Internal Specification

QSPI/SPI model for Brugal

(v1.0)

Summary:

This document describes the Detail Specification of Brugal QSPI/SPI model.

Reference documents				
No.	**Title name**	**Document number**	**Description**	**Path**
1	Brugal_QEMU_functional_requirements.pdf	Nokia QEMU models requirements for Brugal	Functionalities required by Nokia for the QEMU models	-
2	DWC_ssi_databookWM2035099535.pdf	-	Hardware specification of QSPI	WEBtrans/fed_Brugal_QEMU_IP_specifications.zip
3	REQ-SLD-17007_QEMU_QSPI.pptx	REQ-SLD-17007 (Rev 1.0)	Requirement specifications of Brugal QSPI/SPI model	Documents/010_ENG/140_FrontEnd/Project/01_SLD/2_SLD_Project/Model_Documents/01_Project_Document_Management/REQ/2015
4	QEMUandGPL.pptx	QEMU model and GPL	About GNU General Public License of QEMU model	-

Contents

Index of Figures

Index of Tables

1. Model summary

(1) Renesas develop models for Brugal product and integrate them to QEMU environment (https://www.qemu.org) to make the virtual platform.

(2) A model to be integrated to QEMU environment has below features:

(2.1) Not support timing and signals.

(2.2) Separate model interface and model core. Model interface is written in C, model core is written in C++. The model interface communicates with QEMU. The model core implements the main operation of model.

(3) This document describes the detailed specification of QSPI/SPI model. QSPI and SPI instances have same design (described in the hardware specification – ref.[2]). They are different in parameter settings.

2. Supported features

Table 2.1 List of supported features of QSPI/SPI model

Feature item	Description	Support
DMA controller interface		Yes
Boot mode support	DWC_ssi can be set to operate immediately after reset by setting the SSIC_BOOT_MODE_EN parameter to 1.	Yes
AHB interface	AMBA Specification Revsion 2.0; AHB data width fixed to 32 bit	Yes
serial-master operation		Yes
serial-slave operation		No
Programmable serial interface operation	Motorola Serial Peripheral Interface (SPI)	Yes
	Texas Instruments Synchronous Serial Protocol (SSP)	
	National Semiconductors Microwire	
multi-master contention detection		No
Programmable delay on the sample time of received serial data bit (rxd) when configured in Master Mode		No
programmable clock bit rate	Dynamic control of the serial bit rate of the data transfer	No
programmable data item size	4 to 32 bit	Yes
configurable FIFO depth	2 to 256 words deep (FIFO width is fixed at 32bit)	Yes
configurable number of slave select outputs	When operating as a serial master, 1 to 16 serial slave-select output signals can be generated	Yes
configurable hardware/software slave-select	Dedicated hardware slave-select lines or software control can be used to target the serial-slave device	No
configurable combined or individual interrupt lines	All invidual intterupt lines or one combined interrupt from the DWC_ssi to the interrupt controller	Yes
configurable interrupt polarity		No
configurable serial clock polarity		No
configurable serial clock phase	Select the serial-clock phase of the SPI format directly after reset	No

Enhanced / multi-lane (Dual / Quad / Octal) SPI support		Yes
programmable instruction, address length, wait cycles and data frame size		Yes
programmable option to skip Address and Instruction phase in enhanced SPI modes		Yes
Dual Data rate support		No
Read data strobe support in DDR mode for higher frequencies		No
Execute in Place (XIP) mode support	Programmable Instruction and address length in XIP mode	Yes (support XIP range for register access)
	Data frame size mapping directly from AHB transfers	
	Fixed data frame size transfer support	
	Continuous transfer mode support	

Note:

(*) All features are not supported at signal level. Operations at signal level are supported by API function instead.

