

AP3230

Boot ROM Code

TC11IB

32bit

Microcontrollers



Never stop thinking.

Revision History:2002-02V 1.0

Previous Version:V0.3

Page	Subjects (major changes since last revision)
6	Boot ROM version number for new device step added.
8	New features, updated to Table 2 : Boot ROM Options
11	Updated Figure 2: Boot Mode Selection
30	Correction at Section 5.1: Read is done at 0xA0000004-0xA0000006
	Chapter 5: Changes to NAND Flash boot modes, figure 11 etc.

We Listen to Your Comments

Any information within this document that you feel is wrong, unclear or missing at all?
Your feedback will help us to continuously improve the quality of this document.
Please send your proposal (including a reference to this document) to:
mcdocu.comments@infineon.com




Table of Contents	Page
1 TC111B Boot Versioning	6
2 TC111B Boot Options	7
2.1 Reset and Boot	7
2.2 Boot Options	7
2.3 Boot ROM Startup Code	9
2.3.1 Watchdog Handling	9
2.3.2 Other Startup Conditions	10
2.3.3 Boot Mode Selection	10
3 The SSC Bootstrap Loader	12
3.1 Initialization	12
3.2 Entering the SSC Bootstrap Loader	12
3.3 Loading the Startup Code	12
3.4 Exiting the SSC Bootstrap Loader	14
3.5 SSC Boot Algorithm	14
3.6 Software Description	17
3.6.1 Assumptions	17
3.6.2 Software Overview	18
4 The ASC Bootstrap Loader	22
4.1 PC host program - loader.exe	22
4.2 ASC Initialization in Bootstrap Loader	22
4.3 Entering the ASC Bootstrap Loader	22
4.4 Loading the Startup Code	23
4.5 Exiting the ASC Bootstrap Loader	23
4.6 ASC Boot Algorithm	23
4.7 Software Description	25
4.7.1 Assumptions	25
4.7.2 Software Overview	26
5 L_EBU Boot Options	30
5.1 External Memory Boot on Boot Chip Select CS1	30
5.2 Bootstrap Loading from NAND Flash	30
5.2.1 Common	30
5.2.2 Supported NAND Flash Devices	32
5.2.3 NAND Read Algorithm	32
6 Peripheral Mode - Upload via PCI Interface	35
6.1 Reset State of PCI2FPI Interface	35
6.2 PCI Boot	36
6.3 Power Management Setup of PCI2FPI Interface	36
6.4 Subvendor ID, Subvendor Device ID and Registry Entries	36
6.5 Routines Loaded via SSC	37

Table of Contents		Page
7	Peripheral Mode - Upload via L_EBU Interface	38
8	Bootstrap Loading from MMC via ROSA	39
8.1	ROSA Register Set	39
8.2	MMC Boot Algorithm	39
9	Error Handling	42

About this document

The ASC/ SSC boot routines expect some initialization, executed by a global initialization routine at the beginning of the boot ROM code. The routines expect to load executable code.

The context save areas are not initialized by the boot loader. If the user code makes use of CSAs, then the user must set up the CSA memory area. The boot loader subroutines do not use function calls, instead the JL 'Jump and Link' command is used.

For further information please refer to the TriCore Architecture Manual.

1 TC11IB Boot Versioning

The TC11IB boot code follows a standard versioning at boot location 0xDFFFFFFF8 as described here.

Bits 31-16 are used as device revision, where:

31-28= reserved

27-16=design step

Bits 15-0 are used as code revision of ROM program:

15-12= reserved

11-8= incremented each time there is a change in program spec, eg. new routine.

7-4= incremented each time a bug fix release is made.

3-0= a-b for intermediate release

The boot rom version with respect to Stepping is listed in [Table 1](#).

Table 1 Boot ROM Version Identification

Device Step Marking	Boot ROM Version (at 0xDFFFFFFF8)
EES-AA	0x00000000
ES-BA	0x00000000
ES-BB / BB	0x0B15001B

2 TC11IB Boot Options

2.1 Reset and Boot

This chapter describes the different boot options which are provided by the boot ROM. The Reset Status Register (RST_SR) indicates the cause of a reset and the power-on reset latched boot configuration. The Reset Request Register (RST_REQ) defines the boot configuration for a software reset request. These registers are accessed in the beginning of the boot ROM code to determine the boot mode selected.

Both registers contain the status or setting of CFG[3:0], GPIO[2:0] (P0.13, P0.12, P0.11), $\overline{\text{OCDSE}}$ and $\overline{\text{BRKIN}}$. Register RST_SR is read-only, but the boot configuration bits SW_CFG[3:0] within the RST_REQ register can be rewritten. Thus another boot option is configurable with a soft reset.

Note: *Boot configuration is latched on rising edge of $\overline{\text{PORST}}$ (power-on reset). If boot is due to power-on reset, the contents of the memories are undefined. Depending on the type of memory, they could be defined after a wake-up reset from deep sleep, $\overline{\text{HRST}}$ or software reset.*

Note: *Boot code will not distinguish a wake-up reset from deep sleep. Boot code only differentiates hard reset (hard, watchdog, power down, power-on) and soft reset.*

2.2 Boot Options

Table 2 gives the complete list of TC11IB boot rom modes with entry requirements. On boot ROM entry, Program Counter points to 0xDFFF FFFC. Depending on whether it is a hardware $\overline{\text{PORST}}$ reset or software reset, the status of pins or bits of RST_REQ register: $\overline{\text{OCDSE}}$, $\overline{\text{BRKIN}}$, CFG[3:0] and GPIO[2:0] are stored. GPIO[2] is reserved for future use.

Table 2 Boot ROM Options

OCDSE	BRKIN	CFG[3:0]				GPIO[2:0] = P0.x			Mode
						13	12	11	
1	1	X	0	0	1	0	X	X	Reserved INFINEON (Factory test)
1	1	X	0	1	0	0	X	X	ASC0
1	1	X	0	1	1	0	X	X	SSC SPI
1	1	0	1	0	1	0	0	0	EBU as slave, $\overline{CS1}$, buffered Mem-Area
1	1	1	1	0	1	0	0	0	EBU as master, $\overline{CS1}$, buffered Mem-Area
1	1	0	1	0	1	0	0	1	EBU as slave, $\overline{CS1}$, direct connected Mem-Area
1	1	1	1	0	1	0	0	1	EBU as master, $\overline{CS1}$, direct connected Mem-Area
1	1	0	1	0	1	0	1	0	EBU as slave, NAND Flash $\leq 512\text{MBit}$, $\overline{CS1}$, buffered
1	1	1	1	0	1	0	1	0	EBU as master, NAND Flash $\leq 512\text{MBit}$, $\overline{CS1}$, buffered
1	1	0	1	0	1	0	1	1	EBU as slave, NAND Flash $\leq 512\text{MBit}$, $\overline{CS1}$, direct connected
1	1	1	1	0	1	0	1	1	EBU as master, NAND Flash $\leq 512\text{MBit}$, $\overline{CS1}$, direct connected
1	1	X	1	1	0	0	X	X	Reserved INFINEON
1	1	X	1	1	1	0	0	0	Peripheral Mode, Upload via PCI
1	1	X	1	1	1	0	0	1	Peripheral Mode, Upload via EBU (slave mode)
1	1	X	1	1	1	0	1	0	Bootstrap Load from MMC via ROSA
1	1	X	1	1	1	0	1	1	EBU as master, NAND Flash $\geq 1\text{GBit}$, $\overline{CS1}$, buffered

The bit combinations of CFG3, CFG0 and GPIO[1:0] give hardware designers the possibility to choose between different boot options by changing just one bit and not the whole combination, which is important for the very first boot:

- GPIO0 (P0.11) offers the direct connected/buffered option
- GPIO1 (P0.12) offers the NOR/NAND option
- CFG0 offers the $\overline{CS0}/\overline{CS1}$ option
- CFG3 offers the master/slave option

2.3 Boot ROM Startup Code

2.3.1 Watchdog Handling

After RESET the watchdog will be automatically enabled and set to Time-Out mode.

Time-out Mode is indicated by WDTSR.WDTTO = 1. As the name suggests, Time-out Mode lasts for a particular duration, called the Time-out Period. Time-out Mode must be exited before this period is over, or the system is reset. After reset, the time-out period always is 65536 clocks with an input frequency of SYSCLK/16384.

As the watchdog therefore is activated at the hardwired boot options, the loaded code has to trigger the watchdog within the Time-out Period. Some of the boot ROM options will not be able to fulfil this requirement in any case (there is for example no chance to determine, when the PCI system comes up). Thus the first action of the boot ROM startup code is to disable the watchdog within this period of time.

After reset the EndInit protected registers are not protected. With disabling the watchdog the EndInit protected registers are protected again. So all accesses to EndInit protected registers within the boot code routines must be enabled via a password and a modify access to the WDTCN0 register. With disabling the EndInit protection the watchdog automatically starts again and must be disabled after the access to the EndInit protected registers via another modify- and password access to register WDTCN0.

