

# Phase Detector

David

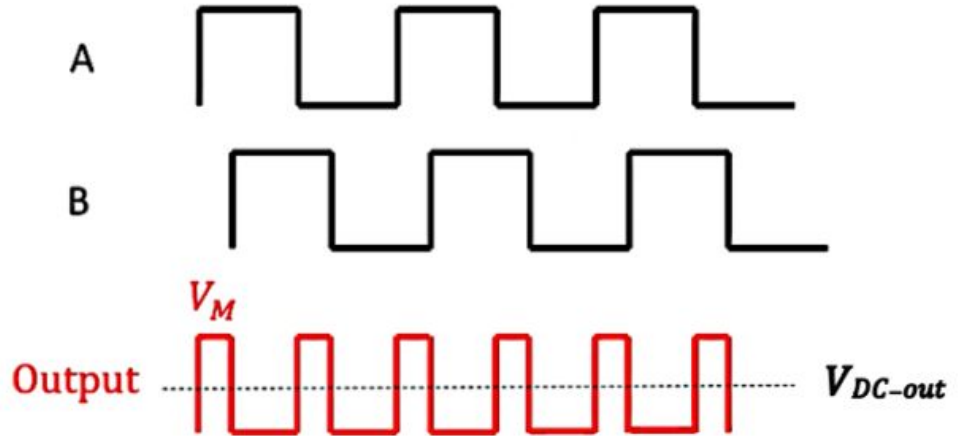
Nov. 26, 2025



# Contents

- Phase Detector (PD)
  - XOR, Phase-Frequency Discriminator (PFD)
- Application: Phase-Locked Loop (PLL)
- LTspice
- ModelSim

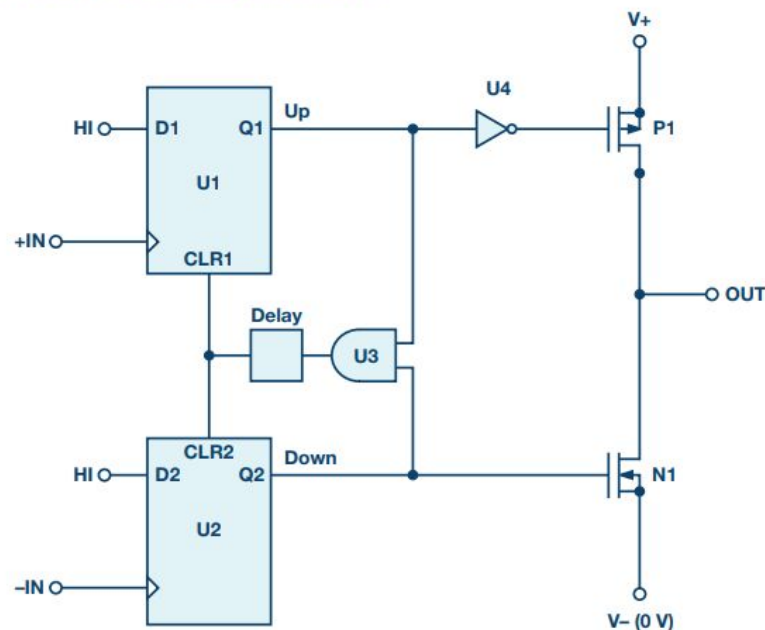
# XOR Phase Detector



# Phase-Frequency Discriminator (PFD)

- A phase-frequency discriminator is a device that converts phase or frequency differences between two input signals into an amplitude output.
- It can compare the phase of two signals or, if they are far apart in frequency, compare their frequency, converting the difference into an amplitude signal.
- This conversion can be achieved through various electronic or optical methods that use interference, delay lines, or flip-flops.

