

Hardware Debugging

**See Vivado Programming and Debugging UG908
(www.xilinx.com)**

Hardware debugging permite analisar os sinais internos na FPGA através da inserção no projecto de um componente “analisador lógico” que faz a amostragem dos sinais pretendidos.

Hardware debugging allows you to analyze the internal signals in the FPGA by inserting into the design a "logic analyzer" component that samples the intended signals.

In-System Logic Design Debugging

Logic analysis feature to perform in-system debugging of the post-implemented design in the SoC FPGA device.

In-system debugging flow:

1. Probing phase: Identify the design signals to probe.
2. Implementation phase: Implement the design including the additional debug IP that is attached to the nets to be probed.
3. Analysis phase: Interact with the debug IP to debug and verify functional issues.

O analisador lógico integrado é ligado aos sinais que se pretende amostrar.

The integrated logic analyzer is connected to the signals you want to sample.

Probing the Design for HW Debugging

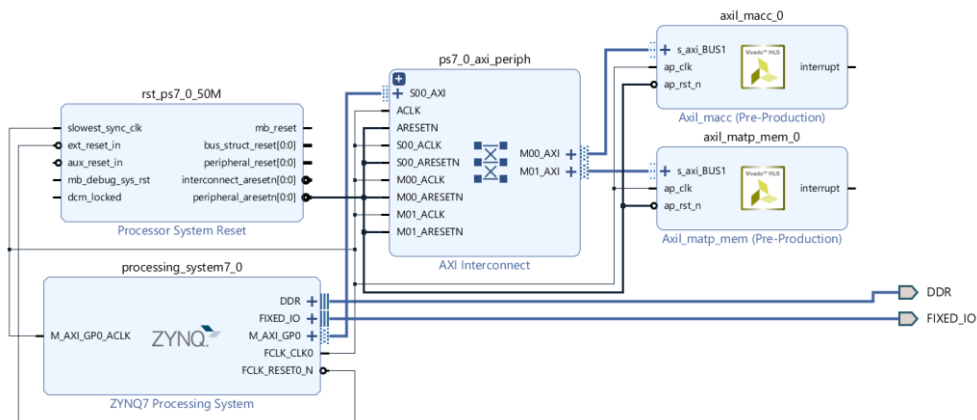
Probing phase steps:

1. Identify what signals or nets you want to probe
2. Add debug cores to your design:
 - i) manually add the debug IP component instances to your design source code
(HDL instantiation probing flow)
 - ii) automatically insert the debug cores into the post-synthesis netlist
(netlist insertion probing flow)**

O componente será ligado usando o método ii)

The component will be interconnected using method ii)

HW/SW Lab0 System with 2 IPs



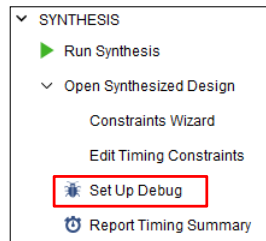
Synthesize the system, as normally ...
then, *open synthesized design* ...

O projecto é sintetizado, como habitualmente.
Após a síntese abre-se o projecto sintetizado, "open synthesized design",
para inserir o analisador lógico.

The project is synthesized, as usual.
After the synthesis, the synthesized project is opened, "open synthesized
design", to insert the logic analyzer.

Insertion of debug cores

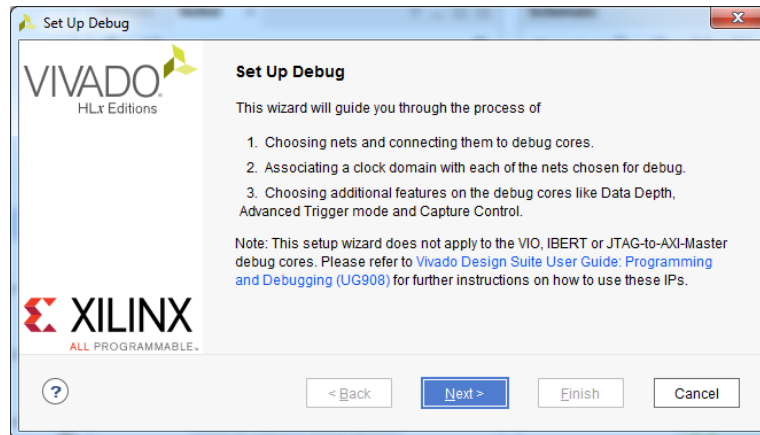
Simple wizard that creates and automatically configures **Integrated Logic Analyzer (ILA)** cores based on the set of nets selected to debug.