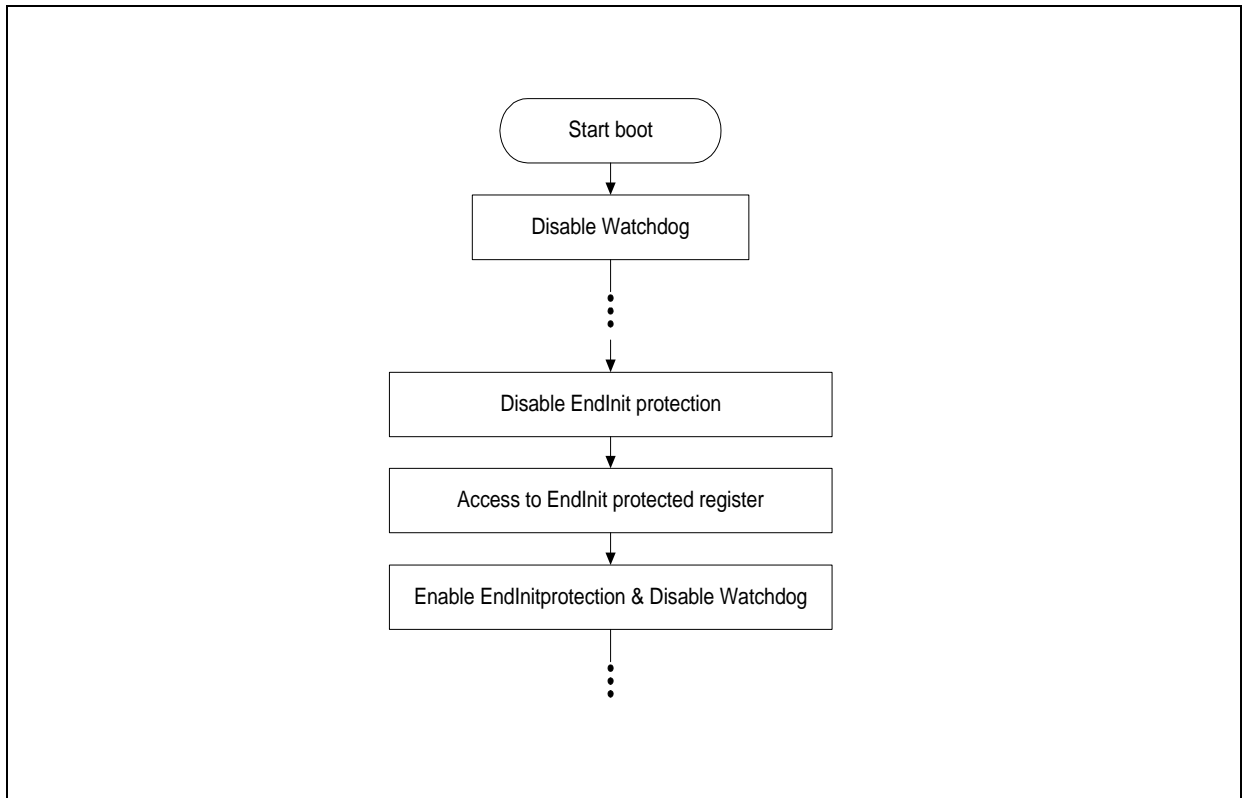


Figure 1 Watchdog Handling

2.3.2 Other Startup Conditions

Supervisor mode has to be activated, as some of the needed registers can only be accessed this way.

2.3.3 Boot Mode Selection

Boot mode selection is done with reference to the registers RST_SR and RST_REQ, according to **Figure 2**.

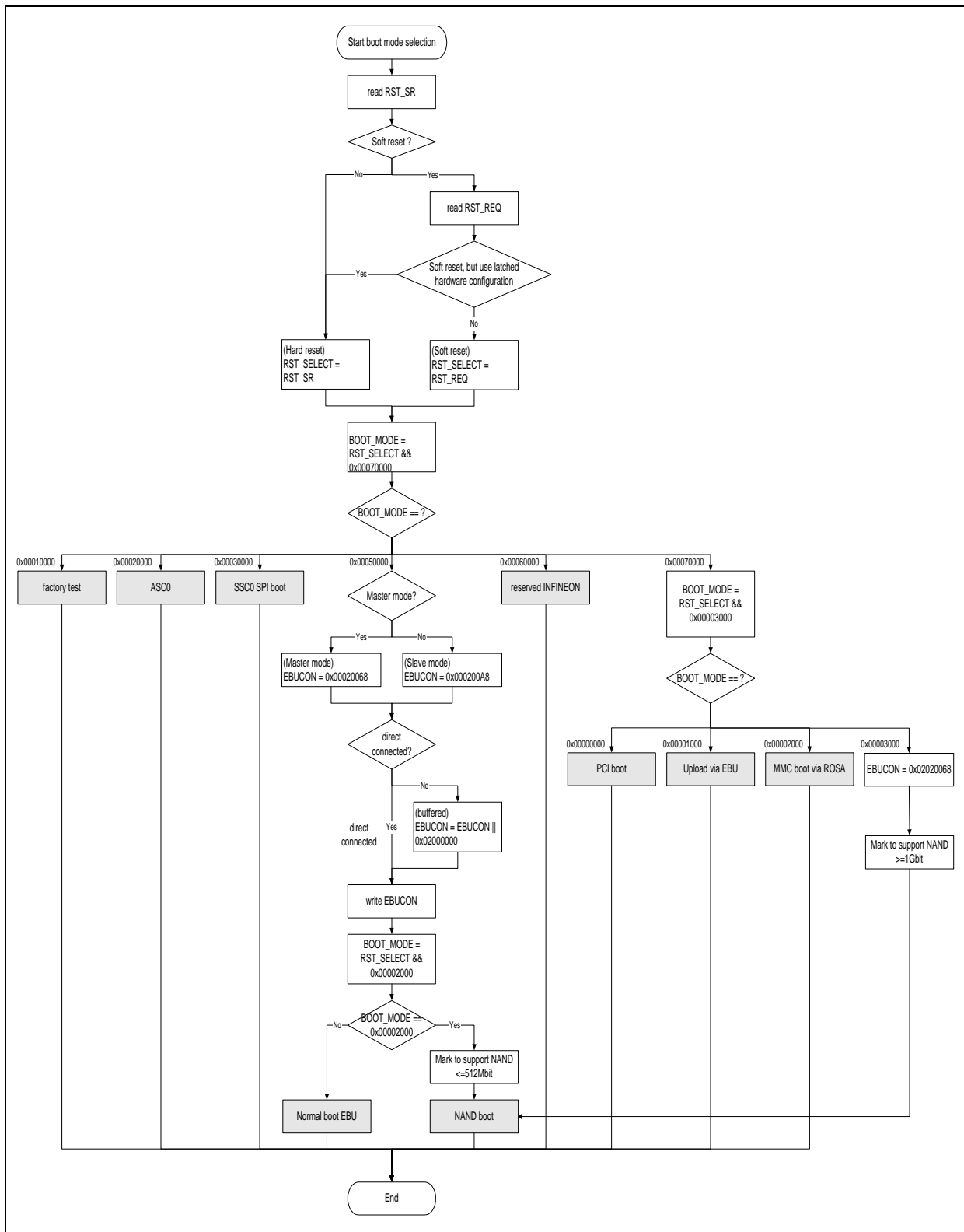


Figure 2 Boot Mode Selection

3 The SSC Bootstrap Loader

The built-in SSC bootstrap loader of the TC11IB boot ROM provides a mechanism to load a program, which is executed after reset, via the SSC interface. In this case no external (ROM) memory or an internal ROM is required for the program code. The bootstrap loader moves code/ data into the internal LMU DRAM.

The SSC bootstrap loader expects external non volatile memory (e.g. EEPROM) at the SSC interface. The loader is compatible with memory devices complying with the Serial Peripheral Interface (SPI) industry standard. The protocol was chosen because it offers a wide range of memory densities and guarantees high speed data transmission as well as noise immunity. The SSC bootstrap loader supports memory devices with 8-bit or 16-bit addressing.

3.1 Initialization

The loader initializes the SSC in the following way:

- CPU is master (serial memory is the slave)
- 8 data bits
- Transmit / Receive MSB first
- Idle clock line = 0
- data latched on leading/ shifted on trailing clock edge
- baud rate defined as 1 M-bit
- SPI memory chip-select via port pin P5.0

3.2 Entering the SSC Bootstrap Loader

The SSC bootstrap loader code is part of the TC11IB boot ROM. The TC11IB enters SSC BSL mode according to configuration of [Table 2](#).

3.3 Loading the Startup Code

After entering SSC BSL mode and the respective initialization of the SSC, the TC11IB sends an SPI-read command (0x03) with a memory start address of zero via the SSC. The bootstrap loader determines the type of memory (8- or /16-bit addressing) by checking at which location the memory identifier byte 'ID_OK' was read from the serial memory.

For memory types with 8-bit addressing this will happen after the first address byte, for types with 16-bit addressing after the second address byte (see **Figure 6**). If the correct memory identifier byte 'ID_OK' cannot be received, the program hangs in an endless loop (see [“Software Overview” on Page 18](#)).

After reception of the memory identifier byte the bootstrap loader reads an index for the number of bytes to be received and stores the received data bytes sequentially into

locations 'RAM_START' (default = 0xAFC00004) through 'RAM_START' + blocksize (variable, see note) of the internal LMU DRAM. Up to 2040 (16-bit) instructions may be placed into the RAM area. After the code is loaded, chip-select line \overline{CS} is set to '1' (inactive) and the bootstrap loader executes a jump to start address 'RAM_START' in the internal LMU DRAM.

The expected memory identifier 'ID_OK' has to be programmed as the first data byte into the external memory (start address: 0x0000) by the user and is expected to be 0x5A. The second byte has to be the 'size_index' byte which informs the loader about the number of bytes to be received: number of bytes = size_index X 16. With this a block of up to 2040 (16-bit) instructions can be defined, unless limited by the size of the serial memory.

Note: Be aware that the block size is always a multiple of 16 bytes.

Table 3 Content of the EEPROM

EEPROM Address	Data value	Meaning
0x0000	0x5A	Memory identifier 'ID_OK'
0x0001	0x01...0xFF	size_index
0x0002...0x0FF2	0x00...0xFF	User code/data

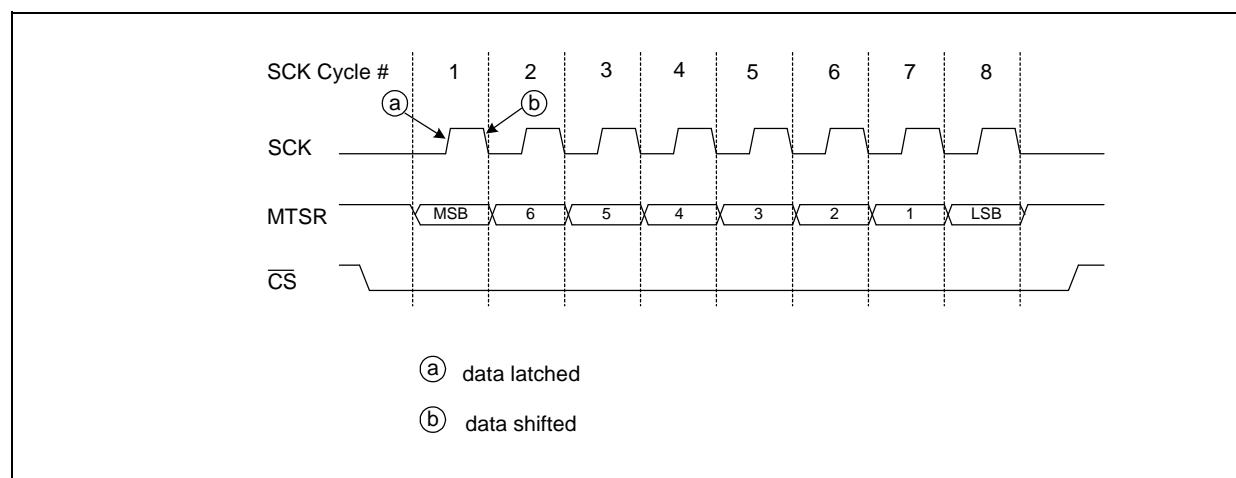


Figure 3 SPI EEPROM: Clockphase Equals Zero Transfer Format

Note: In SSC BSL, SSC is set up with bit PH set to '1' and bit PO set to '0' in register SSCx_CON.

