HW#1 Real-time Debugging of a HW-SW Platform

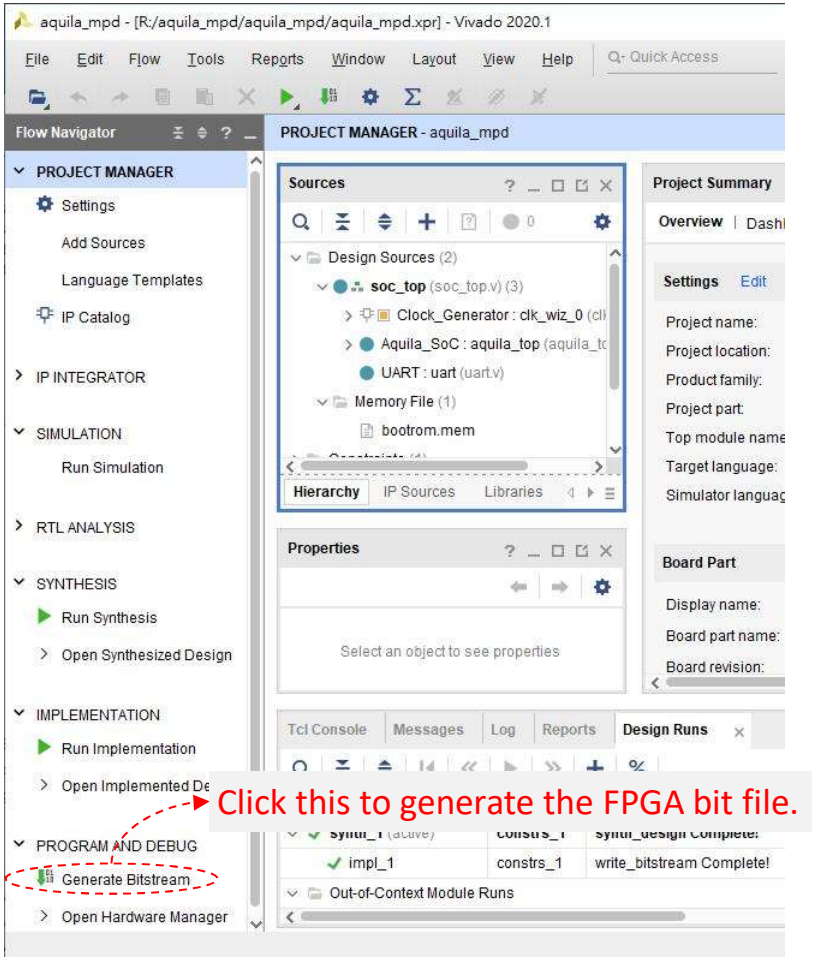
Chun-Jen Tsai
National Chiao Tung University
10/05/2021

Homework Goal

- In this homework, you will learn how to use Xilinx Integrated Logic Analyzer (ILA) for real-time debugging
- 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 modifications must produce an equivalent C program for all functions (e.g. no defective `strcpy()` that only works for Dhrystone!)
- Upload the source code you have modified and a two-page report to E3 by 10/18, 17:00.
- The TA will set up a schedule for you to demo to them.

Circuit Implementation for Real HW

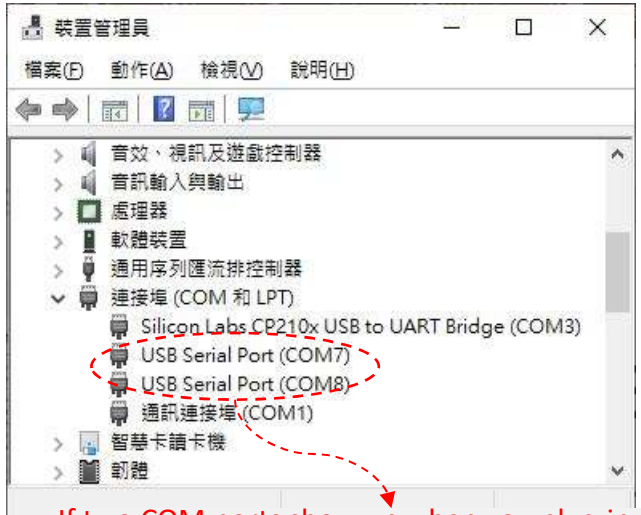
- To run an application on the Aquila SoC:
  1. Generate the FPGA bit file
  2. Connect Arty to the PC
  3. Run a Terminal emulator on the host PC
  4. Program the bit file into Arty
  5. Send a RISC-V application from the host PC to the serial port



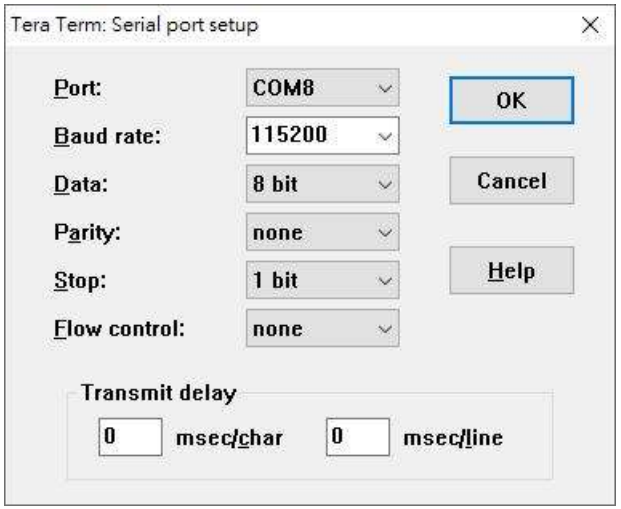
Terminal Settings (1/2)

- You can install the TeraTerm† on Windows, or GTKTerm on Linux to talk to the Aquila SoC in FPGA
  - It is better not to use “minicom” on Linux
- Pick the right COM port and set the UART parameters:

For Windows, check the device manager:



If two COM ports show up when you plug-in Arty, just pick the larger COM number.



† https://ttssh2.osdn.jp/index.html

Terminal Settings (2/2)

- For Linux, use "`sudo dmesg`" to see the Arty devices:

```
[2064339.294090] usb 1-7: Product: Digilent USB Device
[2064339.294091] usb 1-7: Manufacturer: Digilent
[2064339.294092] usb 1-7: SerialNumber: 210319A8C7F1
[2064339.297789] ftdi_sio 1-7:1.0: FTDI USB Serial Device converter detected
[2064339.297801] usb 1-7: Detected FT2232H
[2064339.297928] usb 1-7: FTDI USB Serial Device converter now attached to ttyUSB0
[2064339.299920] ftdi_sio 1-7:1.1: FTDI USB Serial Device converter detected
[2064339.299928] usb 1-7: Detected FT2232H
[2064339.300026] usb 1-7: FTDI USB Serial Device converter now attached to ttyUSB1
```

Arty JTAG programming device.

Arty serial port device.

- GTKTerm configuration:

Program the Aquila SoC into FPGA

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

File Edit Flow Tools Reports Window Layout View Help Quick Access

write_bitstream Complete

Default Layout

Flow Navigator

- IP INTEGRATOR
- SIMULATION
  - Run Simulation
- RTL ANALYSIS
- 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

PROJECT MANAGER - aquila_mpd

Sources

- Design Sources (2)
  - soc_top (soc_top.v) (3)
  - Memory File (1)
- Constraints (1)
- Simulation Sources (3)
- Utility Sources

Hierarchy | IP Sources | Libraries

Properties

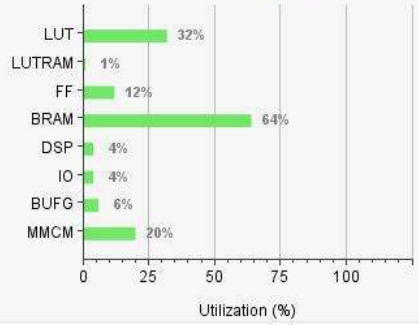
Select an object to see properties

Project Summary

Overview | Dashboard

Utilization: Post-Synthesis | Post-Implementation

Graph | Table



Power Summar

Total On-Chip Power:	0.234 W
Junction Temperature:	26.1 °C
Thermal Margin:	73.9 °C (15.4 W)
Effective ϑJA:	4.8 °C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Medium

Implemented Power Report

Tcl Console | Messages | Log | Reports | Design Runs

Name	Constraints	Status	Incremental	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BR
impl_1	constrs_1	write_bitstream Complete!	Off	3.376	0.000	0.037	0.000	0.000	0.234	0	6649	4887	3

Out-of-Context Module Runs

Click these to connect to FPGA and program it.

