HW#1 Simulation and ILA Probing of Aquila Execution

Chun-Jen Tsai
National Chiao Tung University
10/06/2020

Homework Goal

- In this homework, you will learn how to:
  1. Set up full-system cycle-accurate simulation of Aquila
  2. Use Xilinx Integrated Logic Analyzer for real-time debugging
  3. Trace instruction execution at circuit level
- You must also modify the Dhrystone program and see how you can improve its performance
  - As a first attempt, optimize `strcpy()` and `strcmp()` first
  - Your modification has to produce an equivalent C program for all functions
- You must upload a report to E3 by 10/19, 17:00.

Analyze the Execution of Aquila SoC

- To analyze the behavior of Aquila, you can use a RTL simulator or the Integrated Logic Analyzer (ILA)

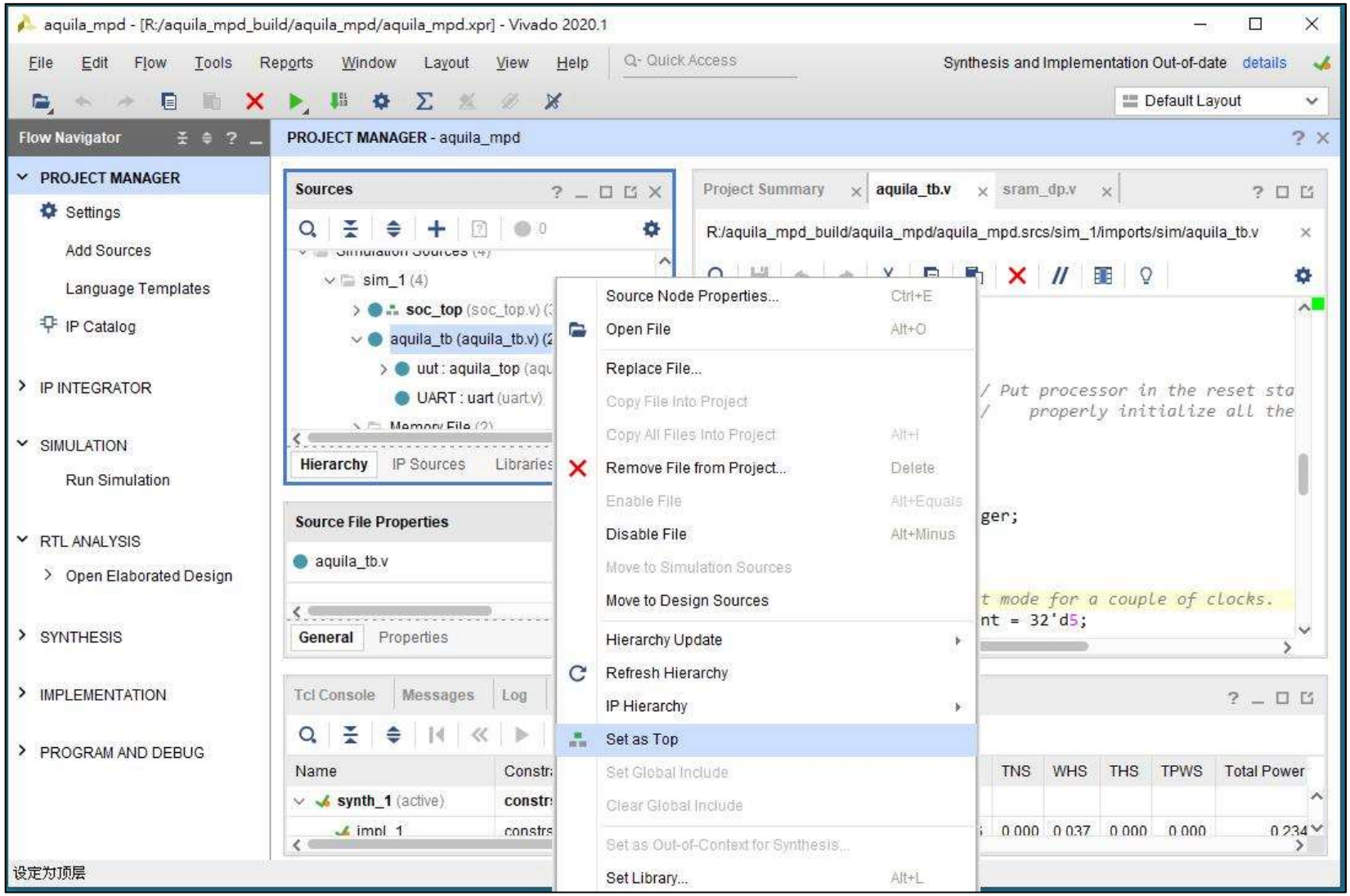
- Full-system cycle-accurate RTL simulation:
  - Simulation of UART device is very slow → no UART I/O !
  - ROM must be modified to contain the program to be analyzed

- Real-time ILA circuit probing:
  - Embed signal probes into your circuit
  - Set a trigger condition to capture signal traces to on-chip RAM
  - Perform a post-mortem analysis on a PC afterwards

Behavior Simulation Using Vivado Sim

- To run behavioral simulation of Aquila, you must replace the top-level, `soc_top.v`, by a testbench module, `aquila_tb.v`, that provides
  - Simulated clock signal
  - Simulated reset signal
  - Simulation models for any I/O devices
- Set `aquila_tb` as the top module for simulation
- Please check `aquila_tb.v` under "Simulation Sources" to see the simulated clock and reset
  - Change the clock frequency to 50MHz

Set `aquila_tb` as the Top



ROM Modification

- Since we have no UART transceiver model (a real FTDI IC is used in the Aquila SoC), we cannot have any UART I/O operations in the system
  - You must compile the Dhrystone program as a ROM image and use it to replace the `uartboot` ROM image
  - You must disable any `printf()` in the Dhrystone program (we have a #define for this purpose).

- For DMIPS calculation, modify the program so that:
  - Only integer computation can be used†
  - The DMIPS/MHz value (scaled to an integer) is stored in a register so that the simulator can show its value

† That is, you can use base-10 fixed-point arithmetic to compute DMIPS/MHz.

Run the Simulation

aquila_mpd - [R:/aquila_mpd_build/aquila_mpd/aquila_mpd.xpr] - Vivado 2020.1

File Edit Flow Tools Reports Window Layout View Help Quick Access Synthesis and Implementation Out-of-date details

Default Layout

Flow Navigator

PROJECT MANAGER - aquila_mpd

PROJECT MANAGER
- Settings
- Add Sources
- Language Templates
- IP Catalog

IP INTEGRATOR

SIMULATION
- Run Simulation

Click!