3. Block diagram

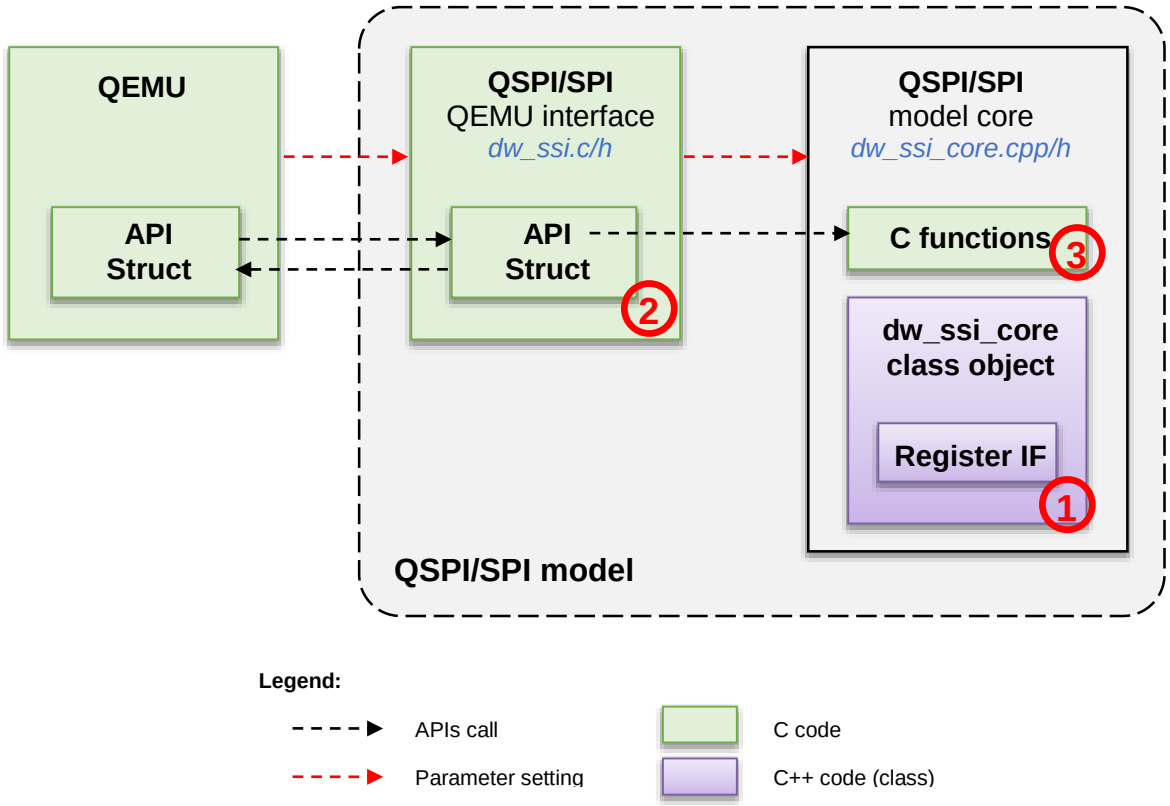

Figure 3.1: Block diagram of QSPI/SPI model.

Explanation:

(1) The model core part includes Register IF part which is generated by Register IG Generator tool. Both the model core and Register IF part are C++ codes (Number 1 in Figure 3.1).

(2) A model integrating to QEMU has to use QEMU APIs and struct definitions. QEMU also call APIs and structs defined by the model. The model's QEMU interface should be written in C (Number 2 in Figure 3.1).

(3) To mix the model core part (C++) with the QEMU interface part, some additional C functions should be added to the model core (Number 3 in Figure 3.1), in order to pass the object of dw_ssi_core C++ class to the C functions of the interface part. Example of additional functions are described in Figure 3.2 below.

(4) With this structure, the QEMU interface has to be released under the GNU General Public License (GPL) of QEMU source code. The model core part can be released without the GPL.

```
extern "C" {

typedef Cdw_ssi_core* MHandle;

MHandle construct_core(char const * s, unsigned int n)
{
   return new Cdw_ssi_core((std::string) s, n);
}

void destruct_core(MHandle p)
{
   delete p;
}

bool reg_read(MHandle p, const bool is_rd_dbg, unsigned int addr, unsigned char
*p_data, unsigned int size)
{
    return p->reg_read(is_rd_dbg, addr, p_data, size);
}

bool reg_write(MHandle p, const bool is_wr_dbg, unsigned int addr, unsigned char
*p_data, unsigned int size)
{
    return p->reg_write(is_wr_dbg, addr, p_data, size);
}

...

} // end of extern “C”
```

Figure 3.2: Block diagram of QSPI/SPI model.

For example of register access, C functions reg_read() and reg_write() will be called by QEMU interface function dw_ssi_read() and dw_ssi_write(), respectively. In turns, they calls the member functions reg_read() and reg_write() of Cdw_ssi_core class.

4. List of registers

Table 4.1 List of registers in QSPI/SPI model

Register	Bit name	Description	Supported
CTRL0	SPI_FRF	SPI Frame Format 0x0: Standard SPI format 0x1: Dual SPI format 0x2: Quad SPI format 0x3: Octal SPI format	yes
	CFS	Control Frame Size (Microwire) 0x0: 01-bit control-word … 0xF: 16-bit control-word	yes
	SSTE	Slave Select Toggle Enable	no
	SRL	Shift Register Loop	no
	SLV_OE	Slave Output Enable	no
	TMOD	Transfer mode 0x0: Transmit and Receive 0x1: Transmit only 0x2: Receive only 0x3: EEPROM Read	yes
	SCPOL		no
	SCPH		no
	FRF	SPI Frame Format 0x0: Motorola SPI Frame Format 0x1: Texas Instrument SSP Frame Format 0x2: National Semiconductors Microwire Frame Format	yes
	DFS	Data Frame Size 0x3: 04-bit data transfer 0x4: 05-bit data transfer … 0x1F: 32-bit data transfer	yes
CTRL1	NDF	Number of Receive Data Frames	yes
SSIENR	SSIC_EN	SSI Enable 0x0: Disabled 0x1: Enabled	yes
MWCR	MHS	Microwire handshaking 0x0: disabled 0x1: enabled	yes
	MDD	Microwire control	yes
	MWMOD	Microwire transfer mode	yes
SER	SER	Slave select	yes
BAUDR	SCKDV	Baud Rate select	no
TXFTLR	TFT	Tx FIFO Threshold	yes
RXFTLR	RFT	Rx FIFO Threshold	yes
TXFLR	TXTFL	Tx FIFO Level	yes
RXFLR	RXTFL	Rx FIFO Level	yes
SR	DCOL	Data Collision Error	no
	TXE	1: Transmission error 0: No error	yes

