

---

# USING VIO AND ILA IN VIVADO

Stefan Johannesson

---

---

# OBJECTIVES OF LAB

- Go over common headaches with Vivado
  - Learn about VIO and ILA in Vivado
  - Modify 4-bit adder project to leverage VIOs and ILAs
  - Learn about ILA triggers
-

---

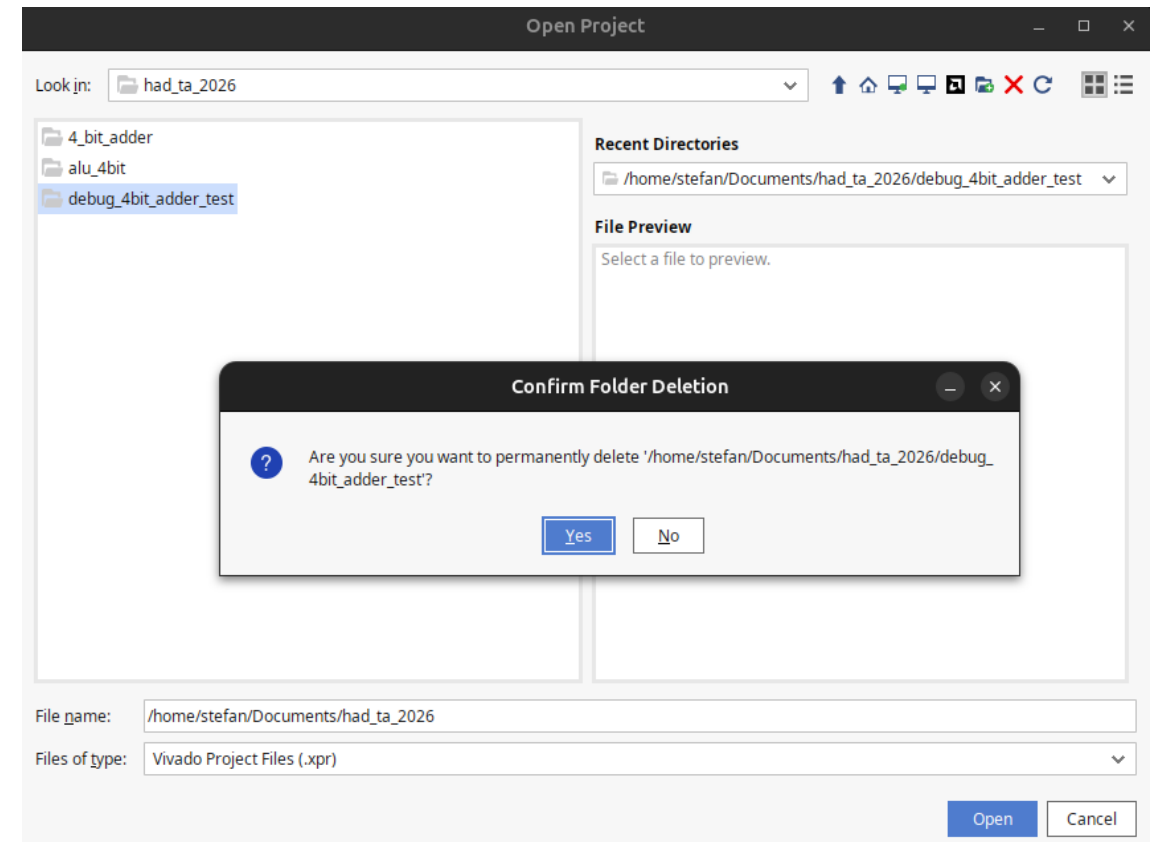
# HEADACHE 1: NAMING CONVENTION

- Vivado has a very strict naming scheme. Only allowed letters:
  - Alphanumeric characters (A-Z, a-z, 0-9)
  - Underscores (\_)
- Even folder names is NOT exempt from naming scheme
  - Saving under "OneDrive – YOUR\_NAME\...\..." will cause same issue
  - Save locally, like "C:\Documents", or "C:\Desktop"



# HEADACHE 2: PHANTOM PROJECTS

- When deleting project folders using a file explorer, **Vivado will still see the 'deleted' project**
- How to solve?
  - Manually delete the projects **within Vivado**.
  - Under File -> Project -> Open, you should have a pop-up window
  - Within this window, delete the project folder you want removed.



---

# WHAT IS VIO AND ILA?

- **VIO** - Virtual Input/Output
  - Virtual control panel of FPGA on your computer
  - Virtually drive signals inside FPGA via USB
- **ILA** - Integrated Logic Analyzer
  - Digital Oscilloscope built directly into FPGA logic
  - Records internal signals at full system speed (100 MHz for Basys 3)

---

# PRELIMINARY: RE-IMPORT 4-BIT ADDER

- Make a new Vivado project called "debug\_4bit\_adder"
- Import the 4 bit adder files from KEATS
- Extract the ZIP file to access individual Verilog files

The screenshot shows a sidebar titled "Labs and Coursework" with a list of items. The items are:

- Vivado Install Instruction
- HAD26LAB Student Handouts
- Lab Q&A Forum (with a note: "You can ask questions regarding to the lab here.")
- Lab 1 Slides** (highlighted with a red border)
- Lab 1 Recording
- Lab 1 4-bit Full adder demo code
- CW Task 1 Template Project: ALU and Shifter Design (with a "Completion" dropdown)
- Coursework Task 2 Instruction: Building Datapath
- CW Task 2 Template Project: Datapath Design (with a "Completion" dropdown)

The screenshot displays the AMD Vivado ML Edition interface. A central dialog box titled "Add Sources" is open, guiding the user through adding and creating sources for the project. The dialog contains three radio button options: "Add or create constraints", "Add or create design sources" (which is selected and underlined in red), and "Add or create simulation sources".