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	Status
xilinx_tcf/Digilent/210319A27A39A	Open
xc7a35t_0 (1)	Programme

Hardware Device Properties

xc7a35t_0

General Properties

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

Resolution:

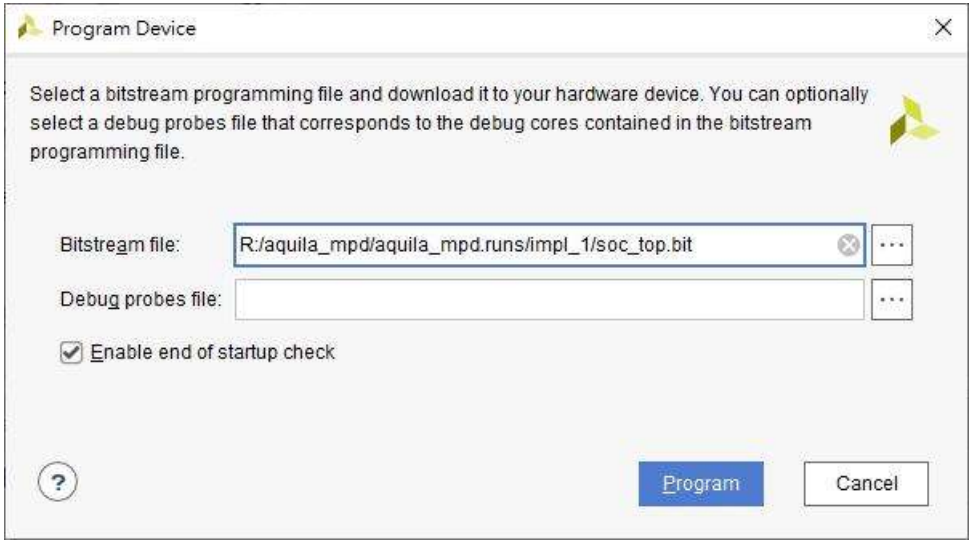
1. Make sure the clock connected to the debug hub (dbg_hub) core is a free running clock and is active.

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

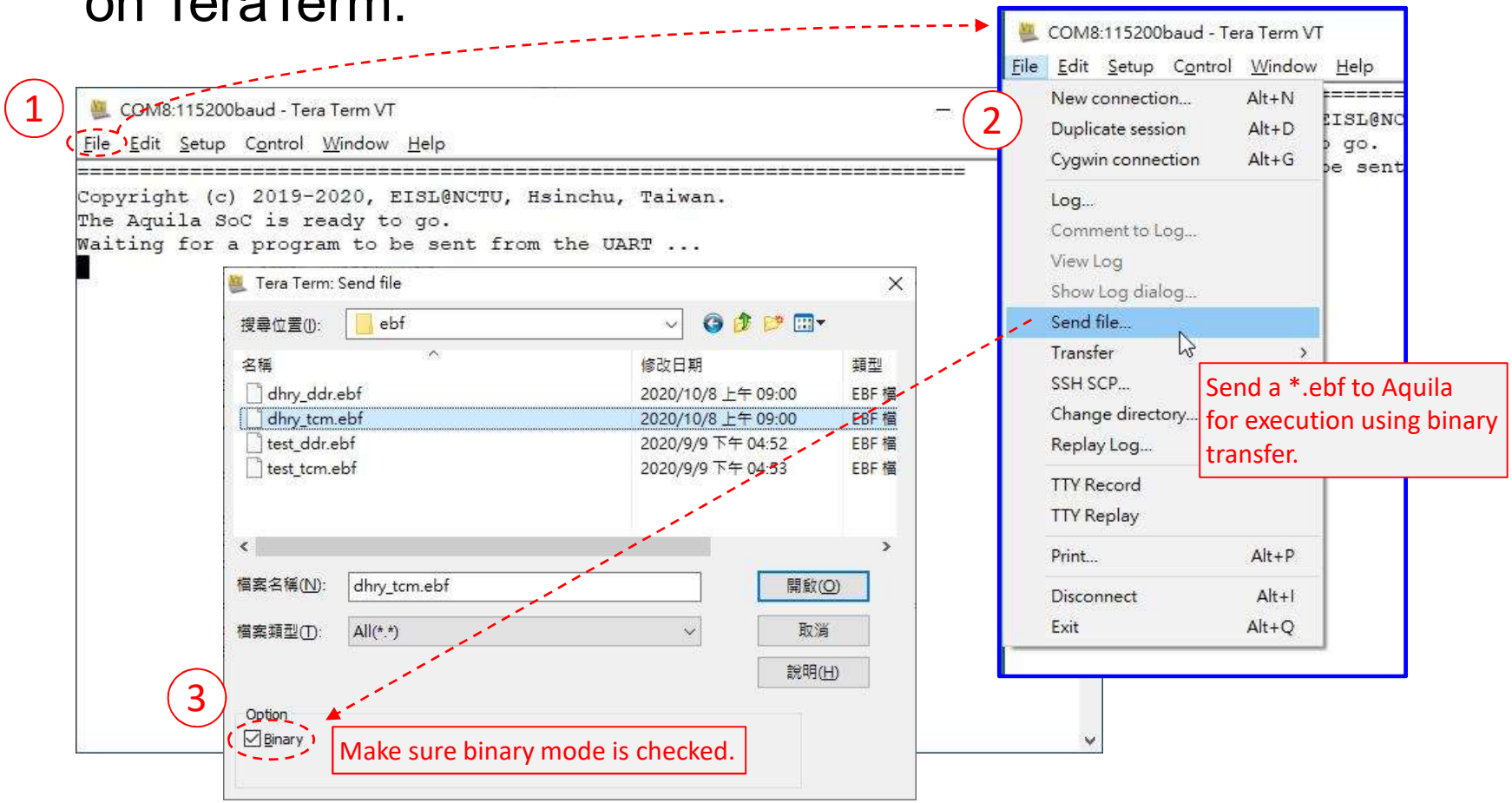
Select the BIT File for Programming

- If circuit synthesis is done, a `soc_top.bit` file will be under `aquila_mpd/aquila_mpd.runs/impl_1/`:



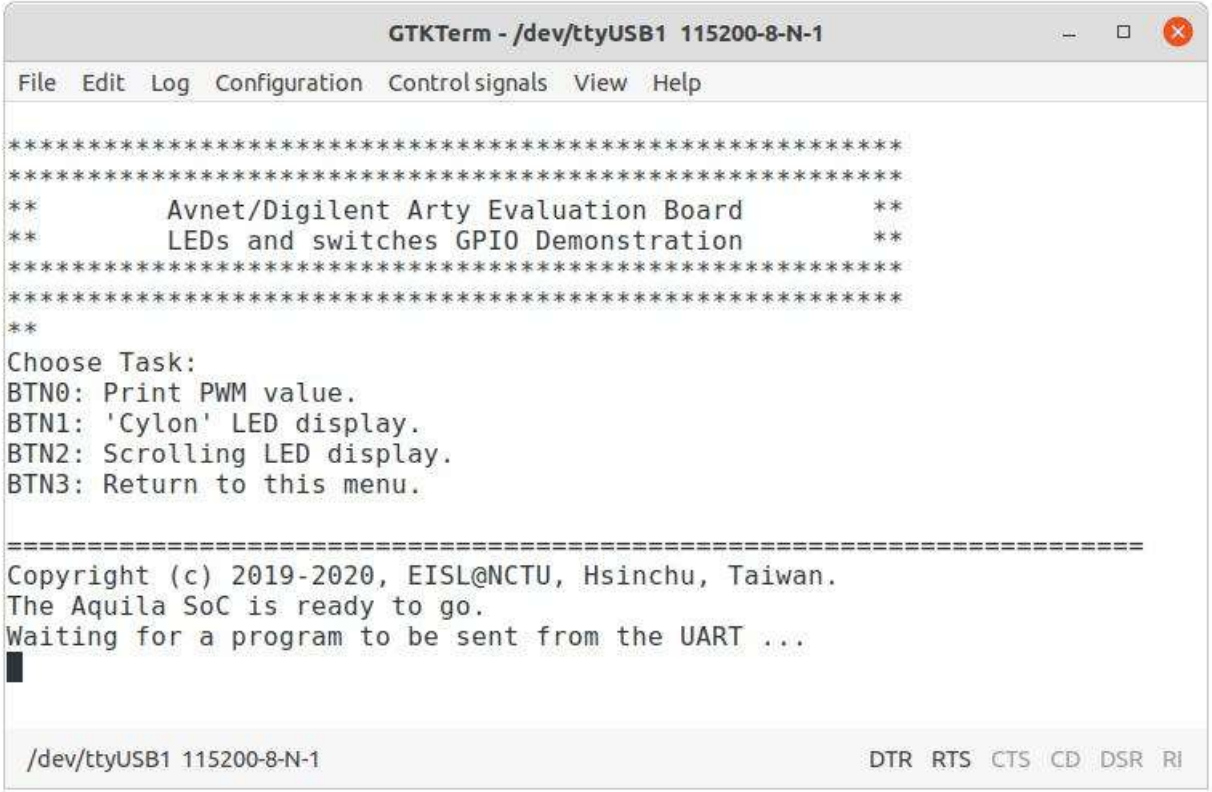
Run an Executable on Windows

- Once FPGA is programmed, a message is displayed on TeraTerm:



Run an Executable on Linux

- Under Linux, you can send an ebf file to Arty using the command: `$ cat dhry.ebf > /dev/ttyUSB1`

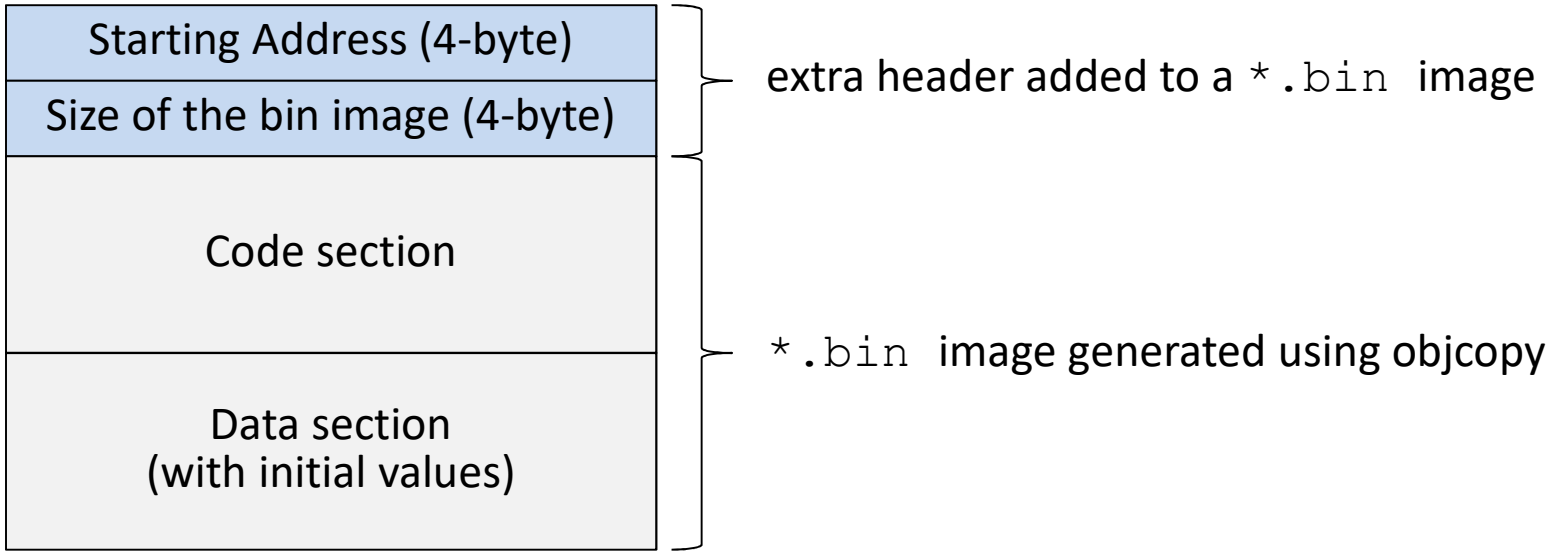


Executable File Format (1/2)

- An executable file tells a computing system how to set up the memory for instruction code and data
  - A common executable file format for a POSIX-compliant system is the ELF format
  - ELF file format is quite complex to parse and load
- For embedded systems, `*.bin` file is often used
  - `*.bin` is a plain runtime memory image
  - GCC command `objcopy` can convert a `*.elf` to a `*.bin`.
  - No location information in `*.bin` files
- For this lab, we use the simple `*.ebf` file format. In the future, we will use the standard `*.elf` format

Executable File Format (2/2)

- Layout of a `*.ebf` file:



Dhrystone Result

- Send `dhry.ebf` to Aquila and you will see:

COM8:115200baud - Tera Term VT

File Edit Setup Control Window Help

```
Enum_Loc:            1
        should be:   1
Str_1_Loc:           DHRYSTONE PROGRAM, 1'ST STRING
        should be:   DHRYSTONE PROGRAM, 1'ST STRING
Str_2_Loc:           DHRYSTONE PROGRAM, 2'ND STRING
        should be:   DHRYSTONE PROGRAM, 2'ND STRING

It tooks 7.99 seconds.
Microseconds for one run through Dhrystone: 7.986669
Dhrystones per Second:                      125208.6
VAX MIPS:                                   71.3
DMIPS/Mhz:                                   0.71


----------------------------------------------------------------------
Aquila execution finished.


Program exit with a status code 0

----------------------------------------------------------------------
Aquila execution finished.
Press <reset> on the FPGA board to reboot the cpu ...
```

Analyze the Execution of Aquila SoC

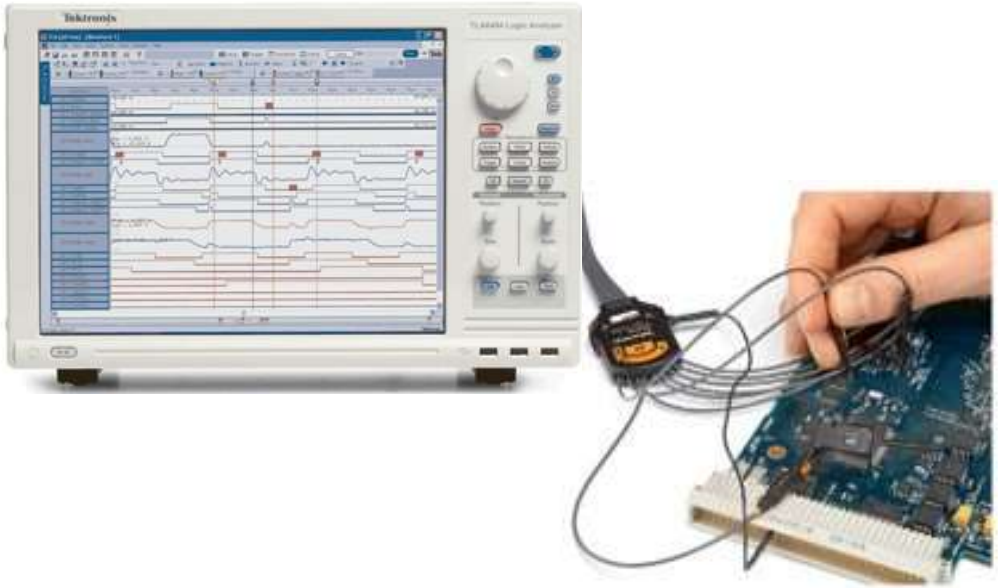
- To analyze the behavior of Aquila, you can use a RTL simulator or the Integrated Logic Analyzer (ILA)

