

**\*\*You:\*\***

please read these files first. user\_project\_wrapper.v: // SPDX-FileCopyrightText: 2020 Efabless Corporation // // Licensed under the Apache License, Version 2.0 (the "License"); // you may not use this file except in compliance with the License. // You may obtain a copy of the License at // // http://www.apache.org/licenses/LICENSE-2.0 // // Unless required by applicable law or agreed to in writing, software // distributed under the License is distributed on an "AS IS" BASIS, // WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. // See the License for the specific language governing permissions and // limitations under the License. // SPDX-License-Identifier: Apache-2.0`default\_nettype none /\* \*-----\* \* user\_project\_wrapper \* \* This wrapper enumerates all of the pins available to the \* user for the user project. \* \* An example user project is provided in this wrapper. The \* example should be removed and replaced with the actual \* user project. \* \*-----\*/ module user\_project\_wrapper #( parameter BITS = 32 ) ( `ifdef USE\_POWER\_PINS inout vdda1, // User area 1 3.3V supply inout vdda2, // User area 2 3.3V supply inout vssa1, // User area 1 analog ground inout vssa2, // User area 2 analog ground inout vccd1, // User area 1 1.8V supply inout vccd2, // User area 2 1.8v supply inout vssd1, // User area 1 digital ground inout vssd2, // User area 2 digital ground `endif // Wishbone Slave ports (WB MI A) input wb\_clk\_i, input wb\_rst\_i, input wbs\_stb\_i, input wbs\_cyc\_i, input wbs\_we\_i, input [3:0] wbs\_sel\_i, input [31:0] wbs\_dat\_i, input [31:0] wbs\_adr\_i, output wbs\_ack\_o, output [31:0] wbs\_dat\_o, // Logic Analyzer Signals input [127:0] la\_data\_in, output [127:0] la\_data\_out, input [127:0] la\_oenb, // IOs input [MPRJ\_IO\_PADS-1:0] io\_in, output [MPRJ\_IO\_PADS-1:0] io\_out, output [MPRJ\_IO\_PADS-1:0] io\_oeb, // Analog (direct connection to GPIO pad---use with caution) // Note that analog I/O is not available on the 7 lowest-numbered // GPIO pads, and so the analog\_io indexing is offset from the // GPIO indexing by 7 (also upper 2 GPIOs do not have analog\_io). inout [MPRJ\_IO\_PADS-10:0] analog\_io, // Independent clock (on independent integer divider) input user\_clock2, // User maskable interrupt signals output [2:0] user\_irq ); /\*-----\* \* User project is instantiated here \* \*-----\*/ user\_proj\_example mprj ( `ifdef USE\_POWER\_PINS .vccd1(vccd1), // User area 1 1.8V power .vssd1(vssd1), // User area 1 digital ground `endif .wb\_clk\_i(wb\_clk\_i), .wb\_rst\_i(wb\_rst\_i), // MGMT SoC Wishbone Slave .wbs\_cyc\_i(wbs\_cyc\_i), .wbs\_stb\_i(wbs\_stb\_i), .wbs\_we\_i(wbs\_we\_i), .wbs\_sel\_i(wbs\_sel\_i), .wbs\_adr\_i(wbs\_adr\_i), .wbs\_dat\_i(wbs\_dat\_i), .wbs\_ack\_o(wbs\_ack\_o), .wbs\_dat\_o(wbs\_dat\_o), // Logic Analyzer .la\_data\_in(la\_data\_in), .la\_data\_out(la\_data\_out), .la\_oenb(la\_oenb), // IO Pads .io\_in({io\_in[37:30],io\_in[7:0]}), .io\_out({io\_out[37:30],io\_out[7:0]}), .io\_oeb({io\_oeb[37:30],io\_oeb[7:0]}), // IRQ .irq(user\_irq ); endmodule // user\_project\_wrapper`default\_nettype wire The other file is. user\_proj\_example.v: // SPDX-FileCopyrightText: 2020 Efabless Corporation // // Licensed under the Apache License, Version 2.0 (the "License"); // you may not use this file except in compliance with the License. // You may obtain a copy of the License at // // http://www.apache.org/licenses/LICENSE-2.0 // // Unless required by applicable law or agreed to in writing, software // distributed under the License is distributed on an "AS IS" BASIS, // WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. // See the License for the specific language governing permissions and // limitations under the License. // SPDX-License-Identifier: Apache-2.0`default\_nettype none /\* \*-----\* \* user\_proj\_example \* \* This is an example of a (trivially simple) user project, \* showing how the user project can connect to the logic \* analyzer, the wishbone bus, and the I/O pads. \* \* This project generates an integer count, which is output \* on the user area GPIO pads (digital output only). The \* wishbone connection allows the project to be controlled \* (start and stop) from the management SoC program. \* \* See the testbenches in directory "mprj\_counter" for the \* example programs that drive this user project. The three \* testbenches are "io\_ports", "la\_test1", and "la\_test2". \* \*-----\* \* / module user\_proj\_example #( parameter BITS = 32 ) ( `ifdef USE\_POWER\_PINS inout vccd1, // User area 1 1.8V supply inout vssd1, // User area 1 digital ground `endif // Wishbone Slave ports (WB MI A) input wb\_clk\_i, input wb\_rst\_i, input wbs\_stb\_i, input wbs\_cyc\_i, input wbs\_we\_i, input [3:0] wbs\_sel\_i, input [31:0] wbs\_dat\_i, input [31:0] wbs\_adr\_i, output wbs\_ack\_o, output [31:0] wbs\_dat\_o, // Logic Analyzer Signals input [127:0] la\_data\_in, output [127:0] la\_data\_out, input [127:0] la\_oenb, // IOs input [15:0] io\_in, output [15:0] io\_out, output [15:0] io\_oeb, // IRQ output [2:0] irq ); wire clk; wire rst; wire [15:0] io\_in; wire [15:0] io\_out; wire [15:0] io\_oeb; wire [15:0] rdata; wire [15:0] wdata; wire [15:0] count; wire valid; wire [3:0] wstrb; wire [31:0] la\_write; // WB MI A assign valid = wbs\_cyc\_i & wbs\_stb\_i; assign wstrb = wbs\_sel\_i & {4{wbs\_we\_i}}; assign wbs\_dat\_o = rdata; assign wdata = wbs\_dat\_i[15:0]; // IO assign io\_out = count; assign io\_oeb = {(15){rst}}; // IRQ assign irq = 3'b000; // Unused // LA assign la\_data\_out = {{{(127-BITS){1'b0}}, count}; // Assuming LA probes [63:32] are for controlling the count register assign la\_write = ~la\_oenb[63:32] & ~{BITS{valid}}; // Assuming LA probes [65:64] are for controlling the count clk & reset assign clk = (~la\_oenb[64]) ? la\_data\_in[64]: wb\_clk\_i; assign rst = (~la\_oenb[65]) ? la\_data\_in[65]: wb\_rst\_i; counter #( .BITS(BITS) ) counter( .clk(clk), .reset(rst), .ready(wbs\_ack\_o), .valid(valid), .rdata(rdata), .wdata(wbs\_dat\_i), .wstrb(wstrb), .la\_write(la\_write), .la\_input(la\_data\_in[63:32]), .count(count) ); endmodule module counter #( parameter BITS = 32 ) ( input clk, input reset, input valid, input [3:0] wstrb, input [15:0] wdata, input [BITS-1:0] la\_write, input [BITS-1:0] la\_input, output ready, output [15:0] rdata, output [15:0] count ); reg ready; reg [15:0] count; reg [15:0] rdata; always @(posedge clk) begin if (reset) begin count <= 0; ready <= 0; end else begin ready <= 1'b0; if (~la\_write) begin count <= count + 1; end if (valid && !ready) begin ready <= 1'b1; rdata <= count; if (wstrb[0]) count[7:0] <= wdata[7:0]; if (wstrb[1]) count[15:8] <= wdata[15:8]; end else if ((la\_write) begin count <= la\_write & la\_input; end

```
end endmodule `default_nettype wire
```

```
-----  
**ChatGPT:**
```

