SCUBA-2 Block Specification

rx_protocol_fsm

April 8, 2004

Table of Contents

1. Block Overview

1.1 Block Location and Block Interface Within System

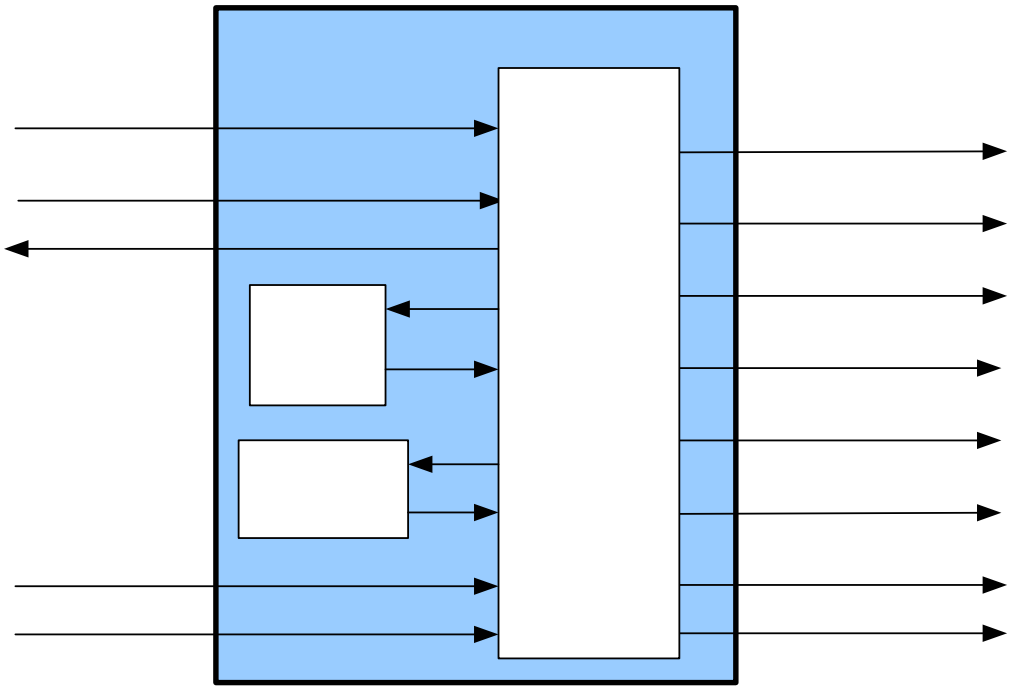
This block is located on the clock card FPGA. It interfaces with the following blocks on the clock card:

- rx_fifo
- cmd_translate
- reply_data_fsm

1.2 Block Functionality / Features

- Monitors “rx_fifo” block for incoming commands.
- Checks all incoming commands are preceded by the appropriate preamble (otherwise disregarded).
- Reads in entire command and compares the sent checksum with locally calculated checksum. If they do not match the output line ‘cksum_err’ is asserted to enable block reply_data_fsm to return an error reply.
- If the checksum is correct then the command is made available to subsequent blocks. The command code, card address, register address, and number of data words, all have separate output lines. The cmd_rdy_o line is asserted high when all these outputs are ready to be processed by subsequent blocks.
- The data word/words are clocked out sequentially on cmd_data_o. The ‘num_data_o’ output indicates the number of data words (0x01 for all commands other than write block). The cmd_rdy_o is asserted high during the entire data clocking out process (see Section 2.2).

