

# Introduction to HDL simulation, FPGA resources, and IP Cores

# Why do we simulate HDL code?

- Generating firmware takes many minutes to hours depending on size of logic.
- Not possible monitor all signals in logic for a FPGA.
  - Example: Let's say we have  $y = f(x)$  firmware in FPGA. Can't monitor all logic inside function  $f(x)$ . So debugging is difficult.
- Simulation generally used to debug logic easier.



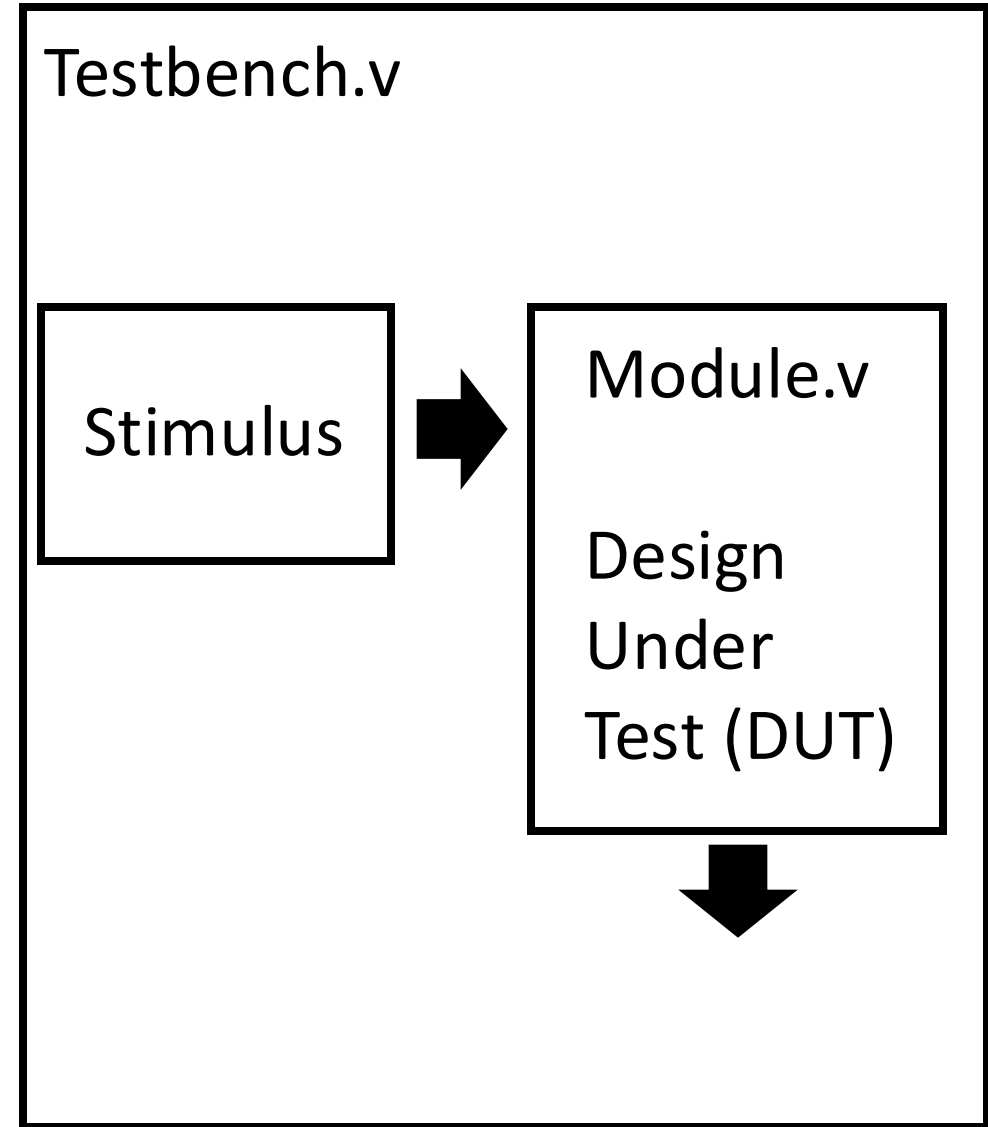
# How can we simulate HDL code?

- A HDL module has input ports and output ports.
- Must provide input data to HDL module to do simulation.
- Logic that provides input data is called “stimulus”

```
module half_adder (  
    input  wire A,  
    input  wire B,  
    output wire SUM,  
    output wire CARRY  
);
```

# How can we simulate HDL code?

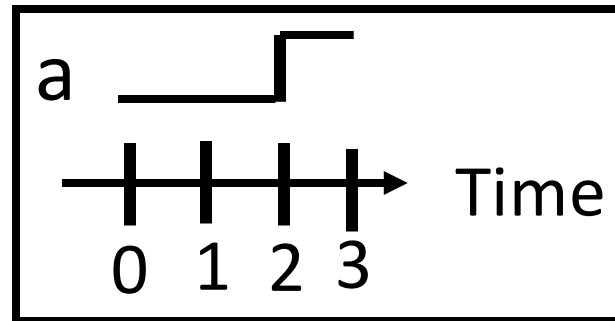
- We make a “testbench” HDL code for simulation.
  - Not used in synthesis of firmware
  - Can use non-synthesizable HDL.
    - ❖ Ex) Read from txt file.
- In simulation, all signals in logic can be seen.



# Writing simple stimulus

- Generate signals at timing

- Define time unit
- Assign value to signal
- Delay by number of time units
- Assign new value to signal



Time unit  
Time precision:  
When to do rounding

↘ ↘

``timescale 1ns/1ps`

`a = 1'b0;`

`#2; // 2 unit delay`

`a = 1'b1;`

`#1; // 1 unit delay`

# Writing simple stimulus

- Simulation code is typically written in **initial blocks**.

