International Journal of Control and Automation
Vol. 8, No. 4 (2015), pp. 97-102
http://dx.doi.org/10.14257/ijca.2015.8.4.12

Design of a High-Speed Serial Programming Interface Compatible with Bluetooth Embedded Systems

Sangook Moon

Department of Electronic Engineering, Mokwon University,
Daejeon 302-729, Korea
smoon@mokwon.ac.kr

Abstract

In this paper, we designed a serial programming interface (SPI) suitable for embedded systems, especially for Bluetooth baseband. Proposed architecture is compatible for the APB bus in AMBA bus architecture. The 8-bit design of the SPI module is in charge of transferring the data and the instructions between the external devices and the coprocessors. We adopted the cyclic redundancy check method for the error correction. Also, we provided the interface for multimedia cards. The designed SPI module was automatically synthesized, placed, and routed. Implementation was performed through the Altera FPGA and well operated at 25MHz clock frequency.

Keywords: *Serial programming interface, Bluetooth, Advanced peripheral bus (APB), AMBA bus, FPGA*

1. Introduction

Bluetooth is attractive in that the service provides a fundamental and ceaseless way to connect people whenever and wherever they want by connecting the information-communication devices with each other, consuming far less electrical power compared to the wireless LAN. The Bluetooth SIG (Special Interest Group) is working on improving specifications and making efforts to develop more convenient consumer products worldwide. To the present, Motorola, Microsoft, Lucent Technology, 3COM, Ericsson, Nokia, IBM, and Toshiba are leading the SIG.

The SPI is a high-speed synchronous serial port for communicating to external devices. The SPI in this document is for external device. SPI is byte-orientated and every command, response and data block is built with a byte (8-bit). SPI messages are built from command, response and data-block tokens. The CP (master) controls all communication between itself and the external device. Serial data transmission through SPI starts when the chip-select (CS) is asserted (i.e. when the CS goes to LOW) and ends when the chip-select is released (i.e. when the CS goes to HIGH).

Every external device token transferred on the data signal is protected by CRC bits. But external device offers non-protected mode that enables a system built with reliable data links to exclude the hardware or firmware required for implementing the CRC generation and verification functions. In non-protected mode, the CRC bits of the command, response and data tokens are still required in the tokens; they are, however, defined as "don't care" for the transmitters and are ignored by the receivers.

MMC is initialized in the non-protected mode. The CP can turn this option on and off using the CRCONOFF command (CMD39). It is assumed that CRC is processed by software. In this paper, we designed a high-speed SPI module and implemented with an FPGA test chip. Also we simulated to verify the operation at post-layout circuit level.

2. Bluetooth Embedded System

Bluetooth was originally designed as a cable replacement technology aimed at providing effortless wireless connectivity for consumer devices in an ad hoc fashion. In order to allow for deployment almost worldwide, the Bluetooth Special Interest Group (SIG) placed the technology in the unlicensed industrial, scientific, and medical (ISM) band at 2.4 GHz. By designing a comparably straightforward system, the designers of Bluetooth intended for it to have widespread use.

Bluetooth networks are organized into “piconets” in which a “master” unit coordinates the traffic to and from up to seven active “slave” units. The master unit originates the request for a connection setup. Within a single piconet, the various slave units can only communicate with each other via the master. Nevertheless, every Bluetooth unit can be a member of up to four different piconets simultaneously (though it can be master in only one of them). A formation in which several piconets are interlinked in such a manner is called a scatternet.

