GTM-IP

Application Note AN016
GTM SPI application

Date: 11.02.2013

Robert Bosch GmbH
Automotive Electronics (AE)

LEGAL NOTICE

The content of this document is subject to continuous developments and improvements. All particulars and its use contained in this document are given by BOSCH in good faith.

NO WARRANTIES: TO THE MAXIMUM EXTENT PERMITTED BY LAW, NEITHER THE INTELLECTUAL PROPERTY OWNERS, COPYRIGHT HOLDERS AND CONTRIBUTORS, NOR ANY PERSON, EITHER EXPRESSLY OR IMPLICITLY, WARRANTS ANY ASPECT OF THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO, INCLUDING ANY OUTPUT OR RESULTS OF THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO UNLESS AGREED TO IN WRITING. THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO IS BEING PROVIDED "AS IS", WITHOUT ANY WARRANTY OF ANY TYPE OR NATURE, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, AND ANY WARRANTY THAT THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO IS FREE FROM DEFECTS.

ASSUMPTION OF RISK: THE RISK OF ANY AND ALL LOSS, DAMAGE, OR UNSATISFACTORY PERFORMANCE OF THIS SPECIFICATION (RESPECTIVELY THE PRODUCTS MAKING USE OF IT IN PART OR AS A WHOLE), SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO RESTS WITH YOU AS THE USER. TO THE MAXIMUM EXTENT PERMITTED BY LAW, NEITHER THE INTELLECTUAL PROPERTY OWNERS, COPYRIGHT HOLDERS AND CONTRIBUTORS, NOR ANY PERSON EITHER EXPRESSLY OR IMPLICITLY, MAKES ANY REPRESENTATION OR WARRANTY REGARDING THE APPROPRIATENESS OF THE USE, OUTPUT, OR RESULTS OF THE USE OF THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO IN TERMS OF ITS CORRECTNESS, ACCURACY, RELIABILITY, BEING CURRENT OR OTHERWISE. NOR DO THEY HAVE ANY OBLIGATION TO CORRECT ERRORS, MAKE CHANGES, SUPPORT THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO, DISTRIBUTE UPDATES, OR PROVIDE NOTIFICATION OF ANY ERROR OR DEFECT, KNOWN OR UNKNOWN. IF YOU RELY UPON THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO, YOU DO SO AT YOUR OWN RISK, AND YOU ASSUME THE RESPONSIBILITY FOR THE RESULTS. SHOULD THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL LOSSES, INCLUDING, BUT NOT LIMITED TO, ANY NECESSARY SERVICING, REPAIR OR CORRECTION OF ANY PROPERTY INVOLVED TO THE MAXIMUM EXTEND PERMITTED BY LAW.

Revision History

Issue	Date	Remark
0.1	11.2.2013	Initial version

Tracking of major changes

Changes between revision 1.x and 1.y

NA

Conventions

The following conventions are used within this document.

ARIAL BOLD CAPITALS	Names of signals
Arial bold	Names of files and directories
`Courier bold`	Command line entries
`Courier`	Extracts of files

References

This document refers to the following documents.

Ref	Authors(s)	Title
1	AE/EIN2	GTM-IP Specification

Terms and Abbreviations

This document uses the following terms and abbreviations.

Term	Meaning
GTM	Generic Timer Module

Table of Contents

1 Overview 1
1.1 Use case 1
1.2 System architecture 2
2 Submodule setup 4
2.1 CMU and TBU setup 4
2.2 SPI transceiver 5
2.3 SPI receiver 8
3 Test Environment 11
3.1 Error Codes 12

1 Overview

This application note provides an example about a GTM related implementation of a simple Serial Peripheral Interface (SPI) transceiver (TX) and SPI receiver (RX) module. The example may be used as a starting point for developing more complex protocol transceivers or receivers with the GTM.

The SPI transceiver module allocates an MCS channel for transceiver protocol integration and three ATOM channels for output signal generation. The receiver module allocates another MCS channel for receiver protocol integration and three TIM channels for input signal capturing.