- Block of code is started at beginning of simulation time 0 unit.
- Only executed once in entire simulation.
  - ❖ Normally HDL code is constantly executed.

```
`timescale 1ns/1ns

module testbench;

reg a;

initial begin
    a = 1'b0;
    # 2;
    a = 1'b1;
    # 1;
end

endmodule
```

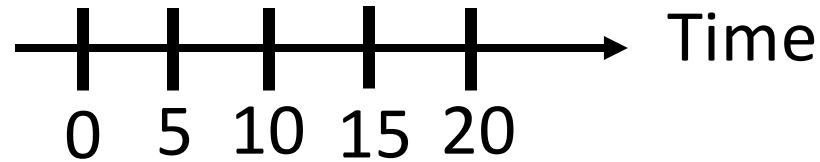
# Ways to write stimulus

- Generating 100 MHz clock

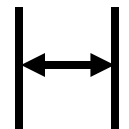
➤ Start at 0 and switch at every 5 ns.

```
reg myclk;  
  
initial begin  
    myclk = 0;  
    forever #5 myclk = ~myclk;  
end
```

myclk 



 Clock cycle is 10 ns = 100 Mhz

 Half of time is active (1b'1) → clock duty cycle is 50%

# Ways to write stimulus

- Also possible to read txt file

Input.txt

10 15

25 13

12 11

...

```
`timescale 1ns/1ps
module tb; // testbench
    reg [7:0] a, b; // Hold integer values (0–255)
    integer fd;    // File descriptor

    adder dut (a, b, sum);

    initial begin
        fd = $fopen("stimulus.txt", "r");
        while (!$feof(fd)) begin
            $fscanf(fd, "%d %d\n", a, b); #10;
        end
        $fclose(fd);
    end

endmodule
```

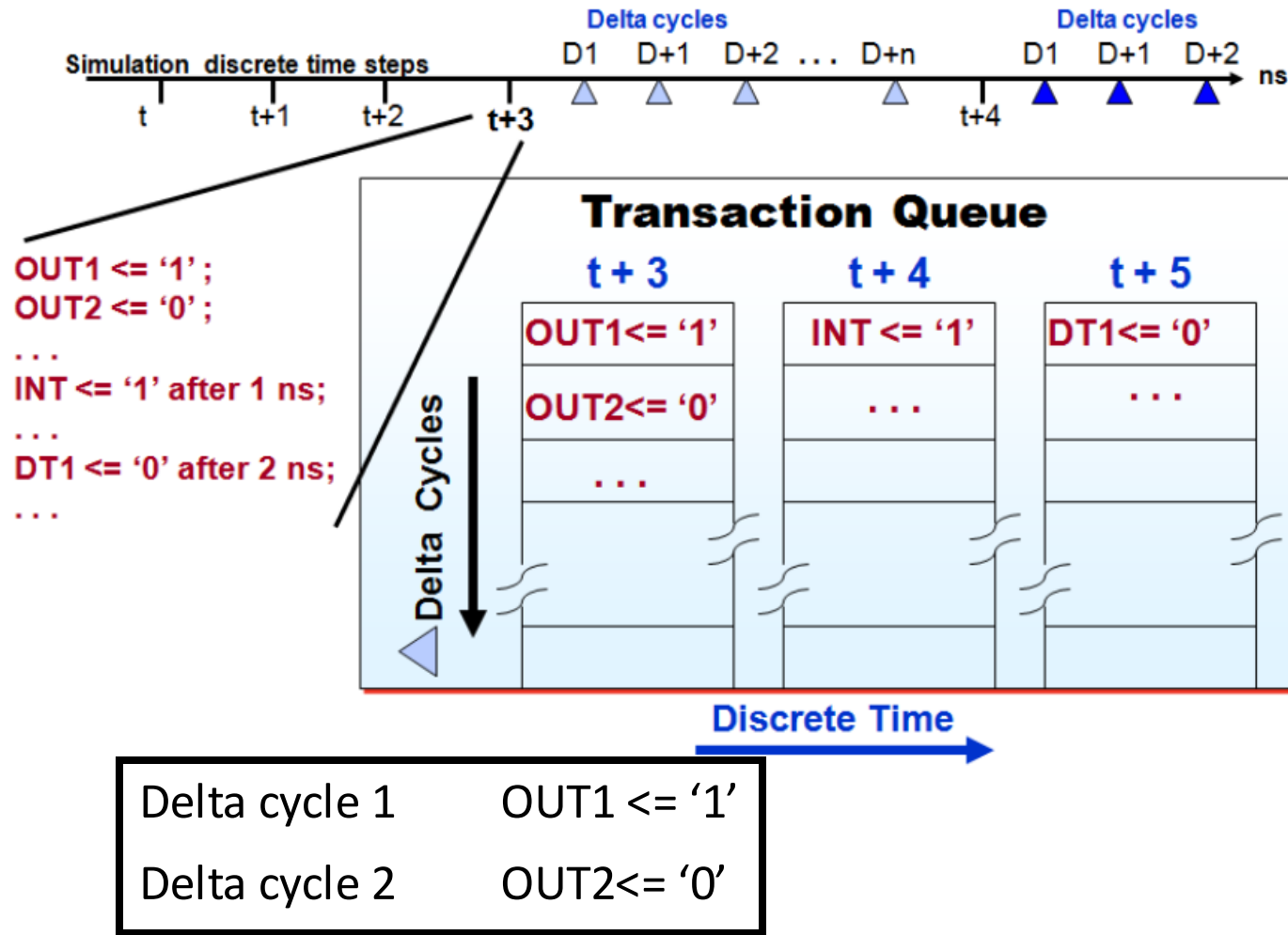


# How does simulation handle concurrency?

- Simulation is done on CPU (Does one task at a time).
  - But HDL can have many tasks at same time.
- So simulation does below
  1. Time step is paused.
  2. Run statements line by line. Schedule when signal change.
  3. If all statements are ran, update signals and go to next time step.