	RFF	1: Rx FIFO is full 0: Rx FIFO is not full	yes
	RFNE	1: Rx FIFO is not empty 0: Rx FIFO is empty	yes
	TFE	1: Tx FIFO is empty 0: Tx FIFO is not empty	yes
	TFNF	1: Tx FIFO is not full 0: Tx FIFO is full	yes
	BUSY	SSI Busy flag 1: module is transferring data 0: module is idle or disabled	no
IMR	MSTIM	Multi-master contention interrupt mask	no
	RXFIM	Rx FIFO Full Interrupt mask	yes
	RXOIM	Rx FIFO Overfow Interrupt mask	yes
	RXUIM	Rx FIFO Underfow Interrupt mask	yes
	TXOIM	Tx FIFO Overfow Interrupt mask	yes
	TXEIM	Tx FIFO Empty Interrupt mask	yes
ISR	MSTIS	Multi-master contention interrupt status	no
	RXFIS	Rx FIFO Full Interrupt status	yes
	RXOIS	Rx FIFO Overfow Interrupt status	yes
	RXUIS	Rx FIFO Underfow Interrupt status	yes
	TXOIS	Tx FIFO Overfow Interrupt status	yes
	TXEIS	Tx FIFO Empty Interrupt status	yes
RISR	MSTIR	Multi-master contention Raw interrupt status	no
	RXFIR	Rx FIFO Full Raw Interrupt status	yes
	RXOIR	Rx FIFO Overfow Raw Interrupt status	yes
	RXUIR	Rx FIFO Underfow Raw Interrupt status	yes
	TXOIR	Tx FIFO Overfow Raw Interrupt status	yes
	TXEIR	Tx FIFO Empty Raw Interrupt status	yes
TXOICR	TXOICR	Clear Tx FIFO Overfow Interrupt	yes
RXOICR	RXOICR	Clear Rx FIFO Overfow Interrupt	yes
RXUICR	RXUICR	Clear Rx FIFO Underfow Interrupt	yes
MSTICR	MSTICR	Clear multi-master contention interrupt	no
ICR	ICR	Clear all interrupts (TXOI, RXOI, RXUI)	yes
DMACR	TDMAE	1: Transmit DMA enabled 0: Transmit DMA disabled	yes
	RDMAE	1: Receive DMA enabled 0: Receive DMA disabled	yes
DMATDLR	DMATDLR	Transmit DMA level	yes
DMARDLR	DMARDLR	Receive DMA level	yes
IDR	IDCODE	Identification code	yes
SSIC_VERSION_ID	SSIC_COMP_VERSION	Synopsys component version	yes
DRx (x = 0..35)	DR	Data register	yes

RX_SAMPLE_DELAY	RSD	rxd sample delay	no
SPI_CTRLR0	SSIC_XIP_CONT_XFER_EN	Enable continuous transfer in XIP mode	no
	XIP_INST_EN	XIP instruction enable bit	no
	XIP_DFS_HC	Fix DFS for XIP transfers	no
	SPI_RXDS_END	rxds enable	no
	INST_DDR_EN	Instruction Dual-data rate enable	no
	SPI_DDR_EN	Dual-data rate enable	no
	WAIT_CYCLES	Wait cycles in Dual/Quad/Octal mode	no
	INTS_L	Dual/Quad/Octal mode instruction length 0x0: No instruction 0x1: 4-bit instruction length 0x2: 8-bit instruction length 0x3: 16-bit instruction length	yes
	XIP_MD_BIT_EN	1: Mode bits enabled in XIP mode 0: Mode bits disabled in XIP mode	no
	ADDR_L	Address length 0x0: No Address 0x1: 4-bit Address length 0x2: 8-bit Address length 0x3: 12-bit Address length … 0xF: 60-bit Address length	yes
	TRANS_TYPE	Address and instruction transfer format (Transmit) 0x0: Instruction and Address are sent in Standard SPI mode. 0x1: Instruction is sent in Standard SPI mode and Address is sent in the mode specified by CTRLR0.SPI_FRF. 0x2: Instruction and Address are sent in the mode specified by CTRLR0.SPI_FRF.	no
DDR_DRIVE_EDGE	TDE	TXD Driving edge	no
XIP_MODE_BITS	XIP_MD_BITS	XIP mode bits	no
XIP_INCR_INST	INCR_INST	XIP INCR transfer opcode	no
XIP_WRAP_INST	WRAP_INST	XIP WRAP transfer opcode	no

5. List of implemented ports

A model on QEMU does not support signal interface.

6. Direction for users

6.1.File structures

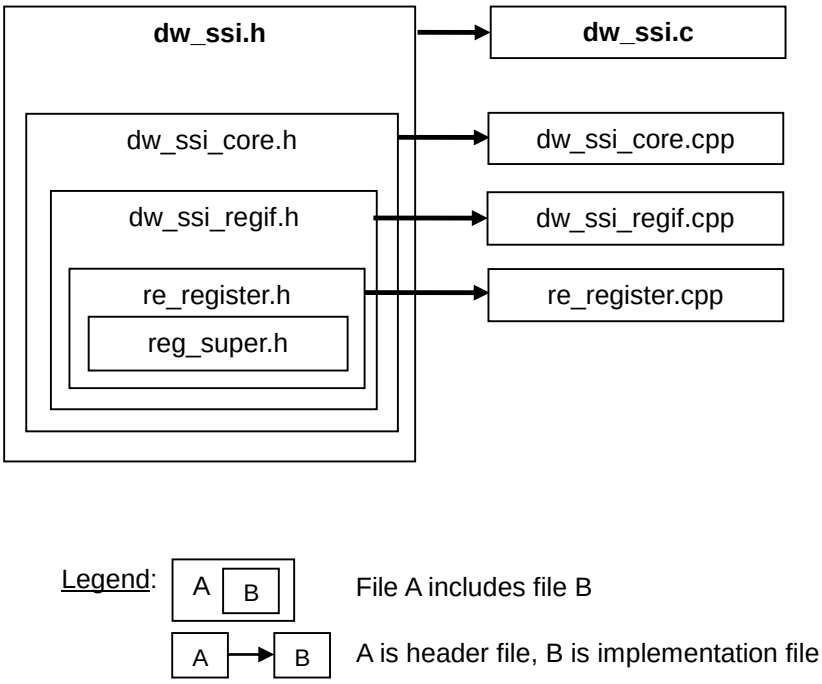

Figure 6.1: File structure of QSPI/SPI model.