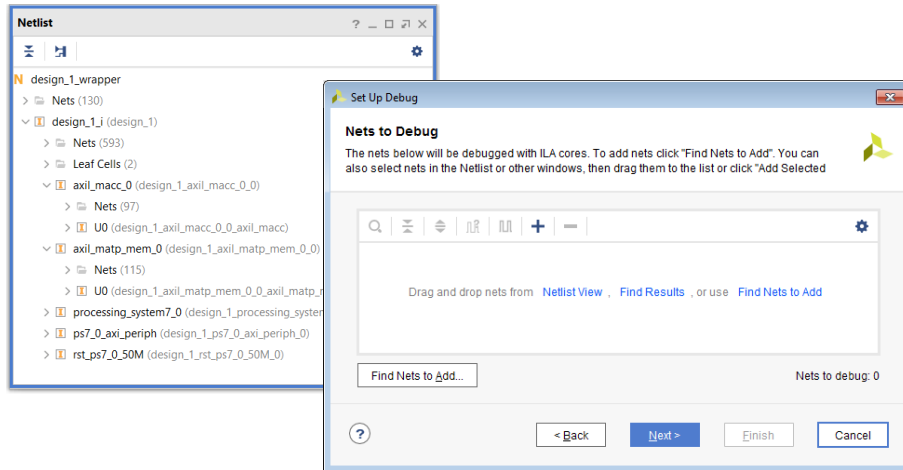
O Vivado providencia um wizard de utilização simples que configura e insere o analisador lógico integrado no circuito.

Vivado provides a user-friendly wizard that configures and inserts the integrated logic analyzer into the circuit.

Set Up Debug



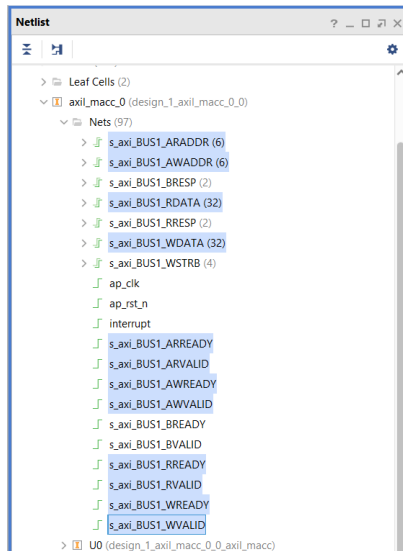
Nets to Debug



O projectista escolhe quais os sinais que pretende analisar, avaliando o compromisso adequado entre a quantidade de sinais necessários para o debug ser efectivo e a quantidade de recursos requerida para essa análise.

The designer chooses which signals to analyze, assessing the appropriate compromise between the number of signals needed for the debug to be effective and the number of resources required for that analysis.

AXI-Lite Interface Signals

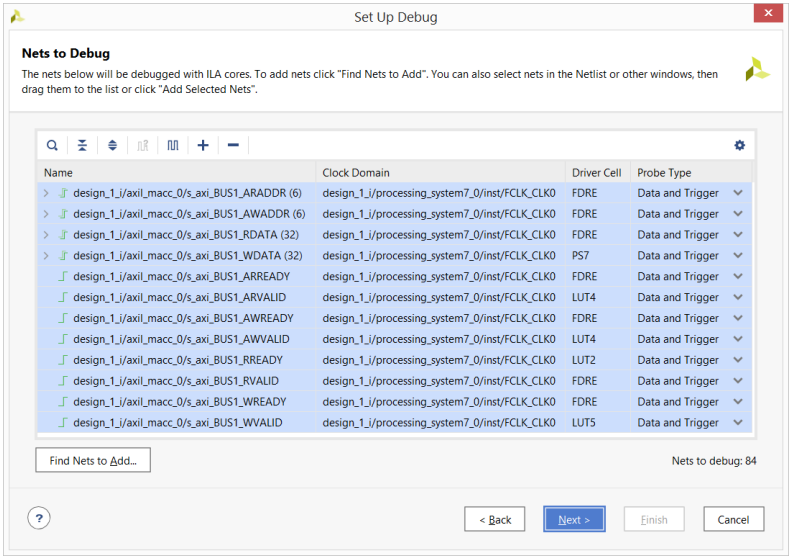


Will debug the AXI-Lite interface signals of the **axil_macc IP**

Neste exemplo, seleccionamos para análise 12 sinais dos sinais da interface AXI-Lite de entrada/saída do componente axil_macc (projectado usando o HLS).