1.1 Use case

Figure 1.1 shows the timing diagram of the SPI protocol that is used in this application note. The signal CE is used as a chip enable signal, which indicates a valid data transfer with a low active signal level. The signal CLK provides the serial clock signal for the data transfer. With each rising edge of CLK, the transceiver drives a new data bit on its output signal SDATA. On the other hand, the SPI receiver module samples a new incoming data bit on the signal SDATA with each falling edge of CLK.

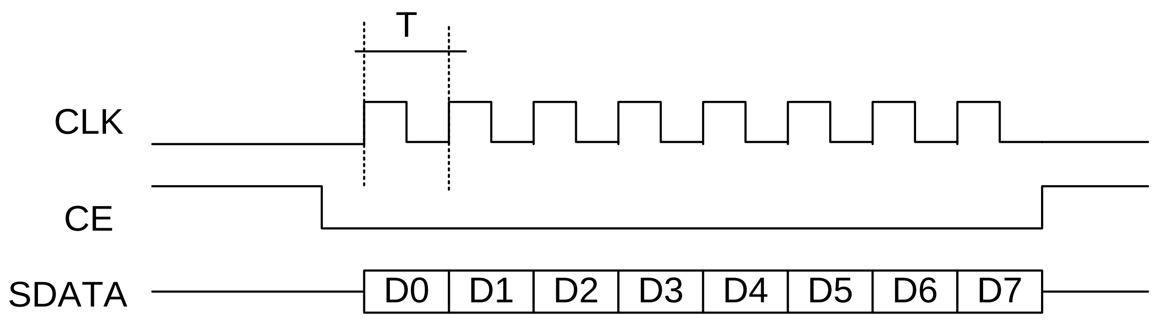

Figure 1.1: Timing diagram of SPI protocol.

The SPI transmitter module presented in this application note allows to send data with a variable bit width W (with W < 24) and a variable bit clock period T with (T > 3·ARU Round trip cycle) for the Transceiver and (T > 1·ARU Round trip cycle but not faster than 21 instruction cycles) for the Receiver. The SPI receiver module can also be configured to accept a desired bit width W (with W < 24). Moreover, the receiver also inspects the bit clock period T of the incoming signal CLK and it will generate an error, when a specified timeout value is expired. If the CE signal is changing to level high during an active transfer, the receiver module will abort and report an error.

1.2 System architecture

For this application note, the GTM is configured as shown in Figure 1.2. The SPI transceiver allocates MCS-channel 0 of an MCS instance MCS[i] and three ATOM channels (channel 0 to 2) of an ATOM instance ATOM[i], whereas channel 0 generates the CLK signal, channel 1 the CE signal and channel 2 the SDATA signal. All three ATOM channels are configured in PWM mode (mode SOMP), which means that the channels are generating PWM signals with varying duty cycles. The receiver module allocates MCS-channel 1 of an MCS instance MCS[i] and three TIM channels 0 to 2 of a TIM instance TIM[i]. The TIM channel 0 is configured in bit compression mode (mode TBCM), whereas the incoming CLK signal is connected to TIM channel 0, the CE signal to TIM channel 1, and the serial input data SDATA to TIM channel 2.

