so this data has first to be read and stored before the "Chip Erase" command is executed and then the NVR data must be rewritten to the NVR area.

The data area (MTP) of the Flash is not affected by this command.

**Note:** An "Erase Chip" command must always be followed by "Check State" command(s) to ensure that the chip is idle before new commands are issued.

### 7.5 Erase Sector

The ‘Erase Sector’ command can erase one code sector (2kB) at a time. No other areas will be affected by this command.

**Note:** An "Erase Sector" command must always be followed by "Check State" command(s) to ensure that the chip is idle before new commands are issued.

### 7.6 Write Flash Sector

This command instructs the programming logic to start the Flash programming procedure. The data that has been written to a range in the lower 2kB of the SRAM is read and then written to the Flash code memory byte by byte. See section 7.3 for a description on how the range is set. The sector which is part of the 'Write Flash Data' command sets the flash sector to be programmed.

**Note:** A "Write Flash Sector" command must always be followed by "Check State" command(s) to ensure that the chip is idle before new commands are issued.

Refer to section 8 for a full description of how to program the Flash.

### 7.7 Check State

The ‘Check State’ command returns a status byte as described in Table 4.
Table 4, State Byte

<table>
<thead>
<tr>
<th>Bit</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CRC busy</td>
<td>This bit is set when a ‘Run CRC Check’ command is issued and it will be cleared when the CRC check procedure is done</td>
</tr>
<tr>
<td>1</td>
<td>CRC done</td>
<td>This bit is cleared when a ‘Run CRC Check’ command is issued and it will be set if the CRC check procedure completes.</td>
</tr>
<tr>
<td>2</td>
<td>CRC failed</td>
<td>This bit is cleared when a ‘Run CRC Check’ command is issued and it will be set if the CRC check procedure fails.</td>
</tr>
<tr>
<td>3</td>
<td>FlashFSM busy</td>
<td>This bit is set when flash controller state machine is busy</td>
</tr>
<tr>
<td>4</td>
<td>-</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td>5</td>
<td>Cont operation refused</td>
<td>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’ command A ‘Continue Read Operation’ is not succeeding a ‘Read Flash’, a ‘Read SRAM’ or a ‘Continue Read’ command</td>
</tr>
<tr>
<td>6</td>
<td>-</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td>7</td>
<td>Exec SRAM mode enabled</td>
<td>This bit is set if the ‘Execute out of SRAM’ Mode has been enabled</td>
</tr>
</tbody>
</table>

7.8 Read Signature

The ‘Read Signature Byte’ command reads a signature byte from the chip. For the 500 Series Z-Wave Chip there is 7 signature bytes. So to read the entire signature the ‘Read Signature Byte’ has to be performed 7 times, each with the Num value incremented by one. The description of the 7 bytes is shown in Table 5.

Table 5, Signature Bytes

<table>
<thead>
<tr>
<th>Signature byte Number</th>
<th>Value (hex)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00h</td>
<td>7Fh</td>
<td>Manufacturer JEDEC ID.</td>
</tr>
<tr>
<td>01h</td>
<td>7Fh</td>
<td></td>
</tr>
<tr>
<td>02h</td>
<td>7Fh</td>
<td></td>
</tr>
<tr>
<td>03h</td>
<td>7Fh</td>
<td></td>
</tr>
<tr>
<td>04h</td>
<td>1Fh</td>
<td></td>
</tr>
<tr>
<td>05h</td>
<td>04h</td>
<td>Chip Type</td>
</tr>
<tr>
<td>06h</td>
<td>xx</td>
<td>Chip revision (current rev is 01h)</td>
</tr>
</tbody>
</table>

7.9 Lock bits

Nine bytes can be set by executing the ‘Write Lock Bits’ command. The bits in these bytes can be set one at the time or both at the same time, but once they are set they can’t be cleared. The state of the lock bits can always be read with the ‘Read Lock Bits’ command.
The lock bits can be read and set from a programming interface. Only an “Erase Chip” operation can erase the lock bits. The lock bits are used to:

- Enable read back-protection of code space
- Auto Programming Mode lock bits.
- Set erase-protection on part of code area. Only protects against erase operations initiated by the MCU. Contained in the bytes EP0-EP7

Table 6, Lock bit bytes

<table>
<thead>
<tr>
<th>Byte</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
</table>

Setting either of these Lock bits will not prevent the internal MCU from reading the Flash memory

Note: A "Set Lockbit" command must always be followed by "Check State” command(s) to ensure that the chip is idle before new commands are issued.