In this example, we selected for analysis 12 signals from the AXI-Lite input/output interface of the axil_macc component (designed using HLS).

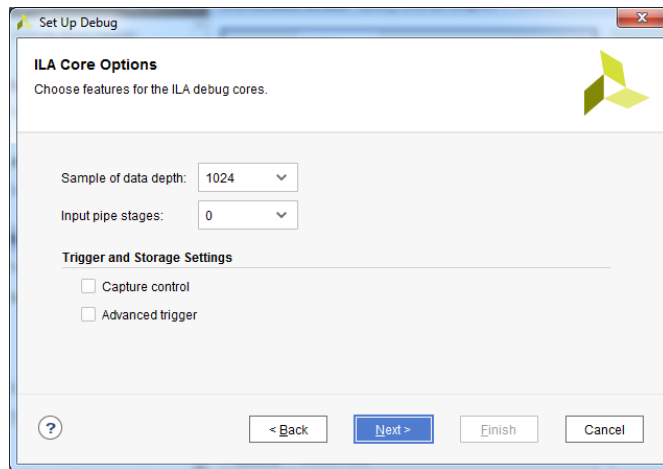
Nets to Debug Window



Os sinais (selecionados neste exemplo a partir da janela “netlist view”) aparecem na janela “nets to debug”.

The signals (in this example, selected from the "netlist view" window) appear in the "nets to debug" window.

Default ILA Options



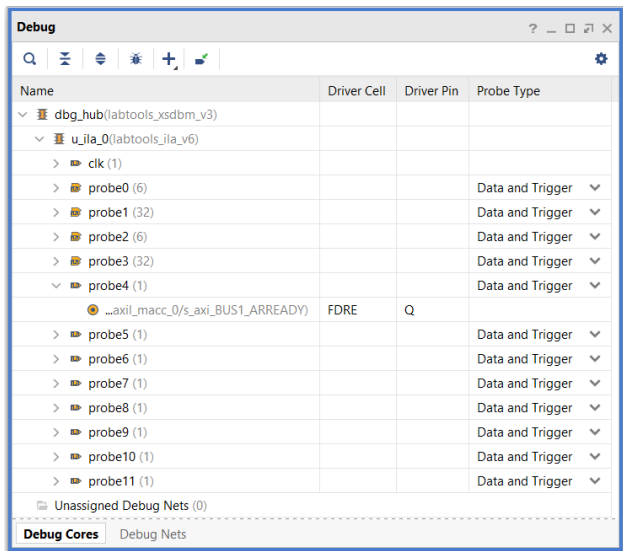
Neste exemplo, mantemos as opções do ILA com os valores por omissão. São armazenadas 1024 amostras de cada sinal, por sessão, e não são inseridos andares de pipeline nas entradas.

In this example, we keep the ILA options with the default values. 1024 samples of each signal are stored per session and no pipeline stages are inserted in the inputs.

Finish Set Up Debug



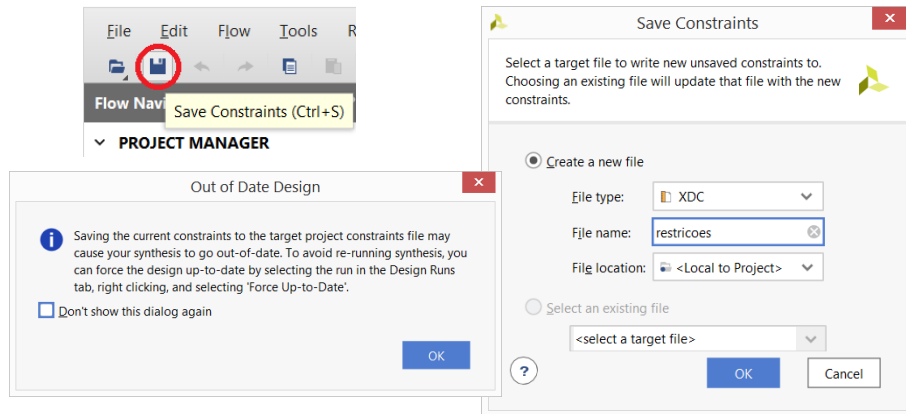
Debug Window



No final do setup, a janela de debug identifica quais os sinais que ficaram ligados a cada uma das 12 “probes” do ILA.

