AN11210

Using SGPIO to emulate an SPI master interface

Rev. 1 — 1 June 2012 Application note

Document information

Info	Content
Keywords	SGPIO, SPI , Master, emulation
Abstract	This application note describes an example of SPI master interface emulation using SGPIO

Revision history

Rev	Date	Description
1	20120601	First release

Contact information

For more information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

1. Introduction

SGPIO (serial general purpose I/O) is a unique peripheral which can be used to emulate communication protocols using either serial or parallel transmission lines.

The whole peripheral concept is constructed on top of a set of building blocks called “slices”.

A slice is essentially a 32-bit wide shift register, which is used to hold the data to be received or transmitted. There are 16 slices within one SGPIO block.

Slices can be used separately (even in self-loop) or concatenated together when handling data bit streams longer than 32.

Additionally, each slice has several clock options to choose from (external pin, local slice counter, or internal slice).

The user also has additional control over the capture (shift) clock signal, which can be “qualified” (gated) by either the level of another slice output, or by an external pin.

Moreover, slices can also be configured to function as an “output enable” signal for other slices, which might be used in case a specific connection line needs to be bidirectional or shared with more than one external device (so that the associated SGPIO pin is not constantly being driven, but can be tri-stated).

Slices can also be programmed to raise an interrupt when the amount of programmed bits has been transmitted, when a bit has been captured in input, or when a certain bit pattern has been received. There are also internal connections available to other on chip blocks (timers, adc, sct, dma).

The information above is just a quick overview of the block’s capabilities, for more details the user manual is the most complete source of information.

The important thing to realize while studying the SGPIO peripheral is that there are restrictions in terms of possible configuration options, since there are several levels of multiplexing at the input of a slice, at the output of a slice, and within a slice itself.

In other words, the user has to be aware that not all SGPIO pins will be available to realize a specific configuration, so in the process of choosing a specific slice for performing a function, the user has to cross check the peripheral configuration for feasibility.

This application note will show how to implement an SPI emulation block by defining the system level requirements and applying them to the SGPIO configuration.

2. System level requirements

An SPI interface is essentially a full duplex serial communication line, which is a point to point connection between two devices.

One of the devices has the master role, and drives the interface by sending data to the slave. The other device has a slave role, and responds to the master activities by providing data to the master.

The physical level interface is made of the following signals:

- Chip select (also called slave select): active low signal defines when a transaction is in progress, driven by the master of the interface
- Clock: provided by the master of the interface
- MOSI (master output / slave input): data line driven by the master of the interface
- MISO (master input / slave output): data line driven by the slave of the interface