7.9.1 RBAP

The RBAP byte, as depicted in Table 7, contains the 2 lock bits for controlling the Non–volatile Auto Prog setting (bits 2 and 1), see section 5.2, for a description of how to use these bits.

The RBAP byte also contains the Read-Back protection lock bit (bit 0). When this bit is 0b, the chip will return 00h when reading from Flash code space. The CRC32 check function still works even if the chip is read-back protected.

Bits 7-3 are unused and MUST be kept high, as they are reserved for future use.

Table 7, RBAP byte

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field</td>
<td>1b</td>
<td>1b</td>
<td>1b</td>
<td>1b</td>
<td>1b</td>
<td>Auto Prog1</td>
<td>Auto Prog0</td>
<td>Readback Protection</td>
</tr>
</tbody>
</table>

7.9.2 EPx


When a bit is 0b, the associated sector is protected from being erases and programmed by the internal MCU.

The bits can only be set to 1b by executing the "Chip Erase” command.

7.10 Set NVR

248 bytes of the NVR area of the flash can be used to store Non-volatile data. The 'Set NVR' command is used to write to the NVR area. One byte can be programmed at a time. Use the 'Erase Chip' command to erase the NVR area. Refer to section 8.

Note: A “Set NVR” command must always be followed by "Check State” command(s) to ensure that the chip is idle before new commands are issued.
7.11 Read NVR

The 'Read NVR command' is used to read the NVR contents from the programming interface. One byte can be read at a time. Refer to section 8.

7.12 Enable EooS Mode

The Chip can be put into a mode where the 4 kbyte SRAM is mapped as the only code memory visible for the internal MCU. This mode is called “Execute out of SRAM Mode” (EooS)

The “Execute out of SRAM Mode” is enabled if the ‘Execute out of SRAM’ command is executed. “Execute out of SRAM Mode” will be disabled after a power-cycle or after having executed the ’ Disable EooS’ Command. A reset of the chip will leave the “Execute out of SRAM Mode” state unaffected. The state of the “Execute out of SRAM Mode” can be read with the ‘Check State’ command.

The reset signal (RESET_N) has to be released to start the MCU and thereby enabling the executing of code form the 4 kbyte SRAM.

Note: Only I RAM can be used as data area for the MCU in EooS Mode

7.13 Disable EooS Mode

This command can disable the "EooS mode". When this command has been called then release reset of chip to restart in normal mode.

7.14 Run CRC Check

Executing the ‘Run CRC Check’ will start a built-in CRC-32 generator in the chip that will read the entire Flash and calculate the CRC-32 checksum.

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

When doing the calculation the CRC-32 registers are initially set to FFFFFFFFh. The calculation is done on all bytes of the memory except for the upper 4 bytes. The CRC-32 generator is fed one byte at the time starting with address 00000h and feeding the generator with the most significant bit first.

The CRC-32 generator polynomial is:

\[ P(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1 \]

The ‘Run CRC-32 Check’ will work even if the Read-back lock bit is set.

Calculation examples:

An empty Flash (all FFh's) will give a CRC32 value of 249E4679h (where 24h is stored in address 1FFFCh, 9Eh is stored in address 1FFFDh, etc.)
A Flash where all bytes are 11h, will give a CRC value of 330F8363h (where 33h is stored in address 1FFFC, 0Fh is stored in address 1FFFFDh, etc.)

Figure 21 provides an example of how to calculate the CRC32 value.

```python
import intelhex
import struct
import binascii

def reverseBit(s):
    r=""
    for c in s:
        x = ord(c)
        x = ((x<<4) & 0xF0)|((x>>4) & 0x0F);
        x = ((x<<2) & 0xCC)|((x>>2) & 0x33);
        x = ((x<<1) & 0xAA)|((x>>1) & 0x55);
        r=r+chr(x)
    return r;

#Convert Z-Wave hex file to binary file with fixed length 128KB-4 bytes.
h = intelhex.IntelHex("serialapi_controller_static_ZW050x_EU.hex")
b = h.tobinstr(0,None, 0xff, 128*1024-4)

#Do a CRC32 reversing the bit order and negating the output
crc = ~(binascii.crc32(reverseBit(b)) & 0xffffffff)

#Unreverse the bit order and output the result little endian
b = b+reverseBit(struct.pack("<I",crc));

#Write to hex file
h.frombytes(array('B',b))
h.tofile("test.hex",format="hex")

#Write to bin file
f=open("flash.bin","w")
f.write(b)
f.close()
```

Figure 21 CRC32 calculation code example in Python

### 7.15 Reset Chip

If the AUTOPROG register bit is set, then this command, 'Reset Chip', will clear the AUTOPROG register bit and restart the chip in normal functional mode.