At the end of the setup, the debug window identifies which signals were linked to each of the 12 ILA probes.

Save and Run Project



Afterwards, Run Implementation, Generate Bitstream, Export Hardware, and prepare Software Application (all as usual)

Após concluir o setup do ILA o Vivado cria um conjunto de “constraints” de projecto relacionadas com o ILA, que é necessário salvar num ficheiro. Após salvar o ficheiro de restrições, deve continuar a implementação, gerar a bitstream e exportar o hardware como habitualmente.

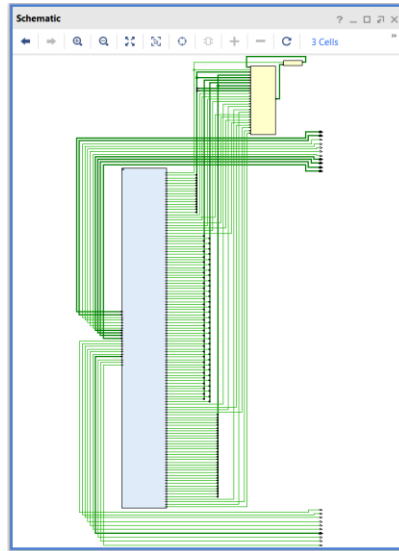
Note-se que, após a actualização do ficheiro de restrições, o Vivado pode indicar que o projecto sintetizado está out-of-date. Pode simplesmente re-sintetizar o projecto (aceitando a sugestão do Vivado quando inicia a implementação) ou previamente forçar o up-to-date conforme indicado.

After completing the ILA setup, Vivado creates a set of ILA-related project constraints, which you need to save to a file.

After saving the restrictions file, you should continue the deployment, generate the bitstream and export the hardware as usual.

Note that after updating the restrictions file, Vivado may indicate that the synthesized project is out-of-date. You can simply re-synthesize the project (accepting Vivado suggestion when you start the implementation) or previously force the up-to-date as indicated.

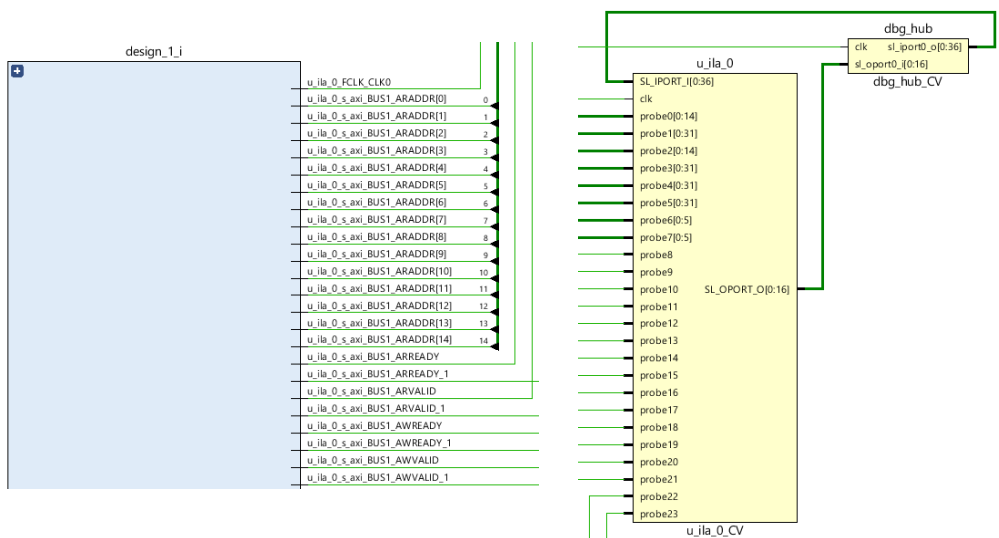
Schematic: Design + ILA



Se rever o esquema do projecto sintetizado, pode notar a ligação do ILA no projecto.