Table 6.1: File description for QSPI/SPI

No.	File name	Version	Developed/ Reused/ Generated	Description
1	*re_register.h*	v2016_09_21	Reused	Header file of the re_register class.
2	*re_register.cpp*		Reused	Implement the attributes and operations of common register class.
3	*reg_super.h*		Reused	General class for models to access to the memory array.
4	*dw_ssi_regif.txt*	-	Developed	Input file of Register IF Generator.
5	*dw_ssi_regif.h*		Generated*	Header file of Register I/F of QSPI/SPI model.
6	*dw_ssi_regif.cpp*		Generated*	Implementation file of Register I/F of QSPI/SPI model.
7	*dw_ssi_core.h*		Developed	Header file of QSPI/SPI core part.
8	*dw_ssi_core.cpp*		Developed	Implementation file of QSPI/SPI core part.
9	*dw_ssi.h*		Developed	Header file of QSPI/SPI model.
10	*dw_ssi.c*		Developed	Implementation file of QSPI/SPI model (C code)

(*) Note:

(1) File *dw_ssi_regif.h* and *dw_ssi_regif.cpp* are generated from Register IF Generator tool v2017_04_13.

6.2.Input/Output file

(1) There is no input/output file for this model.

7. List of parameters

Table 7.1: List of supported parameters.

No.	Parameters	Description	Supported
Top-level parameters			
1	SSIC_ERR_RESP_EN	Enable AHB error response?	Yes
2	SSIC_IS_MASTER	Serial master or slave configuration	Yes
3	SSIC_ENH_CLK_RATIO	Include Enhanced Clock Ratio Architecture?	No
4	SSIC_RX_FIFO_DEPTH	Receive FIFO buffer depth	Yes
5	SSIC_TX_FIFO_DEPTH	Transmit FIFO buffer depth	Yes
6	SSIC_NUM_SLAVES	Number of slave select lines	Yes
7	SSIC_HAS_RX_SAMPLE_DELAY	Include Programmble RXD Sample Logic	No
8	SSIC_RX_DLY_SR_DEPTH	Maximum RXD Sample Delay	No
9	SSIC_ID	Peripheral ID code	Yes
10	SSIC_HAS_DMA	Include DMA handshaking interface signals?	No
11	SSIC_INTR_IO	Configure interrupt pinout	Yes
12	SSIC_INTR_POL	Active interrupt level	No
13	SSIC_SYNC_CLK	Are hclk and ssi_clk synchronous?	No
14	SSIC_CLK_EN_MODE	Generate clock enable input for ssi_clk?	No
15	SSIC_BOOT_MODE_EN	Enable boot mode for DWC_ssi?	Yes
16	SSIC_DFLT_FRF	Default frame format	Yes
17	SSIC_HC_FRF	Hard code the frame format?	Yes
Enhanced SPI parameters			
18	SSIC_SPI_MODE	Select SPI mode	Yes
19	SSIC_IO_MAP_EN	Enable Enhanced SPI I/O mapping	No
20	SSIC_HAS_DDR	Include DDR transfers in SPI frame format?	No
21	SSIC_HAS_RXDS	Include data strobe signal for rxd line?	No
22	SSIC_XIP_EN	Include XIP feature in SPI mode?	Yes
23	SSIC_XIP_INST_EN	Enable Instruction phase in XIP mode?	No
24	SSIC_XIP_CONT_XFER_EN	Enable Continous transfer mode in XIP mode?	No
Register default value parameters			
25	SSIC_DFLT_SLV	Default Slave	Yes
26	SSIC_DFLT_BAUDR	Default Baud Rate	Yes
27	SSIC_DFLT_NDF	Default Number of data frames to be fetched	Yes
28	SSIC_DFLT_DFS	Default data frame size	Yes
29	SSIC_DFLT_CFS	Default control frame size for microwire transfers	Yes
30	SSIC_DFLT_TMOD	Default transfer mode	Yes
31	SSIC_DFLT_SCPOL	Default serial clock polarity	Yes
32	SSIC_DFLT_SCPH	Default serial clock phase	Yes

33	SSIC_DFLT_SPI_FRF	Default SPI frame format	Yes
34	SSIC_DFLT_TRANS_TYPE	Default enhanced SPI transfer type	Yes
35	SSIC_DFLT_ADDR_L	Default Address length	Yes
36	SSIC_DFLT_MD_BITS_EN	Mode bits enabled in XIP operations by default?	Yes
37	SSIC_DFLT_INST_L	Default instruction length	Yes
38	SSIC_DFLT_WAIT_CYCLES	Default wait cycles	Yes
39	SSIC_DFLT_DDR_EN	Enable SPI DDR transfers by default?	No
40	SSIC_DFLT_INST_DDR_EN	Send instruction in DDR format by default?	No
41	SSIC_DFLT_RXDS_EN	Enabled RXDS signalling by default in SPI DDR transfers?	No
42	SSIC_DFLT_DFS_HC	Hardcode data frame size in XIP transfers by default?	Yes
43	SSIC_DFLT_XIP_INST_EN	Enable Instruction phase in XIP transfers by default?	Yes
44	SSIC_DFLT_XIP_CONT_XFER_EN	Enable continous mode in XIP transfers by default?	Yes
45	SSIC_DFLT_MD_BITS	Default value of XIP_MODE_BITS register	Yes
46	SSIC_DFLT_INCR_INST	Default value of XIP_INCR_INST register	Yes
47	SSIC_DFLT_WRAP_INST	Default value of XIP_WRAP_INST register	Yes
48	SSIC_H_2_S_SYNC_DEPTH	hclk to ssi_clk Synchronization Depth	No
49	SSIC_S_2_H_SYNC_DEPTH	ssi_clk to hclk Syncrhonization Depth	No