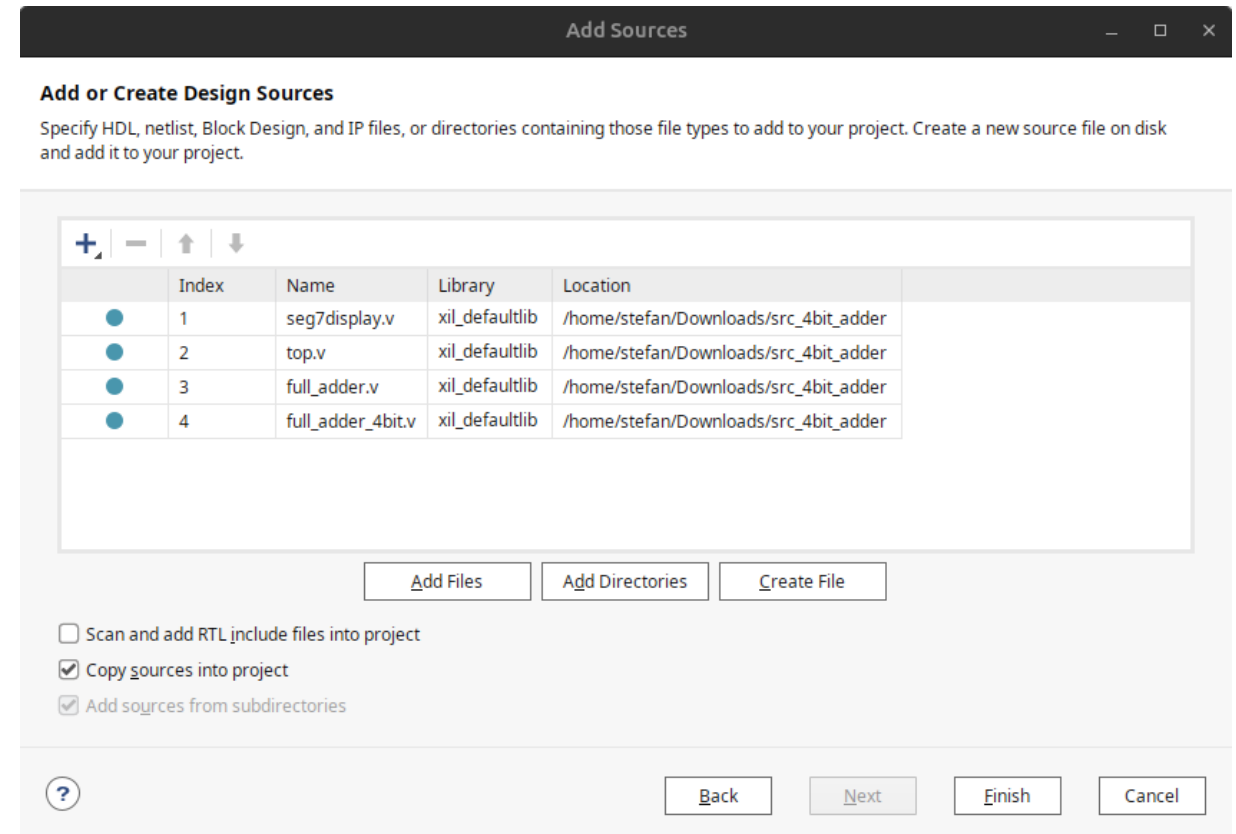
In the background, the Project Manager window shows a tree view of sources: Design Sources, Constraints, Simulation Sources (containing sim\_1), and Utility Sources. A red box highlights a "+" icon in the Sources toolbar. The Project Summary window shows the project name "debug\_4bit\_adder\_test2" and its location. At the bottom, a table displays project synthesis and implementation details.

Name	Constraints	Status	WNS	TNS	WHS	THS	WBSS	TPWS	Total Power	Failed Routes	Methodology	RQA Score	QoR Suggestions	LUT	FF	BRAM	URAM	DSP	Start	Elapsed	Run Strategy	
synth_1	constrs_1	Not started																				Vivado Synthesis Defaults (Vivado Synthesis 2024)
impl_1	constrs_1	Not started																				Vivado Implementation Defaults (Vivado Implementatio

Specify and/or create source files to add to the project

# RE-IMPORT 4-BIT ADDER

- Import your Verilog files from the extracted ZIP file
  - You can omit importing the testbench
- Make sure that "Copy sources into project" is selected (makes file management easier for your coursework!)
- Useful tip: if you have done Lab 1 as a separate project, can directly import constraints file into current project (will show how!)



- Project Hierarchy should now look like this

The screenshot displays the Xilinx Vivado Project Manager interface for a project named 'debug\_4bit\_adder\_test2'. The 'Sources' pane shows a hierarchical tree structure:

- Design Sources (1)
  - top (top.v) (2)
    - seg7display (seg7display.v)
    - full\_adder\_4bit (full\_adder\_4bit.v) (4)
      - full\_adder (full\_adder.v)
      - full\_adder (full\_adder.v)
      - full\_adder (full\_adder.v)
      - full\_adder (full\_adder.v)
- Constraints
  - constrs\_1
- Simulation Sources (1)
  - sim\_1 (1)
    - top (top.v) (2)
- Utility Sources
  - utils\_1

The 'Project Summary' pane provides details for the project:

- Overview | Dashboard**
- Settings | Edit**
- Project name: debug\_4bit\_adder\_test2
- Project location: /home/stefan/Documents/had\_ta\_2026/debug\_4bit\_adder\_test2
- Product family: Artix-7
- Project part: Basys3 (xc7a35tqpg236-1)
- Top module name: Not defined
- Target language: Verilog
- Simulator language: Mixed
- Target Simulator: Vivado Simulator
- Board Part**
- Display name: Basys3
- Board part name: digilentinc.com:basys3:part0:1.2
- Board revision: C.0
- Connectors: No connections
- Repository path: /home/stefan/.Xilinx/Vivado/2024.2/xhub/board\_store/xilinx\_board\_store
- URL: <https://digilent.com/reference/programmable-logic/basys-3/start>
- Board overview: Basys3
- Synthesis | Implementation**
- Status: Not started
- Messages: No errors or warnings
- Part: vr7a35trnn236-1

The 'Design Runs' table at the bottom shows the following data:

Name	Constraints	Status	WNS	TNS	WHS	THS	WBSS	TPWS	Total Power	Failed Routes	Methodology	RQA Score	QoR Suggestions	LUT	FF	BRAM	URAM	DS
synth_1	constrs_1	Not started																
impl_1	constrs_1	Not started																

- Possible to repeat steps in Lab 1 again to define pinouts and voltages...
- ... but if you already did Lab 1, you can simply use the same constraints file!