If you review the schematic of the synthesized project, you may notice the ILA's integration in the project.




Schematic: Design + ILA (Zoom In)



Pode ver-se o ILA e o componente de controlo do debug.

You can see the ILA and the debug control component.

ILA HW utilization

Name	Slice LUTs (17600)	Slice Registers (35200)	DSPs (80)	Block RAM Tile (60)
▼ N design_1_wrapper	3360	4999	10	8.5
>  design_1_i (design_1)	1190	1441	10	3
>  dbg_hub (dbg_hub)	442	727	0	0
>  u_ila_0 (u_ila_0)	1728	2831	0	5.5

A inserção do ILA corresponde obviamente a um acréscimo no consumo de recursos e pode reduzir a frequência máxima de relógio que o circuito consegue suportar.

Neste exemplo, o ILA e o controlador adicionam cerca de 2200 LUTs e 5 BRAMs ao total de recursos usado, o que corresponde a cerca de 12% dos recursos disponíveis na FPGA. Note que são necessárias BRAMs para armazenar as 1024 amostras dos 12 sinais (com um total de 84 bits).

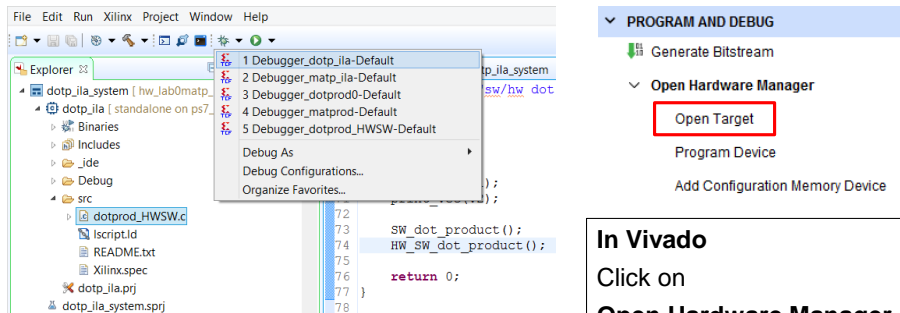
Neste caso, a inclusão do ILA não afecta significativamente a frequência de trabalho (tal poderá acontecer se o nº de amostras e sinais seleccionados for bastante maior).

The insertion of the ILA obviously corresponds to an increase in resource consumption and can reduce the maximum clock frequency that the circuit can support.

In this example, the ILA and controller add about 2200 LUTs and 5 BRAMs to the total resources used, which is about 12% of the resources available on the FPGA. Note that BRAMs are required to store the 1024 samples of the 12 signals (with a total of 84 bits).

In this case, the inclusion of ILA does not significantly affect the working frequency (this may happen if the number of samples and signals selected is much higher).

Test in Hardware



In Vitis

(define the Configuration, as usual)

select **Debug Mode** to download the application.

The program execution starts and suspends at the entry point.

In Vivado

Click on

Open Hardware Manager

to invoke the analyzer.

Then, click on the

Open Target > Auto connect

to establish the connection with the board.

No Vitis, criem a plataforma hardware, o projecto e a configuração como habitual. Executem (Run) uma vez, como habitual, para confirmar que está a funcionar como na versão sem ILA.

Para o debug em hardware, vão alternar entre o software debugger no Vitis e a análise lógica no Vivado.