Phase Frequency Detector

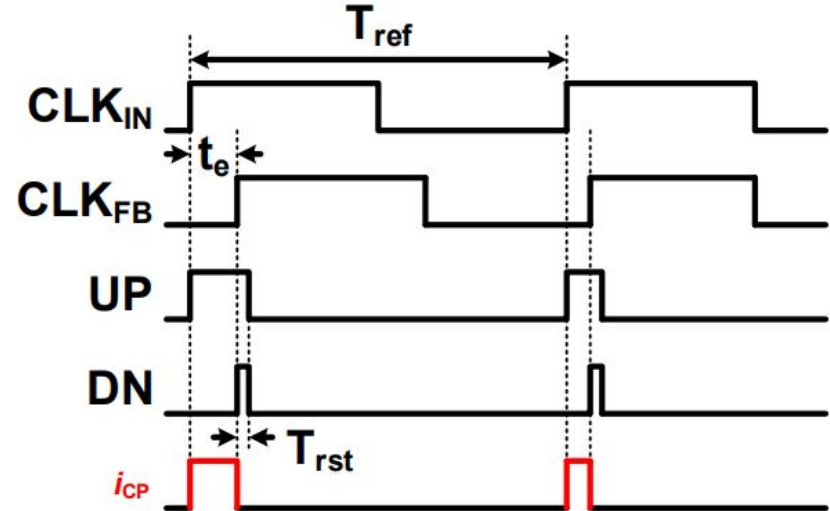
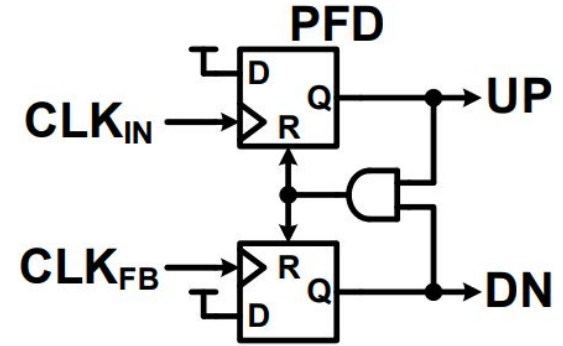
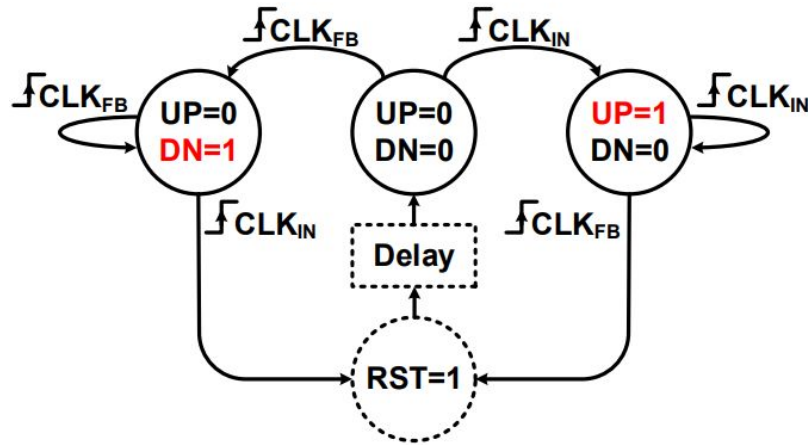


Source:

<https://www.analog.com/media/en/analog-dialogue/volume-52/number-3/phase-locked-loop-pll-fundamentals.pdf>



# Phase-Frequency Discriminator (PFD)



# Phase-Locked Loop (PLL)

A phase-locked loop (PLL) detector is a feedback control system that generates an output signal with a phase that is synchronized to an input reference signal. It works by comparing the phase difference between a reference signal and a feedback signal from a **voltage-controlled oscillator (VCO)**. The difference generates an error signal that is filtered and used to adjust the VCO, locking the output signal's phase and frequency to the input.