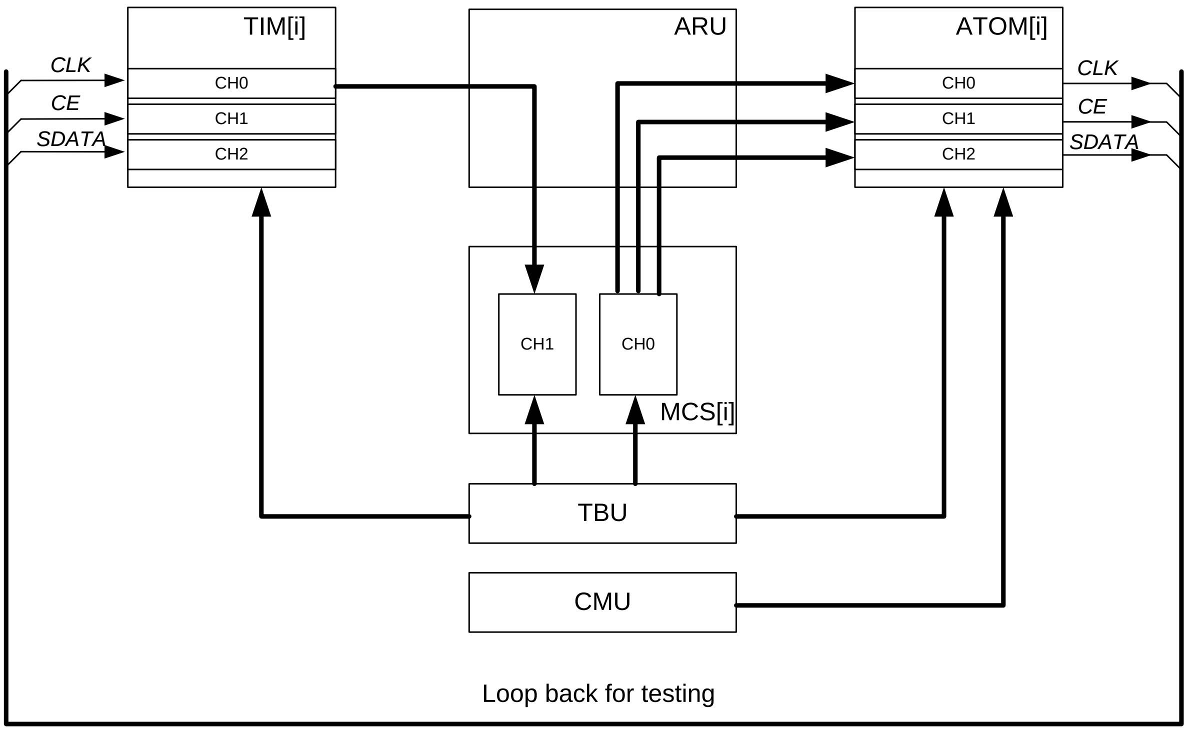

Figure 1.2: Application of the SPI functionality on the GTM device.

In order to test that both, transceiver and receiver modules work properly the outputs of the ATOM module are directly connected to the inputs of the TIM module using the internal loop back of the GTM configured with the registers **GTM_TIM_AUX_IN_SRC** and **TIM_IN_SRC**. The SPI application also makes usage of the modules TBU, providing a common time base to all TIM, ATOM, and MCS modules, and it makes usage of the module CMU that provides a clock signal to the ATOM module.

Further, the application note provides a test bench, in which multiple pairs of SPI transceiver and receiver modules are instantiated and a set of predefined data with varying bit width W and varying bit clock periods T are send out via the SPI transceiver. The corresponding SPI receiver module, verifies that the data of the transceiver arrived correctly and no timeout of the bit clock period occurred.

2 Submodule setup

2.1 CMU and TBU setup

The application note provides by a C-function

```
int spi_common_init(int inst);
```

which initializes the commonly used modules CMU and TBU. If the initialization was successfully, the return value of this function is 0, otherwise occurred error identifier is returned. If more than one error occurs only the last error identifier is returned. The CMU provides a clock signal *CMU_CLK0* that is used by the ATOM module as base clock for PWM generation. The period for this signal is T_{CMU_CLK0} =1 us. Further, the TBU channel 0 is configured to update its time base with the clock signal *CMU_CLK0*. Finally `spi_common_init` connects the internal loop back between the ATOM and the TIM channels of instance `inst`.