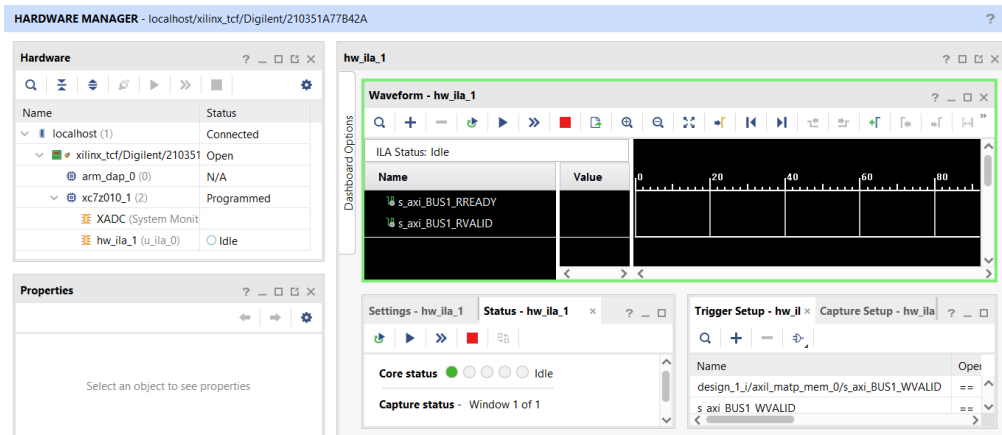
Conforme indicado: no Vitis, iniciem a execução da aplicação no debugger; no Vivado, abram o Hardware Manager e façam a ligação à placa.

In Vitis, create the hardware platform, design, and configuration as usual. Run once, as usual, to confirm that it is working as in the version without ILA.

For hardware debugging, switch between debugger software in Vitis and logical analysis in Vivado.

As indicated: in Vitis, start running the application in the debugger; in Vivado, open Hardware Manager and connect to the board.

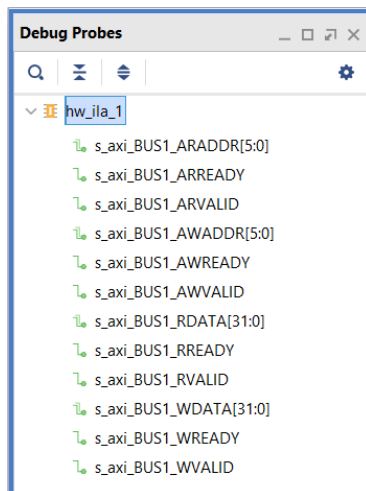
Hardware Session



No Vivado, as janelas no Hardware Manager terão o aspecto indicado, com a janela para análise dos sinais, à direita em cima, e as janelas de configuração do trigger, à direita em baixo.

In Vivado, the windows in Hardware Manager will look as indicated, with the window for analyzing the signals, on the right at the top, and the trigger configuration windows, on the right at the bottom.

Debug Probes Window



(Add Debug Probes Window to view, if necessary)

Select which probes to add to waveform window, if not included by default.

Debugging the signals of the AXi-Lite interface of the axil_macc IP

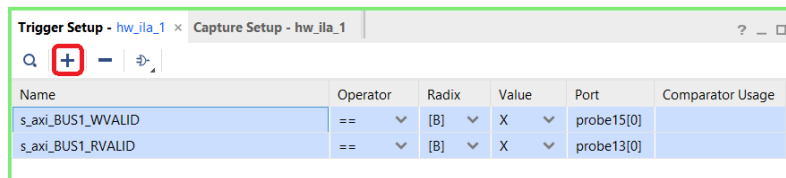
A janela com as debug probes está inativa por omissão. Têm de adicioná-la explicitamente se quiserem adicionar sinais na janela de visualização (que não estejam já lá, por omissão).

The window with the debug probes is inactive by default. You must add it explicitly for adding signs to the preview window (that aren't already there, by default).

Trigger Setup (1)

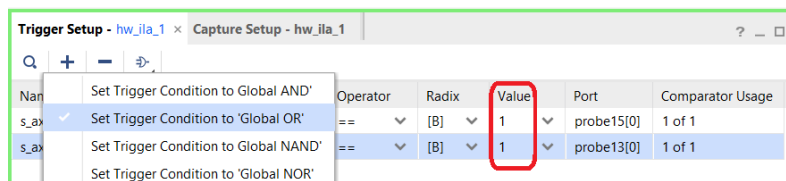
Select which probes to use as a trigger.

In this example, selected **bus1_wvalid** and **bus1_rvalid**



Name	Operator	Radix	Value	Port	Comparator Usage
s_axi_BUS1_WVALID	==	[B]	X	probe15[0]	
s_axi_BUS1_RVALID	==	[B]	X	probe13[0]	

Define trigger as **bus1_wvalid == 1 OR bus1_rvalid == 1**



Name	Operator	Radix	Value	Port	Comparator Usage
s_axi_BUS1_WVALID	==	[B]	1	probe15[0]	1 of 1
s_axi_BUS1_RVALID	==	[B]	1	probe13[0]	1 of 1

O trigger tem de ser configurado com as condições de início de um processo de amostragem.