# Delta cycles

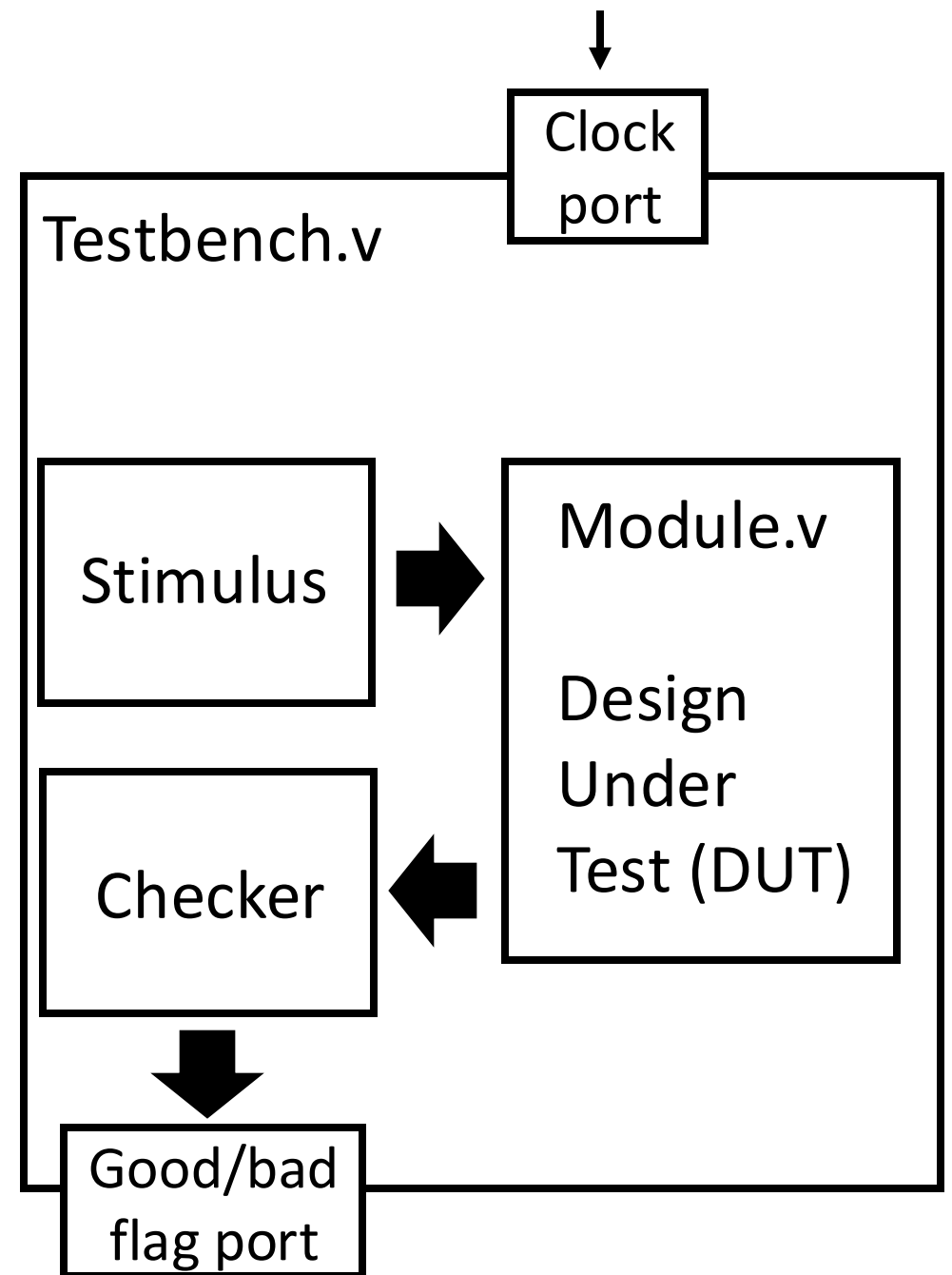
- Multiple statements can have same time step.
- In CPU, they are different “delta cycle”, but in HDL they have same time.
- Sometimes, bad HDL logic depends on delta cycle. →



Firmware on FPGA and simulation will show different results

# Synthesizable test bench

- Sometimes, a synthesizable test bench is used to check if module works correctly.
- Could also simulate this synthesizable test bench.



# FPGA resources

- There are DSP (digital signal processor) and BRAM (Block RAM) inside FPGAs.

- Multiplication can be done on DSP.

[Ref on AMD DSP](#)

- Artix-7 DSP can do up to 25 bit  $\times$  18 bit

multiplication. Good use below 25 or 18 bits in logic.

- Can do multiplication using CLB, but uses lots of CLBs.

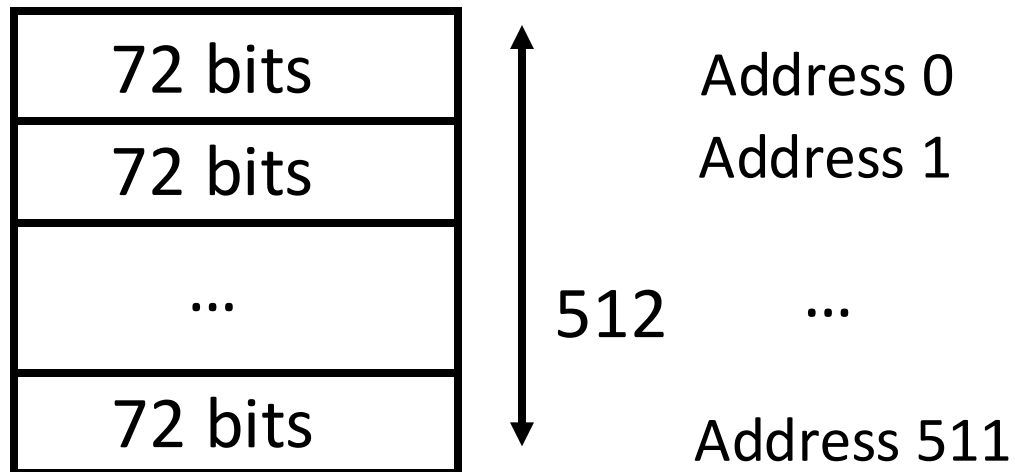
```
C <= A * B; // Vivado will normally automatically use DSP for this.  
// But could also use CLBs if bit width is small.
```

Device Name	Z-7020
Part Number	XC7Z020
Xilinx 7 Series Programmable Logic Equivalent	Artix-7 FPGA
Programmable Logic Cells	85K
Look-Up Tables (LUTs)	53,200
Flip-Flops	106,400
Block RAM (# 36 Kb Blocks)	4.9 Mb (140)
DSP Slices (18x25 MACCs)	220