- You have used the simulator in HW#0:
  - Simulation of a circuit is very slow
  - 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

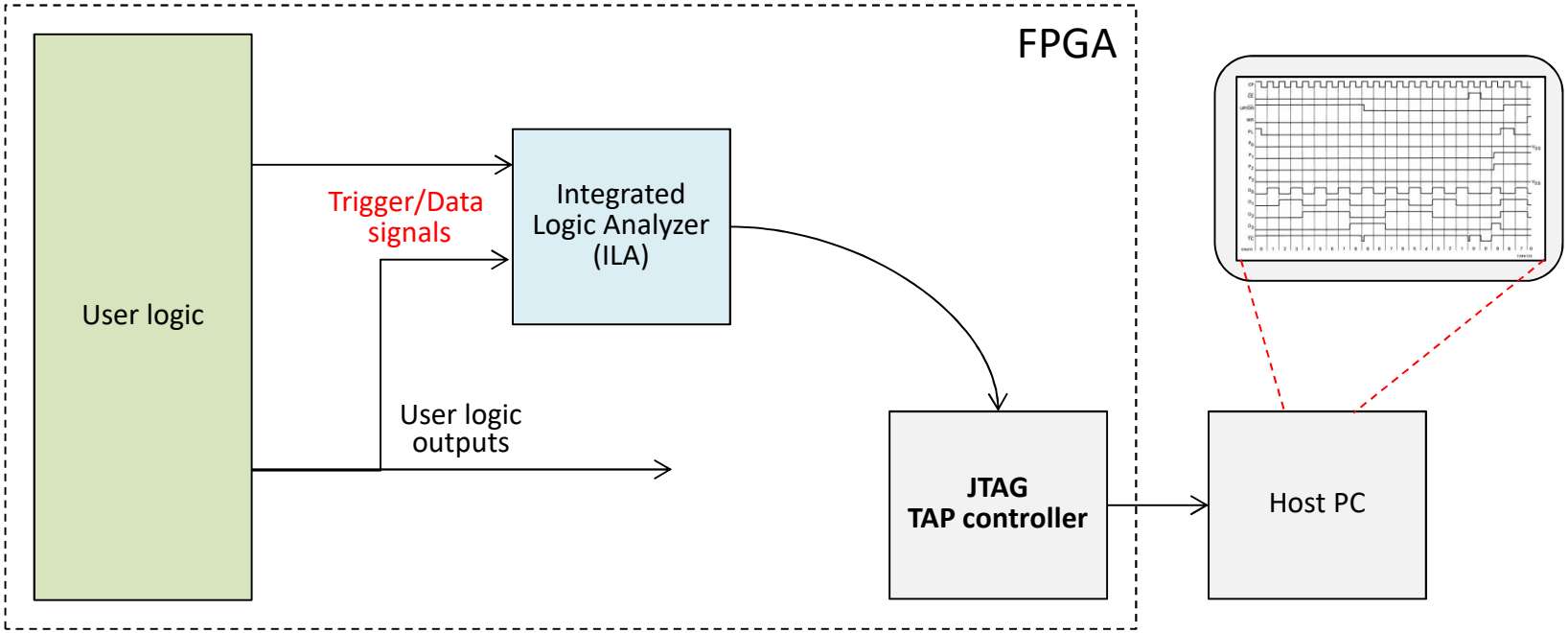
Real-Time Probing Using Vivado

- 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
- Do not mark the system clock. 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:

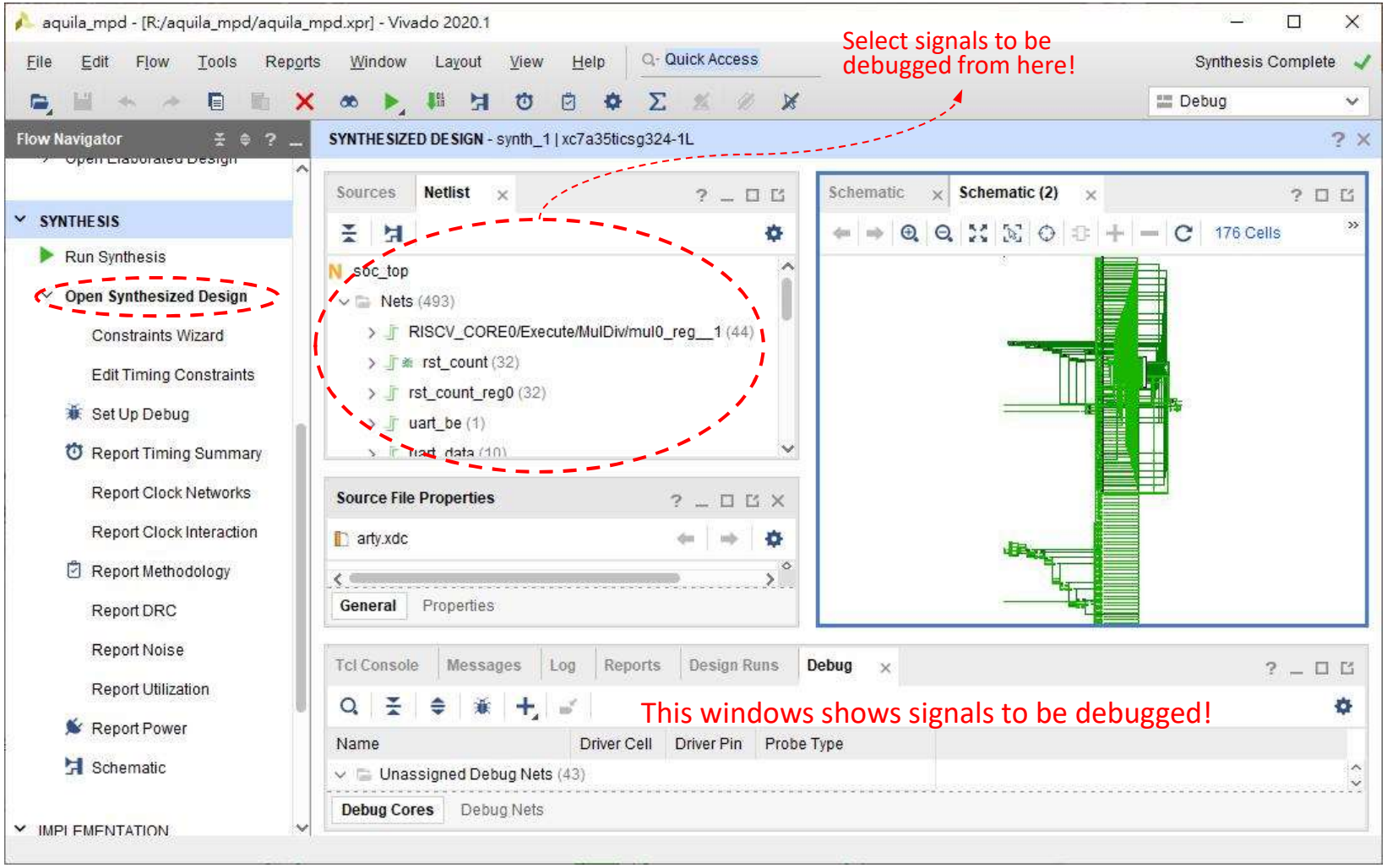
```
(* mark_debug = "true" *) wire my_signal
```