3.4 Exiting the SSC Bootstrap Loader

The SSC bootstrap loader will be terminated by a jump to address 'RAM_START' in the LMU DRAM. A user program may re-initialize system settings which are different to reset state (e.g. Watchdog Timer, System Timer).

3.5 SSC Boot Algorithm

Figure 4 and **Figure 5** show a flowchart of the SSC boot algorithm.

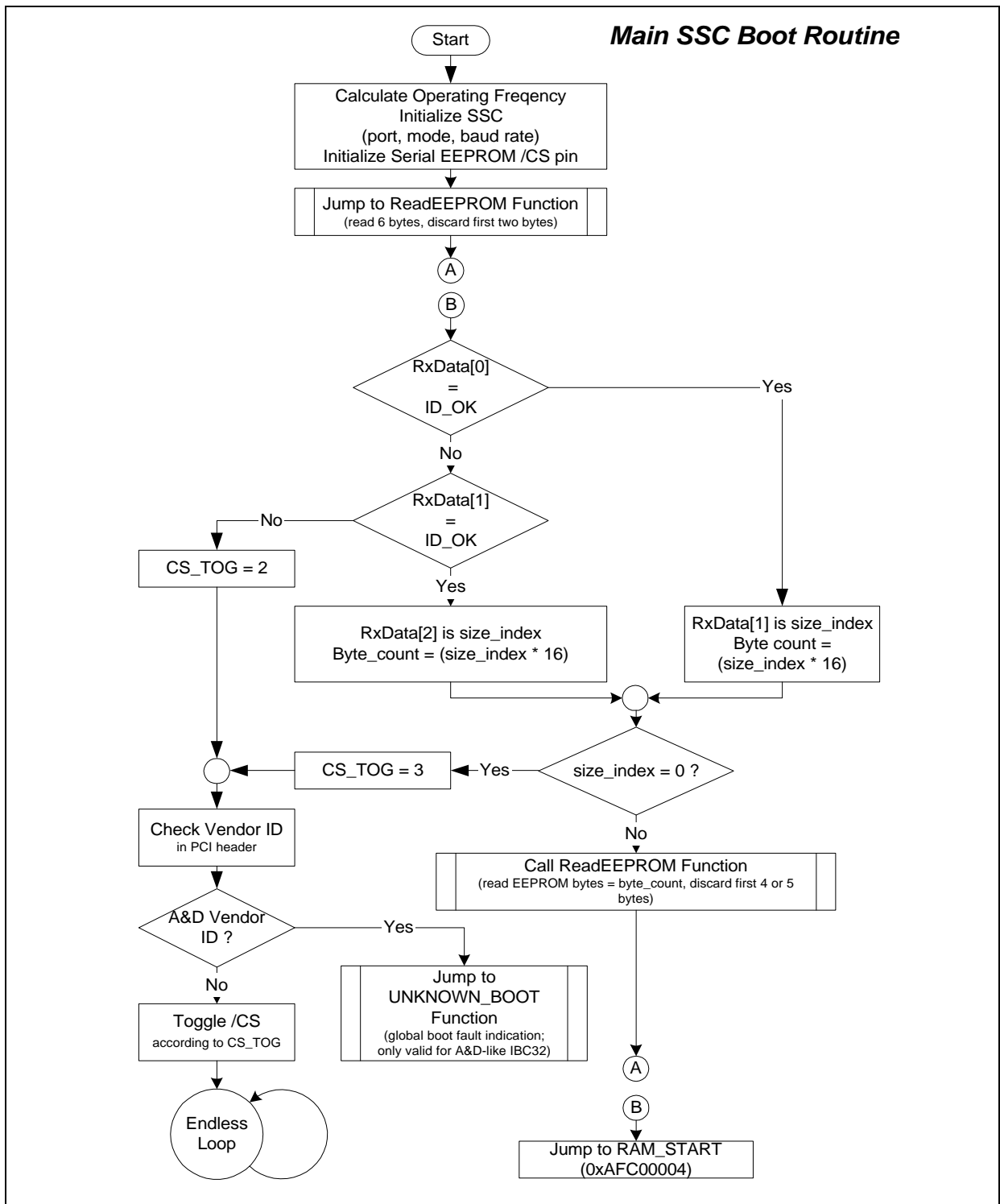


Figure 4 **SSC Boot Algorithm (Sheet 1/2)**

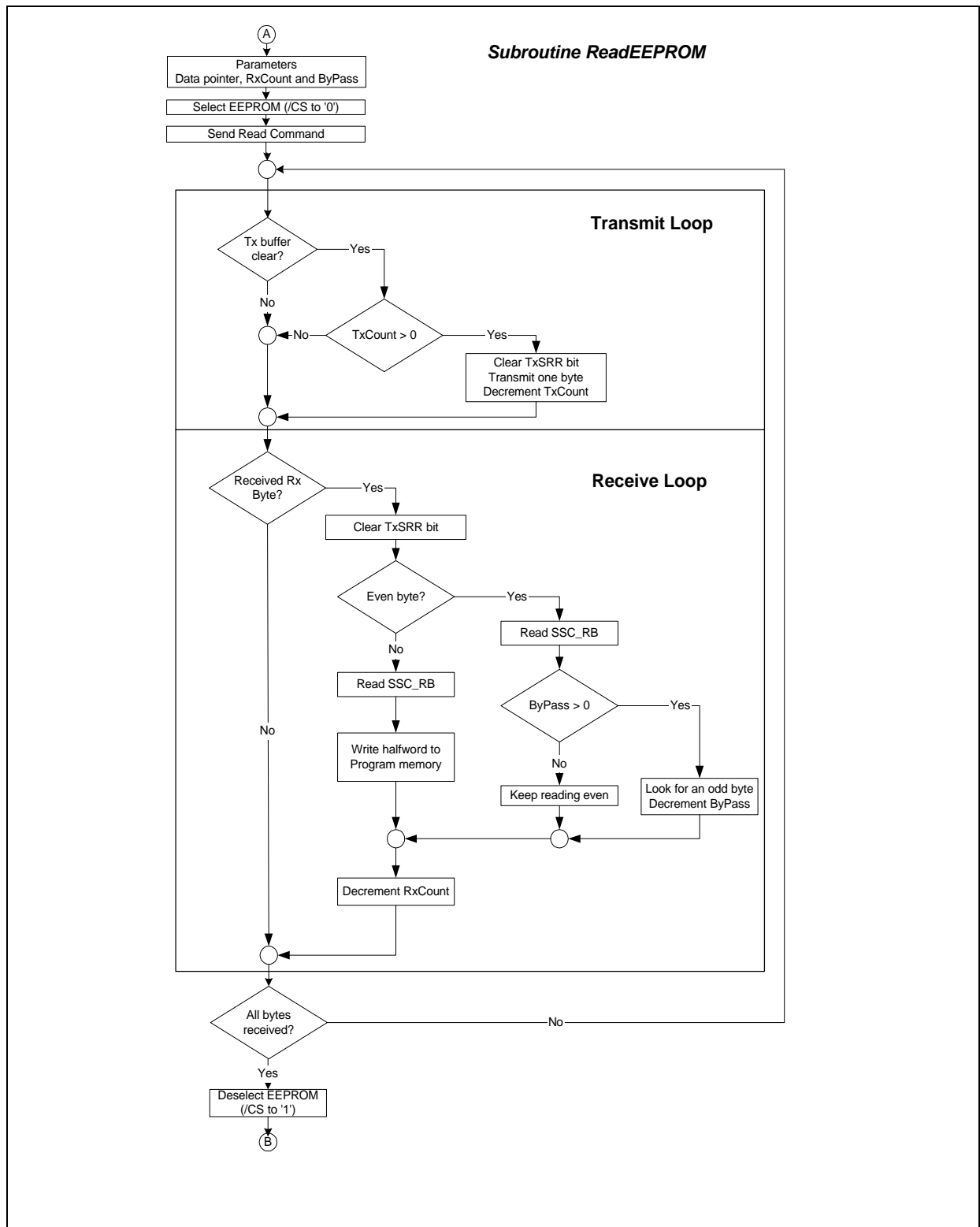


Figure 5 SSC Boot Algorithm (Sheet 2/2)

3.6 Software Description

The programmer must pay special attention to enable/ disable specific equates in the assembly file 'sscboot.asm'. For users, this is otherwise transparent.

3.6.1 Assumptions

The code has been tested with TC111B. Support for other chips as listed is not tested. The watchdog has been disabled. The code is executing fast enough that prevents a receiver overrun.

Note: No SSC errors are monitored or checked by the software.

Table 4 EEPROM Connection and SSC Routing

TC111B	The EEPROM chip select \overline{CS} is connected to port 5.0 and the SSC is connected to port 1 (P1.0/ SCLK, P1.1/ MRST, P1.2/ MTSR).
TC1775	The EEPROM chip select \overline{CS} is connected to port 11.14 and the SSC0 is connected to port 13 (P13.6/ SCLK, P13.7/ MRST, P13.8/ MTSR).

Table 5 Software Defines

CHIP_TC111B	If this define is set true, then the SSC base address is set to 0xF000 0A00. The port address of the SSC is set to 0xF000 2900 and the chip select port is set to 0xF000 2D00.
CHIP_TC1775	If this define is set true, then the SSC base address is set to 0xF000 0A00. The port address of the SSC is set to 0xF000 3500 and the chip select port is set to 0xF000 3300.
SSC_MIN_BAUD	Attempts to limit the SSC baud rate at lower operating frequencies.