# BRAM

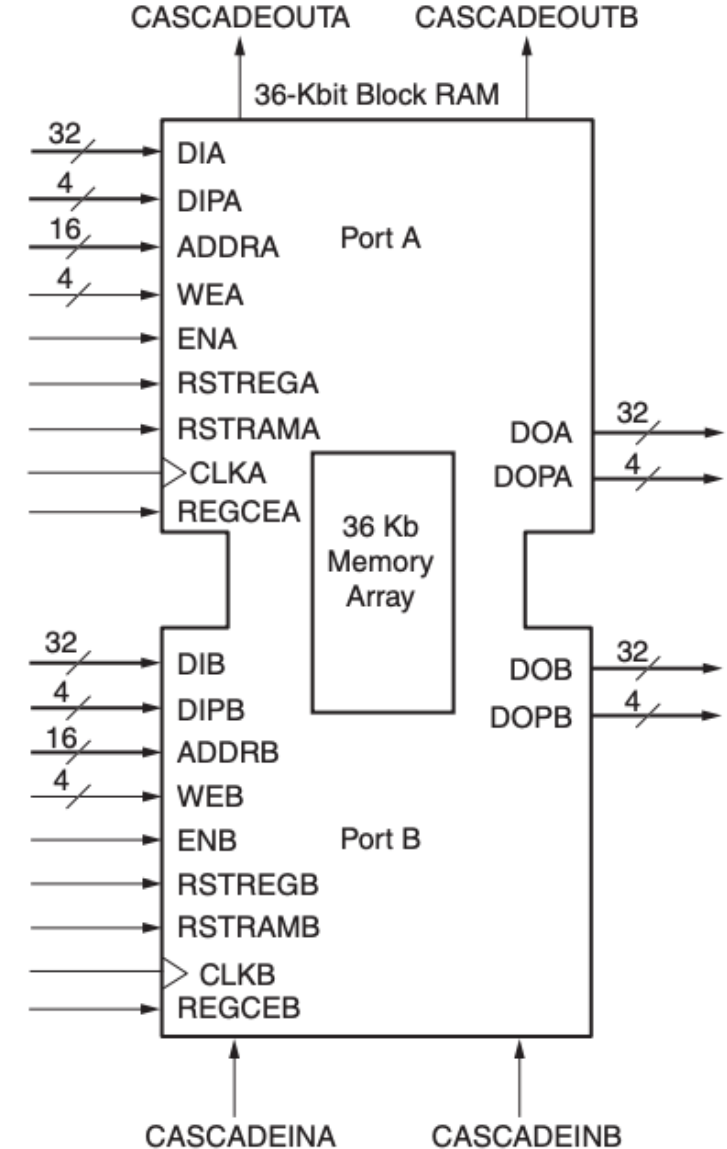
- Each BRAM stores up to 36,864 bits.
  - Can be configured as two 18 Kb RAM
- Can write/read up to 72 bits per clock.

➤ Then there can be 512 of the 72 bits.



Each row will have an address.  
Ex) 9 bit address needed.

[Ref on AMD BRAM](#)



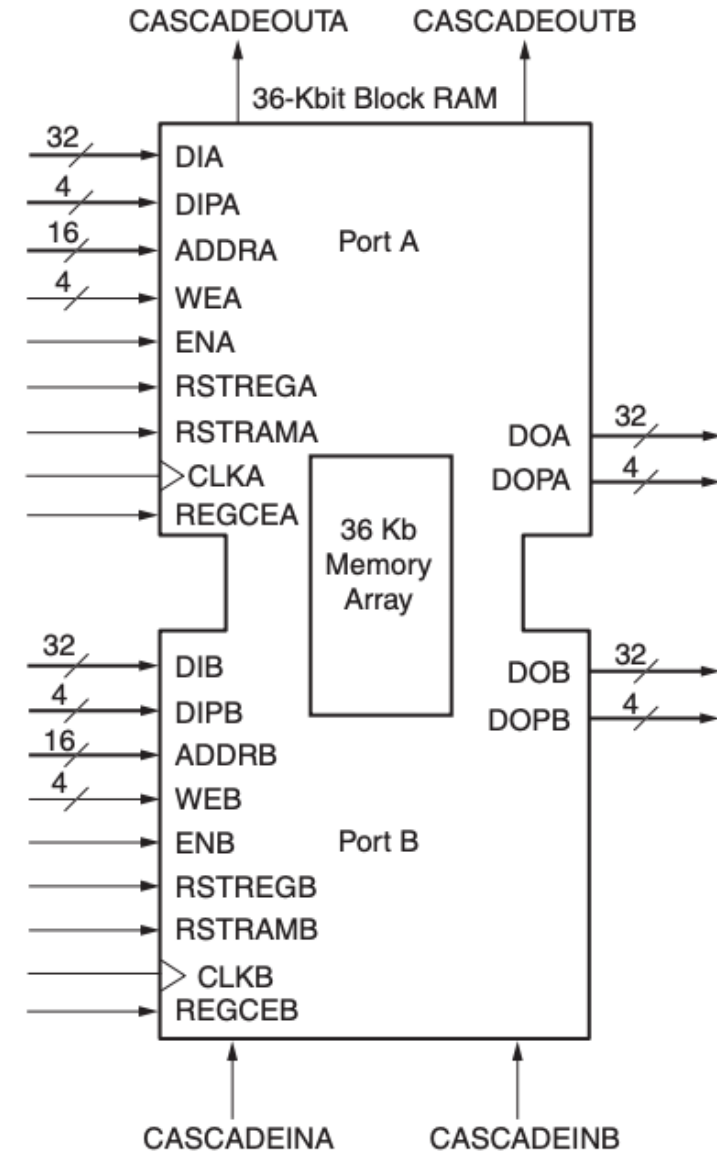
# BRAM has multiple modes

- True dual port mode

- Can write/read memory simultaneously
- Can use port A and port B independently and simultaneously

