NTM88 (Re)programming via SPI

In the NTM88, write and erase operations on the FLASH are performed by the Flash Memory Controller block, which can be accessed by an external MCU via SPI. The SPI block may be enabled by either two methods:

- The NTM88 application software writes to '1' the SPIEN control bit at address $1802. This requires the presence of a pre-programmed bootloader program in the NTM88 that coordinates with the external MCU to enable SPI when necessary. This method can be used to perform Over The Air reprogramming of the NTM88.
- At power application, an external MCU holding the PTA0 pin low for greater than the time t_{SPI_EN} specified in the datasheet. In this case a pre-programmed bootloader program is not needed, but the external MCU must be able to control the power supply of the NTM88. This method can be used to program a blank NTM88.

This document first gives information on the SPI block, then details the two methods allowing to enable SPI, and finally explains the basic FLASH write and erase operations.

Contents

1. Preventing and handling errors during SPI transfers

a. Information on the SPI block

The NTM88 hardware SPI block is configured as a standard slave SPI block. Once SPI is enabled, the master SPI, also referred to as External MCU in this document, can directly perform read commands on the RAM and FLASH memory, and write commands on the RAM memory.
On the NTM88 side, the transfers are handled at block-level, and not application level, which means that no application running is needed for the transfers to be performed. The only point to ensure is that the SPI block is not disabled while the transfers are ongoing by writing '0' to SPIEN, or by the MCU entering a STOP mode or resetting.

b. Handling SPI errors

A read or write command transmitted by the master SPI can fail due to several reasons listed in section "Serial Peripheral Interface" of the user manual. When a parity error, clock fault or bus contention fault occurs during a transfer, the command transmitted during this transfer is not executed and the command transmitted during the following transfer is ignored in order for the SPI block to clear the error.

The External MCU must thus be careful to always check the status of the transfer returned by the NTM88 and perform a re-try (i.e. send the command again) as long as it has not been successfully executed. The user must keep in mind that after an error occurred, the NTM88 ignores the command of the following transfer and returns a status indicating that the command was ignored.

c. Avoiding bus contention faults

One of the cause of error during a transfer is bus contention fault. This is due to the fact that in the NTM88 chip, the SPI and the CPU share the internal address, data and control bus, and are arbitrated such that the SPI will take priority over the CPU if the SPI and CPU request access at the same time. In case the SPI block requests access to a bus while the CPU is already accessing it, access to the bus is denied to the SPI, resulting in a bus contention fault, and will be granted after the CPU completes its current access.
Options are available to prevent bus contention faults during SPI communication:

- The NTM88 CPU and SPI must not try to access the same resource at the same time (RAM, Flash and peripheral bus are separate).
- The NTM88 CPU can be halted by setting the CORE_TR_HOLD bit of the SPIOPS register. This bit can be set by the NTM88 program, or via SPI by the master SPI. This bit must be cleared via SPI by the master SPI to allow the NTM88 program to resume.
  Note that halting the CPU does not halt the clocks, so the functions relying on the clocks, like the watchdog, are still working when the CPU is halted. The watchdog can thus be used to ensure that the NTM88 does not remain stuck in RUN with the CPU halted in case the master SPI fails to clear the SPIOPS register.

2. Preparing for SPI transfers

a. With a pre-programmed bootloader program

When a bootloader application is present in the NTM88, it can enable the SPI block by writing '1' to SPIEN. The External MCU thus only need to notify the NTM88 that SPI transfers are requested, and the NTM88 application enables the SPI block. The notification can be done via GPIO for example: enabling KeyBoard Interrupt on any of the PTA[3;0] pin allows the External MCU to wake up the NTM88 from STOP1 by lowering the KBI pin. Refer to the section "Keyboards Interrupts" of the user manual for more information.

When the NTM88 detects a transfer request, the application could enable SPI and then indicate to the External MCU that it is ready for the transfers by toggling PTB0 or PTB1 GPIO. A PTA pin cannot be used in this case because they are automatically configured to be driven by the SPI block as soon as SPI is enabled, and cannot be used as GPIOs anymore until SPI is disabled.
After that, the NTM88 CPU should halt itself by setting the CORE_TR_HOLD bit of SPIOPS register in order to avoid bus contention faults during the transfers.

An example of connections between the NTM88 and External MCU is illustrated in below figure.