Table 6 Software Equates

BAUD_RATE	This is the baud rate at which the SSC will be configured. For the TC111B it is defined as 1 Mbaud.
CLK_FREQ	This should be defined in MHz to the clock frequency of the crystal being used (i.e. 12 MHz clock oscillator should be defined as 12000000).
DIAGNOSTICS	Toggles the $\overline{\text{CS}}$ pin if faults occur (see Chapter 3.6.2)
RAM_START	This is the start address where the code from the EEPROM is loaded to. After all of the bytes are written a jump is performed to this address and program execution begins (default to 0xAFC0 0004).
SSC_CLC_CONTROL	This enables the clock driver circuit and sets the SSC clock to the same frequency as the system clock.
SSC_CONFIG	This is the SSC configuration value which sets the mode to MSB first, 8-bits data and the data is latched on the rising edge of the clock and shifted on the falling edge of the clock.
WAIT_RECEIVE	When enabled, another byte is not transmitted until a byte has been received from the previous transmission.

3.6.2 Software Overview

The software is straight forward and does not use interrupts. Once the SSC boot routine is entered, an access to WDT_CONx registers is performed to clear EndInit. After writing the CLC register, EndInit is set again.

The SSC clock is enabled and the divider is set to '1' so that the SSC clock frequency is equal to the slow FPI frequency = $f_{\text{SYSCLK}}/2$ (48MHz). The system clock frequency for TC111B is determined from the following formula and is fixed for all except K:

$$f_{\text{SYSCLK}} = \frac{N \cdot f_{\text{OSC}}}{P \cdot 2 \cdot K}$$

Note: The system clock frequency for TC1775 is determined by removing P and '2' from the denominator. In TC111B, P is fixed to '1', N is fixed to '16' and the default value for K is '2'.

The calculation of the reload value for SSC baud rate generation is done by:

$$\langle BR \rangle = \left(\frac{f_{SSC}}{2 \cdot baud_rate_{SSC}} \right) - 1$$

For TC111B the following values are true: $baud_rate_{SSC} = 1$ M-bit and $f_{SSC} = 48$ MHz. The result of BR is taken as an 16-bit integer value. The actual $baud_rate_{SSC}$ value is determined from the following formula:

$$baud_rate_{SSC} = \frac{f_{SSC}}{2 \cdot (\langle BR \rangle + 1)}$$

The SSC is configured as defined before, followed by the ports for the SSC and \overline{CS} are setup.

Now everything has been configured for the SSC to properly read the boot code from either an 8-bit or 16-bit serial EEPROM (SPI compatible). A 6-byte read cycle is made to the serial EEPROM (see **Figure 6**) to read the ID_OK and size_index bytes.

The EEPROM only starts sending valid data after the second (8-bit addressable) or third byte (16-bit addressable). If the ID_OK value is found in the third byte then an 8-bit addressable serial EEPROM is assumed. If the ID_OK byte is found in the fourth byte then a 16-bit addressable serial EEPROM is assumed.

If the ID_OK byte is not found in either of the byte positions a fault is assumed. If a fault has been found the program enters an endless loop and hangs from there. The only way to exit this loop is for the user to perform a reset.

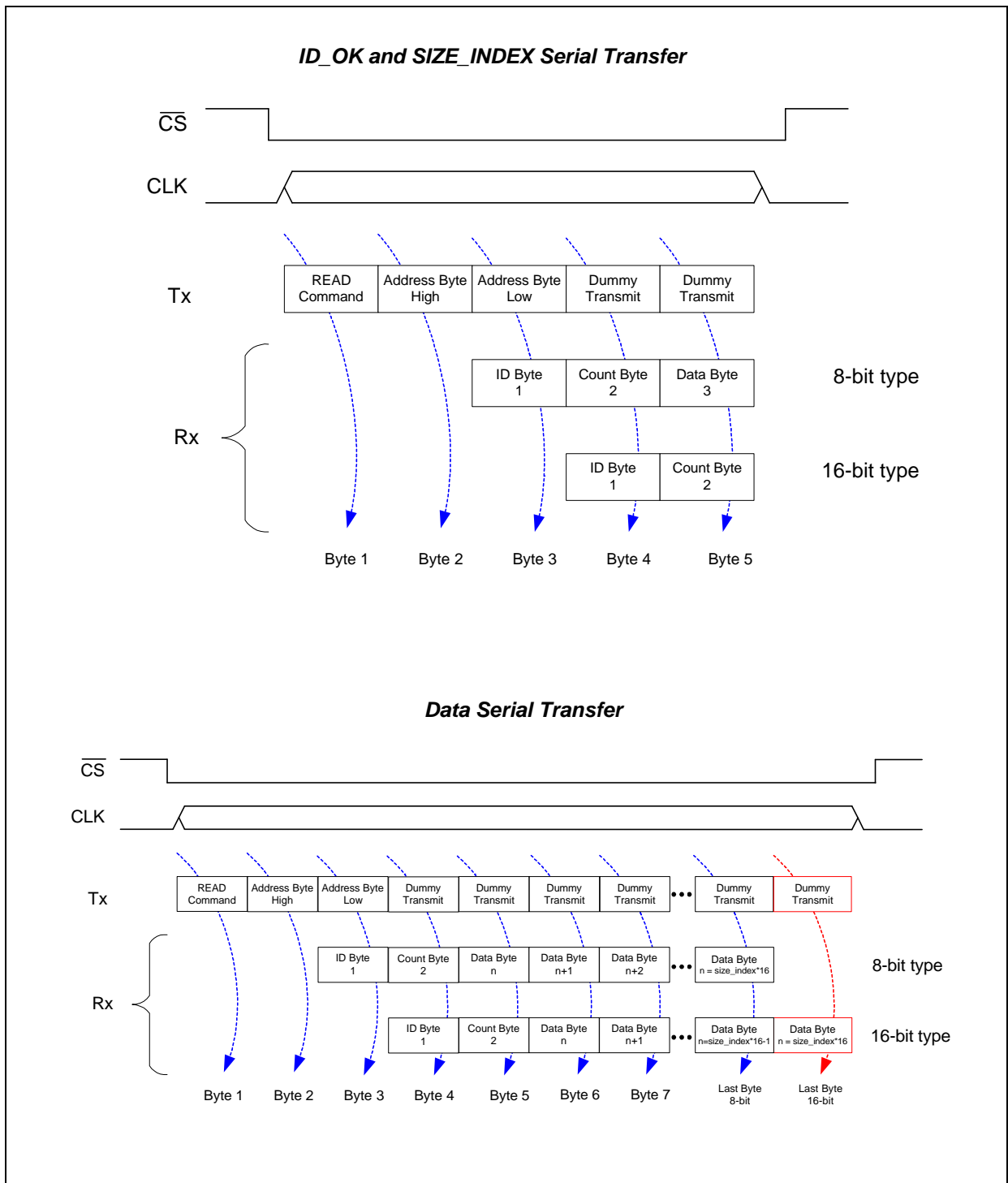


Figure 6 Serial Data Transfer

If the ID_OK byte was read correctly, then the byte that follows is taken as the size_index. The size_index relates to the number of 16-bit instructions to load. The size_index is multiplied by 16 to represent the number of bytes (Byte_count) to be loaded into the internal program memory space (default location is 0xAFC0 0004).

The software then begins reading the serial EEPROM at address zero and discards the first four or five bytes, depending on whether it is an 8-bit or 16-bit addressable serial EEPROM.

To conserve code space a subroutine is added to read the serial EEPROM since the read subroutine has to be able to read either an 8-bit or 16-bit addressable serial memory.

To distinguish between an 8-bit or 16-bit addressable EEPROM two features were added in the subroutine. The first is the ability to detect if one is currently reading an even or odd data byte from the serial memory. The reason for needing to detect an even or odd byte is because the routine writes half-word-wise data into the internal program memory. The half-words are written to program memory whenever an odd byte has been received. The odd and even byte detection is used in conjunction with the bypass feature. The bypass feature serves two purposes: one being to align the instruction to the correct position within the half-word; secondly it eliminates any bytes read from the serial memory that are not program instructions.

The subroutine writes half-words to program memory starting at address RAM_START (0xAFC0 0004). This process continues until all bytes have been read from the serial memory.

Once all bytes have been received and written to program memory, program execution is transferred to the loaded program via a 'Jump Indirect' command to RAM_START.

4 The ASC Bootstrap Loader

The built-in ASC bootstrap loader of the TC111B provides a mechanism to load a program code, which is to be executed after reset, via the serial interface. In this case no external (ROM) memory or an internal ROM is required for the code. The bootstrap loader moves code/ data into the internal LMU DRAM.

A PC DOS-based host program *loader.exe* is available from Infineon for this purpose.

4.1 PC host program - *loader.exe*

To load code (not exceeding 128 bytes) from the PC via the ASC BSL to LMU DRAM, TC111B must be in the ASC Boot mode. The zipped files in *AP323010.exe* should be copied to the same directory on PC. At DOS-prompt, go to the directory where *loader.exe* and *cw3230.dll* are stored and input the command:

loader hexfilename.hex

The HEX file is to be in INTEL HEX format and contain only lines of Record Type *Data*. As an example, *bootled2.hex* can be loaded this way. If loaded correctly, P0.7 will toggle continuously.

4.2 ASC Initialization in Bootstrap Loader

The loader initializes the ASC in the following way:

- baud rate depends on host (auto baud rate detection)
- 8 data bits
- one stop bit
- no parity

4.3 Entering the ASC Bootstrap Loader

After entering ASC BSL mode, the TC111B scans the RxD line to receive a zero byte, i.e. one start bit and eight '0' data bits. From the duration of this zero byte it calculates the corresponding baud rate factor with respect to the current CPU clock. Then, the serial interface ASC is initialized accordingly. Using this baud rate, an identification byte 0xD5 is returned as acknowledgment to the host. This identification byte identifies the device (TC111B) to download code to.

Note: After entering the bootstrap loader sequence a period of time must pass to allow the internal software to begin looking for zero byte transmission from the host.

Note: After reception of the zero byte a period of time must pass to allow calculation of the baud rate. After calculation is completed then the identification byte is sent and the device is ready to receive the upload.

Note: After sending the identification byte the ASC receiver is enabled and is ready to receive the 128 bytes from the host. A half duplex connection is sufficient to feed the BSL.