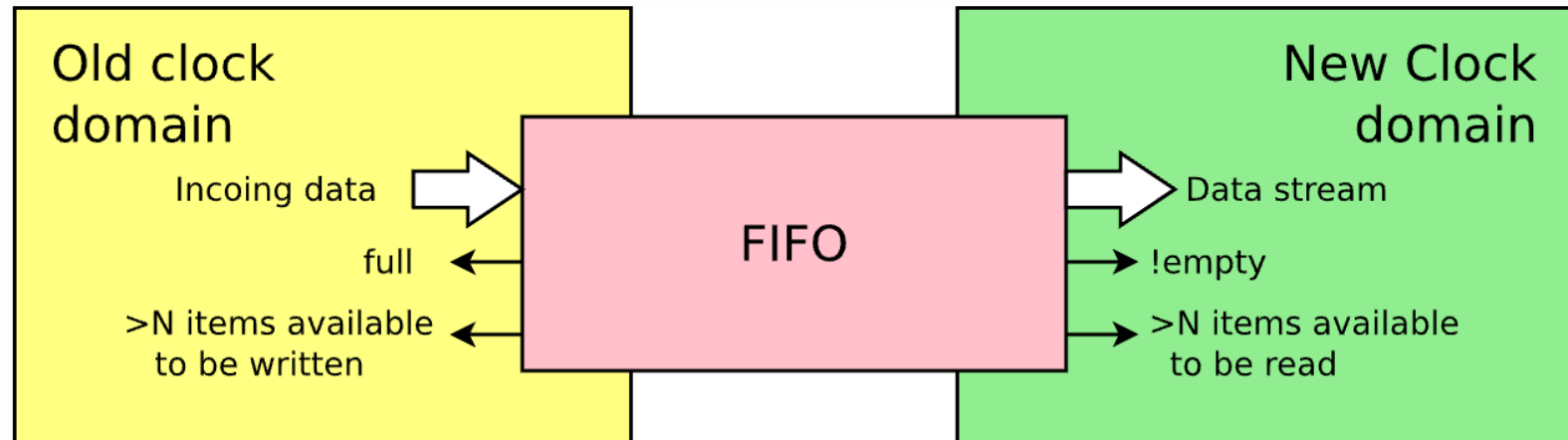
- Simple dual port mode

- Can write/read memory simultaneously.



# Where are BRAM used?

- Used to store information.
- Used to "cross clock domains". (Topic for later)
  - A logic using a certain clock is said to be in a clock domain.
  - Need to be very careful when moving data to different clock domain.



# Where are BRAM used?

- Used to approximate a complex functions  $f(x)$

➤ Complex functions can be difficult to implement with CLB.

➤ We define **BRAM address** to  $x$ .

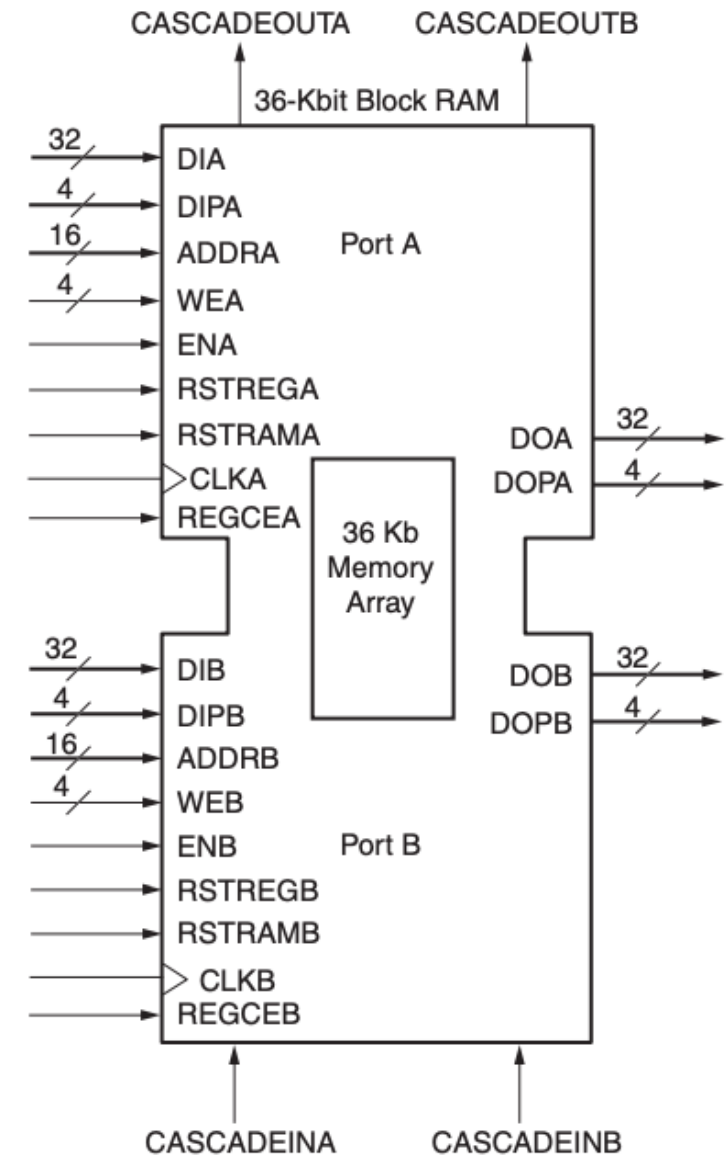
➤ Firmware set to have **initial BRAM data**, which is approx.  $f(x)$

x	Address (3'b fraction) Q1.3	Real cos(x)	Approx. cos(x)	Data (4'b fraction) Q1.4
1	4'b1000	0.540	0.5	5'b01000
0.875	4'b0111	0.641	0.625	5'b01010
0.75	4'b0110	0.731	0.75	5'b01100
0.625	4'b0101	0.811	0.812	5'b01110