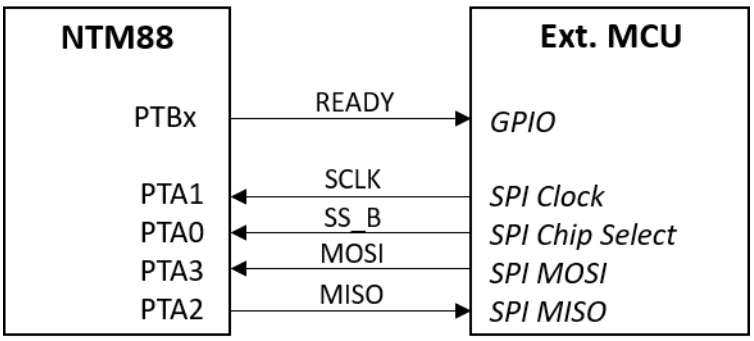

Figure 1: Example of connections when a bootloader program is present in the NTM88

It is expected that the External MCU clears the CORE_TR_HOLD bit at the end of the transfers in order to allow the NTM88 to resume operations. To prevent the NTM88 from being halted for a long time in case the External MCU fails to clear the bit, the watchdog can be enabled in the application. When the watchdog timeout is reached, this triggers a reset that automatically resumes the NTM88. This implies that the External MCU must service the watchdog via SPI in order to perform SPI transfers for a duration longer than the watchdog timeout. To reset the watchdog timeout, a write command must be performed to SIMRS register at address $1800.

When the NTM88 CPU is resumed, the SPI block can be disabled by the application by clearing the SPIEN bit.

b. Without a bootloader program

When no program enabling SPI via register manipulation is present in the NTM88, the External MCU must enable the SPI block of the NTM88 by holding PTA0 pin low at Power On Reset for greater than the time t_{SPI_EN} specified in the datasheet.
Also, in order to prevent the MCU from resetting and disabling the SPI block, pin PTA4/BKGD must also be held low to force the MCU to start in BACKGROUND DEBUG mode.
In summary, the External MCU must hold low both PTA0 and PTA4 at Power On Reset for a time greater than t_{SPI_EN}. This way, the MCU starts in BACKGROUND DEBUG mode with the SPI block enabled.

After that, the External MCU can write '1' via SPI to CORE_TR_HOLD bit in SPIOPS register at address $0038, to prevent SPI bus contention with the debug circuit. Then, the External MCU can start the flash programming sequence.

Once all SPI transfers have completed, the External MCU must do a Power On Reset of the NTM88 while maintaining PTA4 at high state in order to allow the MCU to start normally in non-debug mode. If this is not done, the MCU will continue running in BACKGROUND DEBUG mode, which consumes more power than the non-debug mode and prevents STOP1 entry.

An example of connections between the NTM88 and External MCU is illustrated in below figure.

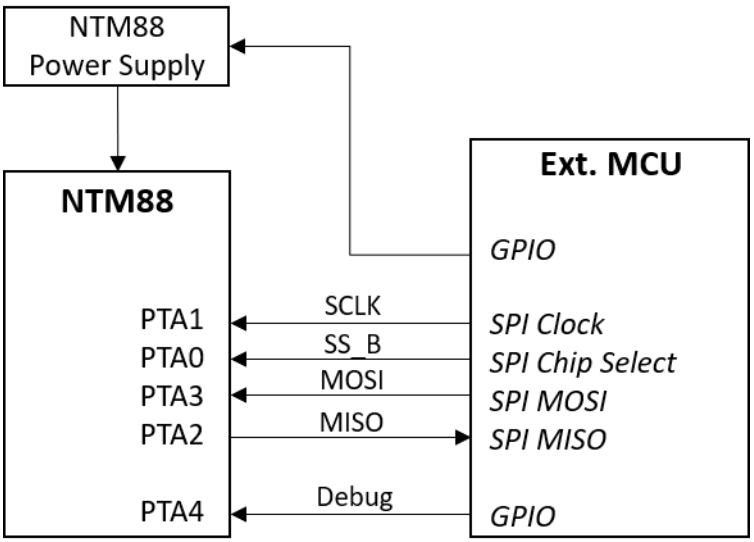

Figure 2: Example of connections when no bootloader program is present in the NTM88

3. Performing Write and Erase commands on the NTM88 FLASH

a. Selecting the FLASH address range

In the SPI command format, the address to read or write is 13-bit long, resulting in a range from $0000 to $1FFF. The NTM88 FLASH address range is $C000 - $FFFF so additional bits are necessary to address the full range of the FLASH.