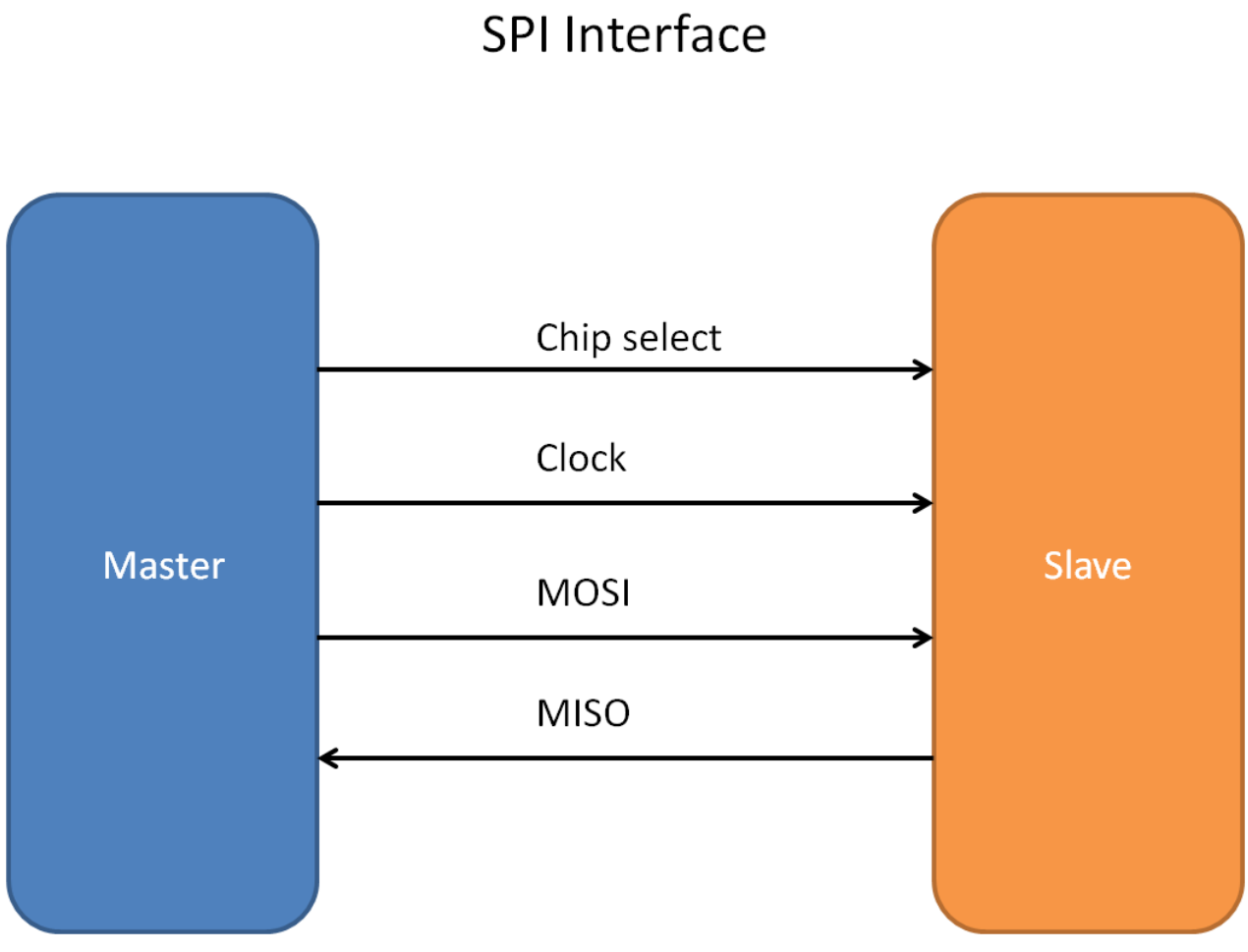

Fig 1. SPI Interface

There is no official standard on SPI, however, all existing implementations in industry consistently specify two configuration parameters which define the behavior of the clock signal and the shifting / sampling operations.

The first parameter is often referenced as “clock phase” (or CPHA), and specifies two different transmission formats.

The second parameter is the clock polarity, which determines the level of the clock line between transmissions (idle high or active low).

A general purpose SPI interface needs to support all four combinations of these two parameters.

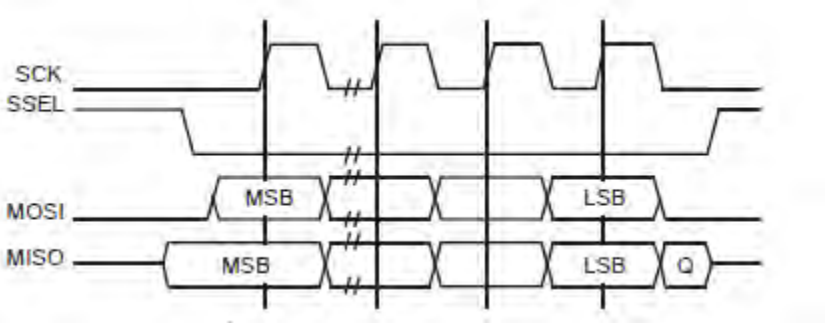

Fig 2. CPOL = 0, CHPA = 0 transmission format

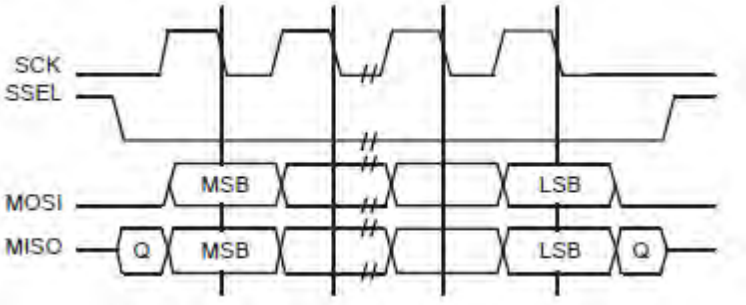

Fig 3. CPOL = 0, CPHA = 1 transmission format

Looking at Fig 2 and Fig 3, the CPHA parameter determines essentially the data sampling convention.

