

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



Chun-Jen Tsai
National Chiao Tung University
10/07/2022

Homework Goal

profiling: criminal profiling

Berkley: analyze H.W

- ❑ In this homework, you will add profiling hardware to the Aquila core to analyze the program execution behavior
- ❑ CoreMark benchmark will be used as the target
 - You will learn how to profile a program on the real platform
 - You will also learn how to use Xilinx Integrated Logic Analyzer (ILA) for real-time debugging
- ❑ Deadline: 10/20, 17:00
 - Upload your source code and a two-page report to E3
 - The TAs will set up a schedule for you to demo

CoreMark is a synthetic benchmark, better than Dhrystone. But still has its problems
↳ precalculate the instructions proportion ... and designed benchmarks.

Synthesis-Execution Flow

- ❑ To run the HW-SW system on an FPGA, you must:
 - Generate the HW bit file
 - Power on the Arty board
 - Use a terminal program to connect to the FPGA via UART
 - Program the FPGA
 - Send an ELF program to the FPGA via UART
 - Wait for the program to execute and print results

Implement the Circuit into FPGA

Synthesize the Aquila SoC

The screenshot displays the Vivado 2022.1 IDE interface for the 'aquila_mpd' project. The Flow Navigator on the left shows the project hierarchy, including 'PROJECT MANAGER', 'SIMULATION', 'RTL ANALYSIS', 'SYNTHESIS', 'IMPLEMENTATION', and 'PROGRAM AND DEBUG'. The Project Manager in the center shows the 'Sources' pane with the project hierarchy and the 'Source File Properties' pane for 'sram_dp.v'. The Project Summary on the right shows the 'Overview' tab with a bar chart of utilization and a table of power metrics. The Design Runs table at the bottom shows the completion of synthesis and implementation.

Project Summary - Power Metrics:

Metric	Value
Total On-Chip Power:	0.321 W
Junction Temperature:	26.5 °C
Thermal Margin:	58.5 °C (12.7 W)
Effective θ_{JA} :	4.6 °C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Medium

Design Runs Table:

Name	Constraints	Status	WNS	TNS	WHS	THS	WBSS	TPWS	Total Power	Failed Routes	Methodology
synth_1 (active)	constrs_1	synth_design Complete!									
impl_1	constrs_1	write_bitstream Complete!	1.926	0.000	0.081	0.000		0.000	0.321	0	36 Warn

Click this to generate the FPGA bit file.

Click this to connect to the FPGA board.

Program The FPGA

Flow Navigator

- Language Templates
- IP Catalog
- 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 Device

Hardware Manager - localhost/xilinx_tcf/Digilent/210319B26B02A

There are no debug cores. **Program device** Refresh device

Name	Status
localhost (1)	Connected
xilinx_tcf/Digilent/210319B26B02	Open
xc7a100t_0 (1)	Not program

Properties

Select an object to see properties

Tcl Console

```
open_hw_target: Time (s): cpu = 00:00:05 ; elapsed = 00:00:06 . Memory (MB): peak = 3324.000 ; gain = 1675.453
set_property PROGRAM_FILE {R:/aquila_build/aquila_mpd/aquila_mpd.runs/impl_1/soc_top.bit} [get_hw_devices xc7a100t_0]
current_hw_device [get_hw_devices xc7a100t_0]
refresh_hw_device -update_hw_probes false [lindex [get_hw_devices xc7a100t_0] 0]
INFO: [Labtools 27-1435] Device xc7a100t (JTAG device index = 0) is not programmed (DONE status = 0).
```

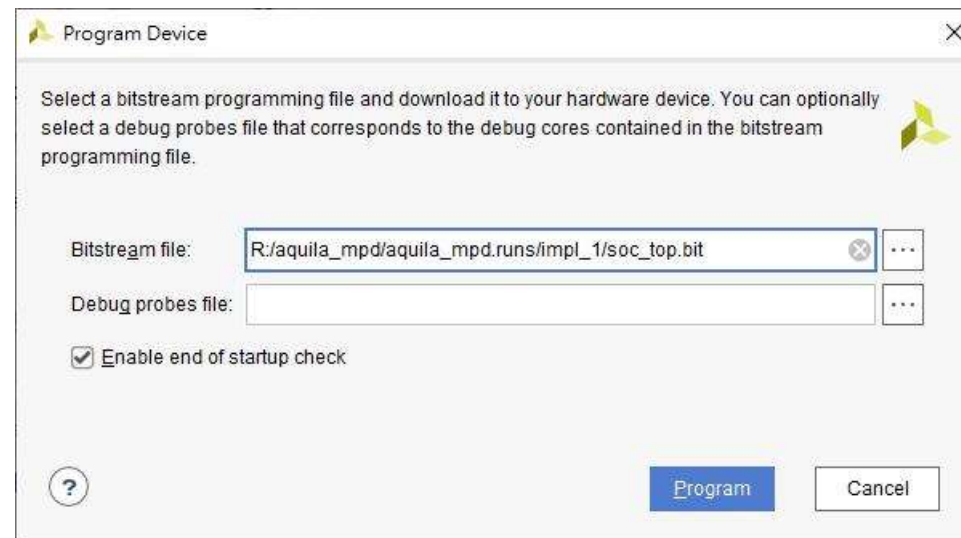
Type a Tcl command here

After power on the Arty board, click this to program the circuit into the FPGA.

But before you program it, you should have a terminal window connects to the correct UART port first.

Select the BIT File for Programming

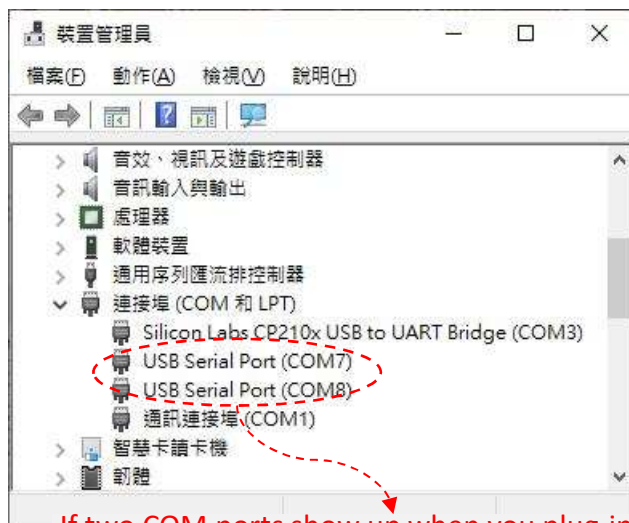
- ❑ If circuit synthesis is done, a `soc_top.bit` file will be under `aquila_mpd/aquila_mpd.runs/impl_1/`:
 - Usually, Vivado will automatically select the file for you, but occasionally, the file browser pop up empty



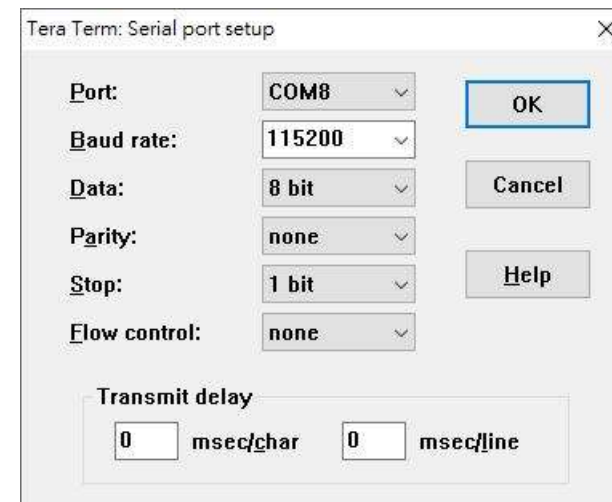
Terminal Settings (1/2)