4.4 Loading the Startup Code

After sending the identification byte the ASC BSL enters a loop to receive 128 bytes (32 words) via ASC. These bytes are stored sequentially into locations 'RAM_START' (default = 0xAFC0 0004) through 'RAM_START' + 128 bytes of the internal LMU DRAM. So up to 64 (16-bit) instructions may be placed into the RAM area. To execute the loaded code the BSL then jumps to location 'RAM_START', ie. the first loaded instruction. The bootstrap loading sequence is now terminated.

It is possible that this loaded routine will load additional code or data. This second receive loop may directly use the already initialized ASC interface to receive data and store it to arbitrary user-defined locations.

Note: The ASC bootstrap loader expects a blocksize of exactly 128 bytes (32 words).

4.5 Exiting the ASC Bootstrap Loader

The ASC bootstrap loader will be terminated by a jump to address 'RAM_START' in the LMU DRAM. A user program may re-initialize system settings which are different to reset state (e.g. Watchdog Timer, System Timer).

4.6 ASC Boot Algorithm

Figure 7 shows a flowchart of the ASC boot algorithm.

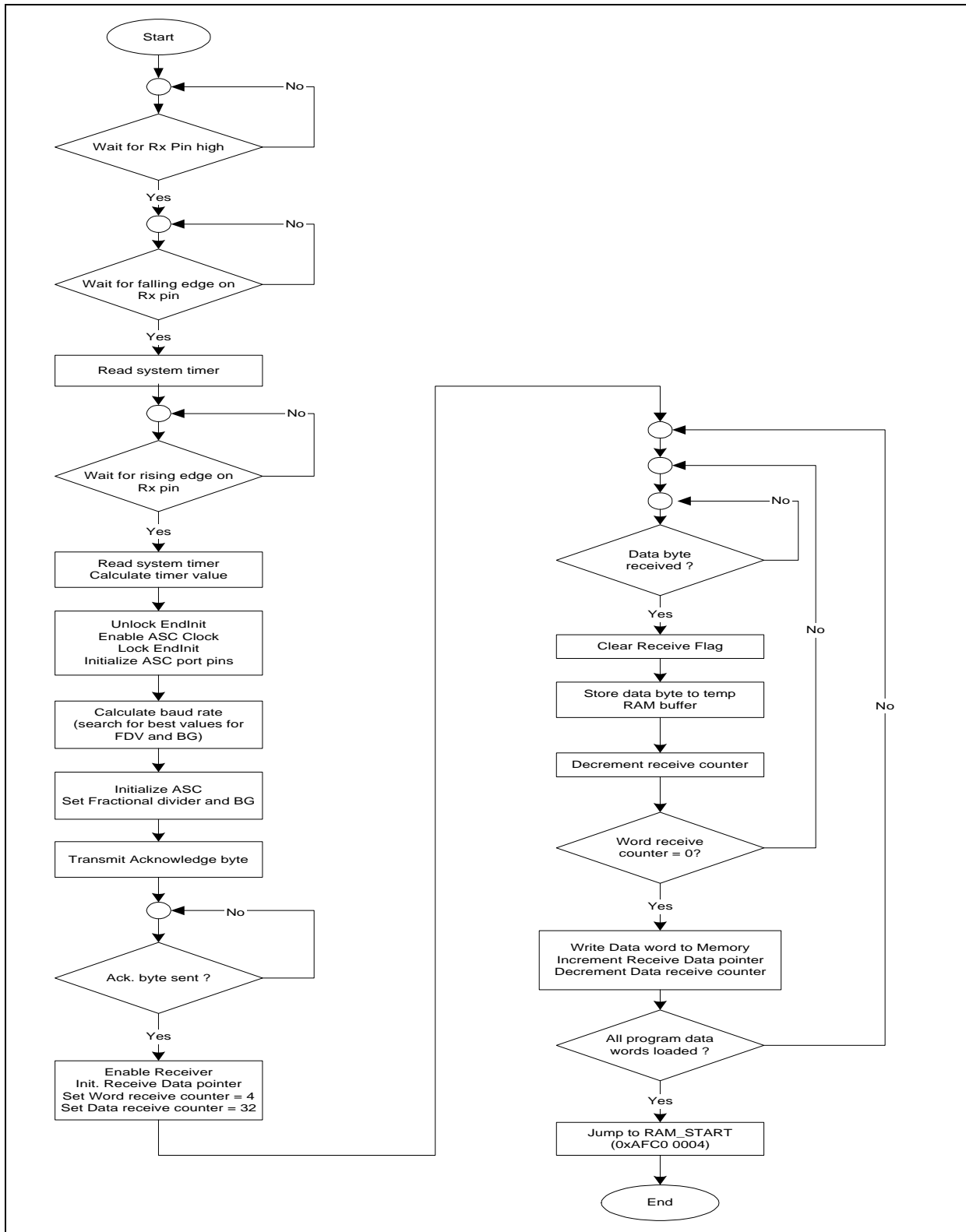


Figure 7 **ASC Boot Algorithm**

4.7 Software Description

The programmer must pay special attention to enable/ disable specific equates in the assembly file 'ascboot.asm'. For users, this is otherwise transparent.

4.7.1 Assumptions

The code has been tested with TC111B. Support for other chips as listed is not tested. The watchdog has been disabled. The code is executing fast enough that prevents a receiver overrun (the receiver is being polled via software for new data, no interrupts are used).

Note: No ASC errors are monitored or checked by the software.

Table 7 ASC0 Routing

TC111B	ASC0 is selected and is connected to port 1 (P1.6/ RXD, P1.7/ TXD).
TC1775	ASC0 is selected and is connected to port 12 (P12.12/ RXD, P12.13/ TXD).

Table 8 Software Defines

CHIP_TC111B	If this define is set true, then the ASC base address is set to 0xF000 0800. The port address of the ASC is set to 0xF000 2900.
CHIP_TC1775	If this define is set true, then the ASC base address is set to 0xF000 0800. The port address is set to 0xF000 3400.

Table 9 Software Equates

ASCM_8ASYNC	This is the ASC configuration value that sets the mode to 8-bit data, 1 start bit, 1 stop bit and asynchronous operation.
RAM_START	This is the address where the data (code) received from the ASC receive buffer is loaded to. After all of the bytes are written to program memory a jump is performed to this address and program execution begins (default to 0xAFC0 0004).

4.7.2 Software Overview

The software is straight forward and does not use interrupts. The first task the ASC boot routine must perform is to determine the baud rate in which the host wants to communicate at.

This is done by capturing the time period to measure a zero byte (one start bit and eight '0' data bits) received at the serial receive pin. Therefore, a polling loop begins by first waiting for the ASC receive pin to be at a high level.

After this the software waits to detect a falling edge. After a falling edge has been detected the system timer is read and the software now waits for a rising edge.

When the rising edge is detected at the pin, the system timer is read again and the time difference between the two reading is calculated. This value represents the time period for one zero byte transmitted by the host and is the basis for calculating the host baud rate with respect to the f_{SYSCLK} clock.

The next step is to enable the ASC clock generator. This requires a password unlock sequence to be performed to the WDT to gain write access to the ASC CLC Control register ($f_{\text{ASC}} = f_{\text{SYSCLK}} / 2 = 48\text{MHz}$). After writing the ASC CLC register the WDT is then locked again.

The ASC bootstrap loader then initialize the respective port pins for ASC serial function, followed with calculating the baud rate. The baud rate is calculated by searching for the best values for the Fraction Divide Value (FDV) and Baud rate Generator (BG) registers.

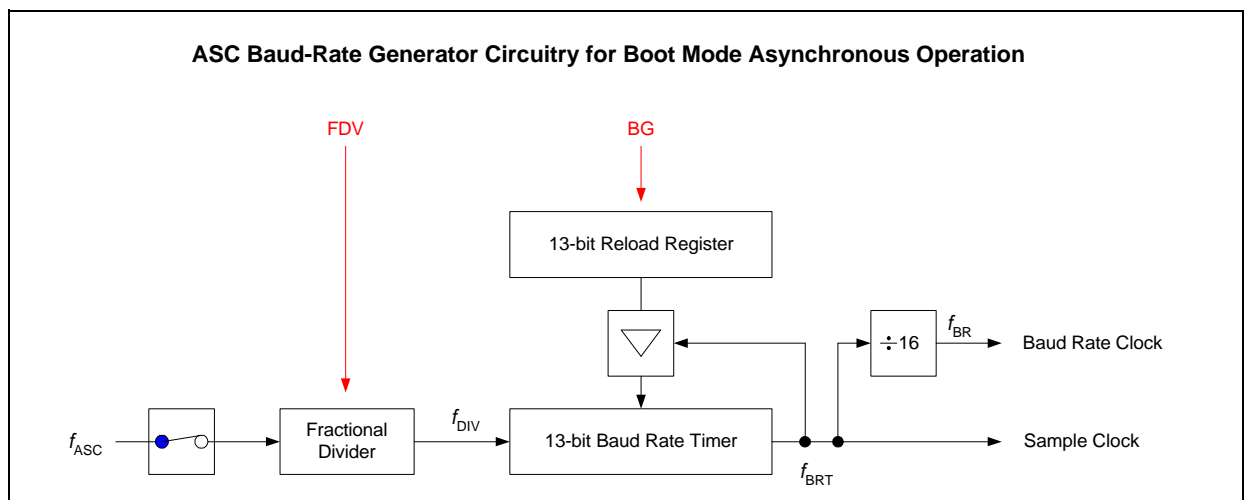


Figure 8 ASC Baud Rate Generator

The values for FDV and BG are derived using an iterative approach with the following formulas:

TimerValue = Measured system timer tick for a zero byte transmission (with start bit)

$$baud_rate_{ASC} = \frac{9 \cdot f_{ASC}}{TimerValue}$$

$$baud_rate_{ASC} = \frac{FDV}{512} \cdot \frac{f_{ASC}}{16 \cdot (BG + 1)}$$

Therefore substituting for $baud_rate_{ASC}$ in the equations and solving for BG :

$$\frac{9}{TimerValue} = \frac{FDV}{(512 \cdot 16 \cdot (BG + 1))}$$

$$\frac{9 \cdot 512 \cdot 16}{TimerValue} = \frac{FDV}{BG + 1}$$

$$FDV = \frac{73728 \cdot BG}{TimerValue} + \frac{73728}{TimerValue}$$

$$\frac{73728 \cdot BG}{TimerValue} = FDV - \frac{73728}{TimerValue}$$

$$73728 \cdot BG = FDV \cdot TimerValue - 73728$$

$$BG = \frac{FDV \cdot TimerValue}{73728} - 1$$

Now one can derive the best-fit values for BG and FDV by checking min/ max boundaries for BG.

