SPI.V:

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 2024/10/19 12:59:21

// Design Name:

// Module Name: SPI

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module SPI\_driver(

input wire clk,

output reg [2:0] cur\_state,

input wire SPI\_MISO,

output reg SPI\_MOSI,

output reg SPI\_CLK,

output reg SPI\_EN,

output reg busy,

input wire command\_read,

input wire rx\_read,

input wire tx\_read,

input wire [1:0] Spi\_rw,

output reg [7:0] Spi\_rx\_reg,

input wire [7:0] Spi\_tx\_reg

);

localparam IDLE = 3'b000;

localparam SPITX = 3'b001;

localparam SPIRX = 3'b010;

localparam SPIED = 3'b100;

reg [1:0] command\_FIFO[15:0];

reg [7:0] tx\_FIFO[15:0];

reg [7:0] rx\_FIFO[15:0];

reg [3:0] command\_addrw;

reg [3:0] command\_addrr;

reg [3:0] tx\_addrw;

reg [3:0] tx\_addrr;

reg [3:0] rx\_addrw;

reg [3:0] rx\_addrr;

wire command\_empty;

wire tx\_empty;

wire rx\_empty;

initial begin

cur\_state = 7'd0;

SPI\_MOSI = 1'b0;

SPI\_CLK = 1'b0;

SPI\_EN = 1'b0;

busy = 1'b0;

command\_addrw = 4'b0;

command\_addrr = 4'd0;

tx\_addrw = 4'd0;

tx\_addrr = 4'd0;

rx\_addrw = 4'd0;

rx\_addrr = 4'd0;

end

assign tx\_empty = &(~(tx\_addrr^tx\_addrw));

assign rx\_empty = &(~(rx\_addrr^rx\_addrw));

assign command\_empty = &(~(command\_addrr^command\_addrw));

always @(posedge clk) begin

if (command\_read == 1'b1) begin

command\_FIFO[command\_addrw] <= Spi\_rw;

command\_addrw <= command\_addrw + 1;

end

if (tx\_read == 1'b1) begin

tx\_FIFO[tx\_addrw] <= Spi\_tx\_reg;

tx\_addrw <= tx\_addrw + 1;

end

if (rx\_read == 1'b1) begin

if(rx\_empty != 1'b1) begin

Spi\_rx\_reg <= rx\_FIFO[rx\_addrr];

rx\_addrr <= rx\_addrr + 1;

end else begin

Spi\_rx\_reg <= 8'b11111111;

end

end

end

reg[2:0] bit\_counter;

reg[2:0] clk\_counter;

reg[7:0] rx\_temp\_reg;

always @(posedge clk) begin

case(cur\_state)

IDLE : begin

busy <= 1'b0;

if(command\_empty == 1'b0)begin

if(command\_FIFO[command\_addrr] == 2'b01)begin

command\_addrr <= command\_addrr + 1;

cur\_state <= SPITX;

clk\_counter <= 3'b000;

bit\_counter <= 3'b111;

busy <= 1'b1;

end

//add error detection if the first command out of IDLE is rx, this is incorrectly set by the controller

// also add error detection if when entering TX, check for TX\_FIFO empty, if not, controller is incorrectly set

end

end

SPITX: begin

case(clk\_counter)

3'b000 : begin

SPI\_EN <= 1'b1;

SPI\_CLK <= 1'b0;

SPI\_MOSI <= tx\_FIFO[tx\_addrr][bit\_counter];

busy <= 1'b1;

clk\_counter <= clk\_counter + 1;

end

3'b100 : begin

SPI\_CLK <= 1'b1;

clk\_counter <= clk\_counter + 1;

end

3'b111 : begin

if(bit\_counter != 3'b000) begin

bit\_counter <= bit\_counter -1;

clk\_counter <= 3'b000;

end else begin

bit\_counter <= 3'b111;

clk\_counter <= 3'b000;

tx\_addrr <= tx\_addrr + 1;

if(command\_empty == 1'b1)cur\_state <= SPIED;

else begin

if (command\_FIFO[command\_addrr] == 2'b01) cur\_state <= SPITX;

else cur\_state <= SPIRX;

command\_addrr <= command\_addrr + 1;

//add error detection if 2'b10 or 2'b01 is not the data read from the FIFO

end

end

end

default : begin

clk\_counter <= clk\_counter + 1;

end

endcase

end

SPIRX: begin

case(clk\_counter)

3'b000 : begin

SPI\_EN <= 1'b1;

SPI\_CLK <= 1'b0;

clk\_counter <= clk\_counter + 1;

end

3'b100 : begin

SPI\_CLK <= 1'b1;

rx\_temp\_reg[bit\_counter] <= SPI\_MISO;

clk\_counter <= clk\_counter +1;

end

3'b111 : begin

if(bit\_counter != 3'b000) begin

bit\_counter <= bit\_counter - 1;

clk\_counter <= 3'b000;

end else begin

bit\_counter <= 3'b111;

clk\_counter <= 3'b000;

rx\_FIFO[rx\_addrw] <= rx\_temp\_reg;

rx\_addrw <= rx\_addrw + 1;

if(command\_empty == 1'b1) cur\_state <= SPIED;

else begin

if(command\_FIFO[command\_addrr] == 2'b01) cur\_state <= SPITX;

else cur\_state <= SPIRX;

command\_addrr <= command\_addrr + 1;

end

end

end

default : begin

clk\_counter <= clk\_counter + 1;

end

endcase

end

SPIED : begin

case(clk\_counter)