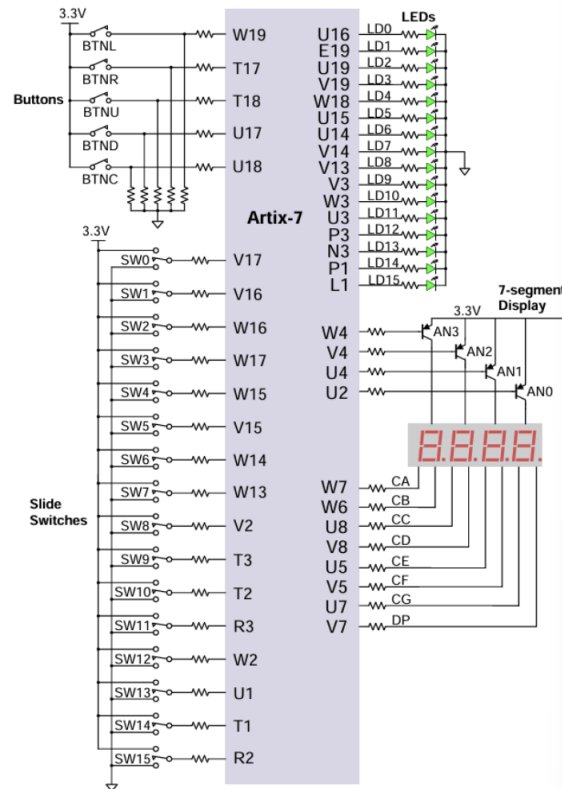


Figure 16. General purpose I/O devices on the Basys3.

The screenshot shows the Xilinx IDE interface. The 'IO Planning' tool is active, displaying a grid of the FPGA package pins. The 'IO Ports' table is visible, showing the configuration for the AN (AND) and led (LED) devices.

Name	Direction	Neg Diff Pair	Package Pin	Fixed	Bank	IO Std	Vcco	Vref	Drive Strength	Slew Type	Pull Type	Off-Chip Termination
AN (4)	OUT		W4	<input checked="" type="checkbox"/>	34	LVCMOS33*	3.300	12			NONE	FP_VTT_50
AN[3]	OUT		V4	<input checked="" type="checkbox"/>	34	LVCMOS33*	3.300	12			NONE	FP_VTT_50
AN[2]	OUT		U4	<input checked="" type="checkbox"/>	34	LVCMOS33*	3.300	12			NONE	FP_VTT_50
AN[1]	OUT		U2	<input checked="" type="checkbox"/>	34	LVCMOS33*	3.300	12			NONE	FP_VTT_50
AN[0]	OUT			<input checked="" type="checkbox"/>	34	LVCMOS33*	3.300	12			NONE	FP_VTT_50
led (16)	OUT			<input type="checkbox"/>		default (LVCMOS18)	1.800	12			NONE	FP_VTT_50
led[15]	OUT			<input type="checkbox"/>		default (LVCMOS18)	1.800	12			NONE	FP_VTT_50
led[14]	OUT			<input type="checkbox"/>		default (LVCMOS18)	1.800	12			NONE	FP_VTT_50
led[13]	OUT			<input type="checkbox"/>		default (LVCMOS18)	1.800	12			NONE	FP_VTT_50
led[12]	OUT			<input type="checkbox"/>		default (LVCMOS18)	1.800	12			NONE	FP_VTT_50
led[11]	OUT			<input type="checkbox"/>		default (LVCMOS18)	1.800	12			NONE	FP_VTT_50
led[10]	OUT			<input type="checkbox"/>		default (LVCMOS18)	1.800	12			NONE	FP_VTT_50
led[9]	OUT			<input type="checkbox"/>		default (LVCMOS18)	1.800	12			NONE	FP_VTT_50
led[8]	OUT			<input type="checkbox"/>		default (LVCMOS18)	1.800	12			NONE	FP_VTT_50

The screenshot displays the Vivado IDE interface. The 'Add Sources' dialog box is the central focus, with the 'Add or create constraints' option selected. The background shows the 'Sources' panel with a project hierarchy, the 'Constraint Set Properties' panel for 'constrs\_1', and the 'Design Runs' table.

Name	Constraints	Status	WNS	TNS	WHS	THS	WBSS	TPWS	Total Power	Failed Routes	Methodology	RQA Score	QoR Suggestions	LUT	FF	BRAM	URAM	DS
synth_1	constrs_1	Not started																
impl_1	constrs_1	Not started																

Specify and/or create source files to add to the project

- 
- Go to where you saved Lab 1
  - Your constraints file should be saved under "`{PROJ_NAME}.srcs\constrs1\new\`"
  - Import it into you project

#### Add or Create Constraints

Specify or create constraint files for physical and timing constraint to add to your project.

Specify constraint set: constrs\_1 (active)

Constraint File	Location
constraints.xdc	/home/stefan/Documents/had_ta_2026/4_bit_adder/4_bit_adder.srcs/constrs_1/new

Copy constraints files into project

? Back Next Finish Cancel

# WHAT YOU SHOULD HAVE

The screenshot displays the Xilinx IDE interface for a project named "debug\_4bit\_adder\_test2". The interface is divided into several panes:

- Flow Navigator:** Shows the project hierarchy with sections for PROJECT MANAGER, IP INTEGRATOR, SIMULATION, RTL ANALYSIS, SYNTHESIS, IMPLEMENTATION, and PROGRAM AND DEBUG.
- Sources:** A tree view showing the project's source files, including Design Sources (top.v), Constraints (constrs\_1), Simulation Sources (sim\_1), and Utility Sources (utils\_1).
- Project Summary:** A window showing the contents of the constraints.xdc file, which lists various properties for LEDs and switches.
- Design Runs:** A table showing the status of the synthesis and implementation runs.

