Synchronous FIFO Demonstration

initial				FIFO is empty
		FIFO		
	write pointer = 0		read pointer = 0	

push:	increment the write pointer		
		FIFO	
	write pointer = 1		
		data0	read pointer = 0

pop:	increment the read pointer			FIFO is empty
		FIFO		
	write pointer = 1		read pointer = 1	
		data0		

push:	increment the write pointer		
		FIFO	
	write pointer = 2		
		data1	read pointer = 1
		data0	

push:	increment the write pointer		
		FIFO	
	write pointer = 3		
		data2	
		data1	read pointer = 1
		data0	

push:	increment the write pointer			
		FIFO		
	write pointer = 4			
		data3		
		data2		
		data1	read pointer = 1	
		data0		

push:	increment the write pointer		
		FIFO	
	write pointer = 5		
		data4	
		data3	
		data2	
		data1	read pointer = 1
		data0	

push:	increment the write pointer		
		FIFO	
	write pointer = 6		
		data5	
		data4	
		data3	
		data2	
		data1	read pointer = 1
		data0	

push:	increment the write pointer		
		FIFO	
	write pointer = 7		
		data6	
		data5	
		data4	
		data3	
		data2	
		data1	read pointer = 1
		data0	

push:	increment the write pointer			
		FIFO		
		data7		
		data6		
		data5		
		data4		
		data3		
		data2		
		data1	read pointer = 1	
	write pointer = 0	data0		

push:	increment the write pointer			FIFO is full
		FIFO		
		data7		
		data6		
		data5		
		data4		
		data3		
		data2		
	write pointer = 1	data1	read pointer = 1	
		data0		

pop:	increment the read pointer		
		FIFO	
		data7	
		data6	
		data5	
		data4	
		data3	
		data2	read pointer = 2
	write pointer = 1	data1	
		data0	

pop:	increment the read pointer		
		FIFO	
		data7	
		data6	
		data5	
		data4	
		data3	read pointer = 3
		data2	
	write pointer = 1	data1	
		data0	

pop:	increment the read pointer			
		FIFO		
		data7		
		data6		
		data5		
		data4	read pointer = 4	
		data3		
		data2		
	write pointer = 1	data1		
		data0		

pop:	increment the read pointer			
		FIFO		
		data7		
		data6		
		data5	read pointer = 5	
		data4		
		data3		
		data2		
	write pointer = 1	data1		
		data0		

pop:	increment the read pointer		
		FIFO	
		data7	
		data6	read pointer = 6
		data5	
		data4	
		data3	
		data2	
	write pointer = 1	data1	
		data0	

pop:	increment the read pointer			
		FIFO		
		data7	read pointer = 7	
		data6		
		data5		
		data4		
		data3		
		data2		
	write pointer = 1	data1		
		data0		