- ❑ 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
```

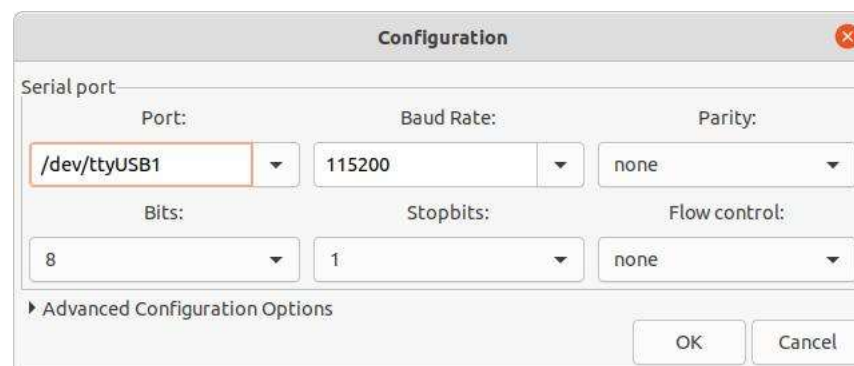
Arty JTAG programming device.

ttyUSB0

ttyUSB1

Arty serial port device.

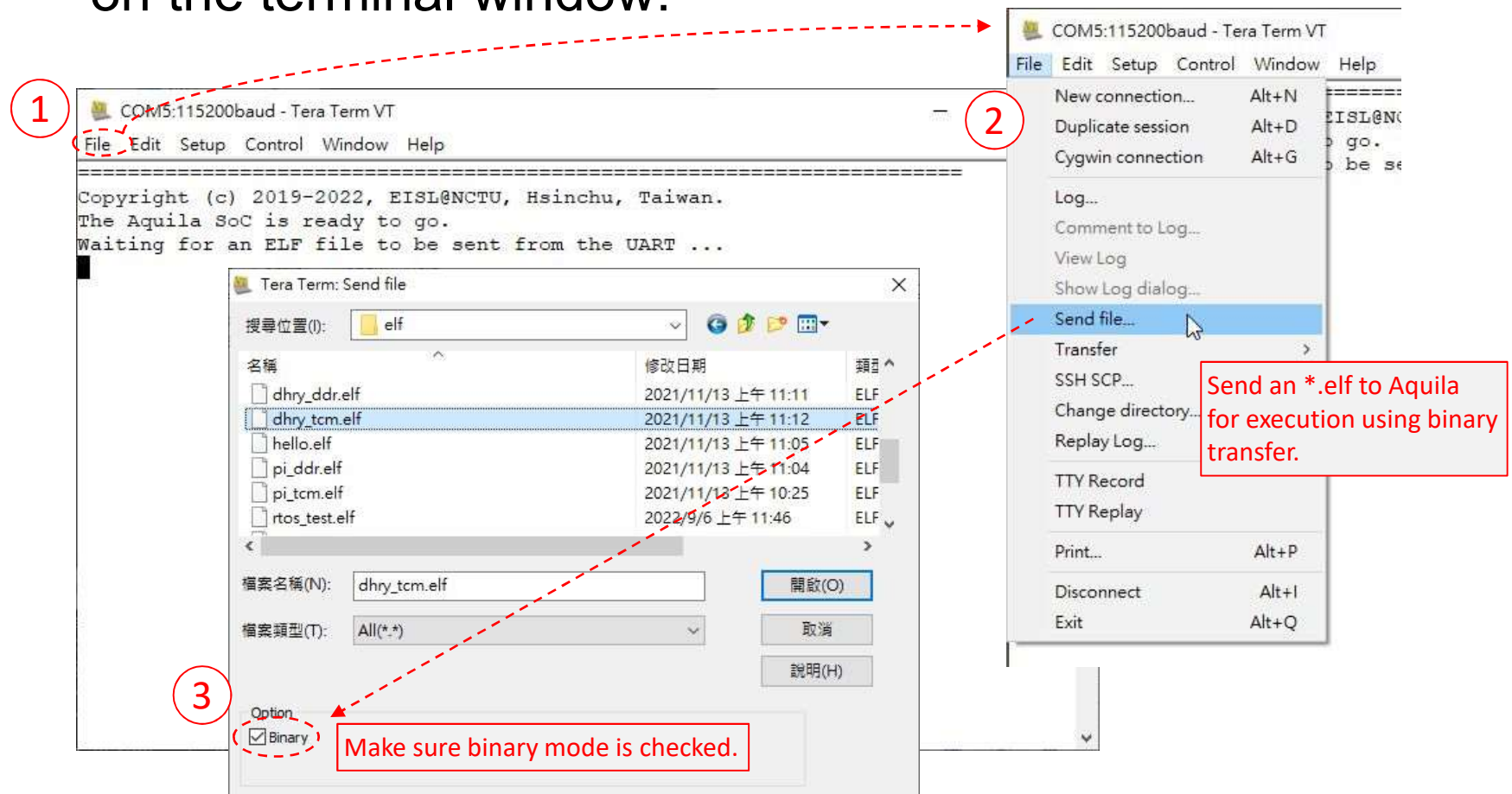
- ❑ GTKTerm configuration:



Run Applications on Real Hardware

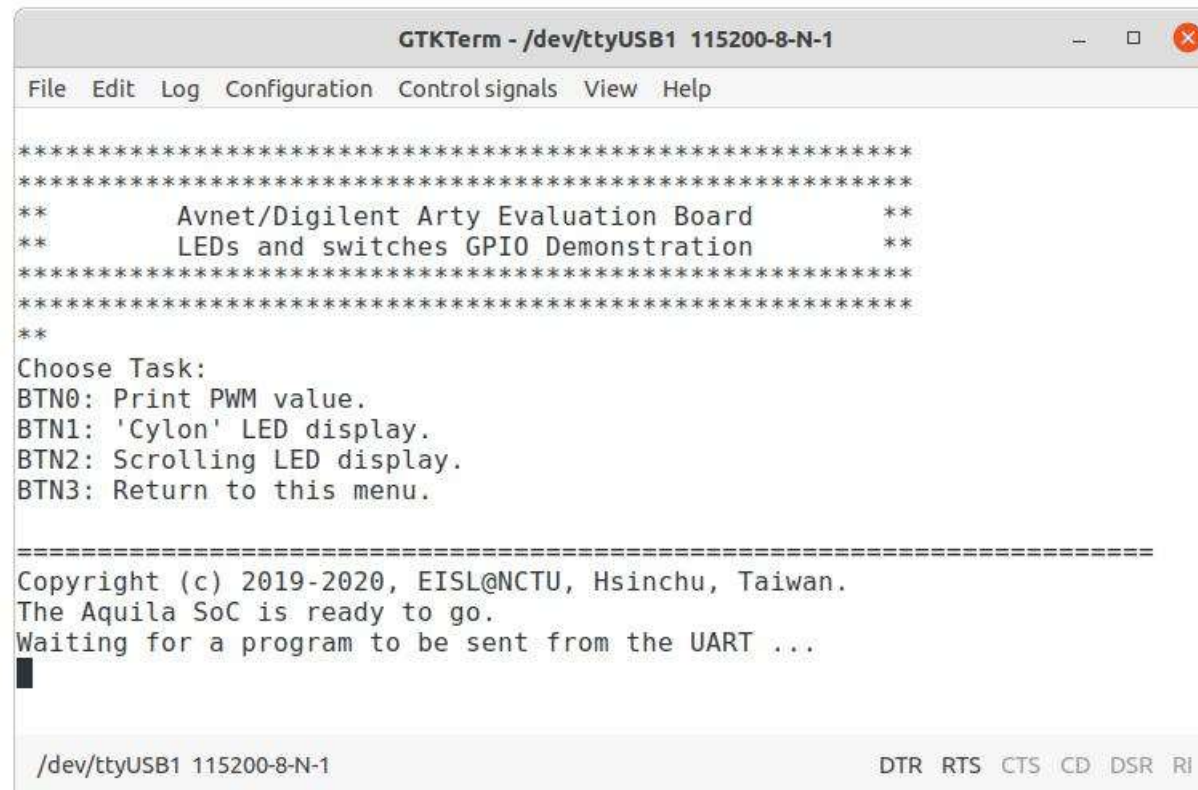
Run an Executable on Windows

- ❑ Once FPGA is programmed, a message is displayed on the terminal window:



Run an Executable on Linux

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



```
GTKTerm - /dev/ttyUSB1 115200-8-N-1
File Edit Log Configuration Controlsignals View Help