The constraints.xdc file contains the following properties:

```
1 set_property IOSTANDARD LVCMOS33 [get_ports {AN[3]}]
2 set_property IOSTANDARD LVCMOS33 [get_ports {AN[2]}]
3 set_property IOSTANDARD LVCMOS33 [get_ports {AN[1]}]
4 set_property IOSTANDARD LVCMOS33 [get_ports {AN[0]}]
5 set_property IOSTANDARD LVCMOS33 [get_ports {led[15]}]
6 set_property IOSTANDARD LVCMOS33 [get_ports {led[14]}]
7 set_property IOSTANDARD LVCMOS33 [get_ports {led[13]}]
8 set_property IOSTANDARD LVCMOS33 [get_ports {led[12]}]
9 set_property IOSTANDARD LVCMOS33 [get_ports {led[11]}]
10 set_property IOSTANDARD LVCMOS33 [get_ports {led[10]}]
11 set_property IOSTANDARD LVCMOS33 [get_ports {led[9]}]
12 set_property IOSTANDARD LVCMOS33 [get_ports {led[8]}]
13 set_property IOSTANDARD LVCMOS33 [get_ports {led[7]}]
14 set_property IOSTANDARD LVCMOS33 [get_ports {led[6]}]
15 set_property IOSTANDARD LVCMOS33 [get_ports {led[5]}]
16 set_property IOSTANDARD LVCMOS33 [get_ports {led[4]}]
17 set_property IOSTANDARD LVCMOS33 [get_ports {led[3]}]
18 set_property IOSTANDARD LVCMOS33 [get_ports {led[2]}]
19 set_property IOSTANDARD LVCMOS33 [get_ports {led[1]}]
20 set_property IOSTANDARD LVCMOS33 [get_ports {led[0]}]
21 set_property IOSTANDARD LVCMOS33 [get_ports {sw[15]}]
22 set_property IOSTANDARD LVCMOS33 [get_ports {sw[14]}]
23 set_property IOSTANDARD LVCMOS33 [get_ports {sw[13]}]
24 set_property IOSTANDARD LVCMOS33 [get_ports {sw[12]}]
25 set_property IOSTANDARD LVCMOS33 [get_ports {sw[11]}]
26 set_property IOSTANDARD LVCMOS33 [get_ports {sw[10]}]
27 set_property IOSTANDARD LVCMOS33 [get_ports {sw[9]}]
28 set_property IOSTANDARD LVCMOS33 [get_ports {sw[8]}]
29 set_property IOSTANDARD LVCMOS33 [get_ports {sw[7]}]
30 set_property IOSTANDARD LVCMOS33 [get_ports {sw[6]}]
31 set_property IOSTANDARD LVCMOS33 [get_ports {sw[5]}]
32 set_property IOSTANDARD LVCMOS33 [get_ports {sw[4]}]
33 set_property IOSTANDARD LVCMOS33 [get_ports {sw[3]}]
34 set_property IOSTANDARD LVCMOS33 [get_ports {sw[2]}]
35 set_property IOSTANDARD LVCMOS33 [get_ports {sw[1]}]
36 set_property IOSTANDARD LVCMOS33 [get_ports {sw[0]}]
37 create_clock -period 10.000 -name clk -waveform {0.000 5.000} [get_ports clk]
38
```

The Design Runs table shows the following data:

Name	Constraints	Status	WNS	TNS	WHS	THS	WBSS	TPWS	Total Power	Failed Routes	Methodology	RQA Score	QoR Suggestions	LUT	FF	BRAM	URAM	DS
synth_1	constrs_1	Not started																
impl_1	constrs_1	Not started																

---

# STEP 1: SETTING UP THE ILA

- We want to explicitly tell the ILA what specific wires in our design we want it to watch
- We can do this by adding the `(* mark_debug = "true" *)` attribute right before the wire declaration:
- In "top.v", replace:

```
//Wires and signals
wire [3:0] a;
wire [3:0] b;
wire      cin;
wire [3:0] sum;
wire      cout;
```

---

# STEP 1

```
// Wires and signals, add mark_debug so ILA can "see" them
(* mark_debug = "true" *) wire [3:0] a;
(* mark_debug = "true" *) wire [3:0] b;
(* mark_debug = "true" *) wire      cin;
(* mark_debug = "true" *) wire [3:0] sum;
(* mark_debug = "true" *) wire      cout;
```

---

# STEP 2: SETTING UP THE VIO

- We want to setup additional, virtual inputs, which can control the FPGA
- However, we may occasionally want to make use of the switches
- We can multiplex the switches and the virtual I/O, and have an additional, virtual signal `vio_sel`, to control which is the input
  - `vio_sel = 0` => using physical switches
  - `vio_sel = 1` => using VIO signals

---

# REPLACE INPUTS WITH MULTIPLEXED VERSION

```
// Assign inputs from switches
assign a    = sw[3:0];
assign b    = sw[7:4];
assign cin  = sw[8];
```



```
// Added VIO signals for debugging
wire [3:0] vio_a;
wire [3:0] vio_b;
wire vio_cin;
wire vio_sel; // VIO control switch (0 =
Physical, 1 = Virtual)
```

```
// Added Multiplexer to allow swapping
between hardware and software inputs
// If vio_sel is 1, use VIO. If 0, use
physical switches.
assign a    = vio_sel ? vio_a    : sw[3:0];
assign b    = vio_sel ? vio_b    : sw[7:4];
assign cin  = vio_sel ? vio_cin  : sw[8];
```