  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

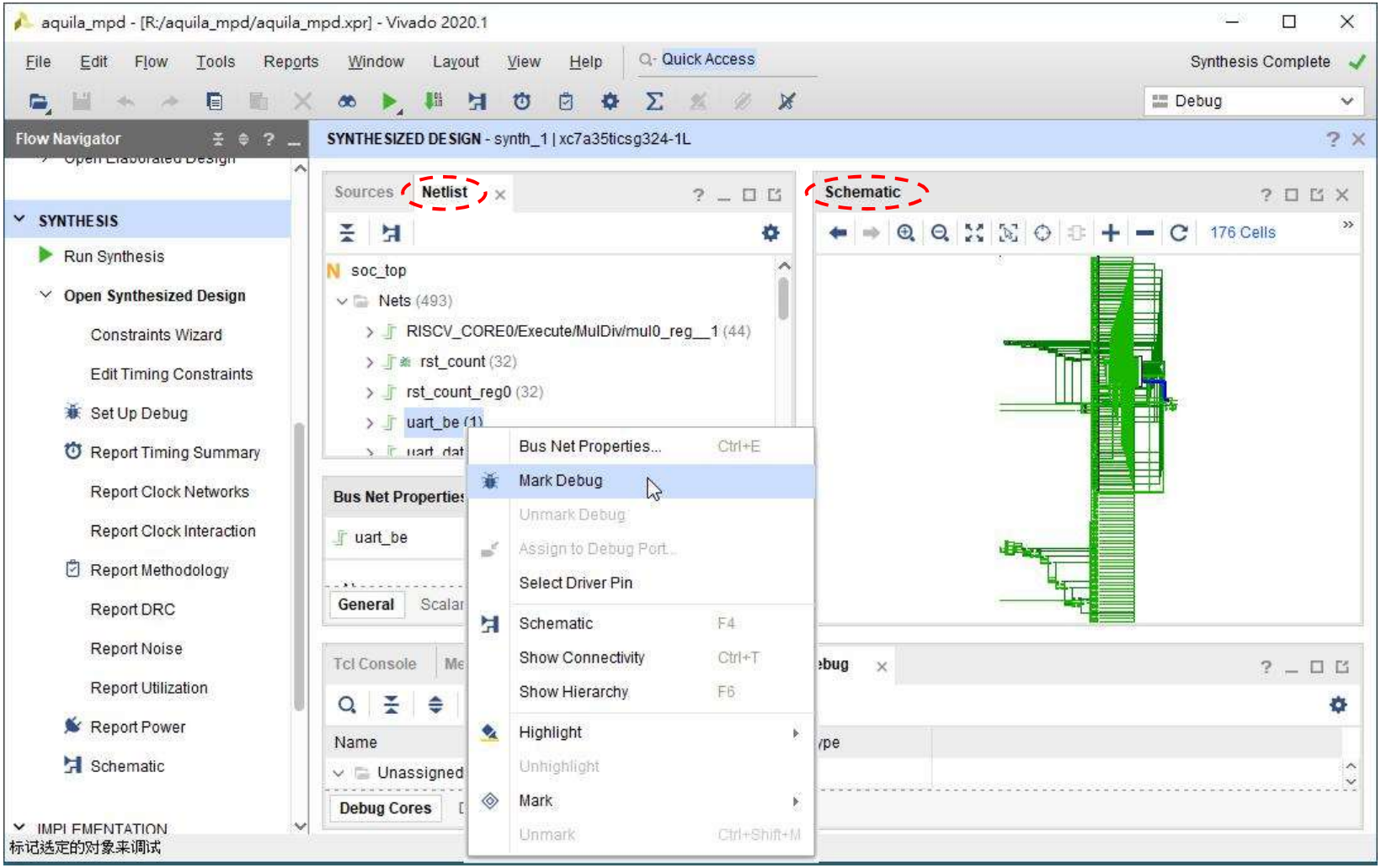
Mark Debug Signals Using GUI (1/2)

- To debug a circuit, open the synthesized design:



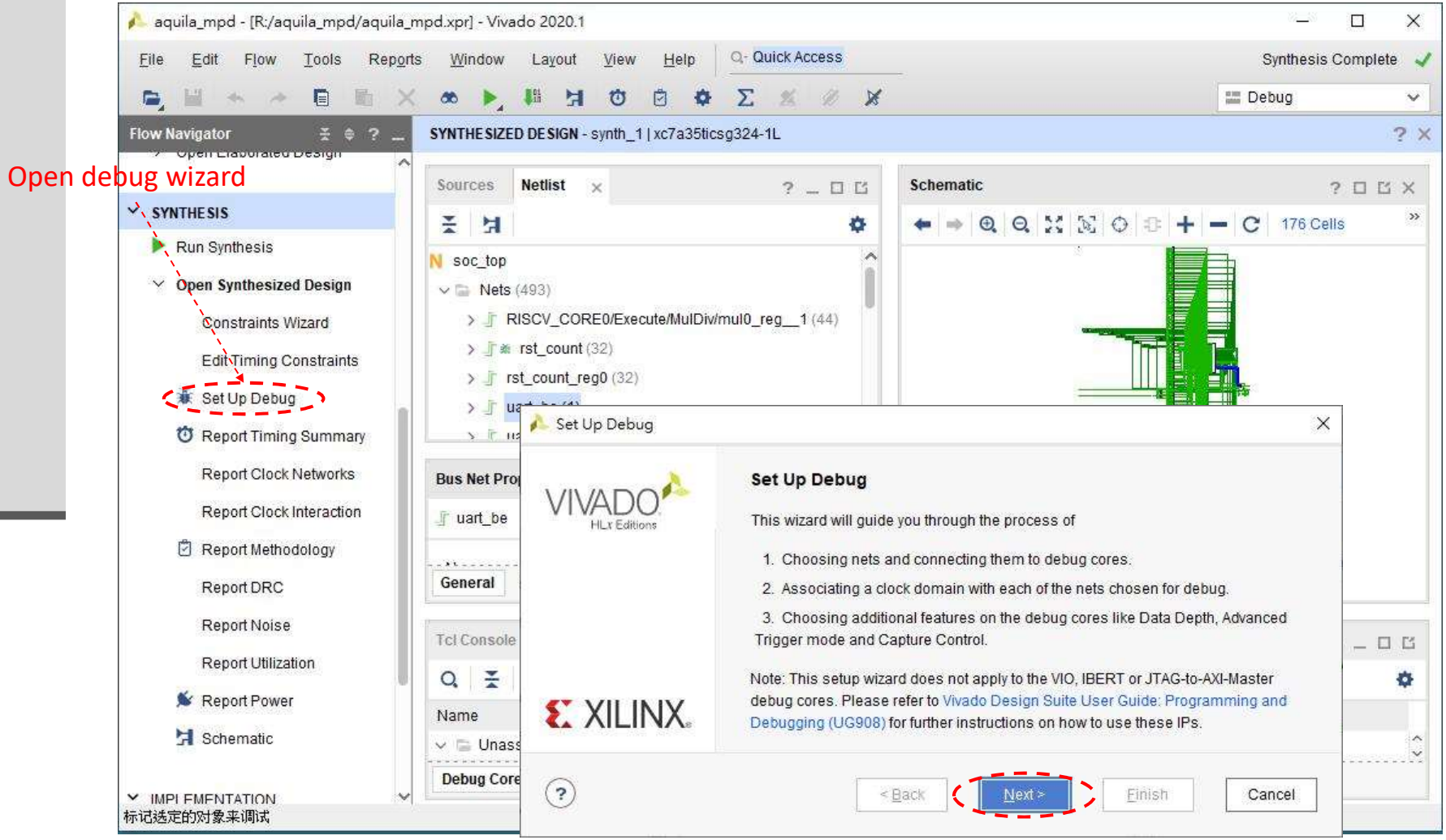
Mark Debug Signals Using GUI (2/2)