The last two bits of the SPIOPS register, FLS_ADDR1/0, allow to select the FLASH range to address as described in the table below:

Table 1: Mapping of SPI address to FLASH address

SPI Address Range		FSL_ADDR[1:0]			
		00	01	10	11
Start	0x0800	0xC000	0xD000	0xE000	0xF000
End	0x17FF	0xCFFF	0xDFFF	0xEFFF	0xFFFF

Note that SPI Address Range 0x0000-0x07FF corresponds to the NTM88 RAM memory (valid range from 0x0000 to 0x028F)

When the External MCU wants to access an address in FLASH, the SPIOPS_FSL_ADDR[1:0] field must be configured first with the appropriate value, and then the READ or WRITE command can be transferred along with the 13-bit address from 0x0800 to 0x17FF.

Example:

- To access address $C010 in FLASH, FLS_ADDR1/0 must be set to '00' and the 13-bit address transferred via SPI must be equal to 0x0810.
- To access address $E010 in FLASH, FSL_ADDR1/0 must be set to '10' and the 13-bit address transferred via SPI must be equal to 0x0810.

Important note on the SPIOPS register: the SPIOPS register located at address $0038 has the following fields:

Table 2: SPIOPS register fields (address $0038)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
-	-	-	-	-	CORE_TR_HOLD	FSL_ADDR1	FSL_ADDR0

Setting the CORE_TR_HOLD bit halts the NTM88 CPU, and clearing the bit resumes it. As described earlier in this document, it is recommended that this bit remains set as long as the transfers have not completed in order to avoid bus contention faults. This implies that when the External MCU writes in the SPIOPS register to update the FSL_ADDR1/0 fields, care must be taken to keep the CORE_TR_HOLD bit set, in order to keep the NTM88 halted.

b. FLASH erase and program flow

i. Flow overview

The flow to perform page erase or byte programming is copied below. It is extracted from the "Flash Memory Controller (FMC) module" section of the user manual which also contains the description of the FCDIV, FSTAT and FCMD registers.