---

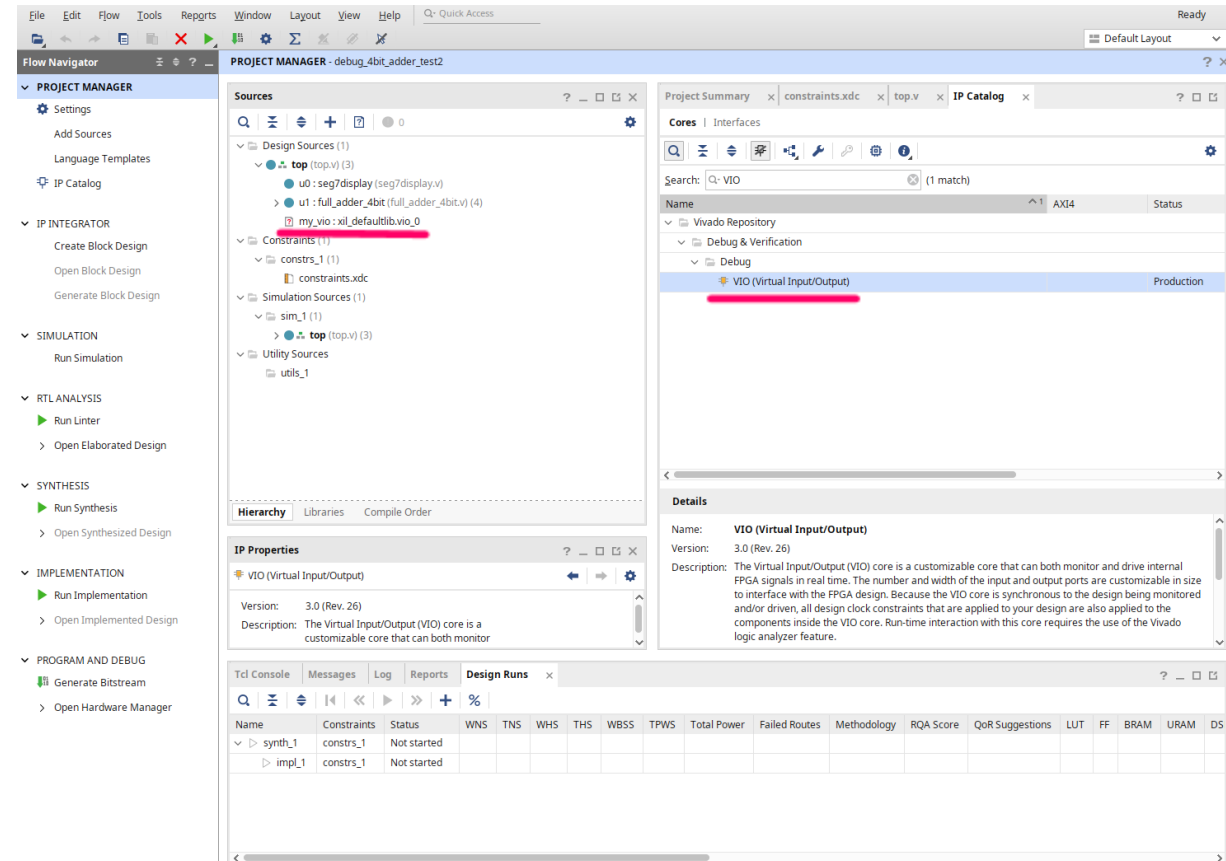
# ADD VIO CORE

- At the bottom of "top.v", add the following code to instantiate the VIO core

```
// --- NEW: Instantiate VIO Core ---
vio_0 my_vio (
    .clk(clk),           // input wire clk
    .probe_in0(sum),    // input wire [3 : 0] probe_in0
    .probe_in1(cout),   // input wire [0 : 0] probe_in1
    .probe_out0(vio_a), // output wire [3 : 0] probe_out0
    .probe_out1(vio_b), // output wire [3 : 0] probe_out1
    .probe_out2(vio_cin), // output wire [0 : 0] probe_out2
    .probe_out3(vio_sel) // output wire [0 : 0] probe_out3
);
```

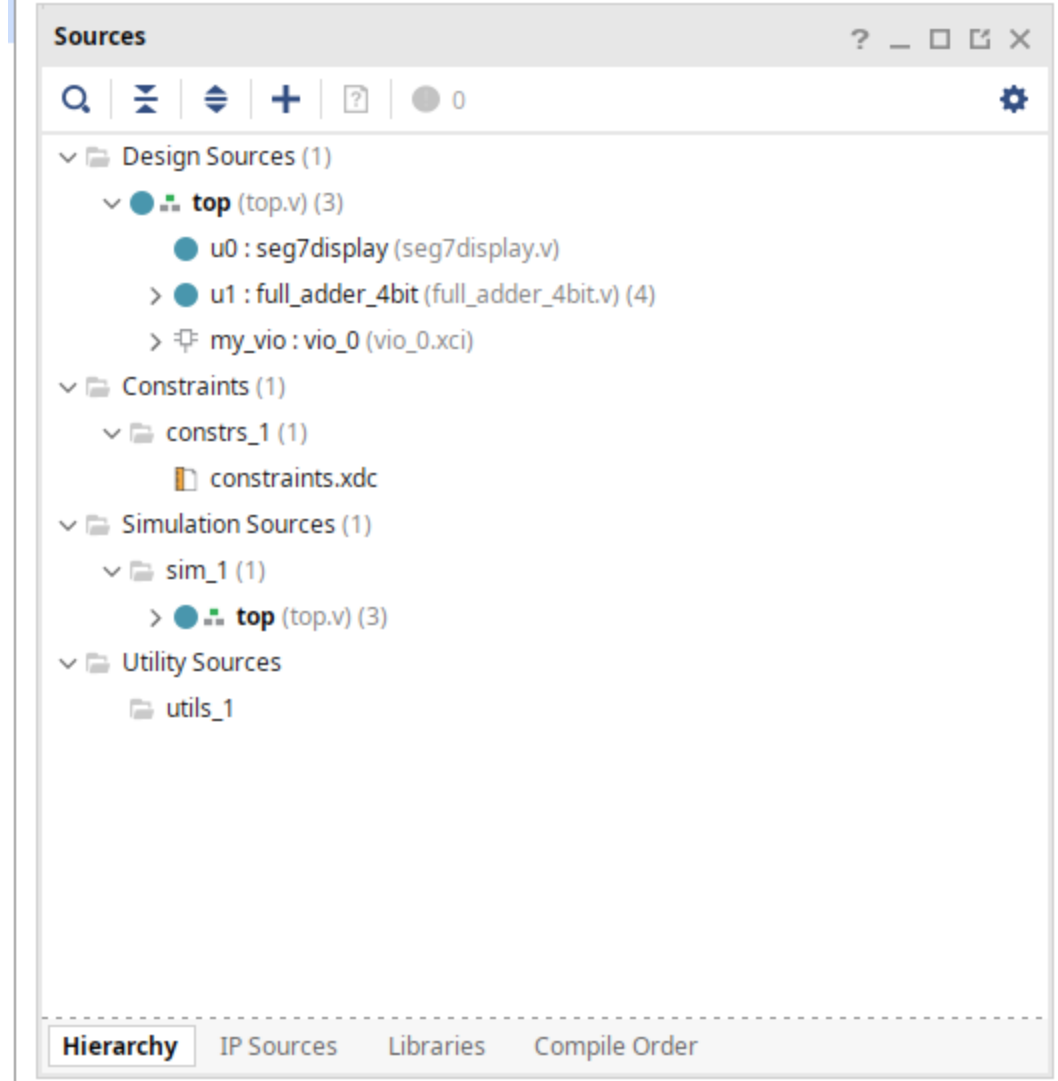
# ADDING THE VIO CORE

- 1) Open the IP catalogue on the left menu
- 2) Search "VIO (Virtual Input/Output)" and select it
- 3) In the configuration window:
  - PROBE\_IN Ports: Set to 2 (we are reading the outputs sum and cout)
  - PROBE\_OUT Ports: Set to 4 (we are controlling vio\_a, vio\_b, vio\_cin, and vio\_sel)



# ADDING THE VIO CORE

- 4) Go to the PROBE\_IN Ports tab
  - PROBE\_IN0 width = 4 (for 4-bit output of sum)
  - PROBE\_IN1 width = 1 (for cout)
- 5) Go to the PROBE\_OUT Ports tab
  - PROBE\_OUT0 width = 4 (for 4-bit input vio\_a)
  - PROBE\_OUT1 width = 4 (for 4-bit input vio\_B)
  - PROBE\_OUT2 width = 1 (for 1-bit input vio\_cin)
  - PROBE\_OUT3 width = 1 (for 1-bit input vio\_sel)
- 6) Under Synthesis Options, click Global.
- 7) Click OK -> Generate



---

# SETTING UP THE ILA CORE FOR DEBUGGING

- On the left menu, **Run Synthesis**
- Click **Open Synthesized Design**
- In the top menu, select **Tools -> Set Up Debug**.
- Click through the wizard. Ideally, you should see all wires (a, b, cin, sum, cout) listed automatically
  - If not (as shown on the right), follow the following steps...