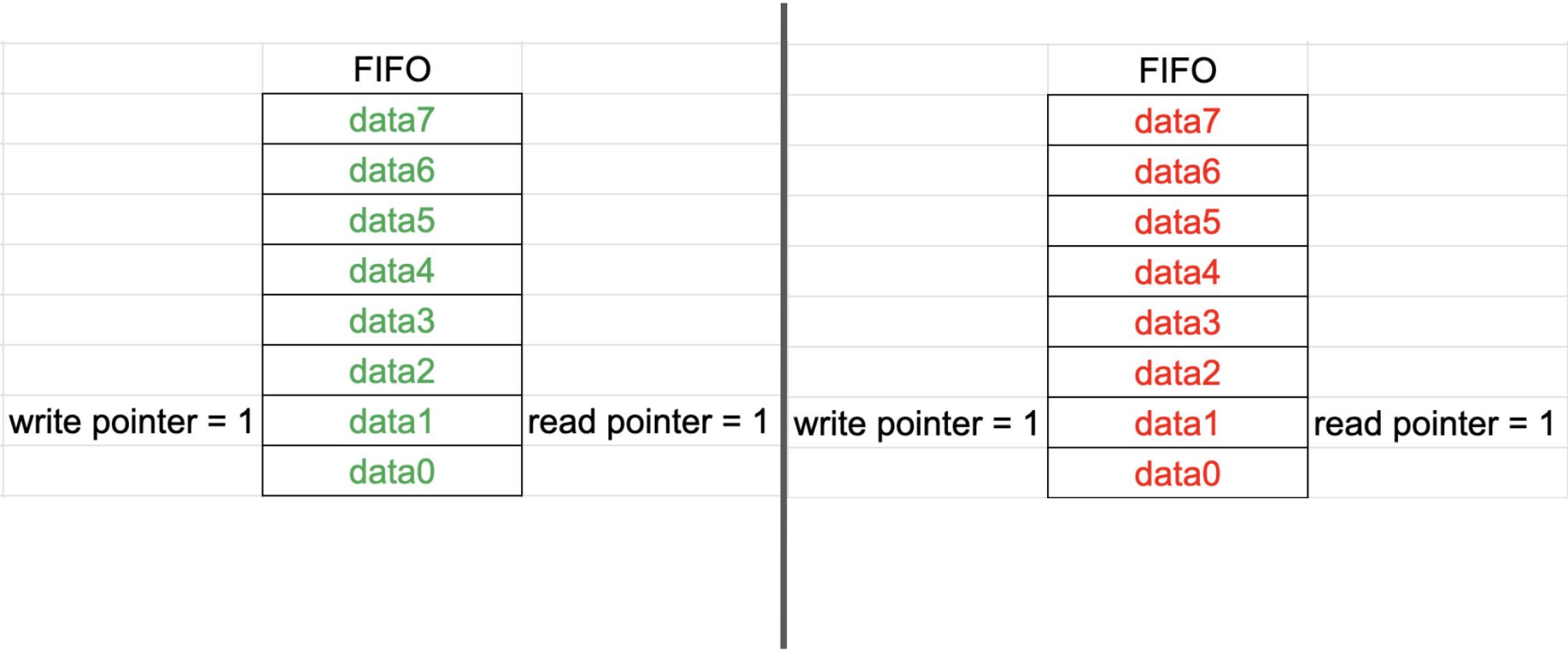
pop:	increment the read pointer			
		FIFO		
		data7		
		data6		
		data5		
		data4		
		data3		
		data2		
	write pointer = 1	data1		
		data0	read pointer = 0	

pop:	increment the read pointer			FIFO is empty
		FIFO		
		data7		
		data6		
		data5		
		data4		
		data3		
		data2		
	write pointer = 1	data1	read pointer = 1	
		data0		

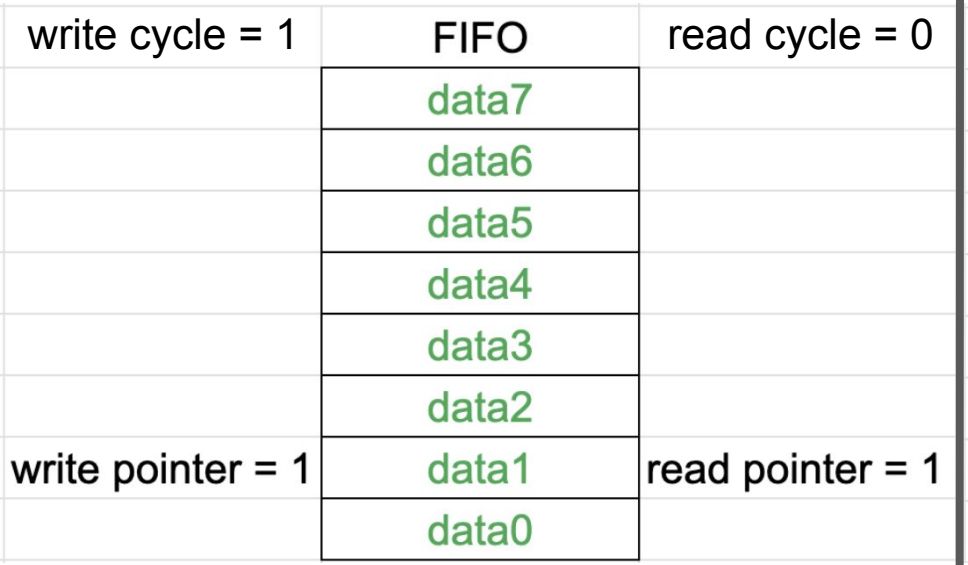
push:	increment the read pointer		
		FIFO	
		data7	
		data6	
		data5	
		data4	
		data3	
	write pointer = 2	data2	
		data1	read pointer = 1
		data0	

push AND pop:	increment both pointers			
		FIFO		
		data7		
		data6		
		data5		
		data4		
	write pointer = 3	data3		
		data2	read pointer = 2	
		data1		
		data0		