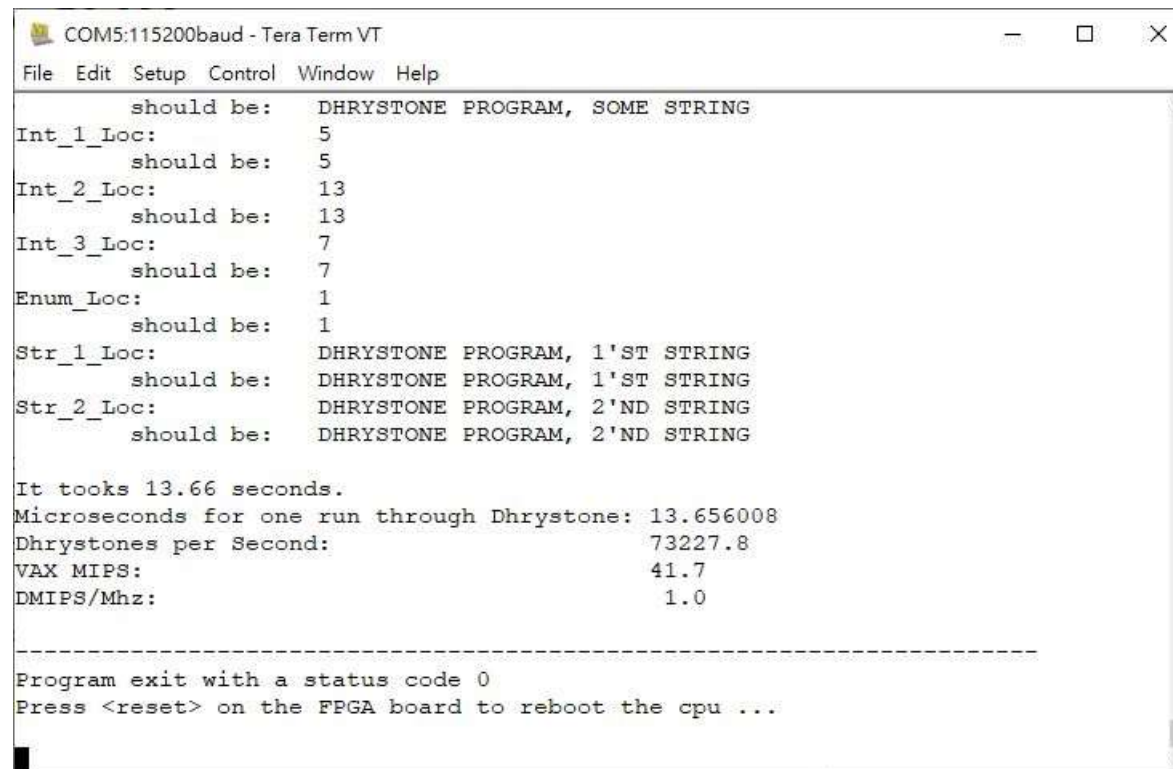
*****
*****
**      Avnet/Digilent Arty Evaluation Board      **
**      LEDs and switches GPIO Demonstration      **
*****
*****
**
Choose Task:
BTN0: Print PWM value.
BTN1: 'Cylon' LED display.
BTN2: Scrolling LED display.
BTN3: Return to this menu.

=====
Copyright (c) 2019-2020, EISL@NCTU, Hsinchu, Taiwan.
The Aquila SoC is ready to go.
Waiting for a program to be sent from the UART ...
█

/dev/ttyUSB1 115200-8-N-1      DTR RTS CTS CD DSR RI
```

Dhrystone Result

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



The screenshot shows a Tera Term VT window titled "COM5:115200baud - Tera Term VT". The window contains the following text:

```
File Edit Setup Control Window Help
      should be:  DHRYSTONE PROGRAM, SOME STRING
Int_1_Loc:       5
      should be:  5
Int_2_Loc:      13
      should be:  13
Int_3_Loc:       7
      should be:  7
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 13.66 seconds.
Microseconds for one run through Dhrystone: 13.656008
Dhrystones per Second:                      73227.8
VAX MIPS:                                   41.7
DMIPS/Mhz:                                  1.0

-----
Program exit with a status code 0
Press <reset> on the FPGA board to reboot the cpu ...
```

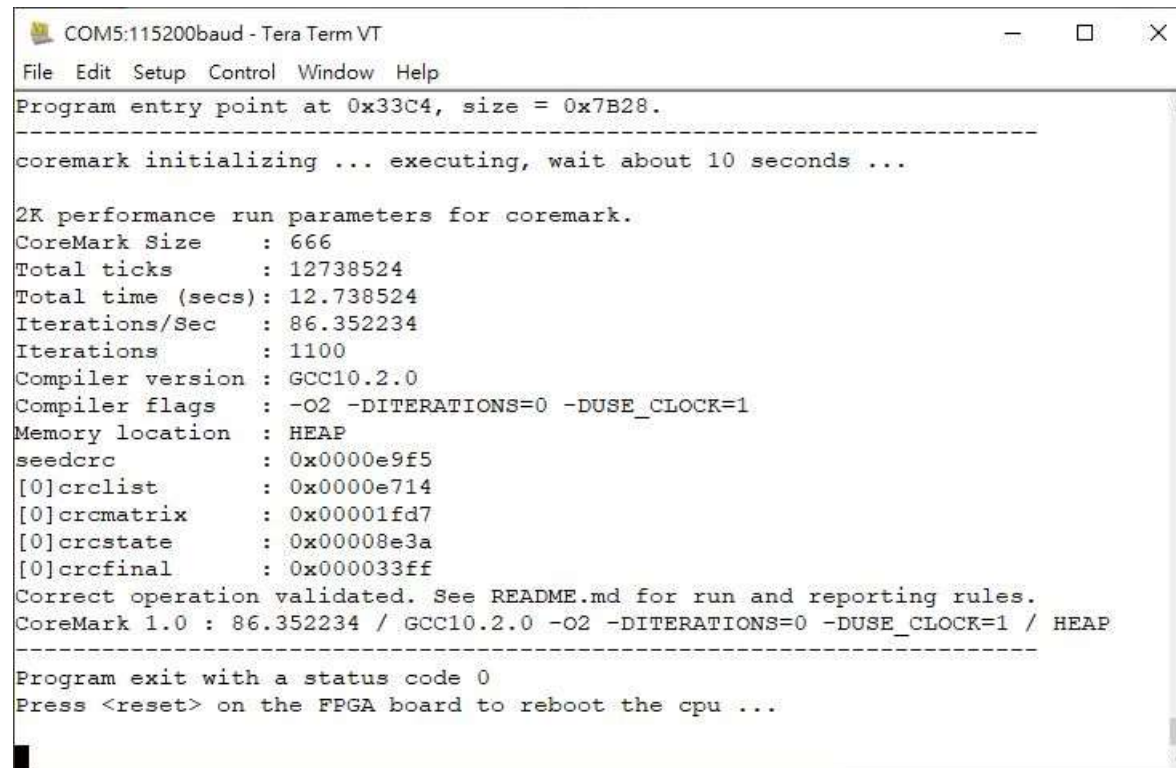
Analyze the Execution of Aquila SoC

- ❑ To analyze the behavior of Aquila, you can use a RTL simulator or the Integrated Logic Analyzer (ILA)
- ❑ The problem with a RTL simulator
 - Simulation of the execution of a program is very slow
 - On-chip memory must be initialized with the program
- ❑ 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

Profiling the Target Benchmark

CoreMark

- ❑ For this homework, we will use CoreMark as the target application:



```
COM5:115200baud - Tera Term VT
File Edit Setup Control Window Help
Program entry point at 0x33C4, size = 0x7B28.
-----
coremark initializing ... executing, wait about 10 seconds ...

