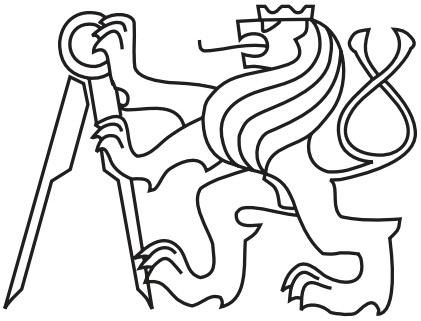
Czech Technical University in Prague

Faculty of Electrical Engineering

Department of Microelectronics



Design of the QSPI master interface

Master's Thesis

Author: Bc. Jan Němeček
Supervisor: doc. Ing. Jiří Jakovenko, Ph.D.
Year: Prague, 2020

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Němeček** Jméno: **Jan** Osobní číslo: **457113**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra mikroelektroniky**

Studijní program: **Elektronika a komunikace**

Specializace: **Elektronika**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Design of QSPI master interface

Název diplomové práce anglicky:

Návrh QSPI master rozhraní

Pokyny pro vypracování:

1. Seznamte se s protokolem QSPI, analyzujte rozdíly mezi Flash paměťmi, které QSPI rozhraní podporují, a připravte systémový návrh QSPI master IP pro procesor RISC V.
2. Implementujte vybrané bloky QSPI master IP na RTL úrovni v jazyce VHDL.
3. Vybranými metodami prokažte správnou funkci navrženého systému.

Seznam doporučené literatury:

1. W25Q128FV Serial flash Memory with dual/quad SPI & QPI Datasheet, Winbond, Revision 1, 2013
2. Digital Design and Computer Architecture 2nd Edition, D. Harris, S. L. Harris, Morgan Kaufmann, 2012
3. RI5CY: User Manual, A. Traber, M. Gautschi, P. D. Schiavone, ETH Zurich, Revision 4.1, 2019

Jméno a pracoviště vedoucí(ho) diplomové práce:

doc. Ing. Jiří Jakovenko, Ph.D., katedra mikroelektroniky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **11.02.2020** Termín odevzdání diplomové práce: **22.05.2020**

Platnost zadání diplomové práce: **30.09.2021**

doc. Ing. Jiří Jakovenko, Ph.D. podpis vedoucí(ho) práce	prof. Ing. Pavel Hazdra, CSc. podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

. _____
Datum převzetí zadání

_____
Podpis studenta

Declaration

I declare that this thesis has been composed solely by myself and that I have used only sources (literature, software, ...) listed in the included list.

In Prague, 22. 5. 2020

.....

Bc. Jan Němeček

Acknowledgements

I would like to thank to my supervision doc. Ing. Jiří Jakovenko Ph.D. for valuable advices during the development of this thesis. Also, I would like to express my gratitude to my colleagues from ASICentrum s.r.o. for cooperation, valuable advices, and guidance throughout my work. Namely to Ing. Jakub Šťastný Ph.D. Last but not least, I would like to thank my family for support during whole studies.

Abstrakt

Tato diplomová práce pojednává o návrhu a implementaci QSPI master rozhraní s procesorovým jádrem RISCV. QSPI protokol byl prostudován z dostupných flash pamětí, které QSPI rozhraní podporují. Byly porovnány rozdíly v protokolu mezi různými flash paměťmi a sestaven jednotný popis protokolu. Dále bylo prostudováno RISCV PULP rozhraní, aby k němu bylo možné připojit QSPI master rozhraní. Protokolové funkce a parametry byly vybrány a byl vytvořen systémový návrh a specifikace. Jednotlivé bloky návrhu byly implementovány na RTL úrovni pomocí VHDL. Během VHDL implementace byl návrh průběžně testován pomocí VHDL testů. Ověření konceptu návrhu bylo provedeno implementací QSPI master rozhraní společně RISCV procesorem do FPGA. Procesor byl naprogramován a byla ověřena komunikace mezi procesorem a QSPI rozhraním. Dále byla ověřena komunikace mezi QSPI rozhraním a připojenou externí flash pamětí. Na závěr byly pomocí UVM verifikačního prostředí testovány základní scénáře použití. Návrh je tím připraven na rozsáhlé testování.

Klíčová slova

QSPI, PULP, RI5CY, RISCV, FPGA, DDR, Flash paměť, VHDL

Abstract

This master's thesis deals with the design and implementation of the QSPI master interface with the RISCV processor core. The QSPI protocol was studied from available flash memories, which support QSPI protocol. Differences in the protocol were compared between studied flash memories, and a unified protocol description was written. The RISCV PULP interface was studied to allow connection of the RISCV with the QSPI master interface. Protocol features and parameters were chosen, and system-level design and design specification was created. Individual blocks of the design were implemented in RTL with VHDL. The design was continuously tested during the VHDL implementation phase with the VHDL testbench. Proof of concept was done by the implementation of the design with the RISCV processor into FPGA. The processor was programmed, and communication between the QSPI interface and the processor was verified. The QSPI communication was verified between the QSPI interface and external flash memory. At last, basic use-cases were verified in the UVM environment implemented in System Verilog. Thereby, the design was prepared for full verification.

Keywords

QSPI, PULP, RI5CY, RISCV, FPGA, DDR, Flash memory, VHDL

Contents

List of Figures

List of Tables

Symbols and Abbreviations

AHB	Advanced High-performance Bus
APB	Advanced Peripheral Bus
ASIC	Application-Specific Integrated Circuit
CPHA	Clock Phase
CPOL	Clock Polarity
CS	Chip Select
DDR	Dual Data Rate
DSPI	Dual Serial Peripheral Interface
FIFO	First In, First Out
FPGA	Field-Programmable gate array
FSM	Finite-State Machine
ILA	Integrated Logic Analyser
LSU	Load Store Unit
MCU	Micro-Controller Unit
MGT	Multi-Gigabit transceiver
PCB	Printed Board Circuit
PPL	Phase-locked loop
PSL	Property Specification Language
PULP	Parallel Ultra-Low-Power
QIO	QSPI Input Output data pin
QSPI	Quad Serial Peripheral Interface
RTL	Register Transfer Language
RX	Receiver
SCLK	Serial Clock
SDR	Single Data Rate
SIOO	Send Instruction Only Once
SOC	System On a Chip
SPI	Serial Peripheral Interface
SSPI	Single Serial Peripheral Interface
TX	Transmitter
UVM	Universal Verification Methodology
VHDL	Very High Speed Integrated Circuit Hardware Description Language
XIP	Execute In Place

1 Introduction

1.1 Motivation

Nowadays, modern embedded processors boot from external flash memories. Typically two types can be distinguished. Either a memory is equipped with a parallel interface or with a serial interface. The parallel interface is usually composed of the address, data, and control buses. The memory with a parallel interface offers higher performance and faster transaction speeds. However, it takes up a considerably larger printed circuit board space, and it uses more processor pins. Therefore it is not suitable for modern smaller low power designs. These disadvantages are solved by usage of the quad serial peripheral interface (QSPI). The interface is composed of the chip select signal, clock signal, and four data lines. Overall, it uses fewer signals to communicate. Therefore signal alignment is simpler. The use of flash memory with the quad serial peripheral interface can significantly decrease the used printed circuit board (PCB) area. Consequently, it can lower the overall cost of the PCB. However, the serial interface has a more complicated design, and it is significantly slower than the parallel interface. The slower speed of the interface is acceptable in most cases if data from memory are not read to often [1].

The RISCV processor cores are becoming more and more popular as they are free to use. They also offer a variety of cores depending on the type of use. For example, RI5CY PULP is intended for ultra low power applications. There is a need for peripherals to create a functional system on a chip. That is why it was decided to design the QSPI master interface block as a peripheral for the RISCV [2].

1.2 Objectives

This master's thesis focus on the design of the QSPI master interface integrable with the RISCV processor core. The thesis is divided into several chapters. First, it is needed to study the QSPI protocol from different sources and create a unified description of the protocol, which will later be used during the design phase. RI5CY PULP interface is also studied to design a protocol bridge between the QSPI and PULP protocol. Second, a specification of the QSPI master interface is prepared based on the study of existing serial flash memories and their features. A system-level design is created. The design is implemented on the Register-transfer level (RTL) in Very High-Speed Integrated Circuit Hardware Description Language (VHDL). The design is verified during the design process by basic VHDL testbench. At last, the design is verified in the universal verification methodology (UVM) test environment implemented in System Verilog, and FPGA implementation with RISCV processor core is done as proof of concept to prove that design can communicate with real processor and QSPI flash memory.

2 Theoretical Introduction

2.1 QSPI Protocol

The Quad Serial Peripheral Interface (QSPI) is a synchronous serial master-slave based interface mostly used for communication between the processor and external memories. The protocol allows connection of single memory or multiple memories in parallel. The interface uses only one master. However, it can handle multiple slave devices. The master is a device that controls the communication. The protocol is based on the classic Serial Peripheral Interface (SPI) interface. The difference is that QSPI allows communicating on four data lines. The main benefits of the interface are mainly fully configurable and flexible data frame format, support of traditional SPI and Dual SPI, reduction of used pins of processor. It helps to reduce printed circuit board size and final cost. The protocol can offer up to four times larger throughput and with support of double data rate mode up to eight times larger throughput than classic SPI [3].

The interface protocol features can vary between different manufactures. The main reason is the non-existent ISO standard for QSPI. Some companies offer their standards such as JEDEC's JESD216 [4] for Serial NOR flash. However, it only describes the content of QSPI frames and not the entire protocol. The primary purpose of this chapter is to make a summary of protocol features supported by flash memories from different manufactures and draw united protocol description. As the main resource device datasheets [1, 3, 5] are used.

2.1.1 Interface

A basic configuration interface with one slave device uses six pins/wires to communicate. The number of used pins increases with the number of connected slave devices. Master generates the serial clock signal at (*SCLK*) pin. The Chip Select (*CS*) pin is used to select/activate slave by pulling line low. Every slave device should have its own *CS* pin driven by the master. The QSPI uses four Quad Input Output data lines (*QIO0-3*), for the data transfer. The pin *QIO0* corresponds to Master Out Slave In (MOSI) pin and *QIO1* to the Master In Slave Out (MISO) pin of SPI if it is used in a single SPI mode. An example configuration is shown in Figure **2.1** [5].

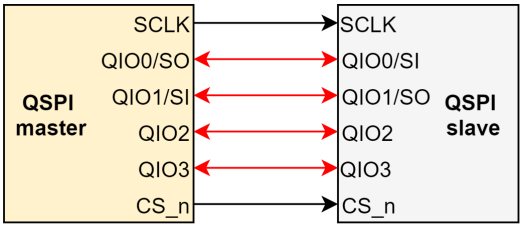


Figure 2.1: QSPI interface configuration with single slave device.

2.1.2 Frame Format

The QSPI offers a flexible frame format. The frame can be composed of 5 phases – command, address, alternate, dummy, and data phase. Some phases can be named differently by manufacturers. Each phase can be skipped. However, an order of phases does not change. Users can individually configure length and mode for each phase. Every phase can be sent over one, two, or four data lines. Example of a QSPI communication frame with highlighted phases can be seen in Figure **2.2**. The description of the depicted phases follows below [3].

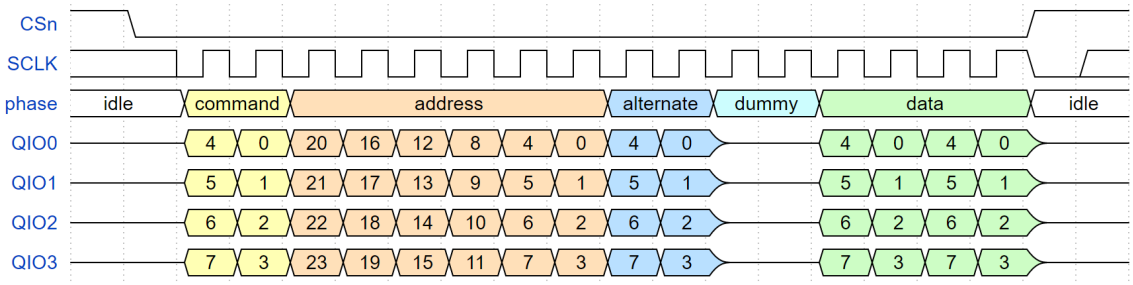


Figure 2.2: Example of full QSPI transaction with highlighted frame phases.

Command Phase

Eight bits long instruction is sent from master to slave during the command phase. The objective of this phase is to specify which operation will be performed (Write enable, Page program, Quad read, Erase, ...). Instructions or commands are unique to specific slave devices or manufacturers. This phase can be sent only in Single Data Rate (SDR) mode. Users can select if the whole byte is sent or phase is skipped. Skipping of the phase is mostly used when the same type of transactions will follow after an initial transaction with command phase [3].

Address Phase

During the address phase, an address bytes are sent from master to slave device to specify the address in slave device memory space from/to which data will be read/written. The phase length is configurable, typically it supports addresses from one byte up to four bytes. The phase supports Dual Data Rate (DDR) mode. User can also skip this phase [3].

Alternate Phase

This phase is an extra phase supported by most of the manufacturers. Typically after address phase master sends an extra byte to the slave. The function of this phase can vary for different manufacturers. Most often it is used to have extra control over operation mode and to keep the slave in operational mode. It usually means that the next transaction frame will skip the instruction phase to avoid repetitiveness. The phase length is usually one byte or in some cases nibble of byte. The phase supports DDR mode [1].

Dummy Phase

The main reason of the dummy phase is to ensure enough turnaround time before the data phase to complete initial read access of the flash array before the master device can read data, mainly if the high clock frequency is used. During the dummy phase, no bits are transmitted, but the user can define whether the whole phase is transmitted as High Z or as 0. Users can configure how many dummy cycles shall be transmitted or skip the phase. One dummy bit lasts one clock cycle even during DDR mode [3].

Data Phase

In this phase, data bytes are sent/received from master/slave to slave/master. It is the only phase when the slave can send data to the master. The phase length is fully configurable, and it is theoretically unlimited. Turnaround phase occurs before the phase if the slave is going to transmit to the master, it means that QIO pins are going to switch directions of communication. The phase supports DDR mode. The data phase can also be skipped [3].

2.1.3 Data Modes

The user can select through how many data lines communication will occur for each frame phase. This offers great flexibility for all types of QPSI devices or it can be used to reduce number of needed GPIO pins. Three separate modes can be distinguished, Single SPI (SSPI), Dual SPI (DSPI), and Quad SPI (QSPI). The user can select logical values in which data lines are kept during transactions for unused data lines. An example of the QPSI transfer with different mode used in each transfer phase is shown in Figure **2.3** [5].

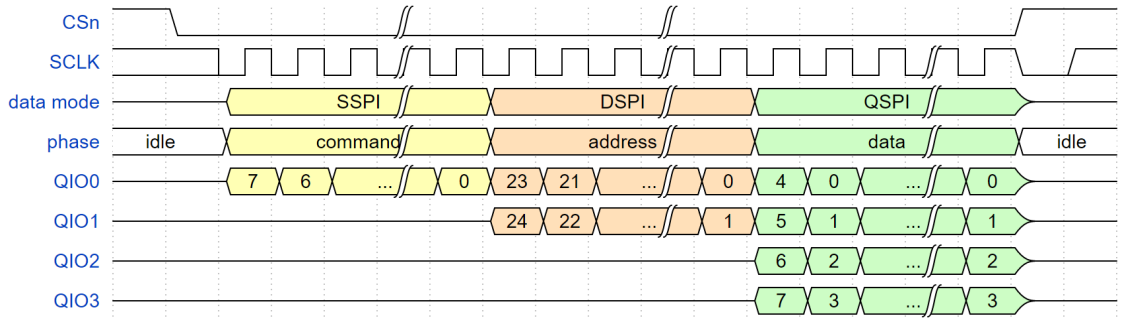


Figure 2.3: QSPI transaction frame with used all data modes.

Single SPI Mode

Only two data lines are used for the SSPI mode. Unused data lines can be set to a selected logical state. The interface configuration with marked directions of lines is shown in Figure **2.4**. It can be seen that *QIO0* is used to transfer data from master to slave and *QIO1* from slave to master [6].

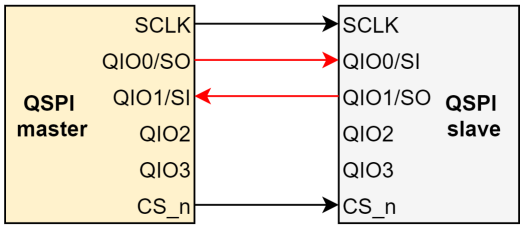


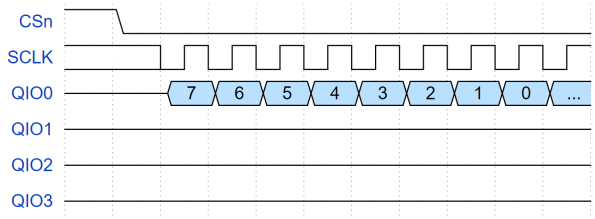
Figure 2.4: SSPI interface configuration.

This mode is based on a classic SPI interface. However, it supports only half-duplex transfer and most of the time without availability of all configurations of polarity and phase. The SPI interface allows users to change Clock Polarity (CPOL) and Phase (CPHA) according to chosen CPHA and CPOL bit. It determines on which clock edge data are being shifted out/sampled and at which logic value is serial clock signal during idle state. All possible SPI modes are shown in Table **2.1**.

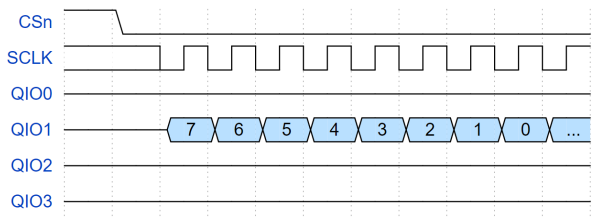
Table 2.1: SPI CPOL and CPHA modes.

Mode	CPOL	CPHA	Idle clock state	Data sampled	Data shifted out
0	0	0	low	rising edge	falling edge
1	0	1	low	falling edge	rising edge
2	1	1	high	falling edge	rising edge
3	1	0	high	rising edge	falling edge

Transaction examples of read and write in SSPI mode are shown in Figure **2.5**. It can be seen that read and write use different data lines [6].



(a)



(b)

Figure 2.5: SSPI a) write and b) read transactions.

Dual SPI Mode

The DSPI mode uses only two data lines for data transfer as same as SSPI transfer. However, both lines are used at the same time to send/receive data. The unused pins *QIO2* and *QIO3* can be optionally set to specific logical values. The interface configuration can be seen in Figure **2.6**. An example of the data transaction using DSPI mode is shown in Figure **2.7** [5].

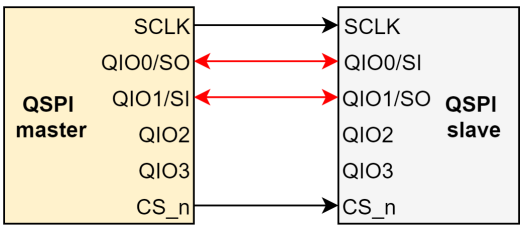


Figure 2.6: DSPI interface configuration.

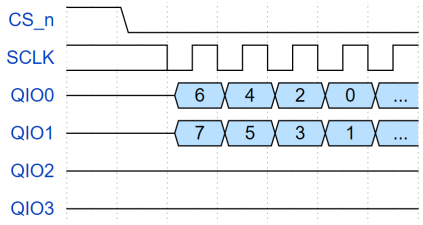


Figure 2.7: DSPI transaction using two data lines.

Quad SPI Mode

The QSPI mode uses all four data lines thus all pins in hardware configuration are being used. The data are being received/sent by all four data lines at the same time. Example of the data transaction using QSPI mode is shown in Figure **2.8** [5].

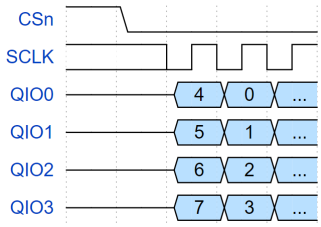


Figure 2.8: QSPI transaction using all four data lines.

2.1.4 SDR and DDR Mode

The Single Data Rate (SDR) means that data are sent on the falling edge and sampled on a rising edge of the serial clock that corresponds to configuration CPOL = 1 and CPHA = 1 or CPOL = 0 and CPHA = 0 of SPI. The SDR mode is set by default.