3'b000 : begin

SPI\_EN <=1'b1;

SPI\_CLK <=1'b0;

clk\_counter <= clk\_counter + 1;

end

3'b100 : begin

SPI\_EN <= 1'b0;

SPI\_CLK <= 1'b0;

SPI\_MOSI <= 1'b0;

clk\_counter <= 3'b000;

cur\_state <= IDLE;

end

default : begin

clk\_counter <= clk\_counter + 1;

end

endcase

end

endcase

end

endmodule

SPI\_controller.v

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 2024/09/22 14:34:42

// Design Name:

// Module Name: TS\_controller

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module SPI\_controller(

input clk,

input wire [31:0] PC\_rx,

input wire [31:0] PC\_addr,

input wire [31:0] PC\_val,

output reg [31:0] PC\_tx,

output reg command\_read,

output reg rx\_read,

output reg tx\_read,

output reg [7:0] tx\_byte,

output reg [1:0] rw,

output reg [9:0] cur\_state,

input wire [7:0] rx\_byte,

input wire busy

);

reg busy\_reg = 0;

reg [7:0] tx\_byte\_reg;

reg [7:0] rx\_byte\_reg;

reg [2:0] read\_counter;

reg [31:0] PC\_rx\_reg1, PC\_rx\_reg2;

localparam idle\_ = 10'b0000000001;

localparam start\_wr = 10'b0000000010;

localparam tx\_wr1 = 10'b0000000100;

localparam tx\_wr2 = 10'b0000001000;

localparam end\_wr = 10'b0000010000;

localparam start\_rt = 10'b0000100000;

localparam tx\_rt = 10'b0001000000;

localparam rx\_rt = 10'b0010000000;

localparam wait\_rt = 10'b0100000000;

localparam end\_rt = 10'b1000000000;

initial begin

cur\_state <= idle\_;

PC\_rx\_reg1 <= 0;

PC\_rx\_reg2 <= 0;

tx\_byte\_reg <= 0;

rx\_byte\_reg <= 0;

busy\_reg <= 1'b0;

read\_counter <= 0;

end

integer i;

always @(posedge clk) begin

for (i=0; i<8; i=i+1) begin

tx\_byte[i] <= tx\_byte\_reg[i];

rx\_byte\_reg[i] <= rx\_byte[i];

end

end

always @(posedge clk) begin

case (cur\_state)

idle\_ : begin

command\_read <= 1'b0;

tx\_read <= 1'b0;

rx\_read <= 1'b0;

PC\_rx\_reg1 <= PC\_rx;

PC\_rx\_reg2 <= PC\_rx\_reg1;

if (PC\_rx\_reg2[0] == 1'b0 && PC\_rx\_reg1[0] == 1'b1) begin

cur\_state <= start\_wr;

end

if (PC\_rx\_reg2[1] == 1'b0 && PC\_rx\_reg1[1] == 1'b1) begin

cur\_state <= start\_rt;

end

end

//Write single byte

start\_wr: begin

tx\_byte\_reg <= {1'b1, PC\_addr[6:0]};

cur\_state <= tx\_wr1;

end

tx\_wr1: begin

tx\_byte\_reg <= PC\_val[7:0];

command\_read <= 1'b1;

rw <= 2'b01;

tx\_read <= 1'b1;

cur\_state <= tx\_wr2;

end

tx\_wr2: begin

command\_read <= 1'b1;

rw <= 2'b01;

tx\_read <= 1'b1;

cur\_state <= end\_wr;

end

end\_wr : begin

tx\_byte\_reg <= {8{1'b0}};

command\_read <= 1'b0;

tx\_read <= 1'b0;

cur\_state <= idle\_;

end

//Read two byte

start\_rt: begin

tx\_byte\_reg <= {1'b0, PC\_addr[6:0]};

cur\_state <= tx\_rt;

end

tx\_rt: begin

command\_read <= 1'b1;

rw <= 2'b01;

tx\_read <= 1'b1;

cur\_state <= rx\_rt;

end

rx\_rt : begin

command\_read <= 1'b1;

tx\_read <= 1'b0;

rw <= 2'b10;

cur\_state <= wait\_rt;

end

wait\_rt : begin

tx\_byte\_reg <= {8{1'b0}};

command\_read <= 1'b0;

rw <= 2'b00;

busy\_reg <= busy;

if (busy\_reg == 1'b1 && busy == 1'b0) begin

rx\_read <= 1'b1;

cur\_state <= end\_rt;

