**Project 3 - Transmit Engine plus PicoBlaze**

Programmer: Victor Espinoza

SID: 010657450

CECS 460, Section 1; Tu/Th 9:30 - 11:45 A.M.

Lab Section 2

Due: Tuesday, October 20, 2015

Contents

[I. Introduction: 3](#_Toc433091393)

[II. Requirements 3](#_Toc433091394)

[III. Top Level Block Diagram: 4](#_Toc433091395)

[IV. Technical Description: 6](#_Toc433091396)

[V. Assembly Code (PSM File): 7](#_Toc433091397)

[VI. Verification Description: 13](#_Toc433091398)

[VII. Source Code: 14](#_Toc433091399)

# I. Introduction:

The purpose of this lab was to create the transmit engine logic and to include the PicoBlaze into our design. For this lab, I was supposed to continually transmit the sequence "CSULB CECS 460<CR><LF>" to the terminal. The purpose of the PicoBlaze in my design was to output the specified characters after the TxRdy bit was set in the Transmit Engine's status register. This was achieved by reading in the value of the status register by using an Input instruction and then subtracting that value by 2 (which would result in a zero value if the TxRdy bit was set). I would continually monitor the status register of the Transmit Engine and once the TxRdy bit was set, I would then output the appropriate character on the PicoBlaze's out\_port using the Output instruction. I also had to decode the write\_strobe and port\_id value coming out of the PicoBlaze so that I could make sure that the Transmit Engine was getting the right write input. Within the Transmit Engine, I also had various R-S flops to control the TxRdy variable and the start variable. I also had a flop to store the data coming out of the PicoBlaze and a register that delayed the write input to the Transmit Engine by one clock cycle. I also had to implement a shift register, a baud-rate counter, and a bit counter to make sure that I was transmitting data at the appropriate bit time (1/Baud Rate) intervals and in the appropriate format (7N1, 7O1, 7E1, 8N1, 8O1, 8E1). This project was somewhat difficult to verify/implement, but I ended up with a successful program in the end.

# II. Requirements

1. The software to be written is simple, monitor the TXRDY bit in the status register (see below)

2. When you see the TXRDY bit you should continuously transmit the sequence “CSULB CECS 460 <CR><LF>”

# III. Top Level Block Diagram:

-On Next Page

# Top Level Block Diagram.jpg

# IV. Technical Description:

The main addition to this project was the Transmit Engine. My Transmit Engine module contains all of the logic needed to communicate with a computer via universal asynchronous receiver/transmitter (UART) communication. In order to create a stable UART connection, I needed to make sure that I could run my program at different baud rates and different data formats (7N1, 7O1, 7E1, 8N1, 8O1, 8E1). I also needed to make sure that both my program and the terminal software were running at the same baud-rate and data format.

For this program, I am shifting out the data in 12-bit data packets. I use a shift register to determine what data gets shifted out. I am able to load in the data that is going to be shifted out by delaying the write[1] strobe from the PicoBlaze by 1 clock cycle (this is because the data on the out\_port of the PicoBlaze is not available until the next clock cycle). If my write1Delay variable is set high, I then load my shift register with the out\_port data of the PicoBlaze, which I stored using a flop within my Transmit Engine. I then update the remaining values in the shift register to make sure that I have accounted for the right parity, stop, and start bits. This is done using the bit8, parity\_en, and odd\_n\_even inputs and decoding their values so that I know what values to put in the b8, b7 bits of the shift register. Once all of the data in the shift register is accurate, I then start shifting out the bits 1 bit at a time. I fill the most significant bit of my shift register with a 1 (which is equivalent to filling up the register with a bunch of stop bits).

I keep track of when I am done shifting out all of the bits in my shift register by using a bit counter. In order to know when a bit is done being shifted out, I also needed to use a baud-rate counter. I first decoded the baud rate by reading in the data from my four baud\_rate switches. The next step was to determine what baud-rate I was running at based on those four switches. After determining the baud rate, I then needed to determine the amount of ticks I would need to count in order to achieve the specified baud-rate. I did this by using the formula baudCountNum = (1/baud rate) / (1/50MHz). This gave me the terminal count number that I needed to reach in order to run at the desired baud rate. I then assigned my baudCountNum variable to this terminal count value. Once that value was reached, I then knew that I had finished transmitting one bit via UART communication. Now the next step was to increment my Bit Counter. I repeated this process until my Bit Counter reached 11, meaning that I had finished transmitting all 12 bits of data in the data packet. For this project, I am always going to end up transmitting 12 bits of data per data packet. That is why my bit time counter limit is always 11 (0 to (12-1)). Once all 12 bits of data have been shifted out, I then set my Done variable to 1, which also sets my TxRdy variable to a 1. I then output this TxRdy variable, along with the other status variables to the PicoBlaze. These status variables are concatenated and represent the status register of the Transmit Engine. The TxRdy bit lets the PicoBlaze know that the Transmit Engine is ready to transmit another byte, at which point the PicoBlaze will output the next character being transmitted in the sequence to the Transmit Engine.

In order to get the PicoBlaze to successfully transmit the sequence "CSULB CECS 460<CR><LF>, I had to create my own assembly code where I output each character after the TxRdy bit from the status register was set high. Here is the assembly code that I wrote for this project:

# V. Assembly Code (PSM File):

;UART Transmit

;

;================================================================

; data constants

;================================================================

;selected ASCII codes

CONSTANT ASCII\_C\_U , 43 ; Uppercase C

CONSTANT ASCII\_S\_U , 53 ; Uppercase S

CONSTANT ASCII\_U\_U , 55 ; Uppercase U

CONSTANT ASCII\_L\_U , 4C ; Uppercase L

CONSTANT ASCII\_B\_U , 42 ; Uppercase B

CONSTANT ASCII\_Space , 20 ; Space

CONSTANT ASCII\_E\_U , 45 ; Uppercase E

CONSTANT ASCII\_4 , 34 ; number 4

CONSTANT ASCII\_6 , 36 ; number 6

CONSTANT ASCII\_0 , 30 ; number 0

CONSTANT ASCII\_CR , 0D ; carriage return <CR>

CONSTANT ASCII\_LF , 0A ; line feed <LF>

;================================================================

; port aliases

;================================================================

;\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_input port definitions\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

CONSTANT rd\_flag\_port, 00 ;status of transmit engine

NAMEREG sd, tx\_data ;data to be tx by uart

;\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_output port definitions\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

CONSTANT uart\_tx\_port, 01 ;outputs to register 1 (Write\_Strobe[1])

;================================================================

; Main Program

;================================================================

main\_program:

load tx\_data , ASCII\_C\_U

call tx\_one\_byte ; transmit C

load tx\_data , ASCII\_S\_U

call tx\_one\_byte ; transmit S

load tx\_data , ASCII\_U\_U

call tx\_one\_byte ; transmit U

load tx\_data , ASCII\_L\_U

call tx\_one\_byte ; transmit L

load tx\_data , ASCII\_B\_U

call tx\_one\_byte ; transmit B

load tx\_data , ASCII\_Space

call tx\_one\_byte ; transmit space

load tx\_data , ASCII\_C\_U

call tx\_one\_byte ; transmit C

load tx\_data , ASCII\_E\_U

call tx\_one\_byte ; transmit E

load tx\_data , ASCII\_C\_U

call tx\_one\_byte ; transmit C

load tx\_data , ASCII\_S\_U

call tx\_one\_byte ; transmit S

load tx\_data , ASCII\_Space

call tx\_one\_byte ; transmit space

load tx\_data , ASCII\_4

call tx\_one\_byte ; transmit 4

load tx\_data , ASCII\_6

call tx\_one\_byte ; transmit 6

load tx\_data , ASCII\_0

call tx\_one\_byte ; transmit 0

load tx\_data , ASCII\_CR

call tx\_one\_byte ; transmit CR (Carriage Return)

load tx\_data , ASCII\_LF

call tx\_one\_byte ; transmit LF (Line Feed)

JUMP main\_program

;================================================================

; routine : tx\_one\_byte

; function : Wait until uart TxRdy bit is set, which signifies that the UART is

; ready to transmit another byte. Then transmit another byte to

; the UART.