- When CPHA = 0, the first edge of the clock signal is used for sampling the data
- When CPHA = 1, the second edge of the clock signal is used to sample the data

In both figures above, the clock polarity is low, which means the clock signal stays at level low when idle.

When CPOL = 1, the clock is active high, but the sampling convention stays the same. The only difference is that the first edge of the clock signal is a falling edge.

CPHA = 1 is a format which is sometimes required by certain slave devices which use the first clock edge to load the new data item within its shift register.

In terms of timings, most SPI interfaces typically require a setup time (an idle time) of half SPI clock period between the falling edge of the slave select signal and the first edge of the SPI clock. Similarly, a hold time of half SPI clock is required between the last SPI clock edge and the rising edge of the slave select signal (when a transaction is complete).

The amount of bits being transmitted within each SPI frame is typically limited, so that the slave select returns to high state (is de-asserted) after one transmitted data item (one item being typically up to 16 or 32 bits).

For performance reasons, some devices (like SPI flash memories) can support a transmission format not requiring the slave select to return idle between each word being transmitted. The master device de-asserts the slave select signal at the beginning of the transmission, and keeps clocking the interface transferring several consecutive data bits (multiple words) per transaction. The slave keeps providing data on the bus (in case of memories this data is related to an address location which is auto-incremented) as long as the master generates the clock, until the slave select signal is de-asserted again.

This mode is not directly supported by this SPI emulation configuration. However, it can be simulated by using a GPIO line to generate the slave select. This has to be set low at the beginning of a multi-item transaction, and driven high again after the desired number of data items has been exchanged on the bus.

3. Implementation details

3.1 Peripheral configuration

This SPI emulation example implementation is able to support:

- Data size per transfer of 4,8,10,12,14,16, 18, 20, 22, 24, 26, 28, 30 or 32 bits
- All of the CPHA and CPOL transmission formats
- Configurable SPI clock frequency
- Up to two emulated independent SPI interfaces

The parameter of the configuration can be specified in a structure or type *SpiParam.*

This structure is a parameter of the *SGPIO_spiOpen* function call which is used to configure the emulated interface.

The slices used for the SPI emulation are configured to work in a start / stop fashion, so the emulated SPI transmits only one item at the time.

A transmission can be started by calling the function *SGPIO_spiWrite()*.

After transmitting the required amount of bits (one data word), the chip select returns high and all the slices stop automatically.

The slice used to generate the chip select signal is configured to trigger an interrupt to the CPU, which is used to determine when one data item has been transmitted (and received).

Until then, the SPI interface is considered busy, and the application queries the status by calling the function *isSpiReady().* When true, the data transmission is complete. The memory location defined while calling *SGPIO_spiWrite()* holds the received data word.

At this point, the application can issue another write command, which program the next item to be transmitted, and restart the transmission.

More details on the available APIs are described in section 7.

The assignment of the slices related to the data transmit and receive is done by the user within the file sgpio_spi.c, in the structures defined within the “user slice assignment section”.

In case of changes, the pin multiplexing for the SGPIO pins on the LPC4300 needs to be adapted as well. The assignment is done within the function SGPIO_spiOpen in sgpio_spi.c module.

USB_0_IND1 green led on the board shall blink during a successful loopback test.

4. Hardware configuration

This example has been tested on a Keil MCB4300 board, using a Ulink2 debug probe.

No specific modifications to the board are required.

Since the interface is tested in loopback mode, the MOSI output needs to be connected externally to the MISO input via an external wire.

All required SGPIO pins can be found on the MCB4300 prototype connectors.

4.1 Slave select

For generating the slave select signal, two approaches are possible:

- Using a GPIO signal
- Using a slice to output a pattern to generate the slave select

The first method is the most simple, although it is difficult to guarantee a precise timing relationship between the chip select and the clock.