1.3 Block Dataflow



2. Block Interfaces

2.1 Interface Signal Description

Table 1: Interface Signals

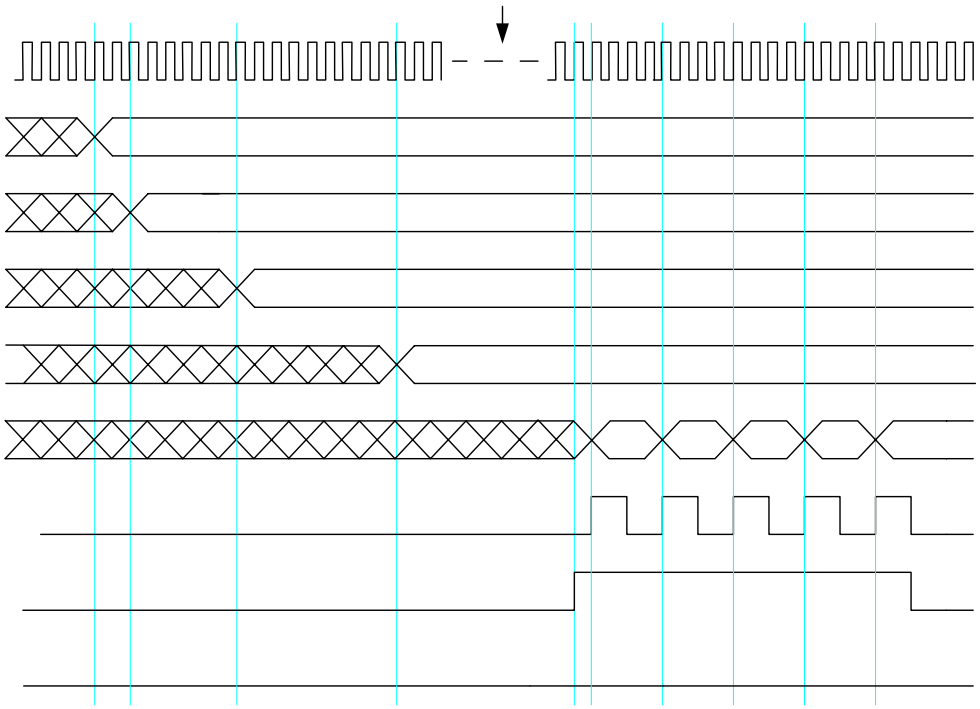
Signal	Description	Direction
Global Signals		
Clk	Global clock signal	in
Brst	Global asynchronous active-high reset	in
Control Signals		
rx_fe_i	Receive FIFO empty line. Monitored to determine when a command is present to be processed.	in
rx_fr_o	Receive FIFO request line. Asserted to get next byte of command. The byte is read through input 'rxd_i'.	out
data_clk_o	Data Clock output. This is used to indicate when a new data word is present on cmd_data_o (only clocked once for a standard command, but clocked several times for write_block command, depending on the number of valid data words).	out
cmd_rdy_o	Command Ready. This is asserted high after a command has been successfully read in by the FSM with no errors in the checksum. It indicates that cmd_code_o, card_addr_o, reg_addr_o, and num_data_o are available to be read by subsequent blocks. In the case of the write block command it is asserted for the entire time that data is being clocked out.	out
cksum_err_o	Checksum error. If the transmitted checksum does not match the locally calculated one then this line is asserted. The reply_data_fsm block would then initiate an error reply message to the host.	out
Data Signals		
rxd_i	byte wide data from FIFO received on request (by asserting rx_fr_o).	in
cmd_code_o	16-bit command code.	out
card_addr_o	8-bit card address.	out
reg_addr_o	24-bit register address.	out
num_data_o	8-bit output indicating the number of valid data words to be clocked out on the 'cmd_data_o' output. This is 0x01 for all commands other than the write_block command.	out
cmd_data_o	16-bit command data. One word for all commands other than write_block. For the write_block command xN words are clocked out of this port (on positive going edge of data_clk_o)	out

2.2 Interface Protocol and Timing

Consider the following write_block command, which contains 5 words of valid data:

Word 1: Preamble 1	0x A5A5A5A5
Word 2: Preamble 2	0x 5A5A5A5A
Word 3: WB Command	0x 20205742
Word 4: Address (card + address)	0x 01ABCDEF
Word 5: Data Valid	0x 00000005
Word 6: Data V1	0x 00001111
Word 7: Data V2	0x 00002222
Word 8: Data V3	0x 00003333
Word 9: Data V4	0x 00004444
Word 10: Data V5	0x 00005555
Word 11..63: Data V6..V58	0x 00000000
Word 64: Checksum	Sequential XOR of Words 3 → Word 64