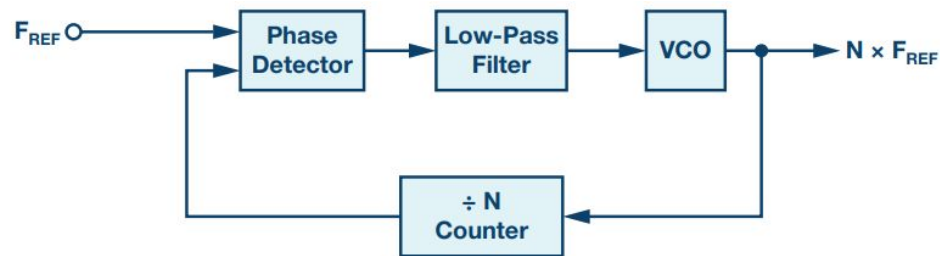
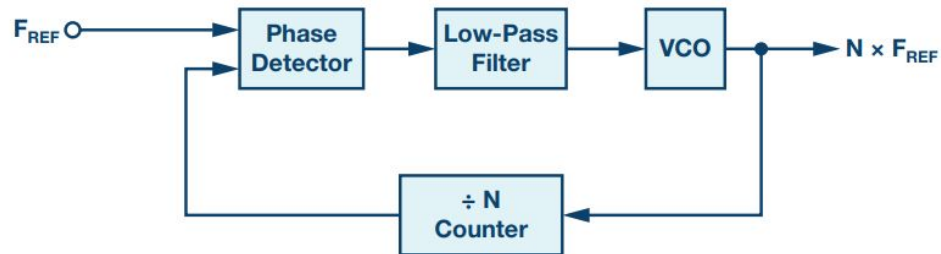


Figure 2. Basic PLL configuration.

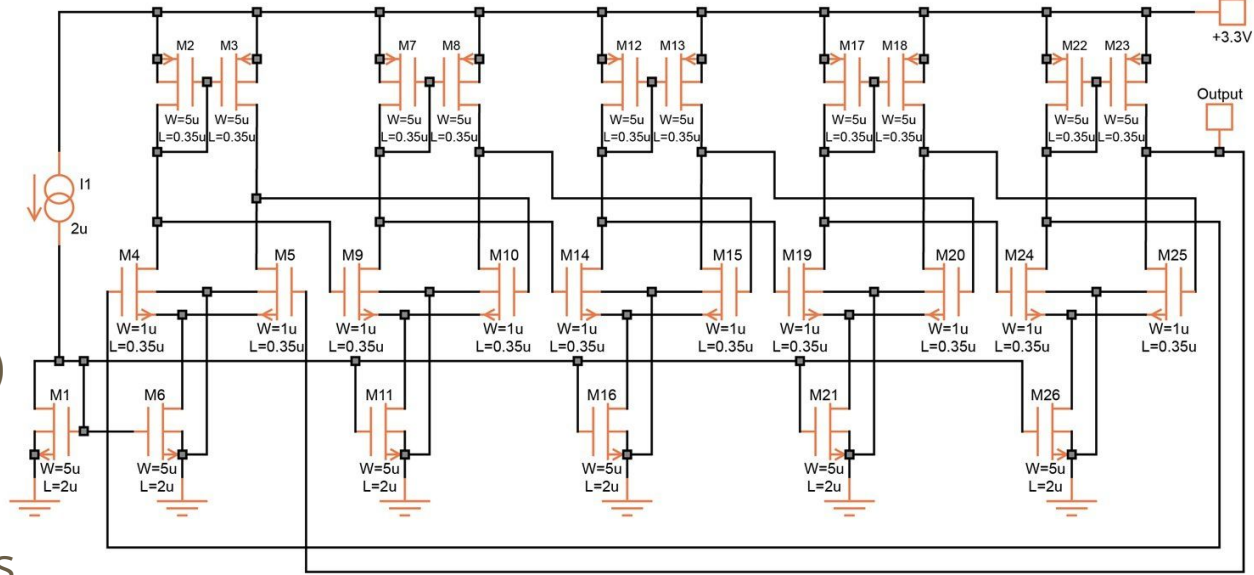
# How a PLL detector works?