The Dual Data Rate (DDR) or the Dual Transfer Rate (DTR) is transfer mode where the data are sampled/sent on both rising and falling edge of the serial clock. The first data are sent on falling edge with the last section sent on rising edge if configuration corresponds to CPOL = 1 and CPHA = 1 or CPOL = 0 and CPHA = 0. The data are always sampled half clock cycle after they are sent. The DDR mode can theoretically double data throughput. It can be very useful, especially when the system runs at lower clock frequency. The command phase is always sent in SDR mode as well as dummy phase, during which each dummy bit lasts one clock cycle. Examples of both SDR and DDR transactions with equal settings of phases and data modes are shown in Figure **2.9**. It can be seen that command phase is always sent in SDR mode [1].

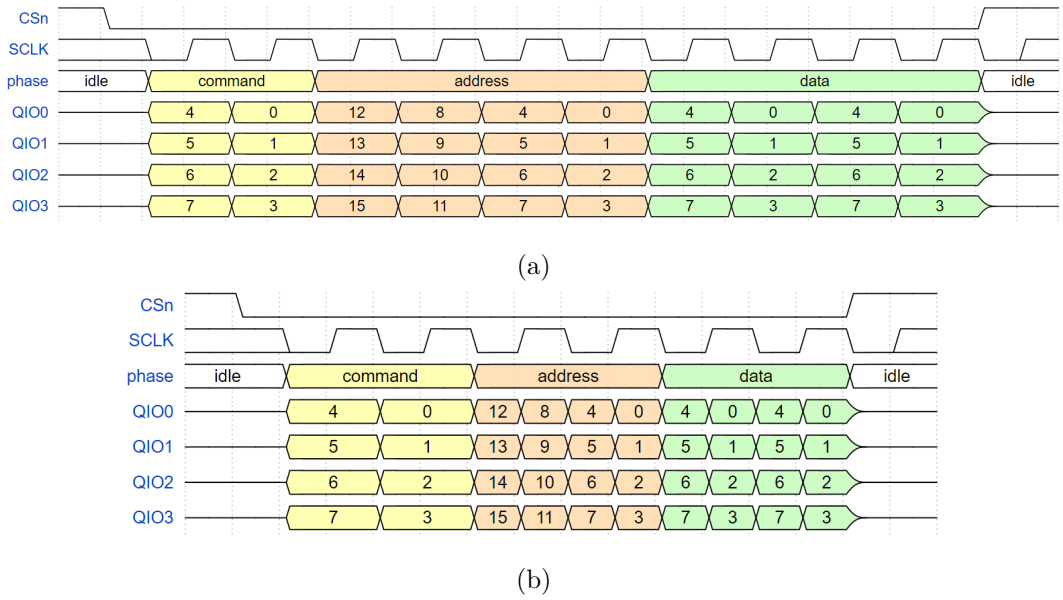


Figure 2.9: QSPI transaction in a) SDR and b) DDR mode.

2.2 RI5CY PULP

RI5CY PULP is 32-bit RISC V processor core with extended support of additional instructions that are not supported in standard RISC V ISA. Parallel Ultra-Low Power (PULP) adds core-specific extensions as post incrementing load and stores, multiply-accumulate extensions, ALU extensions, and hardware loops. A Load Store Unit (LSU) ensures communication with peripheral memories [2].

2.2.1 Load Store Unit

The LSU is used to access a data memory. The data can be loaded or stored either as words (32 bits), half-words (16 bits), or bytes (8 bits). The unit supports misaligned accesses, when access is not aligned on a natural word boundary. The unit needs at least two clock cycles to perform a misaligned access because operation is internally separated into two word-aligned accesses. The LSU also supports post-increment instructions. It ensures that during load/store operations the base address is incremented by specified offset to reduce the number of required instructions [2].

Protocol

The LSU is master-slave based protocol. The master device initiates transactions. LSU signals described in Table **2.2** are used for communication with a memory. Every transaction starts by setting a valid address at *data_addr_o* bus and *data_req_o* signal high. *Data_req_o* has to be kept high until *data_gnt_i* is set high for one clock cycle. It indicates that request is accepted, and address at *data_addr_o* bus can be changed in the next cycle. The *data_we_o* signal is used to indicate if write or read operation is performed. It can not change if *data_req_o* is set high. After the request is accepted, the slave sets read data at *data_rdata_i* bus and signal *data_rvalid_i* high to indicate that data are valid. Even during write transaction *data_rvalid_i* needs to be set high. The write data that should be written to the memory are sent through *data_wdata_o* bus at the same time *data_req_o* is sent.

Table 2.2: Master signals used by Load Store Unit [2].

Signal	Direction	Description
data_req	output	Request for operation, high until data_gnt is high for one cycle
data_addr	output	Address 32 bit
data_we	output	High - write Low - read
data_be	output	Byte enable, data size setting
data_wdata	output	Sent data to memory 32 bit
data_rdata	input	Data read from memory 32 bit
data_rvalid	input	Notify that valid data are hold at data_rdata
data_gnt	input	Memory accepted the request

The examples of transaction timing diagrams are shown in Figure **2.10**. It shows protocol differences based on speed of reply as well as back-to-back read transaction [2].

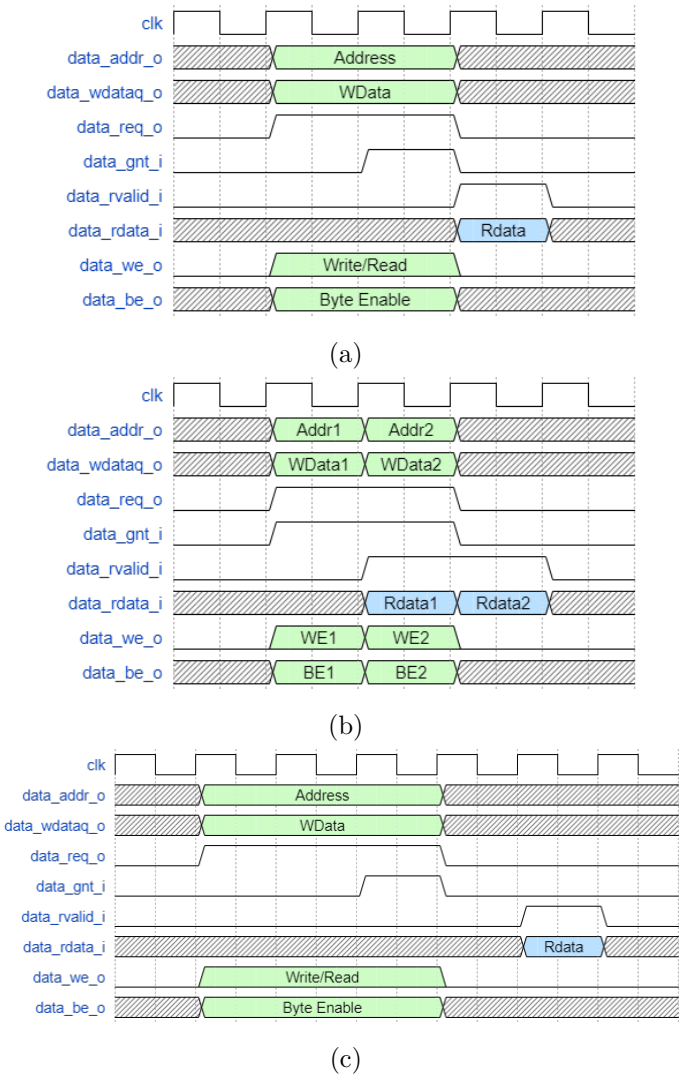


Figure 2.10: LSU a) basic, b) back to back, and c) slow response transaction [2].

2.3 AMBA AHB5 Protocol

The Advanced High-performance Bus (AHB) is a bus interface for a high clock frequency and high-performance designs. It is used as a connection between components of System-on-Chip (SoC). It implements series of features for high-performance systems like burst transfers, single clock edge operation, non-tristate implementation, locked transfers, error transfer termination, protection control, and wide data bus configurations from 32 bits up to 1024 bits. The protocol supports both little-endian and big-endian systems. Advanced Peripheral Bus (APB) protocol is used instead for slower devices. A bridge can be implemented to connect APB and AHB. The AHB supports both multiple master and slave devices. However, the bus does not have three-state character thus interconnect logic has to be implemented [7].

2.3.1 Interconnect Logic

The interconnect logic is used to provide arbitration and signal route control between multiple masters and slaves due to the two-state character of the bus. The most simple form of interconnect logic is in the case of one master and multiple slaves, where only a decoder and a multiplexor is needed. The decoder is used to decode addresses from the master. The decoder provides control signals for multiplexor and drives select signals for slaves according to the current address. The multiplexor connects read data from a proper slave to a master. An example block diagram of the AHB interface with a single master and multiple slaves with interconnect logic is shown in Figure **2.11**. In case of multi-master system, additional arbitration logic is needed [7].

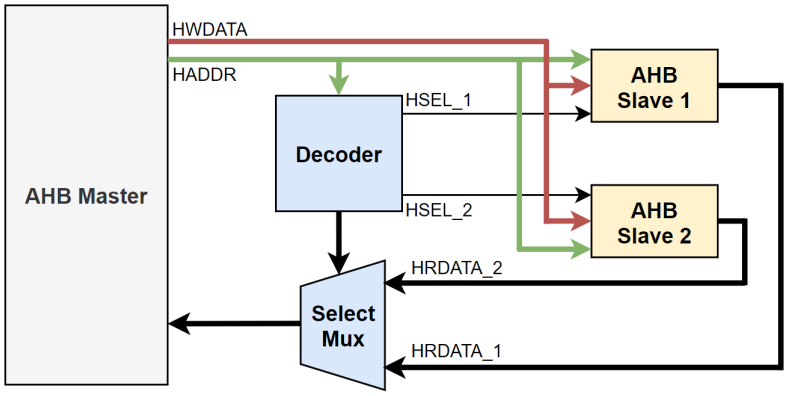


Figure 2.11: AHB interface block diagram with multiple slaves and interconnect logic [7].

2.3.2 Interface

A master device provides address, write data, and control signals to operate bus and interconnect logic. The simplified master device interface is shown in Figure **2.12**.

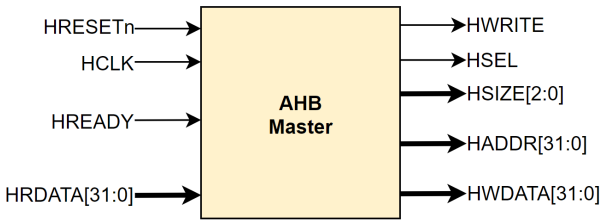


Figure 2.12: AHB master interface [7].

A slave device responds to a master according to provided address and control signals. Slaves are mostly memory devices or high speed peripherals. Interface of a slave device is shown in Figure **2.13**.

A simplified version can be used if advanced features of the bus are not needed. The most important protocol signals are described in Table **2.3** [7].

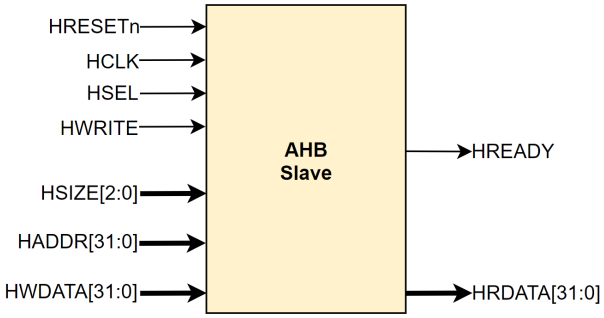


Figure 2.13: AHB slave interface [7].

Table 2.3: List of AHB interface signals with description [7].

Name	Source	Destination	Description
HCLK	clock source	master, slave	The bus clock, rising edge related timings
HRESETn	reset source	master, slave	The bus reset signal, active low
HADDR	master	slave, decoder	The 32-bit Address bus
HSIZE	master	slave	Indication of data size
HWDATA	master	slave	The n-bit data bus
HWRITE	master	slave	high - write transfer, low - read transfer
HRDATA	slave	multiplexor, master	The n-bit data bus
HREADY	slave	multiplexor, master	high - transfer has finished, low - extend data phase
HSELx	master, decoder	slave	Slave select signal

2.3.3 Basic Transfer Protocol

Every transaction is started by a master device by providing a valid address and by driving control signals that provide specific information to slaves about which type of transfer will be performed. A write transfer occurs if data are being transferred from master to slave otherwise it is read transfer. The basic read and write transactions are shown in Figure **2.14**. First, each transfer starts with the address phase that lasts a single clock cycle if it is not extended by the previous data phase. Second, the data phase follows, where data are provided by a master (write) or by a slave (read) at their corresponding data buses for one clock cycle. The master drives the *HWRITE* signal low for reads and high for writes to distinguish between operations. The address phase for the next transfer is in progress during current data phase [7].

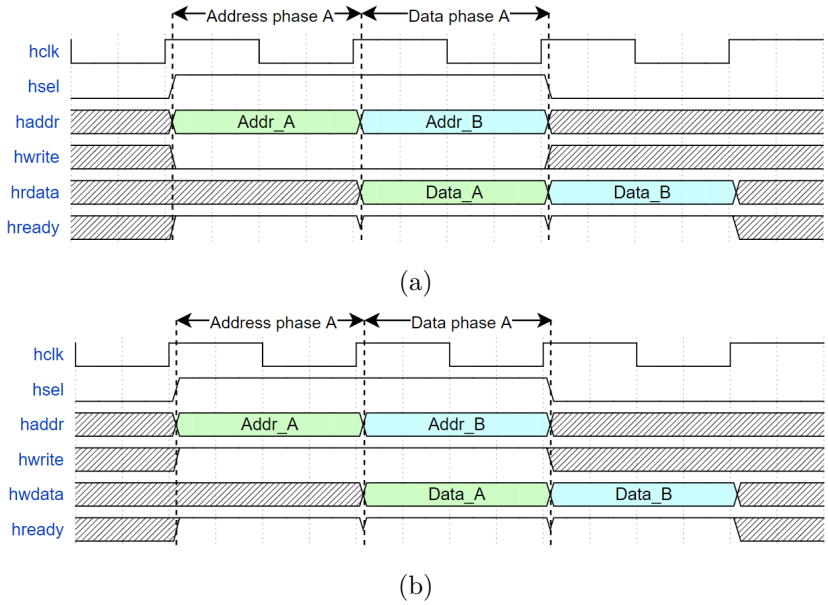


Figure 2.14: AHB basic a) read and b) write transactions [7].

The *HREADY* signal is used by a slave to request an extension of the data phase by a master. If a slave is not ready to receive data, it drives *HREADY* signal low as can be seen in Figure **2.15**. Read and write accesses can be combined during single burst transfer by alternating $HWRITE$ signal value between bursts.

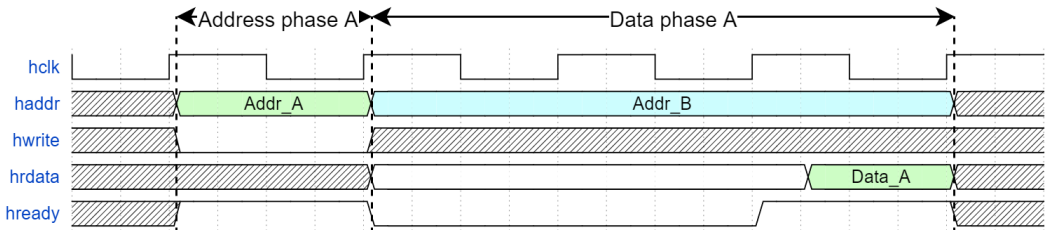


Figure 2.15: AHB read with extended data phase.

A master can indicate a bit size of data in the transfer by *HSIZE* signal. All possible transfer sizes are listed in Table **2.4**. Naturally, the upper limit is limited by used bus width [7].

Table 2.4: Transfer data size according to *HISIZE* signal [7].

HSIZE[2]	HSIZE[1]	HSIZE[0]	Bit size	Description
0	0	0	8	Byte
0	0	1	16	Halfword
0	1	0	32	Word
0	1	1	64	Doubleword
1	0	0	128	4 word line
1	0	1	256	8 word line
1	1	0	512	-
1	1	1	1024	-

3 QSPI Master Design

3.1 Protocol Features Selection

3.1.1 State of the Art

A variety of QSPI flash memories from different manufacturers were compared to decide which features shall be supported by the QSPI master design. Datasheets from Micron [12, 13], Macronix [10, 11], Winbond [5, 9], Cypres [1], NXP [8], and STM [3] were used as a reference. The objective was to study the QSPI protocol and point out deviations between them. Hardware parameters were also compared. Devices sometimes offer special features that are not part of the basic QSPI protocol, but they can improve the overall efficiency and usability of the protocol. Special features were taken into consideration while choosing prospective design features.

All compared QSPI flash memories shared almost the same definition of the QSPI protocol with differences in supported lengths of phases, support of DDR mode, and alternate byte phase. The maximal dummy phase length typically ranges from 8 up to 32 cycles, but no more than 16 cycles are needed in most cases. All memories support a 3-byte address phase. However, a 4-byte address phase is more often supported with the current increase in memory sizes. The alternate phase definition varies the most. Typically a master sends data of size from 1 bit up to 8 bits before the dummy phase. Maximal serial clock frequencies were compared, to show maximal theoretical data throughputs and speed of the interface. Typical maximal values of frequencies in SDR mode vary between 104 MHz and 166 MHz, and in DDR mode between 66 MHz and 100 MHz. Some instructions can operate only at much lower frequencies. The timing of the CS signal after the transaction end can vary as a result of different clock frequencies. This period is called Chip Select High Time. It determines how long the CS should stay high before the next transfer. Typically it can last from one up to several clock cycles [1, 3, 5, 8, 9, 10, 11, 12, 13].

Special Features

All compared QSPI flash memories have several special functions that can be implemented as an addition to the basic protocol. Almost all of them support HOLD, Write Protection (WP), RESET, and Execute In Place (XIP). A less common feature is Send Instruction Only Once (SIOO). The WP and HOLD are only used in DSPI and SSPI modes. Usually, the HOLD is mapped to the $QIO3$ pin and the WP to the $QIO2$ pin. The HOLD is set low to hold transactions while the CS stays high. It is used if more slaves share the same QSPI data lines. The WP signal prevents writes accesses to the status registers if set high. Example of typical timing diagrams for the WP and the $HOLD$ signals are shown in Figure **3.1**. Some memories have multiplexed HOLD and RESET functionality at the $QIO2$ pin.

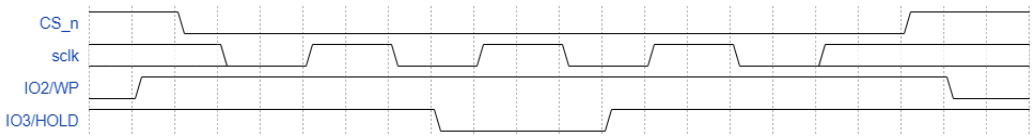


Figure 3.1: WP and HOLD timing.

The XIP mode allows the master to access memory only with the requirement of address without need of commands. The SIOO function ensures that the command phase is transmitted only once in the non-sequential transfer, as shown in Figure **3.2**. This function is usually set/reset during the command or alternate phase [1, 3, 5, 8, 9, 10, 11, 12, 13].

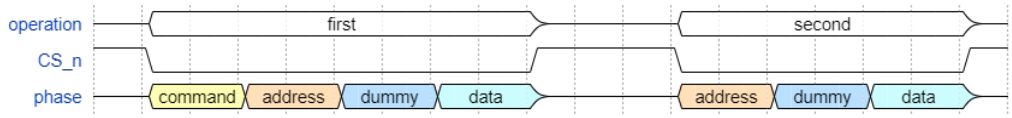


Figure 3.2: Non seqentual transfers with SIOO enabled.

The summary Table **3.1** was created to show the essential features of compared QSPI flash memories and chosen protocol functions for the QSPI master design. Features that are supported by all QSPI devices like DSPI and SSPI are not mentioned in the table.

Table 3.1: QSPI flash memories list of features [1, 3, 5, 8, 9, 10, 11, 12, 13].