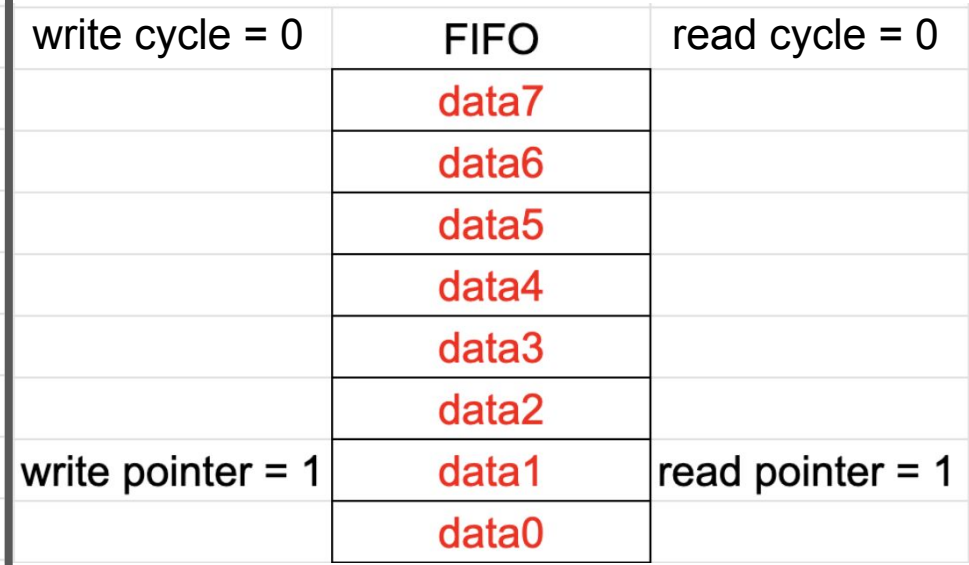
Full or Empty?



Solution: use an extra bit to which cycle the pointer is on



FULL when write_cycle != read_cycle



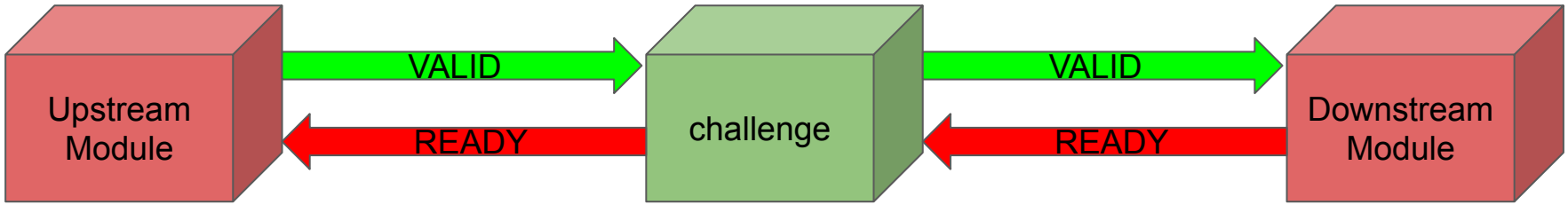
EMPTY when write_cycle == read_cycle

SystemVerilog Microarchitecture Challenge

10/26/2025

The Challenge

- Write a pipelined block that computes: **a**5 + 0.3*b - c**
- You may ONLY use pre-existing blocks for arithmetic operations
- Your design must achieve a throughput of 1 output per cycle when there is no backpressure
- Check your solution against a pre-existing testbench



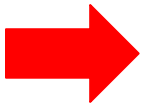
Arithmetic Modules

- You may ONLY use pre-existing blocks for arithmetic operations

```
assign  res = a + b;
```