# IF NOT ALL WIRES SHOW UP IN DEBUGGING WIZARD...

- 1) At the bottom of the wizard, click on "Find Nets to Add..."
- 2) In the new window, type "sum" and press enter
- 3) Select the wire and press "OK"
- 4) Repeat steps for cout

The image shows two screenshots from a software interface. The left screenshot is titled "Nets to Debug" and contains a table with the following data:

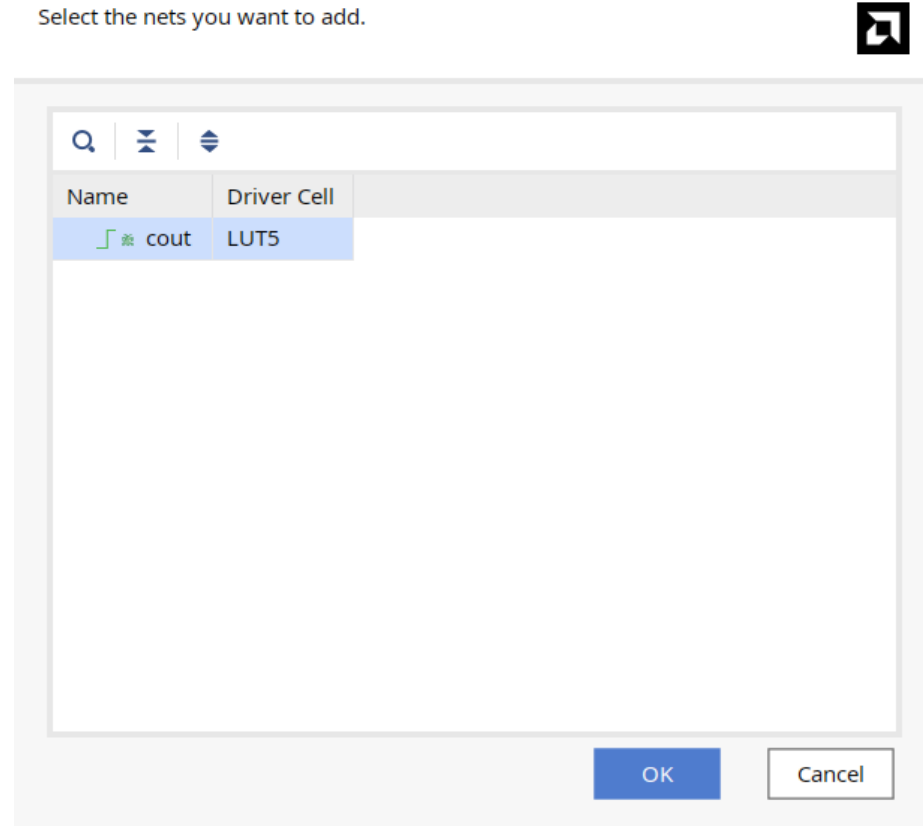
Name	Clock Domain	Driver Cell	Probe Type
>  a (4)	clk_IBUF_BUFG	IBUF	Data and Trigger
>  b (4)	clk_IBUF_BUFG	IBUF	Data and Trigger
>  sum (4)	clk_IBUF_BUFG	(Multiple)	Data and Trigger
cin	clk_IBUF_BUFG	IBUF	Data and Trigger

Below the table is a button labeled "Find Nets to Add..." and a status indicator "Nets to debug: 13". The right screenshot is titled "Find objects by filtering Tcl properties and objects." and shows a search filter set to "NAME" contains "cout". It also includes checkboxes for "Regular expression", "Search hierarchically", and "Display unique nets", and a "Command:" field with the text "T\_ONLY [get\_nets -hierarchical -top\_net\_of\_hierarchical\_... Specify -of\_objects o".

---

# IF NOT ALL WIRES SHOW UP IN DEBUGGING WIZARD...

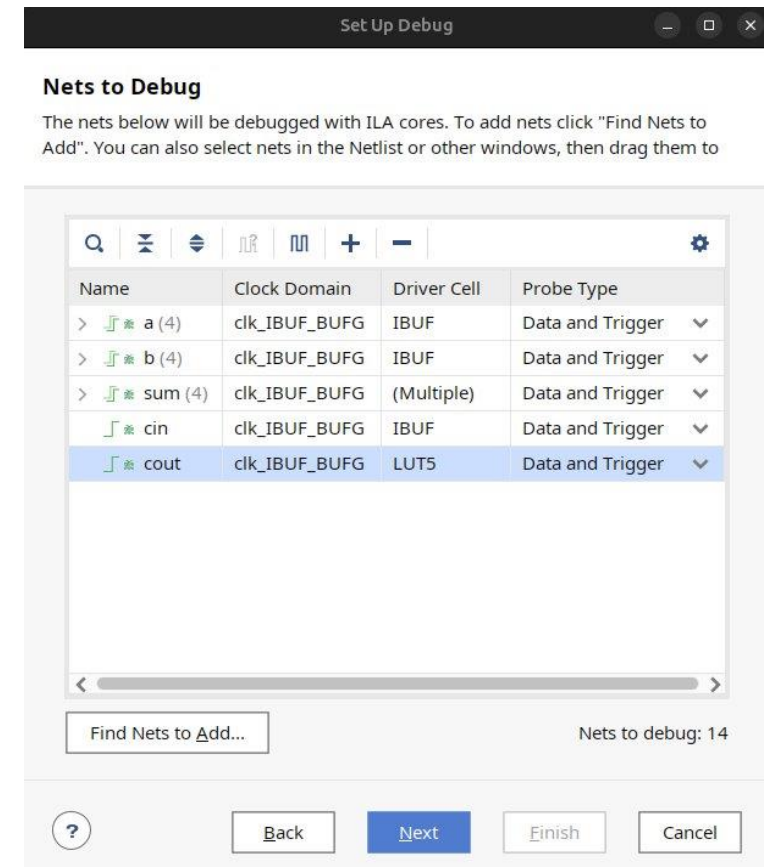
- 1) At the bottom of the wizard, click on "Find Nets to Add..."
- 2) In the new window, type "sum" and press enter
- 3) Select the wire and press "OK"
- 4) Repeat steps for cout