- Run Behavioral Simulation
- Run Post-Synthesis Functional Simulation
- Run Post-Synthesis Timing Simulation
- Run Post-Implementation Functional Simulation
- Run Post-Implementation Timing Simulation

RTL ANALYSIS
- Open Elab

SYNTHESIS

IMPLEMENTATION

PROGRAM AND DEBUG

Sources

- Memory : memory (memory.v)
- Writeback : writeback (writeback
- CSR : csr_file (csr_file.v)
- TCM : sram_dp (sram_dp.v)
- CLINT : clint (clint.v)
- ATOM_U : atomic_unit (atomic_unit

Hierarchy IP Sources Libraries Com

Project Summary | aquila_tb.v | sram_dp.v

R:/aquila_mpd_build/aquila_mpd/aquila_mpd.srcs/sim_1/imports/sim/aquila_tb.v

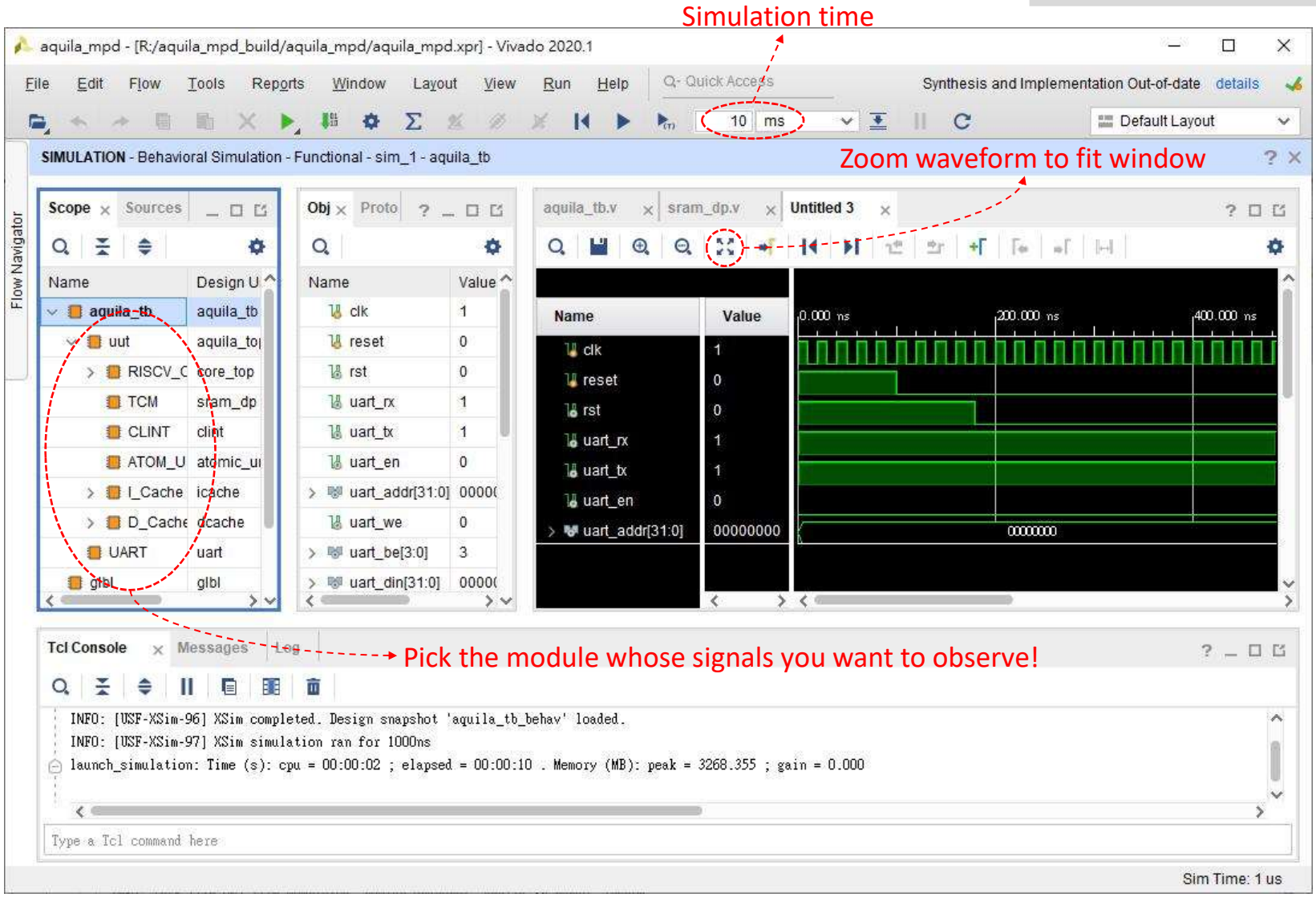
```
73  event reset_trigger;
74  event reset_done_trigger;
75
76  initial begin
77    forever begin
78      @ (reset_trigger);
79      reset = 1;
80      @ (posedge clk); // Put processor in the reset sta
81      @ (posedge clk); //    properly initialize all the
82      @ (posedge clk);
83      @ (posedge clk);
84      @ (posedge clk);
85      reset = 0;
86      -> reset_done_trigger;
87    end
88  end
```

Tcl Console | Messages | Log | Reports | Design Runs

Name	Constraints	Status	Incremental	WNS	TNS	WHS	THS	TPWS	Total Power
synth_1 (active)	constrs_1	Synthesis Out-of-date	Off						
impl_1	constrs_1	Implementation Out-of-date	Off	3.376	0.000	0.037	0.000	0.000	0.234

Vivado Simulator

Vivado Simulator Window



Tracing the Assembly Code

- After you make the ROM image, there should be a `*.objdump` file that contains the assembly code of the compiled program:

Aquila execution begins here!

```
dhry.out:     file format elf32-littleriscv

Disassembly of section .text:

00000000 <boot>:
       0:        ff010113    addi sp,sp,-16
       4:        00112623    sw   ra,12(sp)
       8:        000072b7    lui  t0,0x7
       c:        9d02a103    lw   sp,-1584(t0) # 69d0 <stack_top>
      10:        225030ef    jal  ra,3a34 <main>
      14:        00c12083    lw   ra,12(sp)
      18:        00000513    li   a0,0
      1c:        01010113    addi sp,sp,16
      20:        2b90006f    j    ad8 <exit>

00000024 <Proc_2>:
      24:        000097b7    lui  a5,0x9
      28:        1347c703    lbu  a4,308(a5) # 9134 <Ch_1_Glob>

                        . . . .
```

Showing Register Values

- Note that the registers of Aquila is declared in the file `reg_file.v`.
  - You can add the CPU registers to the signal window in the simulator to show them:

```
reg [XLEN-1 : 0] rf [0 : NUM_REGS-1];
```

  - It is probably easier for you to remap these registers to the ABI names to match the register names in the `*.objdump` file, for example:

```
wire [XLEN-1 : 0] ra = rf[1];
wire [XLEN-1 : 0] sp = rf[2];
          . . . .
```

Tracing the Execution of a Function

- For Dhrystone, you may want to rewrite one of the function, such as `strcpy()`, to speed up the execution
- Again, the `*.objdump` tells you the start address of the function
  - `strcpy()` begins at `0x00000e7c` in my build
- Tracing a program using a simulator is much more tedious than using `GDB`, but it gives you more details

Storing a Debug Value in Registers

- You can use inline assembly to store a local variable (or a global one) into a register so that the simulator can display its value:
  - Similar to debugging a program using `printf()`!