See **Figure 9** for details on how this is performed.

Now that the ASC is initialized (receive remains disabled) to the baud rate of the host, an identification byte (0xD5) is returned to the host indicating the device is ready to accept a 128-byte transfer. After the identification byte has been transmitted, the receiver is enabled.

The software word data pointer is initialized to the address RAM_START (0xAFC0 0004). The ASC receive loop waits until it has received four bytes and then makes one write (word) access to the LMU DRAM. This process repeats until all 32 words have been written.

Once all bytes have been received and written to program memory a 'Jump Indirect' command (JI) is performed to the memory location indicated by the RAM_START equate and program execution begins.

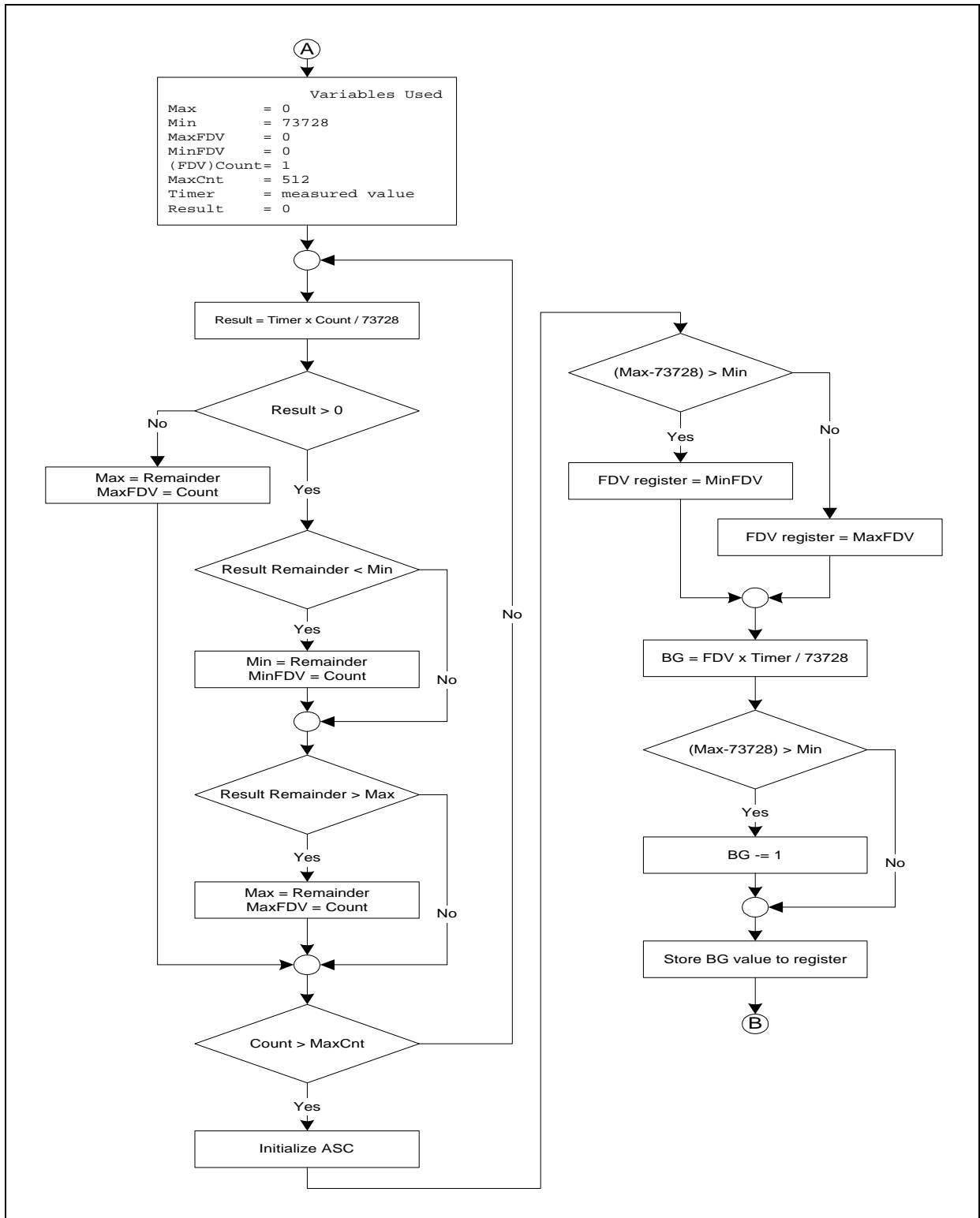


Figure 9 Calculation of FDV and BG to generate Baud Rate

5 L_EBU Boot Options

5.1 External Memory Boot on Boot Chip Select $\overline{CS1}$

This boot option reproduces the hardwired external memory boot mechanism ($\overline{OCDSE} = 1$, $\overline{BRKIN} = 1$, $CFG[2:0] = 100$). There is in principle no difference in behaviour with two exceptions:

In addition to the first 32-bit access on 0xA000 0004 to read the EBU configuration, there is a second 16-bit access on address 0xA000 0006 to test a connection key value 0x6789. This is to ensure that there is really a memory connected and not only the bus default state read. So the boot memory must contain this key value to ensure the boot code will accept the other memory contents. Otherwise boot code will jump to the error handling routine; see [Chapter 9](#) of this specification for further information.

Therefore, if an EBU boot memory will be plugged in a memory adapter at an unknown state, possibly a long time after power on, the bootcode will not be able to read the connection key value, but the bus default state. Thus no deadlock condition will occur.

Note: The bus default state of D15:D0 must not be 0x6789.

Besides of that there is a buffered as well as a direct connected version. The buffered version supports $\overline{CSGLOBAL}$ coincident with boot chip select $\overline{CS1}$, whereas at the direct connected version only $\overline{CS1}$ will be set.

Distinction between master and slave boot is done via CFG3 as with the hardwired boot option.

5.2 Bootstrap Loading from NAND Flash

5.2.1 Common

The NAND Flash boot options support bootstrap loading from NAND Flash devices on $\overline{CS1}$ of L_EBU. **Figure 10** shows the wiring of TC111B L_EBU and the NAND Flash device. The direct connected as well as the buffered mode will be supported. Only an 8-bit device connected to D7:D0 can be used. As 8-bit devices are not known by L_EBU, the boot ROM code has to assemble the data read from the 16-bit interface. 16-bit applications using NAND Flash devices are possible, of course, but boot code will always just take the information of the low byte and ignore the high byte data.

The following signals are not supported by the NAND Flash boot option:

- \overline{WP} (write protect, boot code will not change NAND Flash data)
- \overline{SE} (Spare Area Enable. The spare area of the NAND Flash does not contain any boot information. In addition, the boot code does not support ECC-mechanism to qualify the read data. The routines that write the NAND Flash are responsible for qualifying the data. Besides of that not every NAND Flash contains this Spare Area, so it will not be supported by boot code)
- RY/ \overline{BY} (Ready/Busy Output; Ready/Busy detection of the NAND Flash is also possible by reading a status byte within NAND Flash; therefore the GPIO P3.8 can be reserved)

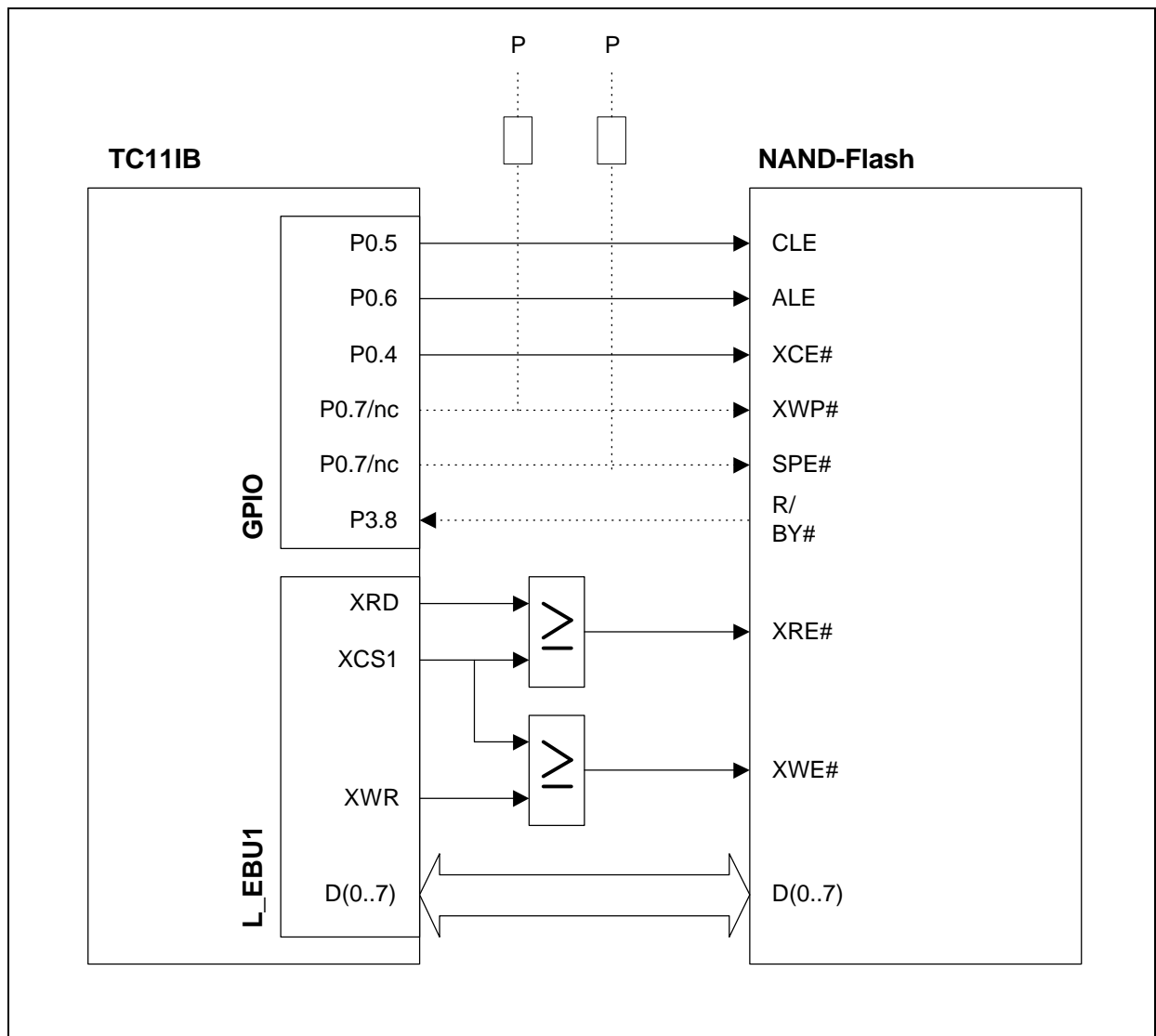


Figure 10 NAND Flash Circuitry

5.2.2 Supported NAND Flash Devices

Due to the fact that there are different NAND Flash devices available, boot code uses only a subset of possible instructions to support as much devices as possible. In addition an algorithm is defined, which will support almost all of the existing block architectures.