; Input Register : tx\_data

; Temp Register : s6 - read port flags

;

;================================================================

tx\_one\_byte :

input s6, rd\_flag\_port ;read in status of the Transmit Engine

sub s6, 02 ;check to see if TxRdy bit is set high

jump nz, tx\_one\_byte ;if it isn't, keep on waiting until it is set high

output tx\_data, uart\_tx\_port ;If it is set high, then transmit byte to uart

return

My assembly code is really simple. I first define all of the constant values for the different ascii characters that I am going to be transmitting to the terminal. I then load the desired value that I want to output to the PicoBlaze's outport. I then call the task tx\_one\_byte, which is a simple task that outputs the appropriate register value for the ASCII character to the PicoBlaze's output. Before I output the value, however, I have to make sure that the TxRdy bit is set high within the Transmit Engine, which signifies that the engine is ready to transmit the next byte. I achieve this by sending the status register of the Transmit Engine (which has the TxRdy bit as the second bit in the register) to the in\_port of the PicoBlaze. I use the input s6, rd\_flag\_port instruction to store the value of the status register into the internal PicoBlaze register s6. I then subtract 02 from this value in order to check to see if the TxRdy bit is set high. If the TxRdy bit is not set high, that means that the Transmit Engine is not done transmitting the current data yet, so I keep on looping back to the beginning of my tx\_one\_byte task, where I continually read in the status register values and wait until the TxRdy bit is set high, at which point I output the value to the PicoBlaze's out\_port using the Output instruction. I output the character value to the port\_id 01 because we want to use the write[1] write\_strobe for our transmitting signal. I repeat this process indefinitely, which results in the sequence of "CSULB CECS 460<CR><LF>" continually being outputted from the PicoBlaze one character at a time. After a character is outputted, it is then loaded into the shift register of the Transmit Engine and then shifted out to the terminal. Once all 12 bits of the shift register are shifted out, the Done bit and the TxRdy bit are set high and then the PicoBlaze outputs the next character in the sequence. This process repeats itself indefinitely, which results in the sequence "CSULB CECS 460<CR><LF> being constantly transmitted to the terminal.

# VI. Verification Description:

For this lab's verification I created a test bench module for both my top\_level and my transmit\_engine files. For the top\_level test bench, all I did was initialize my inputs and set my baud-rate / data format to be 9600 / 8O1. I then ran the simulation and used the waveforms to verify that my Transmit Engine was storing/shifting the correct values out and that the PicoBlaze was outputting the character values at the appropriate times. The Transmit Engine test bench was a lot more involved than my top\_level test bench. In the Transmit Engine test bench, I made sure to test out a variety of data values and I also made sure that I was able to successfully run at all of the different baud-rates specified for my project. I also made sure that each data format (7N1, 7O1, 7E1, 8N1, 8O1m 8E1) worked correctly as well. I verified this by making sure that each 12-bit packet was transmitted in the correct order and by making sure that the appropriate values were being loaded into my shift register I also compared the desired shift register values to the actual shift register values and outputted the result to the console in order to help me debug my shift register and make sure that it was working properly. For a more detailed version of each test bench and its verification, refer to the header section of each test bench module within the source code. It is there that I go into further detail about how I verified the completeness of my design.

# VII. Source Code:

-Starts on next page

**Top\_level Test Bench:**

`timescale 1ns / 1ps

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

//

// Author: Victor Espinoza

// Email: victor.alfonso94@gmail.com

// Project #: Project 3 - Transmit Engine plus PicoBlaze

// Course: CECS 460

//

// Create Date: 15:34:08 10/18/2015

//

// Module Name: proj3\_TransmitEngine\_tb

// File Name: proj3\_TransmitEngine\_tb.v

//

// Description: This test bench is really simple. I simply initialize my clk

// and rstb inputs, set my baud rate to 9600, create a format

// of 8O1 for transmitting data through the UART, and then

// deactivate my low active rstb input. I then checked the

// waveform data to make sure that my program was continually

// transmitting the "CSULB CECS 460<CR><LF>" sequence. The

// PicoBlaze was in charge of providing the right character

// data at the appropriate intervals (when the TxRdy bit of the

// status registe was set high). This was done using assembly

// code which is provided in my report. I verified that the

// correct data was being sent out by checking the waveform

// data and making sure that the proper data was being loaded

// into the shift register and that the bits were shifting out

// in the correct order. Once I verified this, I then concluded

// that my program worked exactly how I intended it to.

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

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

module proj3\_TransmitEngine\_tb;

//Inputs

reg clk;

reg rstb;

reg bit8;

reg parity\_en;

reg odd\_n\_even;

reg [3:0] baud\_val;

//Outputs

wire tx;

//Instantiate the Unit Under Test (UUT)

//module proj3\_TransmitEngine(clk, rstb, bit8, parity\_en, odd\_n\_even, baud\_val,

//tx);

proj3\_TransmitEngine uut(

.clk(clk),

.rstb(rstb),

.bit8(bit8),

.parity\_en(parity\_en),

.odd\_n\_even(odd\_n\_even),

.baud\_val(baud\_val),

.tx(tx)

);

//vary the clk signal every 10ns to mimick a

//period of 20ns (which is the period of our boards)

always #10 clk = ~clk;

//always #20 tx\_Write = ~tx\_Write;

initial begin

//Initialize Inputs

clk = 0;

rstb = 1; //low active reset

bit8 = 1; //8 bits of data

parity\_en = 1; //parity enabled

odd\_n\_even = 1; //odd parity

baud\_val = 4'h4; //baud = 9600

//Wait 100 ns for global reset to finish

#100 @(posedge clk) rstb = 0; // have reset become unactive.

end

endmodule

**Top\_level:**

`timescale 1ns / 1ps

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

//

// Author: Victor Espinoza

// Email: victor.alfonso94@gmail.com

// Project #: Project 3 - Transmit Engine plus PicoBlaze

// Course: CECS 460

//

// Create Date: 11:04:58 10/08/2015

//

// Module Name: proj3\_TransmitEngine

// File Name: proj3\_TransmitEngine.v

//

// Description: This top\_level module basically ties in all of the other

// modules that I made and connects them together. It takes the

// primary inputs of clk, rstb, bit8, parity\_en, odd\_n\_even, and

// baud\_val[3:0]. This project has one output called tx. This tx

// output goes to P9, which is a port that connects to the RS-232

// serial connector that connects to a UART terminal on a

// computer via a serial cable. The clk input comes from the

// clock on the Nexys 2 board, while the rstb input is a push

// button. The bit8, parity\_en, odd\_n\_even, and baud\_val[3:0]

// inputs are all switches on the Nexys 2 board. The rstb input

// is synchronized and distributed to all of the other modules in

// this project. This synchronized rstb value (rstbs) is also

// negated and then used as the reset input to the PicoBlaze

// processor. Inside of the PicoBlaze, I read in the status of

// the transmit engine and wait until it is ready to transit

// another byte by subtracting this value by 02 (the location

// of the TxRdy bit of the status register). Once the TxRdy

// bit is set, I then know that the Transmit Engine is ready to

// transmit another byte and I then output this value to the

// PicoBlaze. On the second clk cycle of the output instruction,

// I decode the Write\_Strobe value by using the Write\_Strobe and

// the Port\_Id port of the picoblaze. I then put these values

// into my write[255:0] register (because there are 2^8 registers

// that can be written to the PicoBlaze: 00 - FF). For my design,

// I output my values to the Port\_ID of 01, so I needed to pass

// my write[1] value into my transmit engine to let it know

// that it should load in the appropriate data bits (Out\_Port)

// into the shift register. The baud\_val[3:0] bits determine the

// baud rate at which the Program is going to be transmitting

// characters. The bit8, parity\_en and odd\_n\_even inputs

// determine the format of the data bits being transmitted (7N1,

// 7E1, 7O1, 8N1, 8O1, and 8E1). All of these inputs go into my

// transmit engine where I update my shift register and baud

// counter accordingly. Once I establish the desired baud rate

// and data format, I then open a terminal window under the same

// baud rate and data format and then I am ready to start

// transmitting characters to the UART terminal. In the PicoBlaze,

// I continually output the values of "CSULB CECS 460<CR><LF>".