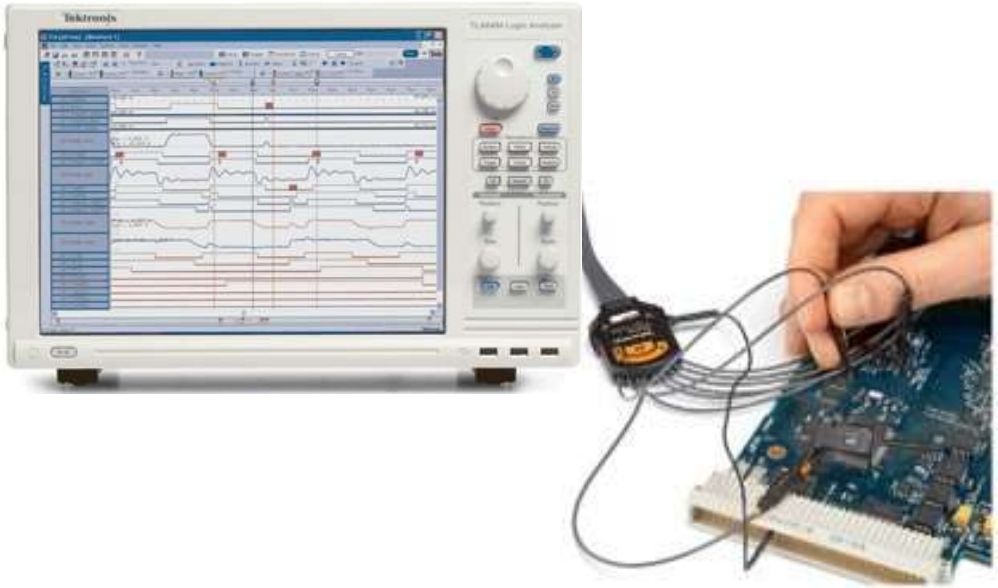
```
unsigned int test_asm()
{
    unsigned int rvalue;

    asm volatile ("lui t1, 0xABCDF");
    asm volatile ("addi t1, t1, -2013");
    asm volatile ("addi %0, t1 ,0" : "=r"(rvalue));

                                . . . .
    return rvalue;
}
```

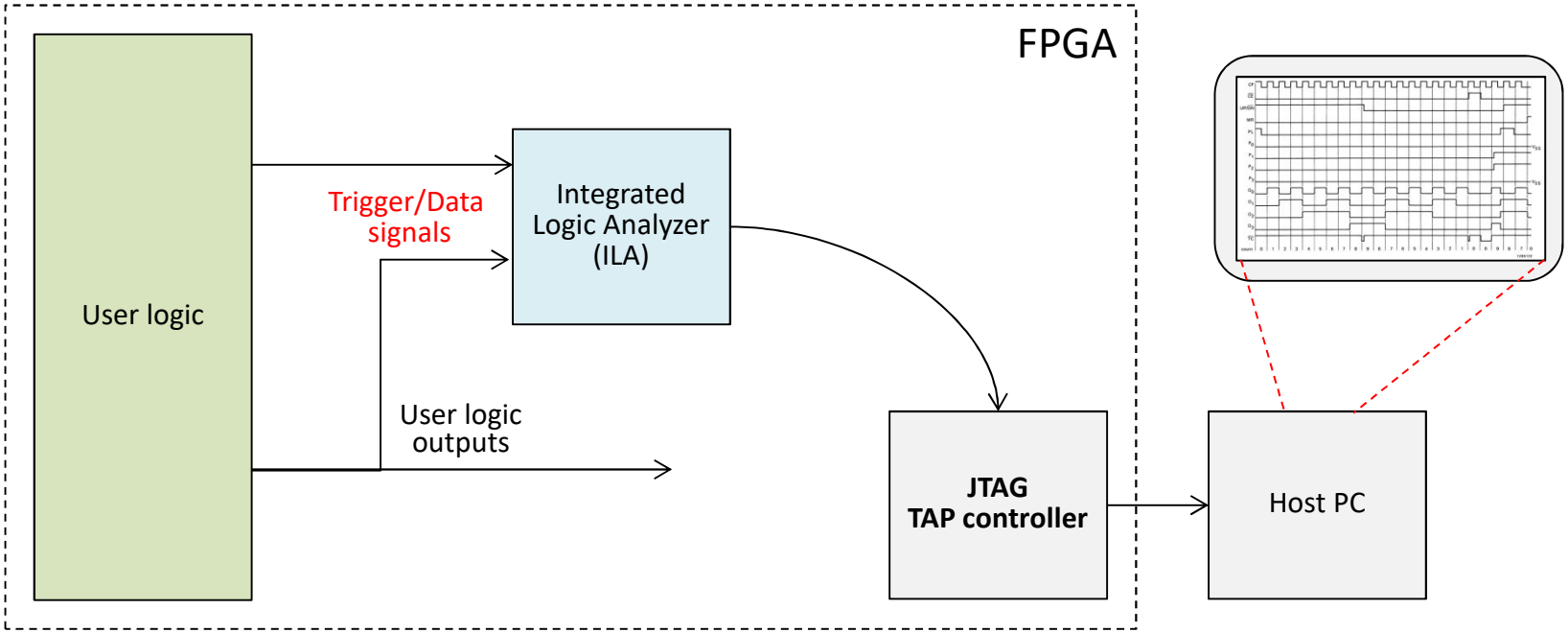
Real-Time Probing Using Vivado

- Full-system simulations for complex logic and software behaviors would take too much time; and real devices are difficult to simulate
- In the good old days, for real-time debugging of a digital circuit, we use a logic analyzer for the job



Vivado Integrated Logic Analyzer

- Vivado Integrated Logic Analyzer (ILA) is an IP that can be integrated into the hardware platform so that some signals in the user IP's can be intercepted and saved in a trace file for analysis



Debug Your Circuit in Real-Time

- To debug your logic in real-time, you must “mark” the signals for debugging with one of the three methods:
  - Using the “synthesis attribute” syntax in Verilog-2001
  - Using the Vivado GUI IDE
  - Using the TCL command console (we don’t use TCL here)
- After marking the signals, you must set up the debug wizard before you use the Hardware Manager to capture the signals at runtime
- Note: do not mark clock signals. The waveform viewer has tick markers.

Mark Debug Signals Using Verilog

- In Verilog-2001, you can set the synthesis attributes of a signal, for example:

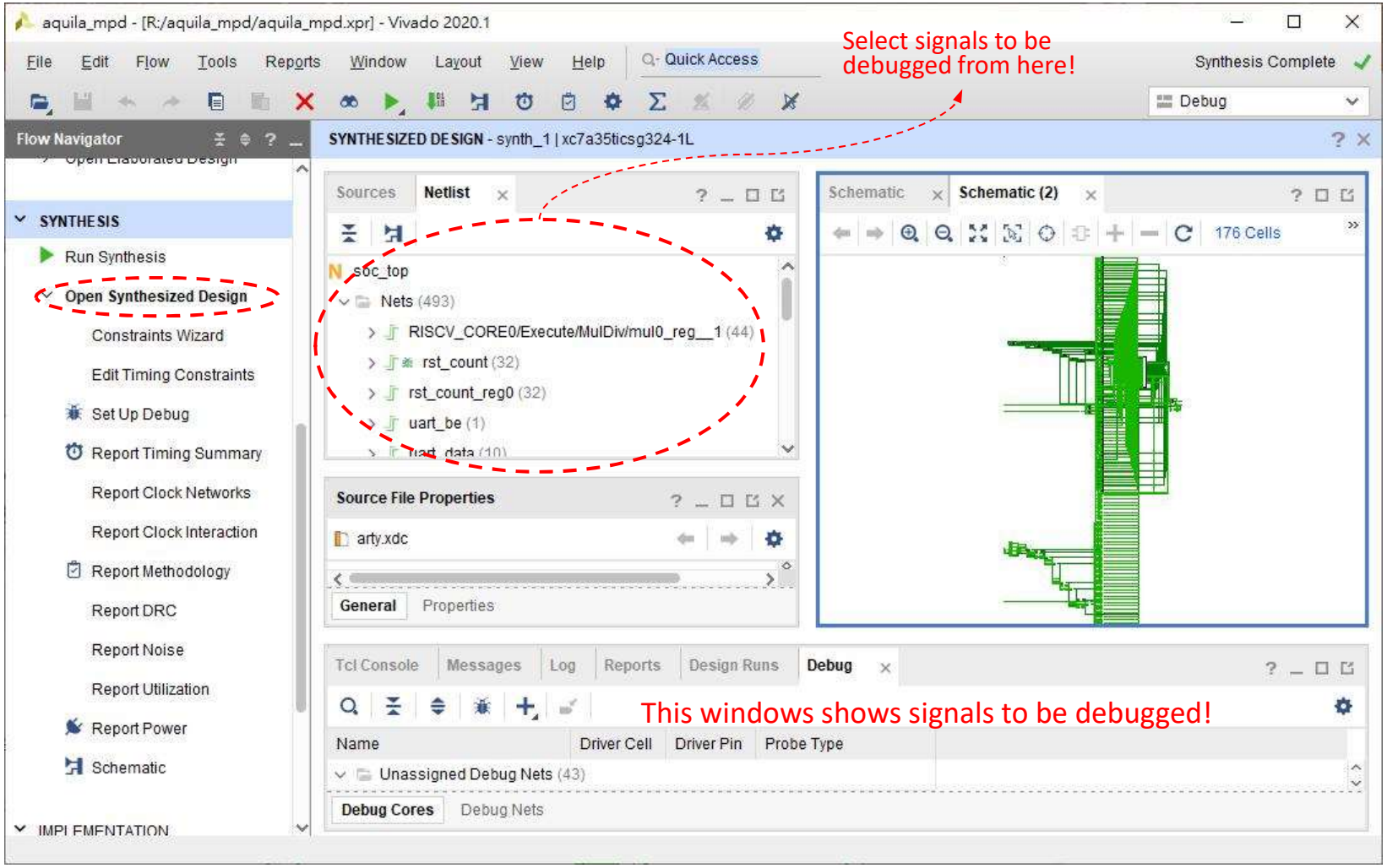
```
(* mark_debug = "true" *) wire my_signal
```