Neste exemplo, configuramos o trigger para activar a amostragem quando um dos dois sinais de data valid passa a '1'.

The trigger must be configured with the conditions for starting a sampling process.

In this example, we set the trigger to activate sampling when one of the two data valid signals becomes '1'.

Trigger Setup (2)

Settings - hw_ila_1 × Status - hw_ila_1

Trigger Mode Settings

Trigger mode: BASIC_ONLY

Capture Mode Settings

Capture mode: ALWAYS

Number of windows: 1 [1 - 1024]

Window data depth: 1024 [1 - 1024]

Trigger position in window: 100 [0 - 1023]

General Settings

Refresh rate: 500 ms

Define trigger position in window as 100
(the trigger will be shown in the first 1/10 part of the window)

Configura-se o instante em que o trigger activa a amostragem para aparecer na posição 100 da janela de visualização. Serão, portanto, visualizadas 99 amostras antes da activação e 924 após a activação.

The instant at which the trigger activates sampling is configured to appear at position 100 of the display window. Therefore, 99 samples will be displayed before activation and 924 after activation.

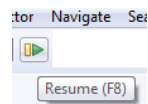
SW Breakpoint

```
67 int main()
68 {
69     init_vecs();
70     print_vec(v1);
71     print_vec(v2);
72
73     SW_dot_product();
74     HW_SW_dot_product();
75
76     return 0;
77 }
```

In Vitis

double click on the left border to set a breakpoint in the software application.

Click on the **Resume** button to execute the program and stop at the breakpoint.



```
67 int main()
68 {
69     init_vecs();
70     print_vec(v1);
71     print_vec(v2);
72
73     SW_dot_product();
74     HW_SW_dot_product();
75
76     return 0;
77 }
```

Passo 1: paramos o debugger antes de se iniciar a primeira transferência de dados para o IP (Vitis).

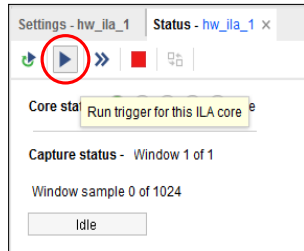
Neste, exemplo paramos antes de executar a função HW_SW_dot_product.

Step 1: we stop the debugger before starting the first data transfer to the IP (Vitis).

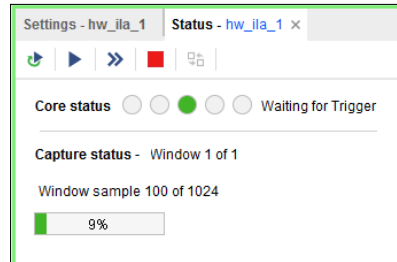
In this example, we stop before executing the function HW_SW_dot_product.

Arming the trigger

In Vivado



Click on the **Run Trigger** button.



The *hw_ila_1* core is **armed**, and its status is shown as **Waiting For Trigger**.

Passo 2: armamos o trigger para iniciar a amostragem, logo que apareça um data-valid na entrada do IP (Vivado).

O trigger tem de ser “armado” sempre que quisermos fazer uma nova amostragem.

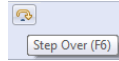
Step 2: set the trigger to start sampling, as soon as a data-valid appears in the IP entry (Vivado).

The trigger must be "armed" whenever we want to do a new sampling.

SW Breakpoint

In Vitis

Click on the **Step Over** button to execute the function `HW_SW_dot_product()`;



```
67 int main()  
68 {  
69     init_vecs();  
70     print_vec(v1);  
71     print_vec(v2);  
72  
73     SW_dot_product();  
74     HW_SW_dot_product();  
75  
76     return 0;  
77 }
```

The function that accesses the IP registers is executed and the program stops at the following instruction.