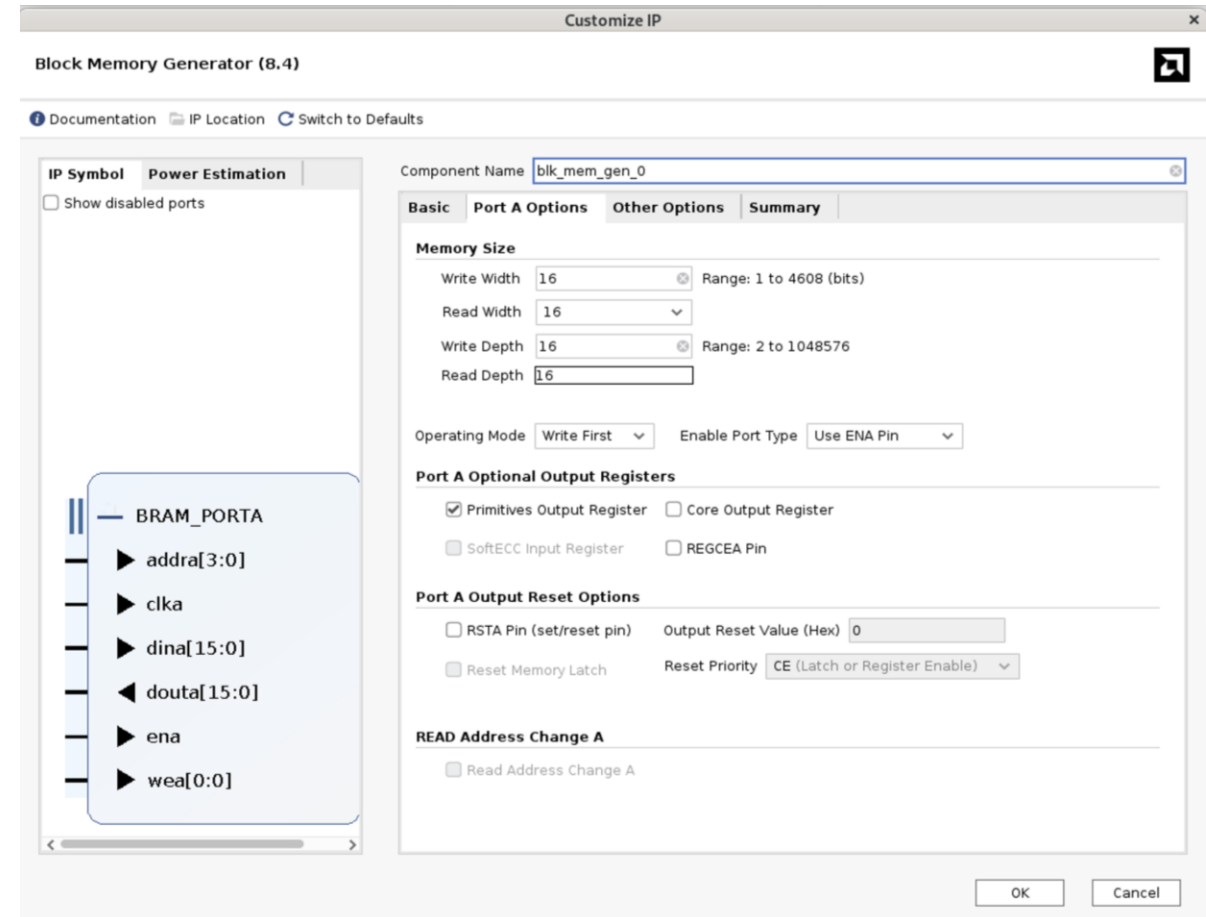
# BRAM settings.

- AMD BRAMs have many settings that change BRAM behavior.
  - True dual port or Simple dual port.
  - When writing data, set BRAM output to written data OR previously stored data OR do not change.
  - ...



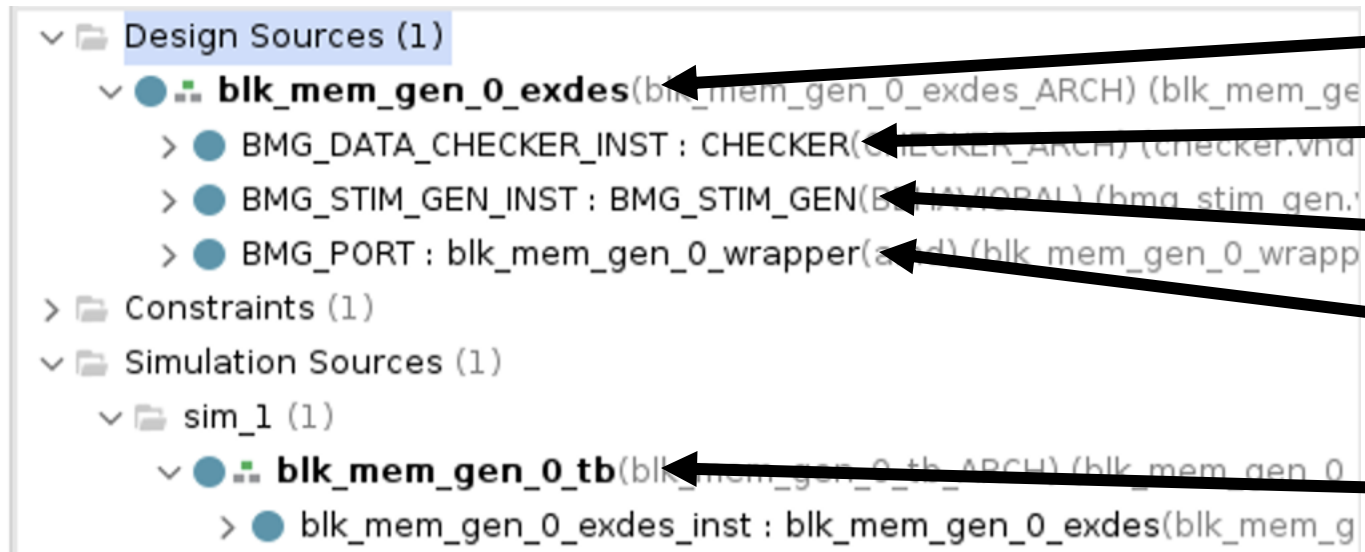
# IPCore for BRAM

- To make it easier to use BRAM, could use AMD IPCore  
(IPCore: Intellectual Property Core)
- There is a AMD IPCore GUI tool to create a memory module.



# Vivado also provides code on how to use IPCore

- Vivado can generate example design project for IPCore
  - HDL will sometimes be Verilog and other times VHDL...
- Generally creates a synthesizable test bench.



Synthesizable test bench

Output data Checker

Input data Stimulus

Design Under Test

Test bench for  
synthesizable test bench

# ILA and VIO IP Cores

- There are commonly / heavily used IP Cores.