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



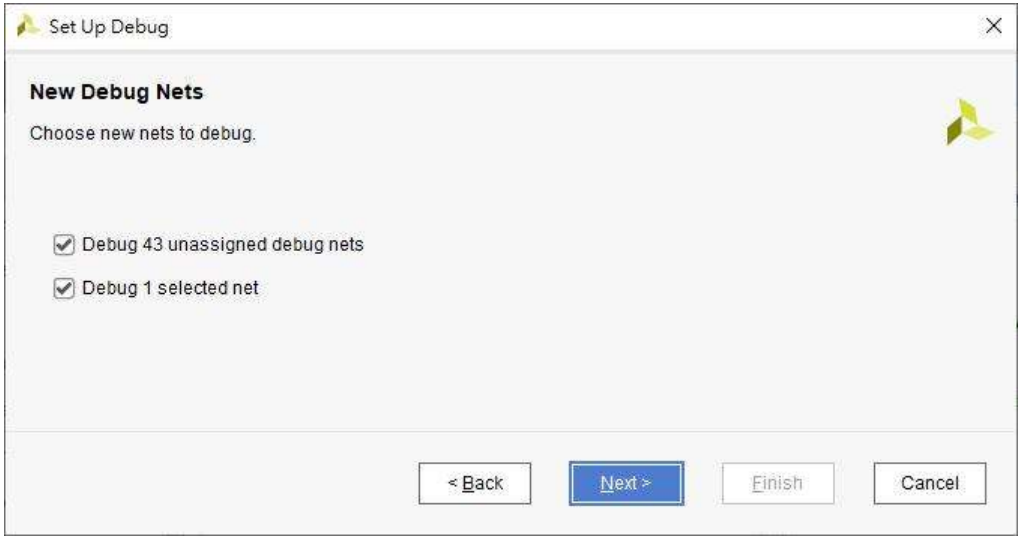
Set Up the Debug Wizard

- Open the “Set Up Debug” wizard:



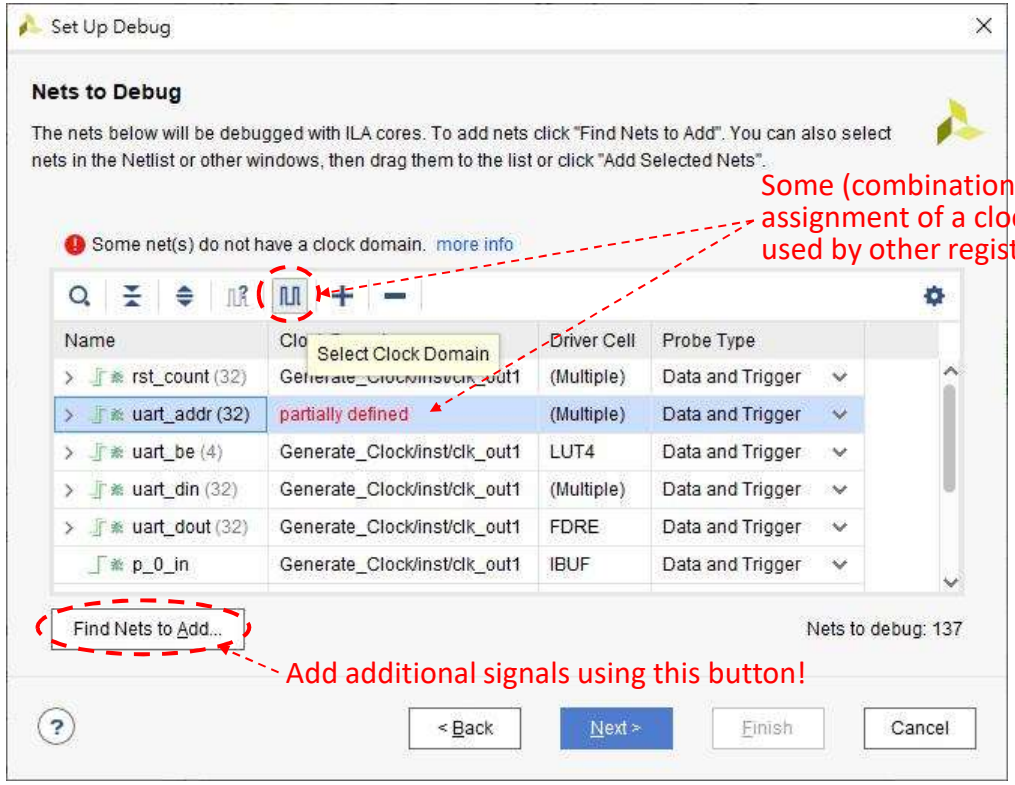
Confirm the Debugged Nets

- 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!

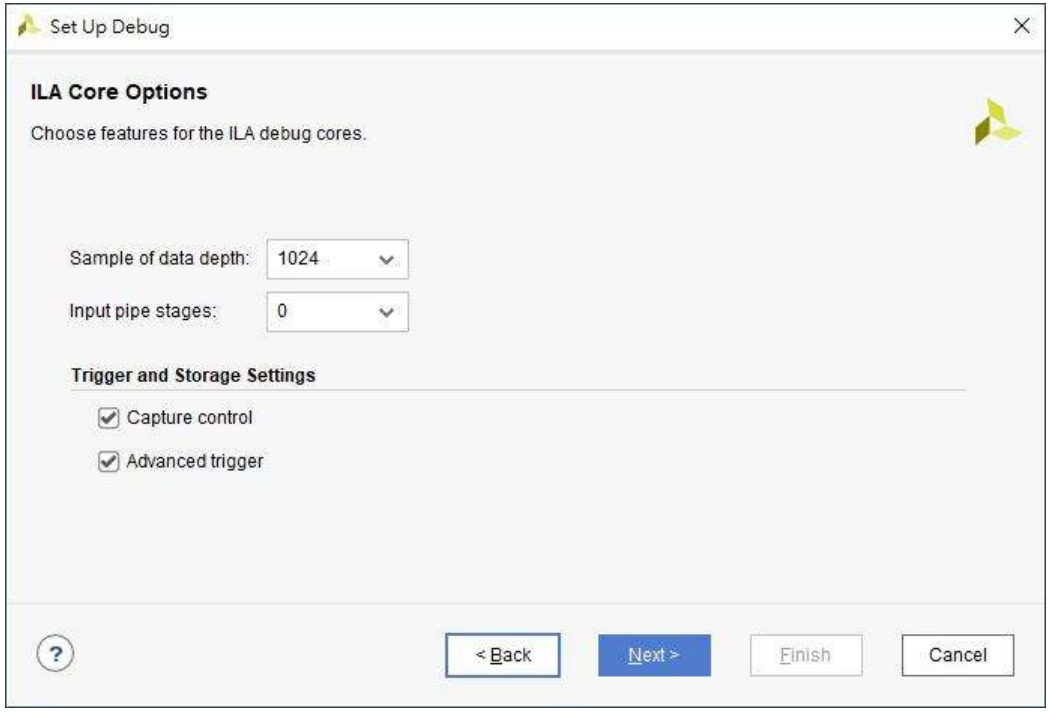


Some (combinational) signals require manual assignment of a clock signal. Just pick the clock used by other registered signals.

Add additional signals using this button!

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

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.