2K performance run parameters for coremark.
CoreMark Size      : 666
Total ticks        : 12738524
Total time (secs)  : 12.738524
Iterations/Sec     : 86.352234
Iterations         : 1100
Compiler version   : GCC10.2.0
Compiler flags     : -O2 -DITERATIONS=0 -DUSE_CLOCK=1
Memory location    : HEAP
seedcrc            : 0x0000e9f5
[0]crclist         : 0x0000e714
[0]crcmatrix       : 0x00001fd7
[0]crcstate        : 0x00008e3a
[0]crcfinal        : 0x000033ff
Correct operation validated. See README.md for run and reporting rules.
CoreMark 1.0 : 86.352234 / GCC10.2.0 -O2 -DITERATIONS=0 -DUSE_CLOCK=1 / HEAP
-----
Program exit with a status code 0
Press <reset> on the FPGA board to reboot the cpu ...
```


Profiling a Program for Optimization

- ❑ Often, 80% of the program execution time resides in 20% of the code → **how do you find the hotspots?**
- ❑ An easy way is to use a software profiler to do:
 - Sampling-based hot spot analysis
 - Insert a timer interrupt service routine (ISR) into your program
 - Interrupt the processor at a fixed frequency, say, 100 Hz, and the ISR collects samples of the PC during execution
 - The PC samples are used to estimate the time spent in a function
 - Counter-based call analysis
 - Insert counter code into every function
 - Record the caller and calling frequency

prof.

Example: Profiling on PC

- ❑ In real Linux, GCC can be used to profile a program:

```
$ gcc -O2 -pg my_prog.c -o my_prof  
$ ./my_prog  
$ gprof ./my_prog gmon.out > profile.txt
```

- A program compiled with the `-pg` flag will have profiling code inserted into the program
- When executed, the profiling code will collect runtime information and store it to the binary file `gmon.out`
- The command `gprof` can convert `gmon.out` to a text file
- Note that under Windows Subsystem for Linux 1.0, `gprof` only provide call analysis, not hotspot analysis

The Profile of CoreMark

- ❑ On Linux64 PC, `profile.txt` looks like:

```
Flat profile:
Each sample counts as 0.01 seconds.
   %   cumulative   self           self      total
time  seconds  seconds   calls   us/call   us/call   name
25.75    2.71    2.71    1022220   2.66    10.14   core_bench_list
25.37    5.39    2.67   105288660   0.03     0.03   core_list_find
11.21    6.57    1.18    2044440   0.58     0.58   matrix_mul_matrix_bitextract
. . . . .

Call graph (explanation follows)
granularity: each sample hit covers 2 byte(s) for 0.09% of 10.55 seconds
index % time    self  children    called          name
<spontaneous>
[1]    98.5    0.02   10.37
          2.71    7.65 1022220/1022220        iterate [1]
          0.01    0.00 1022220/149244120      core_bench_list [2]
                                     crcul6 [12]
. . . . .
```

- ❑ You can use `gprof2dot.py` + `graphviz`,
to draw a call-graph from `profile.txt`:

```
$ gprof2dot.py profile.txt | dot -Tsvg -o coremark.svg
```

Visualization of Call-Graph

Hotspots for CoreMark:

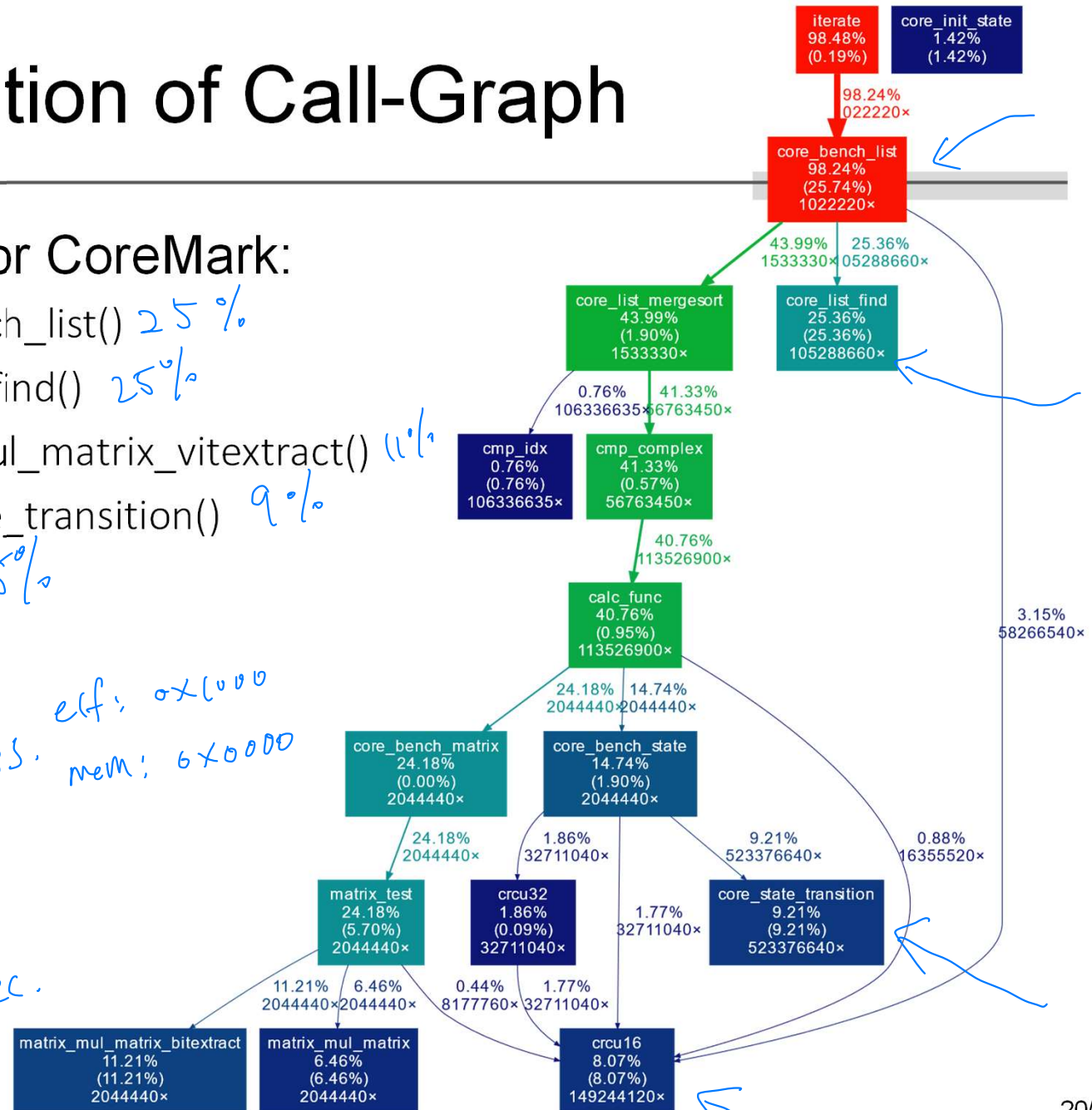
- core_bench_list() 25%
- core_list_find() 25%
- matrix_mul_matrix_vitextract() 11%
- core_state_transition() 9%
- crcu16() 8%

CPU cycles
CPU ratio

mem → elf.
1. start address. elf: 0x6000 mem: 6x0000

2. Mallocfile

3. Coremark #10 sec.



Limitations of Software Profiler

- ❑ May not be available on the target platform
 - GCC+Newlib has no built-in profiling facility
- ❑ Timer interrupts can be obtrusive to the program, especially when the program has its own ISR
 - Storing profiling data in memory can be expensive
- ❑ Recursive functions can not be profiled properly
 - Need better granularity
- ❑ Do not differentiate between computations and memory/device accesses

Profiling Using CPU Hardware

- ❑ Since we can modify the hardware code of the CPU, can we design a CPU that profile a program during execution automatically?
- ❑ To count the execution cycles inside a function we can design a hardware counter that check if PC is in a function every clock cycles
 - The starting address and the length of a function is in the *.map file from the compiler
 - We can further analyze that, for a particular function, the ratio of computation cycles versus memory cycles
 - Note, you must take into account the stall cycles

Your Homework

- ❑ Add HW code to Aquila such that it can collect the runtime profiling data for the five hotspots in page 20
 - You should also count the total CPU cycles and compute the CPU ratio of each function
 - You should calculate the ratio of computation versus memory cycles for each function

- ❑ Write a 2-page double-column report[†]:
 - Discuss what you have done to collect runtime statistics
 - How do you input the function address data into Aquila SoC?
 - What you have learned based on the runtime statistics you have collected

[†] A report template can be downloaded from E3 website.

Comments on Report Grading

- ❑ Your report will be graded using the following points:
 - Organization and writing style (25%)
 - The design of your profiling mechanism (25%)
 - Discussions on the profiling results (30%)
 - Comparison of the result to that on a PC
 - What you can say about the computation vs. memory cycles
 - What you can say about the stall cycles