Sure, here is a brief summary of the two Verilog files you've provided:

1. `user_project_wrapper.v`: This is a Verilog wrapper module for a user project, as the name suggests. This module essentially enumerates all the available pins for the user project and provides an example of a user project. Inside this wrapper, there's an instantiation of the module `user_proj_example`. This wrapper is compatible with different power supply setups (`ifdef USE_POWER_PINS`) and includes various interface options like a Wishbone slave interface, a logic analyzer interface, and I/O pins.
2. `user_proj_example.v`: This is a Verilog file that represents a user project. The file includes an example project which generates an integer count, output on the GPIO pads. The Wishbone connection allows the project to be controlled (start and stop) from the management SoC program. The module also contains a nested module `counter`, which is a simple up counter that can be controlled via a wishbone interface or directly through logic analyzer probes.

Here are some additional specific details:

- The `counter` module: This is a simple counter with a reset input. If the reset is not asserted, the counter will increment every clock cycle. The counter's value can be written via wishbone or logic analyzer interface.
- Logic Analyzer (LA) interface: The LA interface allows for observation and control of internal signals. For example, in `user_proj_example.v`, the `la_data_out` is connected to `count`, which means the counter's value can be read out using the LA. Also, the counter can be controlled by writing specific values to it using the LA (if `la_write` and `la_input` signals are used).
- Wishbone interface: This is a standard bus interface used for connecting different modules in a system. In this context, it's used for controlling and monitoring the `counter` module. The `counter`'s value can be read out, and it can also be written with a new value via this interface.
- I/O interface: This interface is connected to the `io_in`, `io_out`, and `io_oeb` signals. It appears to connect directly to the physical I/O pads. In this example, the `counter`'s value is outputted on the I/O pads (`io_out`).

Remember that these are example projects, and you are expected to modify them to suit your specific needs.

```
-----  
**You:**
```

If you understand, please connect the following core to `user_project_wrapper.v` as a file to replace the example module core #( parameter RESET\_VECTOR = 32'h8000\_0000 ) ( input clk, input reset, input meip, // Memory interface output ext\_valid, output ext\_instruction, input ext\_ready, output [31:0] ext\_address, output [31:0] ext\_write\_data, output [3:0] ext\_write\_strobe, input [31:0] ext\_read\_data ); Give me the modified `user_project_wrapper.v`, please note that the input and output definitions of `user_project_wrapper.v` cannot be changed In the case, these are additionally used in the example, and in this I wonder if I should also process these data/WB MI A in some way assign `valid = wbs_cyc_i && wbs_stb_i`; assign `wstrb = wbs_sel_i & {4{wbs_we_i}}`; assign `wbs_dat_o = rdata`; assign `wdata = wbs_dat_i[15:0]`; //IO assign `io_out = count`; assign `io_oeb = {(15){rst}}`; // IRQ assign `irq = 3'b000`; // Unused // LA assign `la_data_out = {{{(127-BITS){1'b0}}, count}}`; // Assuming LA probes [63:32] are for controlling the count register assign `la_write = ~la_oenb[63:32] & ~{BITS{valid}}`; // Assuming LA probes [65:64] are for controlling the count clk & reset assign `clk = (~la_oenb[64]) ? la_data_in[64]: wb_clk_i`; assign `rst = (~la_oenb[65]) ? la_data_in[65]: wb_rst_i`;

```
-----  
**ChatGPT:**
```

