//==========================================================================================================

// RK05 Emulator

// Processor SPI Interface

// File Name: spi\_interface.v

// Functions:

// read and write FPGA hardware control registers.

// read and write SDRAM data.

// write SDRAM address register for processor SDRAM accesses.

//

//==========================================================================================================

module spi\_interface(

input wire clock, // master clock 40 MHz

input wire reset, // active high synchronous reset input

input wire spi\_clk, // SPI clock

input wire spi\_cs\_n, // SPI active low chip select

input wire spi\_mosi, // SPI controller data output, peripheral data input

input wire [15:0] dram\_readdata, // 16-bit read data from DRAM controller

input wire dram\_readack, // dram read acknowledge ==== maybe no longer needed?? ============

input wire dram\_writeack, // dram read acknowledge

input wire BUS\_WT\_PROTECT\_L, // write protect strobe from the bus

input wire [7:0] Cylinder\_Address, // input to be able to read the Cylinder Address

input wire Head\_Select, // input to be able to read the Head Select bit

input wire Selected\_Ready, // input to be able to read Selected\_Ready

input wire BUS\_RK11D\_L, // RK11 mode for Drive Select BUS\_RK11D\_L

input wire [3:0] BUS\_SEL\_DR\_L, // Drive Select signals

input wire [7:0] BUS\_CYL\_ADD\_L, // Cylinder Address

input wire BUS\_STROBE\_L, // Strobe to enable movement, gates the Cylinder Address or Restore line

input wire BUS\_HEAD\_SELECT\_L, // Head Select

//BUS\_WT\_PROTECT\_L, // Write Protect

input wire BUS\_WT\_DATA\_CLK\_L, // Composite write data and write clock

input wire BUS\_WT\_GATE\_L, // Write gate, when active enables write circuitry

input wire BUS\_RESTORE\_L, // Restore, moves heads to cylinder 0

input wire BUS\_RD\_GATE\_L, // Read gate, when active enables read circuitry

input wire [7:0] major\_version,

input wire [7:0] minor\_version,

input wire [4:0] Sector\_Address, // Sector Address to be read test visibility mode

input wire strobe\_selected\_ready,

input wire read\_selected\_ready,

input wire write\_selected\_ready,

input wire clkenbl\_1usec, // 1 usec clock enable input from the timing generator

input wire SPARE\_PIO1\_24,

output reg spi\_miso, // SPI controller data input, peripheral data output

output reg load\_address\_spi, // enable from SPI to command the sdram controller to load address 8 bits at a time

output reg [7:0] spi\_serpar\_reg, // 8-bit serpar register used for writing to the sdram address register

output reg dram\_read\_enbl\_spi, // read enable request to DRAM controller

output reg dram\_write\_enbl\_spi, // write enable request to DRAM controller

output reg [15:0] dram\_writedata\_spi, // 16-bit write data to DRAM controller

output reg [2:0] Drive\_Address, // 3-bits internal drive address

output reg File\_Ready, // disk contents have been copied from the microSD to the SDRAM.

output reg Write\_Protect, // CPU register that indicates the drive write protect status.

output reg Fault\_Latch, // included for future support. Software will always write zero to Fault\_Latch.

output reg cpu\_dc\_low, // DC low signal driven by a CPU register

output reg [7:0] test\_reg\_1,

output reg [7:0] test\_reg\_2,

output reg [11:0] test\_reg\_3,

//output reg [7:0] postamble\_length,

output wire [4:0] number\_of\_sectors\_minus\_1,

output reg [7:0] bitclockdivider\_clockphase,

output reg [7:0] bitclockdivider\_dataphase,

output reg [7:0] bitpulse\_width,

output reg [15:0] microseconds\_per\_sector,

output reg interface\_test\_mode,

output reg command\_interrupt,

output reg Servo\_Pulse\_FPGA,

output reg atten\_mode\_enable

);

//============================ Internal Connections ==================================