When the chip is in APM mode and the AUTOPROG register bit is cleared, then clear the AutoProg1 lock bit and power-cycle the chip or toggle RESET_N to bring the device into normal functional mode.

**Note:** In USB and UART mode it is not guaranteed that the four command bytes are returned from the chip.
8 NVR STORAGE

The NVR contains non-volatile lock bit data, data programmed by Silicon Labs and data programmed by the Z-Wave application manufacturer.

A part of the non-volatile data contains fields that stores calibration values and values that describe the application hardware: External NVM, SAW filter, etc. This part is validated by a CRC-16 checksum.

Another part of the non-volatile data in the NVR can be used by the Z-Wave application manufacturer to store unique ID's, application specific calibration data etc.

Refer to [4] for a detailed description of the NVR contents.

8.1 NVR considerations when developing a programmer

Before running the "Erase Chip" command, which erases the code space and the NVR, the programmer MUST read the NVR contents and store the data. Right after the chip is erased it is recommended to write the NVR data. In some cases the NVR data has to be modified, data has to be added and/or data has to be deleted before the data is written to the NVR.

8.1.1 REV field

The NVR space can have many different layouts. The revision field (REV) is used to identify the specific layout. If any changes are to be made to the NVR data the REV field value must be verified to be a layout known by the programmer. The REV field verification must be performed before any changes are made to the NVR. If the layout is unknown then the operator must be warned.

Refer to [4] for a description of the REV field.

8.1.2 CRC16 field

A CRC16 checksum is included in the NVR field called CRC16. This checksum is calculated over a part of the NVR. If the REV field value describes a known layout then the CRC16 checksum must be checked to verify the integrity of the covered data. If the REV field values is a known one and if the CRC16 check fails the operator must be warned. The CRC16 verification must be performed before any changes are made to the NVR.

Refer to [4] for a description of the CRC16 calculation.

8.1.3 Mandatory fields

It is good practice that the programmer always checks that all mandatory fields in the NVR have been filled-in. Refer to [4] for a description of the mandatory fields.
9 PROGRAMMING FLOW

Writing data to the sectors of the flash MUST be performed like this (as depicted in Figure 28):
1. Store the data belonging to a Flash sector in a buffer (from now on called buffer)
2. Strip leading FFh’s from buffer. Set address to the number of stripped bytes.
3. Strip trailing FFh’s from buffer
4. Calculate $t = \text{(number_of_bytes(buffer)-1)} \mod 3$
5. If $t = 0$ go to 14.
6. If first byte in buffer is FFh go to 10.
7. Execute “Write SRAM” command to transfer first data byte in buffer to SRAM @address
8. Execute “Write Flash Sector” command to transfer data in SRAM to a flash sector
9. Repeat executing “Check State” command until Write Flash Sector operation is done
10. Strip first byte from buffer
11. Increment address by one
12. Decrement $t$ by one
13. Go to 5.
14. Execute “Write SRAM” command to transfer first data byte of buffer to SRAM @address
15. Execute “Continue Write” command to transfer the following three data bytes in buffer to the SRAM
16. If buffer is not empty go to 15.
17. Execute “Write Flash Sector” command to transfer data in SRAM buffer to a flash sector
18. Repeat executing “Check State” command until Flash write operation is done
19. If there is more Flash sectors to be programmed go to 1.
20. Done

This section describes how a full programming flow is performed. The flow is described by the use of flow charts. A description of the flow chart box types is shown in Figure 22.

The programming flow is divided into several flow charts. The symbol for Sub Flow Chart, as shown in Figure 22, directs the flow to another flow chart, a sub chart. At the end of the sub flow chart (at the Sub Flow exit symbol) the flow returns to the originating flow chart. The flow charts and the associated notes:

- Figure 23: The main flow chart.
  - When enabling APM, use the method specific for the application. These methods could be: either calling the Z-Wave SerialAPI function, \( \text{ZW_FLASH_auto_prog_set()} \), tie a pin low
or other.

**Note:** If the device does not respond with an APM ACK after the attempt to enable APM mode, no valid APM enabling code has been loaded to the device. This is an error situation. The device must be power cycled to revert to normal operation mode, and proper action must be taken in order to enable the device for APM programming.