Boot code assumes that the connected NAND Flash device has a page size of 256 bytes minimum (virtual page size). Therefore all devices with larger page sizes are also supported. Boot code reads one virtual page with 256 bytes and stores the data in the internal DRAM at address 0xAFC0 0004. After that TC11IB will start from internal DRAM at address 0xAFC0 0004.

5.2.3 NAND Read Algorithm

The NAND boot options supports devices of 1G-bit and bigger (see [Table 2](#)). There are four NAND boot modes with four address cycles for a read command, supporting devices up to 512 M-bit. A fifth NAND boot mode supports devices of 1G-bit and more (first read command + five address cycles + second read command).

First, a Reset command 0xff is passed to the device. After a delay of 500 us (to be sure that internal NAND Flash reset is finished), the read command + address is passed, followed by a delay of 50 μs (to be sure that NAND Flash is ready for data read). After this, the 256 data bytes are read. From the 256 bytes which are read, the first six byte are ecc bytes, the next 249 bytes are data bytes (in reversed order) and the last byte is not use.

Table 10 **Data in NAND Flash**

Address	Data
0x00...0x05	RSC(0)...RSC(5)
0x06...0xFE	Data(248)...D(0)
0xFF	Not used

Because bit error is always a possibility when reading NAND Flash, an error detection and correction algorithm is included. This Reed-Solomon Code can detect minimum 4 bit errors and correct maximum 3 bit errors. If more than 3 errors are detected which cannot be corrected, then the watchdog is activated while the program jumps to an endless loop. The watchdog will generate a soft reset and the Boot sequence starts again.

The 249 bytes are stored to DMU for error checking and correction, so the following addresses are reserved:

Table 11 Data Space Allocation for NAND Flash Boot Mode

Name	Address	Size	Comments
bss.rs_code	0xD000 0800	0x01C	variable
NANDI	0xD000 2000	0x100	data read from NAND Flash
ARRAYS	0xD000 2100	0x308	variable
ECC_DATA	0xD000 2800	0x010	variable

Finally, the 249 bytes are stored sequentially into locations 'RAM_ADDRESS' (=0xAFC0 0004) through 'RAM_ADDRESS' + 248 bytes of the internal LMU DRAM. To execute the loaded code the BSL then jumps to location 'RAM_ADDRESS', ie. the first loaded instruction. The bootstrap loading sequence is now terminated.

The loaded code can be used to load additional code or data, probably by accessing the same NAND Flash at another address. It is possible to use the boot code routine *rsdecode* located at 0xDFFF E000 for error detection and correction. To make use of this Reed-Solomon algorithm, the code or data must first be read from Flash and stored to DMU at *NANDI*. Before the CALL to *rsdecode* routine, register D4 should be updated with the new base address (e.g. RAM_ADDRESS + 248) to copy to.

The following EBU access parameters (AP = Address Phase, CP = Command Phase, DH = Data Hold) are specified, based on an EBU clock of ¼ LMB clock, this means a 42 ns clock period (@f_{CPU}= 12 MHz).

Table 12 EBU Access Parameters

Access Type	Address Phase	Command Phase	Data Hold
Read cycle	3 clk	5 clk	-/-
Write cycle	3 clk	5 clk	4 clk

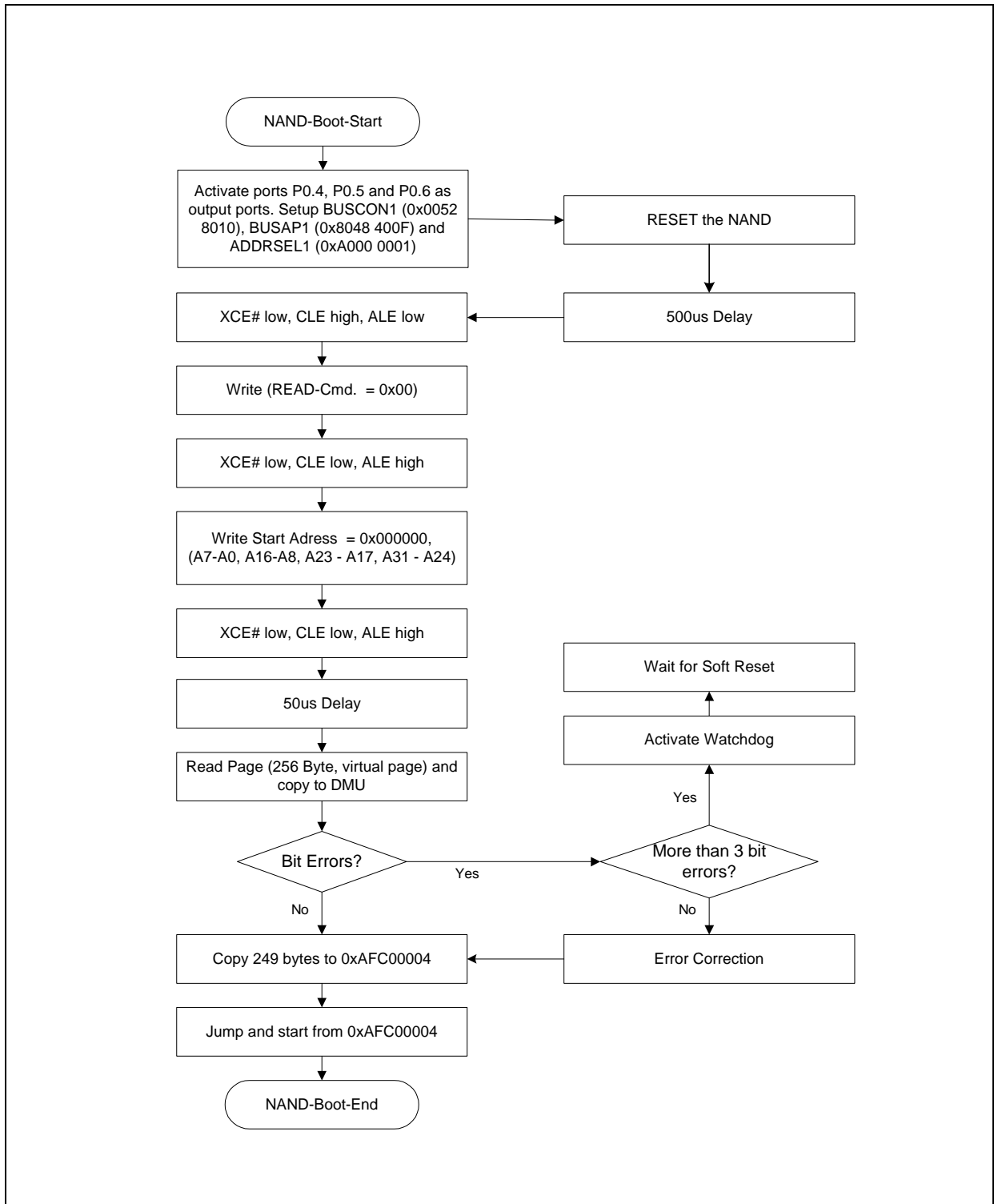


Figure 11 NAND Read Algorithm

6 Peripheral Mode - Upload via PCI Interface

6.1 Reset State of PCI2FPI Interface

The Boot routines included in boot ROM have to install the PCI2FPI Interface and turn on the PCI interface for configuration by the PCI host. As the boot code does not know which external interfaces of the TC111B are used, only a basic configuration, which releases distinct parts of the TC111B internal memory for PCI accesses, will be supported. This basic PCI configuration is the RESET state of the PCI2FPI interface. So the boot code just has to release the PCI interface for configuration accesses of the PCI host.

Reset state of PCI2FPI interface supports function 1 and function 2. The following BARs are activated:

Table 13 Function 1

BAR #	Address Space	Mapped at internal Address	BAR points to...
1	2 M-byte	0xBFEE0 0000	COMDRAM
2	8 M-byte	0xE800 0000	EDRAM & Scratchpad TriCore
4	4 M-byte	0xF000 0000	Peripherals, svm mode

Note: BAR 3,5 and 6 are disabled.

Table 14 Function 2

BAR #	Address Space	Mapped at internal Address	BAR points to...
2	512 K-byte	0xE800 0000	EDRAM TriCore
4	2 M-byte	0xF000 0000	Peripherals, svm mode
5	4 K-byte	0xF800 0000	EBU setup, LMU setup, svm mode
6	32 byte I/O	0xA000 0000	Memory mapped I/O on EBU

Note: BAR 1 and 3 are disabled

6.2 PCI Boot

Bootcode resets UPLOAD KEY value at address 0xAFC0 0000 to 0x0000 0000. Afterwards the PCI2FPI bridge will be activated to accept accesses via PCI bus (Clock Control = 1, PCI Mode = 0x0100 010B).

The PCI host has to install the BAR2 memory region of function 1 in his internal address map A_BAR2 (Alternatively, also BAR2 of function 2 can be selected). Afterwards the PCI host has to load the boot routines in the internal DRAM starting at the mapped address of BAR2 (host memory map) A_BAR2 + 4. Internally this address corresponds to 0xE800 0004 (FPI address map). Loaded code has a maximum size of 512 K-bytes. Following the PCI host will write the 32-bit upload key (0x1234 5678) at address A_BAR2 which corresponds to an access at 0xE800 0000 (FPI address map). TC111B will poll and wait for a change of the PCI UPLOAD key at address 0xAFC0 0000 (TC111B memory map) to become the upload key value. After that TC111B will start from internal DRAM at address 0xAFC0 0004.