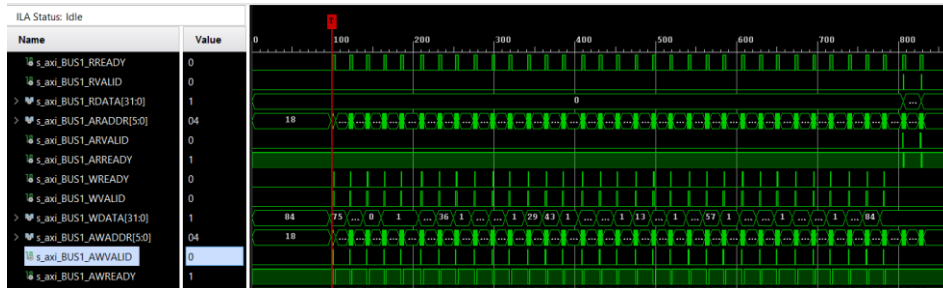
Passo 3: na aplicação (Vitis), executamos o passo de envio (ou recepção) de dados.

Neste exemplo, executamos 1 vez a função `HW_SW_dot_product` (com um step over).

Step 3: in the application (Vitis), execute a step of sending (or receiving) data.

In this example, we run the `HW_SW_dot_product` function 1 time (with a step over).

ILA waveform



The ILA waveform shows 1024 samples of the debug probes, starting 100 samples before the trigger.

The ILA was triggered by the first wvalid==1.

Passo 4: a sequência de sinais amostrados aparecerá na janela de visualização (Vivado).

Neste exemplo, a janela de visualização mostra a activação da amostragem no instante 100 (wvalid == '1').

(cada instante corresponde a um evento de relógio)

Vê-se também a sequência de elementos a ser recebida pelo IP.

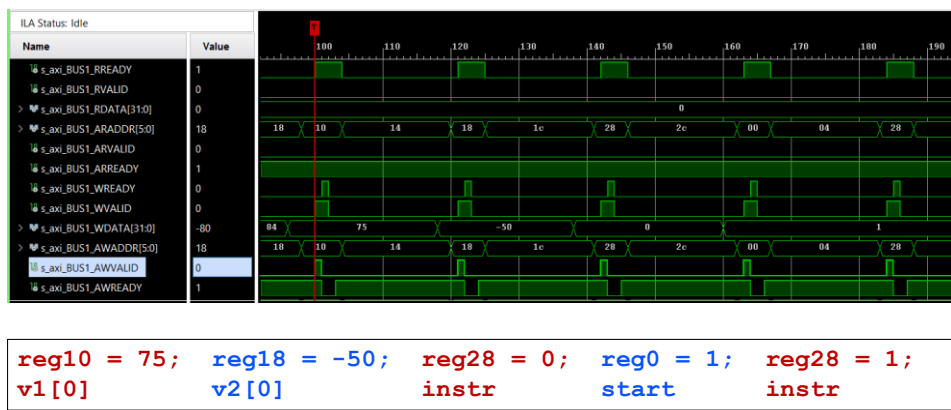
Step 4: the sequence of sampled signals appears in the preview window (Vivado).

In this example, the preview window shows the sampling activation at instant 100 (wvalid == '1').

(each instant corresponds to a clock event)

The sequence of elements being received by the IP is also shown.

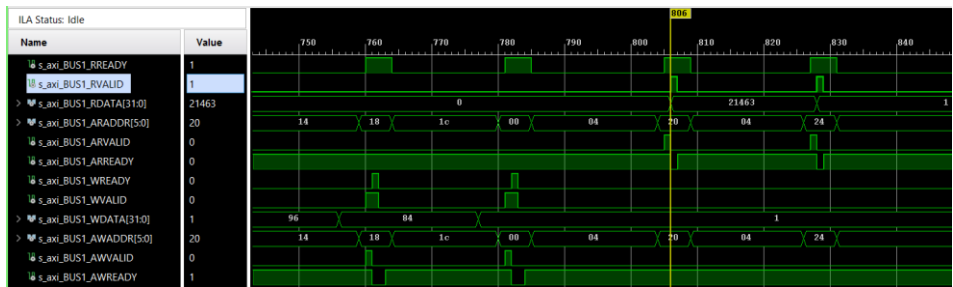
1st five AXI-Lite data writes



Aqui vêm-se os primeiros dados a serem escritos nos registros do IP (e o comando de start).

Here, the first data to be written to the IP registers (and the start command) are shown.

2 AXI-Lite data reads



The final result is read from **reg20 = 21463**.
The c_vld is **reg24 = 1**.

Aqui é lido o resultado.

Here, the result is read.