Fig. 1 shows the general block diagram of Bluetooth baseband. The RF (Radio Frequency) module, which is placed in the right side of the figure, modulates the frequencies from 2.4GHz ISM band to the intermediate frequencies. The data received by the RF module are de-skewed and re-sampled as the exact 1MHz sampled data and given to the low-pass filter to remove the possible noise, later to be passed into the baseband block. Data are divided into 64-bit blocks, the head of which begins with ‘1010’ or ‘0101’, depending on the type of the packet. Serially injected data are transformed to parallel data and separated into header and payload parts. Header and payload packets are examined in the error correction blocks respectively. FEC (Forward Error Check) scheme is used to examine the packets by the Bluetooth specification [1][2]. There are two kinds of Bluetooth packet, SCO (Synchronous Connection Oriented) and ACL (Asynchronous Connection oriented Link). SCO packets are used in applications with the voice communication, which needs real-time data slots. ACL packets are used to transfer timely flexible data [3][4].

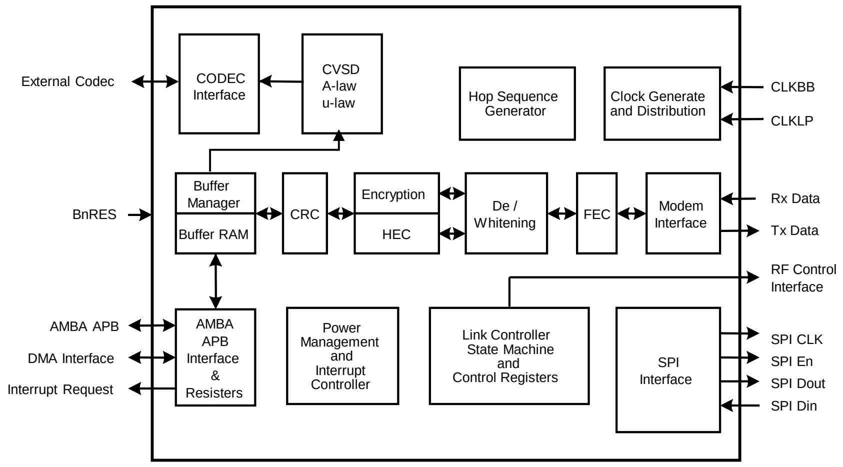

Figure 1. Block Diagram of Target Bluetooth Baseband

3. SPI Module

The signals assigned for communication between the SPI and the Bluetooth embedded system consists of 4 types, as shown in Table 1. The implementation should have control registers for the communication between the SPI and the coprocessor that conforms to the instruction code described in the register values. SPI controller has registers such as SPI control register, status register, a counter for data exchange, 8-bit TX data buffer, 8-bit RX data buffer, test register, and the special register for pin sharing with GPIO module. Table 2 show the SPI register names, register widths, read or write activity, and the default value.

Table 1. Signal Description between the SPI and the Embedded System

Name	Type	Description
MOSI	O	Host to card data signal
MISO	I	Card to host data signal
SPICLK	O	Host to card clock signal
CS	O	Host to card chip select signal

Table 2. SPI Control Registers Summary

Name	Width	R/W	Default
SPICR	8	R/W	0x20
SPISR	8	R	0x00
XCHCOUNTER	10	R/W	0x00
Txdatabuffer	8	W	0x00
Rxdatabuffer	8	R	0x00
Testregister1	8		0x00
Testregister2	8		0x00
ResetReg	8	R/W	0x00
TIC	8	R/W	0x00
SPI_ON	1	R/W	0x01

4. SPI Module Implementation

Author We designed the SPI module targeted to have a high frequency. After CP (co-processor) writes a sequence of data to the TX FIFO, the content of the FIFO is loaded into the TX shift register and is shifted out serially one byte at a time. When all elements in the TX FIFO are transferred to the TX shift register, the SPI-MMC issues an interrupt to CP, which may fill the TX FIFO for further data transfer. Serial input data is shifted into the RX shift register. After 8 bits are shifted in, the content of the RX shift register is copied into the RX FIFO. When the RX FIFO is full, the SPI-MMC issues an interrupt to CP through the SPIIRQ signal. CP reads the content of the RX FIFO in an interrupt service routine. The timing and control block produces all necessary control signals of the SPI-MMC block, including SPICLK. The frequency of SPICLK signal is programmable.