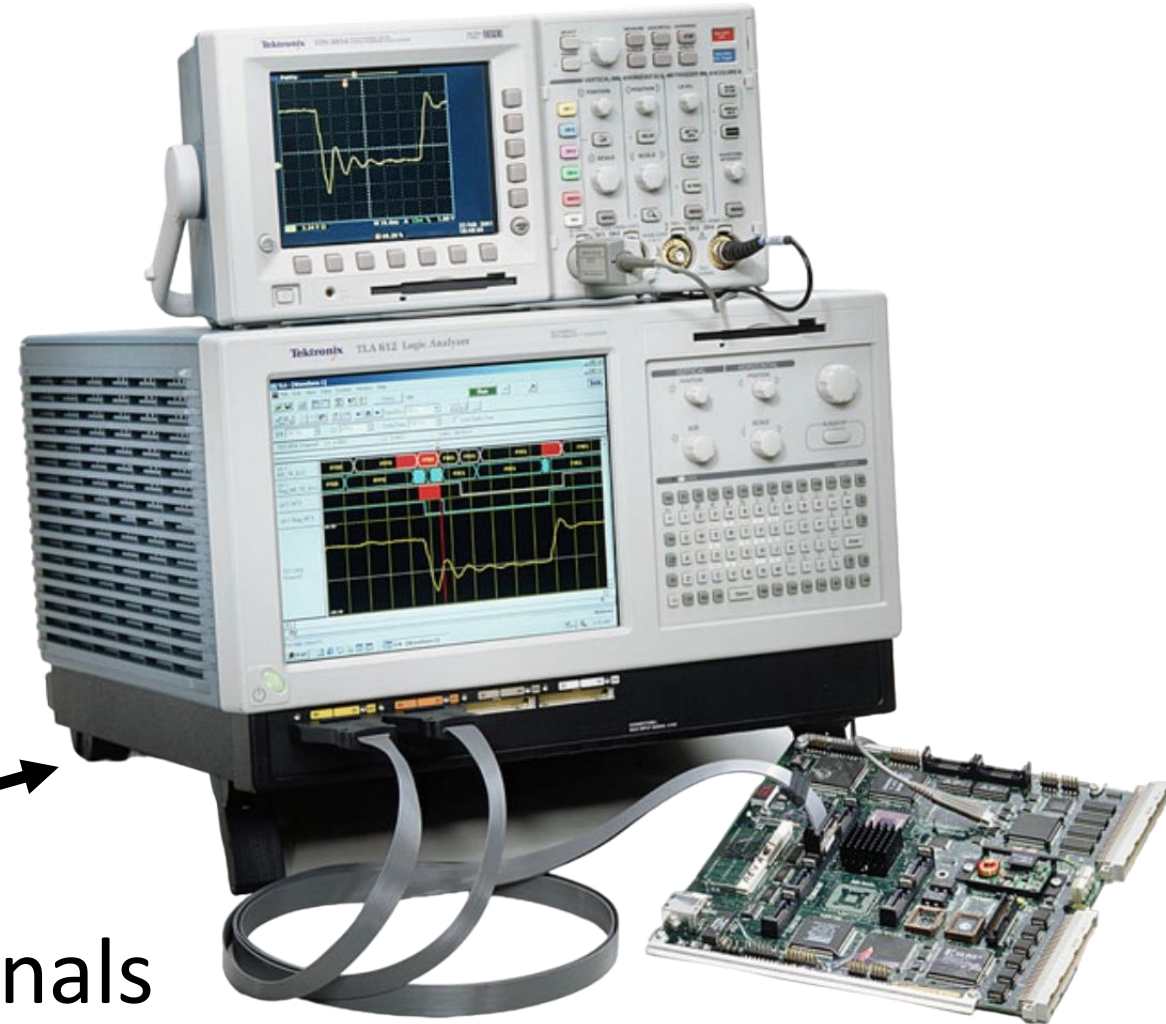
- ILA (Integrated Logic Analyzer)

- VIO (Virtual Input Output)

- What is a logic analyzer?

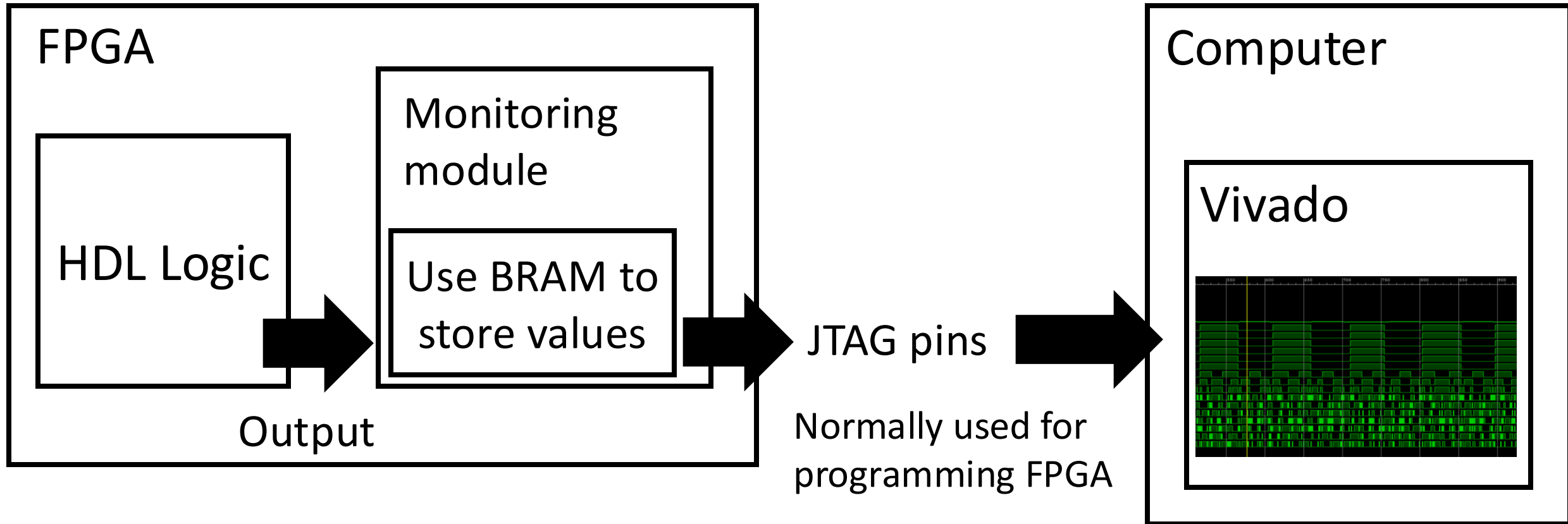
- Monitors many digital logic signals

- Used to debug design logic.