  This will turn on the "debug" attribute of `my_signal`.

- In Vivado, if your logic has signals with the debug attribute enabled, then:
  - The signals will not be "optimized-out" by the logic synthesizer
  - Vivado will insert an ILA IP into the synthesized design to monitor and capture these signals at runtime

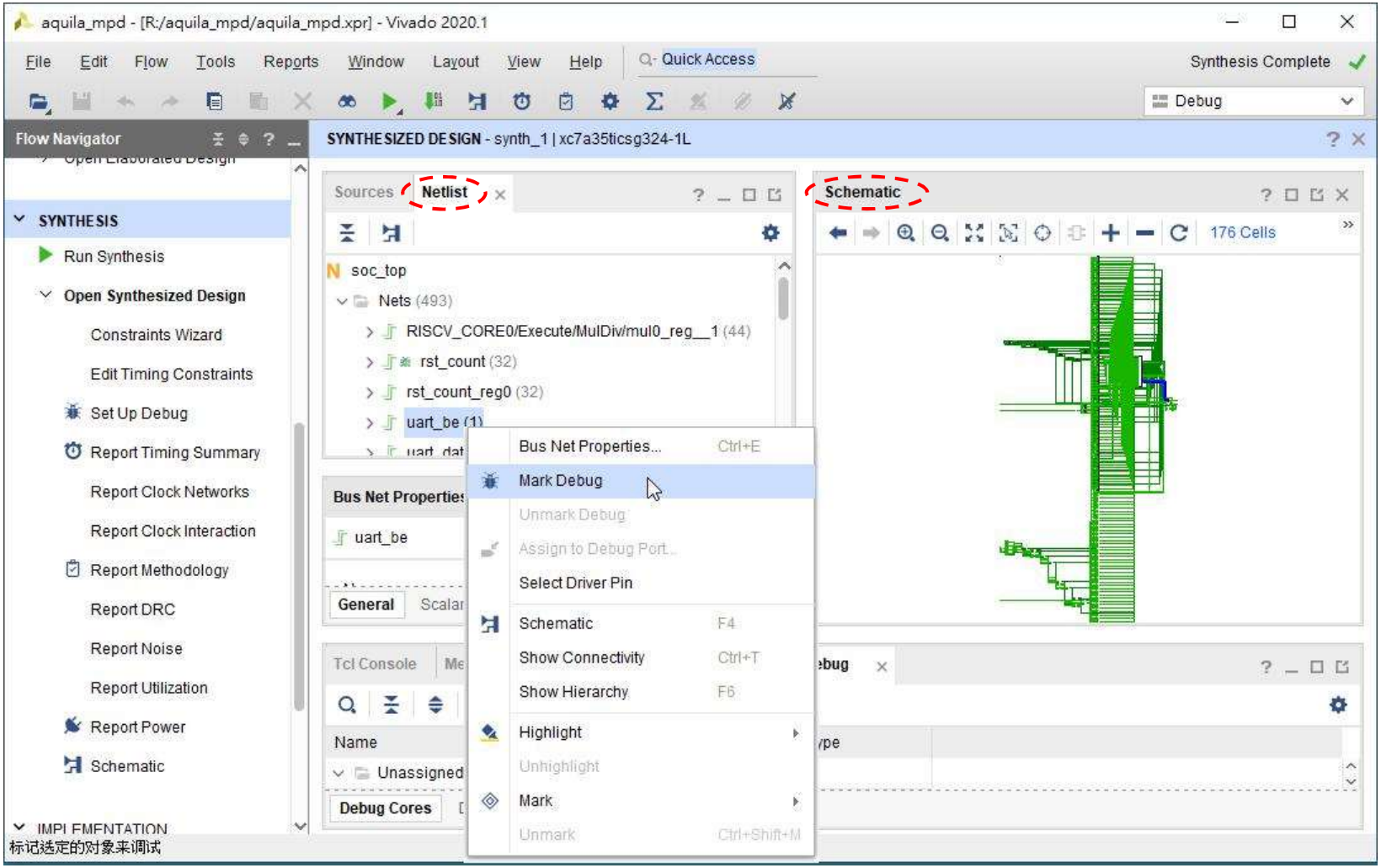
Mark Debug Signals Using GUI (1/2)

- To debug a circuit, open the synthesized design:



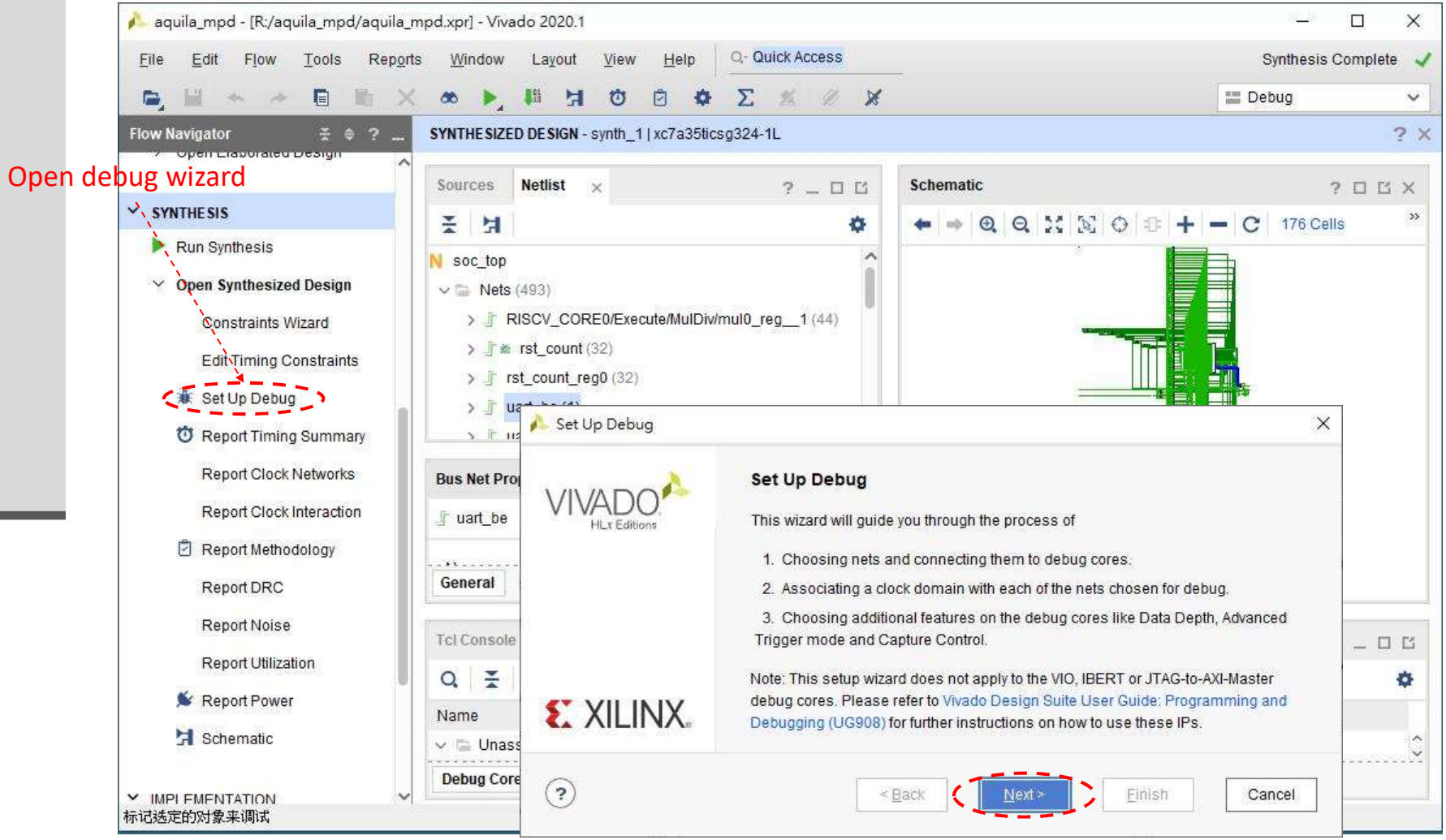
Mark Debug Signals Using GUI (2/2)

- Mark the signal in the “Netlist” or “Schematic” windows:



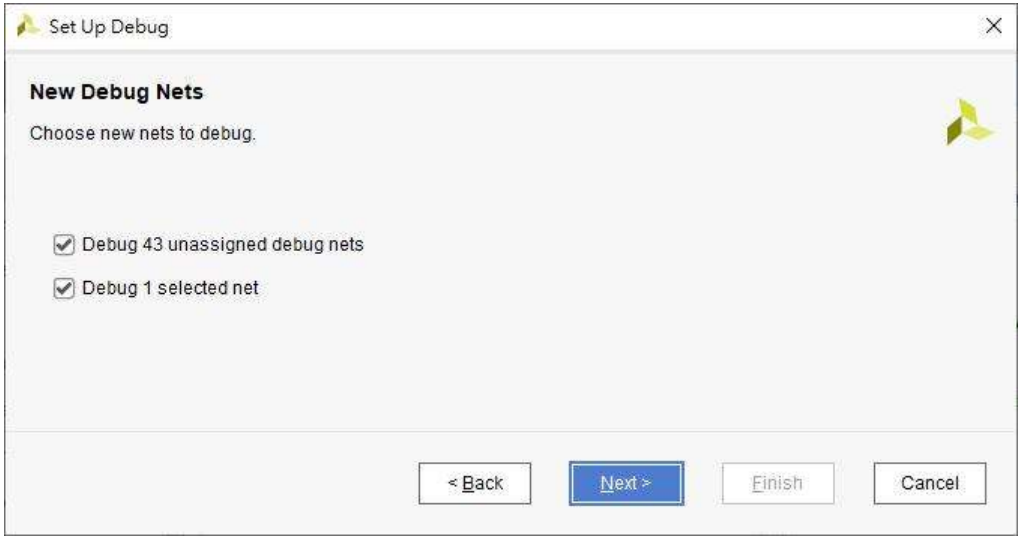
Set Up the Debug Wizard

- Open the “Set Up Debug” wizard:



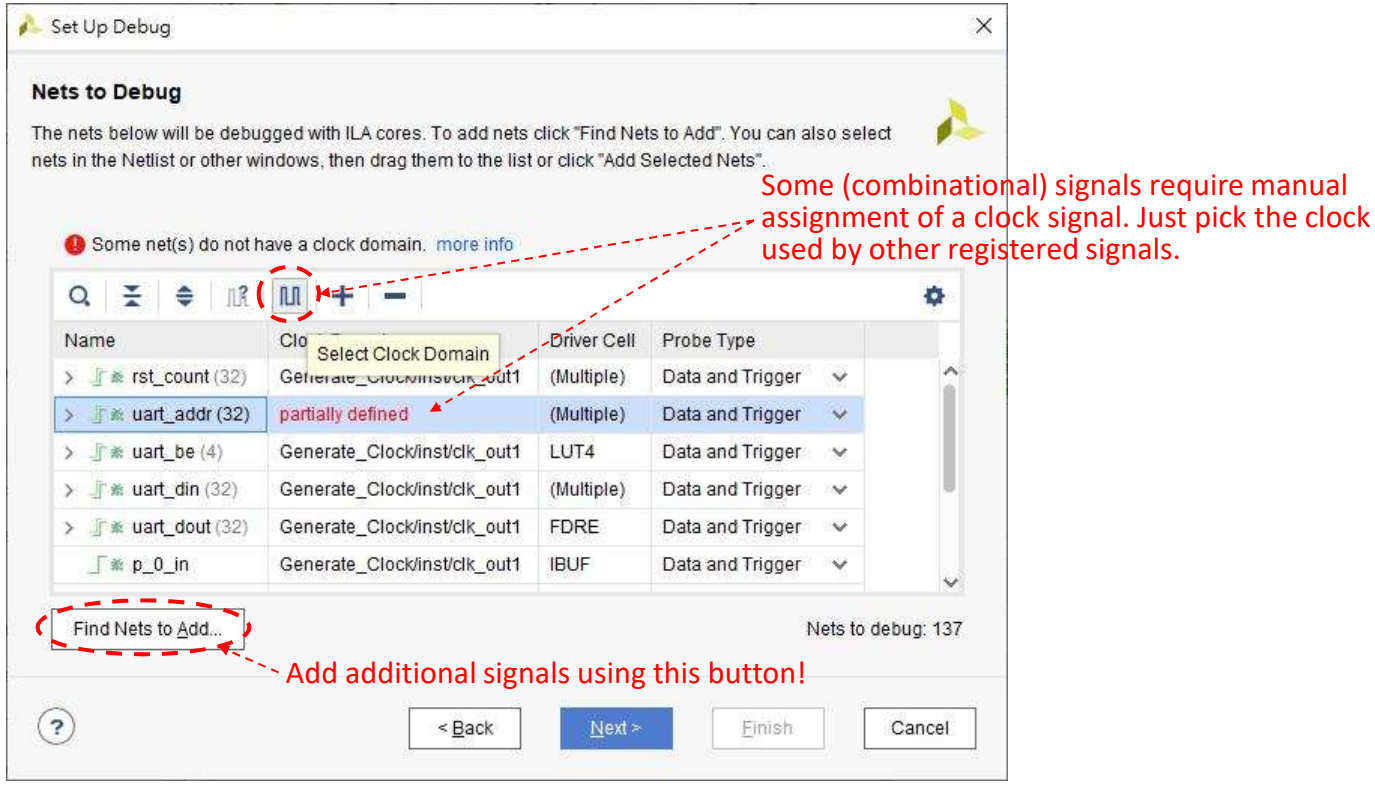
Confirm the Debugged Nets

- Just hit “Next”