7.1.1.Register RW messages style

Table 7.2: Dump Register RW message description

Condition	This message is dumped out when DW_SSI's registers are accessed.
Output	This message is printed to standard output (console).
Format: Info: (<hier_instance_name>): [<time> ns] REG [ECM: <reg_name>] <operation> Size = <size> Addr = <reg_address> Data = <reg_value> **Example:** Info: QSPI: [75 ns] REG [SPI0:CTRL0] R Size= 4 Addr= 0x00000100 Data= 0x5	

Tag name	**Description**
time	Simulation time (not available in QEMU).
hier_instance_name	Hierarchy instance name of model.
reg_name	Name of accessed register.
operation	R or W (read or write).
size	Accessed register size (in byte).
reg_address	Address of accessed register.
reg_value	Register value.

7.1.2.Interrupt messages style

Table 7.3: Dump Interrupt message description

Condition	This message is dumped out when safety interrupt of QSPI/SPI is asserted and parameter DumpInterrupt is true.
Output	This message is printed to standard output (console).
Format: Info [<time>ns] (hier_instance_name) INT [QSPI/SPI: interrupt_name] Assert **Example:** Info [2010 ns] (QSPI/SPI) INT [QSPI/SPI: ECMTI] Assert	
Tag name	**Description**
time	Simulation time (not available in QEMU).
hier_instance_name	Hierarchy instance name of model.

7.1.3.Error and debugging message style

Table 7.4: Description of error messages

Condition	This kind of message is output when error or fatal occurs or some important events occur. Detailed conditions are described in the “Description” column of Table 7.5.
Output	This kind of message is printed to standard output (console).
Format: <Severity> [<time>ps] (<hier_instance_name>) [Message content] **Example:** Info [1234000ps] (reslx.QSPI/SPI) Reset signal is asserted.	
Tag name	**Description**
severity	Kind of severity of the message
time	Simulation time (not available in QEMU).
hier_instance_name	Hierarchy instance name of QSPI/SPI model

7.1.4.List of error and debugging messages

Table 7.5: Error and debugging message of QSPI/SPI model

No.	Type	Severity	Message	Description
1	Users	Error	Writing access size to %s at address 0x%08X is wrong: %d byte(s).	Dump this message when tgt_acc function is called with write access size is different the register size.
2	Users	Error	Reading access size to %s at address 0x%08X is wrong: %d byte(s).	Dump this message when tgt_acc function is called with read access size is different the register size.
3	Users	Error	Invalid access address 0x%08X	Dump this message when register is read/written with wrong address.
4	Users	Warning	%s forbids to write 0.	Dump this message when a read-only bit is written 0, or write-1-only bit.
5	Users	Warning	%s forbids to write 1.	Dump this message when a read-only bit is written 1.
6	Users	Warning	%s forbids to read.	Dump this message when a write-only register is read.
7	Users	Warning	Cannot write 1 to reserved bit.	Dump this message when a reserved bit is written 1.
8	Users	Warning	Should read all bit in a register.	Dump this message when only some bits of register are read.

9	Users	Info	Reset signal is asserted.	Dump this message when reset function is called.
10	Users	Info	INT [QSPI/SPI: %s] Assert.	Dump this message when an interrupt request is sent.

7.2.Define macro and template

(1) This model requires "**REGIF_NOT_USE_SYSTEMC**" macro to be defined, in order to disable the SystemC source code in Register IF class.