---

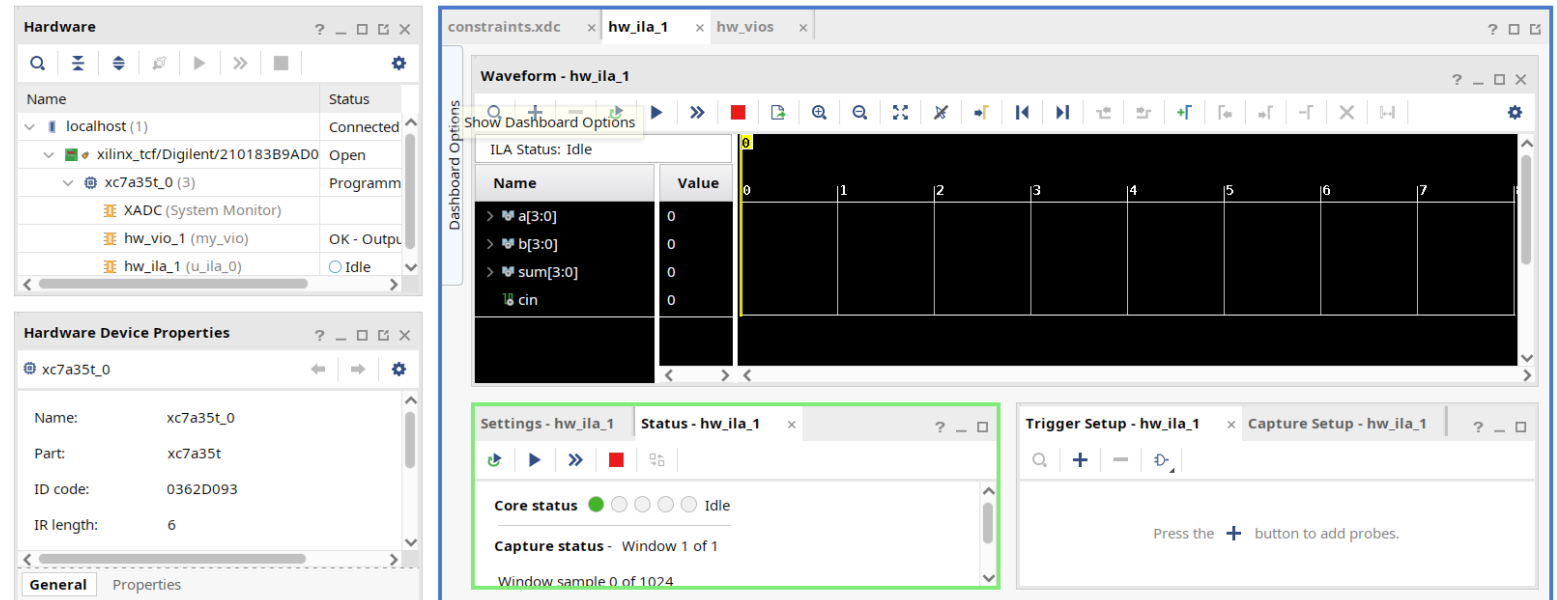
# FINISHING SETTING UP DEBUGGING CORE

- You should now have the following on the right
- Click through and finish the Debugging Wizard
- Once done, hit CTRL+S to save the debug constraints to your imported .xdc file
  
- Your constraints file should now have both the settings of the pinouts, and the setup for the debugging core



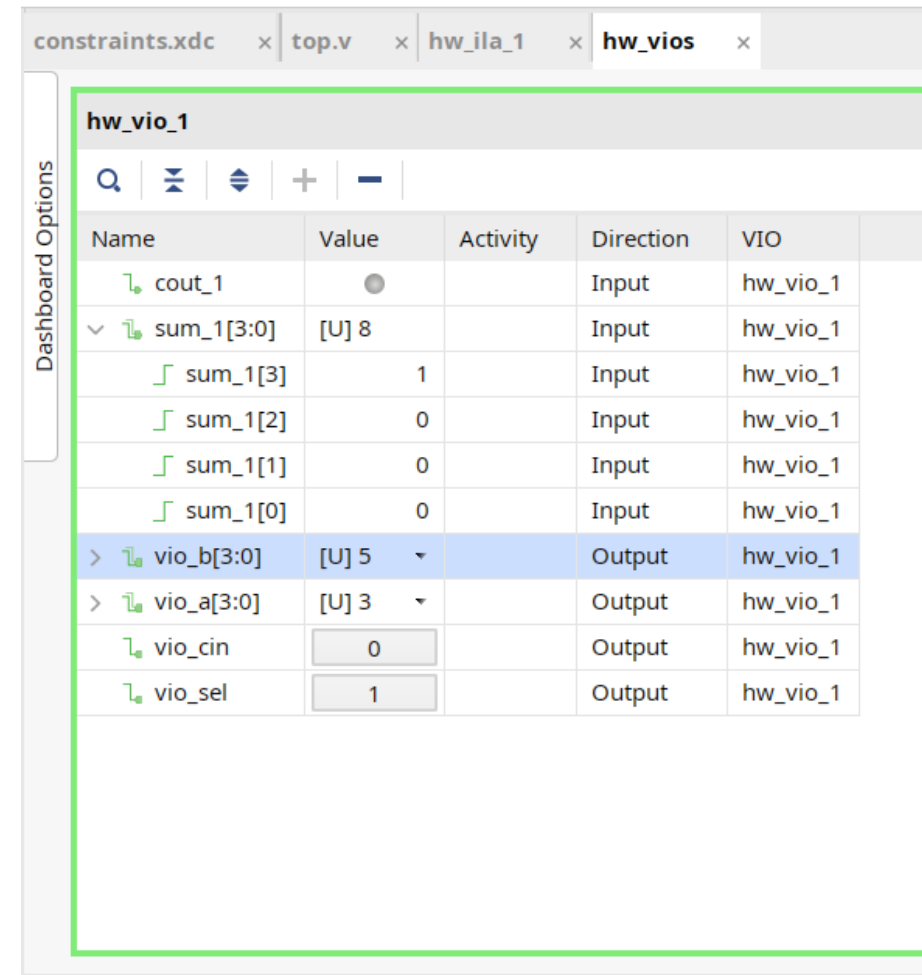
# GENERATE THE BITSTREAM

- On the left menu, select **Generate Bitstream**
- Program your Basys 3 board in the Hardware Manager.
  - The VIO dashboard and ILA dashboard will appear automatically in Vivado