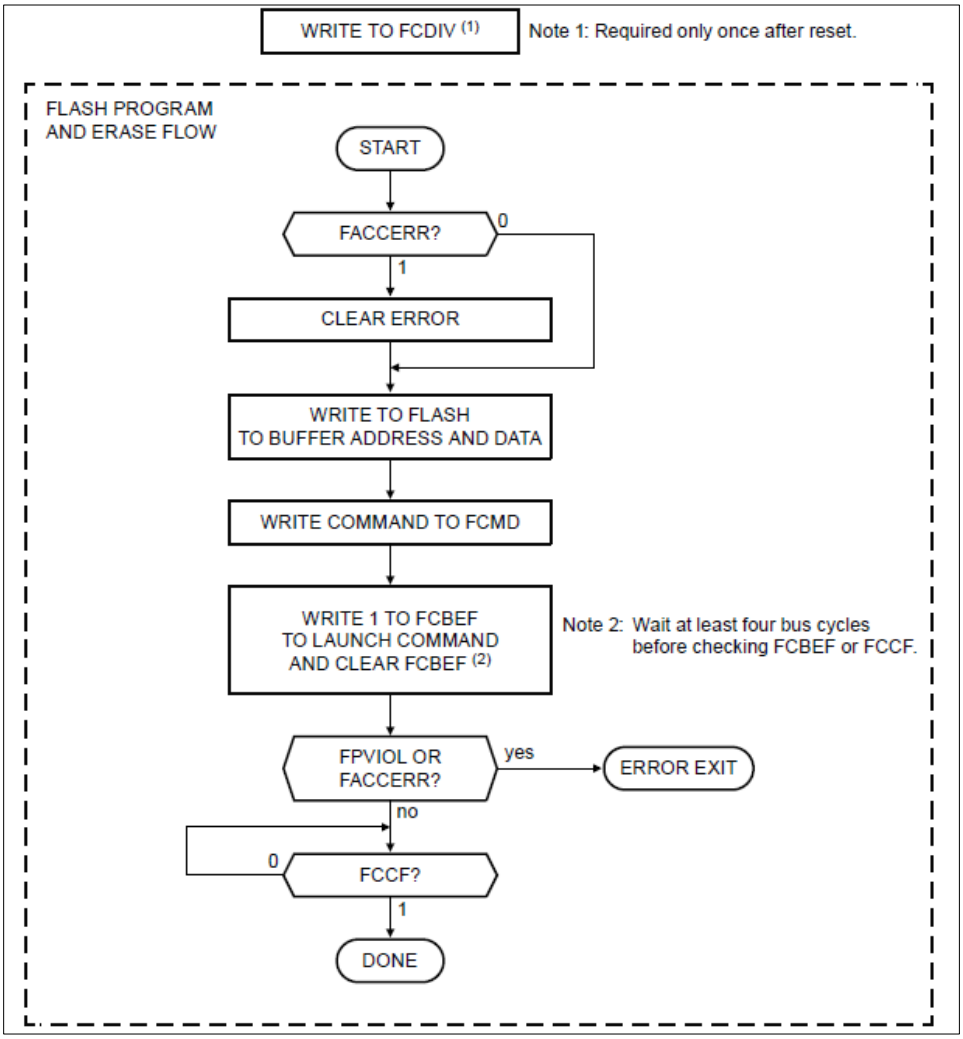

Figure 3: FMC program and erase command flow diagram extracted from the NTM88 User Manual

ii. Configuring FCDIV

Before FLASH write and erase commands can be performed, the FCDIV register at address $1820 must be configured so that the FLASH clock has a frequency between 150kHz and 200kHz. The FLASH clock frequency is derived from the BUSCLK so the divider value to write depends on the BUSCLK frequency configured.

To configure FCDIV, the External MCU must first read the BUSCLK value configured in SIMOPT2 register, and then write FCDIV with the appropriate value as listed in the table below.

Table 3: Correspondence between BUSCLK frequency and FCDIV value

BUSCLK frequency configured in SIMOPT2	4MHz	2MHz	1MHz	0.5MHz
Value to write in FCDIV	19	9	4	2

Note that the FCDIV register can be written only once after reset (reset source includes exit from STOP1), so this configuration is necessary only once after NTM88 reset.

iii. Program and Erase operations

Once FCDIV has been configured, the flow to perform single byte program or page erase is the following:

- Read FSTAT register to check the value of FACCERR and FPVIOL flags. If one of the bits, or both, are not '0' then write '1' to them in order to clear them.
- Then:
  - In case of a byte programming action, write the desired value at the desired address in FLASH.
  - In case of a page erase command, write a dummy value at an address inside the FLASH page to be erased.

    Note that prior to sending the write command, the SPIOPS_FLS_ADDR1/0 fields must be configured with the appropriate value in order to access the desired FLASH address range.
- Write FCMD register with the appropriate value: 0x20 for byte programming or 0x40 for page erase.
- Trigger the write or erase command by writing '1' to FCBEF in FSTAT register.
- Wait for a duration of at least four BUSCLK cycles.
- Read FSTAT register:
  - If FACCERR or FPVIOL is set, it means an error occurred during the flash write or erase operation. This can be due to a previous read or write command not successfully performed because of an error during SPI transfers. Make sure to always check the status of the SPI transfers and perform a re-try in case the command was not successfully executed.
  - If both bits are cleared then no error occurred. It is necessary to wait for the completion of the command: read FSTAT register until bit FCCF of FSTAT register is set.

iv. Illustration with scope screenshots

The scope screenshots below show the sequence of read and write commands implementing byte programming or page erase. The SPI CS pin is monitored.

Byte programming sequence:

The different transfers leading to the byte programming are shown below. The transfers highlighted with the red square are detailed in the next figure.