 - Discussions on how to improve Aquila (20%)

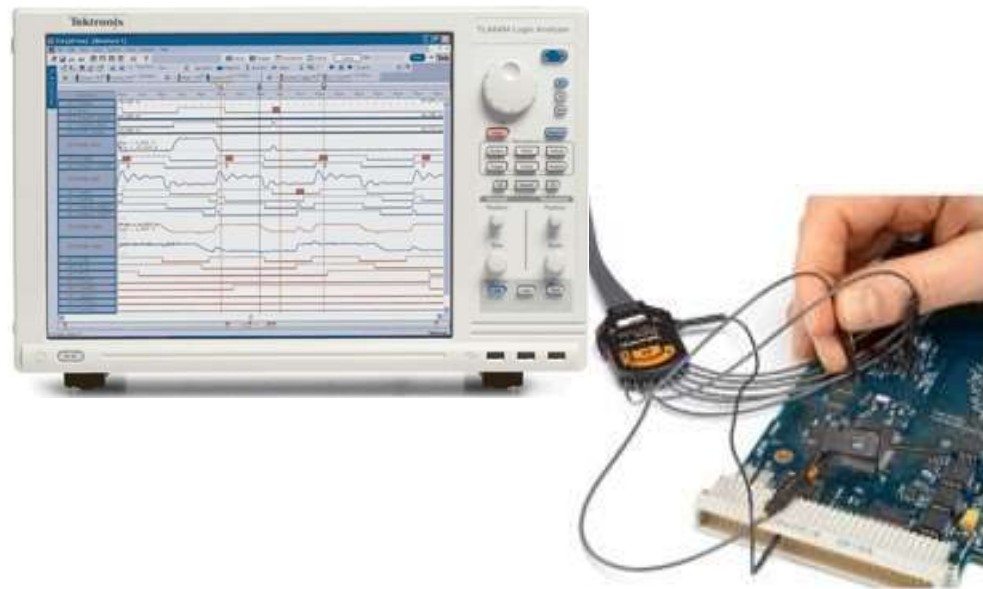
Appendix: ILA for Real-Time Circuit Debugging

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

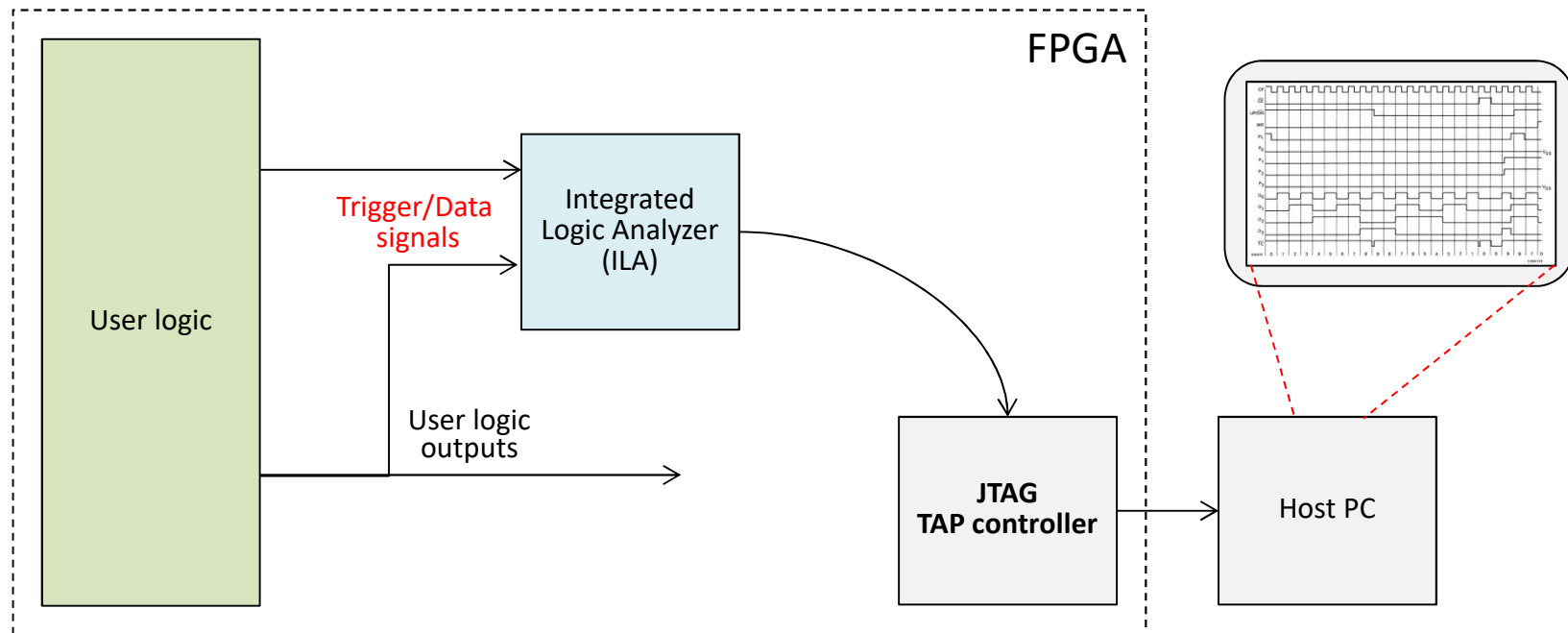
Simulation pass but.
realtime doesn't.

1. timing violation
2. multidrive-net.
3. latch



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 so that ILA can 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, **unless the signal is void**
 - Vivado will insert an ILA IP into the synthesized design to monitor and capture these signals at runtime

Mark Debug Signals Using GUI

- ❑ To debug a circuit, open the synthesized design:

The screenshot shows the Vivado 2022.1 interface. The 'SYNTHESIZED DESIGN - synth_1 | xc7a100tcs324-1' window is active. The 'Flow Navigator' on the left shows the 'SYNTHESIS' section with 'Open Synthesized Design' highlighted. The 'Sources' pane shows the 'data_sel_r' net selected. The 'Context Menu' for 'data_sel_r' is open, with 'Mark Debug' highlighted. A red dashed circle highlights the 'Mark Debug' option. A red arrow points from the 'Mark Debug' option to the 'Debug' window. The 'Debug' window is open, showing a list of debug nets. A red dashed circle highlights the 'Debug' window. A red arrow points from the 'Debug' window to the 'Debug' window. The 'Debug' window shows a table of debug nets.

You can also select signals to be debugged from here!

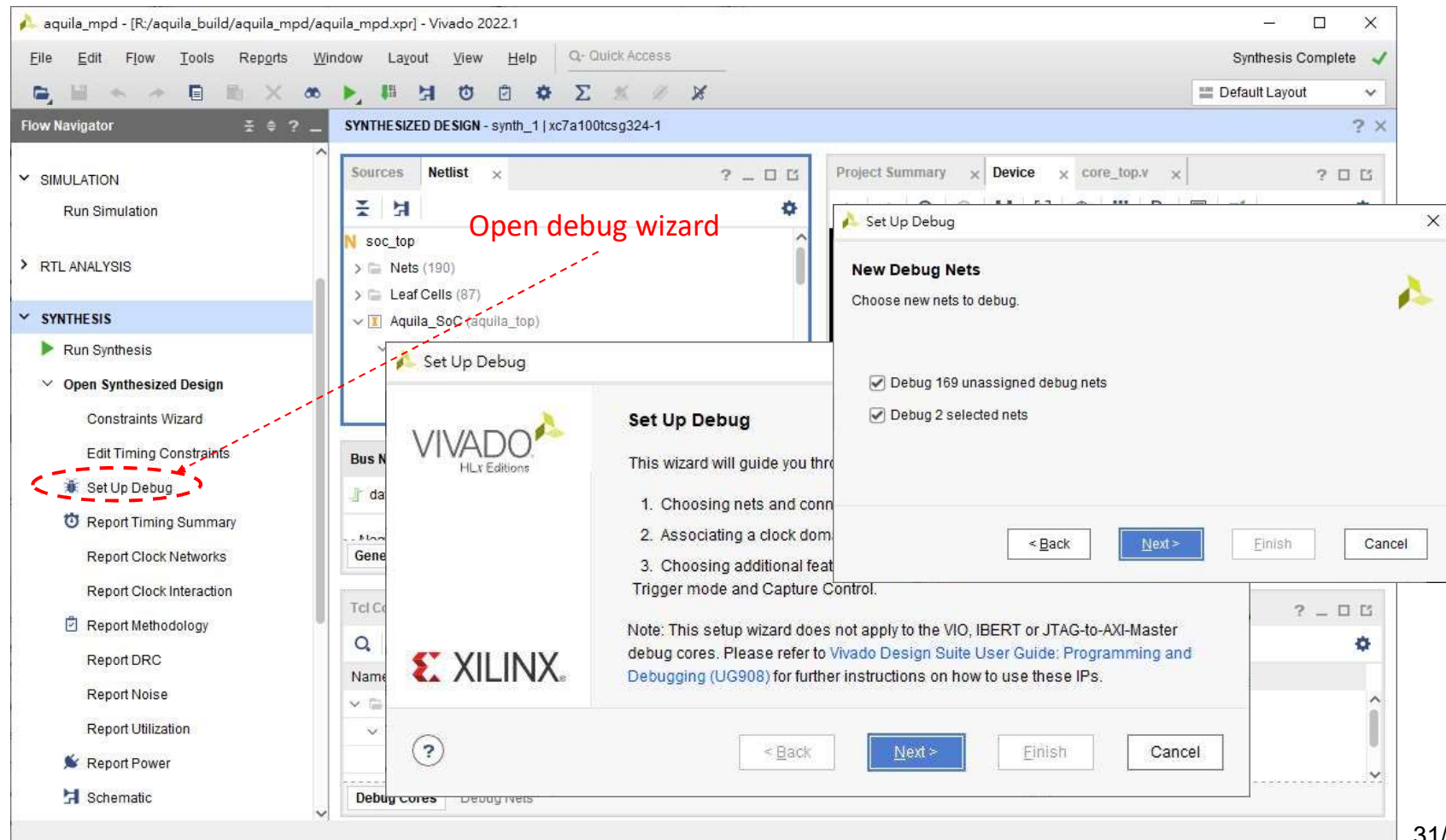
open debug window from the menu bar

This windows shows signals to be debugged!

Name	Driver Cell	Driver Pin	Probe Type
Unassigned Debug Nets (169)			
Aquila_SoC/RISCV_CORE0/code_addr_o (19)	FDRE	Q	
Aquila_SoC/RISCV_CORE0/code_addr_o[2]	FDRE	Q	

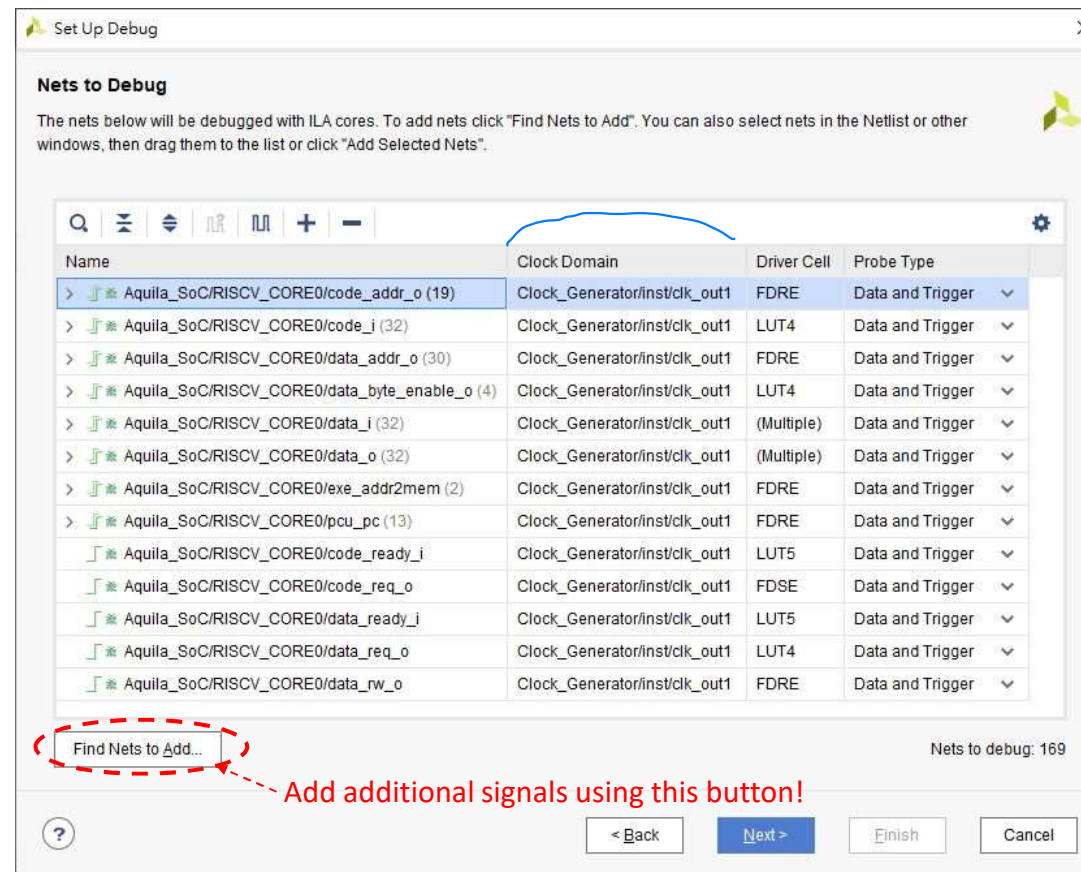
Set Up the Debug Wizard

- ❑ Open the “Set Up Debug” wizard:



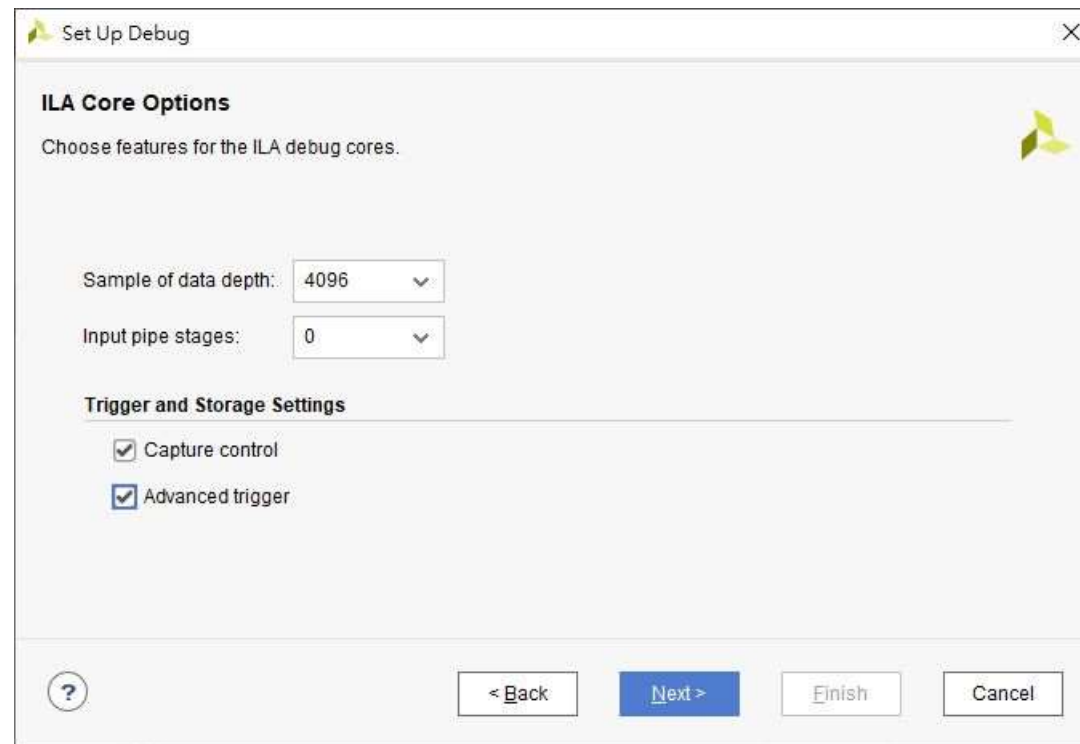
Double-Check Nets to Be Debugged

- ❑ You can add any missing signals in this dialog box
 - Some signals in your Verilog code may be optimized out!



Modify Trigger Options

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



Set Up Debug

ILA Core Options

Choose features for the ILA debug cores.

Sample of data depth: 4096

Input pipe stages: 0

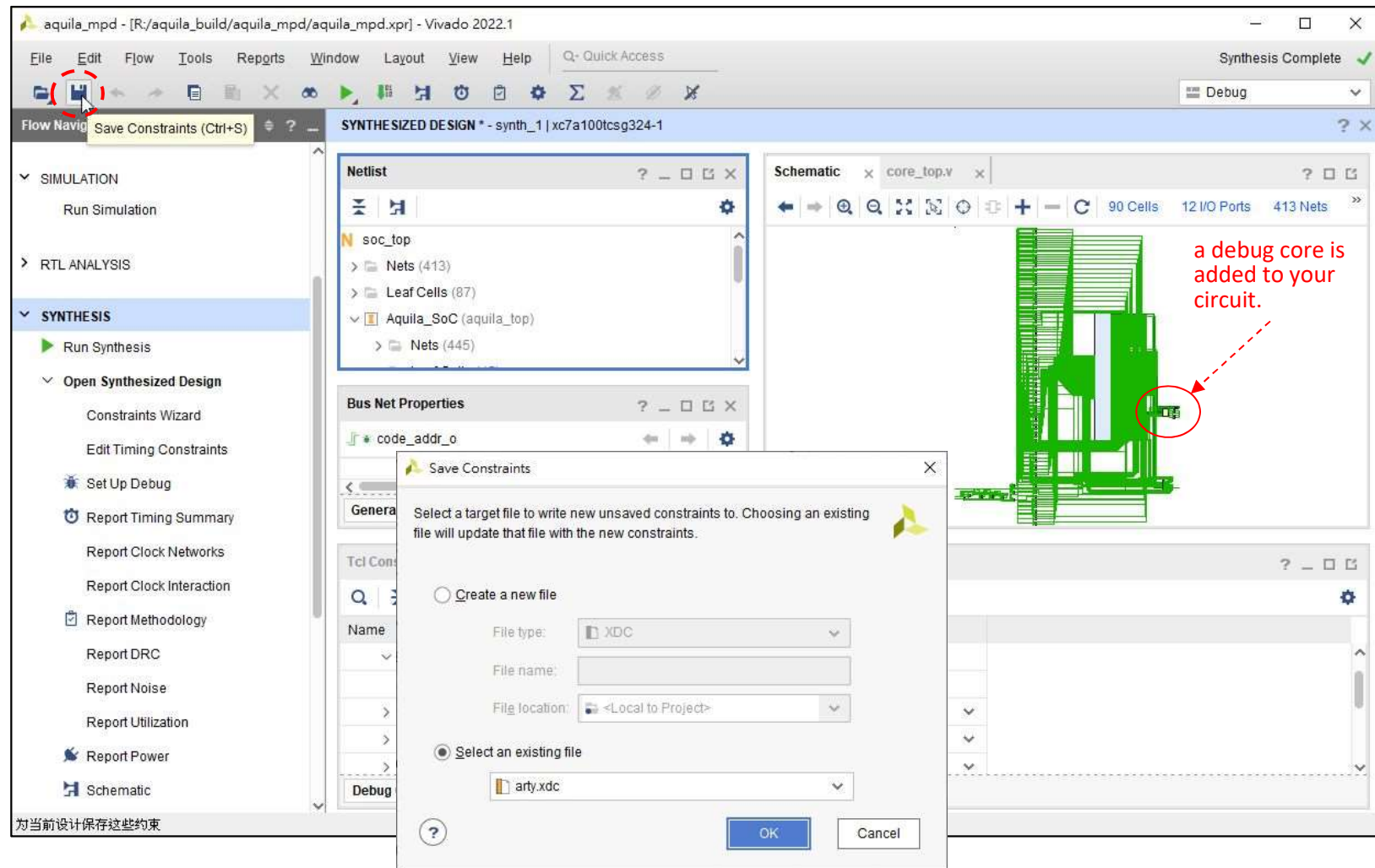
Trigger and Storage Settings

☒ Capture control

☒ Advanced trigger

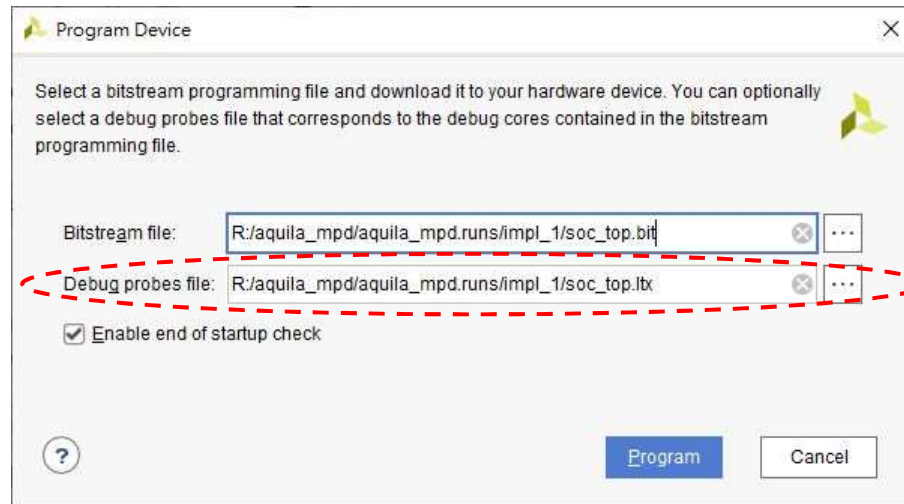