2.2 SPI Transceiver

From the software point of view, the SPI transceiver is configured by a C-function

```
int spi_tx_init(int inst);
```

whereas the parameter `inst` is defining the instantiation index for allocated ATOM[i] and the allocated MCS[i]. If the initialization was successfully, the return value of this function is 0, otherwise occurred error identifier is returned. If more than one error occurs only the last error identifier is returned. The function is configuring the three ATOM channels with the PWM mode SOMP in the register **ATOM[i]_CH[x]_CTRL** with x= 0..2, and it is setting up further ATOM parameters in order to signalize an idle state on the outputs, this is implemented by setting default values to the registers **ATOM[i]_CH[x]_SR0** and **ATOM[i]_CH[x]_SR1** with x= 0..2. This means, that the output CE is set to high, and the signals CLK and SDATA are set to low. Moreover, the function `spi_tx_init` establishes the ARU connection between each ATOM channel and the common MCS channel, by writing the first three ARU write addresses of the corresponding instance MCS[i] into its ARU read registers **ATOM[i]_CH[x]_RDADDR** with x= 0..2. Considering the MCS-channel 0 of MCS instance i, the function `spi_tx_init` first loads the micro code for the transceiver into the MCS[i] RAM and additionally enables the MCS channel 0 interrupt. The function is also enabling the three ATOM channels and configuring the ATOM registers **ATOM[i]_AGC_ENDIS_CTRL**, **ATOM[i]_AGC_OUTEN_CTRL**, and **ATOM[i]_AGC_GLB_CTRL**).