8. Flow diagram

8.1.Sequence flow

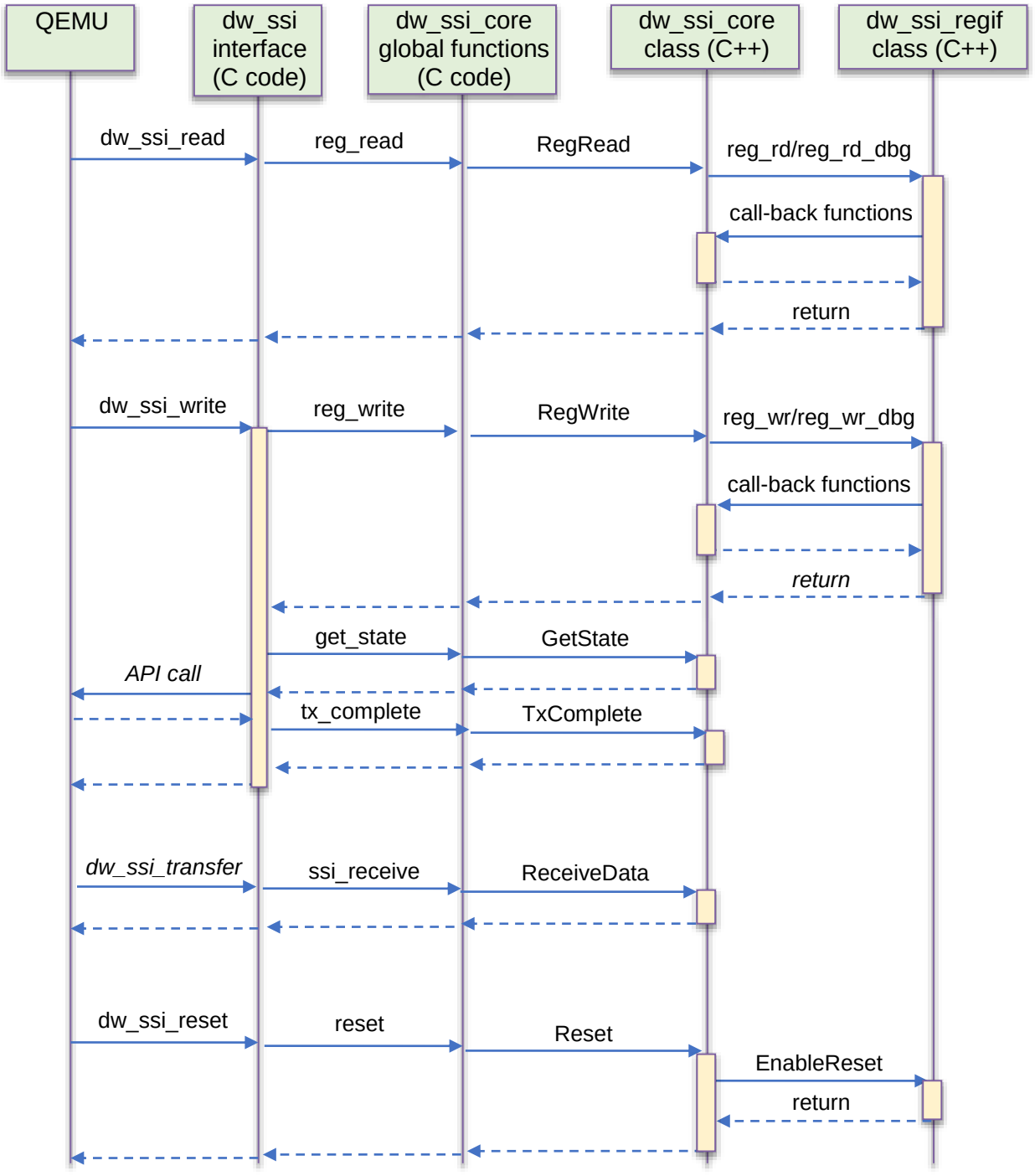

Figure 8.1: Sequence diagram of QSPI/SPI model

Explanation:

(1) This model (dw_ssi) supports below API functions:

(1.1) dw_ssi_read() for reading value of a register.

(1.2) dw_ssi_write() for writing value of a register.

(1.3) dw_ssi_transfer() for sending data to SPI model (similar to RXD).

(1.4) dw_ssi_reset() for reset.

(2) Above APIs will call the corresponding functions of dw_ssi_core class.

(3) When a register of model is read or written, the request will be passed to the dw_ssi_regif class which manages all model registers. dw_ssi_regif might call the call-back functions of dw_ssi_core class in case that the write or read request triggers a process in the model. Call-back functions are virtual functions of Register IF class and are defined in the model core class.

(4) In some processes, QSPI/SPI model need to call the global C functions in order to communicate with other modules, for example, sending interrupt request, sending SSI data to other SPI modules, sending DMA request. Because the core class cannot call external (global) functions, it stores its status in internal variables and update the status after a read/write transaction. The dw_ssi model (C interface) will check the status of dw_ssi_core via GetState() function. Based on the status of the core class, the interface will call the global C functions of QEMU to communicate with other modules.

(5) When a reset is asserted, the core class will initialize itself and call the EnableReset() of Register IF class to initialize all registers.