// This results in the sequence of "CSULB CECS 460<CR><LF> being

// continually transmitted to the UART Terminal.

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

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

module proj3\_TransmitEngine(clk, rstb, bit8, parity\_en, odd\_n\_even, baud\_val, tx);

//Input and Output Declarations

input clk, rstb, bit8, parity\_en, odd\_n\_even;

input [3:0] baud\_val;

output wire tx;

//Local Declarations

//wire rstsb, TxSet, TxRst, Done, sSet, sRst, ld, btu, shift;

wire rstsb, tx\_out;

wire [7:0] in\_port;

wire interrupt, read\_strobe, write\_strobe, interrupt\_ack;

wire [7:0] out\_port;

wire [7:0] port\_id;

reg [255:0] write;

assign tx = tx\_out;

//synch\_reset module instantiation

//module synch\_reset(clk, rstb, rstsb);

synch\_reset Synchronizer\_Circuit(

.clk(clk),

.rstb(rstb),

.rstsb(rstsb)

);

//instantiate the picoblaze

embedded\_kcpsm3 ekcp3(

.port\_id(port\_id),

.write\_strobe(write\_strobe),

.read\_strobe(read\_strobe),

.out\_port(out\_port),

.in\_port(in\_port),

.interrupt(interrupt),

.interrupt\_ack(interrupt\_ack),

.reset(!rstsb),

.clk(clk)

);

assign interrupt = 0;

//write strobe decode

always@(\*)begin

write = 0;

write[port\_id] = write\_strobe;

end

//transmit\_engine module instantiation

//module transmit\_engine(clk, rstb, bit8, parity\_en, odd\_n\_even, baud\_val, txWrite,

//inData, tx, pico\_data);

transmit\_engine transmit(

.clk(clk),

.rstb(rstsb),

.bit8(bit8),

.parity\_en(parity\_en),

.odd\_n\_even(odd\_n\_even),

.baud\_val(baud\_val),

.txWrite(write[1]),

.inData(out\_port),

.tx(tx\_out),

.pico\_data(in\_port)

);

endmodule

**AISO:**

`timescale 1ns / 1ps

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

//

// Author: Victor Espinoza

// Email: victor.alfonso94@gmail.com

// Project #: Project 3 - Transmit Engine plus PicoBlaze

// Course: CECS 460

// Create Date: 19:08:36 04/09/2015

//

// Module Name: synch\_reset

// File Name: synch\_reset.v

//

// Description: This is the Syncronization Circuit module that has an

// asynchronous input and a synchronous output. The purpose of this

// module is to take the asyncronous reset input and make it low-

// leveled active when pressed. This means that the reset value

// should be zero whenever the button is activated. When the reset

// button is pressed on the Nexys 2 board, its value comes into

// this module as the rstb input. However, in this module there are

// flip-flops that are designed with asynchronous (low active)

// resets. Also in this module, there is a synchronous reset that

// will neglect the asynchronous rstb input and force the 2

// flip-flops to zero at the posedge of the clock. This is how the

// rstb input turns from an input of 1 to an output of 0 and

// vice-versa. In other words, this is how the asynchronous rstb

// input turns into the synchronous rstsb output.

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

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

module synch\_reset(clk, rstb, rstsb);

//Input and Output declarations

input clk, rstb;

output wire rstsb;

//local variables / flip-flop declarations

reg rst\_reg\_delay;

reg rst\_synch\_out;

assign rstsb = rst\_synch\_out;

always@(posedge clk, posedge rstb)begin

if(rstb)begin //force rstb to be low when activated instead of being high

rst\_reg\_delay <= 1'b0;

rst\_synch\_out <= 1'b0;

end

else begin //Synchronize the output

rst\_reg\_delay <= 1'b1;

rst\_synch\_out <= rst\_reg\_delay;

end

end

endmodule

**Processor:**

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

// Copyright (c) 2004 Xilinx, Inc.

// All Rights Reserved

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

// \_\_\_\_ \_\_\_\_

// / /\/ /

// /\_\_\_/ \ / Vendor: Xilinx

// \ \ \/ Version: 1.01

// \ \ Filename: embedded\_kcpsm3.v

// / / Date Last Modified: 08/04/2004

// /\_\_\_/ /\ Date Created: 06/03/2003

// \ \ / \

// \\_\_\_\/\\_\_\_\

//

//Device: Xilinx

//Purpose:

// This file instantiates the KCPSM3 processor macro and connects the

// program ROM.

//Reference:

// None

//Revision History:

// Rev 1.00 - kc - Start of design entry in VHDL, 06/03/2003.

// Rev 1.01 - sus - Converted to verilog, 08/04/2004.

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

// NOTE: The name of the program ROM will probably need to be changed to

// reflect the name of the program (PSM) file applied to the assembler.

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

// Contact: e-mail picoblaze@xilinx.com

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

//

// Disclaimer:

// LIMITED WARRANTY AND DISCLAIMER. These designs are

// provided to you "as is". Xilinx and its licensors make and you

// receive no warranties or conditions, express, implied,

// statutory or otherwise, and Xilinx specifically disclaims any

// implied warranties of merchantability, non-infringement, or

// fitness for a particular purpose. Xilinx does not warrant that

// the functions contained in these designs will meet your

// requirements, or that the operation of these designs will be

// uninterrupted or error free, or that defects in the Designs

// will be corrected. Furthermore, Xilinx does not warrant or

// make any representations regarding use or the results of the

// use of the designs in terms of correctness, accuracy,

// reliability, or otherwise.

//

// LIMITATION OF LIABILITY. In no event will Xilinx or its

// licensors be liable for any loss of data, lost profits, cost

// or procurement of substitute goods or services, or for any

// special, incidental, consequential, or indirect damages

// arising from the use or operation of the designs or

// accompanying documentation, however caused and on any theory

// of liability. This limitation will apply even if Xilinx

// has been advised of the possibility of such damage. This

// limitation shall apply not-withstanding the failure of the

// essential purpose of any limited remedies herein.

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

module embedded\_kcpsm3(

port\_id,

write\_strobe,

read\_strobe,

out\_port,

in\_port,

interrupt,

interrupt\_ack,

reset,

clk);

output[7:0] port\_id;

output write\_strobe;

output read\_strobe;

output[7:0] out\_port;

input[7:0] in\_port;

input interrupt;

output interrupt\_ack;

input reset;

input clk;

wire [7:0] port\_id;

wire write\_strobe;

wire read\_strobe;

wire [7:0] out\_port;

wire [7:0] in\_port;

wire interrupt;

wire interrupt\_ack;

wire reset;

wire clk;

wire [9:0] address;

wire [17:0] instruction;

//----------------------------------------------------------------------------------

//

// declaration of KCPSM3

//

//

// declaration of program ROM

//

//----------------------------------------------------------------------------------

//

// Start of test circuit description

//

kcpsm3 processor

( .address(address),

.instruction(instruction),

.port\_id(port\_id),

.write\_strobe(write\_strobe),

.out\_port(out\_port),

.read\_strobe(read\_strobe),

.in\_port(in\_port),

.interrupt(interrupt),

.interrupt\_ack(interrupt\_ack),

.reset(reset),

.clk(clk));

//

// prog\_rom program

// ( .address(address),

// .instruction(instruction),

// .clk(clk));

uart\_tx program

( .address(address),

.instruction(instruction),

.clk(clk));

endmodule

//----------------------------------------------------------------------------------

//

// END OF FILE EMBEDDED\_KCPSM3.V

//

//----------------------------------------------------------------------------------

**Processor (just part of code):**

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

// Copyright (c) 2004 Xilinx, Inc.