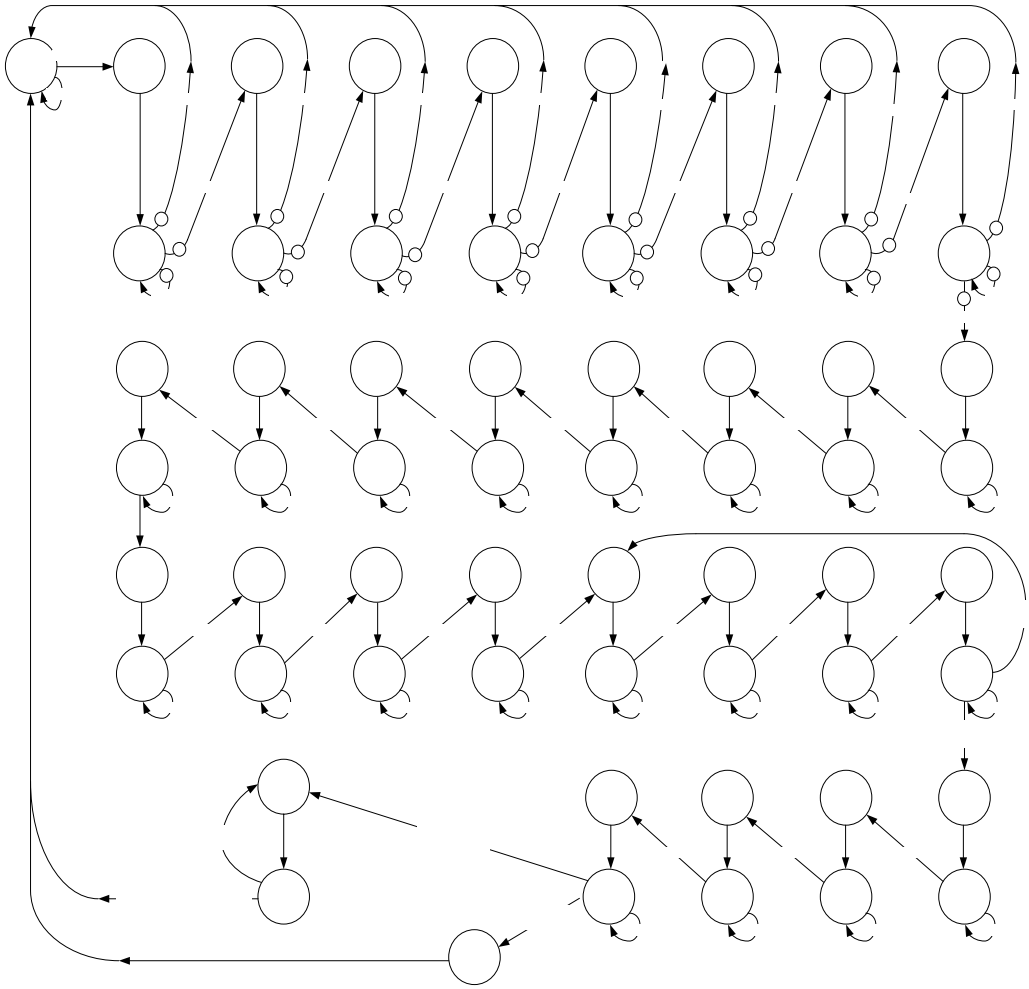
This command would be buffered byte by byte (little endian) in the receiver FIFO ‘rx_fifo’ block. Once the rx_protocol_fsm has read the bytes and processed the command its outputs would be as follows (assuming there is no checksum error):



Note that if an error was found in the checksum, then ‘cksum_err_o’ would be asserted high so that an error reply could be generated, while ‘cmd_rdy_o’ and ‘data_clk_o’ would remain low.

3. High-Level Description

3.1 State Machine Description



rx_fe_i = ‘0’ IDLE RQ_ PRE0 RQ_ PRE1

3.2 Internal Signal Description

Table 2: Internal Signals

rx_fe_i /= ‘0’ rxd_i /= X“A5" rx

Signal	Description
	rx_fe_i = ‘0’
	1

1

CK_ PRE0 2 CK_ PRE1

3 3

rx_fe_i /= ‘0’ rx_f

3.3 State Machine Outputs

STATE		OUTPUTS	
		PORTS	**INTERNAL SIGNALS**
Default Assignments		cksum_err_o <= ‘0’; cmd_rdy_o <= ‘0’; rx_fr_o <= ‘0’; data_clk_o <= ‘0’;	write_mem <= ‘0’; read_mem <= ‘0’; reset_mem <= ‘0’; check_update <= ‘0’; check_reset <= ‘0’;
IDLE	000000		number_data <= 1; reset_mem <= ‘1’; check_reset <= ‘1’; cksum_rcvd <= X“00000000” cksum_in <= X “00000000”
RQ_PRE0	000001	rx_fr_o <= '1' ;	
CK_PRE0	000010		
RQ_PRE1	000011	rx_fr_o <= '1' ;	
CK_PRE1	000100		
RQ_PRE2	000101	rx_fr_o <= '1' ;	
CK_PRE2	000110		
RQ_PRE3	000111	rx_fr_o <= '1' ;	
CK_PRE3	001000		
RQ_PRE4	001001	rx_fr_o <= '1' ;	
CK_PRE4	001010		
RQ_PRE5	001011	rx_fr_o <= '1' ;	
CK_PRE5	001100		
RQ_PRE6	001101	rx_fr_o <= '1' ;	
CK_PRE6	001110		
RQ_PRE7	001111	rx_fr_o <= '1' ;	
CK_PRE7	010000		
RQ_CMD0	010001	rx_fr_o <= '1' ;	
LD_CMD0	010010	cmd_code_o(7 downto 0) <= rxd_i(7 downto 0);	cksum_in(7 downto 0) <= rxd_i(7 downto 0); command(7 downto 0) <= rxd_i(7 downto 0);