? < Back Next > Finish Cancel

Save the New Debug Constraints



Program the FPGA for ILA

- ❑ After generating the bit file, we can program the FPGA
- ❑ Once you hit the “program device” menu item, you will see that an extra ILA configuration file is selected:



The Hardware Manager with ILA View

The screenshot displays the Vivado 2022.1 Hardware Manager interface. The top menu bar includes File, Edit, Flow, Tools, Reports, Window, Layout, View, and Help. The toolbar shows various icons for file operations and execution. The main window is titled "HARDWARE MANAGER - localhost:xilinx_tcf/Digilent/210319B26B02A".

On the left, the Flow Navigator pane shows the project hierarchy, including "Set Up Design", "Report Timing", "Report Clocks", "Report Constraints", "Report Messages", "Report Design", "Report Utilization", "Report Power", and "Schematic". The "IMPLEMENTATION" section is expanded, showing "Run Implementation", "Open Implementation", and "Generate Bits". The "PROGRAM AND DEPLOY" section is also expanded, showing "Open Hardware", "Open Target", "Program", and "Add Configuration".

The central pane is divided into several sections:

- Hardware:** A table listing hardware components. The table has columns "Name" and "Status". The components listed are "localhost (1)" (Connected), "xilinx_tcf/Digilent/210319B" (Open), "xc7a100t_0 (2)" (Programmed), and "XADC (System Monitor)".
- Hardware Device Properties:** A section for the selected device "xc7a100t_0". It shows the Name, Part, and ID code (13631093).
- Waveform - hw_ila_1:** A section for the selected ILA core. It shows the ILA Status (Idle) and a table of signals. The table has columns "Name" and "Value". The signals listed are "Aquila_S..._1[16:2]" (0000) and "Aquila_So..._0[31:28]" (0). The waveform window shows a signal trace with markers 0, 1, 2, 3, and 4.
- Settings - hw_ila_1:** A section for the selected ILA core. It shows the Core status (Idle) and the Capture status (Window 1 of 1).
- Trigger Setup - hw_ila_1:** A section for the selected ILA core. It shows the Trigger Setup and Capture Setup.

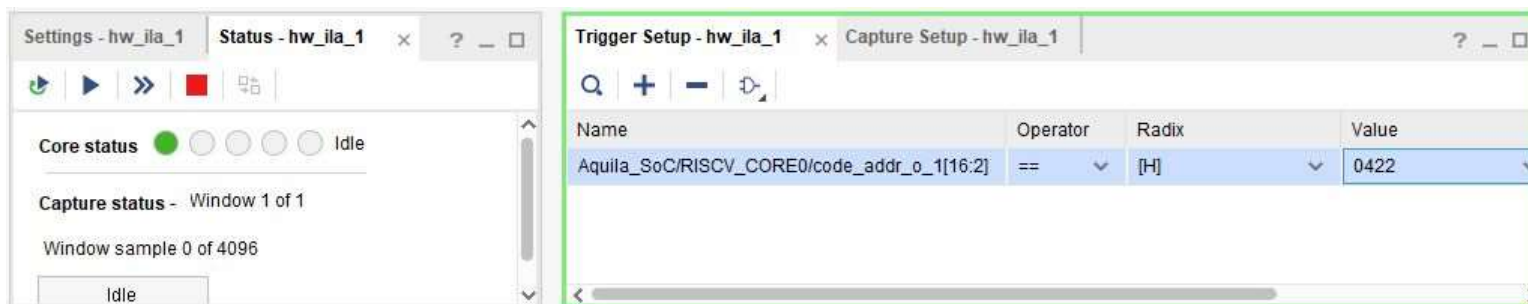
Red dashed arrows and text labels point to specific features:

- "Each clock domain has one ILA tab." points to the "hw_ila_1" tab in the Waveform window.
- "Runtime waveform window." points to the waveform display area.
- "Trigger setup window" points to the Trigger Setup section.