// All Rights Reserved

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

// \_\_\_\_ \_\_\_\_

// / /\/ /

// /\_\_\_/ \ / Vendor: Xilinx

// \ \ \/ Version: 1.30

// \ \ Filename: kcpsm3.v

// / / Date Last Modified: August 5 2004

// /\_\_\_/ /\ Date Created: May 19 2003

// \ \ / \

// \\_\_\_\/\\_\_\_\

//

//Device: Xilinx

//Purpose:

// Constant (K) Coded Programmable State Machine for Spartan-3 Devices.

// Also suitable for use with Virtex-II and Virtex-IIPRO devices.

//

// Includes additional code for enhanced verilog simulation.

//

// Instruction disassembly concept inspired by the work of Prof. Dr.-Ing. Bernhard Lang.

// University of Applied Sciences, Osnabrueck, Germany.

//

// Format of this file.

// --------------------

// This file contains the definition of KCPSM3 as one complete module This 'flat'

// approach has been adopted to decrease

// the time taken to load the module into simulators and the synthesis process.

//

// The module defines the implementation of the logic using Xilinx primitives.

// These ensure predictable synthesis results and maximise the density of the implementation.

//

//Reference:

// None

//Revision History:

// Rev 1.00 - kc - Start of design entry, May 19 2003.

// Rev 1.20 - njs - Converted to verilog, July 20 2004.

// Verilog version creation supported by Chip Lukes,

// Advanced Electronic Designs, Inc.

// www.aedbozeman.com,

// chip.lukes@aedmt.com

// Rev 1.21 - sus - Added text to adhere to HDL standard, August 4 2004.

// Rev 1.30 - njs - Updated as per VHDL version 1.30 August 5 2004.

//

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

// Contact: e-mail picoblaze@xilinx.com

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

//

// Disclaimer:

// LIMITED WARRANTY AND DISCLAIMER. These designs are

// provided to you "as is". Xilinx and its licensors make and you

// receive no warranties or conditions, express, implied,

// statutory or otherwise, and Xilinx specifically disclaims any

// implied warranties of merchantability, non-infringement, or

// fitness for a particular purpose. Xilinx does not warrant that

// the functions contained in these designs will meet your

// requirements, or that the operation of these designs will be

// uninterrupted or error free, or that defects in the Designs

// will be corrected. Furthermore, Xilinx does not warrant or

// make any representations regarding use or the results of the

// use of the designs in terms of correctness, accuracy,

// reliability, or otherwise.

//

// LIMITATION OF LIABILITY. In no event will Xilinx or its

// licensors be liable for any loss of data, lost profits, cost

// or procurement of substitute goods or services, or for any

// special, incidental, consequential, or indirect damages

// arising from the use or operation of the designs or

// accompanying documentation, however caused and on any theory

// of liability. This limitation will apply even if Xilinx

// has been advised of the possibility of such damage. This

// limitation shall apply not-withstanding the failure of the

// essential purpose of any limited remedies herein.

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

`timescale 1 ps / 1ps

module kcpsm3(

address,

instruction,

port\_id,

write\_strobe,

out\_port,

read\_strobe,

in\_port,

interrupt,

interrupt\_ack,

reset,

clk) ;

output [9:0] address ;

input [17:0] instruction ;

output [7:0] port\_id ;

output write\_strobe, read\_strobe, interrupt\_ack ;

output [7:0] out\_port ;

input [7:0] in\_port ;

input interrupt, reset, clk ;

//

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

//

// Start of Main Architecture for KCPSM3

//

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

//

// Signals used in KCPSM3

//

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

//

// Fundamental control and decode signals

//

wire t\_state ;

wire not\_t\_state ;

wire internal\_reset ;

wire reset\_delay ;

wire move\_group ;

wire condition\_met ;

wire normal\_count ;

wire call\_type ;

wire push\_or\_pop\_type ;

wire valid\_to\_move ;

//

// Flag signals

//

wire flag\_type ;

wire flag\_write ;

wire flag\_enable ;

wire zero\_flag ;

wire sel\_shadow\_zero ;

wire low\_zero ;

wire high\_zero ;

wire low\_zero\_carry ;

wire high\_zero\_carry ;

wire zero\_carry ;

wire zero\_fast\_route ;

wire low\_parity ;

wire high\_parity ;

wire parity\_carry ;

wire parity ;

wire carry\_flag ;

wire sel\_parity ;

wire sel\_arith\_carry ;

wire sel\_shift\_carry ;

wire sel\_shadow\_carry ;

wire [3:0] sel\_carry ;

wire carry\_fast\_route ;

**Memory:**

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

// Copyright (c) 2004 Xilinx, Inc.

// All Rights Reserved

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

// \_\_\_\_ \_\_\_\_

// / /\/ /

// /\_\_\_/ \ / Vendor: Xilinx

// \ \ \/ Version: v1.30

// \ \ Application : KCPSM3

// / / Filename: uart\_tx.v

// /\_\_\_/ /\

// \ \ / \

// \\_\_\_\/\\_\_\_\

//

//Command: kcpsm3 uart\_tx.psm

//Device: Spartan-3, Spartan-3E, Virtex-II, and Virtex-II Pro FPGAs

//Design Name: uart\_tx

//Generated 18Oct2015-16:18:06.

//Purpose:

// uart\_tx verilog program definition.

//

//Reference:

// PicoBlaze 8-bit Embedded Microcontroller User Guide

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

`timescale 1 ps / 1ps

module uart\_tx (address, instruction, clk);

input [9:0] address;

input clk;

output [17:0] instruction;

RAMB16\_S18 ram\_1024\_x\_18(

.DI (16'h0000),

.DIP (2'b00),

.EN (1'b1),

.WE (1'b0),

.SSR (1'b0),

.CLK (clk),

.ADDR (address),

.DO (instruction[15:0]),

.DOP (instruction[17:16]))

/\*synthesis

init\_00 = "00210D4500210D4300210D2000210D4200210D4C00210D5500210D5300210D43"

init\_01 = "00210D0A00210D0D00210D3000210D3600210D3400210D2000210D5300210D43"

init\_02 = "0000000000000000000000000000000000000000A000CD015421C60246004000"

init\_03 = "0000000000000000000000000000000000000000000000000000000000000000"

init\_04 = "0000000000000000000000000000000000000000000000000000000000000000"

init\_05 = "0000000000000000000000000000000000000000000000000000000000000000"

init\_06 = "0000000000000000000000000000000000000000000000000000000000000000"

init\_07 = "0000000000000000000000000000000000000000000000000000000000000000"

init\_08 = "0000000000000000000000000000000000000000000000000000000000000000"

init\_09 = "0000000000000000000000000000000000000000000000000000000000000000"

init\_0A = "0000000000000000000000000000000000000000000000000000000000000000"

init\_0B = "0000000000000000000000000000000000000000000000000000000000000000"

init\_0C = "0000000000000000000000000000000000000000000000000000000000000000"

init\_0D = "0000000000000000000000000000000000000000000000000000000000000000"

init\_0E = "0000000000000000000000000000000000000000000000000000000000000000"

init\_0F = "0000000000000000000000000000000000000000000000000000000000000000"

init\_10 = "0000000000000000000000000000000000000000000000000000000000000000"

init\_11 = "0000000000000000000000000000000000000000000000000000000000000000"

init\_12 = "0000000000000000000000000000000000000000000000000000000000000000"

init\_13 = "0000000000000000000000000000000000000000000000000000000000000000"

init\_14 = "0000000000000000000000000000000000000000000000000000000000000000"

init\_15 = "0000000000000000000000000000000000000000000000000000000000000000"

init\_16 = "0000000000000000000000000000000000000000000000000000000000000000"

init\_17 = "0000000000000000000000000000000000000000000000000000000000000000"

init\_18 = "0000000000000000000000000000000000000000000000000000000000000000"