Not allowed

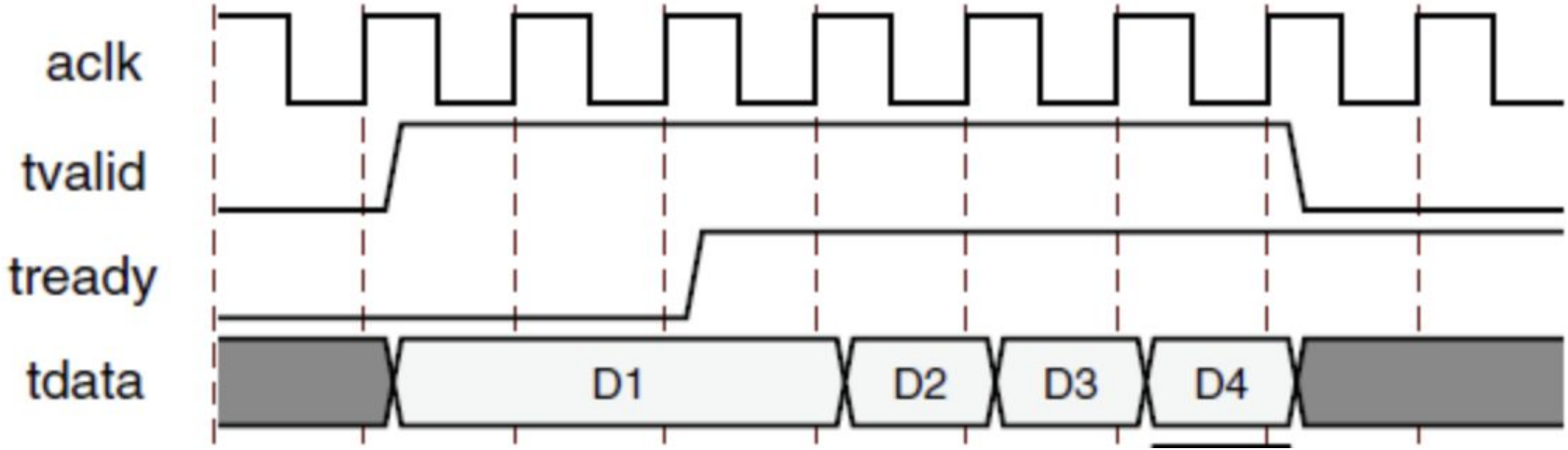
```
module f_add (
    input                 clk,
    input                 rst,
    input  [FLEN - 1:0] a, b,
    input                 up_valid,
    output [FLEN - 1:0] res,
    output                down_valid,
    output                busy,
    output                error
);
```



Instantiation of f_add

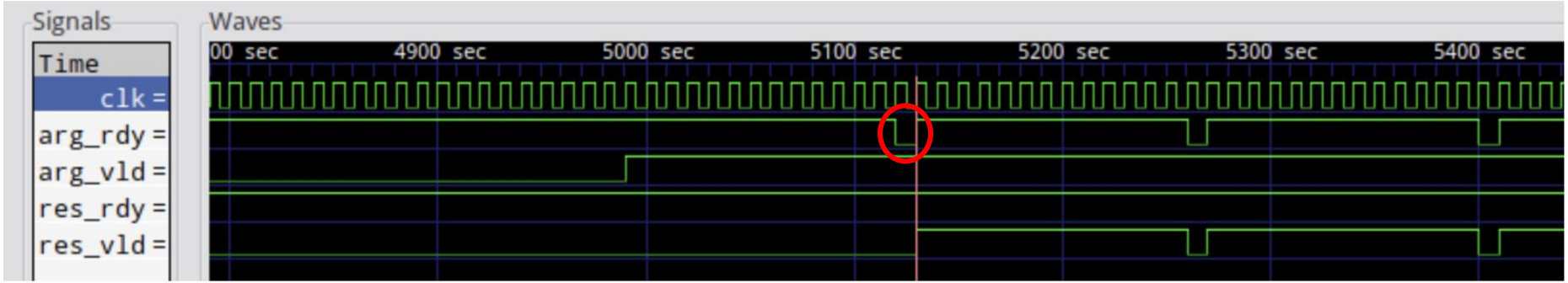
```
f_add a_plus_b (
    .clk        ( clk          ),
    .rst        ( rst          ),
    .up_valid   ( up_valid     ),
    .a          ( a            ),
    .b          ( b            ),
    .down_valid ( down_valid   ),
    .res        ( res          )
);
```