QSPI role	manufacturer	device name	description	DDR	alternate phase	address phase length (bytes)	dummy phase length (clock cycles)	data phase length (bytes)	SIOO	XIP	maximal serial clock frequency
slave	Winbond	W25Q128FV	3V 128Mbit flash memory	no	yes	1 - 3	8/16/24/32	1 -256	no	yes	104
		W25Q32JW_DTR	1.8V 32Mbit flash memory	yes	yes	1- 3	8/16/24/32	1 -256	no	yes	133 / DDR 66
	Micron	N25Q128	3V 128Mbit flash memory	no	no	1- 3	1 -15	1 -256	no	yes	108
		MT25QU512ABB	1.8V 512Mbit NOR flash memory	yes	yes	1- 4	1 -15	1-256	no	yes	166 / DDR 90
	Cypress	S79FL01GS	3V 128Mbit flash memory	yes	yes	1- 4	1 -24	1 - 512	yes	yes	104 / DDR 80
	Macronix	MX35LF2G14AC	3V 2G-bit NAND Flash Memory	no	no	1- 3	8	1-2048	no	no	104
		MX66L1G45G	3V 1G-bit NAND Flash Memory	yes	yes	1- 4	1 -10	1-256	yes	no	166 / DDR 83
master	STMicroelectonics	AN4760	Quad-SPI interface on STM32	yes	yes	1- 4	1 -31	1-∞	yes	yes	133 / DDR 100
	NXP	AN4512	QSPI master module	yes	yes	1-4	1-8	1-∞	no	yes	104
QSPI master design				yes	yes	1- 4	1 -31	511	yes	yes	50 - 100

3.1.2 Selection of Protocol Features

The QSPI master design has to support a flexible frame format to anticipate its all possible combinations. Thus the user should be able to set length and data modes for each phase.

Supported data modes are QSPI/DSPI/SSPI and SDR/DDR. The 4-byte address phase will be implemented as 4-byte addressing is more common nowadays. The design will support dummy phase length from 1 up to 31 cycles. It should be sufficient for most commands. Maximal data phase length of 511 bytes was chosen, considering that most memories use 256-byte page size. The alternate phase will allow to send from 1 bit up to 8 bits. The design shall be able to change logical values of $QIO2$ and $QIO3$ data lines while in SSPI or DSPI modes to support WP and HOLD functions. The clock prescaler of 1/2/4/8 division will be used to allow a user to slow down the output serial clock. Memory-mapped mode with SIOO function will be supported to allow the execution of code via the QSPI interface. Memory transfer caused by an instruction fetch of an microcontroller unit (MCU) will be translated to read access on the QSPI interface. Alternatively, if MCU contains a cache, the interface will react to requests for fetching cache lines, which are issued by the cache controller. The summary of all supported features with descriptions is listed in Table **3.2**.

Table 3.2: QSPI master design features list with description and configurations.

Feature	Description	Configuration
command phase	The command is sent.	0/8 bits length
address phase	The address is sent.	0/1/2/3/4 bytes length
alternate phase	The alternate byte or byte nibble is sent.	0-8 bits length
dummy phase	Dummy cycles with no actual data are transmitted.	0-31 cycles
data phase	Data are read or sent.	0-511 bytes length
SSPI and DSPI data modes	Use only one or two data lines for transfer.	It can be set separately for each phase. The alternate phase shares a setting with the address phase.
DDR mode	Data are transferred on both edges of the clock signal.	It can be set separately for each phase except for the command phase. The alternate phase shares a setting with the address phase.
SCLK prescaler	Prescaler to divide the system clock to QSPI clock.	0/2/4/8 clock division
SCLK mode	SPI clock polarity and phase setting	0/1 mode
QIO2 and QIO3 out	It controls QIO2 and QIO3 data lines when they are not used. It can be used WP and HOLD functions.	0/1
HIZ dummy	It controls whether the dummy phase is transmitted as high Z or as 0.	0/Z
CS high time	It defines a minimal number of clock cycles after the end of transfer during which the CS signal has to stay in a high state.	1-8 cycles
XIP	Access to memory only with the requirement of address with no need for commands.	-
SIOO	Send instruction only once.	-
memory mapped mode	Direct access of read initiated by the internal LSU PULP bus.	-

3.2 System Level Design

The design is aimed to create a QSPI master IP core with the following features:

– Easy to integrate with the RISCV processor with the PULP bus.

– Support all protocol features as discussed in subsection 3.1.2.

The QSPI master interface is shown in Figure **3.3**. It can be observed that two PULP LSU interfaces are used. The PULP LSU is used to connect the RISCV processor to the memory space. The first PULP bus (PPI) is intended for basic operations. The second (PCI) is used for memory-mapped mode only. The QSPI interface is composed out of four data input/output signals, the serial clock, chip select signal, and four output enable signals.

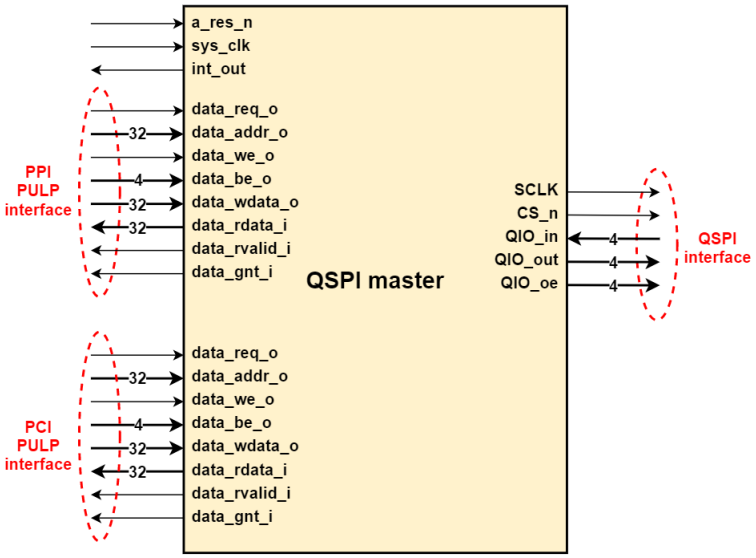


Figure 3.3: The QSPI master design interface.

The design can be operated in different scenarios. Three basic scenarios were chosen for simplification – basic read, basic write, and memory-mapped mode. A typical process for each scenario is described in Table **3.3**. Configuration registers must be set before the start of the transfer as it can be seen from each scenario. Data need to be written to the TX FIFO while in write mode otherwise protocol controller will hold the transaction. Both FIFOs are observed for an empty or full status flag in order to write data to TX FIFO and read data from RX FIFO.

3.2.1 Block Level Design

The QSPI master was decomposed into smaller functional blocks. The Block diagram of the system is shown in Figure **3.4**. PULP slaves convert the PULP bus to internal AHB. The AHB is used to control configuration registers and the RX FIFO in the memory-mapped mode. The conversion bridge is used between RX FIFO and PULP slave. Configuration registers are used to store configuration data for QSPI protocol. Two FIFOs are implemented

Table 3.3: Three typical QSPI transaction scenarios described in steps.

ACTIONS ON INTERFACES							
BASIC MODE WRITE							
PPI	Setting up Configuration registers	Insert data into TX FIFO	Initiation of basic mode request			Insert remaing data into TX FIFO	
QSPI master				Transmits all set phases	If during data phase of QSPI protocol TX FIFO becomes empty, than QSPI master stall transaction and wait for new data	Resume transaction	Transaction ends and done interrupt is generated
BASIC MODE READ							
PPI	Setting up Configuration registers	Initiation of basic mode request			Reads data from RX FIFO		Reads out rest of data from RX FIFO
QSPI master			Transmits all set phases	If during data phase of QSPI protocol RX FIFO becomes full, than QSPI master stall transaction and wait	Resume transaction	Transaction ends and done interrupt is generated	
MEMORY MAPPED MODE							
PPI	Setting up Configuration registers	Switch to memory mapped mode					
PCI			Read transaction on PCI bus, initiation of memory mapped request	Transaction is hold while RXFIFO is empty	Transaction is resumed	Reads out rest of data from RX FIFO	
QSPI master				Transmits all set phases	Data are being read and written to RX FIFO	Transaction ends and done interrupt is generated	

STEPS

to buffer RX/TX data from/to QSPI protocol. The QSPI protocol controller block is designed to operate the QSPI protocol. Each sub-block was designed to fulfill the selected protocol functions discussed in the subsection 3.1.2.

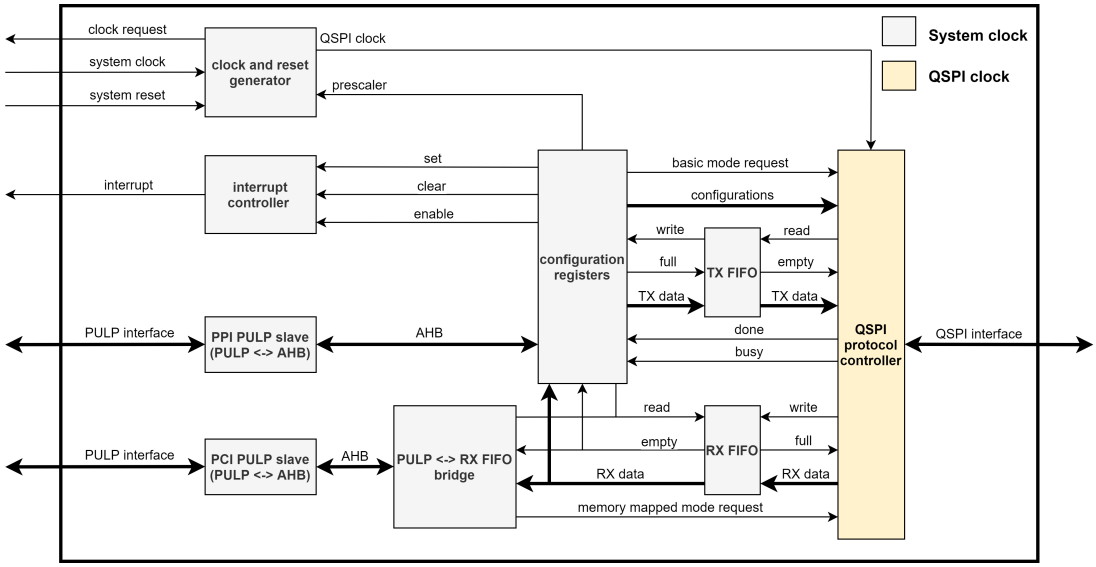


Figure 3.4: The block diagram of the QSPI master design.

3.3 Blocks Description and RTL Implementation

The objective of this section is to provide a detailed description of each sub-block. Functionality and implementation of each sub-block is described. Each block of the design was implemented on the Register-transfer level (RTL) in Very High Speed Integrated Circuit Hardware Description Language (VHDL). RTL codes are not included in this chapter. However, they are included in the appendix chapters only for printed version of this text. Assertions were written in Property Specification Language (PSL) to help with debugging on the RTL level.

3.3.1 Clock and Reset Generator

Clock Generator

The clock generator block was implemented to generate the QSPI protocol clock and a request for the system clock. The clock structure of the design is implemented as synchronous with only one input system clock. The design is intended for clock frequencies from 50 MHz up to 100 MHz. The prescaler was used to create the QSPI protocol clock with a lower frequency than the system clock.

The block diagram of implementation is shown in Figure **3.5**. The system clock request is generated by OR logical function between all sub-block requests. Thus result request is active while at least one sub-block request is also active. The request is assumed glitch-free behavior for each sub-block.

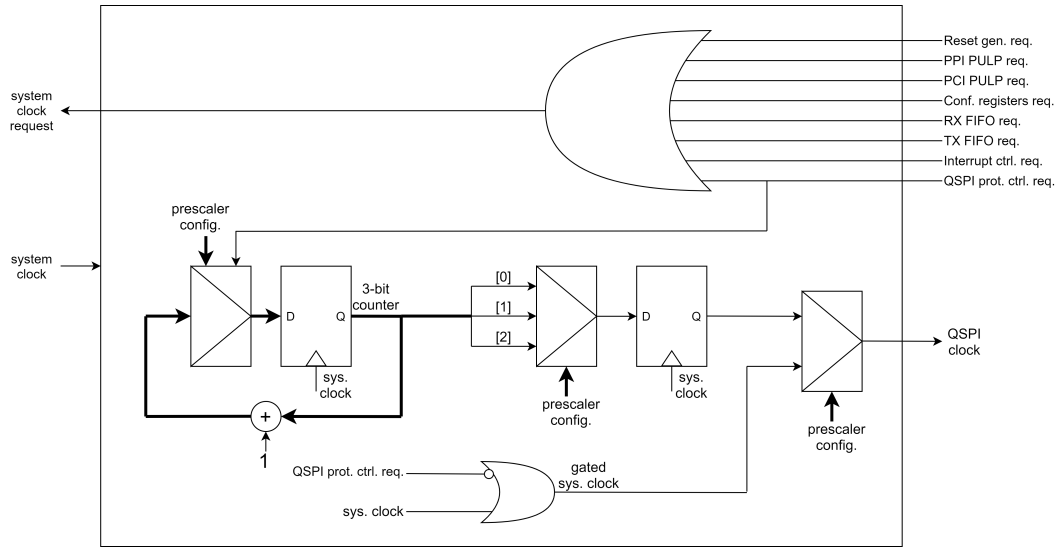


Figure 3.5: Block diagram of the clock generator.

The prescaler divides the system clock by 1/2/4/8 to generate the QSPI protocol clock. The value for a division is set in the configuration registers block. It is assumed that the prescaler value can not change while the transaction is active. The prescaler is not active when the prescaler is set to 1. In this case the QSPI clock is implemented as a direct copy of the system clock gated by the OR gate. Otherwise, a 3-bit counter is used for dividing by 2/4/8. The counter counts up by one on each rising edge while it is active. Each divided clock signal is represented by one bit of the counter (/2 – first bit LSB, /4 – second bit, /8 – third bit). The timing diagram is shown in the Figure **3.6**. It describes the intended functionality of the block. Clock gating is used to disable the clock signal if it is not needed to lower power consumption.

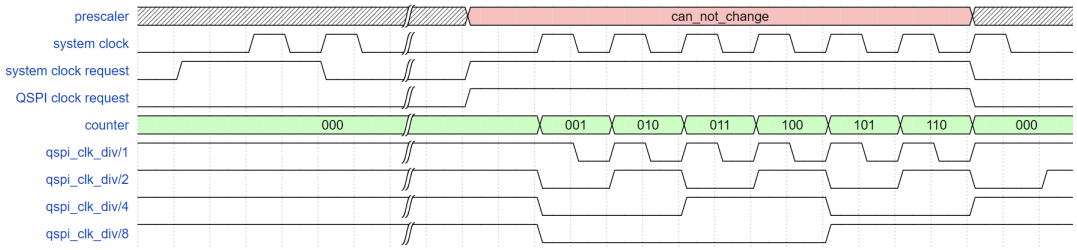


Figure 3.6: Timing diagram of the clock generator.

Reset Generator

A reset generator was implemented to handle the input global reset. It is assumed that the input global reset is asynchronous to the system clock, therefore synchronization of the release was needed. The Synchronization is done by two registers, as shown in Figure **3.7** [15]. It can be seen that the block is requesting a clock when the reset is in an active state. Otherwise, internal clock gating is used to disable a clock signal if it is not needed. The block also generates reset for the QSPI protocol controller and both FIFOs from system reset and the QSPI enable signal to keep mentioned blocks in the reset state while the QSPI interface is disabled. The timing diagram is shown in Figure **3.8**. It describes the functionality of the block [16].

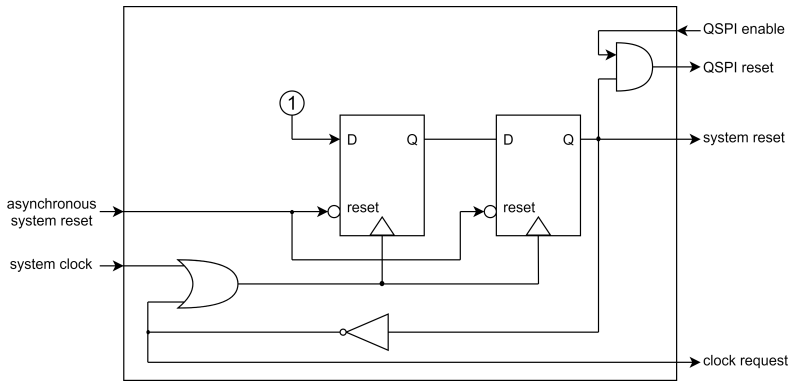


Figure 3.7: Block diagram of the reset generator.

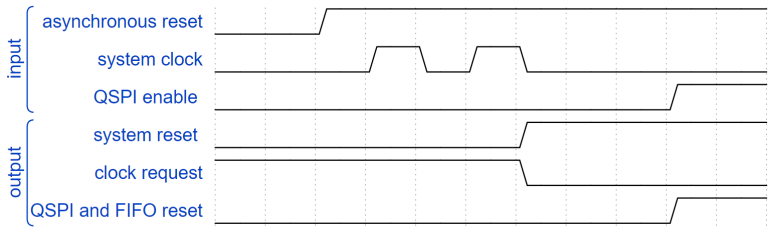


Figure 3.8: Timing diagram of the reset generator.

3.3.2 LSU PULP Slave

The PULP slave block was designed to convert the PULP LSU to/from the AHB protocol. The block acts as a PULP slave for RISCV PULP master and an AHB master for internal AHB. The block diagram of implementation is shown in Figure **3.9**. Both protocols were described in section 2.3 and section 2.2.

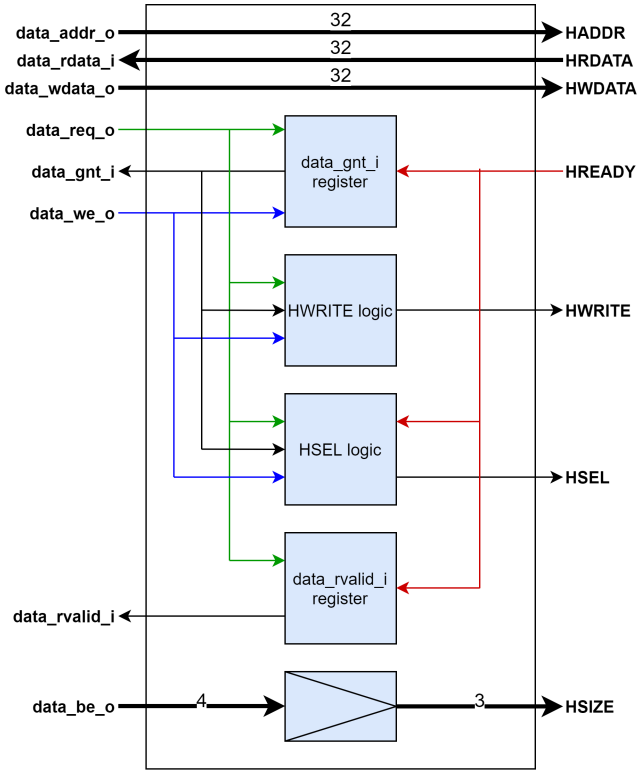


Figure 3.9: Block diagram of the LSU PULP slave.

The *HADDR* signal is connected as a direct copy of the *data_addr_o* signal during both write and read operations. The same goes for *HWDATA* and *data_rdata_i*, as they are directly driven by *data_wdata_o* and *HRDATA*. The *data_be_o* is converted to the *HSIZE* according to Table **3.4**. Unaligned accesses are not supported.

Table 3.4: *Data_be_o* and *HSIZE* conversion table.

data_be_o	*HSIZE*	Transfer size
0001, 0010, 0100, 1000	000	8 bits
0011, 1100	001	16 bits
1111	010	32 bits

Read access on the PULP and the AHB is in principle similar. Therefore only one clock cycle is needed to read data except for the first phase, as shown in Figure **3.10**. A transaction can be held by driving the *HREADY* signal low for read operations, as shown in Figure **3.11**.

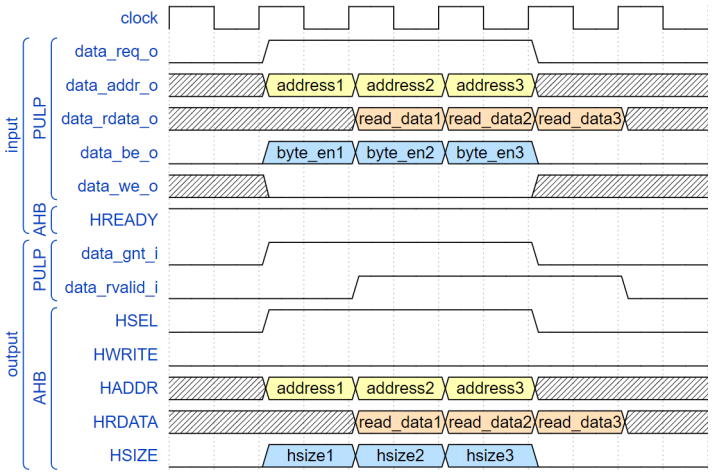


Figure 3.10: Timing diagram of read access on PULP slave.