# CONFIGURING THE VIO

- 1) Go to the **hw\_vios** tab in the Hardware Manager
- 2) At the top, press '+'
- 3) Select all of the wires present and press 'OK' (if the previous steps were done correctly, you should have the listed signals on the right). These will be your probes
- 4) Configure your probes by right-clicking on them
- 5) Make sure to have `vio_sel = 1` for the virtual I/O to work



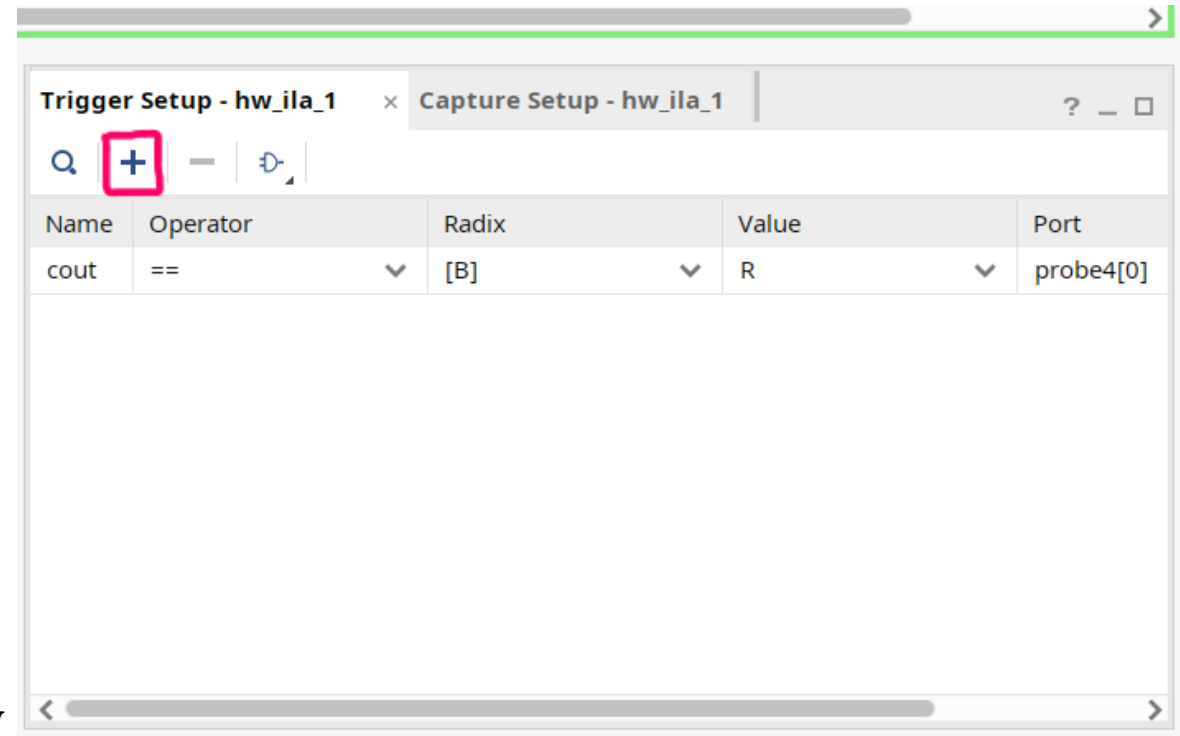
The screenshot shows the Hardware Manager interface with the `hw_vios` tab selected. The interface displays a table of signals and their configurations. The table has columns for Name, Value, Activity, Direction, and VIO. The signals listed are:

Name	Value	Activity	Direction	VIO
<code>cout_1</code>	<input type="radio"/>		Input	hw_vio_1
<code>sum_1[3:0]</code>	[U] 8		Input	hw_vio_1
<code>sum_1[3]</code>	1		Input	hw_vio_1
<code>sum_1[2]</code>	0		Input	hw_vio_1
<code>sum_1[1]</code>	0		Input	hw_vio_1
<code>sum_1[0]</code>	0		Input	hw_vio_1
<code>vio_b[3:0]</code>	[U] 5		Output	hw_vio_1
<code>vio_a[3:0]</code>	[U] 3		Output	hw_vio_1
<code>vio_cin</code>	<input type="text" value="0"/>		Output	hw_vio_1
<code>vio_sel</code>	<input type="text" value="1"/>		Output	hw_vio_1

---

# SETTING UP ILA TRIGGERS 1

- Let's say we want to capture the exact clock cycle when overflow happens. We can setup a ILA trigger, to show when exactly overflow occurred in our setup
- In Hardware Manager, go to the **hw\_ila\_1** tab, and press the '+' in the **Trigger Setup** window
- Select **cout**
- Under **Operator**, ensure it's set to '=='
- Under **Value**, ensure it's set to 'R (0-to-1 transition)'
- Press the play button at the top of the **Waveform window** to arm the trigger, and play around with the VIO



constraints.xdc x top.v x hw\_ila\_1 x hw\_vios x

### Waveform - hw\_ila\_1

ILA Status: Idle

Dashboard Options

Name	Value
> a[3:0]	2
> b[3:0]	13
> sum[3:0]	15
cin	0
cout	0

Updated at: 2026-Feb-19 15:30:17

# SETTING UP ILA TRIGGERS 2

- We can set up compound/multiple trigger conditions in the ILA. Let's say we want to only want the ILA to trigger when BOTH  $cout = 1$  and  $sum = 0$
- Go back to the **Trigger Setup**, and add 'sum' as a trigger condition
- For 'sum', set **Operator** to '==' and **Value** to 0
- Under **Trigger Mode** button (red box), ensure its set to 'Global AND'.
- Press the play button on the Waveform window and play around with the VIO