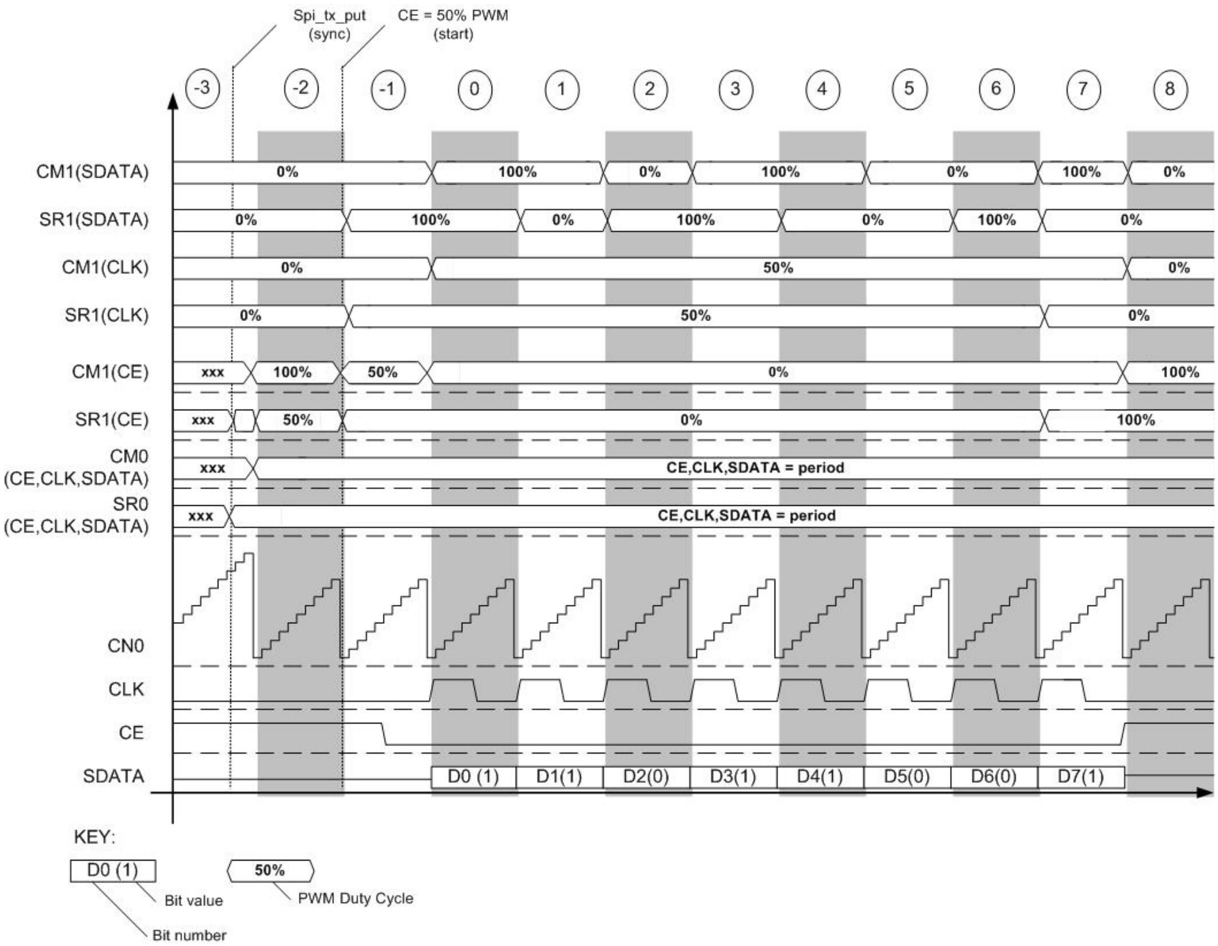

Figure 2.1: Timing diagram with additional ATOM Registers

The SPI transceiver provides another C-function

```
int spi_tx_put(int inst, int data, int width, int period);
```

that initiates sending of new data on the SPI transceiver module. Parameter `inst` denotes the instance number of the transceiver module and the parameter `data` holds the value to be sent via SPI. The W lower significant bits of parameter `data` are sent out via SPI. A data width parameter `width` (equals W in the description above) and a bit clock period `period` are configured for the SPI receiver module.
The bit clock period is specified as integer multiples of the CMU period T_{CMU_CLK0} .If the data sending was initialized successfully, the return value of this function is 0, otherwise occurred error identifier is returned. If more than one error occurs only the last error identifier is returned.

If the function is called during an active data transmission, the function will terminate with an error and no changes the actual configuration are applied. The implementation of the function `spi_tx_put` tells the MCS micro program for the transceiver the parameters `data, period` and `width` by writing them to variables allocated in the MCS memory. Furthermore it triggers the MCS micro program to start with a new data transmission, by setting the program counter to the start address of the MCS micro code and enables the current MCS-channel (0). Moreover the function configures the update mechanism of the ATOM channels by writing the registers **ATOM[i]_AGC_GLB_CTRL**. In order to force an immediate update of the ATOM parameters the register **ATOM[i]_AGC_FUPD_CTRL** also has to be configured. To synchronize the ATOM channels to each other and to the MCS micro program, the function spi_tx_put schedules the beginning of the PWM generation in the ATOM channels by writing a TBU related start time to the **ATOM[i]_AGC_ACT_TB** register and it tells the MCS the same start time plus one micro second by writing this start time to another variable allocated in the MCS RAM, marked in Figure 2.1 as spi_tx_put (sync). The additional micro second is responsible to start with the MCS code at a counter value of 0 by the ATOM counter CN0, marked in Figure 2.1 as (start). The MCS micro program first synchronizes to the ATOM channels, by waiting to the scheduled start time using the WURM instruction. After that, the MCS micro program tells the ATOM channel 1 to drive up a 50 percent PWM for one ATOM period at the output CE and afterwards drive up a low on output CE continuously by setting up a 0 percent PWM using the AWRI instruction, shown in cycles (-1) and (0) of Figure 2.1. After a half period, the MCS tells the ATOM channel 0 to generate an active CLK signal by setting up a 50 percent PWM using the AWRI instruction. The serial data bits of SDATA are generated within a loop by writing a 100 percent or 0 percent PWM value to ATOM channel 2 using the AWRI instruction again, shown in cycles (0) to (7) Figure 2.1. After all data bits are sent out, the MCS code is setting up the ATOM channels to drive output values for an idle state and it rises an interrupt signalizing that the transmission is finished and finally the MCS code disables the MCS-channel, as shown in cycle (8) of Figure 2.1.