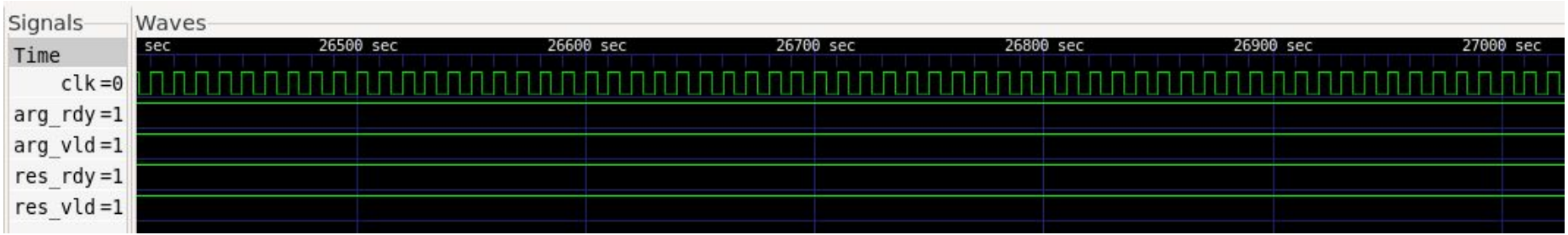
Valid/Ready from AXI-Stream



https://docs.amd.com/r/en-US/pg256-sdfec-integrated-block/AXI4-Stream-Interface

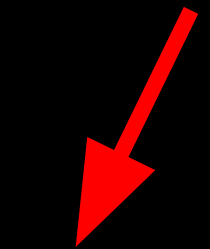
Throughput 1 Output per Cycle and No Empty Cycle Gaps





Empty Cycle Gap (bubble) from simulation.log

```
********** Constraint random values back-to-back
testbench.sv time   26200 cycle  2619     arg a:24508.6 b:10827.4 c:758.144
testbench.sv time   26210 cycle  2620     arg a:8378.34 b:22603 c:33365.4
testbench.sv time   26220 cycle  2621     arg a:8516.09 b:1546.2 c:21634.7
testbench.sv time   26230 cycle  2622     arg a:22622.9 b:31396.9 c:26600.9
testbench.sv time   26240 cycle  2623     arg a:41411.1 b:33249 c:42445
testbench.sv time   26250 cycle  2624     arg a:16645.6 b:16603.9 c:14273.6
testbench.sv time   26260 cycle  2625     arg a:40716.2 b:10036.5 c:1543.27
testbench.sv time   26270 cycle  2626     arg a:33541.9 b:41807 c:17358.3
testbench.sv time   26280 cycle  2627     arg a:19461.9 b:16574.3 c:29250.2
testbench.sv time   26290 cycle  2628     arg a:15863.7 b:3803.28 c:8498.15
testbench.sv time   26300 cycle  2629     arg a:8377.72 b:22389.4 c:10776.2
testbench.sv time   26310 cycle  2630     arg a:24224.8 b:35814.3 c:13220.4
testbench.sv time   26320 cycle  2631     arg a:12604 b:1596.27 c:1129.98
testbench.sv time   26330 cycle  2632     arg a:6988.65 b:30471.5 c:17892.7 NOT READY arg_rdy:0
testbench.sv time   26340 cycle  2633     arg a:6988.65 b:30471.5 c:17892.7     res:8.84292e+21
```



No Bubble

```
********** Constraint random values back-to-back
testbench.sv time   26200 cycle  2619     arg a:24508.6 b:10827.4 c:758.144
testbench.sv time   26210 cycle  2620     arg a:8378.34 b:22603 c:33365.4
testbench.sv time   26220 cycle  2621     arg a:8516.09 b:1546.2 c:21634.7
testbench.sv time   26230 cycle  2622     arg a:22622.9 b:31396.9 c:26600.9
testbench.sv time   26240 cycle  2623     arg a:41411.1 b:33249 c:42445
testbench.sv time   26250 cycle  2624     arg a:16645.6 b:16603.9 c:14273.6
testbench.sv time   26260 cycle  2625     arg a:40716.2 b:10036.5 c:1543.27
testbench.sv time   26270 cycle  2626     arg a:33541.9 b:41807 c:17358.3
testbench.sv time   26280 cycle  2627     arg a:19461.9 b:16574.3 c:29250.2
testbench.sv time   26290 cycle  2628     arg a:15863.7 b:3803.28 c:8498.15
testbench.sv time   26300 cycle  2629     arg a:8377.72 b:22389.4 c:10776.2
testbench.sv time   26310 cycle  2630     arg a:24224.8 b:35814.3 c:13220.4
testbench.sv time   26320 cycle  2631     arg a:12604 b:1596.27 c:1129.98
testbench.sv time   26330 cycle  2632     arg a:6988.65 b:30471.5 c:17892.7
testbench.sv time   26340 cycle  2633     arg a:183.035 b:14652.7 c:23991.4     res:8.84292e+21
```