init\_19 = "0000000000000000000000000000000000000000000000000000000000000000"

init\_1A = "0000000000000000000000000000000000000000000000000000000000000000"

init\_1B = "0000000000000000000000000000000000000000000000000000000000000000"

init\_1C = "0000000000000000000000000000000000000000000000000000000000000000"

init\_1D = "0000000000000000000000000000000000000000000000000000000000000000"

init\_1E = "0000000000000000000000000000000000000000000000000000000000000000"

init\_1F = "0000000000000000000000000000000000000000000000000000000000000000"

init\_20 = "0000000000000000000000000000000000000000000000000000000000000000"

init\_21 = "0000000000000000000000000000000000000000000000000000000000000000"

init\_22 = "0000000000000000000000000000000000000000000000000000000000000000"

init\_23 = "0000000000000000000000000000000000000000000000000000000000000000"

init\_24 = "0000000000000000000000000000000000000000000000000000000000000000"

init\_25 = "0000000000000000000000000000000000000000000000000000000000000000"

init\_26 = "0000000000000000000000000000000000000000000000000000000000000000"

init\_27 = "0000000000000000000000000000000000000000000000000000000000000000"

init\_28 = "0000000000000000000000000000000000000000000000000000000000000000"

init\_29 = "0000000000000000000000000000000000000000000000000000000000000000"

init\_2A = "0000000000000000000000000000000000000000000000000000000000000000"

init\_2B = "0000000000000000000000000000000000000000000000000000000000000000"

init\_2C = "0000000000000000000000000000000000000000000000000000000000000000"

init\_2D = "0000000000000000000000000000000000000000000000000000000000000000"

init\_2E = "0000000000000000000000000000000000000000000000000000000000000000"

init\_2F = "0000000000000000000000000000000000000000000000000000000000000000"

init\_30 = "0000000000000000000000000000000000000000000000000000000000000000"

init\_31 = "0000000000000000000000000000000000000000000000000000000000000000"

init\_32 = "0000000000000000000000000000000000000000000000000000000000000000"

init\_33 = "0000000000000000000000000000000000000000000000000000000000000000"

init\_34 = "0000000000000000000000000000000000000000000000000000000000000000"

init\_35 = "0000000000000000000000000000000000000000000000000000000000000000"

init\_36 = "0000000000000000000000000000000000000000000000000000000000000000"

init\_37 = "0000000000000000000000000000000000000000000000000000000000000000"

init\_38 = "0000000000000000000000000000000000000000000000000000000000000000"

init\_39 = "0000000000000000000000000000000000000000000000000000000000000000"

init\_3A = "0000000000000000000000000000000000000000000000000000000000000000"

init\_3B = "0000000000000000000000000000000000000000000000000000000000000000"

init\_3C = "0000000000000000000000000000000000000000000000000000000000000000"

init\_3D = "0000000000000000000000000000000000000000000000000000000000000000"

init\_3E = "0000000000000000000000000000000000000000000000000000000000000000"

init\_3F = "0000000000000000000000000000000000000000000000000000000000000000"

initp\_00 = "000000000000000000000000000000000000000000000AD3CCCCCCCCCCCCCCCC"

initp\_01 = "0000000000000000000000000000000000000000000000000000000000000000"

initp\_02 = "0000000000000000000000000000000000000000000000000000000000000000"

initp\_03 = "0000000000000000000000000000000000000000000000000000000000000000"

initp\_04 = "0000000000000000000000000000000000000000000000000000000000000000"

initp\_05 = "0000000000000000000000000000000000000000000000000000000000000000"

initp\_06 = "0000000000000000000000000000000000000000000000000000000000000000"

initp\_07 = "0000000000000000000000000000000000000000000000000000000000000000" \*/;

// synthesis translate\_off

// Attributes for Simulation

defparam ram\_1024\_x\_18.INIT\_00 = 256'h00210D4500210D4300210D2000210D4200210D4C00210D5500210D5300210D43;

defparam ram\_1024\_x\_18.INIT\_01 = 256'h00210D0A00210D0D00210D3000210D3600210D3400210D2000210D5300210D43;

defparam ram\_1024\_x\_18.INIT\_02 = 256'h0000000000000000000000000000000000000000A000CD015421C60246004000;

defparam ram\_1024\_x\_18.INIT\_03 = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_04 = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_05 = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_06 = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_07 = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_08 = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_09 = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_0A = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_0B = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_0C = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_0D = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_0E = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_0F = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_10 = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_11 = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_12 = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_13 = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_14 = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_15 = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_16 = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_17 = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_18 = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_19 = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_1A = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_1B = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_1C = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_1D = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_1E = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_1F = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_20 = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_21 = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_22 = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_23 = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_24 = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_25 = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_26 = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_27 = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_28 = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_29 = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_2A = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_2B = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_2C = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_2D = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_2E = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_2F = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_30 = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_31 = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_32 = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_33 = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_34 = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_35 = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_36 = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_37 = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_38 = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_39 = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_3A = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_3B = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_3C = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_3D = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_3E = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INIT\_3F = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INITP\_00 = 256'h000000000000000000000000000000000000000000000AD3CCCCCCCCCCCCCCCC;

defparam ram\_1024\_x\_18.INITP\_01 = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INITP\_02 = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INITP\_03 = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INITP\_04 = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INITP\_05 = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INITP\_06 = 256'h0000000000000000000000000000000000000000000000000000000000000000;

defparam ram\_1024\_x\_18.INITP\_07 = 256'h0000000000000000000000000000000000000000000000000000000000000000;

// synthesis translate\_on

// Attributes for XST (Synplicity attributes are in-line)

// synthesis attribute INIT\_00 of ram\_1024\_x\_18 is "00210D4500210D4300210D2000210D4200210D4C00210D5500210D5300210D43"

// synthesis attribute INIT\_01 of ram\_1024\_x\_18 is "00210D0A00210D0D00210D3000210D3600210D3400210D2000210D5300210D43"

// synthesis attribute INIT\_02 of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000A000CD015421C60246004000"

// synthesis attribute INIT\_03 of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_04 of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_05 of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_06 of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_07 of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_08 of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_09 of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_0A of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_0B of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_0C of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_0D of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_0E of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_0F of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_10 of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_11 of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_12 of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_13 of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_14 of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_15 of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_16 of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_17 of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_18 of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_19 of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_1A of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_1B of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_1C of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_1D of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_1E of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_1F of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_20 of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_21 of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_22 of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_23 of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_24 of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_25 of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_26 of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_27 of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_28 of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_29 of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_2A of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_2B of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_2C of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_2D of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_2E of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_2F of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_30 of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_31 of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_32 of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_33 of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_34 of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_35 of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_36 of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_37 of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_38 of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_39 of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_3A of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_3B of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_3C of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_3D of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_3E of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INIT\_3F of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INITP\_00 of ram\_1024\_x\_18 is "000000000000000000000000000000000000000000000AD3CCCCCCCCCCCCCCCC"

// synthesis attribute INITP\_01 of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INITP\_02 of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INITP\_03 of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INITP\_04 of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INITP\_05 of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INITP\_06 of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

// synthesis attribute INITP\_07 of ram\_1024\_x\_18 is "0000000000000000000000000000000000000000000000000000000000000000"

endmodule

// END OF FILE uart\_tx.v

**Transmit Engine Test Bench:**

`timescale 1ns / 1ps

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

//

// Author: Victor Espinoza

// Email: victor.alfonso94@gmail.com

// Project #: Project 3 - Transmit Engine plus PicoBlaze

// Course: CECS 460

//

// Create Date: 14:41:51 10/15/2015

//

// Module Name: transmit\_engine\_tb

// File Name: transmit\_engine\_tb.v

//

// Description: For this test bench I first initialized my Transmit Engine

// to be running at a 300 Baud Rate and at a 7N1 format. I then

// made sure that the correct value was getting loaded into my

// shift register and outputted the result of this comparison

// to the console. I then waited until the Done bit was set high,

// which signified that I was finished transmitting the 12 bits

// of data. I then loaded up the next byte of data, changed the

// baud rate to 1200, kept the 7N1 format, and repeated the

// process mentioned above. I made sure to transmit different

// combinations of data and to test out every format and

// baud rate available in my design. Once I verified that all

// of the data was being loaded and shifted out correctly at

// each of the different baud rates and data formats, I then

// concluded that my transmit engine was working correctly.