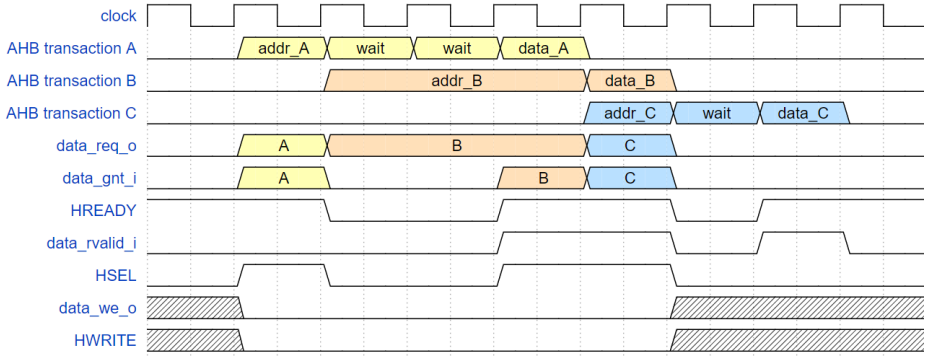


Figure 3.11: Timing diagram of read access on PULP slave with hold.

Write access needs two clock cycles because PULP protocol requires to acknowledge data by the *data_gnt_i* as shown in Figure **3.12**.

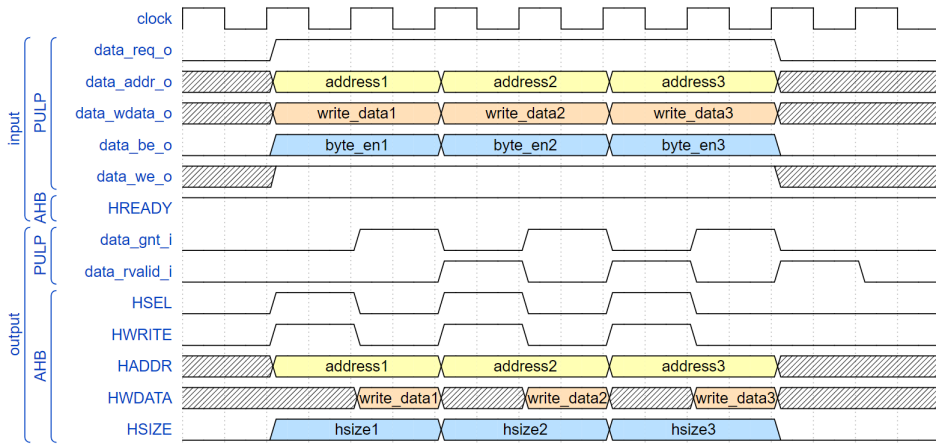


Figure 3.12: Timing diagram of write on PULP slave.

Read Access

The *data_gnt_o* is driven by *data_req_o* if the *HREADY* is high. Otherwise, it is set low. The *HWRITE* is kept low. The *HSEL* is a direct copy of *data_req_o* unless the *HREADY* is low. The *data_rvalid_i* is set high if *HREADY* is high.

Write Access

The *HREADY* is not used to hold transactions during write access because it could block the bus for writing to the register-map. The *data_gnt_o* is driven by an inverted value of the registered *data_req_o* signal, while *data_req_o* is active. The *HWRITE* has always inverted the value of *data_gnt_i* while *data_req_o* is high. The same applies to the *HSEL*. The *data_rvalid_i* is driven same as during read access.

3.3.3 Configuration Registers

The block is used to store configuration data for the QSPI protocol controller and to allow read, enable, and clear interrupt and status flag registers. A block diagram is shown in Figure **3.13**. It can be observed that the core of the block is formed by the tool generated register map with access through AHB. Each register has 32-bit size. Registers were generated to support all configurations for QSPI features, as discussed in subsection 3.1.2. Altogether ten 32-bit registers were generated. Configuration registers (QSPICFG0, QSPICFG1, QSPIMODE, QSPIINTSTR0, QSPIINTSTR1) are read/write type except for transaction request register. The write-only register is implemented for the basic mode transaction request register. The register is cleared at the moment when the QSPI controller stops being busy. Interrupt and status registers are used to notify a user about the current state of the device. Status registers (QSPISTATUS) are read-only thus can not be cleared or rewritten by the user. Interrupt registers are divided into interrupt write-only enable registers (QSPINTENA) and read-only interrupt status registers (QSPINTSTS). Interrupt status registers are cleared

by writing one to the register. To write/read to/from TX/RX FIFO registers (QSPIWDATA, QSPIRDATA) are implemented as write-only/read-only type registers and act only as a bridge between AHB and FIFOs.

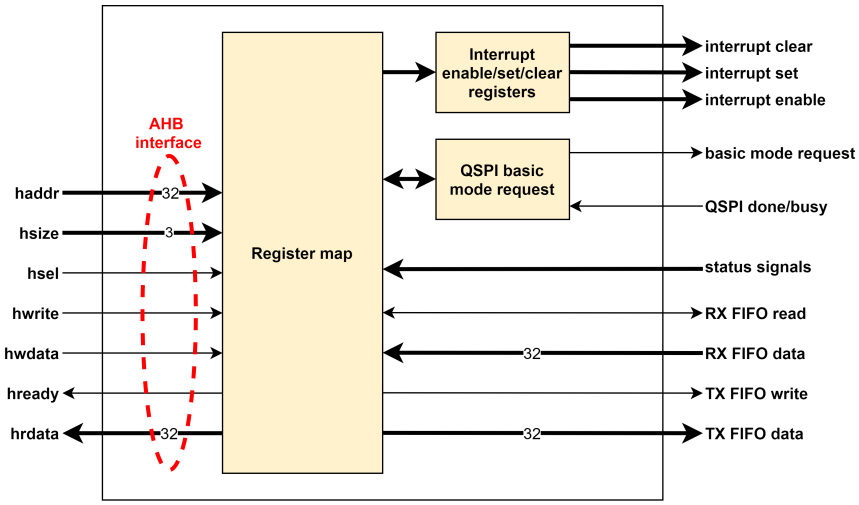


Figure 3.13: Configuration registers block diagram.

Registers

QSPICFG0

31	30	29 28	27	26 25 24	23 22 21 20 19 18 17 16
qspi_ req	qspi_ mode	qspi_ prescaler	qspi_ sioo	qspi_cs_ht	qspi_data_length
W	R/W	R/W	R/W	R/W	R/W

15	14	13 12 11	10 9 8	7 6 5 4 3	2	1	0
qspi_ data_ length	qspi_dummy_length		qspi_cfg_length	qspi_addr_length	cmd_ length	qspi_ wr	qspi_ en
R/W	R/W		R/W	R/W	R/W	R/W	R/W

Figure 3.14: QSPICFG0 register bit map.

[31] **qspi__req**

Initiates the Basic mode request only if the QSPI enable bit is set to '1' and device is set to Basic mode.

This bit is automatically cleared by hardware after the transaction is done.

[30] **qspi__mode**

Select device mode:

0 - Basic mode

1 - Memory Mapped Mode

This bit can be modified only if QSPI is not in a busy state.

[29:28] **qspi_prescaler**

Prescaler to divide system clock to QSPI clock:

00 - Do not divide

01 - Divide by 2

10 - Divide by 4

11 - Divide by 8

The field can be modified only if QSPI is not in a busy state.

[27] **qspi_sioo**

Send instruction only once. When enabled, the Command phase will be sent only during the first QSPI transfer while device in Memory Mapped mode.

This bit can be modified only if QSPI is not in a busy state.

[26:24] **qspi_cs_ht**

Define a minimal number of clock cycles after the end of the transaction, during which the chip select signal stays high before the next transaction.

0x0 - 1 clock cycle

...

0x7 - 8 clock cycles

The field can be modified only if QSPI is not in a busy state.

[23:15] **qspi_data_length**

Number of bytes sent during the Data phase of the QSPI transaction.

0x0 - no data bytes

...

0x1FF - 511 data bytes

The field can be modified only if QSPI is not in a busy state.

[14:10] **qspi_dummy_length**

Number of clock cycles in the Dummy phase of the QSPI transaction.

0x0 - no dummy cycles

...

0x1F - 31 dummy cycles

The field can be modified only if QSPI is not in a busy state.

[9:6] **qspi_cfg_length**

Number of bits in the Alternate/Config. phase of the QSPI transaction.

0x0 - no Alternate phase

...

0x8 - 8 bits

The field can be modified only if QSPI is not in a busy state.

[5:3] **qspi_addr_length**

Number of bytes in the Address phase of the QSPI transaction.

0x0 - no Address phase

0x1 - 1 Byte

0x2 - 2 Bytes

0x3 - 3 Bytes

0x4 - 4 Bytes

The field can be modified only if QSPI is not in a busy state.

[2] **qspi_cmd_length**

1 - Command byte is sent.

0 - Command byte is not sent.

This bit can be modified only if QSPI is not in a busy state.

[1] **qspi_wr**

For the Basic mode data phase:

1 - write

0 - read

For Memory Mapped mode, it's not relevant - read is always performed.

[0] **qspi_en**

QSPI interface enable bit. Any QSPI transactions are initiated only if this bit is set to '1'.

If set to '0' then the QSPI interface is kept in the reset state.

QSPICFG1

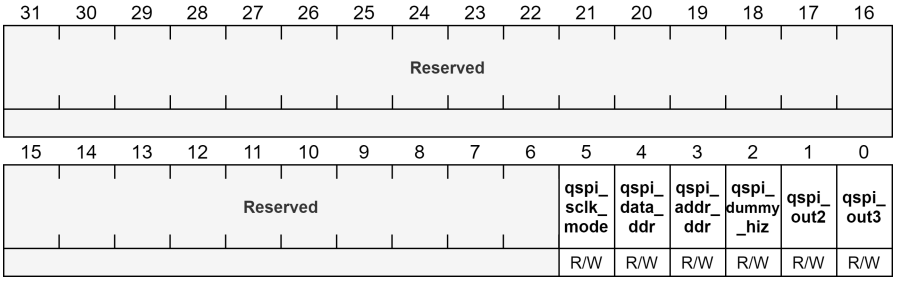


Figure 3.15: QSPICFG1 register bit map.

[5] **qspi_sclk_mode**

SPI clock mode idle state of the QSPI clock

0 - mode 0 (idle low)

1 - mode 1 (idle high)

This bit can be modified only if QSPI is not in a busy state.

[4] **qspi_data_ddr**

Data bytes are received/sent in the DDR mode.

This bit can be modified only if QSPI is not in a busy state.

[3] **qspi_addr_ddr**

Address bytes are sent in the DDR mode.

This bit can be modified only if QSPI is not in a busy state.

[2] **qspi_dummy_hiz**

Controls whether the dummy phase is transmitted as High Z or as O.

0 - 'Z'

1 - '0'

This bit can be modified only if QSPI is not in a busy state.

[1] **qspi_out2**

Controls QIO[2] data line when is not used by QSPI protocol.

This bit can be modified only if QSPI is not in a busy state.

[0] **qspi_out3**

Controls QIO[3] data line when is not used by QSPI protocol.

This bit can be modified only if QSPI is not in a busy state.

QSPIMODE

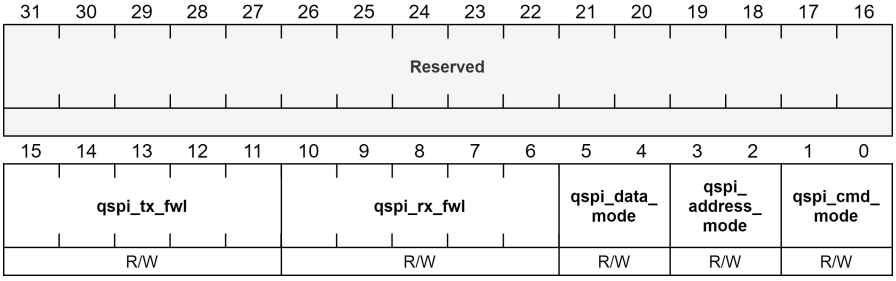


Figure 3.16: QSPIMODE register bit map.

[15:11] **qspi_tx_fwl**

TX FIFO Watermark level.

This field can be modified only if QSPI is not in a busy state.

[10:6] **qspi_rx_fwl**

RX FIFO Watermark level.

This field can be modified only if QSPI is not in a busy state.

[5:4] **qspi_data_mode**

Mode used during the QSPI data phase:

0x00 - SSPI

0x01 - DSPI

0x10 - QSPI

0x11 - reserver

This field can be modified only if QSPI is not in a busy state.

[3:2] **qspi_address_mode**

Mode used during the QSPI address phase:

0x00 - SSPI

0x01 - DSPI

0x10 - QSPI

0x11 - reserver

This field can be modified only if QSPI is not in a busy state.

[1:0] **qspi_cmd_mode**

Mode used during the QSPI command phase:

0x00 - SSPI

0x01 - DSPI

0x10 - QSPI

0x11 - reserver

This field can be modified only if QSPI is not in a busy state.

QSPIINTSTR0

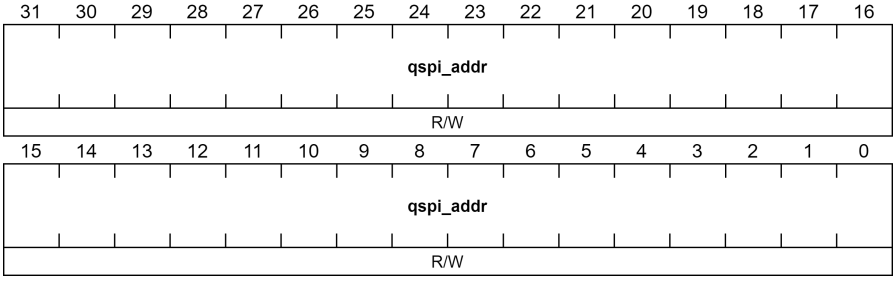


Figure 3.17: QSPIINTSTR0 register bit map.

[31:0] qspi_addr

Address sent during the QSPI Address phase. Relevant only in Basic mode.

This field can be modified only if QSPI is not in a busy state.

QSPIINTSTR1

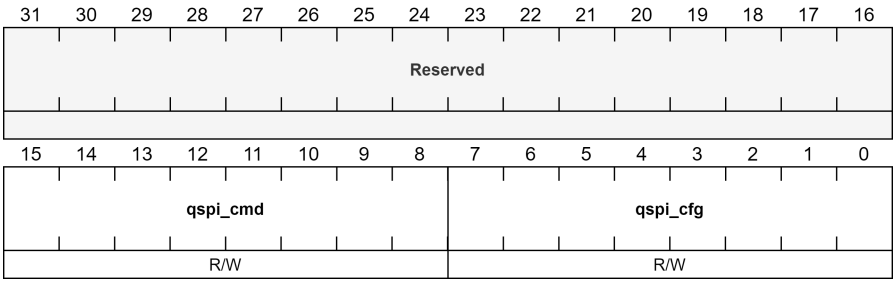


Figure 3.18: QSPIINTSTR1 register bit map.

[15:8] qspi_cmd

Command sent during the QSPI Command phase.

This field can be modified only if QSPI is not in a busy state.

[7:0] qspi_cfg

Byte sent during the QSPI Alternate phase.

This field can be modified only if QSPI is not in a busy state.

QSPIWDATA

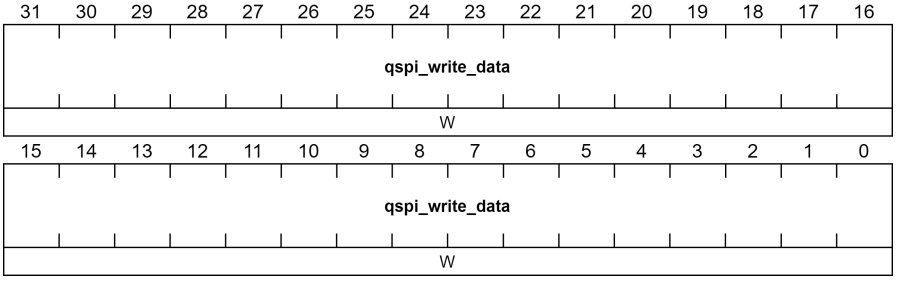


Figure 3.19: QSPIWDATA register bit map.

[31:0] qspi_write_data

Write data to TX FIFO.

32 bit write always expected

Data can be written only if the QSPI interface is enabled.

QSPIRDATA

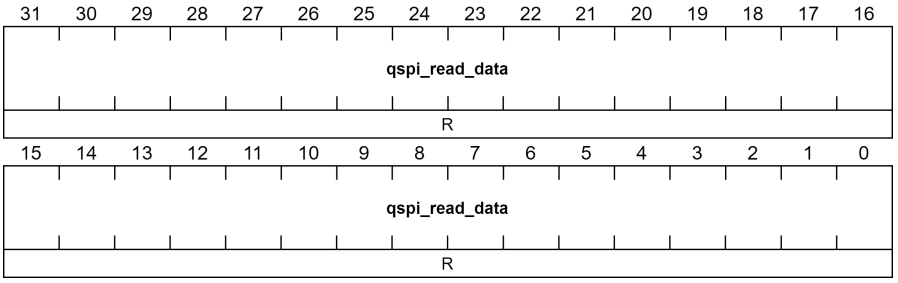


Figure 3.20: QSPIRDATA register bit map.

[31:0] qspi_read_data

Read data from RX FIFO.

32 bit read always expected

Data can be written only if the QSPI interface is enabled.

QSPISTATUS

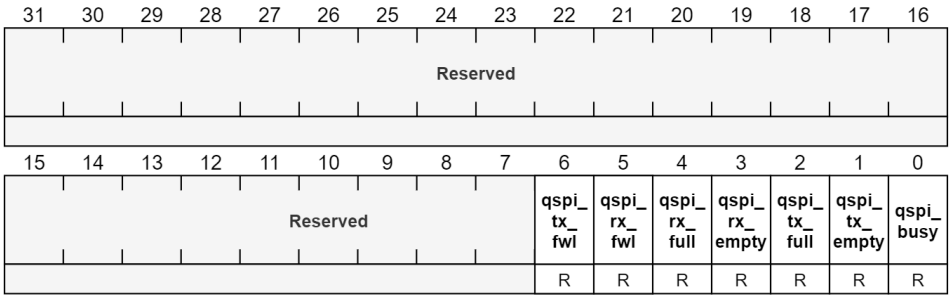


Figure 3.21: QSPISTATUS register bit map.

[6] qspi_tx_fwl
TX FIFO watermark level reached status

[5] qspi_rx_fwl
RX FIFO watermark level reached status

[4] qspi_rx_full
RX FIFO full status

[3] qspi_rx_empty
RX FIFO empty status

[2] qspi_tx_full
TX FIFO full status

[1] qspi_tx_empty
TX FIFO empty status

[0] qspi_busy
QSPI transaction is in progress status

QSPINTSTS

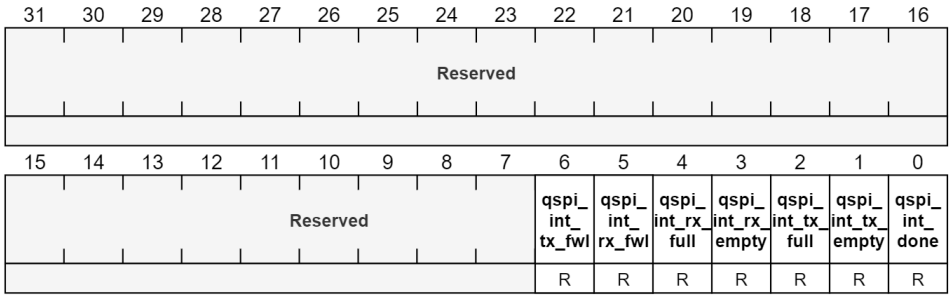


Figure 3.22: QSPINTSTS register bit map.

[6] qspi_int_tx_fwl
TX FIFO watermark level reached interrupt.
Clear by writing '1'.

[5] qspi_int_rx_fwl
RX FIFO watermark level reached interrupt.
Clear by writing '1'.

[4] **qspi_int_rx_full**

RX FIFO full interrupt.

Clear by writing '1'.

[3] **qspi_int_rx_empty**

RX FIFO empty interrupt.

Clear by writing '1'.

[2] **qspi_int_tx_full**

TX FIFO full interrupt.

Clear by writing '1'.

[1] **qspi_int_tx_empty**

TX FIFO empty interrupt.

Clear by writing '1'.

[0] **qspi_int_done**

QSPI transaction is finished interrupt.

Clear by writing '1'.