- **Phase Detector:** This component receives the input reference signal and the feedback signal from the VCO. It outputs an **error** signal that represents the phase and frequency difference between the two.
- **Loop Filter:** The error signal from the phase detector is smoothed by a **low-pass filter**. This filter removes high-frequency noise and produces a stable DC voltage.
- **Voltage-Controlled Oscillator (VCO):** This is the source of the output signal. The stable voltage from the loop filter is used to control the frequency of the VCO.
- **Feedback Loop:** The output signal from the VCO is fed back into the phase detector, creating a continuous feedback loop that adjusts the VCO's frequency until its output is locked to the phase and frequency of the input reference signal.

# Ring Oscillator

- A ring oscillator is a circuit made of an **odd** number of inverters (**NOT** gates) connected in a ring, where the output of the last inverter feeds back into the first.



[Figure] Current-controlled ring oscillator for operation between 6 MHz and 300 MHz.

<https://www.allaboutcircuits.com/textbook/designing-analog-chips/phase-locked-loops/voltage-controlled-oscillators-for-plls/>

- This setup creates a continuous oscillation where the output signal flips between a high and low voltage level, as there is no stable state in an odd-number ring.
- The frequency of the oscillation is determined by the number of inverters and their individual propagation delays.



# Simulation

- LTspice
- ModelSim

---



- **SPICE**, standing for "**Simulation Program with Integrated Circuit Emphasis**," is a computer program that simulates the behavior of electronic circuits, enabling engineers to analyze and test them virtually before building them physically.
- **LTspice** is a free **SPICE simulator** software from **Analog Devices** that is used to design, test, and analyze electronic circuits before building physical prototypes.

# PLL in LTspice (\*.asc)

```
61 SYMBOL DigitalLogic\Xor 48 128 R0
62 WINDOW 0 -28 54 Left 2
63 SYMATTR InstName U1
64 SYMBOL voltage -176 368 R0
65 WINDOW 123 0 0 Left 2
66 WINDOW 39 0 0 Left 2
67 SYMATTR InstName V1
68 SYMATTR Value PULSE(0 5 0 100n 100n .1m .2m 500)
69 SYMBOL SpecialFunctions\LTC6900 736 160 R0
70 SYMATTR InstName U3
71 SYMBOL voltage 640 -144 R0
72 WINDOW 123 0 0 Left 2
73 WINDOW 39 0 0 Left 2
74 SYMATTR InstName V2
75 SYMATTR Value 5
76 SYMBOL bi 576 400 R180
77 WINDOW 0 24 80 Left 2
78 WINDOW 3 24 0 Left 2
79 SYMATTR InstName B1
80 SYMATTR Value I=V(ctrl)*6.2u
81 SYMBOL res 240 160 R90
82 WINDOW 0 0 56 VBottom 2
83 WINDOW 3 32 56 VTop 2
84 SYMATTR InstName R1
85 SYMATTR Value 500
86 SYMBOL cap 272 208 R0
87 SYMATTR InstName C1
88 SYMATTR Value 100n
89 TEXT 840 472 Left 2 !.tran 100m
```