Getting Started

- Break the problem down as much as you can
- You can ask ChatGPT how to implement commonly seen modules (ie. FIFO)
- Treat it as co-pilot: double check the generated code to verify that it makes sense

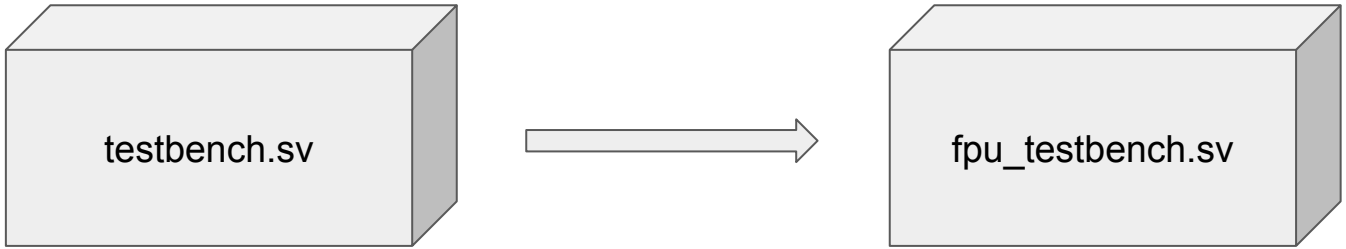
Solution

Approaching the Challenge

1. Submodule latencies
2. Calculate the formula
3. Add flow control

Finding the Latencies of Submodules

- Use what's provided!



Checks a**5 + 0.3*b - c

Modify to check:
a*b
a+b
a-b

tesbench.sv

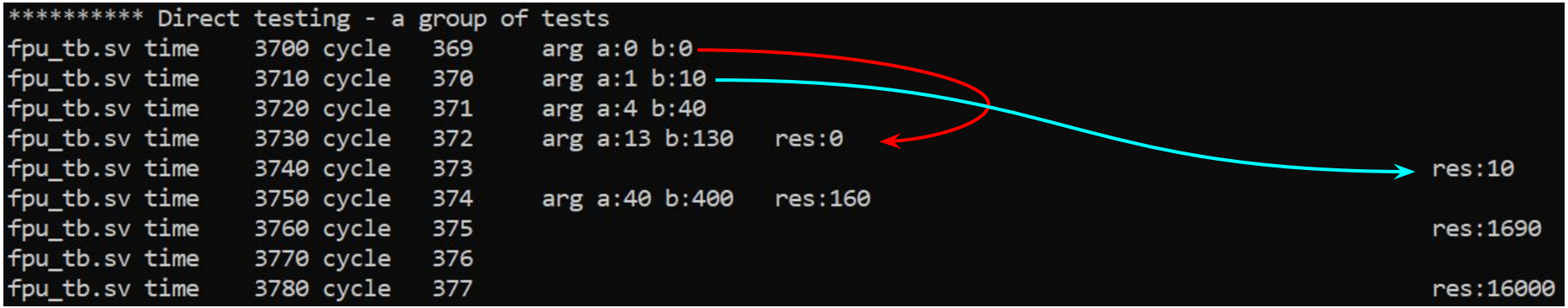
```
module testbench;

    //------------------------------------------------------------------------
    // Signals to drive Device Under Test - DUT

    logic               clk;
    logic               rst;

    logic               arg_vld;
    wire                arg_rdy;

    logic  [FLEN - 1:0] a;
    logic  [FLEN - 1:0] b;
    logic  [FLEN - 1:0] c;

    wire                res_vld;
    logic               res_rdy;

    wire   [FLEN - 1:0] res;

    //------------------------------------------------------------------------

    // If we change FLEN to 32, we have to change these constants

    localparam [FLEN - 1:0] inf     = 64'h7FF0_0000_0000_0000,
                            neg_inf = 64'hFFF0_0000_0000_0000,
                            zero    = 64'h0000_0000_0000_0000,
                            nan     = 64'h7FF1_2345_6789_ABCD;

    //------------------------------------------------------------------------
    // Instantiating DUT

    challenge dut (.*);

    //------------------------------------------------------------------------
```