QSPINTENA

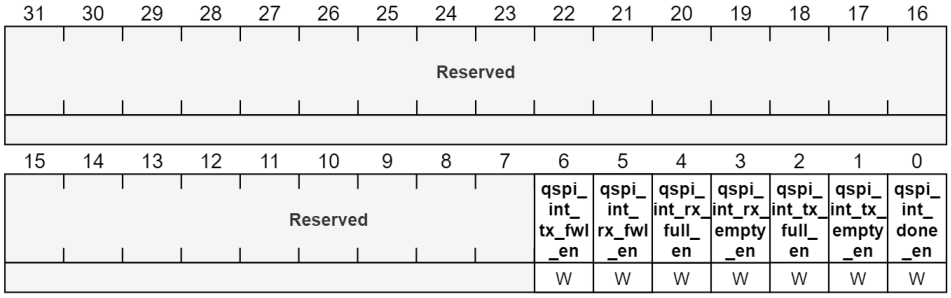


Figure 3.23: QSPINTENA register bit map.

[6] **qspi_int_tx_fwl_en**

TX FIFO watermark level reached interrupt enable.

[5] **qspi_int_rx_fwl_en**

RX FIFO watermark level reached interrupt enable.

[4] **qspi_int_rx_full_en**

RX FIFO full interrupt enable.

[3] **qspi_int_rx_empty_en**

RX FIFO empty interrupt enable.

[2] **qspi_int_tx_full_en**

TX FIFO full interrupt enable.

[1] **qspi_int_tx_empty_en**

TX FIFO empty interrupt enable.

[0] **qspi_int_done_en**

QSPI transaction is finished interrupt enable.

3.3.4 PULP Slave and Rx FIFO Bridge

The bridge was needed to connect the AHB bus of PCI PULP slave and RX FIFO. The objectives of the block are to be able to read from RX FIFO via AHB, notify PCI PULP that transaction should be held while RX FIFO is empty, to generate memory-mapped QSPI transaction request, load address for QSPI transaction, and to multiplex read signals from PCI PULP and PPI PULP depending on the selected QSPI mode. The block diagram is shown in Figure **3.24**.

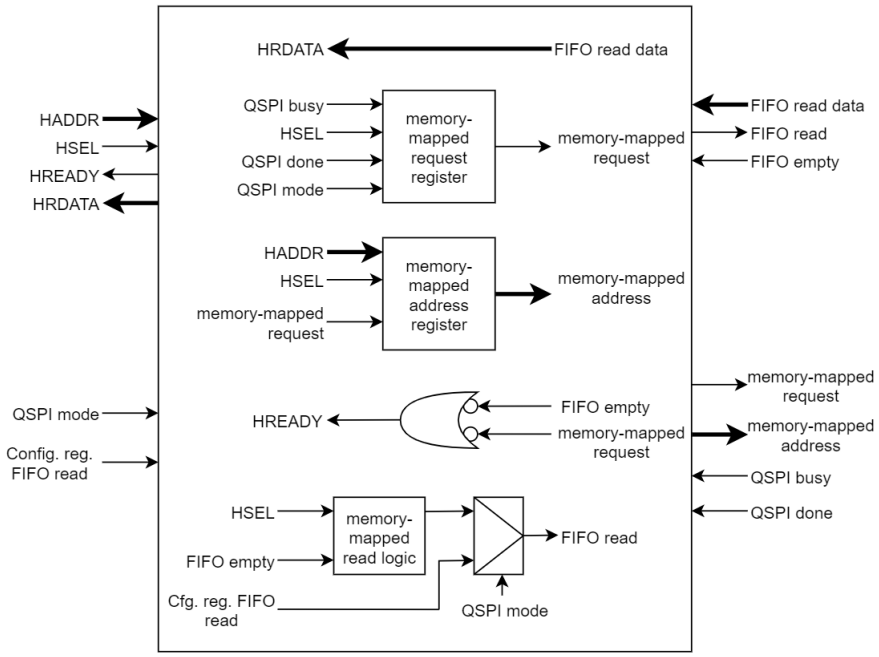


Figure 3.24: Block diagram of PULP slave and RX FIFO bridge.

A register for the memory-mapped request was implemented. The memory-mapped request is set high if the device is in memory-mapped mode, QSPI protocol controller is not in a busy state, and transaction on the AHB is active. The register is kept high while the QSPI interface is busy. It is reset if a transaction done signal from the QSPI interface is active. Bytes for the address phase needs to be obtained. Thus first address on the AHB bus is registered exactly before the transaction becomes active. The *HREADY* signal is set low if the Rx FIFO is empty. The PPI PULP and the PCI PULP read signals are multiplexed for RX FIFO according to set QSPI mode to ensure that both PULP slaves are able to read RX FIFO. Example timing diagram of read operation from RX FIFO during memory-mapped mode can be seen in Figure **3.25**.

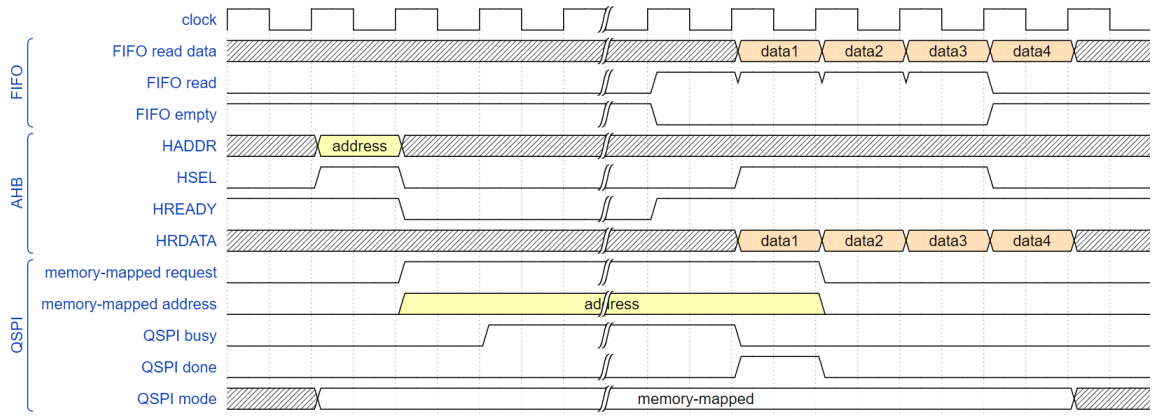


Figure 3.25: Timing diagram of PULP slave and RX FIFO bridge with memory-mapped read operation.

3.3.5 Tx and Rx FIFO

Both FIFOs are used to buffer data between PULP and QSPI interface. The 32-bit data width was chosen due to the 32-bit size of the PULP data signal. A generic variable sets the depth of FIFOs with supported widths, which equals 2^n, where n is a natural number. The block was designed as synchronous. FIFO also supports watermark reach, empty, full status flags and write/read operations in one clock cycle. The block diagram is shown in Figure **3.26**. The design is decomposed into general FIFO with dual-port RAM and wrapper with design specific functions. An example timing diagram is shown in Figure **3.27**.

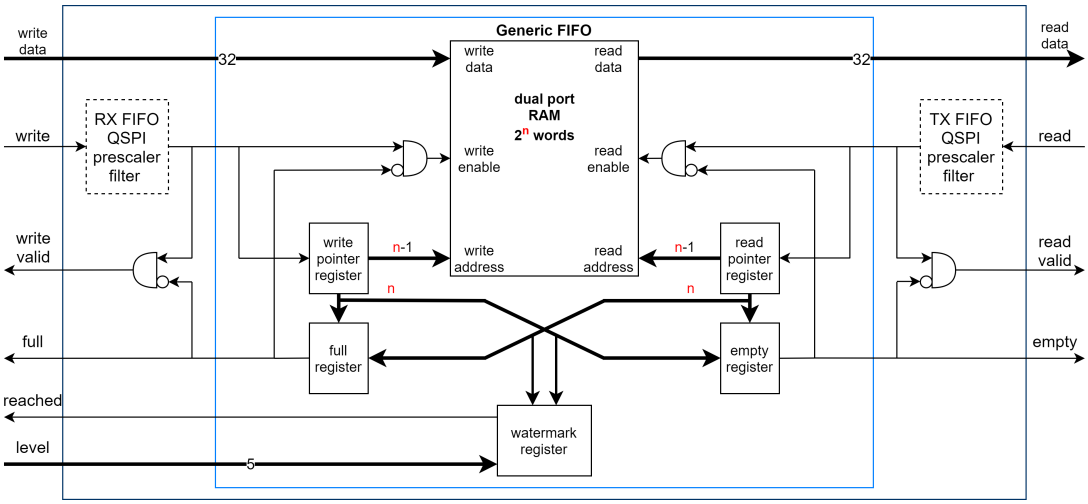


Figure 3.26: Block diagram of the TX/RX FIFO.

The dual-port RAM was used as a memory component to be able to read and write to/from RAM at the same time. Write and read pointer counters were implemented to keep track of addresses for write and read. Each counter has a generic bit size, which equals to the size of the RAM address bus plus an additional bit. The write request is filtered out if the request is active and FIFO is full. Otherwise, the write pointer is increased by one. The

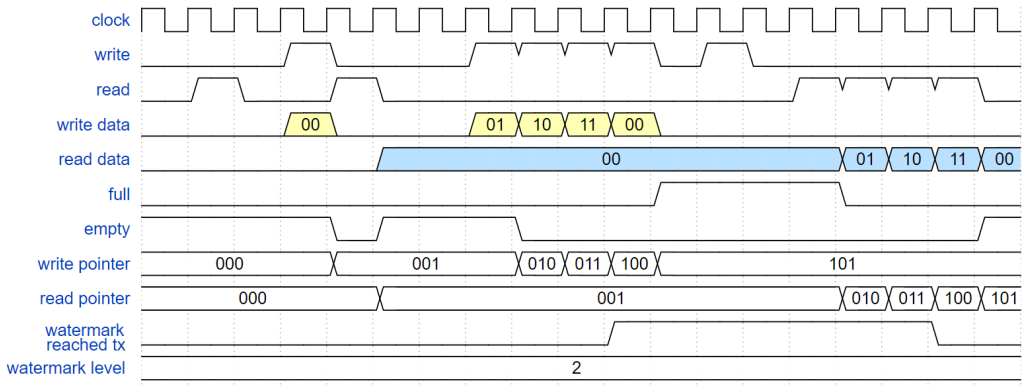


Figure 3.27: Example timing diagram of the TX/RX FIFO.

write pointer points to the position to which will be written during the next request. Write request also operates as enable signal for RAM. The data are written to the memory on a rising edge of the clock. Read request operates similarly as write with a difference that data are read and the read pointer points to the current read register. The data are read on a rising edge of the clock [17].

Empty and full conditions are determined by comparing write and read pointers with the use of an additional bit for both them. An empty condition is set when both pointers are equal. A full condition is also set when both pointers are equal except for an additional bit, which is not equal [17].

A watermark level was calculated from write and read pointer. The generic variable is implemented to distinguish between RX and TX FIFO. The watermark level for TX FIFO is set if the total number of words stored in the RAM is greater than the set watermark level. RX FIFO watermark is set if the total number of words stored in RAM is lesser then set watermark level.

Protocol wrapper around FIFOs is implemented to filter read and write requests from the QSPI protocol controller while a prescaler is used. Because one request from the QSPI protocol can last more than one clock cycle and will cause more than one read/write operation. The request is let through during the first clock cycle and loaded to a register. The output of the register is compared with the current request value. The request is filtered out if both compared values are set high.

3.3.6 QSPI Protocol Controller

The protocol controller block was designed to control the QSPI protocol as a master device. Features of the protocol block were chosen to support the most functions of the current QSPI flash memories, as was in more detail described in subsection 3.1.2. The most notable implemented features are DDR mode, Memory Mapped mode, XIP, alternate phase, and SIOO. The block was decomposed into two separate sub-blocks, as it is shown in Figure **3.28**. The FSM sub-block controls the protocol, thus generates clock and control

signals for transmitter sub-block and QSPI interface. The transmitter sub-block controls data path, therefore converts data between QSPI interface and internal FIFOs, and notifies the FSM whether all data bytes were transmitted/received. All configuration signals are led from the configuration registers block. It was described in subsection 3.3.3.

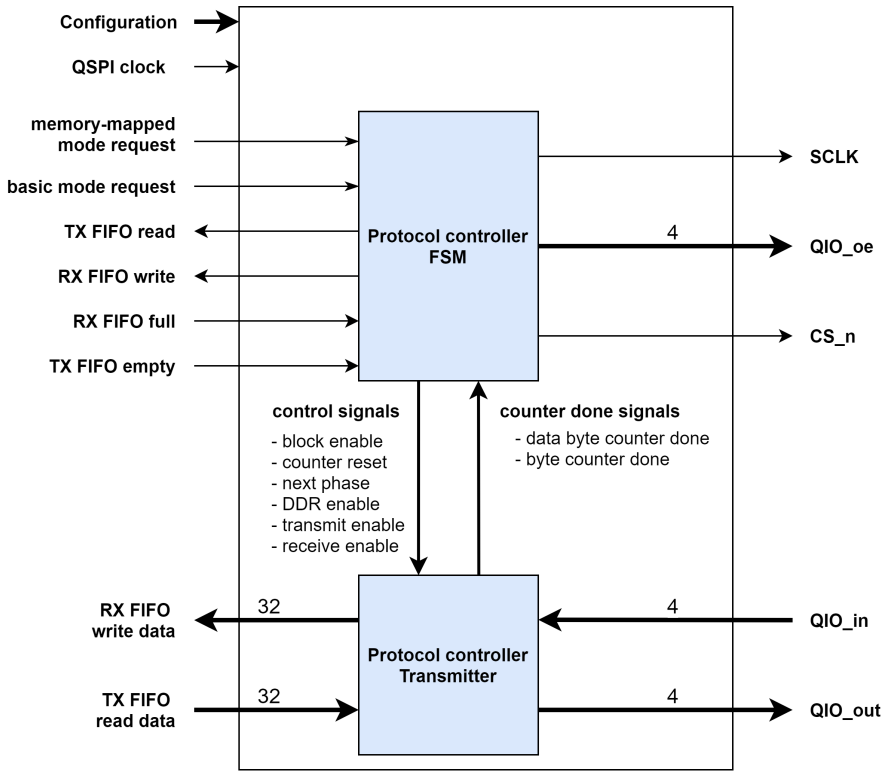


Figure 3.28: Block diagram of protocol controller.

Protocol Controller FSM

The FSM block drives clock and control signals for the QSPI protocol interface and transmitter block. The composition of the transaction frame is configured via the input configuration bus from the configuration registers block. The sub-block itself can be decomposed into smaller functional parts, as it is shown in Figure **3.29**. It can be seen that the main part is made up of a Finite State Machine (FSM), which acts as control logic for the whole protocol controller block.

The clock request for the block is set while the interface is enabled, set to correct mode and any transaction request is active. Only basic mode requests can be accepted if the basic mode is set. The same applies to the memory-mapped mode and its request. This is to prevent the FSM to run a transaction in incorrect mode. The appropriate data for the address phase are selected according to set request mode. The address phase bytes are loaded from the configuration registers for the basic mode. The memory-mapped mode takes the first address used in the PULP transaction, which initiates the request. For each phase, a enable signal was created to notify the FSM about all phases that shall occur during a transaction frame.

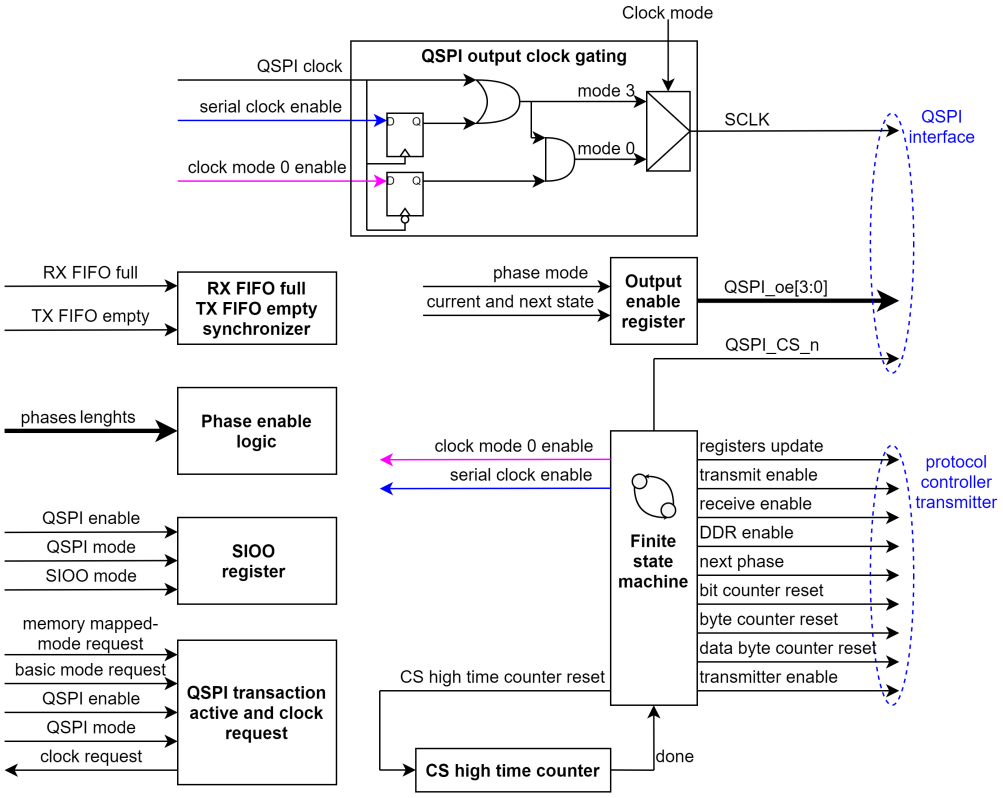


Figure 3.29: Block diagram of protocol controller FSM.

A phase is enabled when its length is more than zero. However, the command phase in memory-mapped mode can be active only during the first transaction, while SIOO is high. Therefore register for SIOO was implemented. The register is set after the first transaction with a command phase. It sets the command enable signal low until the register value is set low. The register resets when the device is switched to the basic mode or the QSPI interface is disabled.

The FSM operates on a falling edge of the QSPI clock. It is implemented with nine states, as it is shown in Figure **3.30**. The diagram is simplified and shows only possible transitions between states and it does not show unnecessary conditions for transitions. Each phase can be left out. The order of phases is always respected and it is not possible to shuffle it. The example of a typical timing diagram of the FSM with control signals is shown in Figure **3.31**. All transmitter counters are reset and new data are loaded to shift registers before the next phase. A request for transition to the next state is initiated by byte counter done signal from the transmitter, except for the data phase. The data byte counter done signal is used for the data phase.

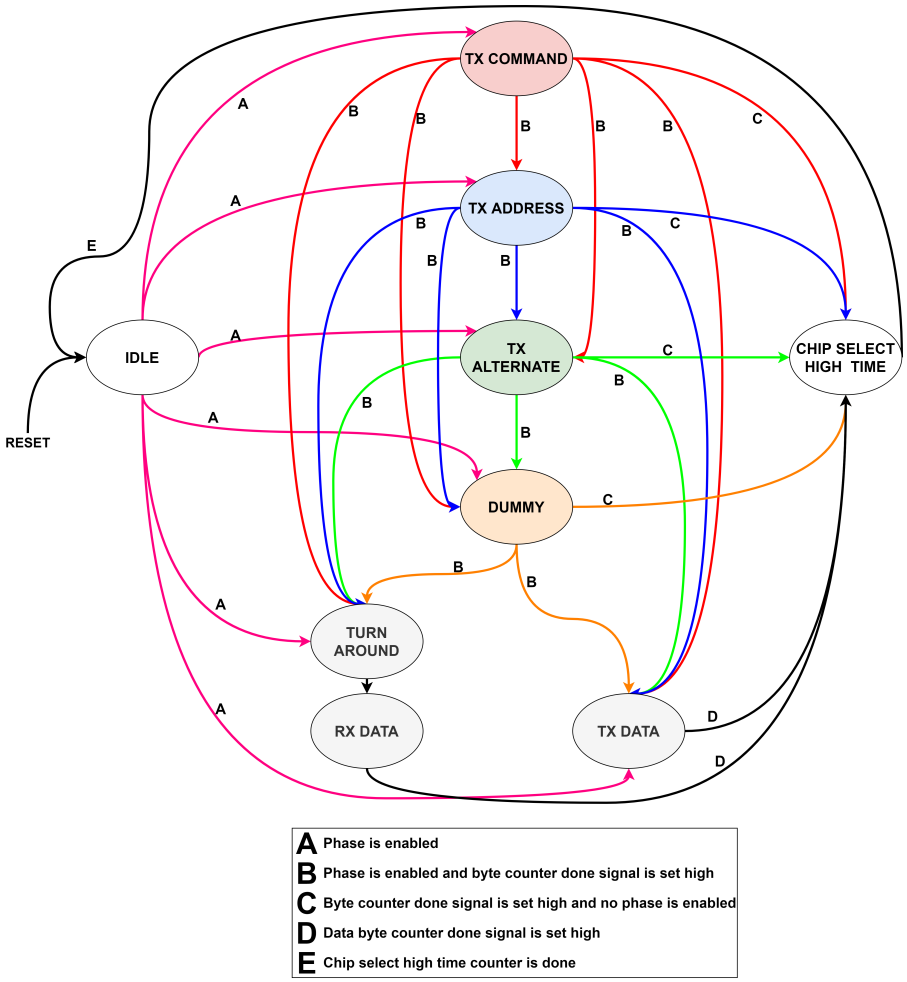


Figure 3.30: State diagram of protocol controller FSM.