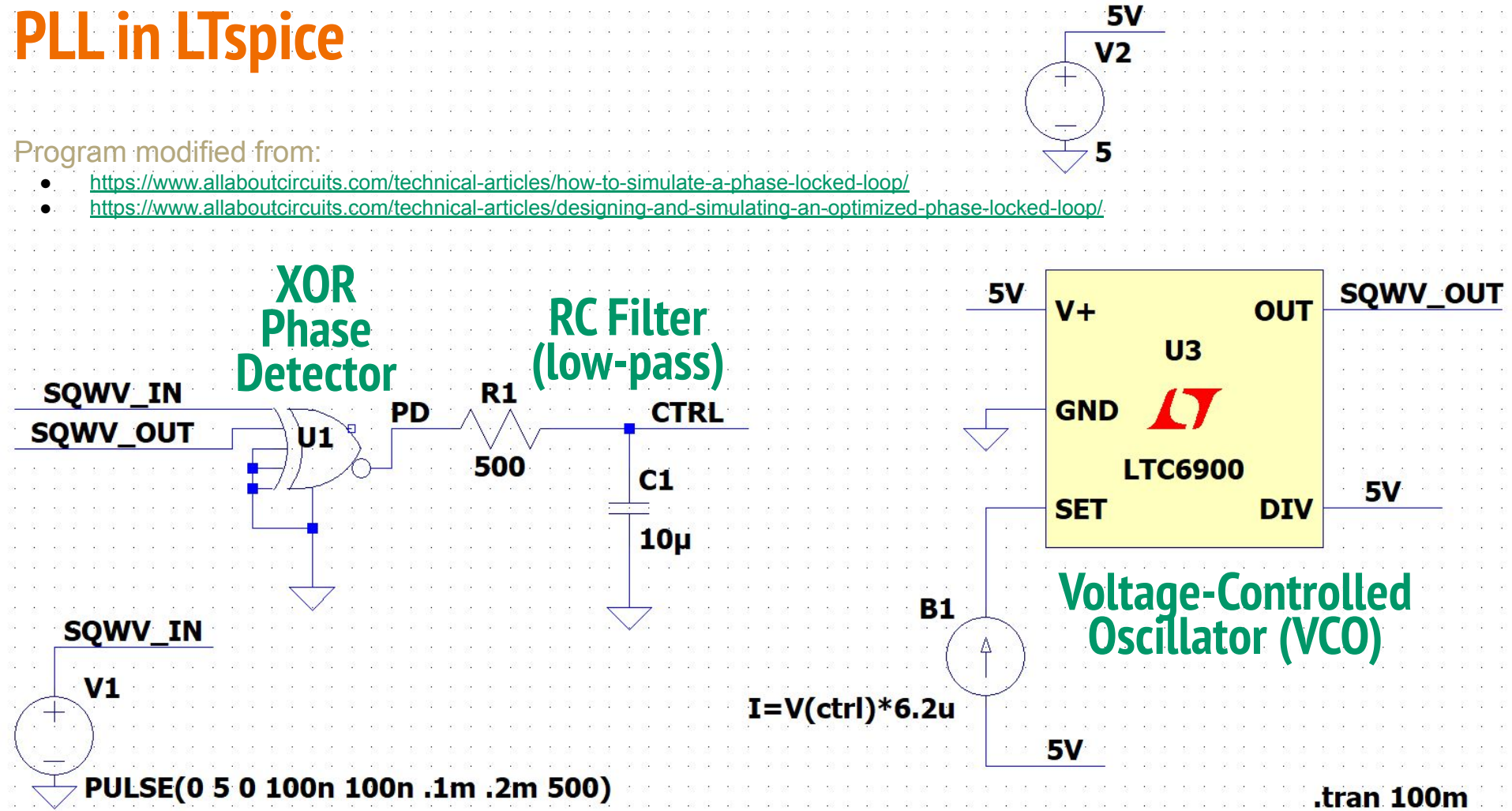
Program modified from:

- <https://www.allaboutcircuits.com/technical-articles/how-to-simulate-a-phase-locked-loop/>
- <https://www.allaboutcircuits.com/technical-articles/designing-and-simulating-an-optimized-phase-locked-loop/>

# PLL in LTspice

Program modified from:

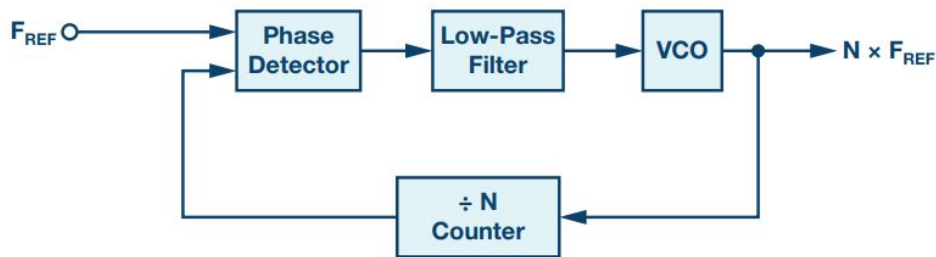
- <https://www.allaboutcircuits.com/technical-articles/how-to-simulate-a-phase-locked-loop/>
- <https://www.allaboutcircuits.com/technical-articles/designing-and-simulating-an-optimized-phase-locked-loop/>