Unless a hardware-based timer delay is used, the time between the toggling of the slave select line and the start of the transmission of the clock and the data by the SGPIO will not be synchronized.

The overhead and complexity of introducing a sufficiently long delay to ensure the chip select setup and hold times are respected might not be negligible when performing individual transfers.

Because of this, the implementation uses a simple bit pattern to generate the slave select signal.

The bit pattern consists of a logic level 1, followed by a number of zeros equal to the number of transmitted data bits, followed by a final logic one.

This implies that to support 32 bit transfers, a total of 34 bits needs to be used, which is only possible by concatenating two slices together. The second slice will be used only internally, and will not be provided as a physical output.

Additionally, since the pattern being used has a size which is not a multiple or an integer divisor of 32 bit, it will need to be reloaded after each transmission.

4.2 Clock

The clock signal can be generated by shifting out a simple pattern based on repeating sequences of zeros and ones. The slices can be configured in self loop mode, meaning the bit being shifted out for the least significant bit (bit 0) is looped back-in on the most significant bit (bit 31).

To generate an active low or active high clock, the patterns 0x55555555 and 0xAAAAAAAA are used.

To avoid the shadow register and the data register being swapped after the amount of clock pulses being transmitted, the slice is configured in match mode. Although the bit pattern recognition feature is not used, this prevents the two registers from being swapped when the bit count (the POS counter) reaches zero.

This slice can be configured at the beginning and does not need to be reloaded again at runtime, since it is connected in self loop.

4.3 Data transmission

For transmitting the data, one dedicated slice is chosen among those being left over from the rest of the configuration.

The same slice cannot be used simultaneously for receiving data, since the capture clock and the shift clock for the slice are related to the same edge, whereas in the SPI protocol the data is shifted out on the opposite clock edge of a data capture.

The slice is configured to use the clock signal generated with the slice mentioned in section 4.2, provided via an external pin.

4.4 Data reception

For transmitting the data, one dedicated slice is chosen among those remaining from the rest of the configuration.

The same slice cannot be used simultaneously for sending data, since the capture clock and the shift clock for the slice are related to the same edge, whereas in the SPI protocol the data is shifted out on the opposite clock edge of a data capture.

The SPI interface allows receiving data while a transmission is in progress (full-duplex). In order to receive an item, something needs to be transmitted (it can be also a dummy word, but this is application dependant). The *SGPIO_spiWrite()* function call allows the application to pass a pointer to a memory location where the received data item will be stored.

4.5 CHPA = 1 support

In CHPA mode 1 the LSB on the MOSI line should be valid (shifted out) on the first clock edge.

However, in the SGPIO peripheral, the LSB bit of the slice is immediately applied to the output when the data register is written, before the first edge.

This implies that by default at the first edge the second LSB would be already shifted.

For this reason, it is necessary to gate (qualify) the clock signal by making sure that the first clock edge is ignored by the slice, so that the second LSB gets shifted out only on the third edge as required (see Fig 3).

The idea is to re-use the chip select slice (slice A) to act as a clock qualifier for the transmitting and receiving slice. In this way it is possible to introduce an offset within the slice shift clock, so that it is possible to skip the first edge which on the SGPIO master side needs to be ignored

5. Resource allocation

Table 1 shows the allocation of resources of the SGPIO peripheral in relation to the MCB4300 board.

Table 1. Allocation of SGPIO resources on the MCB4300 board

Device port	SGPIO output pin	Function	Slice
P9_0	SGPIO_0	Slave select 0	A
	SGPIO_1	Internal - Slave select 0	I
P2_3	SGPIO_12	Clock 0	D
P9_2	SGPIO_2	MOSI 0	E
P7_2	SGPIO_6	MISO 0	F
	SGPIO_14	Internal - Slave select 1	H
P4_10	SGPIO_15	Slave select 1	P
PC_14	SGPIO_13	Clock 1	O
PF_9	SGPIO_3	MOSI 1	J
PF_6	SGPIO_5	MISO 1	K

Table 2 shows the list of used and available slices and input / output pins from the slice perspective:

Table 2. SGPIO IP resources usage example (1 bit mode)

SGPIO input pin	Input	Slice	Output	SGPIO output pin
SGPIO_0		A	MASTER SS 0	SGPIO_0
SGPIO_8	SLAVE 0 SS	B		SGPIO_8
SGPIO_4		C		SGPIO_4
SGPIO_12		D	MASTER CLK 0	SGPIO_12
SGPIO_2		E	MASTER 0 MOSI	SGPIO_2
SGPIO_6	MASTER 0 MISO	F		SGPIO_6
SGPIO_10	SLAVE 1 SS	G		SGPIO_10
SGPIO_14		H	MASTER SS 1 Internal	SGPIO_14
SGPIO_1		I	MASTER SS 0 Internal	SGPIO_1
SGPIO_3		J	MASTER 1 MOSI	SGPIO_3
SGPIO_5	MASTER 1 MISO	K		SGPIO_5
SGPIO_7		L		SGPIO_7
SGPIO_9	SLAVE 0 CLK	M		SGPIO_9
SGPIO_11	SLAVE 1 CLK	N		SGPIO_11
SGPIO_13		O	MASTER CLK 1	SGPIO_13
SGPIO_15		P	MASTER SS 1	SGPIO_15

The rows marked in green and orange relate to slices that are reserved for the configuration of two SPI master interfaces.

Any of the free slices might be picked and assigned to transmit or receive data.

Note: within the peripheral, SGPIO8, SGPIO9, SGPIO10, SGPIO11 are pins capable of inputting a signal to be used as a clock or qualifier; these should preferably not be used to input / output data, in order to leave this option free if needed.

For example, the rows marked in blue could be used to implement the Slave Select and Clock signal for an SPI slave interface.

The following table shows the example allocation of resources in relation to the internal slice chaining, to highlight the implication of the slice selection vs. the slice concatenation options.

Table 3. SGPIO IP resources from slice chaining perspective (1 bit mode)

SGPIO slice	Function	SGPIO slice	Function
A	MASTER0 SS	B	SLAVE0 SS
I	MASTER0_SS	M	SLAVE0 CLK
E	MASTER0 MOSI	G	SLAVE1 SS
J	MASTER1 MOSI	N	SLAVE1 CLK
C		D	MASTER0 CLK
K	MASTER1 MISO	O	MASTER1 CLK
F	MASTER0 MISO	H	MASTER1 SS
L		P	MASTER1 SS

The items in yellow will need to be reserved to the SPI master zero interface, whereas the ones in green are reserved to the SPI master one interface.

As previously mentioned, four SGPIO pins can be used to input an external clock to the slices, or as a clock qualifier signal.

Each implemented slave interface would require one of those SGPIO pins to be used as a clock source, and one to input the slave select signal (used as a clock qualifier for the input clock).

This implies there can be a maximum of two slave interfaces emulated on one SGPIO block (the pins which will need to be reserved for the slave interface are highlighted in blue).

Any other unreserved slice can be assigned to the master or slave interfaces in order to receive and transmit data.

6. Project configuration

The default setting for the SPI emulation interface assumes an SGPIO clock of 12 MHz (note: this is related to the SGPIO IP internal clock, not the emulated SPI clock).

This is defined within the *sgpio.h* header file. The user has to modify this definition and recompile the code in case SGPIO gets clocked by a faster or slower clock source.

Configuration of slices A, I, D & O, H, P shall not be modified by the user.

By default slices E, F and J, K are used for master 0 and master 1 data transmission / reception. This might be assigned differently by the user.

The assignment of the slices related to the data transmit and receive is done by the user within the file *sgpio_spi.c*, in the structures defined within the “user slice assignment section”.

In case of changes, the pin multiplexing for the SGPIO pins on the LPC4300 needs to be adapted as well. The assignment is done within the function SGPIO_spiOpen in sgpio_spi.c module.

USB_0_IND1 green led on the board shall blink during a successful loopback test.