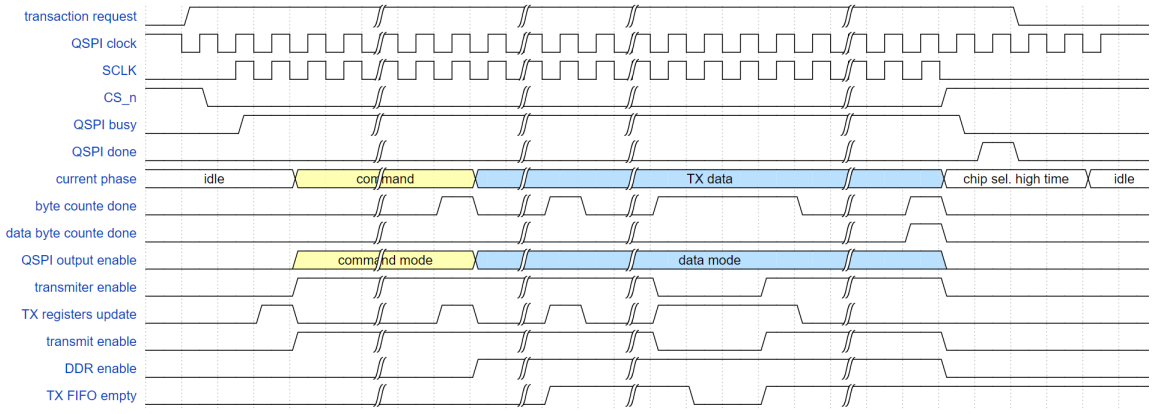


Figure 3.31: Example timing diagram of protocol controller FSM.

IDLE State

The FSM always starts in an IDLE state. During this state, the device anticipates a request for the start of a new transfer frame. All counters in the transmitter are kept in a reset state. The protocol clock signal is disabled and the chip select signal is kept high. A transition to the next phase occurs when a transaction request becomes active. The next

phase is chosen according to enable signals and priority order of the phases. The QSPI busy flag is set high and chip select is set low. The internal clock request for the QSPI and transmitter clock signals is enabled.

TX COMMAND State

The command phase is transmitted. Therefore transmit registers are enabled. During this state, the DDR mode is always disabled. Transition to a next phase occurs when the byte counter done signal is high.

TX ADDRESS and TX ALTERNATE State

The address or alternate phase is transmitted. Transmit registers are enabled. DDR enable signal is set high for both states if DDR is enabled for the address phase. Transition to a next phase occurs when the byte counter done signal is high.

DUMMY State

The dummy phase is transmitted. No data are sent or received during this state, therefore transmit and receive registers are disabled. DDR mode is always disabled. Transition to a next phase occurs when the byte counter done signal is high.

TURN-AROUND State

The turn around state is used to insert one wait clock cycle before the read data phase, therefore receive registers are disabled. However, this does not apply in the case of DDR read. Transition to an RX data state occurs after one clock cycle.

TX and RX DATA States

Only one RX or TX data state can be used per a transfer frame. During these states, data are sent/received. Transmit/receive registers are enabled. DDR is enabled if it is configured for the data phase. Request for read/write from/to FIFO blocks is generated while the byte counter done signal is set to high. If TX FIFO becomes empty during the TX data phase, the transaction is held until new data are written to the FIFO. A similar case applies to the RX data phase. If RX FIFO becomes full, then the transaction is held until data are read from the FIFO. Both empty and full FIFO signals are resynchronized. The transactions are held by disabling the QSPI output protocol clock and transmitter block registers. Transition to a Chip select high time state occurs when the data byte counter done signal is high.

CHIP SELECT HIGH TIME State

This state is always the last before the IDLE state. The transaction has ended and QSPI done flag signal is generated for one clock cycle. The QSPI busy flag is set low and the transmitter block is disabled. However, Chip select signal is kept high for a set number of cycles before the next transaction can start. This allows the user to start the next transaction immediately after chip select high time. A 3-bit binary counter was implemented to count high time clock cycles. Transition to an IDLE state occurs after chip select high counter is done.

Output Clock Gating

Gating of output interface QSPI clock was implemented to provide a clock signal for the QSPI interface. Enable clock signal is generated by the FSM. Two basic gates and registers are used to create two clock signals, which correspond to SPI clock mode 0 and mode 3. The OR gate is used for the gating clock signal for the SPI mode 3. The AND gate is used to create a clock for SPI mode 0. A multiplexer was added to select a correct clock signal to the interface output according to clock mode configuration.

Pads Output Enable Register

This register is used to enable outputs of interface pads. It is always set according to the current data mode and phase. The output is always disabled in a reset state and if the QSPI interface is disabled. QIO2 and QIO3 data lines outputs are enabled to support HOLD and WP functions if all phases are set to SSPI or DSPI.

Protocol Controller Transmitter

The transmitter block was implemented to control a data path between FIFO blocks and the QSPI interface. The primary function is to convert 1,2, or 4-bit size QSPI data bus to/from 32-bit size data buses of FIFOs and registers, and to count transmitted and received bytes to be able to notify the FSM about phase end. The block is composed out of three data counters, transmit/receive shift registers, and receiver buffer, as can be observed in the Figure **3.32**.

One bit counter and two-byte counters were implemented. Namely bit, byte and data byte counter. The timing diagram describing the functionality of all counters is shown in Figure **3.33**. It can be observed dependency of byte and date byte counters on the bit counter.

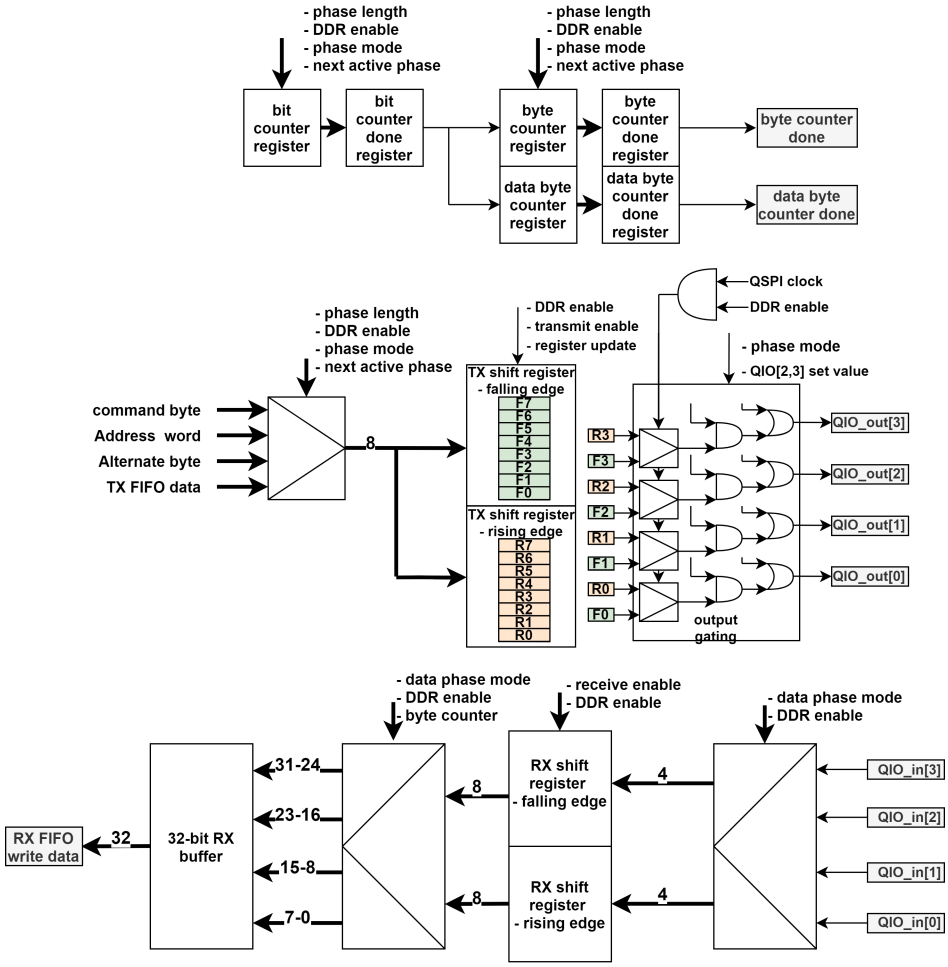


Figure 3.32: Block diagram of protocol controller transmitter.

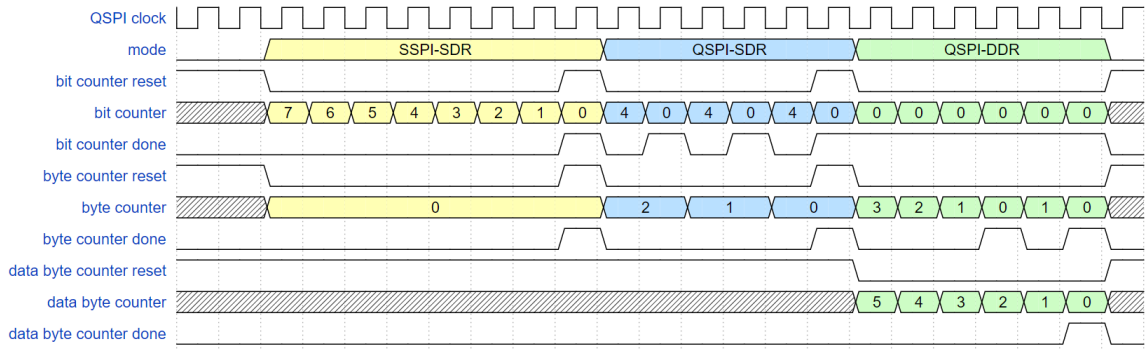


Figure 3.33: Timing diagram of protocol controller transmitter counters.

Bit Counter

The bit counter was implemented as a 3-bit wide binary counter. Its primary function is to count bits of the transaction. The counter is always reset before the next phase. Start and subtract values are also set. That is because each phase has a different number of bits that are transferred per one clock cycle and the phase length can be less than eight bits. Both values are chosen according to phase settings, as is described in the Table **3.15**. It ensures

that the current value corresponds to a number of transmitted bits, which is for example used for sorting received data in the design. The counter subtracts set value on falling edge of clock signal after counter reset was released. Bit counter done signal is set high while value of the counter reaches zero.

Table 3.15: Reset and subtract values for bit counter.

		SDR						DDR					
		SSPI		DSPI		QSPI		SSPI		DSPI		QSPI	
		reset value	substract value	reset value	substract value	reset value	substract value	reset value	substract value	reset value	substract value	reset value	substract value
PHASE	command	7	1	6	2	4	4	-	-	-	-	-	-
	address	7	1	6	2	4	4	6	2	4	4	0	-
	alternate	0-7	1	0/2/4/6	2	0/4	4	0/2/4/6	2	0/4	4	0	-
	dummy	0-7	1	0-7	1	0-7	1	0-7	1	0-7	1	0-7	1
	data	7	1	6	2	4	4	6	2	4	4	0	-

Byte Counter

The byte counter was implemented as a 2-bit binary counter. The function of the counter is to notify the FSM that the current phase is done except for a data phase where it serves as a notifier that four bytes were received/transmitted. The counter is reset and the start value is set before the next phase. The start value is chosen according to next phase as is shown in the Table **3.16**. That is because each phase can have a different length. One is subtracted on each falling edge of the clock after the reset was released. Byte counter done signal is set high while value of the counter is zero. The done signal can be set high regardless of the counter current state in special cases during which whole phase lasts only one clock cycle.

Table 3.16: Reset values for byte counter.

Phase	Reset value	Phase length
command	0	8 bits
address	3	32 bits
	2	24 bits
	1	16 bits
	0	8 bits
alternate	0	1 - 8 bits
dummy	3	31-25 bits
	2	24-17 bits
	1	16-9 bits
	0	8-1 bits
data	3	>= 4 bytes till end
	2	3 bytes till end
	1	2 bytes till end
	0	1 byte till end

Data Byte Counter

The function of the counter is to count data bytes during the data phase. The counter is kept in a reset state until the data phase occurs. The start value is set according to its length. The data byte counter done signal is set high to notify the FSM about the end of the phase.

Data Transmitter

The transmitter part of the block is composed out of sorting muxes, two shift registers and output gating. Muxes sorts data from data registers or TX FIFO depending on the current phase and its properties such as length. Data are arranged to ensure direct connection from a position in the shift register and specific pad. Sorted data are then sampled by two shift registers. Both have eight-bit length. First, data are sampled on the falling edge by first shift register. Second, data are sampled on the rising edge by second shift register. Thus data can be sent on both clock edges during DDR mode. Both shift registers can be disabled when they are not used. Each register shifts and load data according to phase and its properties, as is described in the Table **3.17**. The Shifting and loading of data is enabled by transit enable and register update signals from the FSM. If only SDR mode is used, then only a falling edge register is enabled. An effort was made to put as less as possible combinational logic between shift registers and pads during design phase to minimize clock skew and signal glitches. However, three-stage gating was implemented to be able to use DDR mode and force outputs to selected values.

Table 3.17: Load and Shift sizes of TX transmitter shift registers.

register polarity	data mode	SSPI		DSPI		QSPI	
		shift size	shift during load	shift size	shift during load	shift size	shift during load
falling edge	SDR	1	0	2	0	4	0
falling edge	DDR	2	0	4	0	0	0
rising edge	DDR	2	1	4	2	0	4

The first stage consists of a multiplexer. Its inputs are driven by outputs of shift registers. Its output is connected to the next stage. The gated clock signal is used ss a select signal. The signal disabled and kept at a constant value in SDR mode. The Clock signal is enabled and switches multiplexer each half cycle of the clock during DDR mode. The second and third stage is a combination of OR and AND gate to be able to force values to logic high or low. This is used mainly in SSPI or DSPI mode where QIO[2] and QIO[3] data lines are forced to a specific value by setting in configuration registers.

Data Receiver

The receiving part of the transmitter block has 32-bit buffer and also two 8-bit shift registers for each polarity of the clock signal and sorting muxes. Input data from QSPI interface pads are sorted and sampled by shift registers. The load position depends on the current phase setting, as is described in Table **3.18**. Both registers sample data if DDR is enabled. Otherwise, only falling edge polarity register is active. Sampled byte is sorted and stored to the corresponding position in a 32-bit buffer. The data are written to RX FIFO in one clock cycle on a rising edge if the buffer becomes empty or the transaction has ended.

Table 3.18: Load and Shift sizes of RX transmitter shift registers.

		register polarity		
Data mode		falling edge	rising edge	
SDR	SSPI(QIO 1)	cnt	-	store position
	DSPI (QIO 1..0)	cnt+1, cnt	-	
	QSPI (QIO 3..0)	cnt+3, cnt+2, cnt+1, cnt	-	
DDR	SSPI(QIO 1)	cnt	cnt+1	
	DSPI (QIO 1..0)	cnt+1, cnt	cnt+3, cnt+2	
	QSPI (QIO 3..0)	3,2,1,0	7,6,5,4	

cnt : represents current value of the bit counter

3.3.7 Interrupt Controller

The block is used to notify the processor unit that an interrupt in the system had occurred. The block controls the interrupt output signal. The design supports up to five different sources of interrupts – TX FIFO watermark level reached, RX FIFO watermark level reached, RX FIFO full, RX FIFO empty, TX FIFO full, TX FIFO empty, and QSPI transaction done. For each interrupt source, three signals are used – interrupt enable, interrupt set, and interrupt clear. The interrupt output signal is set if at least one set signal from all sources is active and its enable signal is active. Clear signals are used to set output register low. It is assumed that a clear input signal is active only if the corresponding interrupt is enabled and active. Interrupt is set high if clear and set signals became active in same clock cycle. Level based interrupts are set again after clear is low, but if the set value is still in an active state.

4 Design Verification

The verification of the design was divided into three steps. First, a simple VHDL testbench was used to verify basic functionality during the design phase. Second, the design was implemented with the RISCV processor on FPGA as a proof of concept. The processor was programmed to configure the QSPI master interface. The interface programmed/erased/read data to/from external QSPI flash memory. Last, the UVM verification environment was used. Tests were written for typical use cases for the QSPI master interface.

4.1 VHDL Testbench

A simple VHDL testbench was used during the design phase to verify basic system functionality on the RTL. Checks were done visually or by PSL assertions, which are placed inside design. The testbench is directly connected to the design. It also provides clock signal, pad models, and power up reset signal. The testbench contains a variety of QSPI transactions, mostly of a random character. The design was verified to the stage when it could be used for FPGA implementation. The example of simulation on RTL is shown in Figure **4.1**. The figure shows QSPI write transfer with DDR mode set for the address phase.

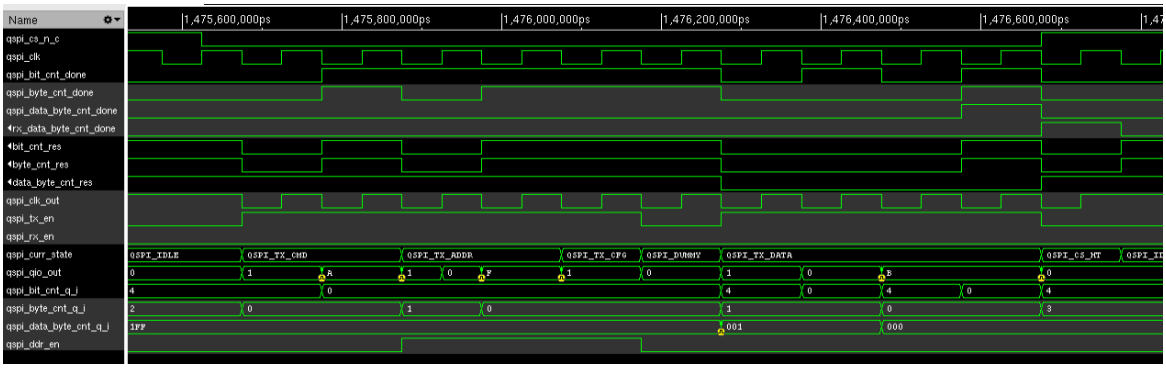


Figure 4.1: Example simulation of simple QSPI write transfer.

4.2 FPGA Proof of Concept

An FPGA implementation was chosen, to prove that protocol specification was understood properly and the design is able to communicate with real flash memory. The main goal was to verify integration with the RISCV processor core and QSPI interface functionality. So it would be possible to run a simple program in the processor core. It will configure QSPI master, initiate transactions, and write/read/erase flash memory.

4.2.1 RTL Integration

The block diagram of integrated system is shown in Figure **4.2**. The integration of subsystems with the processor core is done through the PULP interface. The RISCV uses PULP to access instruction memory, data memory, and JTAG for debug purposes. Therefore it contains three master PULP interfaces. Thus QSPI master is accessed as a slave PULP memory from the processor standpoint. ROM (4096 Byte) and RAM (4096 Byte) were also integrated. The ROM was used as an instruction memory with a stored program. The purpose of the RAM was to store program variables and processor STACK. Both memories were implemented as PULP slaves. Register with AHB interface was used to control LEDs. A conversion block between AHB and PULP was needed to be able to connect the register map with the processor. The PULP crossbar was implemented to route PULP master request and data to appropriate slave memory. For the crossbar, the pre-generated RTL template was edited. A specific address space was chosen for each PULP as it is described in Table **4.1**. Upper bits were used as identifiers and then filtered out from output address heading to a memory space. The boot address of the processor was set to the start of the ROM.