//`define EMULATOR\_FPGA\_CODE\_VERSION 8'h51 // hex number that indicates the version of the FPGA code

reg [7:0] spiserialreg;

reg [3:0] metaspi;

reg [2:0] metawprot;

reg dramwrite\_lowhigh;

reg dramread\_lowhigh;

reg [4:0] spicount; // define as 5 bits instead of 4 to prevent the first bit from wrapping around at the end of transmitting the 16-bit data (8 addr + 8 data)

reg [7:0] serialaddress;

wire [7:0] muxed\_read\_data;

wire pre\_spi\_miso;

reg frdlyd;

reg toggle\_wp;

reg [1:0] operation\_id;

reg [4:0] number\_of\_sectors;

//reg [7:0] servo\_pw;

//reg [10:0] counter\_servo\_20ms\_period; // counts 1250 16-usec intervals for the 20 ms servo pulse period

//reg [3:0] counter\_16usec;

wire spi\_start;

//============================ Start of Code =========================================

// SB\_DFFS - D Flip-Flop, Set is asynchronous to the clock.

SB\_DFFS SPI\_DFFS\_inst (

.Q(spi\_start), // Registered Output, "Q" output of the DFF

.C(~spi\_clk), // rising-edge Clock, so with ~spi\_clk as the input, Q changes on the falling edge of spi\_clk

.D(1'b0), // Data, clocks in a zero on the falling edge of spi\_clk

.S(spi\_cs\_n) // Asynchronous active-high Set, we perform async set of the DFF while spi\_cs\_n is inactive

);

//assign number\_of\_sectors\_minus\_1 = (number\_of\_sectors < 2) ? 1 : number\_of\_sectors - 1;

assign number\_of\_sectors\_minus\_1 = number\_of\_sectors - 1;

assign muxed\_read\_data = (serialaddress == 8'h81) ? Cylinder\_Address[7:0] :

((serialaddress == 8'h82) ? {Sector\_Address[3:0], operation\_id[1:0], Selected\_Ready, Head\_Select} :

((serialaddress == 8'h83) ? {7'h00, Sector\_Address[4]} :

//((serialaddress == 8'h84) ? 8'h00 :

//((serialaddress == 8'h85) ? 8'h00 :

((serialaddress == 8'h89) ? {1'b0, 7'h0} : // bit 7 == 0 identifies the FPGA as an emulator, bits 6:0 are presently unused

((serialaddress == 8'h90) ? major\_version[7:0] :

((serialaddress == 8'h91) ? minor\_version[7:0] :

((serialaddress == 8'h94) ? BUS\_CYL\_ADD\_L[7:0] :

((serialaddress == 8'h95) ? {BUS\_RD\_GATE\_L, BUS\_RESTORE\_L, BUS\_WT\_GATE\_L, BUS\_WT\_DATA\_CLK\_L, BUS\_WT\_PROTECT\_L, BUS\_HEAD\_SELECT\_L, BUS\_STROBE\_L, BUS\_RK11D\_L} :

((serialaddress == 8'h96) ? {3'b000, SPARE\_PIO1\_24, BUS\_SEL\_DR\_L[3:0]} :

((serialaddress == 8'ha0) ? {cpu\_dc\_low, 1'b0, Fault\_Latch, File\_Ready, 1'b0, Drive\_Address[2:0]} : // read-back of register 0x0

//((serialaddress == 8'ha7) ? test\_reg\_1\_length[7:0] :

//((serialaddress == 8'ha8) ? test\_reg\_2\_length[7:0] :

//((serialaddress == 8'ha9) ? {4'h0, test\_reg\_3[11:8]} :

//((serialaddress == 8'haa) ? test\_reg\_3[7:0] :

//((serialaddress == 8'hab) ? postamble\_length[7:0] :

((serialaddress == 8'hac) ? {3'b000, number\_of\_sectors[4:0]} : // commented out to reduce PLBs consumed so the design fits in the part

((serialaddress == 8'had) ? bitclockdivider\_clockphase[7:0] :

((serialaddress == 8'hae) ? bitclockdivider\_dataphase[7:0] :

((serialaddress == 8'haf) ? bitpulse\_width[7:0] :

//((serialaddress == 8'hb0) ? microseconds\_per\_sector[15:8] : // removed microseconds\_per\_sector in v2.9 to reduce PLBs consumed

//((serialaddress == 8'hb1) ? microseconds\_per\_sector[7:0] :

((serialaddress == 8'h88) ? (dramread\_lowhigh ? dram\_readdata[15:8] : dram\_readdata[7:0]) : 8'b0))))))))))))));

// dram\_readdata[15:0] always has the data ready that was read at the dram\_address.