end

end

end\_rt: begin

rx\_read <= 1'b0;

read\_counter <= read\_counter + 1;

if (read\_counter == 2) begin

PC\_tx[7:0] <= rx\_byte\_reg;

read\_counter <= 0;

cur\_state <= idle\_;

end

end

default : begin

tx\_byte\_reg <= {8{1'b0}};

cur\_state <= idle\_;

end

endcase

end

endmodule

Python

# -\*- coding: utf-8 -\*-

#%%

# import various libraries necessary to run your Python code

import time # time related library

import sys,os # system related library

ok\_sdk\_loc = "C:\\Program Files\\Opal Kelly\\FrontPanelUSB\\API\\Python\\x64"

ok\_dll\_loc = "C:\\Program Files\\Opal Kelly\\FrontPanelUSB\\API\\lib\\x64"

sys.path.append(ok\_sdk\_loc) # add the path of the OK library

os.add\_dll\_directory(ok\_dll\_loc)

import ok # OpalKelly library

#%%

def reset\_image\_sensor():

dev.SetWireInValue(0x01, 1)

dev.UpdateWireIns()

time.sleep(1)

dev.SetWireInValue(0x01, 0)

dev.UpdateWireIns()

time.sleep(0.1)

#%%

def write\_to\_device(reg\_addr, value):

dev.SetWireInValue(0x00, 0)

dev.UpdateWireIns()

dev.SetWireInValue(0x02, reg\_addr)

dev.SetWireInValue(0x03, value)

dev.UpdateWireIns() # Update the WireIns

time.sleep(0.2)

dev.SetWireInValue(0x00, 1) # Write trigger

dev.UpdateWireIns() # Update the WireIns

time.sleep(0.2)

dev.SetWireInValue(0x00, 0)

dev.UpdateWireIns() # Update the WireIns

#%%

def read\_from\_device(reg\_addr):

dev.SetWireInValue(0x00, 0)

dev.UpdateWireIns() # Update the WireIns

time.sleep(0.2)

dev.SetWireInValue(0x02, reg\_addr)

dev.SetWireInValue(0x00, 2) # Read trigger

dev.UpdateWireIns() # Update the WireIns

time.sleep(0.2)

dev.UpdateWireOuts()

read = dev.GetWireOutValue(0x20)

# if slave\_addr == 0x3C:

# m\_L = read // 2\*\*8

# m\_H = read - (m\_L \* 2\*\*8)

# read = m\_H \* 2\*\*8 + m\_L

# if read >= 2\*\*15:

# read = read - 2\*\*16 # deal with 2's complement

dev.SetWireInValue(0x00, 0)

dev.UpdateWireIns()

return read

#%%

# Define FrontPanel device variable, open USB communication and

# load the bit file in the FPGA

dev = ok.okCFrontPanel() # define a device for FrontPanel communication

SerialStatus=dev.OpenBySerial("") # open USB communication with the OK board

# We will NOT load the bit file because it will be loaded using JTAG interface from Vivado

# Check if FrontPanel is initialized correctly and if the bit file is loaded.

# Otherwise terminate the program

print("----------------------------------------------------")

if SerialStatus == 0:

print ("FrontPanel host interface was successfully initialized.")

else:

print ("FrontPanel host interface not detected. The error code number is:" + str(int(SerialStatus)))

print("Exiting the program.")

sys.exit ()

#%% Reg and value constants

start1\_h = 3

start1\_l = 4

#%%

# Define the two variables that will send data to the FPGA

# We will use WireIn instructions to send data to the FPGA

time.sleep(1)

reset\_image\_sensor()

write\_to\_device(3, 8)

write\_to\_device(4, 160)

write\_to\_device(57, 3)

write\_to\_device(58, 44)

write\_to\_device(59, 240)

write\_to\_device(60, 10)

write\_to\_device(69, 9)

write\_to\_device(80, 2)

write\_to\_device(83, 187)

write\_to\_device(97, 240)

write\_to\_device(98, 10)

write\_to\_device(100, 112)

write\_to\_device(101, 98)

write\_to\_device(102, 34)

write\_to\_device(103, 64)

write\_to\_device(106, 94)

write\_to\_device(107, 110)

write\_to\_device(108, 91)

write\_to\_device(109, 82)

write\_to\_device(110, 80)

write\_to\_device(117, 91)

print(read\_from\_device(3))

print(read\_from\_device(4))

print(read\_from\_device(57))

print(read\_from\_device(58))

print(read\_from\_device(59))

print(read\_from\_device(60))

print(read\_from\_device(69))

print(read\_from\_device(80))

print(read\_from\_device(83))

print(read\_from\_device(97))

print(read\_from\_device(98))

print(read\_from\_device(100))

print(read\_from\_device(101))

print(read\_from\_device(102))

print(read\_from\_device(103))

print(read\_from\_device(106))

print(read\_from\_device(107))

print(read\_from\_device(108))

print(read\_from\_device(109))

print(read\_from\_device(110))

print(read\_from\_device(117))

dev.Close