//

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

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

module transmit\_engine\_tb;

//Inputs

reg clk;

reg rstb;

reg bit8;

reg parity\_en;

reg odd\_n\_even;

reg [3:0] baud\_val;

reg tx\_Write;

reg [7:0] in\_data;

//Outputs

wire tx; //transferLimitReached;

wire [7:0] pico\_data;

//Instantiate the Unit Under Test (UUT)

//module transmit\_engine(clk, rstb, bit8, parity\_en, odd\_n\_even, baud\_val,

//txWrite, inData, tx, pico\_data);

transmit\_engine uut(

.clk(clk),

.rstb(rstb),

.bit8(bit8),

.parity\_en(parity\_en),

.odd\_n\_even(odd\_n\_even),

.baud\_val(baud\_val),

.txWrite(tx\_Write),

.inData(in\_data),

.tx(tx),

.pico\_data(pico\_data)

);

//vary the clk signal every 10ns to mimick a

//period of 20ns (which is the period of our boards)

always #10 clk = ~clk;

//always #20 tx\_Write = ~tx\_Write;

initial begin

//Initialize Inputs

clk = 0;

rstb = 0; //low active reset

//7N1 (300 Baud) Transmitting 0x65 = 110\_0101

bit8 = 0; //7 bits of data

parity\_en = 0; //parity disabled

odd\_n\_even = 0; //no parity

baud\_val = 4'h0; //baud = 300

in\_data = 8'h65; //data to be transmitted

//Wait 100 ns for global reset to finish

#100 @(posedge clk) rstb = 1; // have reset become unactive.

tx\_Write = 1; //assert tx\_Write after reset.

#20 @(posedge clk) tx\_Write = 0;

@(posedge clk) if(uut.write1Delay == 1)begin

#20; //wait for next clock cycle (when data is updated)

if (uut.sreg\_data != 12'hf95) //7N1 tx -> 0x65 = f95

$display("Incorrect Data! Expected: 0xf95 , Actual: 0x%h",

uut.sreg\_data[11:0]);

else

$display("Correct Data! Expected: 0xf95 , Actual: 0x%h",

uut.sreg\_data[11:0]);

end

wait (uut.Done == 1); //wait until byte is done being transmitted.

wait (uut.TxRdy == 1); //wait until transmit engine is ready

//to receive another byte.

wait (uut.Done == 0); //wait until byte is done being transmitted.

//7N1 (1200 Baud) Transmitting 0x54 = 101\_0100

bit8 = 0; //7 bits of data

parity\_en = 0; //parity disabled

odd\_n\_even = 1; //no parity

baud\_val = 4'h1; //baud = 1200

in\_data = 8'h54; //data to be transmitted

tx\_Write = 1; //assert tx\_Write

#20 @(posedge clk) tx\_Write = 0; //deassert tx\_Write

@(posedge clk) if(uut.write1Delay == 1)begin

#20; //wait for next clock cycle (when data is updated)

if (uut.sreg\_data != 12'hf51) //7N1 tx -> 0x54 = f51

$display("Incorrect Data! Expected: 0xf51 , Actual: 0x%h",

uut.sreg\_data[11:0]);

else

$display("Correct Data! Expected: 0xf51 , Actual: 0x%h",

uut.sreg\_data[11:0]);

end

wait (uut.Done == 1); //wait until byte is done being transmitted.

wait (uut.TxRdy == 1); //wait until transmit engine is ready

//to receive another byte.

wait (uut.Done == 0); //wait until byte is done being transmitted.

//7E1 (2400 Baud) Transmitting 0x72 = 111\_0010

bit8 = 0; //7 bits of data

parity\_en = 1; //parity enabled

odd\_n\_even = 0; //even parity

tx\_Write = 1; //assert tx\_Write

baud\_val = 4'h2; //baud = 2400

in\_data = 8'h72; //data to be transmitted

#20 @(posedge clk) tx\_Write = 0; //deassert tx\_Write

@(posedge clk) if(uut.write1Delay == 1)begin

#20; //wait for next clock cycle (when data is updated)

if (uut.sreg\_data != 12'hdc9) //7E1 tx -> 0x72 = dc9

$display("Incorrect Data! Expected: 0xdc9 , Actual: 0x%h",

uut.sreg\_data[11:0]);

else

$display("Correct Data! Expected: 0xdc9 , Actual: 0x%h",

uut.sreg\_data[11:0]);

end

wait (uut.Done == 1); //wait until byte is done being transmitted.

wait (uut.TxRdy == 1); //wait until transmit engine is ready

//to receive another byte.

wait (uut.Done == 0); //wait until byte is done being transmitted.

//7O1 (4800 Baud) Transmitting 0x4F = 100\_1111

bit8 = 0; //7 bits of data

parity\_en = 0; //parity disabled

odd\_n\_even = 1; //no parity

tx\_Write = 1; //assert tx\_Write

baud\_val = 4'h3; //baud = 4800

in\_data = 8'h4F; //data to be transmitted

#20 @(posedge clk) tx\_Write = 0; //deassert tx\_Write

@(posedge clk) if(uut.write1Delay == 1)begin

#20; //wait for next clock cycle (when data is updated)

if (uut.sreg\_data != 12'hf3d) //7O1 tx -> 0x4f = f3d

$display("Incorrect Data! Expected: 0xf3d , Actual: 0x%h",

uut.sreg\_data[11:0]);

else

$display("Correct Data! Expected: 0xf3d , Actual: 0x%h",

uut.sreg\_data[11:0]);

end

wait (uut.Done == 1); //wait until byte is done being transmitted.

wait (uut.TxRdy == 1); //wait until transmit engine is ready

//to receive another byte.

wait (uut.Done == 0);

//8N1 (9600 Baud) Transmitting 0x4F = 0100\_1111

bit8 = 1; //8 bits of data

parity\_en = 0; //parity enabled

odd\_n\_even = 0; //no parity

tx\_Write = 1; //assert tx\_Write

baud\_val = 4'h4; //baud = 9600

in\_data = 8'h4F; //data to be transmitted

#20 @(posedge clk) tx\_Write = 0; //deassert tx\_Write

@(posedge clk) if(uut.write1Delay == 1)begin

#20; //wait for next clock cycle (when data is updated)

if (uut.sreg\_data != 12'hd3d) //8N1 tx -> 0x4f = d3d

$display("Incorrect Data! Expected: 0xd3d , Actual: 0x%h",

uut.sreg\_data[11:0]);

else

$display("Correct Data! Expected: 0xd3d , Actual: 0x%h",

uut.sreg\_data[11:0]);

end

wait (uut.Done == 1); //wait until byte is done being transmitted.

wait (uut.TxRdy == 1); //wait until transmit engine is ready

//to receive another byte.

wait (uut.Done == 0);

//8N1 (19200 Baud) Transmitting 0xAE = 1010\_1110

bit8 = 1; //8 bits of data

parity\_en = 0; //parity enabled

odd\_n\_even = 1; //no parity

tx\_Write = 1; //assert tx\_Write

baud\_val = 4'h5; //baud = 19200

in\_data = 8'hAE; //data to be transmitted

#20 @(posedge clk) tx\_Write = 0; //deassert tx\_Write

@(posedge clk) if(uut.write1Delay == 1)begin

#20; //wait for next clock cycle (when data is updated)

if (uut.sreg\_data != 12'heb9) //8N1 tx -> 0xAE = eb9

$display("Incorrect Data! Expected: 0xeb9 , Actual: 0x%h",

uut.sreg\_data[11:0]);

else

$display("Correct Data! Expected: 0xeb9 , Actual: 0x%h",

uut.sreg\_data[11:0]);

end

wait (uut.Done == 1); //wait until byte is done being transmitted.