// The DRAM word read function is triggered after the odd byte is read.

// The next word is requested after reading the high byte from register 0x88.

assign pre\_spi\_miso = ((spicount == 5'd7) & muxed\_read\_data[7]) |

((spicount == 5'd8) & muxed\_read\_data[6]) |

((spicount == 5'd9) & muxed\_read\_data[5]) |

((spicount == 5'd10) & muxed\_read\_data[4]) |

((spicount == 5'd11) & muxed\_read\_data[3]) |

((spicount == 5'd12) & muxed\_read\_data[2]) |

((spicount == 5'd13) & muxed\_read\_data[1]) |

((spicount == 5'd14) & muxed\_read\_data[0]);

always @ (posedge spi\_clk)

begin : SPICLKPOSFUNCTIONS // block name

// Reset the SPI bit counter using the DFF that is set when spi\_cs\_n is inactive

// The SPI bit counter is used by a mux to serialize the SPI read data.

spicount <= spi\_start ? 5'd0 : spicount + 1;

serialaddress <= (spicount == 6) ? {spiserialreg[6:0], spi\_mosi} : serialaddress;

if(spi\_cs\_n == 1'b0) begin

spiserialreg[7:0] <= {spiserialreg[6:0], spi\_mosi};

end

else begin

spiserialreg[7:0] <= 8'hff;

end

end

always @ (negedge spi\_clk)

begin : SPICLKNEGFUNCTIONS // block name

spi\_miso <= pre\_spi\_miso;

end

always @ (posedge spi\_cs\_n)

begin : SPICSPOSFUNCTIONS // block name

spi\_serpar\_reg <= spiserialreg;

end

always @ (posedge clock)

begin : HSCLOCKFUNCTIONS // block name

if(reset == 1'b1) begin

dram\_read\_enbl\_spi <= 1'b0;

dram\_write\_enbl\_spi <= 1'b0;

Drive\_Address <= 3'd0;

File\_Ready <= 1'b0;

frdlyd <= 1'b0;

Write\_Protect <= 1'b0;

Fault\_Latch <= 1'b0;

load\_address\_spi <= 1'b0;

dramwrite\_lowhigh <= 1'b0;

dramread\_lowhigh <= 1'b0;

metaspi <= 4'b0000;

metawprot <= 3'b000;

cpu\_dc\_low <= 1'b0;

atten\_mode\_enable <= 1'b0;

toggle\_wp <= 1'b0;

test\_reg\_1 <= 8'd0;

test\_reg\_2 <= 8'd0;

test\_reg\_3 <= 12'h0;

number\_of\_sectors[4:0] <= 5'd16; //default for RK8-E, 16 sectors or 0x10

bitclockdivider\_clockphase <= 8'd14; //default for RK8-E, system clock divided by 14 for half a bit, clock phase

bitclockdivider\_dataphase <= 8'd14; //default for RK8-E, system clock divided by 14 for half a bit, data phase

bitpulse\_width <= 8'd6; //default for RK8-E, clock and data pulse width is 6 system clocks

microseconds\_per\_sector <= 16'd2500; //default for RK8-E, 2.5 msec per sector, 2.5 \* 16 = 40 msec per revolution, 0x09c4

interface\_test\_mode <= 1'b0;

operation\_id <= 2'b00;

command\_interrupt <= 1'b0;

//servo\_pw <= 8'd47; // 0.75 msec is 47, 16 usec intervals

//counter\_servo\_20ms\_period <= 11'h0;

//counter\_16usec <= 4'h0;

Servo\_Pulse\_FPGA <= 1'b0;

end

else begin

//counter\_16usec <= clkenbl\_1usec ? counter\_16usec + 1 : counter\_16usec;

//counter\_servo\_20ms\_period <= (clkenbl\_1usec & (counter\_16usec == 15)) ?