In addition, Figure 2.1 shows the shadow and operating registers for each ATOM channel. The AWRI instruction of the MCS writes the duty cycle values (0%, 50% or 100%) to the associated shadow registers and in the next ATOM period the content of the shadow register is moved to the operating register. This process is shown several times in the timing diagram for example in cycles (1) and (2) for the SDATA signal.

In order to abort active SPI transfers the SPI transceiver provides a function

```
int spi_tx_abort(int inst);
```

which is aborting any active transfer and setting the associated ATOM channels to an idle state. The Information about the abort is transferred via the RAM to the MCS micro code. If the aborting was successfully, the return value of this function is 0 and the MCS channel 0 of instance `inst` is disabled, otherwise the occurred error identifier is returned. If more than one error occurs only the last error identifier is returned. If the SPI transceiver is disabled or an abort is already active, the function `spi_tx_abort` returns an error.

2.3 SPI Receiver

The SPI transceiver is configured by a C-function

```
int spi_rx_init(int inst);
```

whereas the parameter `inst` is defining the instantiation index for the allocated module MCS[i]. If the initialization was successfully, the return value of this function is 0, otherwise occurred error identifier is returned. If more than one error occurs only the last error identifier is returned. The function `spi_rx_init` first loads the micro code for the receiver into the MCS RAM, and enables the MCS channel interrupt.

The SPI receiver provides another C-function

```
int spi_rx_configure(int inst, int width, int timeout);
```

The parameter `inst` identifies the instance of the used receiver module. The return value of this function is 0, otherwise occurred error identifier is returned. If more than one error occurs only the last error identifier is returned. If the function is called during an active data transmission, the function will terminate with an error and no changes the actual configuration are applied. The underlying implementation of this function is telling the parameters `width` and `TIM_WRADDR`(ARU write address of the corresponding TIM channel that is used for indirect ARU read accesses to obtain the sampled input data within the MCS-channel) to the MCS-micro program by writing these variables to the commonly used MCS memory. The parameter timeout is used to configure the Timeout Detection Unit (TDU) of the TIM sub module in the registers **TIM[i]_CH[x]_TDUV** and **TIM[i]_CH[x]_CTRL**. The timeout value is specified as integer multiples of the CMU period T_{CMU_CLK0}.

Function `spi_rx_configure` is configuring the TIM channel 0 with the bit compression mode TBCM in the register **TIM[i]_CH0_CTRL**. By writing a value 0x100 to the register **TIM[i]_CH0_CNTS** the bit compression mode is configured in way that it samples all neighboring channels with each falling edge of input channel 0 (CLK signal) followed by writing the sampled data together with an annoted timestamp to its dedicated ARU write address. The Timeout Detction Unit (TDU) of the TIM is configured to report to the associated MCS channel that a timeout event has occurred.
Considering the MCS-channel 1 of MCS instance i, the MCS micro program is organized in a loop which reads the sampled data from the TIM module and it composes a parallel data word of width `width` from the serial input stream. The receiver module also inspects the ACB bits in order to detect a timeout event. Moreover, it inspects the signal CE, which has to be zero during the whole data transfer. Additionally, the data width of a received serial word must also match the configured data width `width`.

If the SPI receiver detects an error, it puts an error code in a variable and it terminates with an MCS channel interrupt. On the other hand, if the SPI receiver obtained a complete data word without an error it clears the error variable and raises an MCS channel interrupt to signalize that a complete data word is received before the MCS code disables the MCS channel.

The C-function

```
int spi_rx_get_error(int inst);
```

simply returns the state of the error variable that is associated the SPI receiver module of instance `inst`. The function `spi_rx_error` returns one of the following error codes:

- 0 – successful transfer
- 21 – CE is set to high (abort occurred)
- 22 – Input edge overwritten by subsequent edge
- 23 – Timeout detected without valid edge

Moreover, the SPI receiver provides the C-function

```
int spi_rx_get(int inst);
```

which returns the received data of SPI receiver module of instance `inst`. In the case of a successfully transfer, the `width` lower significant bits denote the received data. If an error occurred during transmission, for example the transmission is not completed (error code 25), the returned result is 0 or the associated error. Errors are detected by reading the error variable using function `spi_rx_get_error`.

