DE0 Controller Glue Logic Guide

Timothy Ajmani

12/13/2014

I. Files

I. top_level.vhd

This file contains a structural architecture of the top level file for the entire project. All entities and the required signals are instantiated. More registers will need to be instantiated if more than two input vectors are required for the circuit. The top level has one generic parameter that sets the width of the data used in the entire circuit.

II. serial_to_parallel_reg.vhd

This is a serial-to-parallel shift register. This takes serial data that comes out of the virtual jTAG entity and converts it to a parallel vector of bits that is used in the rest of the circuit. The data is clocked in if the ir_in and v_sdr bits are high and the parallel vector is shifted out if the udr signal is set to 1, indicating all bits have been shifted in.

III. reg_gen.vhd

This is a generic register of an arbitrary length that clocks in data when the enable signal is set high.

IV. decoder7seg.vhd

This is a decoder used to send data to the LEDs on the DE0 board. This was developed in the EEL4712 Digital Design course. The hexadecimal LEDs show the inputs to the ALU. The row of single bit LEDs shows the output of the ALU.

V. alu_ns.vhd

This is a basic ALU application used for the primary testing of our design. Data for the 2 input vectors comes from the data sent through the GUI on the PC to the jTAG and finally to the storage registers. The selects for the different ALU operations can be changed through using the switches on the DE0 board.

VI. addr_wrapper.vhd

This is the top level file for the address generation and glue logic. This includes files for the glue logic, address register, and address counter. The address register sends the address to the glue logic which determines if the data is an instruction or data, and sends the respective data to the input registers and turns on/off the enables. The counter determines when the each piece of data sent from the vJTAG is finished being transferred and when to start the next transfer.

VII. sdr_count.vhd

The counter determines when to store the next address/data. When the count reaches two, it is reset, and an enable is sent to the address register to clock in the address/data.

VIII. vJTAG.qip (vjtag.vhd)

This is the virtual jTAG component being used in the design. This handles the data being transferred from the PC and sends out serial data to be converted into parallel using the SIPO entity.

IX. d_logic.vhd

This determines where to place data being transferred from the vJTAG. It determines if the current data is an address or data to be placed into a register. Depending on the application, this will need to be edited if more registers and enables are needed.

X. tdo_shifter.vhd

This is a huge state machine that handles transferring data back to the GUI on the PC end. A valid bit determines if the data being transferred back to the GUI is an input to the circuit or an output to the circuit. The state machine is hard coded to the number of bits because of testing issues that included data being delayed or sent too early. This was the only way determined to have the exact timing that was needed.

XI. mem_pkg.vhd

This library includes constants for addresses and bit widths used in the entire application. If more registers are needed for use in the circuit, their addresses can be added in this file.

II. Operation

Basically the general idea follows these steps:

1. Data taken in from the PC and sent to the vJTAG
2. jTAG shifts data to the serial to parallel shift register and is converted to a parallel vector to be used in the circuit
3. This vector is sent to an address conversion entity (addr_wrapper.vhd) and the inputs to the input registers used in the circuit.
4. The address conversion entity determines which enables are to be set to the input/output registers.
5. The output register sends the output vector of the circuit to the Parallel-In-Serial-Out entity (tdo.vhd) which determines if the data being sent back to the GUI is either an input or output, and then shifts the corresponding vector serially.

III. Comments

This was pretty difficult to attempt to do. There isn't much documentation on the internet regarding the vJTAG entity and much of it was done by trial and error. The hardest part of the VHDL for this project was getting the data to be sent back to the GUI. Another complicated issue was working with arbitrary bit lengths. TCL can send 32 bits at a time at maximum. The majority of the time working on the code was spent working with 8 bits. The return data was not correctly being read in when the bit lengths were changed. Kyle Silvares discovered that zeroes

needed to be sent in for a certain amount of time while the GUI is making a connection to the circuit.

The next steps for working on the project are to try different circuits, like applications using controllers and datapaths, and further optimizing the VHDL to remove the hard coding needed for the state machine that handles transferring data serially back to the PC.