- The programmer should store the information whether it has started the programming procedure in normal Programming mode or in Auto Programming Mode (APM).
- The Decision "AutoProg0==0 and AutoProg1==1" test whether the lock bit AutpProg0 is cleared and the lock bit AutoProg1 is set.
- If the decision "AutoProg0==0 and AutoProg1==1" test is positive then a programmer variable (here called APM_LB) should be set true, otherwise it should be set false. The state of this variable is tested in other parts of the programming flow.
- **Note:** A new flow step has been added (red text). This is done to be able to handle the case where programming is initiated of a USB device where a previous programming process was interrupted after the `usb_loader_part2.hex` had been programmed to the device.

- **Figure 24: Enable and Sync Interface**
  - Up to 32 cycles are performed when using SPI. (Bit synchronize)
  - Up to 4 cycles are performed when using UART or USB. (Byte synchronize)

- **Figure 25: Handle Error case and disable programming mode**

- **Figure 26: SPI/UART Program Flash Code Space**
  - When the "Write Flash Sector" has been executed the optional delay depends on the number of bytes (byte_count) that has to be written to the flash sector

- **Figure 27: USB Program Flash Code Space.**
  - USB HWID is the bcdDevice field in the USB device descriptor. It can be read in the Windows registry and in Linux `/sys` file structure.

- **Figure 28: Program Flash sector**
  - Common for all interfaces (SPI/UART/USB)

- **Figure 29: CRC Check**

- **Figure 30: Program NVR**

- **Figure 31: Program Lock Bits**
Figure 23, Main programming flow chart
Figure 24, Enable and Sync Interface
Failed

APM enabled?

Yes

No

APM_LB == 1?

Yes

De-assert
RESET_N

No

Power Cycle or
toggle RESET_N

Send Reset Chip
command

Reset Chip

Failed

Figure 25, Handle Error

Program Flash code
Sector

All Data written
to Flash?

Yes

No

Figure 26, SPI/UART Program Flash Code Space
Clear AutoProg0
Lock bit
Program target code into sector 0 Address 508h-FFFh
Program target code into sector 0 Address 000h-4FFh
Program target code to sectors 1-63
Program usb_loader_part1.hex into flash sector 0
Program usb_loader_part2.hex into flash sector 0
Erase sector 0
Erase Sector
Disable APM by sending "Reset Chip"
Reset Chip
Tell user to unplug/plug (power cycle) device
Is APM_LB=1
No
Yes
Verify OK ?
No
Yes
Program Flash code Sector

Figure 27, USB Program Flash Code Space
Figure 28, Program Flash Sector
Run CRC Check

Wait t\textsubscript{max}, Optional

Check Flash
Status Byte
Check State

CRC Done ?

Yes

CRC Passed ?