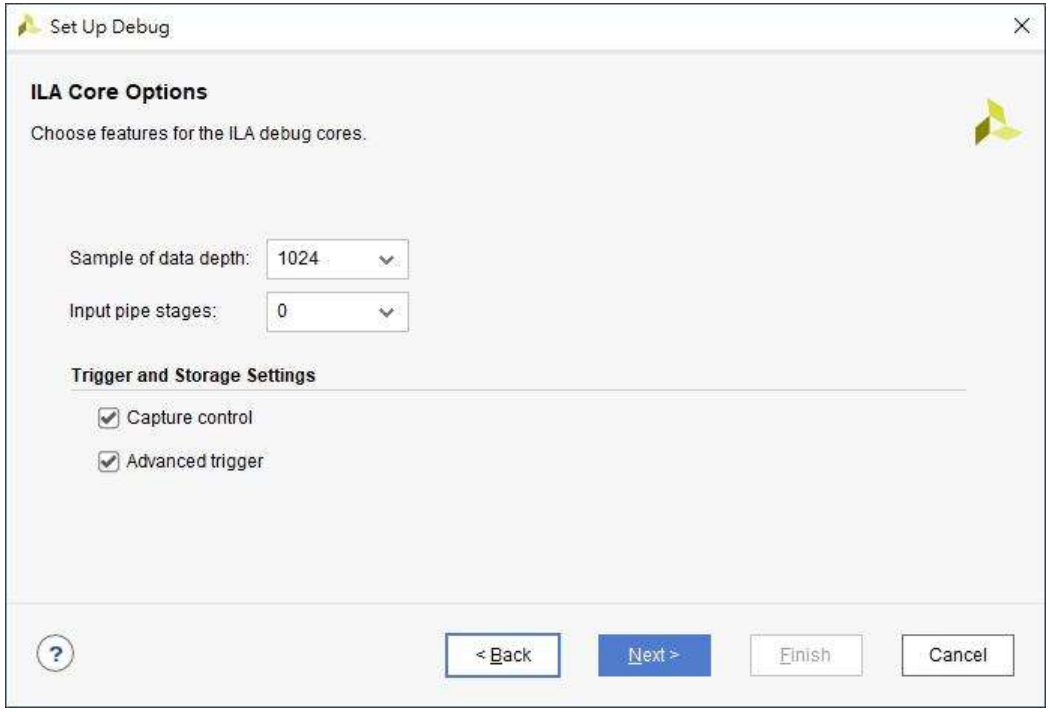
Double-Check Nets to Be Debugged

- You can add any missing signals in this dialog box
  - Note: some signals in your Verilog code may be missing due to the optimization process of the logic synthesizer!



Modify Trigger Options

- You can check both the “Capture control” and the “Advanced trigger” boxes



Save the New Debug Constraints

aquila_mpd - [R:/aquila_mpd/aquila_mpd.xpr] - Vivado 2020.1

File Edit Flow Tools Reports Window Layout View Help Quick Access

Synthesis Complete

Debug

Save Constraints (Ctrl+S)

SYNTHESIZED DESIGN * - synth_1 | xc7a35ticsg324-1L

SYNTHESIS

Run Synthesis

Open Synthesized Design

Constraints Wizard

Edit Timing Constraints

Set Up Debug

Report Timing Summary

Report Clock Networks

Report Clock Interaction

Report Methodology

Report DRC

Report Noise

Report Utilization

Report Power

Schematic

IMPLEMENTATION

Sources Netlist

soc_top

Nets (547)

RISCV_CORE0/Execute/MulDiv/mul0_reg__1

rst_count (32)

rst_count_reg0 (32)

sl_iport0_o_0 (37)

Schematic

176 Cells

Bus Net Properties

rst_count

General Scalar Nets

Tcl Console Messages Log Reports

Name Driver Cell

dbg_hub(labtools_xsdbm_v3)

Debug Cores Debug Nets

Save Constraints

Select a target file to write new unsaved constraints to. Choosing an existing file will update that file with the new constraints.

Create a new file

File type: XDC

File name:

File location: <Local to Project>

Select an existing file

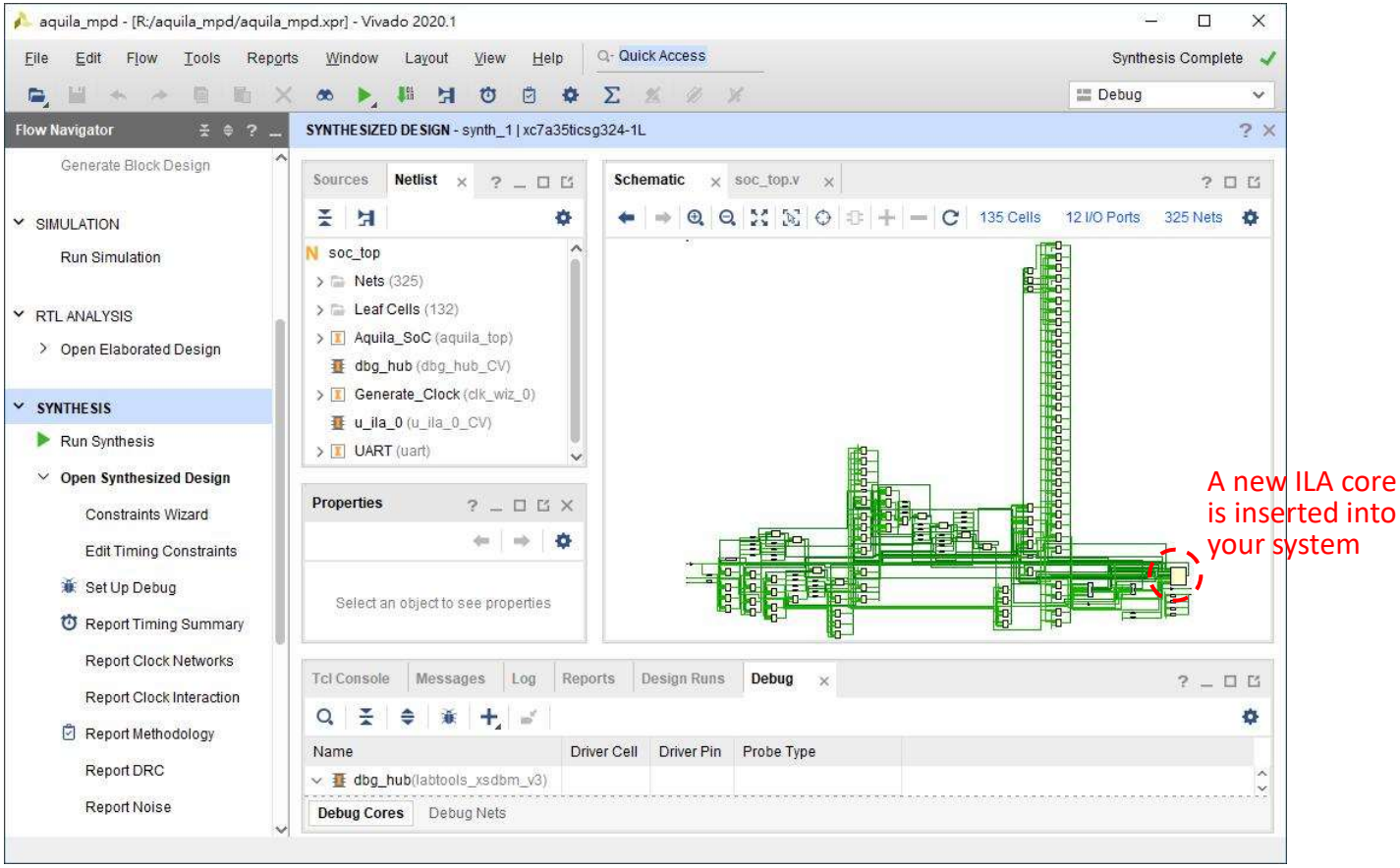
arty.xdc

OK Cancel

为当前设计保存这些约束

Re-Synthesis to Add ILA Debug Core

- An extra ILA IP will be added after forced re-synthesis
- Now, go ahead and generate the bitstream