8.2.State diagram

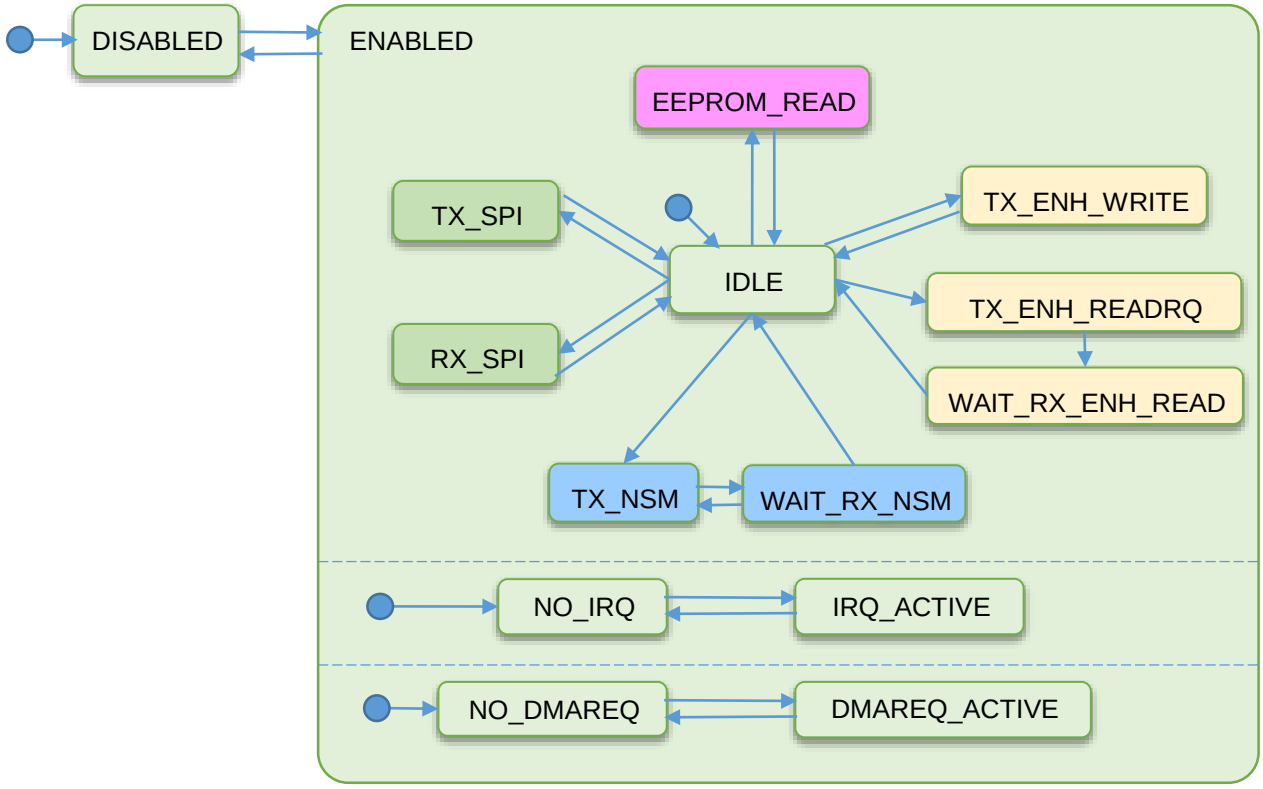

Figure 8.2: State diagram of QSPI/SPI model

Explanation:

(1) DISABLED: this is the initial state, when SSIC_EN bit is 0. Model is deactivated..

(2) ENABLED: when SSIC_EN is 1, model operation is enabled. Within this state, there are three sub orthogonal regions in which the states of transfer operation, interrupt request and DMA request are defined.

(3) For transfer operation, state of transmitting and reception of data are defined as below:

(3.1) IDLE: no transfer (initial value)

(3.2) TX_SPI: standard SPI transmission. Model is transmitting data to outside. In this state, the model interface will call API of opposite model to transmit the data.

(3.3) RX_SPI: standard SPI reception, when the RxTransfer() is called.

(3.4) EEPROM_READ: transmit request and read data in EEPROM read mode. In this state, the model interface will call API of EEPROM model to read its memory.

(3.5) TX_NSM: transmit Control Word or TX Data Word in Microwire format.

(3.6) WAIT_RX_NSM: after transmission in Microwire format is done, model waits for RxTransfer() to be called.

(3.7) TX_ENH_WRITE: transmit data of Write transaction in Enhanced SPI mode (data is composed of instruction phase, address phase and data phase).

(3.8) TX_ENH_READRQ: transmit request of Read transaction in Enhanced SPI mode (data is composed of instruction phase and address phase)

(3.9) WAIT_RX_ENH_READ: after Read request in Enhance SPI mode is sent, model waits for RxTransfer() to be called.

(4) For interrupt request, there are two states:

(4.1) NO_IRQ: no interrupt is active.

(4.2) IRQ_ACTIVE: there is at least one interrupt is active. The active interrupt is stored in a variable. Model interface checks that variable and send interrupt request by calling QEMU's API.

(5) For DMA request, there are two states:

(5.1) NO_DMAREQ: DMA request is not needed.

(5.2) DMAREQ_ACTIVE: model need to send DMA request to DMA model. Model interface will call DMA's API to send request.

8.3. State transition table

8.3.1. Disabled/Enabled states

Table 8.1: State transition table (Disabled/Enabled)

State / Event	**DISABLED**	**ENABLED**
Write SSIC_EN = 1	**⇨ ENABLED**	-
Write SSIC_EN = 0	-	**⇨ DISABLED** Clear all FIFO entries

8.3.2. State transition within ENABLED state

(1) When the model is switched to ENABLED state, the initial sub-state is IDLE.

(2) When data is written to a DR register, the data will be pushed into the FIFO. Based on the settings defined by registers' value, the sub-state may change from IDLE to another state to start the transfer. The condition of state change from IDLE state is described in *Table 8.2*.

Table 8.2: State transition from IDLE state at event "Write value to DR register"

SPI_FRF	FRF / TMOD	0 (SPI)	1 (SSP)	2 (NSM)
0 (Single SPI)	0 (Tx & Rx)	**IDLE ⇨ TX_SPI** Transmit data		**IDLE ⇨ TX_NSM** Transmit Ctrl Word
	1 (Tx only)	**IDLE ⇨ TX_SPI** Transmit data		
	2 (Rx only)	**-**		
	3 (EEPROM Read)	**IDLE ⇨ EEPROM_READ** Read data from EEPROM	-	-
1,2,3 (Enhanced SPI)	1 (Write)	**IDLE ⇨ TX_ENH_WRITE** Transmit Write package (instruction + address + data)	-	-
	2 (Read)	**IDLE ⇨ TX_ENH_READREQ** Transmit Read request package (instruction + address)	-	-

(*) Other states are not affected by this event.

(3) The model interface check the state right after the writing value to registers.

(3.1) When state changes to TX_***, the model interface will call the API of opposite model to transmit the data (previously stored in FIFO). After the data is transferred to the opposite model, the model interface will call TxComplete() function to notify the model core about the completion of transmission. The state will continue to change according to *Table 8.3*.

*Table 8.3: State transition from TX_*** states*

State / Event	**TX_SPI**	**TX_NSM**	**TX_ENH_WRITE**	**TX_ENH_READREQ**
TxComplete() is called	**⇨ IDLE**	**⇨ WAIT_RX_NSM**	**⇨ IDLE**	**⇨ WAIT_RX_ENH_READ**

(*) Other states are not affected by this event.

(3.2) When state changes to EEPROM_READ, the model interface will call the API of EEPROM model to read the memory value. After getting the data from EEPROM model, the model interface will transfer it to the model core by calling ReceiveData().

(4) When opposite model send data to this model, it will call the function dw_ssi_transfer() of the model interface, which in turn calls ReceiveData() of the model core. The state transitions when a data is received are described in *Table 8.4*.