No

Yes

Handle error

Figure 29, CRC Check
Program NVR byte
Set NVR

Wait tWP
Optional

Check Flash status
Check State

Busy ?
Yes
No

Read NVR byte
Read NVR

NVR data OK ?
Yes
No

More NVR data to be written ?
Yes
No

Handle error

Figure 30, Program NVR

Program Lock bit
Byte
Set Lock Bits

Wait tWP
Optional

Check Flash status
Check State

Busy ?
Yes
No

Read Lock Bit byte
Read Lock Bits

Lock Bits data OK ?
Yes
No

More Lock Bits data to be written ?
Yes
No

Handle error

Figure 31, Program Lock Bits
## 10 TROUBLE SHOOTER

<table>
<thead>
<tr>
<th>Problem</th>
<th>Cause</th>
<th>Possible solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Can't enter programming mode</td>
<td>Poor power supply (Measure power supply close to chip using a scope)</td>
<td>- Reduce cable lengths</td>
</tr>
<tr>
<td></td>
<td>Poor UART/USB/SPI signal quality (Measure signals for the use programming interface close to chip using a scope)</td>
<td>- Check decoupling</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Check connectors/pogo pins/needles</td>
</tr>
<tr>
<td></td>
<td>Another programming interface has already been enabled</td>
<td>- Power cycle chip, toggle RESET_N</td>
</tr>
<tr>
<td></td>
<td>Programming unit and chip is out of sync (Nothing is returned when trying to enable interface)</td>
<td>- Run &quot;Enable and sync interface&quot; process. See Figure 24</td>
</tr>
<tr>
<td>Can't program</td>
<td>Poor power supply (Measure power supply close to chip using a scope)</td>
<td>- Reduce cable lengths</td>
</tr>
<tr>
<td></td>
<td>Poor UART/USB/SPI signal quality (Measure signals for the use programming interface close to chip using a scope)</td>
<td>- Check decoupling</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Check connectors/pogo pins/needles</td>
</tr>
<tr>
<td></td>
<td>APM is enabled and RESET_N is asserted</td>
<td>- De-assert RESET_N</td>
</tr>
<tr>
<td>Can't run firmware after programming is finalized</td>
<td>APM is enabled and Lock bits AutoProg0=0 and AutoProg1=1</td>
<td>- Clear AutoProg1 and then toggle RESET_N or power cycle chip</td>
</tr>
<tr>
<td></td>
<td>APM is enabled and APM register bit is set</td>
<td>- Run &quot;Reset Chip&quot; programming command</td>
</tr>
<tr>
<td></td>
<td>RESET_N is asserted</td>
<td>- De-assert RESET_N</td>
</tr>
<tr>
<td>Wrong data is returned from chip</td>
<td>APM is enabled and Lock bits AutoProg0=0 and AutoProg1=1 and programming unit and chip is out of sync</td>
<td>- Power cycle chip and retry programming process</td>
</tr>
<tr>
<td></td>
<td>APM is enabled and APM register bit is set and programming unit and chip is out of sync</td>
<td>- Run &quot;Reset Chip&quot; programming command, enable APM and retry programming process</td>
</tr>
<tr>
<td>Only 00h's are returned from chip when reading Flash Code space to verify contents</td>
<td>Read Back protection is enabled</td>
<td>- Run &quot;CRC Check&quot; to verify Flash Code space contents</td>
</tr>
</tbody>
</table>
11 PROGRAMMING EXTERNAL NVM MEMORY CONNECTED TO SPI1

In a typical Z-Wave hardware application there is an external NVM memory connected to SPI1. SPI1 is also used for programming the 500 series chip, so special care must be taken when the NVM memory is to be programming in the application hardware.

To ensure that the 500 series chip is kept in an off state when the external NVM is being programmed the RESET_N signal on the 500 series chip must be pulsed with a signal that fulfills the following widths of the high and low periods:

\[
\begin{align*}
31.25\text{ns} & \ < \ \text{reset low} \ < \ 4.096\text{ms} \\
125\text{ns} & \ < \ \text{reset high} \ < \ 4\text{us}
\end{align*}
\]

This is fulfilled with e.g. 500kHz square wave signal with 50% duty cycle.
12 PROGRAMMING TIMING

Figure 32. SPI Flash Programming Control Signal Timing

Figure 33. SPI Flash Programming Data and Clock Timing
### Table 8, SPI timing

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Cond.</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>f_SCK</td>
<td>SCK frequency</td>
<td>-</td>
<td>-</td>
<td>4.5 MHz</td>
<td></td>
</tr>
<tr>
<td>t Low</td>
<td>SCK low period</td>
<td>110</td>
<td>-</td>
<td>-</td>
<td>ns</td>
</tr>
<tr>
<td>t High</td>
<td>SCK high period</td>
<td>110</td>
<td>-</td>
<td>-</td>
<td>ns</td>
</tr>
<tr>
<td>t Rise</td>
<td>SCK rise time</td>
<td>10% to 90%</td>
<td>-</td>
<td>-</td>
<td>30 ns</td>
</tr>
<tr>
<td>t Fall</td>
<td>SCK fall time</td>
<td>90% to 10%</td>
<td>-</td>
<td>-</td>
<td>30 ns</td>
</tr>
<tr>
<td>t Idle</td>
<td>Idle before Start</td>
<td>32</td>
<td>-</td>
<td>-</td>
<td>ns</td>
</tr>
<tr>
<td>t Setup</td>
<td>Data setup time</td>
<td>2</td>
<td>-</td>
<td>-</td>
<td>ns</td>
</tr>
<tr>
<td>t Hold</td>
<td>Data hold time</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>ns</td>
</tr>
<tr>
<td>t Data</td>
<td>Delay from SCK falling edge to valid data</td>
<td>-</td>
<td>-</td>
<td>109 ns</td>
<td></td>
</tr>
</tbody>
</table>