//((counter\_servo\_20ms\_period < 11'd1249) ? counter\_servo\_20ms\_period + 1 : 11'd0) : counter\_servo\_20ms\_period;

//Servo\_Pulse\_FPGA <= ({3'h0, servo\_pw[7:0]} > counter\_servo\_20ms\_period);

Servo\_Pulse\_FPGA <= 1'b0;

command\_interrupt <= strobe\_selected\_ready || read\_selected\_ready || write\_selected\_ready;

operation\_id <= strobe\_selected\_ready ? 2'h0 : (read\_selected\_ready ? 2'h1 : (write\_selected\_ready ? 2'h2 : operation\_id));

frdlyd <= File\_Ready;

metaspi[3:0] <= {metaspi[2:0], ~spi\_cs\_n};

metawprot[2:0] <= {metawprot[1:0], (~BUS\_WT\_PROTECT\_L & Selected\_Ready)};

//clear Write\_Protect when there's a change in File\_Ready

//Write\_Protect <= metawprot[2] | // if metawp then set wp

//(~(File\_Ready ^ frdlyd) & ~metawprot[2] & (serialaddress == 8'h04) & ~metaspi[2] & metaspi[3] & spi\_serpar\_reg[0] & ~Write\_Protect) |

//(~(File\_Ready ^ frdlyd) & ~metawprot[2] & ((serialaddress == 8'h04) | metaspi[2] | ~metaspi[3] | ~spi\_serpar\_reg[0]) & Write\_Protect);

// Q <= (Q | Set) & ~Reset

// Set when metawprot[2] or (toggle\_wp & ~Q)

// Reset when (toggle\_wp & Q) or (File\_Ready ^ frdlyd)

toggle\_wp <= (serialaddress == 8'h04) & ~metaspi[2] & metaspi[3] & spi\_serpar\_reg[0]; //toggle\_wp is separated only so the code is more readable

Write\_Protect <= (Write\_Protect | (metawprot[2] | (toggle\_wp & ~Write\_Protect))) & ~((toggle\_wp & Write\_Protect) | (File\_Ready ^ frdlyd));

// register address 0x00

Drive\_Address[2:0] <= ((serialaddress == 8'h00) && ~metaspi[2] && metaspi[3]) ? spi\_serpar\_reg[2:0] : Drive\_Address[2:0];

File\_Ready <= ((serialaddress == 8'h00) && ~metaspi[2] && metaspi[3]) ? spi\_serpar\_reg[4] : File\_Ready;

Fault\_Latch <= ((serialaddress == 8'h00) && ~metaspi[2] && metaspi[3]) ? spi\_serpar\_reg[5] : Fault\_Latch;

atten\_mode\_enable <= ((serialaddress == 8'h00) && ~metaspi[2] && metaspi[3]) ? spi\_serpar\_reg[6] : atten\_mode\_enable;

cpu\_dc\_low <= ((serialaddress == 8'h00) && ~metaspi[2] && metaspi[3]) ? spi\_serpar\_reg[7] : cpu\_dc\_low;

// register address 0x05

load\_address\_spi <= (serialaddress == 8'h05) & ~metaspi[2] & metaspi[3]; // command to load 8 bits of address from SPI

// register address 0x06

dram\_writedata\_spi[7:0] <= ((serialaddress == 8'h06) && ~metaspi[2] && metaspi[3]) ? dram\_writedata\_spi[15:8] : dram\_writedata\_spi[7:0];

dram\_writedata\_spi[15:8] <= ((serialaddress == 8'h06) && ~metaspi[2] && metaspi[3]) ? spi\_serpar\_reg : dram\_writedata\_spi[15:8];

dram\_write\_enbl\_spi <= (serialaddress == 8'h06) & ~metaspi[2] & metaspi[3] & dramwrite\_lowhigh;