Here's how `user_project_wrapper.v` could be modified to instantiate the `core` module instead of the `user_proj_example` module. Please note that connections were made under the assumption that `core` is a CPU-like module with a memory interface.

```
verilog  Copy code
```

```

`default_nettype none
/*
*-----
*
* user_project_wrapper
*
* This wrapper enumerates all of the pins available to the
* user for the user project.
*
* An example user project is provided in this wrapper. The
* example should be removed and replaced with the actual
* user project.
*-----
*/

module user_project_wrapper #(
    parameter BITS = 32,
    parameter RESET_VECTOR = 32'h8000_0000
) (
    `ifdef USE_POWER_PINS
        inout vdda1,    // User area 1 3.3V supply
        inout vdda2,    // User area 2 3.3V supply
        inout vssa1,    // User area 1 analog ground
        inout vssa2,    // User area 2 analog ground
        inout vccd1,    // User area 1 1.8V supply
        inout vccd2,    // User area 2 1.8v supply
        inout vssd1,    // User area 1 digital ground
        inout vssd2,    // User area 2 digital ground
    `endif

    // Wishbone Slave ports (WB MI A)
    input  wb_clk_i,
    input  wb_rst_i,
    input  wbs_stb_i,
    input  wbs_cyc_i,
    input  wbs_we_i,
    input  [3:0] wbs_sel_i,
    input  [31:0] wbs_dat_i,
    input  [31:0] wbs_adr_i,
    output wbs_ack_o,
    output [31:0] wbs_dat_o,

    // Logic Analyzer Signals
    input  [127:0] la_data_in,
    output [127:0] la_data_out,
    input  [127:0] la_oenb,

    // IOs
    input  [`MPRJ_IO_PADS-1:0] io_in,
    output [`MPRJ_IO_PADS-1:0] io_out,
    output [`MPRJ_IO_PADS-1:0] io_oeb,

    // Analog (direct connection to GPIO pad---use with caution)
    // Note that analog I/O is not available on the 7 lowest-numbered
    // GPIO pads, and so the analog_io indexing is offset from the
    // GPIO indexing by 7 (also upper 2 GPIOs do not have analog_io).
    inout [`MPRJ_IO_PADS-10:0] analog_io,

    // Independent clock (on independent integer divider)
    input  user_clock2,

    // User maskable interrupt signals
    output [2:0] user_irq
);

/*-----*/
/* User project is instantiated here */
/*-----*/

core my_core (
    `ifdef USE_POWER_PINS
        .vccd1(vccd1), // User area 1 1.8V power

```

```

    .vssd1(vssd1), // User area 1 digital ground
`endif

    .clk(wb_clk_i),
    .reset(wb_rst_i),
    .meip(user_irq[0]), // Assuming user_irq[0] is MEIP (Machine External Interrupt Pending)

    // Memory interface (connected to Wishbone)
    .ext_valid(wbs_cyc_i & wbs_stb_i), // External Valid signal asserted when Wishbone cycle and strobe are high
    .ext_instruction(wbs_we_i), // Assuming ext_instruction is equivalent to Wishbone write-enable
    .ext_ready(wbs_ack_o), // Assuming ext_ready is equivalent to Wishbone acknowledge
    .ext_address(wbs_adr_i),
    .ext_write_data(wbs_dat_i),
    .ext_write_strobe(wbs_sel_i),
    .ext_read_data(wbs_dat_o)
);

/*-----*/
/* Unused signals are tied off here */
/*-----*/

// I/Os
assign io_out = {`MPRJ_IO_PADS{1'b0}}; // Unused I/Os are tied off to ground
assign io_oeb = {`MPRJ_IO_PADS{1'b1}}; // Unused I/Os are tied off to ground

// IRQ
assign user_irq = 3'b000; // Unused

// LA
assign la_data_out = 128'h0; // Unused

endmodule

```

This modification removes all signals related to the counter and logic analyzer, and instead connects the `core` module to the Wishbone interface. It also assumes that the `core` is a CPU-like module with a memory interface. The unused signals are tied off to the ground to maintain stability.

Please note, this modification is based on assumptions about your design requirements. You may need to revise or modify it to fit the actual `core` module's interface and requirements.

-----