- "ILA configuration window" points to the Settings - hw_ila_1 section.
- "Signals which you can capture" points to the table of signals in the Waveform window.

The bottom pane is the Tcl Console, showing the command "program_hw_devices [get_hw_devices xc7a100t_0]" and the output "INFO: ILAtools 27-31641 End of startup status: HIGH".

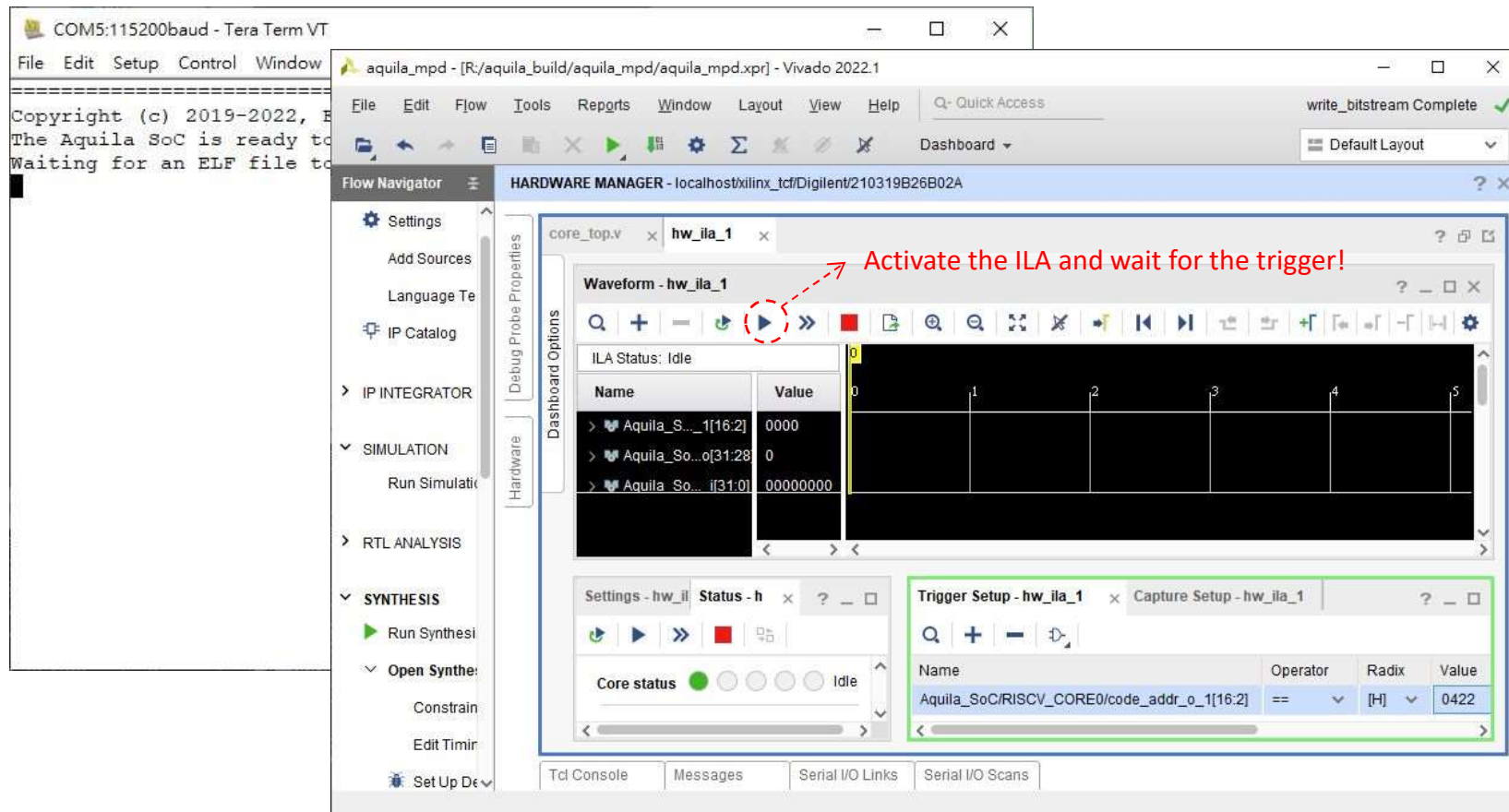
Setting a Trigger

- ❑ A trigger is a signal condition that tells the ILA to begin capturing waveforms
- ❑ Set the trigger condition
 - The instruction address is `code_addr_o[31:0]` in `core_top.v`
 - If the `main()` is at `0x1088`, we trigger the ILA when `code_addr_o[16:2]` equals `0x422`.



Capturing the Signals

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



Analyze the Captured Waveform

aquila_mpd - [R:/aquila_build/aquila_mpd/aquila_mpd.xpr] - Vivado 2022.1

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

Dashboard ▾ Default Layout ▾

Flow Navigator

HARDWARE MANAGER - localhost/xilinx_tcf/Digilent/210319B26B02A

core_top.v x hw_ila_1 x

Waveform - hw_ila_1

ILA Status: Idle

Name	Value
Aquila_SoC/RISCV...de_addr_o_1[16:2]	0422
Aquila_SoC/RISCV...de_addr_o[31:28]	0
Aquila_SoC/RISCV_CORE0/code_i[31:0]	95c2a103
Aquila_SoC/RISCV...yte_enable_o[3:0]	3
Aquila_SoC/RISCV...data_addr_o[31:2]	3fff72b
Aquila_SoC/RISCV...RE0/pcu_pc_1[1:0]	0
Aquila_SoC/RISCV...RE0/pcu_pc_0[1:0]	0

Updated at: 2022-Oct-05 18:28:10

Settings - hw_ila_1 Status - hw_ila_1

Core status: Idle

Capture status - Window 1 of 1

Trigger Setup - hw_ila_1 Capture Setup - hw_ila_1

Name	Operator	Radix	Value
Aquila_SoC/RISCV_CORE0/code_addr_o_1[16:2]	==	[H]	0422

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

Store a Trigger Value in Registers

- ❑ In ILA, you can see waveforms of registers easily, but not so easy to check C variables
- ❑ You can use inline assembly code to store a variable into a register so the ILA can display its value:
 - Can be used to **set a trigger point!**
 - Note, the local variable is assumed to be stored in a register

```
int var1 = 123;

void main()
{
    int var2 = 456;

    /* Save a global variable to register t1 */
    asm volatile ("lui t0, %hi(var1)");
    asm volatile ("lw  t1, %lo(var1)(t0)");

    /* Save a local variable to register t1 */
    asm volatile ("addi t1, %1, 0" : "=r"(var2) : "r"(var2));
}
```