// register address 0x07

test\_reg\_1 <= ((serialaddress == 8'h07) && ~metaspi[2] && metaspi[3]) ? spi\_serpar\_reg[7:0] : test\_reg\_1;

// register address 0x08

test\_reg\_2 <= ((serialaddress == 8'h08) && ~metaspi[2] && metaspi[3]) ? spi\_serpar\_reg[7:0] : test\_reg\_2;

// register address 0x09

test\_reg\_3[11:8] <= ((serialaddress == 8'h09) && ~metaspi[2] && metaspi[3]) ? spi\_serpar\_reg[3:0] : test\_reg\_3[11:8];

// register address 0x0a

test\_reg\_3[7:0] <= ((serialaddress == 8'h0a) && ~metaspi[2] && metaspi[3]) ? spi\_serpar\_reg[7:0] : test\_reg\_3[7:0];

// register address 0x0b

//postamble\_length <= ((serialaddress == 8'h0b) && ~metaspi[2] && metaspi[3]) ? spi\_serpar\_reg[7:0] : postamble\_length;

// register address 0x0c

number\_of\_sectors <= ((serialaddress == 8'h0c) && ~metaspi[2] && metaspi[3]) ? spi\_serpar\_reg[4:0] : number\_of\_sectors;

// register address 0x0d

bitclockdivider\_clockphase <= ((serialaddress == 8'h0d) && ~metaspi[2] && metaspi[3]) ? spi\_serpar\_reg[7:0] : bitclockdivider\_clockphase;

// register address 0x0e

bitclockdivider\_dataphase <= ((serialaddress == 8'h0e) && ~metaspi[2] && metaspi[3]) ? spi\_serpar\_reg[7:0] : bitclockdivider\_dataphase;

// register address 0x0f

bitpulse\_width <= ((serialaddress == 8'h0f) && ~metaspi[2] && metaspi[3]) ? spi\_serpar\_reg[7:0] : bitpulse\_width;

// register address 0x10

microseconds\_per\_sector[15:8] <= ((serialaddress == 8'h10) && ~metaspi[2] && metaspi[3]) ? spi\_serpar\_reg[7:0] : microseconds\_per\_sector[15:8];

// register address 0x11

microseconds\_per\_sector[7:0] <= ((serialaddress == 8'h11) && ~metaspi[2] && metaspi[3]) ? spi\_serpar\_reg[7:0] : microseconds\_per\_sector[7:0];

// register address 0x12

//servo\_pw[7:0] <= ((serialaddress == 8'h12) && ~metaspi[2] && metaspi[3]) ? spi\_serpar\_reg[7:0] : servo\_pw[7:0];

// register address 0x20

interface\_test\_mode <= ((serialaddress == 8'h20) && ~metaspi[2] && metaspi[3]) ? (spi\_serpar\_reg[7:0] == 8'h55) : interface\_test\_mode;

// register address 0x88

dram\_read\_enbl\_spi <= (serialaddress == 8'h88) & ~metaspi[2] & metaspi[3] & dramread\_lowhigh;

// dram\_readdata[15:0] always has the data ready that was read at the dram\_address.

// The read function is triggered after the odd byte is read.

// The next word is requested after reading the high byte when the SPI address is 8'h88.

// toggle respective lowhigh bits on a write or read, clear both bits on address load, otherwise lowhigh bits remain the same

dramwrite\_lowhigh <= ((serialaddress == 8'h06) && ~metaspi[2] && metaspi[3]) ? ~dramwrite\_lowhigh :

(((serialaddress == 8'h05) && ~metaspi[2] && metaspi[3]) ? 1'b0 : dramwrite\_lowhigh);

dramread\_lowhigh <= ((serialaddress == 8'h88) && ~metaspi[2] && metaspi[3]) ? ~dramread\_lowhigh :

(((serialaddress == 8'h05) && ~metaspi[2] && metaspi[3]) ? 1'b0 : dramread\_lowhigh);

end

end // End of Block HSCLOCKFUNCTIONS

endmodule // End of Module spi\_interface