3 Test Environment

This application note also provides a test environment that is instantiating several pairs SPI transmitter and receiver modules that are working simultaneously. The transceiver and receiver modules are connected via the GTM internal feedback loop as mentioned in Figure 1.2 in order to verify the results. The test environment sending test data with varying bit clock periods T and variable word width W. The test environment is also simulating the Abort mechanisms of the SPI modules. The Test Environment contains the main C-function and two additional C-functions as well as the SPI receiver interrupt function.

The starting point for the test environment is the C function

```
int mcs_spi_app();
```

which initiates the transfers between the SPI transceivers and the SPI receivers with respect to user defined pre-processor parameters.

The C-function

```
int spi_print_error(int error);
```

simply translates error codes to readable error messages.

The Test Environment provides another C-function

```
int spi_fill_test_array();
```

which fills a test array in dependency of user limitations and random parameters. Moreover the function intersperses the abort functionality of the SPI transceiver.

Moreover, the Test Environment provides the C-function for the ISR

```
int mcs_spi_rx_isr(int number);
```

whereas the parameter number is defining the instantiation index for the allocated receiver interrupt. As long as the test array has data vailable, the function `mcs_spi_rx_isr` will initiate new data transmissions.

3.1 Error Codes

Error Code	Meaning	Function
	Common initialisation Errors	
ERROR(1)	Wrong data received	mcs_spi_rx_isr
ERROR(2)	Unknown loopback	mcs_spi_common
ERROR(3)	Unknown loopback	mcs_spi_common
ERROR(4)	Not enough TIM instances	mcs_spi_app
ERROR(5)	Not enough MCS instances	mcs_spi_app
ERROR(6)	Wrong data in register CMU_CLK_EN	mcs_spi_common
ERROR(7)	Wrong data in register TBU_CHEN	mcs_spi_common
ERROR(8)	Wrong data in register TIM_IN_SRC	mcs_spi_common
	Transceiver Errors	
ERROR(10)	Wrong data in register MCS_CH_IRQ_EN	spi_tx_init
ERROR(11)	Wrong data in register ATOM_AGC_GLB_CTRL	spi_tx_put
ERROR(12)	Wrong data in register ATOM_AGC_FUPD_CTRL	spi_tx_put
ERROR(13)	MCS channel 0 already running	spi_tx_init
ERROR(14)	SPI transfer already running	spi_tx_put
ERROR(15)	SPI MCS channel 0 disabled no abort possible	spi_tx_abort
ERROR(16)	Abort already active	spi_tx_abort
	Receiver Errors	
ERROR(21)	MCS-Code: CE note enabled	spi_print_error
ERROR(22)	MCS-Code: Input edge overwritten by subsequent edge	spi_print_error
ERROR(23)	MCS_Code: Timeout detected without valid edge	spi_print_error
ERROR(24)	SPI receiver already enabled	spi_rx_configure
ERROR(25)	SPI transfer not completed	spi_rx_get
ERROR(26)	Wrong data in register MCS_CH_IRQ_EN	spi_rx_configure
ERROR(27)	Wrong data in register TIM_CH_CTRL	spi_rx_configure
	Maximum runtime Errors	
ERROR(30)	Too much interrupt calls	mcs_spi_rx_isr
ERROR(31)	Entering isr after maximum runtime	mcs_spi_rx_isr
	User Parameter Errors	
ERROR(40)	User parameter NUM_ABORTS is incorrect	mcs_spi_app
ERROR(41)	User parameter NUM_OF_MODES is incorrect	mcs_spi_app
ERROR(42)	Mode 1 not implemented	mcs_spi_app