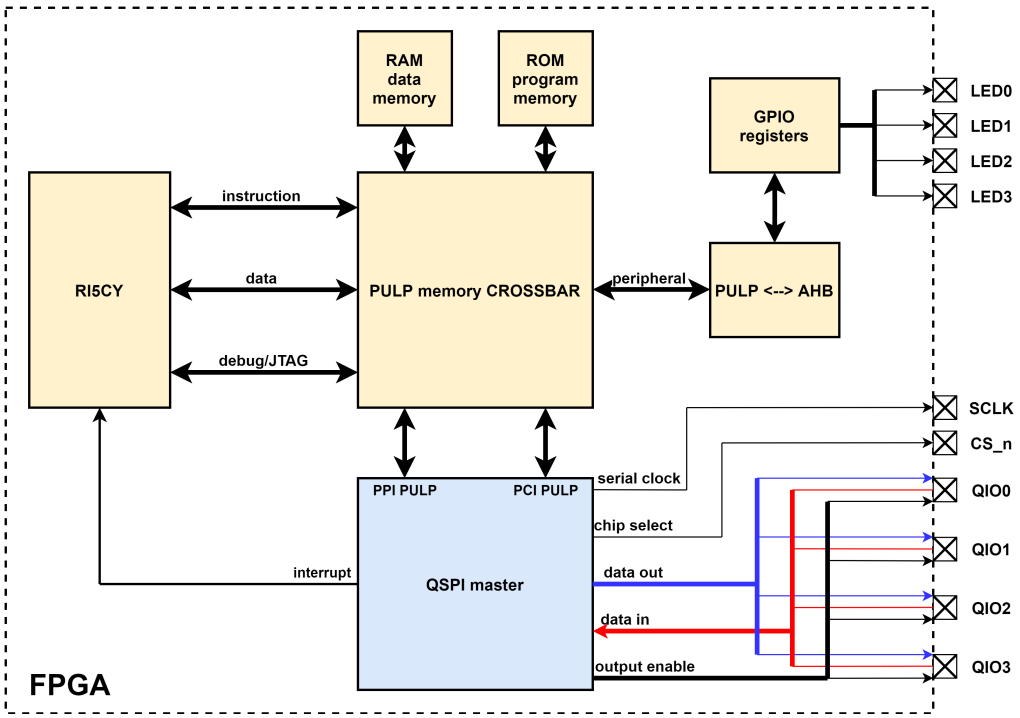
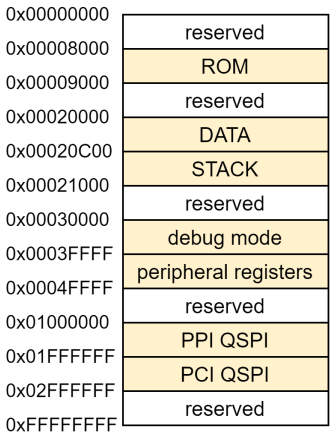


Figure 4.2: Block diagram of integrated system with RISCV and QSPI master.

Table 4.1: Address space used for PULP interface.



4.2.2 Physical Implementation

Artix-7 xc7a200tfbg484 FPGA was chosen for synthesis and physical implementation. Timing and FPGA pads constraints were written specifying clock, reset signals and pad delays, package location and I/O standard. Clock source was specified according to the used development board Trenz te0712. The board has two clock sources phase-locked loop (PPL) at 50 MHz and Multi-Gigabit transceiver (MGT) at 125 MHz. The PPL clock was used as a source clock for implemented design. A synthesis was done using the Vivado 2019.1 tool. The hierarchy flattening was switched off for synthesis to keep netlist hierarchy similar to RTL hierarchy. It was needed for debugging purposes. The Integrated Logic Analyser (ILA)/hardware debug core was created using Vivado debug Wizard, after the design was synthesized. The debug core operates in implemented design as an internal logic analyser, which communicates via JTAG with the VIVADO. The result of implementation with the highlighted QSPI master is shown in Figure **4.3**. Resource utilization for the QSPI master design is described in Table **4.2** and timing report from timing analysis after implementation is described in Table **4.3**. A bitstream was generated to be able to program the FPGA [18].

Table 4.2: FPGA resource utilization table for QSPI master interface.

Resource	Utilization	Available	Utilization (%)
Look up table	769	133800	0.57
Flip-flop	918	267600	0.34

Table 4.3: FPGA timing report for QSPI master interface with the clock at 50 MHz.

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0.876 ns	Worst Hold Slack (WHS):	0.147 ns	Worst Pulse Width Slack (WPWS):	9.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	2741	Total Number of Endpoints:	2741	Total Number of Endpoints:	920

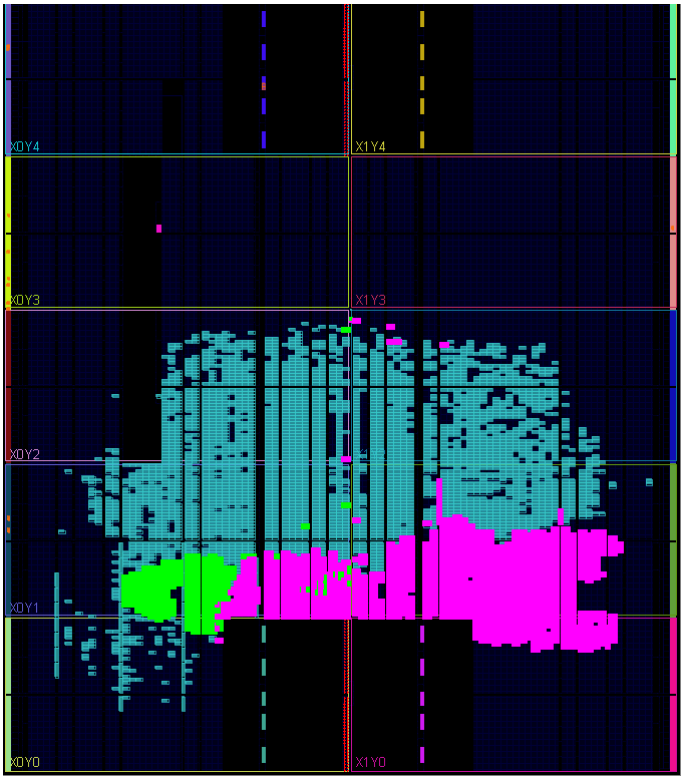


Figure 4.3: Implemented design in FPGA. QSPI master is highlighted in green and RISCV processor with crossbar and memories blocks in purple.

4.2.3 Software Toolchain

Before the FPGA was programmed, program data needed to be loaded into ROM instance. Populating of memory is done in the Vivado tool by rewriting the original bitstream with memupdate command. The command needs bitstream, ".elf" and ".mmi" files as its input. As a result, a new bitstream is generated with data loaded into specified memory cells. The ".mmi" is an XML file that is defined by used address space in ".elf" file, memory type and cell position in an FPGA. The elf file was compiled using riscv32-pulp-elf-gcc toolchain. The toolchain uses linker ".ld", control ".S", and C program file ".c" for compilation. Address spaces of used RAM and ROM were specified inside the linker file so it can link ROM as instruction memory and RAM as data and stack memory. The control file initiates processor. A simple test program was written into the ".c" file. The test should set appropriate QSPI registers and QSPI flash registers, write data to the flash and read data from the same address out. Data are compared and LEDs are lighted if data are the same. The program is described by a flowchart, which can be seen in Figure **4.4** [19].

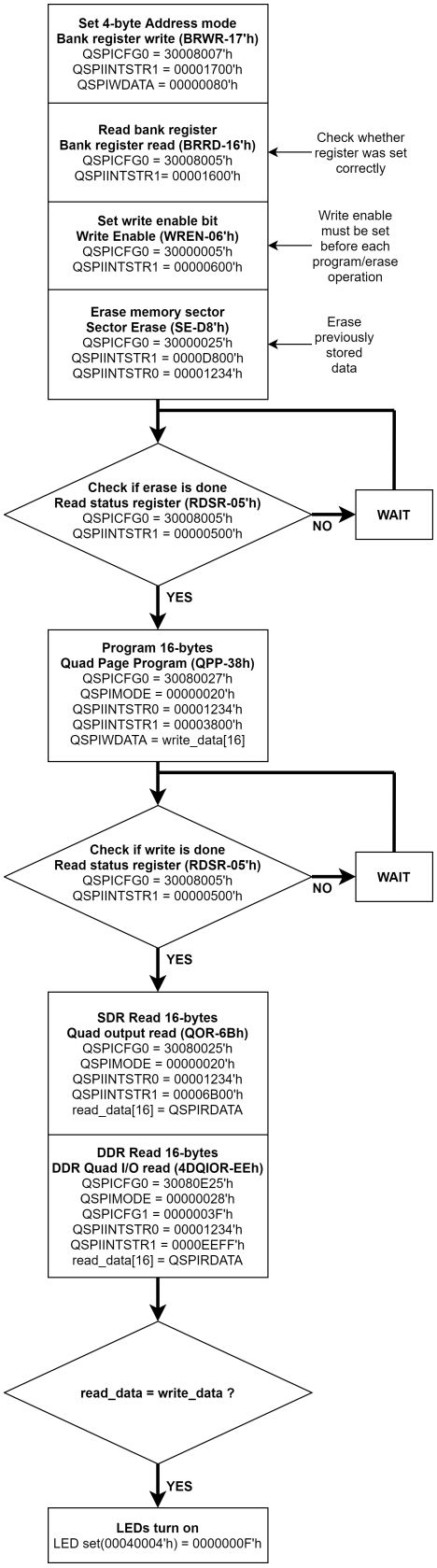


Figure 4.4: Flowchart of simple program for RISCV processor.

4.2.4 Results

The test set-up consists of the FPGA evaluation board (Artix-7 xc7a200tfbg484 FPGA), an eight-channel logic analyzer (MCU123 Saleae Logic clone), and module with QSPI flash memory (FLASH 4 CLICK – S25FL512S flash memory) [14]. It is shown in Figure **4.5**. Internal and external logic analysers verified the implemented design. The results from both analysers were compared with the theoretical assumption for the timing diagrams.

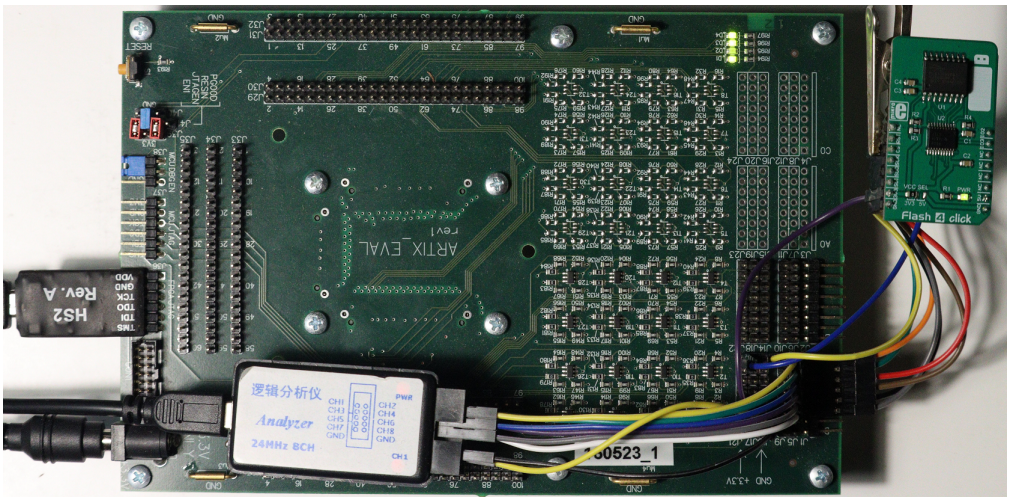


Figure 4.5: Test set-up for implemented QSPI master interface with RISCV on FPGA.

First, the communication was checked between RISCV and QSPI master interface. The example of measured PULP transactions is shown in Figure **4.6**. It shows that the processor without problem can access the PPI PULP slave, and it can set configuration registers inside the interface.

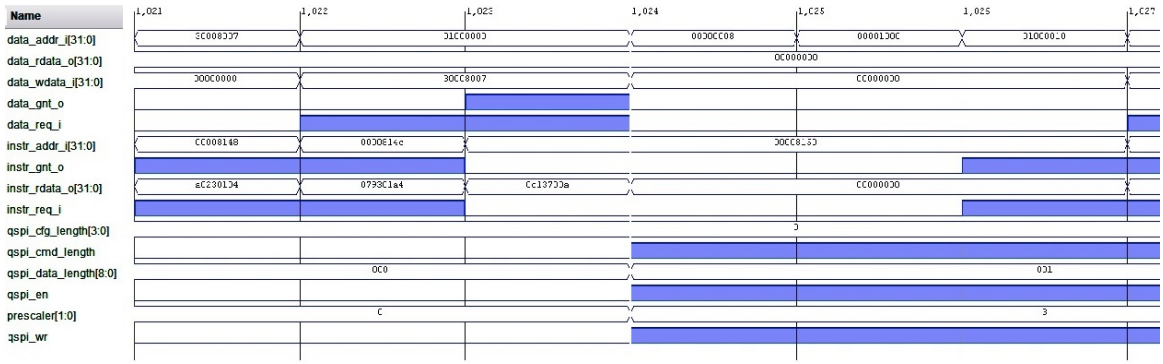
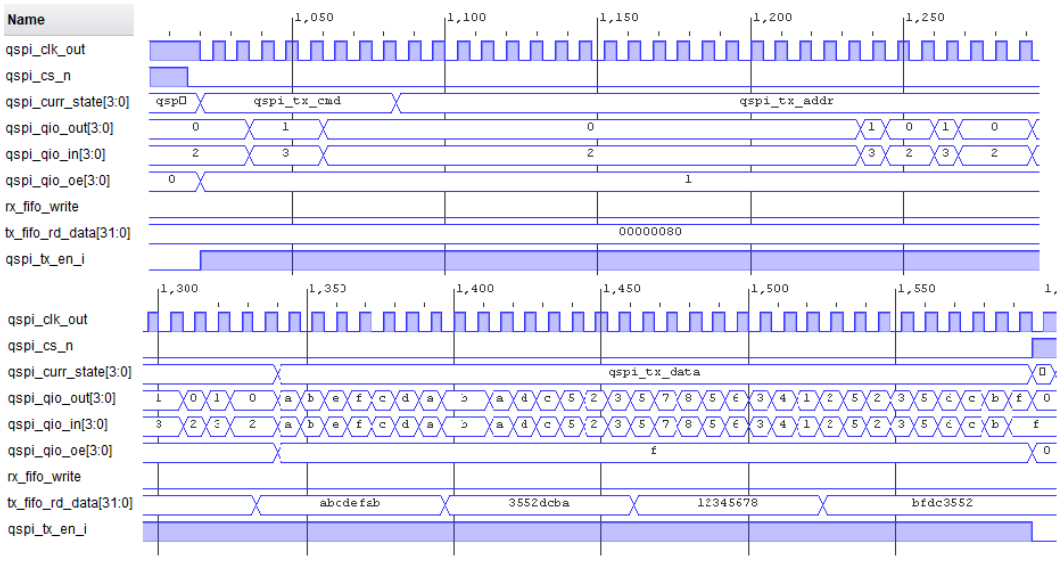


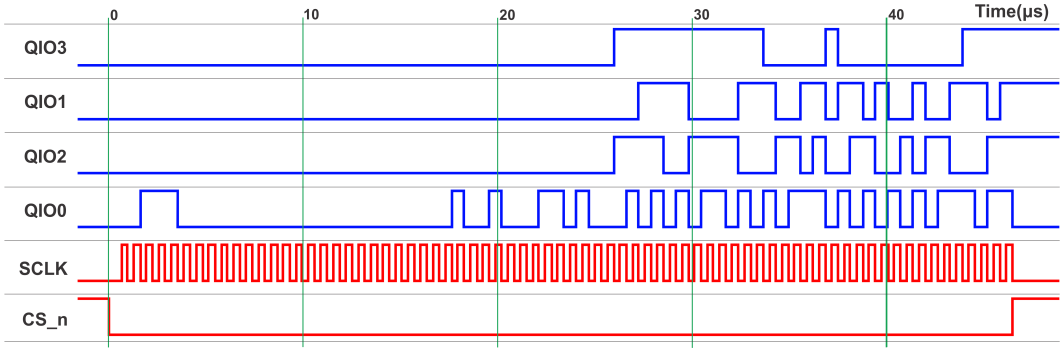
Figure 4.6: RISCV PULP bus transactions measured by internal FPGA Vivado logic analyser.

Second, QSPI transactions were observed between the QSPI master interface and external flash memory. Each QSPI transaction was operated at 1.5 MHz with a clock prescaler set to 8 to run at the lowest transaction frequency. This was set because the QSPI protocol uses single-ended signaling with all data lines referenced against common ground and the flash memory is too far from the FPGA. Thus long ground loop is created. It causes that

impedance of data lines changes and noise resistance is worsen. The example of measured QSPI write transfer by internal and external logic analyser is shown in Figure **4.7**. The write transfer consists of the command phase (SSPI, SDR, 38'h), address phase(SSPI, SDR, 00001234'h), and TX data phase (QSPI, SDR, 16 bytes – 1) ABCDEFAB'h, 2) 3552DCBA'h, 3) 12345678'h, and 4) BFDC3552'h). It shows that the transfer captured by both analysers is the same, and it corresponds with the theoretical timing diagram provided by flash memory datasheet. The written were verified by following read transfer.



(a)

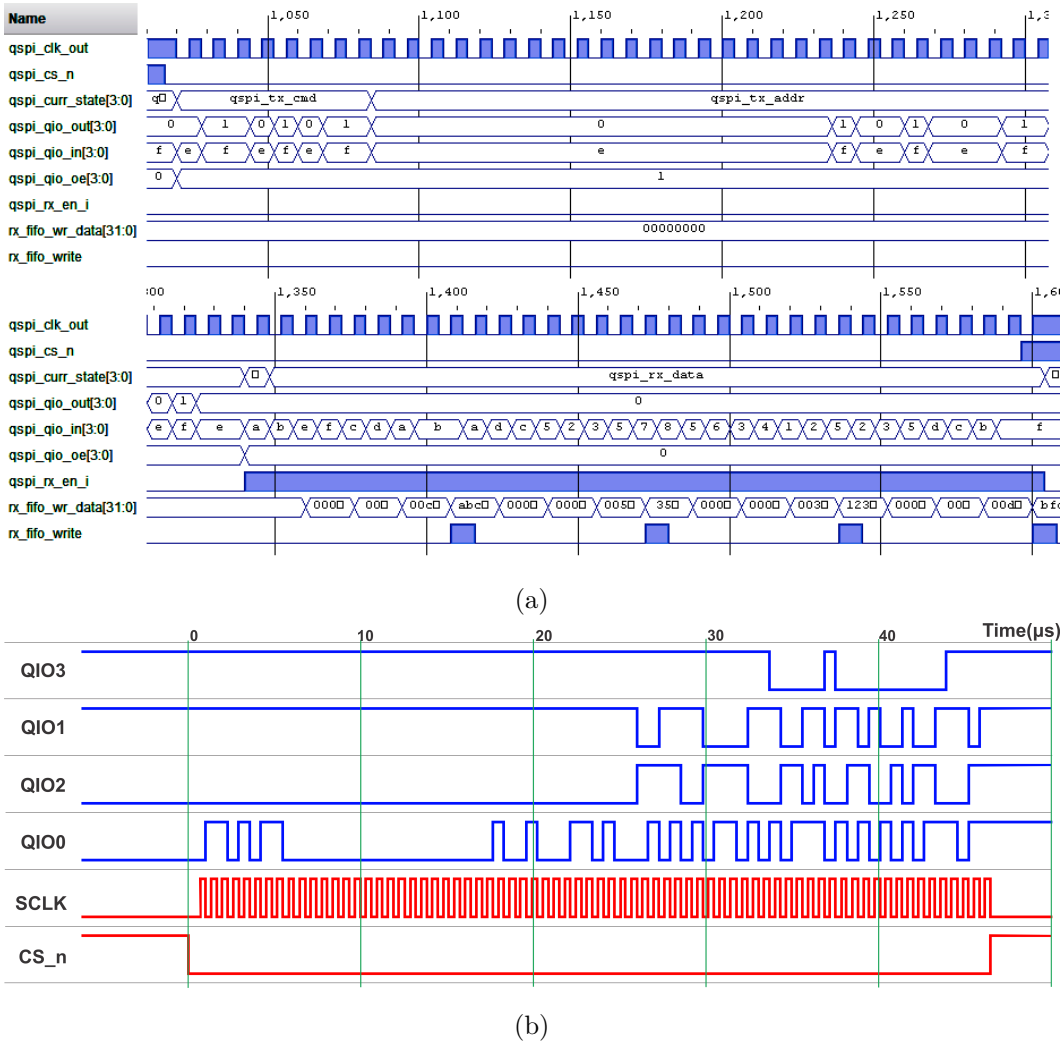


(b)

Figure 4.7: QSPI write transaction measured by a) internal and b) external logic analyser between QSPI master interface and QSPI flash memory.