Boot code will not support any error handling on PCI2FPI bridge of TC111B. The PCI host is responsible for a correct completion of the PCI access.

Note: The PCI reset behaviour after a soft reset is programmable. As the bootcode does not change the default settings of the PCIBCR registers, any changes within this registers will not be corrected by the boot code routines. So if the PCI2FPI reset values have been changed during runtime, RREXT of RST_REQ has to be set to '1' to bring the PCI2FPI reset values to it's default values for PCI boot.

6.3 Power Management Setup of PCI2FPI Interface

Every FPI agent can force TriCore in power down states via the PMG_CSR register. This register is part of the Infineon peripherals and can be accessed via BAR 4, function 2. So the essential power management setup can be done by the according PC driver. PCI boot does not change the default value of the PMG_CSR register.

6.4 Subvendor ID, Subvendor Device ID and Registry Entries

The Vendor ID, Device ID and Subvendor ID (which is identical with the Vendor ID) will determine which function to activate. The PC registry entry of an add-in- card depends on this information. The registry entry allows to distinguish different PCI devices, so that PC drivers will not interfere. In addition, Subvendor Device ID - SID1 (0x0001) and SID2 (0x0002) allow two registry entries (one for each function) and thus an additional diversification of the according PC drivers.

Table 15 PCI Vendor ID Entries

	SVID	SID
Function 1	Vendor ID	0x0001
Function 2	Vendor ID	0x0002

6.5 Routines Loaded via SSC

In some cases, the default PCI configuration described in the chapter above will not suit in the chosen TC111B design. This is for example, if additional memory is used on L_EBU. In this case the PCI upload can be changed through booting from an EEPROM connected to the SSC. This EEPROM contains the PCI default configuration of the PCI2FPI interface. Please note that the loaded code via SSC is responsible for installing the PCI interface. There is no support via the boot ROM code through any kind of enhanced PCI boot option. Besides the SSC0 boot all other boot options can be used to install a different reset state of the PCI2FPI bridge. Therefore an additional PCI boot option can be installed.

7 Peripheral Mode - Upload via L_EBU Interface

Boot code resets UPLOAD KEY value at address 0xAFC0 0000 to 0x0000 0000. Afterwards the EBU will be configured to accept accesses from an external master via EBU (EBUCON = 0x0200 00AE).

The EBU master has to load the boot routines to the internal DRAM starting at the 24-bit EBU address 0x00 0004 (A[23:22] = 00b-> aext0 [9:0] = 1110 1000 00b -> access to TC111B internal memory map 0xE800 0004). The loaded code has a maximum size of 512 Kbyte. Following the EBU master will write the 32-bit UPLOAD KEY (0x1234 5678)) at address 0x0000 0000 (external host memory map, internal memory FPI map 0xE800 0000). TC111B will poll and wait for a change of the UPLOAD KEY at address 0xAFC0 0000 to get the upload key value. After that TC111B will run from internal DRAM at address 0xAFC0 0004.

8 Bootstrap Loading from MMC via ROSA

This boot option supports bootstrap loading from MultiMediaCard “MMC” (see “The MultiMediaCard System Specification”). The connection of the MMC-Bus with the TriCore will be set-up with an MMC adapter named 'Record on Silicon Adapter' ROSA (see “MultimediaCard Adapter Specification and VHDL Reference”), which is connected to the TC111B slow FPI-bus. The loader reads 512 bytes from an MMC card and writes it to the RAM_START address (default 0xAFC0 0004) in the internal DRAM. After that, the loader jumps to this address to execute the loaded code.

Please note the following information and restrictions related to the MMC bootstrap loader:

- only one MMC card will be supported
- MMC interrupts will not be supported
- only Flash- and ROM-cards will be supported
- data will be read per single block transfer mode of 512 bytes
- on any error condition, the loader will branch to “Error Handling”

8.1 ROSA Register Set

Table 16 shows the ROSA register set which is particularly described in “The MultiMediaCard System Specification”.

Table 16 ROSA Registers

Register	Address	Comment
MMCI_CLC	0xF000 0B00	MMCI Interface Clock Control Register
MMCI_SRC	0xF000 0BFC	MMCI Interface Service Request Control Register
MMCI_CMD	0xF000 0B14	MMCI Command Register
MMCI_DATA	0xF000 0B10	MMCI Data Register
MMCI_ID	0xF000 0B08	MMCI Interface Identification Register

8.2 MMC Boot Algorithm

The flow chart in **Figure 12** shows the MMC boot algorithm.

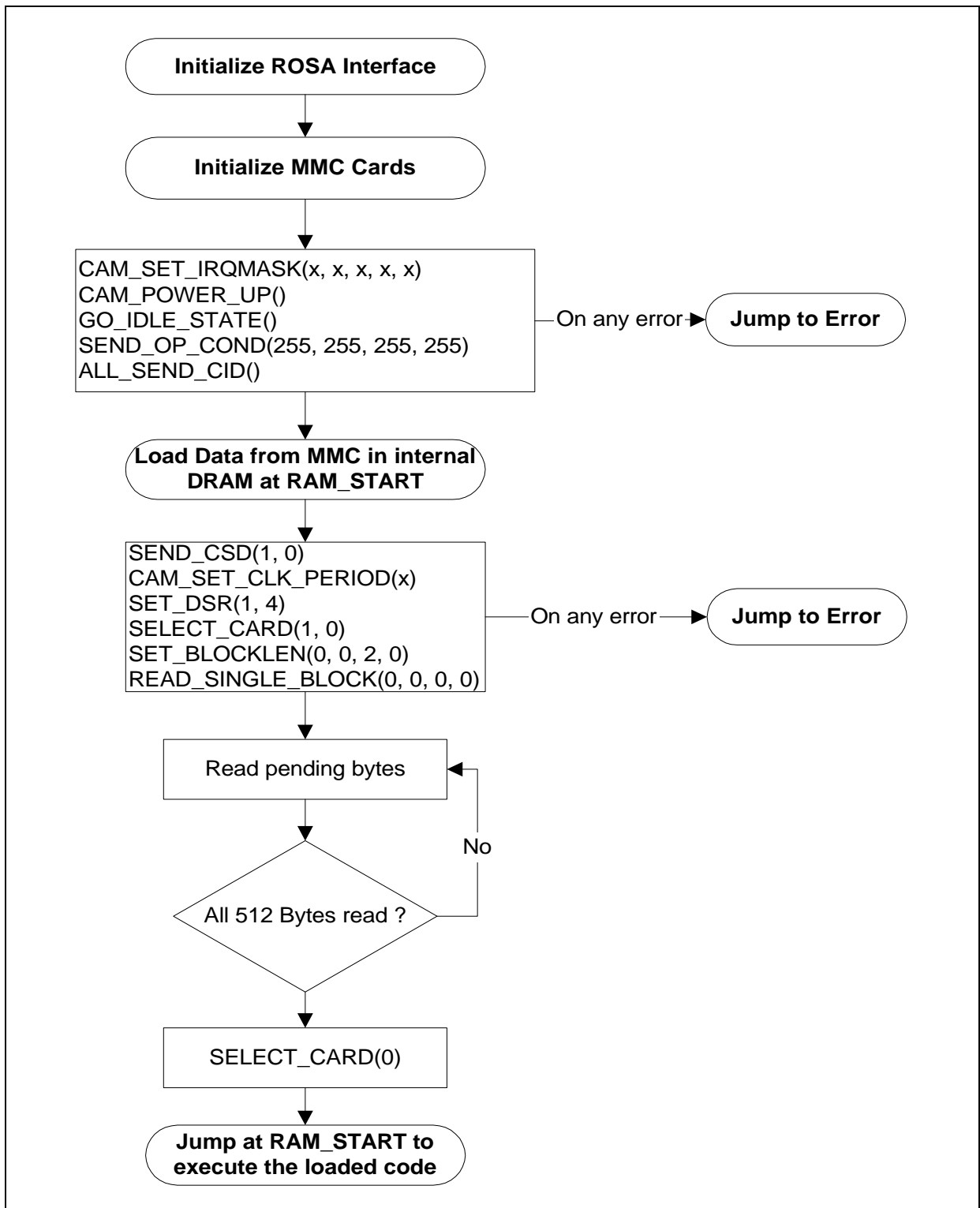


Figure 12 MMC Boot Sequence

Table 17 Command Overview

Command	Meaning
ALL_SEND_CID()	Send a broadcast to discover a card; The card send it's card ID (CID) back
CAM_POWER_UP()	Power on; Set the MMC cards in idle state
CAM_SET_CLK_PERIOD(x)	Set the MMC bus frequency
CAM_SET_IRQMASK(x, x, x, x, x)	Set the IRQ mask register of the ROSA interface
GO_IDLE_STATE()	Reset all cards to idle state
READ_SINGLE_BLOCK(x, x, x, x)	Read the block
SELECT_CARD(x)	Bring the selected card in inactive state
SELECT_CARD(x, x)	Select a card by it's relative address and bring it in transfer state
SEND_CSD(1, 0)	Read the card specific register (CSD) from the addressed card
SEND_OP_COND(x, x, x, x)	Ask all cards in idle state to send their operation condition register and bring it in ready state; bit 31 in the response indicates, that a minimum of one card is busy
SET_BLOCKLEN(x, x, x, x)	Set the blocklength for the following block transfer
SET_DSR(x, x)	Programs the driver stage register (DSR) of all cards
SET_RELATIVE_ADDR(x, x)	Assign a relative address to the card

9 Error Handling

If an error occurs during some parts of the boot sequence, port 4.9 will be set up as output port and will toggle permanently. There are exceptions e.g. during SSC bootstrap loader mode or ASC bootstrap loader mode.

Infineon goes for Business Excellence

“Business excellence means intelligent approaches and clearly defined processes, which are both constantly under review and ultimately lead to good operating results.

Better operating results and business excellence mean less idleness and wastefulness for all of us, more professional success, more accurate information, a better overview and, thereby, less frustration and more satisfaction.”

Dr. Ulrich Schumacher

<http://www.infineon.com>