# Integrated Logic Analyzer (ILA) IPCore

- Instead of expensive logic analyzer (\$10,000), could make HDL module (=ILA IPCore) to monitor FPGA logic.



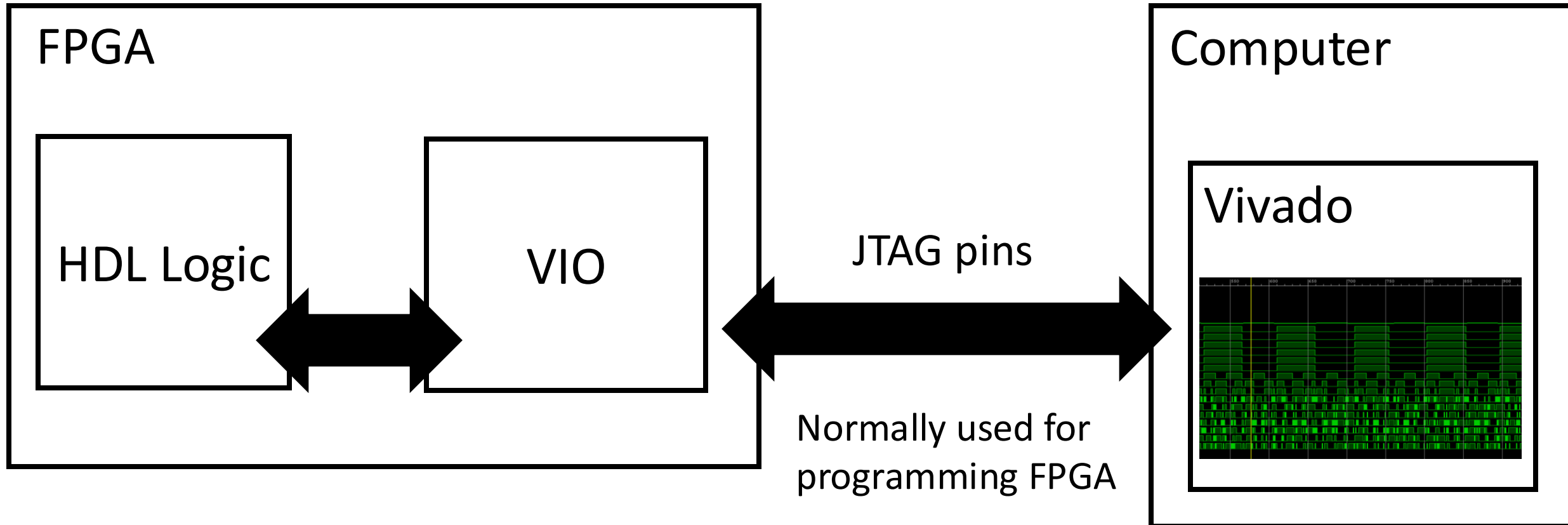
# Virtual I/O (VIO) IPCore

(FPGAs time scale is nanoseconds)

- Similarly can send “slow” data to FPGA (Also receive.)

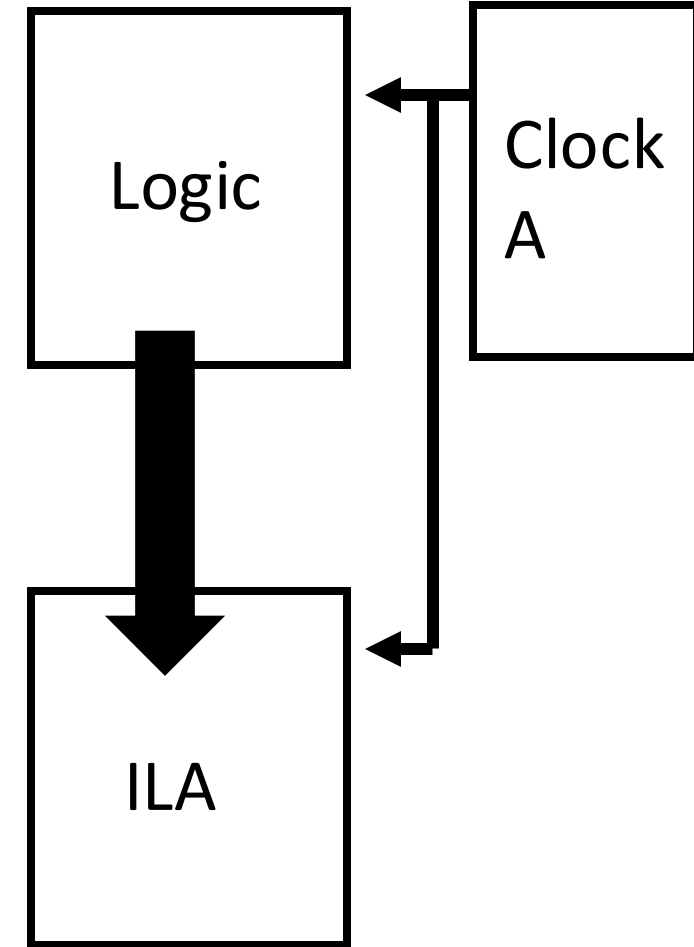
(milliseconds)

➤ Because computers are slow, sent data will be slow.



# ILA and VIO IP Cores

- ILA and VIO used for debugging logic.
- ILA and VIO IP Core receives a clock, where the signals monitored at clock.
  - [Important] If design logic using “clock A”,  
(Do not cross clock domain)  
ILA should also use “clock A”.
  - Can not monitor “clock A” with ILA.



- How much did you understand? [www.kahoot.it](http://www.kahoot.it)