fpu_tb.sv

I/O are different!

```
module fpu_tb (
);

reg     clk;
reg     rst;

reg     up_valid;
reg     [FLEN - 1:0]    a;
reg     [FLEN - 1:0]    b;

wire    [FLEN - 1:0]    res;
wire    down_valid, busy, error;

//wire    arg_rdy  = ~busy;
wire    arg_rdy  = 1;
//------------------------------------------------------------------------

// If we change FLEN to 32, we have to change these constants

localparam [FLEN - 1:0] inf     = 64'h7FF0_0000_0000_0000,
                        neg_inf = 64'hFFF0_0000_0000_0000,
                        zero    = 64'h0000_0000_0000_0000,
                        nan     = 64'h7FF1_2345_6789_ABCD;
//------------------------------------------------------------------------
// Instantiating DUT

f_mult dut (.*);
//f_sub dut (.*);
//f_add dut (.*);

//------------------------------------------------------------------------
```

Output of simulation



3 cycles to produce output after the argument is accepted

f_mult - 3 cycle latency

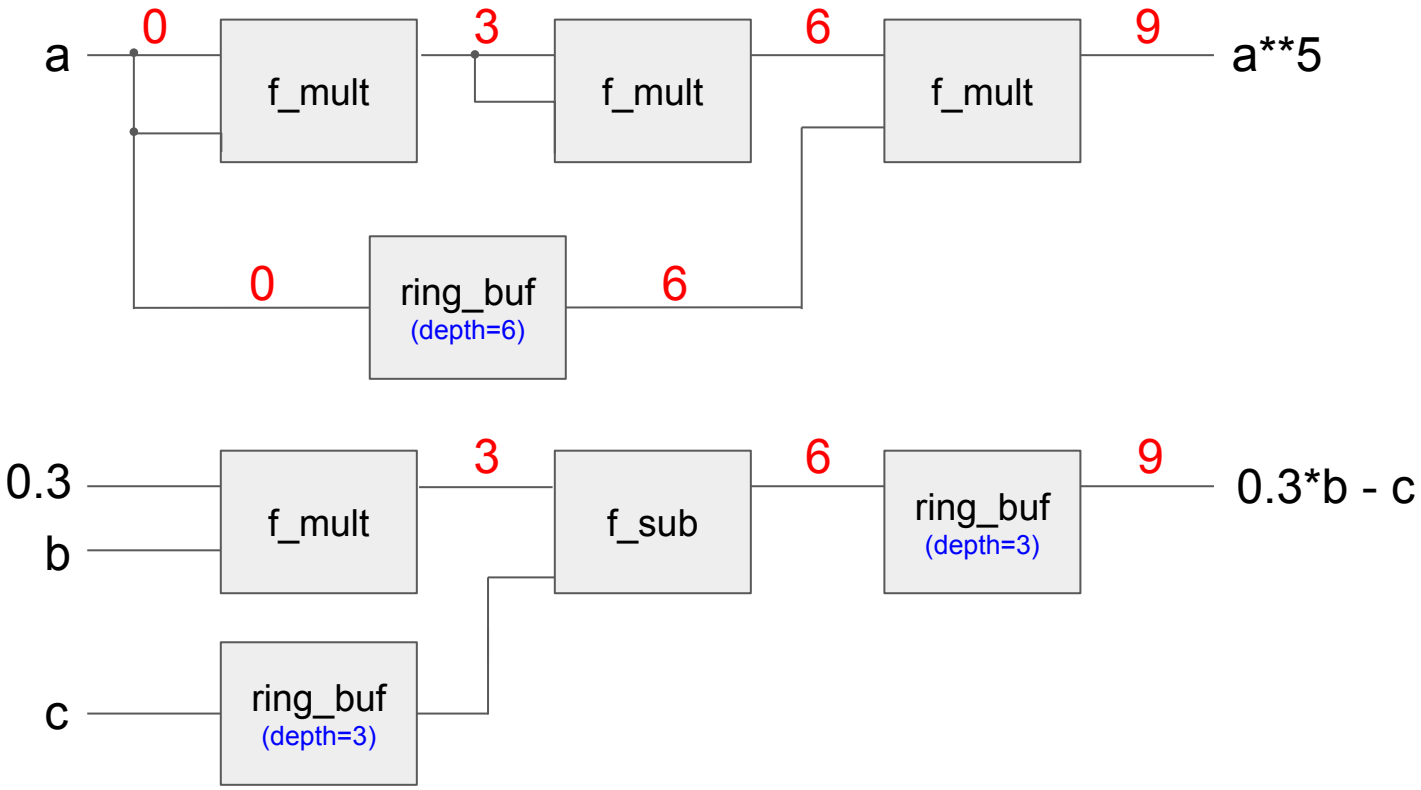
f_sub - 3 cycle latency

f_add - 4 cycle latency

Cycle #

Designing a Suitable Architecture

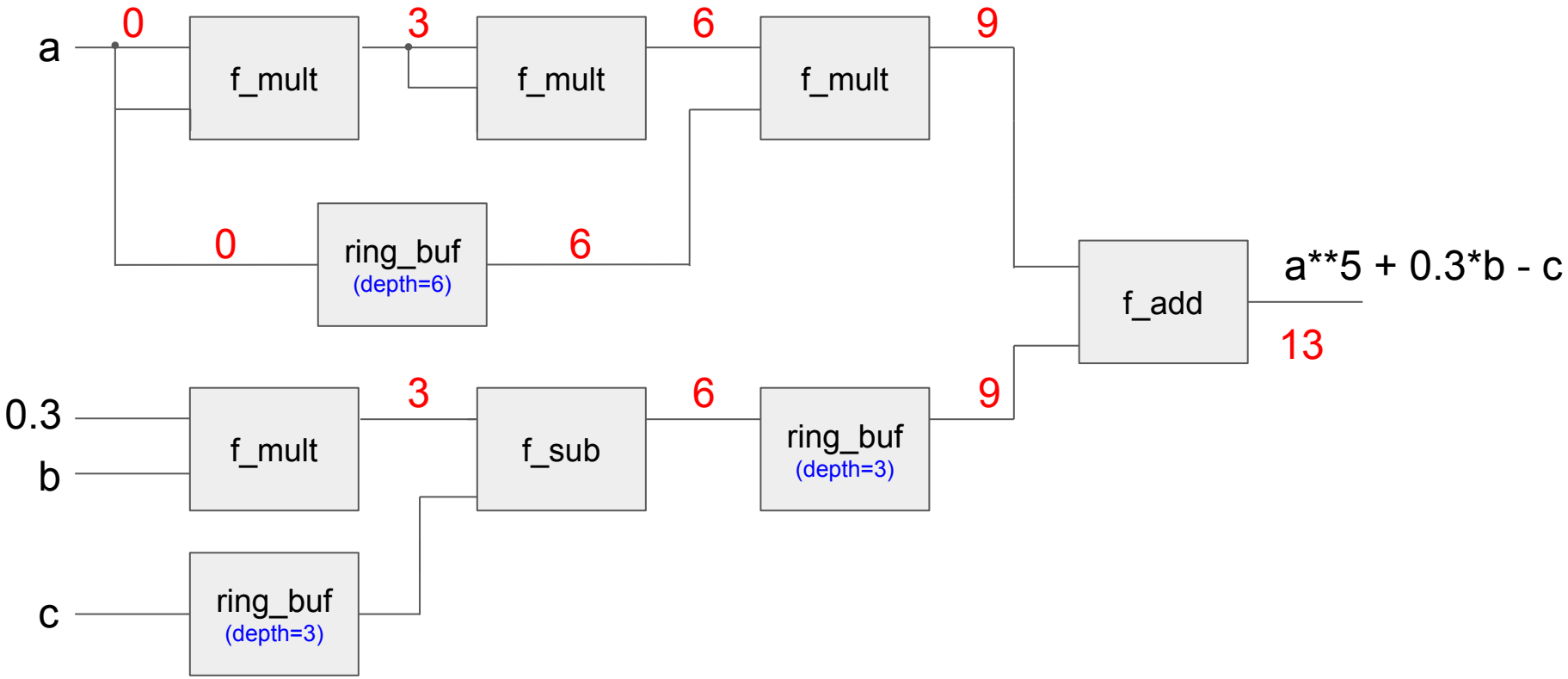
a**5 + 0.3*b - c



Cycle #

Designing a Suitable Architecture

a**5 + 0.3*b - c



Adding Flow Control