SPI-MMC transfer's protocol is command and response. Whenever CP sends a command to MMC (via SPI), MMC sends CP (via SPI) a response. The response is variable length for command--for example, there is 1-, 6-, and 17-byte. There are only 6 bytes in command.

The sequence of operations that occur in a read transfer is as follows:

1. CP sends a reset signal to the SPI-MMC block. In other words, CP writes “0” to bit in the ResetReg register. The signal is used to clear counters inside the block. Before new exchange begins and the content of XCHCOUNTER and transmit mode is changed (XCHMODE BIT in the SPICR), CP must send a reset signal to the SPI-MMC block.
2. First, CP set up the SPICR register. In this example, XCHMODE is send mode.

3. CP writes number to send into XCHCOUNTER register.
4. CP writes "Data read command (CMD17)" into the TX FIFO.
5. CP asserts CS signal. In other words, CP write 0 to CS bit in the SPICR.
6. CP sends a start signal to SPI-MMC. In other words, CP set XCH bit in the SPICR.
7. The SPI-MMC block sends out 6 bytes of command data from TX FIFO through TX shift register.
8. The SPI-MMC block issues the interrupt after it sends all data in TX FIFO.
9. The CP reads the SPISR register in the SPI-MMC block and disable start signal (reset XCH bit). In other words, CP writes the SPICR register.
10. CP sends a reset signal to the SPI-MMC block. In other words, CP writes 0 to bit in the ResetReg register. The signal is used to clear counters inside the block. Before new exchange begins and the content of XCHCOUNTER is changed, and transmit mode is changed (XCHMODE BIT in the SPICR), CP must send a reset signal to the SPI-MMC block.
11. CP changes transmit mode (XCHMODE is receive mode).
12. The CP writes number to be received into XCHCOUNTER register.
13. CP sends a start signal to SPI-MMC (set XCH bit).
14. Then SPI-MMC controller receives response from MMC.
15. After SPI-MMC receives 1 byte (for CMD17 command), it sets XCH DONE status bits and it issues an interrupt to a CP.
16. The CP reads the SPISR register in the SPI-MMC block and disable start signal (reset XCH bit). In other words, CP writes the SPICR register.
17. The CP reads data RX FIFO.
18. After CP takes this response data and examines it, CP acts as response data. If there is no error indication in response, CP informs SPI-MMC block that MMC sends data to it.
19. CP sends a reset signal to the SPI-MMC block. In other words, CP writes 0 to bit in the Reset register. The signal is used to clear counters inside the block. Before new exchange begins and the content of XCHCOUNTER and transmit mode is changed (XCHMODE BIT in the SPICR), CP must send a reset signal to the SPI-MMC block.
20. The CP writes number to be received into XCHCOUNTER register.
21. CP sends a start signal to SPI-MMC (set XCH bit).
22. The SPI-MMC block receives data from MMC (for example, data length is from 4 byte to 515 byte).
23. If SPI-MMC receives data like RX FIFO size, SPI-MMC block sets the “RX FIFO full” status bit and issues an interrupt to CP. At this time SPICLK disables start signal for prevention of RX FIFO overrun. If CP takes all data in RX FIFO, CP sends a start signal and receives response to remain. Repeat it.
24. After SPI-MMC block receives all data from MMC, it sets the XCH DONE status bit and issues an interrupt to CP.

25. The CP reads the SPISR register in the SPI-MMC block and disable start signal (reset XCH bit). In other words, CP writes the SPICR register.
26. After CP takes last data from RX FIFO, CP de-asserts CS signal.

Fig. 2 shows the SPI command and response timing diagram. We show the SPI data read timing diagram in Fig. 3, and we show the SPI data write timing diagram in Fig. 4.