Program the FPGA

aquila_mpd - [R:/aquila_mpd/aquila_mpd.xpr] - Vivado 2020.1

File Edit Flow Tools Reports Window Layout View Help Q- Quick Access write_bitstream Complete

Dashboard Default Layout

Flow Navigator

Run Simulation

RTL ANALYSIS

Open Elaborated Design

SYNTHESIS

Run Synthesis

Open Synthesized Design

IMPLEMENTATION

Run Implementation

Open Implemented Design

PROGRAM AND DEBUG

Generate Bitstream

Open Hardware Manager

Open Target

Program Device

Add Configuration Memory D

HARDWARE MANAGER - localhost/xilinx_tcf/Digilent/210319A27A39A

There are no debug cores. Program device Refresh device

Hardware

Name

xilinx_tcf/Digilent/210319A27A39A

xc7a35t_0 (1)

Hardware Device Properties

xc7a35t_0

General Properties

Tcl Console Messages Serial I/O Links

Resolution:

1. Make sure the clock connected to the debug hub (dbg_hub) core is a free running clock and is active.

2. Make sure the BSCAN_SWITCH_USER_MASK device property in Vivado Hardware Manager reflects the user scan chain setting in the de

Type a Tcl command here

Program Device

Select a bitstream programming file and download it to your hardware device. You can optionally select a debug probes file that corresponds to the debug cores contained in the bitstream programming file.

Bitstream file: R:/aquila_mpd/aquila_mpd.runs/impl_1/soc_top.bit

Debug probes file: R:/aquila_mpd/aquila_mpd.runs/impl_1/soc_top.ltx

Enable end of startup check

Program Cancel

The Hardware Manager with ILA View

aquila_mpd - [R:/aquila_mpd/aquila_mpd.xpr] - Vivado 2020.1

File Edit Flow Tools Reports Window Layout View Help Q- Quick Access write_bitstream Complete

Dashboard Default Layout

Signals which you can monitor and set triggers on.

Runtime waveforms

Flow Navigator

Open Block Des

Generate Block

SIMULATION

Run Simulation

RTL ANALYSIS

Open Elaborate

SYNTHESIS

Run Synthesis

Open Synthesiz

IMPLEMENTATION

Run Implement

Open Implemer

PROGRAM AND DEI

Generate Bitstre

Open Hardware

Open Targe

Program De

HARDWARE MANAGER - localhost/xilinx_tcf/Digilent/210319A27A39A

Hardware

Name Stat

localhost (1) Cor

xilinx_tcf/Digilent/210319A27A39A Ope

xc7a35t_0 (2) Pro

XADC (System Monitor)

Debug Probe Properties

uart_addr[31:0]

Source: NETLIST

Type: ILA

Probe type: Data and Trigger

General Properties Enumeration

hw_ila_1

Dashboard Options

Waveform - hw_ila_1

ILA Status: Idle

Name Value

rst_count[31:0]

uart_din[31:0]

Settings - hw_ila_1 Status - hw_ila_1

Core status Idle

Capture status - Window 1 of 1

Trigger Setup - hw Capture Setup - hw_i

Press the + button to add probes.

Trigger setup window

ILA configuration window

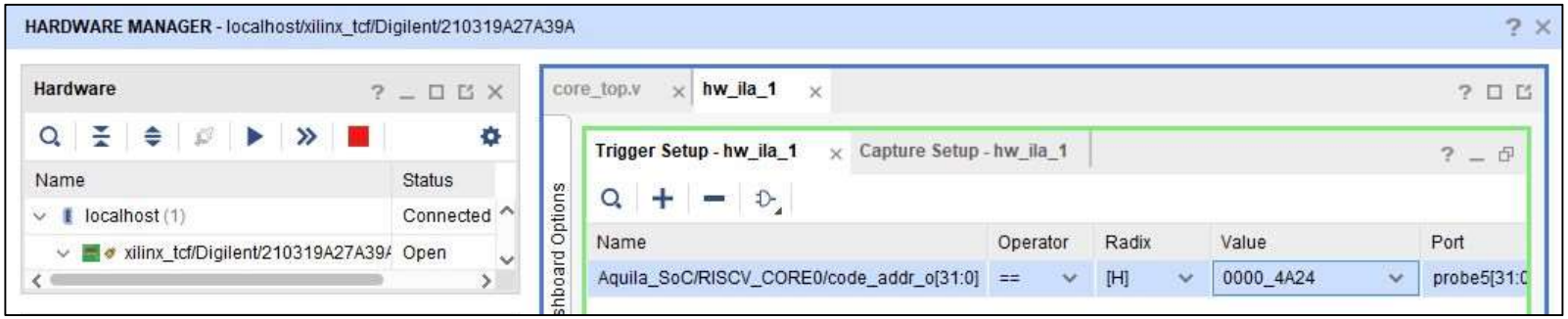
Tcl Console Messages Serial I/O Links Serial I/O Scans

```
add_wave -into {hw_ila_data_1.wcfg} -radix hex { {uart_addr} {uart_be} }
```