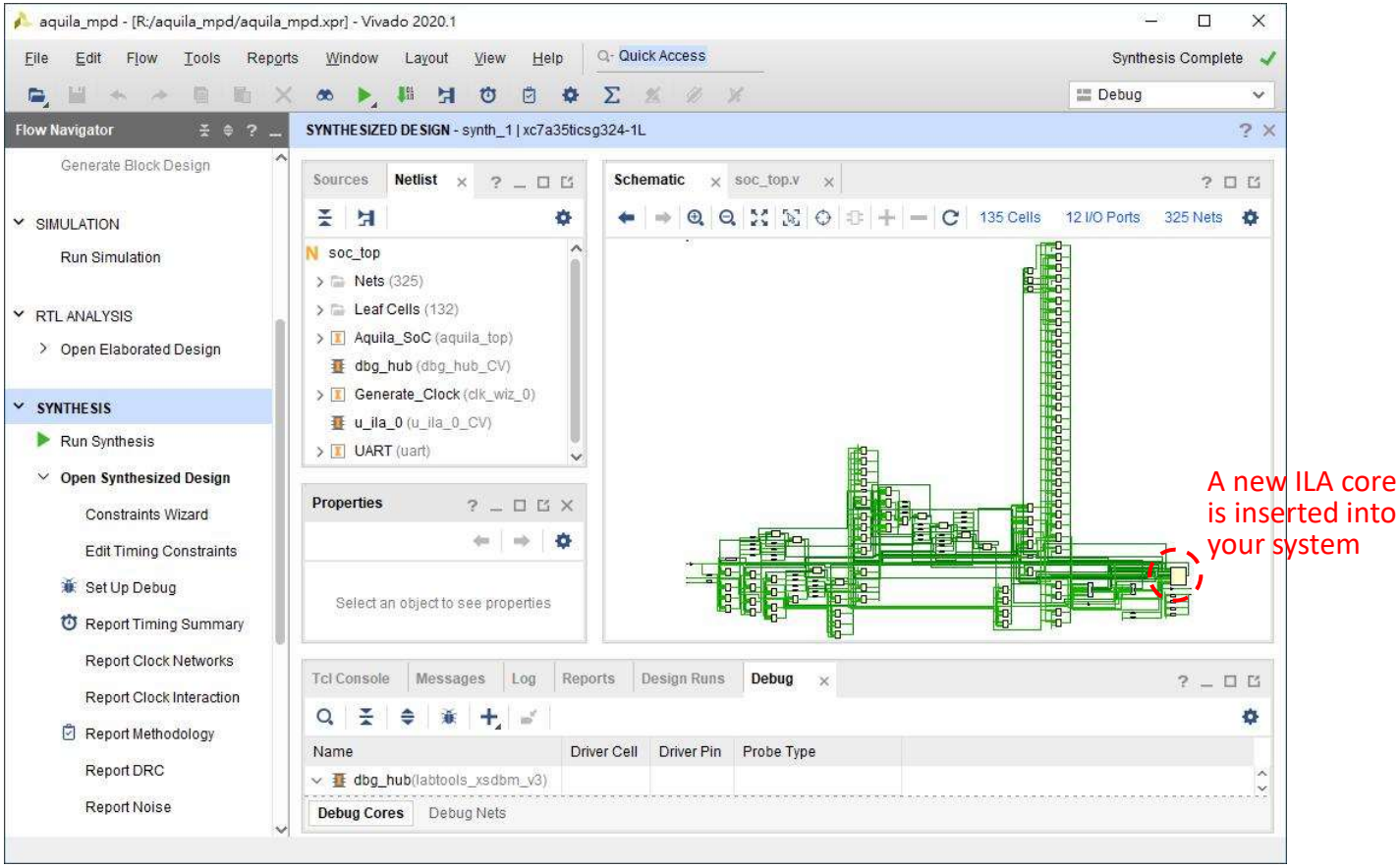
- Create a new file
  - File type: XDC
  - File name:
  - File location: <Local to Project>
- Select an existing file
  - arty.xdc

OK Cancel

为当前设计保存这些约束

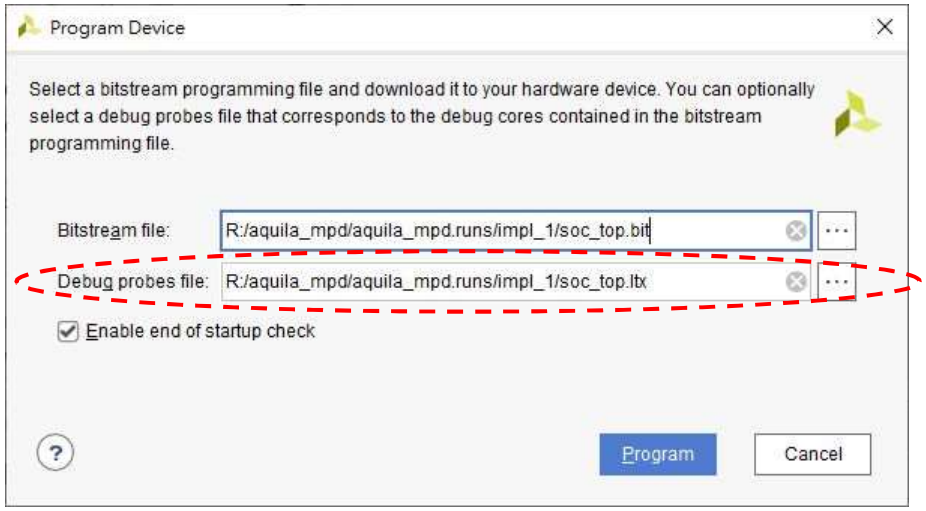
Re-Synthesis to Add ILA Debug Core

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



Program the FPGA for ILA

- Once you hit the “program device” menu item, you will see an extra ILA configuration file is selected:



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

Signals which you can monitor and set triggers on.

Runtime waveforms

Trigger setup window

ILA configuration window

HARDWARE MANAGER - localhost/xilinx_tcf/Digilent/210319A27A39A

Waveform - hw_ila_1

ILA Status: Idle

Name | Value

rst_count[31:0]

uart_din[31:0]

Debug Probe Properties

uart_addr[31:0]