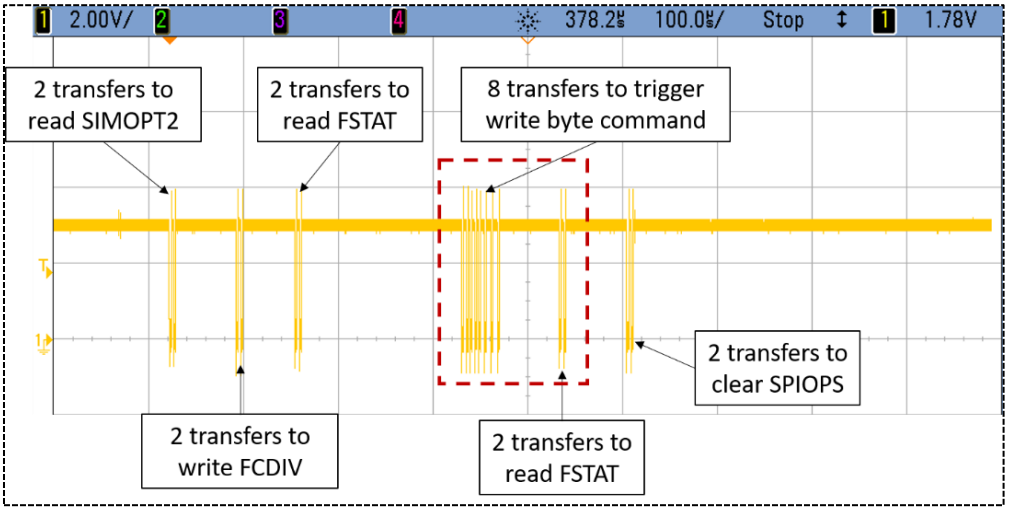

Figure 4: Overview of the SPI transfers leading to byte write

The first four transfers to read the BUSCLK configuration in SIMOPT2 and write FCDIV register are to be performed only once after NTM88 reset, and not every time a write or erase command is performed. Also, the read to SIMOPT2 register is to be done only if the External MCU does not know the NTM88 BUSCLK configuration.
The transfers to clear SPIOPS register are to be performed only at the end of the SPI transfer sequence, after all write and erase actions have been performed.

In the figure below are highlighted the transfers allowing to perform the write byte command.

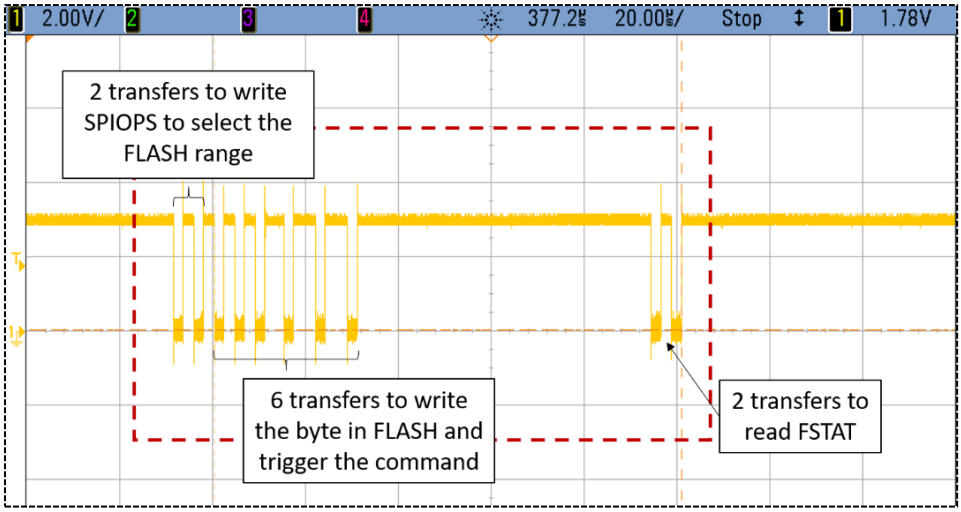

Figure 5: Detail of the transfers performing the byte write

The first two transfers to update the FSL_ADDR1/0 fields are necessary after NTM88 reset and when the flash range needs to be updated. If several bytes are written consecutively in the same flash range, it is not necessary to write these fields every time.

After the write command has been triggered, there must be a delay of minimum 4 BUSCLK cycles before the FSTAT register is accessed again. This makes a duration between 1µs for a BUSCLK frequency of 4MHz and 8µs for a frequency of 0.5MHz.

In the sequence illustrated here, when the FSTAT register is read, there are no error flags raised and the FCCF flag, indicating command completion, is already set. Since the byte programming action takes around 45µs to complete, it is expected to see the FCCF flag set since it is read 60µs after the command was triggered.

If additional bytes were to be written in the FLASH memory, the External MCU could start the next byte write command at this moment, after reading FSTAT register, given the FCCF flag is set and no error flag are raised. If error flags are raised, it is necessary to clear them by writing 1 to them.