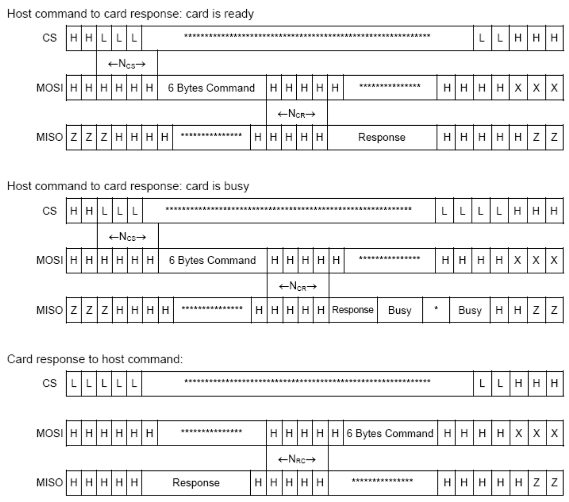
Host command to card response: card is ready
CS
MOSI
MISO
6 Bytes Command
Response
Host command to card response: card is busy
Busy
Card response to host command:
N_CS
N_CR
N_RC

Figure 2. SPI Command-response timing Diagram

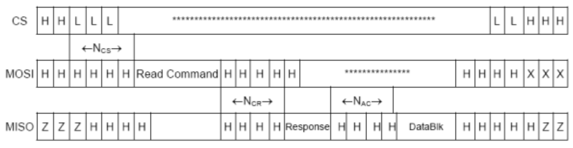

Figure 3. SPI Data Read Timing Diagram

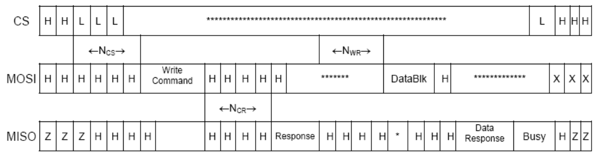

Figure 4. SPI Data Write Timing Diagram

5. Conclusions

For the sake of good communication between the SPI and the CP, well-defined control registers are required. In our proposed SPI module, we defined the control registers for the communication between the SPI and the coprocessor, which stores the instruction code for fast data communication. SPI controller has registers such as SPI control register, status register, a counter for data exchange, 8-bit TX data buffer, 8-bit RX data buffer, test register, and the special register for pin sharing with GPIO module.

In this paper, we designed a high speed SPI module for embedded systems, especially for the Bluetooth baseband system. We proposed our SPI design to conform to AMBA APB bus architecture for low power consumption. The designed SPI module was automatically synthesized, placed and routed with Synopsys synthesis tool. We implemented the result circuit with an Altera FPGA, StatixII.

The FPGA implemented module operated well at 25Mhz clock frequency, which was within our frequency boundary.

Acknowledgement

This research was supported in part by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2014R1A1A2A16053925) and in part by the 2010 Sabbatical year grant in aid from Mokwon University.

References

[1] http://www.bluetooth.com.
[2] Bluetooth Special interest Group, “Specifications of the Bluetooth System”, vol. 1, v.4.0 ‘Core’ and vol. 2 v.4.0 ‘Profiles’, **(2010)**.
[3] A. Das, “Adaptive link-level error recovery mechanisms in Bluetooth”, IEEE International Conference on Personal Wireless Communications, **(2000)**.
[4] C. H. Park, “Coexistence mechanism based on adaptive frequency hopping for interference-limited WPAN applications”, 7th International Symposium on Signal Processing and Its Applications, **(2003)**.

Author

Sangook Moon, he received his Bachelor’s degree, the Master’s degree, and the Ph.D. degree in the Department of Electrical and Electronic Engineering from Yonsei University, Seoul, Korea in 1995, 1997, and 2002, respectively. From 2002 to 2004, he was with the Hynix Semiconductor, Seoul, Korea, where he developed Bluetooth baseband SoCs. Since 2004 he has been with the Department of Electronic Engineering at Mokwon University, Daejeon, Korea, where he is currently serves as an Associate Professor. His research interests are in computer architecture, embedded systems, SoCs, data encryption, and computer arithmetic.