# PLL in LTspice

The LTC6900 is a silicon oscillator that produces a precision CMOS **square-wave**; a single external resistor programs its frequency across a wide range (commonly specified as 1 kHz to 20 MHz).

Assume the source  
**SQWV\_IN** with 5 kHz



Independent Voltage Source - V1

Functions

- ☐ (none)
- ☒ PULSE(V1 V2 Tdelay Trise Tfall Ton Period Ncycles)
- ☐ SINE(Voffset Vamp Freq Td Theta Phi Ncycles)
- ☐ EXP(V1 V2 Td1 Tau1 Td2 Tau2)
- ☐ SFFM(Voff Vamp Fcar MDI Fsig)
- ☐ PWL(t1 v1 t2 v2...)
- ☐ PWL FILE:

Vinitial[V]:	0
Von[V]:	5
Tdelay[s]:	0
Trise[s]:	100n
Tfall[s]:	100n
Ton[s]:	.1m
Tperiod[s]:	.2m
Ncycles:	500

Smooth half-sine transitions ☐

Make this information visible on schematic: ☒

SQWV\_IN

V1

PULSE(0 5 0 100n 100n .1m .2m 500)

# PLL in LTspice (\*.net)

\* Generated by LTspice 24.1.10 for Windows.

A\$U1 SQWV\_IN SQWV\_OUT 0 0 0 PD 0 0 XOR

V1 SQWV\_IN 0 PULSE(0 5 0 100n 100n .1m .2m 500)