Note on the byte burst programming:

The NTM88 offers the possibility to perform burst programming, allowing to program consecutive bytes faster than using the single write command. To perform burst programming, after a write byte command has been triggered and delay of minimum 4 BUSCLK cycles performed, FSTAT must be read and the next write command triggered before the current command completes. This implies that the External MCU must be capable of reading FSTAT and queueing the next write byte command in less than 20µs.

Table 4: Extract from the User Manual showing the program and erase times

Table 183. Program and erase times

Parameter	Cycles of FCLK	Time[1] assuming FCLK = 200 kHz
Byte program	9	45 µs
Byte program (burst)	4	20 µs
Page erase	4000	20 ms
Mass erase	20,000	100 ms

Page erase sequence:

The figure below gives an overview of the page erase sequence. Transfers triggering the page erase command are first performed, followed by a 20ms delay. The page erase command takes around 20ms to execute, so the 20ms delay is here to let time to the command to complete before checking the FCCF flag and error flags. It would also have been possible to keep reading the FSTAT register waiting for the FCCF flag or error flag to be raised, and in that case the External MCU would have been looping reading the FSTAT register during 20ms.

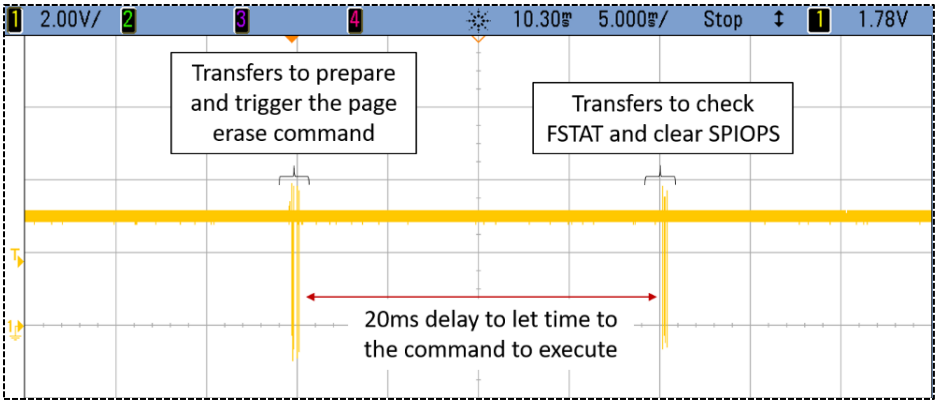

Figure 6: Overview of the SPI transfers leading to page erase

c. Preserving the trim coefficients

The trim coefficients are programmed by NXP between $FD40 and $FDFF. They are necessary as they allow sensor measurements and compensation to return valid values. They are unique to each chip so they cannot be copied from one chip to another.

Writing the FLASH memory is done byte by byte, but erasing it is done page by page. In the NTM88, a page is 512-byte long and the full FLASH memory contains 16 pages. The trim coefficients are located in the page that starts at address $FC00 and ends at address $FDFF.

In the FMC module there are two options to erase the FLASH memory: either page by page, which takes 20ms per page, or via a mass erase during which the 16 pages are erased at the same time, which takes 100ms. When FLASH memory is erased, it must be ensured that the trim coefficients are never lost:

- Either by erasing the memory page by page and avoiding the trim page. This way the trim coefficients are never erased.
- Or by reading and storing the trim coefficients in the External MCU memory before performing a mass erase. The trim coefficients must then be written again in the NTM88 memory.

Legal disclaimer

All information hereunder is per NXP's best knowledge. This document does not provide for any representation or warranty express or implied by NXP. NXP makes no representation or warranty that customer's applications or design will be suitable for customers' specified use without further testing or modification. Customers are responsible for the design and operation of their applications and products using NXP products, and NXP accepts no liability for any assistance with applications or customer product design. Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

For reliable information on the NXP product please consult the respective NXP data sheet.

The information given hereunder is non-binding and preliminary and provided without legal commitment whatsoever. The information may be subject to changes and amendments. As with any project, inherent uncertainties can lead to the termination or delay of the project at any time. NXP does not accept any liability with regard to the project description given hereunder nor to any project realization. Any project commitment is subject to conclusion of a separate duly signed contract.

The dates provided herein are non-binding and preliminary and provided without legal commitment whatsoever. The timeline, and the assumptions underlying that timeline, are subject to change at any time. NXP does not accept any liability with regard to the dates provided. Any dates or other information provided by NXP are binding only upon conclusion of a written contract signed by customer and NXP.