7. API interface

The following is the list of available functions to control the emulated SPI

Table 4. SGPIO emulation API list

Function	Return type	Parameter list
SGPIO_spiInit	void	Void
SGPIO_spiOpen	void	spiNum_t spiId, spiParam const *config
SGPIO_spiWrite	void	spiNum_t spiId, uint32_t data, uint32_t* dataRead
SGPIO_spiClose	void	spiNum_t spiId
isSpiReady	bool	spiNum_t spiId
SGPIO_spiEmuCheckErrors	spiEmuStatus	void

7.1 SGPIO_spiInit

This function initializes the SGPIO peripheral, resets the block and zeros out all associated software control variables.

7.2 SGPIO_spiOpen

This function opens one emulated SPI interface for communication.

The parameters are the number of the SPI interface (SPI_EMU_0 or SPI_EMU_1) and a pointer to a structure of type spiParam which defines the interface characteristics.

```
typedef struct SpiParam {
  mode_t     spiMode;
  wordSize_t   wordLength;
  clockPhase_t clockPhase;
```

```
    clockPolarity_t clockPolarity;
    bitRate_t    bitRateHz;
    CsPattern    chipSelect;

} SpiParam;
```

The parameters are the following:

- spiMode: can be set to MASTER or SLAVE (only master supported at the moment)
- wordLength: can be configured from DBIT_4 to DBIT_32, see sgpio_spi.h
- clockPhase: CPHA_0 or CPHA_1
- clockPolarity: IDLE_LOW or IDLE_HIGH
- chipSelect: always set this to the macro CHIP_SELECT_PATTERN(N) where N is the wordLength defined above

7.3 SGPIO_spiClose

Used to close one of the interfaces, and reset the register to the values.

The required parameter is the SPI identifier, SPI_EMU_0 or SPI_EMU_1

7.4 SGPIO_spiWrite

This function is called to write a data item to the SPI interface.

The parameters specify:

- the SPI identifier (SPI_EMU_0 or SPI_EMU_1)
- the data item to be written
- the pointer to the location where the data being read is going to be stored.

This function returns immediately and is not blocking.

7.4.1 isSpiReady

This function can be used to query the emulated peripheral, to check if a transaction has been performed.

The parameter is the SPI identifier (SPI_EMU_0 or SPI_EMU_1), and returns the value TRUE or FALSE.

The return value will be set true when an ongoing transaction is terminated, and the emulated SPI is ready to initiate a new transmission

7.4.2 SGPIO_spiEmuCheckErrors

This function can be used during development to check if there are any configuration errors in terms of parameters.

Any error will set a global variable of type spiEmuStatus, which can be retrieved with this function.

For a static configuration, this function can be removed from a production system.

8. Legal information

8.1 Definitions

Draft — The document is a draft version only. The content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included herein and shall have no liability for the consequences of use of such information.

8.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Evaluation products — This product is provided on an "as is" and "with all faults" basis for evaluation purposes only. NXP Semiconductors, its affiliates and their suppliers expressly disclaim all warranties, whether express, implied or statutory, including but not limited to the implied warranties of non-infringement, merchantability and fitness for a particular purpose. The entire risk as to the quality, or arising out of the use or performance, of this product remains with customer.

In no event shall NXP Semiconductors, its affiliates or their suppliers be liable to customer for any special, indirect, consequential, punitive or incidental damages (including without limitation damages for loss of business, business interruption, loss of use, loss of data or information, and the like) arising out the use of or inability to use the product, whether or not based on tort (including negligence), strict liability, breach of contract, breach of warranty or any other theory, even if advised of the possibility of such damages.

Notwithstanding any damages that customer might incur for any reason whatsoever (including without limitation, all damages referenced above and all direct or general damages), the entire liability of NXP Semiconductors, its affiliates and their suppliers and customer's exclusive remedy for all of the foregoing shall be limited to actual damages incurred by customer based on reasonable reliance up to the greater of the amount actually paid by customer for the product or five dollars (US$5.00). The foregoing limitations, exclusions and disclaimers shall apply to the maximum extent permitted by applicable law, even if any remedy fails of its essential purpose.

8.3 Trademarks

Notice: All referenced brands, product names, service names and trademarks are property of their respective owners.

9. List of figures

10. List of tables

11. Contents

For more information, visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 1 June 2012
Document identifier: AN11210