Table 8.4: State transition at event "Receive data"

State / Event	**IDLE**	**EEPROM_READ**	**WAIT_RX_NSM**	**WAIT_RX_ENH_READ**
ReceiveData() is called	If SPI_FRF is 0 and TMOD is 0 or 2 ⇨ **RX_SPI** Store data to FIFO	⇨ **IDLE** Store data to FIFO	Continuous transmission: ⇨ **TX_NSM** Transmit next data	Continuous transmission: ⇨ **TX_ENH_READREQ** Transmit next package
	Else: ⇨ **IDLE**		Single transmission: ⇨ **IDLE** Store data to FIFO	Single transmission: ⇨ **IDLE** Store data to FIFO

(*) Other states are not affected by this event.

8.4. Register access

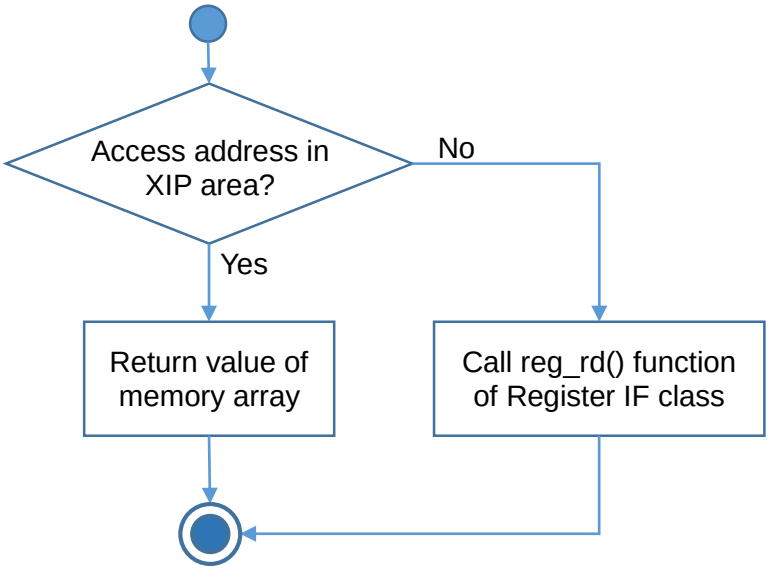

Figure 8.3: Flow of a reg_read() function.

Explanation:

(1) Registers are accessed by calling dw_ssi_write() or dw_ssi_read().

(2) When dw_ssi_write() is called, the model interface will call reg_write() in model core, which in turn call reg_wr() of Register IF class

(3) For read access, reg_read() in model core will be called. The Figure 8.3 describes the flow of red_read() function.

(3.1) If XIP (Execute-in-Place) access is supported, and the access address is within the XIP range, model core will return the value of its internal memory array.

(3.2) Otherwise, model core will call reg_rd() function of Register IF class.

9. Class explanation

9.1. Model interface

Table 9.1: List of function of model interface (dw_ssi.c and dw_ssi.h)

No.	Function prototype	Description
1	static void dw_ssi_init(Object *obj)	Initialize the model
2	static void dw_ssi_read(void *opaque, hwaddr offset, unsigned size)	Read value of model registers
3	static void dw_ssi_write(void *opaque, hwaddr offset, uint64_t val, unsigned size)	Write value to model registers
4	uint32_t dw_ssi_transfer(SSIBus *bus, uint32_t value)	Send data to this model

9.2. Model core

Table 9.2: List of function of model core (dw_ssi_core.cpp and dw_ssi_core.h)

No.	Function prototype	Type	Description
1	void construct_core(Cdw_ssi_core* p, unsigned int n)	global	Call the constructor of Cdw_ssi_core class
2	void destruct_core(Cdw_ssi_core*)	global	Call the destructor of Cdw_ssi_core class
3	bool reg_read(Cdw_ssi_core* p, const bool is_rd_dbg, unsigned int addr, unsigned char *p_data, unsigned int size)	global	Read value of model registers
4	bool reg_write(Cdw_ssi_core* p, const bool is_wr_dbg, unsigned int addr, unsigned char *p_data, unsigned int size)	global	Write value to model registers
5	get_state(void)	global	Get the SPI operation state of model core
6	get_irq_state(void)	global	Get the interrupt state of model core
7	get_dma_state(void)	global	Get the DMA request state of model core
8	void tx_complete(void)	global	Notify completion of transmission
9	bool ssi_receive(…)	global	Receive data from outside
10	bool Cdw_ssi_core::RegRead(…)	public	Called by reg_read(), this function will call Cdw_ssi_regif::reg_rd()
11	bool Cdw_ssi_core::RegWrite(…)	public	Called by reg_write(), this function will call Cdw_ssi_regif::reg_wr()
12	Cdw_ssi_core::GetState(void)	public	Called by get_state()
13	Cdw_ssi_core::GetIrqState(void)	public	Called by get_irq_state()

14	Cdw_ssi_core::GetDmaState(void)	public	Called by get_dma_state()
15	void Cdw_ssi_core::TxComplete(void)	public	Called by tx_complete().
16	bool Cdw_ssi_core::ReceiveData(…)	public	Called by ssi_receive().
17	bool Cdw_ssi_core::cb_<regname>_<bitname> (RegCBstr str)	protected	Call-back functions when a register is accessed.

Revision History					
Rev.	**Modified Contents**	**Agreed by Customer**	**Approved by RVC**	**Checked**	**Created**
1.0	New creation		Yen Nguyen 2017/11/13	Yen Nguyen 2017/11/13	Duc Duong 2017/11/13