1 The maximum frequency of the

### Table 9, Reset/Boot, programming command and power-up timing

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Cond.</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>t PE</td>
<td>The minimum time RESET_N must asserted before programming mode is enabled</td>
<td>5.1</td>
<td>-</td>
<td>-</td>
<td>ms</td>
</tr>
<tr>
<td>f B</td>
<td>Boot time when Auto Programming Mode has been enabled</td>
<td>400</td>
<td>-</td>
<td>-</td>
<td>µs</td>
</tr>
<tr>
<td>t Reset pulse</td>
<td>The minimum reset (RESET_N) pulse</td>
<td>32</td>
<td>-</td>
<td>-</td>
<td>ns</td>
</tr>
<tr>
<td>f ACC</td>
<td>Read operation access time</td>
<td>-</td>
<td>-</td>
<td>1.41 µs</td>
<td></td>
</tr>
<tr>
<td>f CE</td>
<td>Chip Erase duration</td>
<td>-</td>
<td>-</td>
<td>51 ms</td>
<td></td>
</tr>
<tr>
<td>f SE</td>
<td>Sector Erase duration</td>
<td>-</td>
<td>-</td>
<td>6 ms</td>
<td></td>
</tr>
<tr>
<td>f WP</td>
<td>Flash Write Byte duration</td>
<td>-</td>
<td>-</td>
<td>6 µs</td>
<td></td>
</tr>
<tr>
<td>f WL</td>
<td>Write NVR/Lock Bits Byte duration</td>
<td>-</td>
<td>-</td>
<td>6 µs</td>
<td></td>
</tr>
<tr>
<td>f WH</td>
<td>CRC-32 check duration</td>
<td>-</td>
<td>-</td>
<td>21 ms</td>
<td></td>
</tr>
</tbody>
</table>

### POWER-UP TIMING

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Cond.</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>t Power-up</td>
<td>Power VDD=3.3V, main crystal not oscillating, RESET_N=0, Stable main crystal oscillation and stable internal mains clock.</td>
<td>-</td>
<td>-</td>
<td>260 µs</td>
<td></td>
</tr>
</tbody>
</table>
13 REFERENCES

[1] Silicon Labs, DSH12206, Datasheet, SD3502
[2] Silicon Labs, DSH12207, Datasheet, ZM5101
[3] Silicon Labs, INS12213, Instruction, 500 Series Integration Guide
[6] Silicon Labs, INS12852, Instruction, Z-Wave ZM5304/UZB applications based on SDK v6.51.02
[7] Silicon Labs, DSH12469, Datasheet, SD3503
Disclaimer
Silicon Labs intends to provide customers with the latest, accurate, and in-depth documentation of all peripherals and modules available for system and software implementers using or intending to use the Silicon Labs products. Characterization data, available modules and peripherals, memory sizes and memory addresses refer to each specific device, and "Typical" parameters provided can and do vary in different applications. Application examples described herein are for illustrative purposes only. Silicon Labs reserves the right to make changes without further notice and limitation to product information, specifications, and descriptions herein, and does not give warranties as to the accuracy or completeness of the included information. Silicon Labs shall have no liability for the consequences of use of the information supplied herein. This document does not imply or express copyright licenses granted hereunder to design or fabricate any integrated circuits. The products are not designed or authorized to be used within any Life Support System without the specific written consent of Silicon Labs. A "Life Support System" is any product or system intended to support or sustain life and/or health, which, if it fails, can be reasonably expected to result in significant personal injury or death. Silicon Labs products are not designed or authorized for military applications. Silicon Labs products shall under no circumstances be used in weapons of mass destruction including (but not limited to) nuclear, biological or chemical weapons, or missiles capable of delivering such weapons.

Trademark Information
Silicon Laboratories Inc.®, Silicon Laboratories®, SiLabs®, SiLabs and the Silicon Labs logo®, Bluegiga®, Bluegiga Logo®, Clockbuilder®, CMEMS®, DSPLL®, EFM®, EFM32®, EFR, Ember®, Energy Micro, Energy Micro logo and combinations thereof, "the world’s most energy friendly microcontrollers", Ember®, EZLink®, EZRadio®, EZRadioPRO®, Gecko®, ISOmodem®, Micrium, Precision32®, ProSLIC®, Simplicity Studio®, SiPHY®, Telegesis, the Telegesis Logo®, USBXpress®, Zentri, Z-Wave and others are trademarks or registered trademarks of Silicon Labs. ARM, CORTEX, Cortex-M3 and THUMB are trademarks or registered trademarks of ARM Holdings. Keil is a registered trademark of ARM Limited. All other products or brand names mentioned herein are trademarks of their respective holders.