RQ_CMD1	010011	rx_fr_o <= '1' ;	
LD_CMD1	010100	cmd_code_o(15 downto 8) <= rxd_i(7 downto 0);	cksum_in(15 downto 8) <= rxd_i(7 downto 0); command(15 downto 8) <= rxd_i(7 downto 0);
RQ_CMD2	010101	rx_fr_o <= '1' ;	
LD_CMD2	010110		cksum_in(23 downto 16) <= rxd_i(7 downto 0);
RQ_CMD3	010111	rx_fr_o <= '1' ;	
LD_CMD3	011000		cksum_in(31 downto 24) <= rxd_i(7 downto 0); check_update <= '1';
RQ_ADDR0	011001	rx_fr_o <= '1' ;	
LD_ADDR0	011010	reg_addr_o(7 downto 0) <= rxd_i(7 downto 0);	cksum_in(7 downto 0) <= rxd_i(7 downto 0);
RQ_ADDR1	011011	rx_fr_o <= '1' ;	
LD_ADDR1	011100	reg_addr_o(15 downto 8) <= rxd_i(7 downto 0);	cksum_in(15 downto 8) <= rxd_i(7 downto 0);
RQ_ADDR2	011101	rx_fr_o <= '1' ;	
LD_ADDR2	011110	reg_addr_o(23 downto 16) <= rxd_i(7 downto 0);	cksum_in(23 downto 16) <= rxd_i(7 downto 0);
RQ_ADDR3	011111	rx_fr_o <= '1' ;	
LD_ADDR3	100000	card_addr_o(7 downto 0) <= rxd_i(7 downto 0);	cksum_in(31 downto 24) <= rxd_i(7 downto 0); check_update <= '1';
RQ_CKSM0	100001	rx_fr_o <= '1' ;	
LD_CKSM0	100010		cksum_rcvd(7 downto 0) <= rxd_i(7 downto 0);
RQ_CKSM1	100011	rx_fr_o <= '1' ;	
LD_CKSM1	100100		cksum_rcvd(15 downto 8) <= rxd_i(7 downto 0);
RQ_CKSM2	100101	rx_fr_o <= '1' ;	
LD_CKSM2	100110		cksum_rcvd(23 downto 16) <= rxd_i(7 downto 0);
RQ_CKSM3	100111	rx_fr_o <= '1' ;	
LD_CKSM3	101000		cksum_rcvd(31 downto 24) <= rxd_i(7 downto 0);
RQ_NDA0	101001	rx_fr_o <= '1' ;	
LD_NDA0	101010	num_data_o <= rxd_i(7 downto 0);	cksum_in(7 downto 0) <= rxd_i(7 downto 0);

			number_data <= To_integer(Unsigned(rxd_i(7 downto 0)));
RQ_NDA1	101011	rx_fr_o <= '1' ;	
LD_NDA1	101100		cksum_in(15 downto 8) <= rxd_i(7 downto 0);
RQ_NDA2	101101	rx_fr_o <= '1' ;	
LD_NDA2	101110		cksum_in(23 downto 16) <= rxd_i(7 downto 0);
RQ_NDA3	101111	rx_fr_o <= '1' ;	
LD_NDA3	110000		cksum_in(31 downto 24) <= rxd_i(7 downto 0); check_update <= '1';
RQ_BLK0	110001	rx_fr_o <= '1' ;	
LD_BLK0	110010		cksum_in(7 downto 0) <= rxd_i(7 downto 0); data_in (7 downto 0) <= rxd_i(7 downto 0);
RQ_BLK1	110011	rx_fr_o <= '1' ;	
LD_BLK1	110100		cksum_in(15 downto 8) <= rxd_i(7 downto 0); data_in (15 downto 8) <= rxd_i(7 downto 0);
RQ_BLK2	110101	rx_fr_o <= '1' ;	
LD_BLK2	110110		cksum_in(23 downto 16) <= rxd_i(7 downto 0);
RQ_BLK3	110111	rx_fr_o <= '1' ;	
LD_BLK3	111000		cksum_in(31 downto 24) <= rxd_i(7 downto 0); write_mem <= '1'; check_update <= '1';
CKSM_FAIL	111001	cksum_err_o <= '1' ;	
READ_DATA	111010	cmd_rdy_o <= '1' ; data_clk_o <= '0' ;	read_mem <= '1';
TX_DATA	111011	cmd_data_o <= data_out ; cmd_rdy_o <= '1' ; data_clk_o <= '1' ;	

4. Files of the Block

4.1 Source Code

4.1.1 rx_control_fsm.vhd