The read consists of the command phase (SSPI, SDR, 6B'h), address phase (SSPI, SDR, 00001234'h), and RX data phase (QSPI, SDR, 16-bytes). The measured read transfer is shown in Figure **4.8**. It shows that the transfer captured by both analysers is the same, and it corresponds with the theoretical timing diagram provided by flash memory datasheet. Data read by transaction are 1) ABCDEFAB'h, 2) 3552DCBA'h, 3) 12345678'h, and 4) BFDC3552'h. The data are equal to written data, thereby proving that protocol specifications were implemented. The other transactions(sector erase, status register read and write, ...) were also

observed. Each command was checked, whether it corresponds to theoretical assumptions. The memory sector with data was erased and then read to verify the sector erase command. Data read from the erased sector was all set high [14].

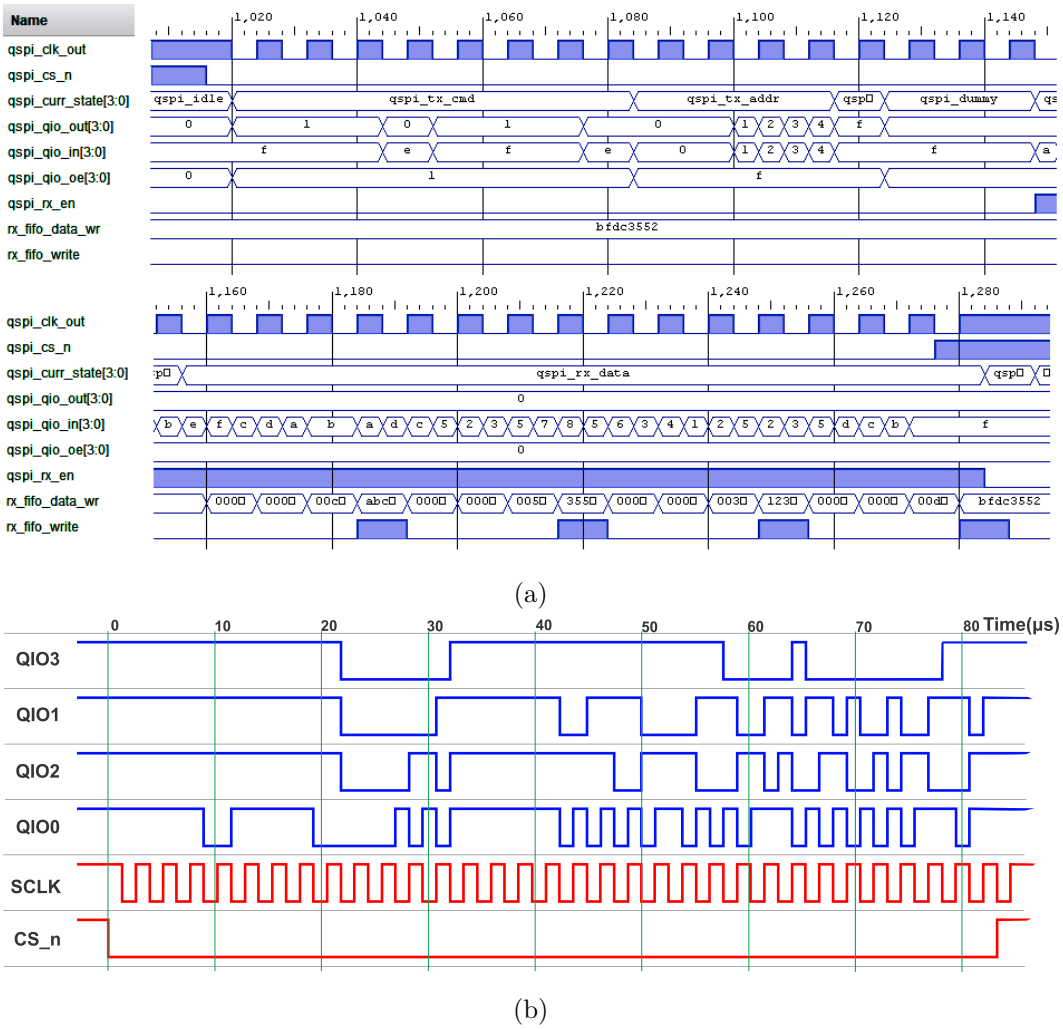


(a)

(b)

Figure 4.8: QSPI read transaction measured by a) internal and b) external logic analyser between QSPI master interface and QSPI flash memory.

At last, DDR read transaction was measured. The clock frequency of system was needed to be decreased due to previously mentioned connection between the FPGA and the external memory. The QSPI transactions were operated at 400 kHz during this measurement. The DDR read consists of command phase (SSPI, SDR, EE'h), address phase (QSPI, DDR, 00001234'h), alternate phase (QSPI, DDR, FF'h), dummy phase (3 cycles), and RX data phase (QSPI, DDR, 16-bytes). The measured read transfer is shown in Figure **4.9**. It shows that the transfer captured by both analysers is the same, and it corresponds with the theoretical timing diagram provided by flash memory datasheet. Data read by transaction are 1) ABCDEFAB'h, 2) 3552DCBA'h, 3) 12345678'h, and 4) BFDC3552'h. It equals to written data [14].



(a)

(b)

Figure 4.9: QSPI DDR read transaction measured by a) internal and b) external logic analyser between QSPI master interface and QSPI flash memory.

4.3 UVM Test Environment

The UVM environment in development written in System Verilog was used for verification of the QSPI master design on RTL. The objective of this phase was to prepare a basic verification plan, write test cases, and fix bugs of the design. Overall, full verification is an extensive and time-consuming phase in design development. The verification usually takes up about 70% time done on a project. Therefore, only the basic use-cases were verified in this chapter to prepare the design for further verification. The structure of the used test environment is shown in Figure **4.10**. It consists of two PULP agents, the QSPI agent, and the scoreboard. All agents drive their corresponding buses according to the set test sequence. Each test is defined by a sequence, which describes test inputs and procedures. The scoreboard is used to compare the inputs of a test and responds to them.

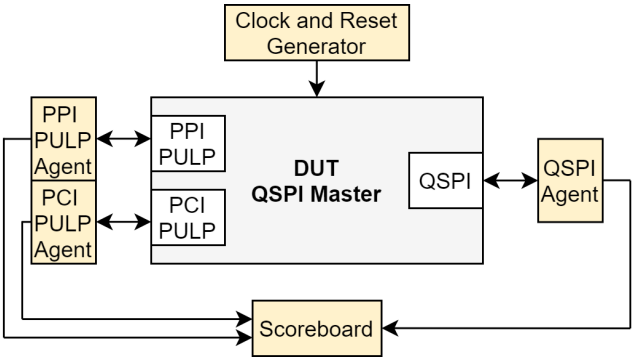


Figure 4.10: Block diagram of used UVM environment.

4.3.1 Verification Plan

A verification plan was written to describe individual test cases to test the main functions of the design, especially of the QSPI protocol block. The description of each test case consists of input data, settings of configuration registers, and test objectives, as can be seen in simplified Table **4.5**. Configurations of registers not mentioned in the table are all set to a random value. Tests listed in the table are sorted from higher to lower priority. The priorities have been assigned to divide time effort for each test appropriately to its importance.

4.3.2 Results

All write or random access previously mentioned test cases were created. However, the UVM environment currently does not support read transfers yet. Therefore read access test cases were not added to the environment, and only simpler VHDL testbenches were temporarily used. All major bugs were fixed, so the basic functionality of the design was verified. Test coverage of each test-case is shown in Table **4.4**. Since code coverage and functional coverage are not 100%, it is still possible that the design contains uncovered

parts and therefore bugs. Achieving 100% coverage (so-called verification closure) is a time-demanding task and was beyond the scope of this thesis. Usually verification takes up about 70% time done on a project.

Table 4.4: Test coverage of verification test-cases.

Testcase	Overral Average Grade (%)	Overall Covered	Assertion Status Grade (%)
QSPI disabled	44.68	5360/19424 (27.57%)	51.36
QSPI SDR max write	71.51	10443/19442 (53.71%)	83.78
QSPI SDR max read (VHDL)	58.59	10060/21318 (47.19%)	85.81
QSPI SDR min write	67.19	8783/19442 (45.18%)	79.73
QSPI SDR min read (VHDL)	58.92	9907/21233 (46.66%)	85.81
QSPI DDR max write	71.80	10618/19442 (54.61%)	89.19
QSPI DDR max read (VHDL)	63.33	11322/21207 (53.39%)	87.16
QSPI DDR min write	69.16	9476/19442 (48.74%)	89.16
QSPI DDR min read (VHDL)	58.60	10086/21217 (47.55%)	84.46
QSPI SDR random	73.51	11811/19442 (60.75%)	85.14
QSPI DDR random	73.06	11284/19442 (58.04%)	90.54
SSPI SDR random	70.55	11605/19442 (59.69%)	83.76
SSPI DDR random	71.60	11872/19442 (61.06%)	89.19
PCI memory mode random (VHDL)	73.50	12540/21219 (59.01%)	86.49
DSPI SDR random	73.53	11869/19442 (61.05%)	87.84
DSPI DDR random	73.86	12036/19442 (61.91%)	87.84
PPI basic mode random	76.51	12186/19442 (62.54%)	90.49

Table 4.5: Simplified verification plan for QSPI master interface.

Test-case	Description	Checks	Transfer	Configurations												
				QSPI enable	request mode	Command phase		Address phase			Alternate phase	Data phase	Data phase			SIOO
						length	mode	length	mode		length	direction	length	mode		
QSPI disabled	Interface is disabled. Transaction has random configuration.	Check if transaction starts with disabled QSPI interface	1	disabled	random	random	random	random	random	random	random	random	random	random	random	random
QSPI SDR max	Basic mode request. QSPI and SDR mode are set for all phases. Maximal length is set for each phase.	• phases lengths • input/output phases data • status/interrupts registers • FIFO full/empty transfer hold	1	enabled	basic	8 bits	QSPI	4 Bytes	QSPI	SDR	8 bits	write	511 Bytes	QSPI	SDR	disabled
			2	enabled	basic	8 bits	QSPI	4 Bytes	QSPI	SDR	8 bits	read	511 Bytes	QSPI	SDR	disabled
QSPI SDR min	Basic mode request. QSPI and SDR mode are set for all phases. Minimal length is set for each phase.	• phases lengths • input/output phases data • status/interrupts registers	1	enabled	basic	8 bits	QSPI	1 Byte	QSPI	SDR	1 bit	write	1 Byte	QSPI	SDR	disabled
			2	enabled	basic	8 bits	QSPI	1 Byte	QSPI	SDR	1 bit	read	1 Byte	QSPI	SDR	disabled
QSPI DDR max	Basic mode request. QSPI and DDR mode are set for all phases. Maximal length is set for each phase	• phases lengths • input/output phases data • status/interrupts registers • FIFO full/empty transfer hold	1	enabled	basic	8 bits	QSPI	4 Bytes	QSPI	DDR	8 bits	write	511 Bytes	QSPI	DDR	disabled
			2	enabled	basic	8 bits	QSPI	4 Bytes	QSPI	DDR	8 bits	read	511 Bytes	QSPI	DDR	disabled
QSPI DDR min	Basic mode request. QSPI and DDR mode are set for all phases. Minimal length is set for each phase	• phases lengths • input/output phases data • status/interrupts registers	1	enabled	basic	8 bits	QSPI	1 Byte	QSPI	DDR	1 bit	write	1 Byte	QSPI	DDR	disabled
			2	enabled	basic	8 bits	QSPI	1 Byte	QSPI	DDR	1 bit	read	1 Byte	QSPI	DDR	disabled
QSPI SDR random	Basic mode request. QSPI and SDR mode are set for all phases. Random length of phases. Random phase lengths.	• phases lengths • input/output phases data • status/interrupts registers • FIFO full/empty transfer hold	1	enabled	basic	random	QSPI	random	QSPI	SDR	random	random	random	QSPI	SDR	disabled
QSPI DDR random	Basic mode request. QSPI and DDR mode are set for all phases. Random phase lengths.	• phases lengths • input/output phases data • status/interrupts registers • FIFO full/empty transfer hold	1	enabled	basic	random	QSPI	random	QSPI	DDR	random	random	random	QSPI	DDR	disabled
SSPI SDR random	Basic mode request. SSPI and SDR mode are set for all phases. Random phase lengths.	• phases lengths • input/output phases data • status/interrupts registers • FIFO full/empty transfer hold	1	enabled	basic	random	SSPI	random	SSPI	SDR	random	random	random	SSPI	SDR	disabled
SSPI DDR random	Basic mode request. SSPI and DDR mode are set for all phases. Random phase lengths.	• phases lengths • input/output phases data • status/interrupts registers • FIFO full/empty transfer hold	1	enabled	basic	random	SSPI	random	SSPI	DDR	random	random	random	SSPI	DDR	disabled
PCI memory mode random	Memory mode request. Random phases settings. Command enabled. SIOO eanbled	• if command is send only once • initiation of transfer and read data by PCI PULP	1	enabled	memory	8 bits	random	random	random	random	random	random	random	random	random	enabled
			2	enabled	memory	8 bits	random	random	random	random	random	random	random	random	random	enabled
DSPI SDR random	Basic mode request. DSPI and SDR mode are set for all phases. Random phase lengths.	• phases lengths • input/output phases data • status/interrupts registers • FIFO full/empty transfer hold	1	enabled	basic	random	DSPI	random	DSPI	SDR	random	random	random	DSPI	SDR	disabled
DSPI DDR random	Basic mode request. DSPI and DDR mode are set for all phases. Random phase lengths.	• phases lengths • input/output phases data • status/interrupts registers • FIFO full/empty transfer hold	1	enabled	basic	random	DSPI	random	DSPI	DDR	random	random	random	DSPI	DDR	disabled
PPI basic mode random	All configurations are random	• phases lengths • input/output phases data • status/interrupts registers • FIFO full/empty transfer hold	1	enabled	basic	random	random	random	random	random	random	random	random	random	random	disabled

5 Conclusion

The objective of this thesis was to design the QSPI master interface integrable with the RISCV processor. First, the QSPI protocol was studied from different sources, and a unified description of the protocol was written. The RI5CY PULP interface was also studied to design a protocol bridge between the QSPI and PULP protocol. Second, the specification of the QSPI master interface was created based on the study of existing serial flash memories and their features. A system-level design was prepared from the specification. The design was implemented on the RTL in VHDL. It was continuously verified during the design phase by a simple VHDL testbench.

At last, the interface was implemented with the RISCV processor in Artix7 FPGA as a proof design works with real memory. The processor was programmed, and the communication was checked between the QSPI master interface and external QSPI flash memory. The interface was able to write/read/erase to/from the memory. The design was verified in the UVM test environment implemented in System Verilog. The verification plan for UVM verification was written. Tests were created for UVM according to the verification plan. The basic functionality of the design was verified.

The thesis tasks were fulfilled, and the QSPI master interface development will continue in the ASICentrum s.r.o. To use QSPI IP Core in ASIC, finishing verification to 100% coverage would be desirable as well as performing ASIC synthesis.

Bibliography

[1] *S79FL01GS – Serial flash memory.* Cypress., 2018, [online], [ref. 2020-05-22]. available from:
http://cypress.com/file/177986/.

[2] Traber, A. Gautschi, M. Schiavone, P. *RI5CY: User Manual.* Pulp platform., 2019, [online], [ref. 2020-05-22]. available from:
http://pulp-platform.org/docs/ri5cy_user_manual.pdf.

[3] *AN4760 – Quad-SPI interface (QUADSPI).* STMicroelectronics., 2019, [online], [ref. 2020-05-22]. available from:
http://st.com/resource/en/application_note/dm00227538-quadspi\
-interface-quadspi-on-stm32-microcontrollers-stmicroelectronics.pdf.

[4] *JESD216D.01.* JEDEC SOLID STATE TECHNOLOGY ASSOCIATION., 2019, [online], [ref. 2020-05-22]. available from:
http://jedec.org/standards-documents/docs/jesd216b?destination=node/8406.

[5] *W25Q32JW-DTR – Serial flash memory.* Winbond., 2017, [online], [ref. 2020-05-22]. available from:
http://winbond.com/resource-files/w25q32jw/20dtr/20revb/20dtr/201512017.
pdf.

[6] Dhaker, P. *Introduction to SPI Interface.* Signal integrity journal., 2018, [online], [ref. 2020-05-22]. available from:
http://signalintegrityjournal.com/articles/967-introduction-to-spi-interface.

[7] *ARM AMBA 5 AHB Protocol specification.* ARM Developer., 2015, [online], [ref. 2020-05-22]. available from:
http://static.docs.arm.com/ihi0033/bb/IHI0033B_B_amba_5_ahb_protocol_
spec.pdf.

[8] *Quad Serial Peripheral Interface module.* NXP., 2012, [online], [ref. 2020-05-22]. available from:
http://nxp.com/docs/en/application-note/AN4512.pdf.

[9] *W25Q128FV – Serial flash memory.* Winbond., 2013 ,[online], [ref. 2020-05-22]. available from:
http://winbond.com/resource-files/w25q128fv_revhh1_100913_website1.pdf.

[10] *MX35LF2G14AC – Serial flash NAND memory.* Macronix., 2017, [online], [ref. 2020-05-22]. available from:
http://macronix.com/Lists/Datasheet/Attachments/6867/MX35LF2G14AC,/203V,/202Gb,/20v1.0.pdf.

[11] *MX66L1G45G – Serial flash memory.* Macronix., 2018, [online], [ref. 2020-05-22]. available from:
http://macronix.com/Lists/Datasheet/Attachments/7543/MX66L1G45G,/203V,/201Gb,/20v1.4.pdf.

[12] *MT25QU512AB – Serial NOR flash memory.* Micron., 2013, [online], [ref. 2020-05-22]. available from:
http://micron.com/-/media/client/global/documents/products/data-sheet/nor-flash/serial-nor/mt25q/die-rev-b/mt25q_qlkt_u_512_abb_0.pdf.

[13] *N25Q128A – Serial NOR flash memory.* Micron., 2012, [online], [ref. 2020-05-22]. available from:
http://micron.com/-/media/client/global/documents/products/data-sheet/nor-flash/serial-nor/n25q/n25q_128mb_1_8v_65nm.pdf.

[14] *S25Fl512S – Serial flash memory.* Cypress., 2019, [online], [ref. 2020-05-22]. available from:
http://cypress.com/file/177971/.

[15] Cummings, C. *Asynchronous & Synchronous Reset Design Techniques.* Sunburst Design, Inc., 2003, [online], [ref. 2020-05-22]. available from:
http://sunburst-design.com/papers/CummingsSNUG2003Boston_Resets.pdf.

[16] Stastny, J. *FPGA prakticky.* BEN – odborna literatura s.r.o., 2011, [ref. 2020-05-22].

[17] Cummings, C. *Simulation and Synthesis Techniques for Asynchronous FIFO Design.* Sunburst Design, Inc., 2002, [online], [ref. 2020-05-22]. available from:
http://sunburst-design.com/papers/CummingsSNUG2002SJ_FIFO1.pdf.

[18] *Vivado Design Suite User Guide – Synthesis.* XILINX., 2019, [online], [ref. 2020-05-22]. available from:
http://xilinx.com/support/documentation/sw_manuals/xilinx2019_1/ug901-vivado-synthesis.pdf.

[19] Waterman, A. Asanovic, K. *The RISC-V Instruction Set Manual.* RISC-V., 2017, [online], [ref. 2020-05-22]. available from:
http://content.riscv.org/wp-content/uploads/2017/05/riscv-spec-v2.2.pdf.

Used software:

Microsoft Office 365, Logic 1.2.18, MiKTeX 2.9, CorelDRAW X7, Draw.io 13.0.9, Wavedrom 2.6.3, Vivado 2019.2, NCSim 15.20.07

A RTL Codes

RTL codes for the design and FPGA implementation are not available in the public/online version of the thesis. All codes are confidential to ASICentrum s.r.o.