wait (uut.TxRdy == 1); //wait until transmit engine is ready

//to receive another byte.

wait (uut.Done == 0);

//8E1 (38400 Baud) Transmitting 0xBD = 1011\_1101

bit8 = 1; //8 bits of data

parity\_en = 1; //parity enabled

odd\_n\_even = 0; //even parity

tx\_Write = 1; //assert tx\_Write

baud\_val = 4'h6; //baud = 38400

in\_data = 8'hBD; //data to be transmitted

#20 @(posedge clk) tx\_Write = 0; //deassert tx\_Write

@(posedge clk) if(uut.write1Delay == 1)begin

#20; //wait for next clock cycle (when data is updated)

if (uut.sreg\_data != 12'haf5) //8E1 tx -> 0xBD = af5

$display("Incorrect Data! Expected: 0xaf5 , Actual: 0x%h",

uut.sreg\_data[11:0]);

else

$display("Correct Data! Expected: 0xaf5 , Actual: 0x%h",

uut.sreg\_data[11:0]);

end

wait (uut.Done == 1); //wait until byte is done being transmitted.

wait (uut.TxRdy == 1); //wait until transmit engine is ready

//to receive another byte.

wait (uut.Done == 0);

//8O1 (57600 Baud) Transmitting 0x4F = 0000\_1000

bit8 = 1; //8 bits of data

parity\_en = 1; //parity enabled

odd\_n\_even = 1; //odd parity

tx\_Write = 1; //assert tx\_Write

baud\_val = 4'h7; //baud = 57600

in\_data = 8'h08; //data to be transmitted

#20 @(posedge clk) tx\_Write = 0; //deassert tx\_Write

@(posedge clk) if(uut.write1Delay == 1)begin

#20; //wait for next clock cycle (when data is updated)

if (uut.sreg\_data != 12'h821) //8O1 tx -> 0x08 = 821

$display("Incorrect Data! Expected: 0x821 , Actual: 0x%h",

uut.sreg\_data[11:0]);

else

$display("Correct Data! Expected: 0x821 , Actual: 0x%h",

uut.sreg\_data[11:0]);

end

wait (uut.Done == 1); //wait until byte is done being transmitted.

wait (uut.TxRdy == 1); //wait until transmit engine is ready

//to receive another byte.

wait (uut.Done == 0);

//8O1 (115200 Baud) Transmitting 0x4F = 0100\_1111

bit8 = 1; //8 bits of data

parity\_en = 1; //parity enabled

odd\_n\_even = 1; //odd parity

tx\_Write = 1; //assert tx\_Write

baud\_val = 4'h8; //baud = 115200

in\_data = 8'h4F; //data to be transmitted

#20 @(posedge clk) tx\_Write = 0; //deassert tx\_Write

@(posedge clk) if(uut.write1Delay == 1)begin

#20; //wait for next clock cycle (when data is updated)

if (uut.sreg\_data != 12'h93d) //8O1 tx -> 0x4F = 93d

$display("Incorrect Data! Expected: 0x93d , Actual: 0x%h",

uut.sreg\_data[11:0]);

else

$display("Correct Data! Expected: 0x93d , Actual: 0x%h",

uut.sreg\_data[11:0]);

end

wait (uut.Done == 1); //wait until byte is done being transmitted.

wait (uut.TxRdy == 1); //wait until transmit engine is ready

//to receive another byte.

wait (uut.Done == 0);

//8O1 (230400 Baud) Transmitting 0x00 = 0000\_0000

bit8 = 1; //8 bits of data

parity\_en = 1; //parity enabled

odd\_n\_even = 1; //odd parity

tx\_Write = 1; //assert tx\_Write

baud\_val = 4'h9; //baud = 230400

in\_data = 8'h00; //data to be transmitted

#20 @(posedge clk) tx\_Write = 0; //deassert tx\_Write

@(posedge clk) if(uut.write1Delay == 1)begin

#20; //wait for next clock cycle (when data is updated)

if (uut.sreg\_data != 12'hc01) //8O1 tx -> 0x00 = c01

$display("Incorrect Data! Expected: 0xc01 , Actual: 0x%h",

uut.sreg\_data[11:0]);

else

$display("Correct Data! Expected: 0xc01 , Actual: 0x%h",

uut.sreg\_data[11:0]);

end

wait (uut.Done == 1); //wait until byte is done being transmitted.

wait (uut.TxRdy == 1); //wait until transmit engine is ready

//to receive another byte.

wait (uut.Done == 0);

//8O1 (460800 Baud) Transmitting 0x55 = 0101\_0101

bit8 = 1; //8 bits of data

parity\_en = 1; //parity enabled

odd\_n\_even = 1; //odd parity

tx\_Write = 1; //assert tx\_Write

baud\_val = 4'hA; //baud = 460800

in\_data = 8'h55; //data to be transmitted

#20 @(posedge clk) tx\_Write = 0; //deassert tx\_Write

@(posedge clk) if(uut.write1Delay == 1)begin

#20; //wait for next clock cycle (when data is updated)

if (uut.sreg\_data != 12'hd55) //8O1 tx -> 0x55 = d55

$display("Incorrect Data! Expected: 0xd55 , Actual: 0x%h",

uut.sreg\_data[11:0]);

else

$display("Correct Data! Expected: 0xd55 , Actual: 0x%h",

uut.sreg\_data[11:0]);

end

wait (uut.Done == 1); //wait until byte is done being transmitted.

wait (uut.TxRdy == 1); //wait until transmit engine is ready

//to receive another byte.

wait (uut.Done == 0);

//8O1 (921600 Baud) Transmitting 0x4F = 0010\_0000

bit8 = 1; //8 bits of data

parity\_en = 1; //parity enabled

odd\_n\_even = 1; //odd parity

tx\_Write = 1; //assert tx\_Write

baud\_val = 4'hB; //baud = 921600

in\_data = 8'h20; //data to be transmitted

#20 @(posedge clk) tx\_Write = 0; //deassert tx\_Write

@(posedge clk) if(uut.write1Delay == 1)begin

#20; //wait for next clock cycle (when data is updated)

if (uut.sreg\_data != 12'h881) //8O1 tx -> 0x20 = 881

$display("Incorrect Data! Expected: 0x881 , Actual: 0x%h",

uut.sreg\_data[11:0]);

else

$display("Correct Data! Expected: 0x881 , Actual: 0x%h",

uut.sreg\_data[11:0]);

end

wait (uut.Done == 1); //wait until byte is done being transmitted.

wait (uut.TxRdy == 1); //wait until transmit engine is ready

//to receive another byte.

wait (uut.Done == 0);

$stop;

end

endmodule

**Transmit Engine:**

`timescale 1ns / 1ps

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

//

// Author: Victor Espinoza

// Email: victor.alfonso94@gmail.com

// Project #: Project 3 - Transmit Engine plus PicoBlaze

// Course: CECS 460

//

// Create Date: 11:37:29 10/15/2015

//

// Module Name: transmit\_engine

// File Name: transmit\_engine.v

//

// Description: My transmit engine module contains all of the logic needed to

// communicate with a computer via universal asynchronous

// receiver/transmitter (UART) communication. In order to create

// a stable UART connection, I needed to make sure that I could

// run my program at different baud rates and different data

// formats (7N1, 7O1, 7E1, 8N1, 8O1, 8E1). For this program, I

// am shifting out the data in 12-bit data packets. I use a

// shift register to determine what data gets shifted out. I am

// able to load in the data that is going to be shifted out

// by delaying the write[1] strobe from the PicoBlaze by 1 clock

// cycle (this is because the data on the out\_port of the

// PicoBlaze is not available until the next clock cycle). If

// my write1Delay variable is set high, I then load my shift

// register with the out\_port data of the PicoBlaze. I then

// update the remaining values in the shift register to make

// sure that I have accounted for the right parity, stop, and

// start bits. This is done using the bit8, parity\_en, and

// odd\_n\_even inputs. Once all of the data in the shift register

// is accurate, I then start shifting out the bits 1 bit at a

// time. I fill the most significant bit of my shift register

// with a 1 (which is equivalent to filling up the register

// with a bunch of stop bits). I keep track of when I am done

// shifting out all of the bits in my shift register by using

// a bit counter. In order to know when a bit is done being

// shifted out, I also needed to use a baud-rate counter.