Type a Tcl command here

Setting a Trigger

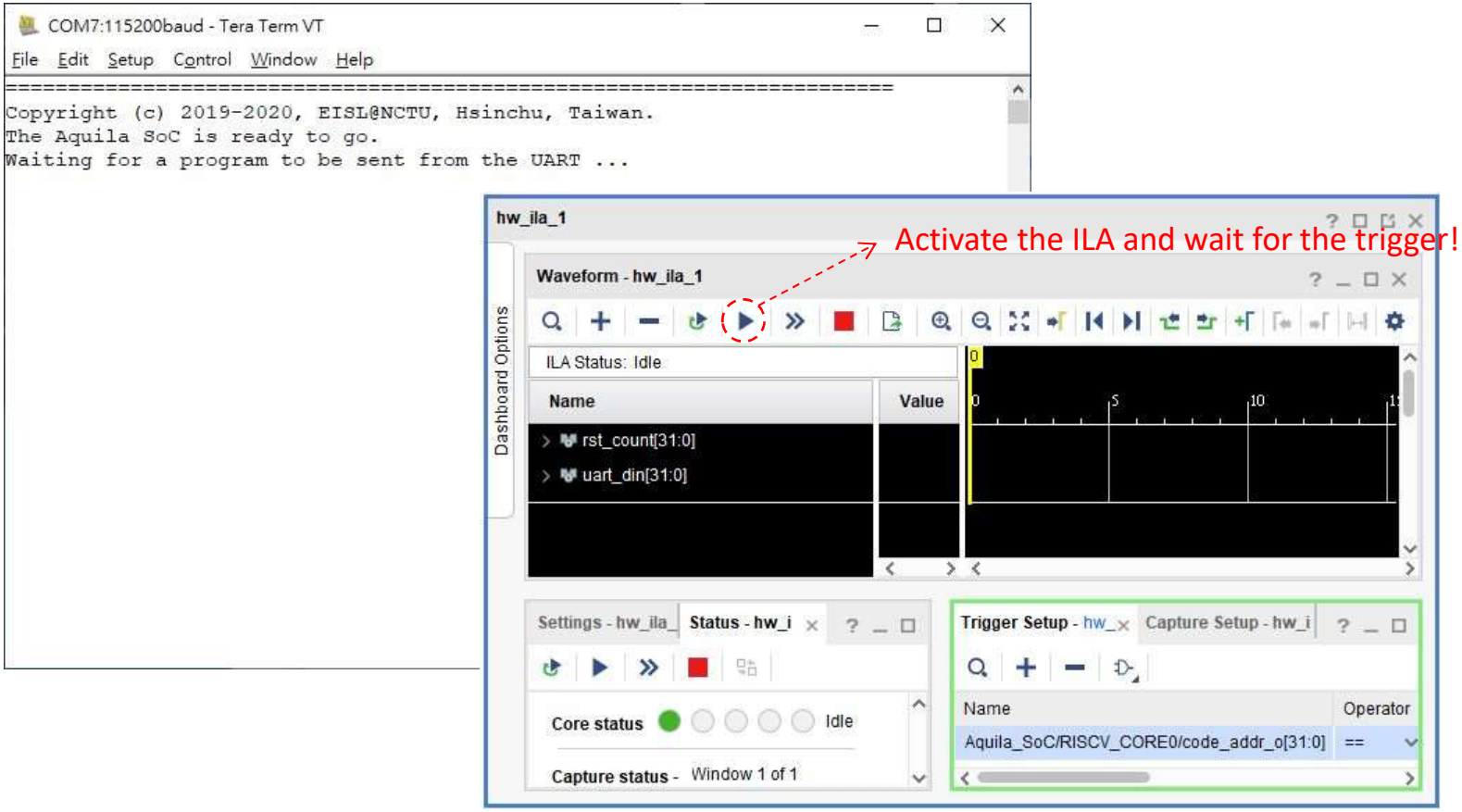
- A trigger is a signal condition that tells the ILA to begin capturing waveforms
  - Drag a signal from “Signal Name” window to “Trigger Setup” window to use it as a trigger
- Set the trigger condition:



- When `code_addr_o` in `core_top.v` equals `0x4A24` the ILA will be triggered to capture 1024 cycles of signals

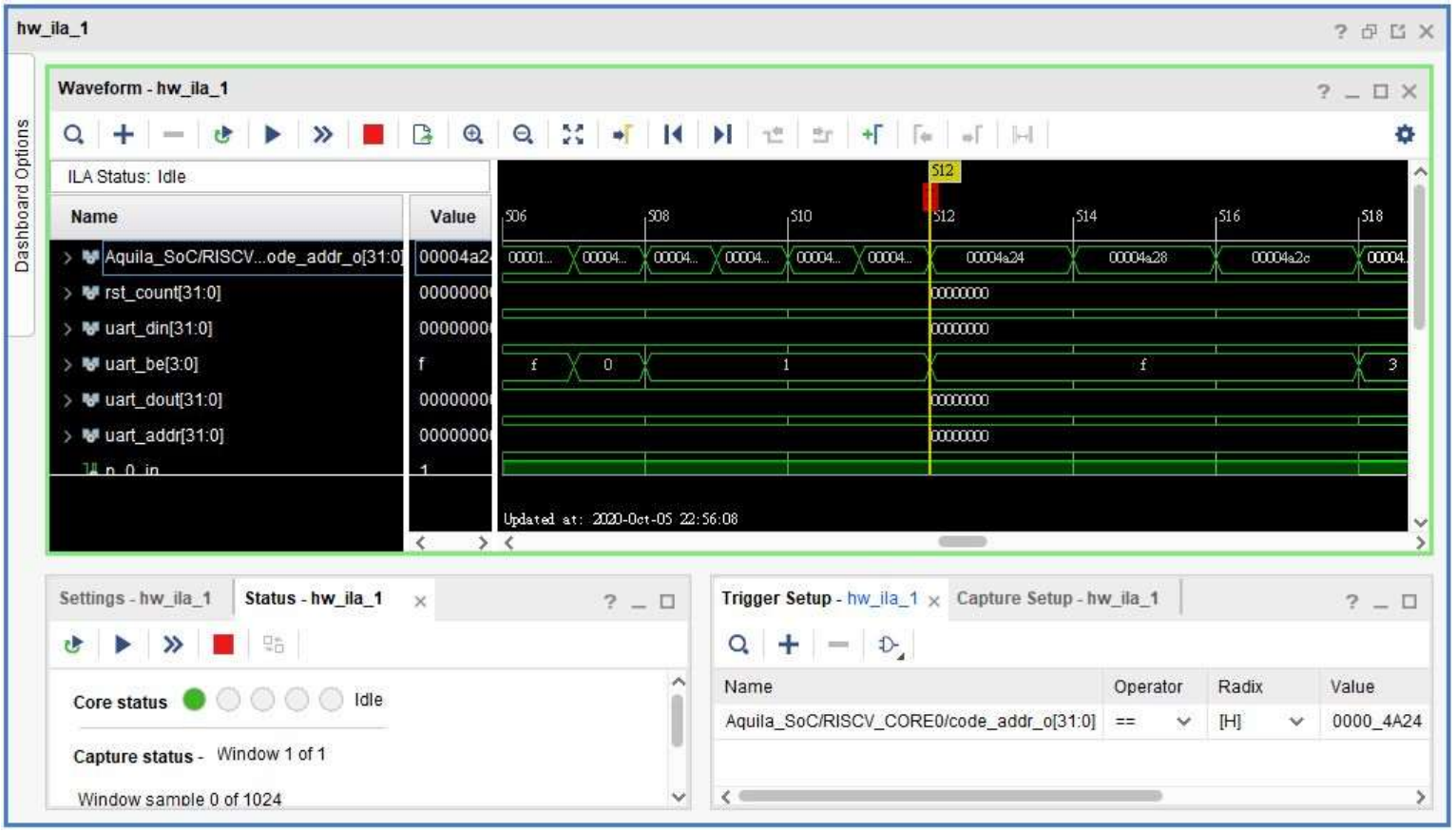
Capturing the Signals

- Now, you can activate the ILA, and send a program from the UART to FPGA to capture the waveform



Analyze the Captured Waveform

- When Aquila hits the main() of Dhrystone at `0x4A24`, the ILA will begin to capture waveforms:



Dhrystone Benchmarks Issues

- There is no perfect benchmarks. For Dhrystone, it's much less than perfect†:
  - Too many fixed-length string operations (`strcpy()` and `strcmp()`)
  - Code/data size too small to test cache performance
  - Did (could) not take into account RISC, VLIW, SIMD, and superscalar architecture
  - Dirty compilers that optimize for Dhrystone can achieve extra 50% higher DIMPS numbers
  - Code patterns do not reflect modern applications (is CPU performance critical here?)
  - *So, why do we use it in the first place?*

† https://www.eembc.org/techlit/datasheets/dhrystone_wp.pdf

Your Homework

- Go through the behavior simulation flow and the ILA probing flow.
- Rewrite `strcpy()` and `strcmp()`, see if you can increase the DMIPS/MHz performance
- Use the simulator or ILA to analyze the execution of your code and compare it against the original code
- Write a 4-page double-column report†:
  - Discuss what you have done to optimize the SW for DMIPS
  - Discuss what you have found using the Simulator or the ILA to analyze the execution of the program

† A report template can be downloaded from E3 website.