Source: NETLIST

Type: ILA

Probe type: Data and Trigger

Core status Idle

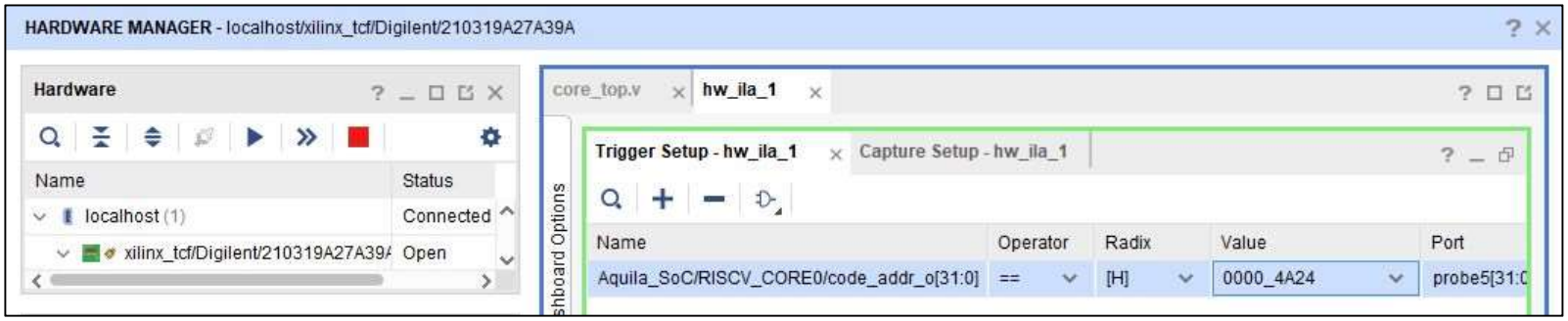
Capture status - Window 1 of 1

Tcl Console

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

Setting a Trigger

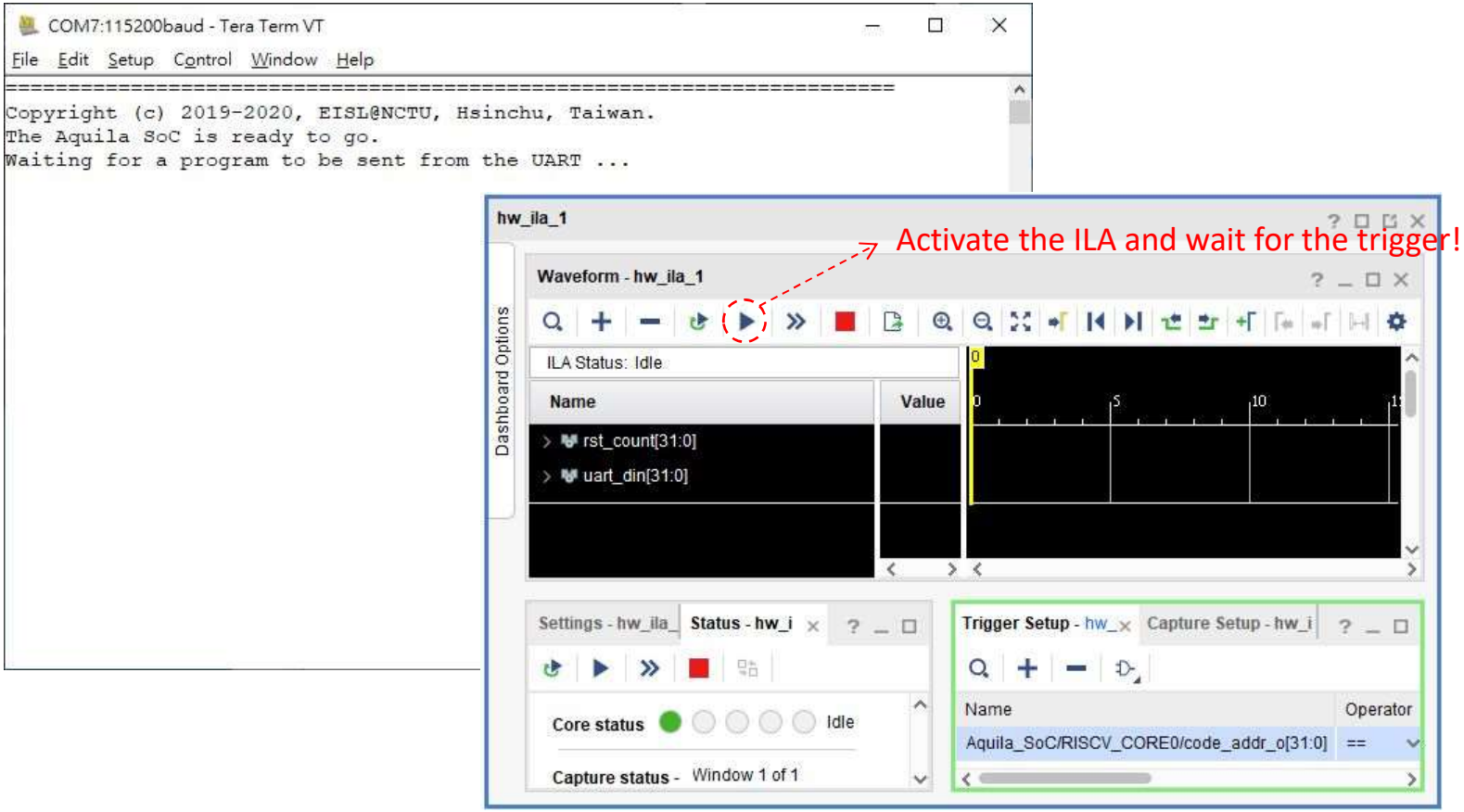
- A trigger is a signal condition that tells the ILA to begin capturing waveforms
  - You can drag a signal from the “Signal Name” window to the “Trigger Setup” window to use it as a trigger
- Set the trigger condition:



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

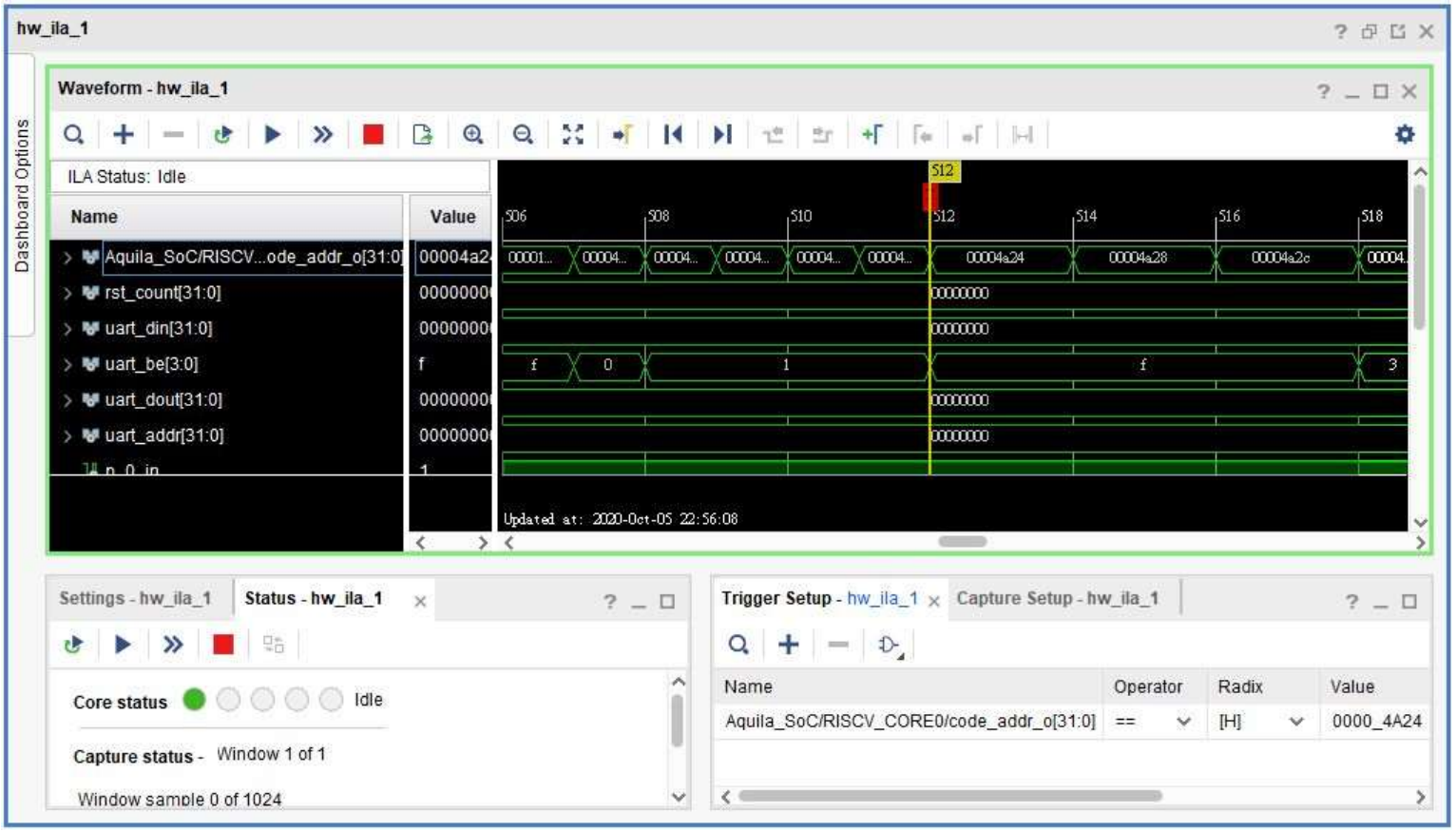
Capturing the Signals

- 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:



Tracing the Execution of a Function

- One of the reasons that DMIPS of Aquila is a little bit low is because the C library (`elibc`) is not optimized!

- For example, you can use ILA to trace the execution of the `strcpy()` and analyze why the processor cannot execute the function efficiently
  - The `*.objdump` tells you the start address of the function
  - `strcpy()` begins at `0x00001ea0` in my build
  - Pay attention to the stall cycles for program execution

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
  - Dirty compilers that optimize for Dhrystone can achieve extra 50% higher DIMPS numbers
  - Did not take into account architecture features (e.g., RISC, VLIW, SIMD, and superscalar)
  - Code patterns do not reflect modern applications (is CPU critical for modern applications?)
  - *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 2-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.