// I first decoded the baud rate by reading in the data on my

// four baud\_rate switches. The next step was to determine what

// baud-rate I was running at based on those four switches. After

// determining the baud rate, I then needed to determine the

// amount of ticks I would need to achieve that specified baud-rate.

// I did this by using the formula baudCountNum = (1/baud rate) /

// (1/50MHz). This gave me the terminal count number that I needed

// to achieve the desired baud rate. I then assigned my baudCountNum

// variable to this terminal count. Once that value was reached, I

// then knew that I had finished transmitting one byte via UART

// communication. Now the next step was to increment my Bit Counter.

// I repeated this process until my Bit Counter reached 11, meaning

// that I had finished transmitting all 12 bits of data in the

// data packet. For this project, we are always going to end up

// transmitting 12 bits of data per data packet. That is why my

// bit time counter limit is always 11 (0 - (12-1)). Once all 12

// bits of data have been shifted out, I then set my Done variable

// to 1, which also sets my TxRdy variable to a 1. I then output

// this TxRdy variable, along with the other status variables

// to the PicoBlaze. These status variables are concatenated and

// represent the status register of the Transmit Engine. The

// TxRdy bit lets the PicoBlaze know that the Transmit Engine is

// ready to transmit another byte, at which point the PicoBlaze

// will output the next character being transmitted in the

// sequence to the Transmit Engine.

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

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

module transmit\_engine(clk, rstb, bit8, parity\_en, odd\_n\_even, baud\_val, txWrite,

inData, tx, pico\_data);

//Input and Output Declarations

input clk, rstb, bit8, parity\_en, odd\_n\_even, txWrite;

input [3:0] baud\_val;

input [7:0] inData;

output tx;

output [7:0] pico\_data;

//output transferLimit;

//Local Declarations

wire TxSet, TxRst, Done, sSet, sRst, ld, btu, shift;

reg [7:0] data;

//reg [6:0] transferLimitCounter;

reg [17:0] baudCountNum, baudCount, nBaudCount;

reg [3:0] bitCount, nBitCount;

reg TxRdy, start, write1Delay;

reg [11:0] sreg\_data;

reg b7, b8;

assign TxSet = Done;

assign TxRst = txWrite;

//R-S Flop that controlls the TxRdy input

//for the UART

always@(posedge clk, negedge rstb)

if(!rstb)

TxRdy <= 1'b1;

else if(TxSet)

TxRdy <= 1'b1;

else if(TxRst)

TxRdy <= 1'b0;

else

TxRdy <= TxRdy;

assign pico\_data[7:0] = {6'b0, TxRdy, 1'b0};

assign sSet = txWrite;

assign sRst = Done;

//R-S Flop that controlls the start input

//for the UART

always@(posedge clk, negedge rstb)

if(!rstb)

start <= 1'b0;

else if(sSet)

start <= 1'b1;

else if(sRst)

start <= 1'b0;

else

start <= start;

assign ld = txWrite;

//register that holds the outport data

always@(posedge clk, negedge rstb)

if(!rstb)

data <= 8'b00;

else if (ld)

data <= inData;

else

data <= data;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//Shift register logic

//register that delays the write strobe

//for one clock cycle

always@(posedge clk, negedge rstb)

if(!rstb)

write1Delay <= 1'b0;

else

write1Delay <= txWrite;

//logic used to determine b7 and b8 of the shift register (value changes

//depending on what bit8, parity\_en, and odd\_n\_even is).

always @(\*)

case ({bit8,parity\_en,odd\_n\_even})

3'b000: {b8,b7} = 2'b11; //7N1

3'b001: {b8,b7} = 2'b11; //7N1

3'b010: {b8,b7} = {1'b1, ^data[6:0]}; //7E1

3'b011: {b8,b7} = {1'b1, ~(^data[6:0])}; //7O1

3'b100: {b8,b7} = {1'b1, data[7]}; //8N1

3'b101: {b8,b7} = {1'b1, data[7]}; //8N1

3'b110: {b8,b7} = {(^data[7:0]), data[7]}; //8E1

3'b111: {b8,b7} = {~(^data[7:0]), data[7]}; //8O1

default: {b8,b7} = 2'b00; //no bits

endcase

//update shift register value uf write1Delay is set high

always @( posedge clk, negedge rstb)

if (!rstb)

sreg\_data <=12'hFFF; //load with all 1's

else if (write1Delay)

sreg\_data <= {1'b1,b8,b7,data[6:0], 1'b0, 1'b1};//load data into sreg

else if (shift)begin

sreg\_data <= sreg\_data >> 1; //shift data right by 1

sreg\_data [11] <= 1; //fill msb of shift register with a 1

end

assign tx = sreg\_data[0];

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//BAUD TIME COUNTER LOGIC:

//baud rate count decoder:

//values are derived using (1/baud rate) / (1/50MHz)

always@(\*)

case(baud\_val)

4'b0000: baudCountNum = 166667 -1; //300 BAUD Rate

4'b0001: baudCountNum = 41667 - 1; //1200 BAUD Rate

4'b0010: baudCountNum = 20833 - 1; //2400 BAUD Rate

4'b0011: baudCountNum = 10417 - 1; //4800 BAUD Rate

4'b0100: baudCountNum = 5208 - 1; //9600 BAUD Rate

4'b0101: baudCountNum = 2604 - 1; //19200 BAUD Rate

4'b0110: baudCountNum = 1302 - 1; //38400 BAUD Rate

4'b0111: baudCountNum = 868 - 1; //57600 BAUD Rate

4'b1000: baudCountNum = 434 - 1; //115200 BAUD Rate

4'b1001: baudCountNum = 217 - 1; //230400 BAUD Rate

4'b1010: baudCountNum = 109 - 1; //460800 BAUD Rate

4'b1011: baudCountNum = 54 - 1; //921600 BAUD Rate

default: baudCountNum = 166667 - 1; //300 BAUD Rate

endcase

//assign btu output tick and shift wire

assign btu = (baudCount == baudCountNum) ? 1'b1 : 1'b0;

assign shift = btu;

//Determine next state of baudCount

always@(\*)

case({btu, start})

2'b00 : nBaudCount = 18'b00;

2'b01 : nBaudCount = baudCount + 1;

2'b10 : nBaudCount = 18'b00;

2'b11 : nBaudCount = 18'b00;

default : nBaudCount = 18'b00;

endcase

//update baudCount accordingly

always @(posedge clk, negedge rstb)

if(!rstb) //check reset bit

baudCount <= 18'b0; //reset counter

else

baudCount <= nBaudCount; //update baudCount

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//BIT COUNTER LOGIC:

//For this lab we are always going to be transmitting 12 bits of data at a

//time. Once our counter has established that all 12 bits have been

//transmitted, we then set the Done variable high.

//assign Done output tick

assign Done = (bitCount == 11) ? 1'b1 : 1'b0;

//Determine next state of bitCount

always@(\*)

case({btu, start})

2'b00 : nBitCount = 4'b00;

2'b01 : nBitCount = bitCount;

2'b10 : nBitCount = 4'b00;

2'b11 : nBitCount = bitCount + 1;

default : nBitCount = 4'b00;

endcase

//update bitCount accordingly

always @(posedge clk, negedge rstb)

if(!rstb) //check reset bit

bitCount <= 4'b0; //reset counter

else

bitCount <= nBitCount; //update bitCount

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

endmodule