X\$U3 5V 0 N001 5V SQWV\_OUT LTC6900 (\$pnba V+)GND)SET)DIV)OUT

V2 5V 0 5

B1 5V N001 I=V(ctrl)\*6.2u

R1 CTRL PD 500

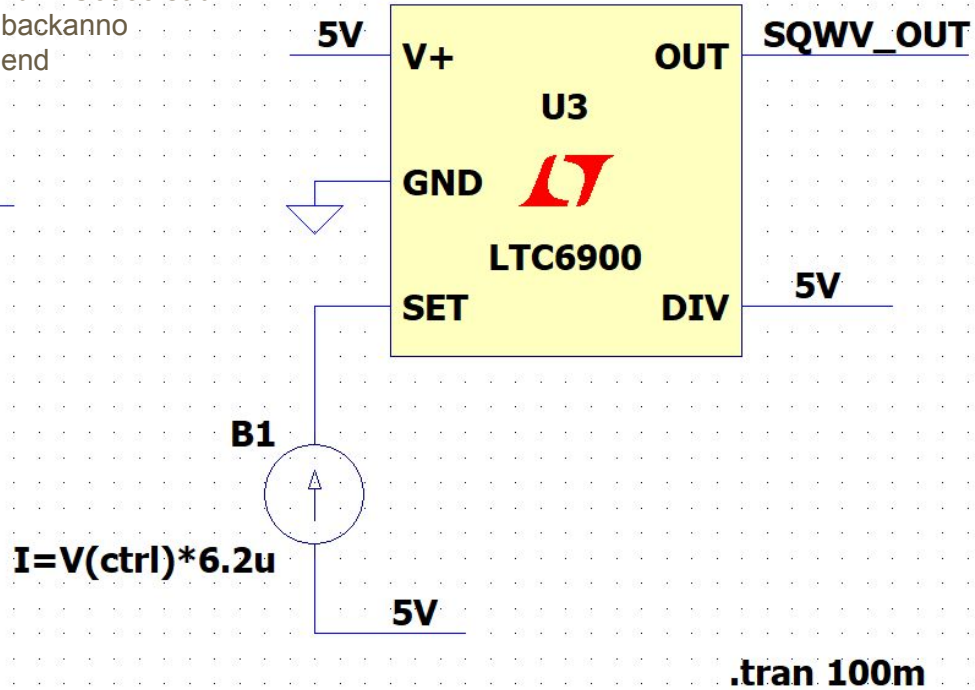
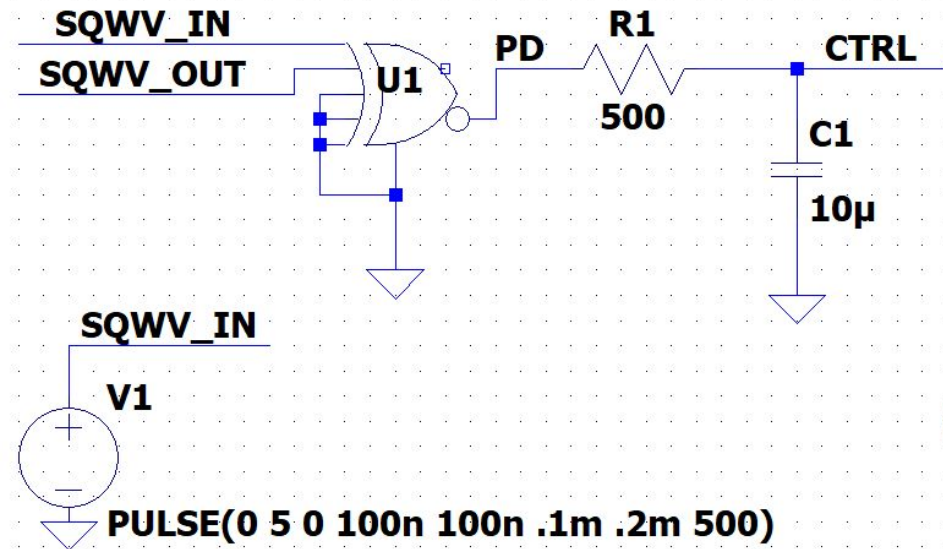
C1 CTRL 0 10μ

.tran 100m

.lib LTC6900.sub

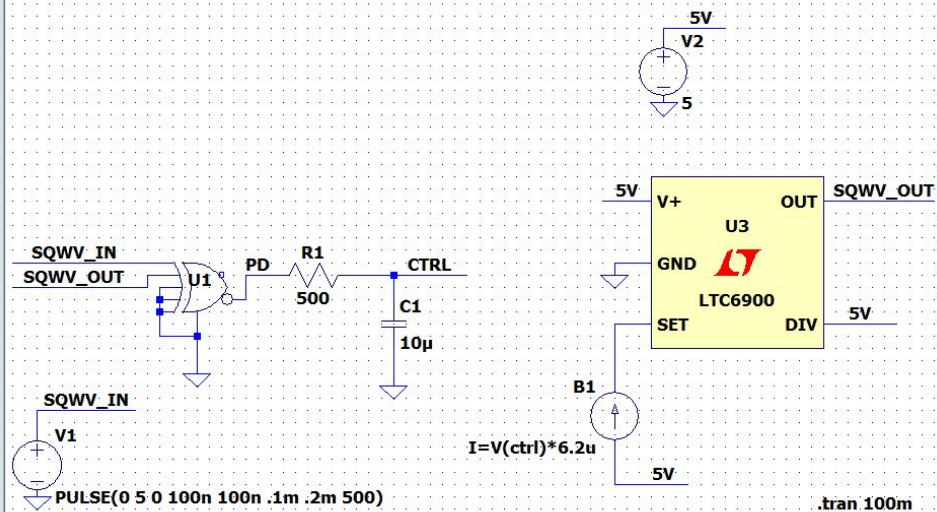
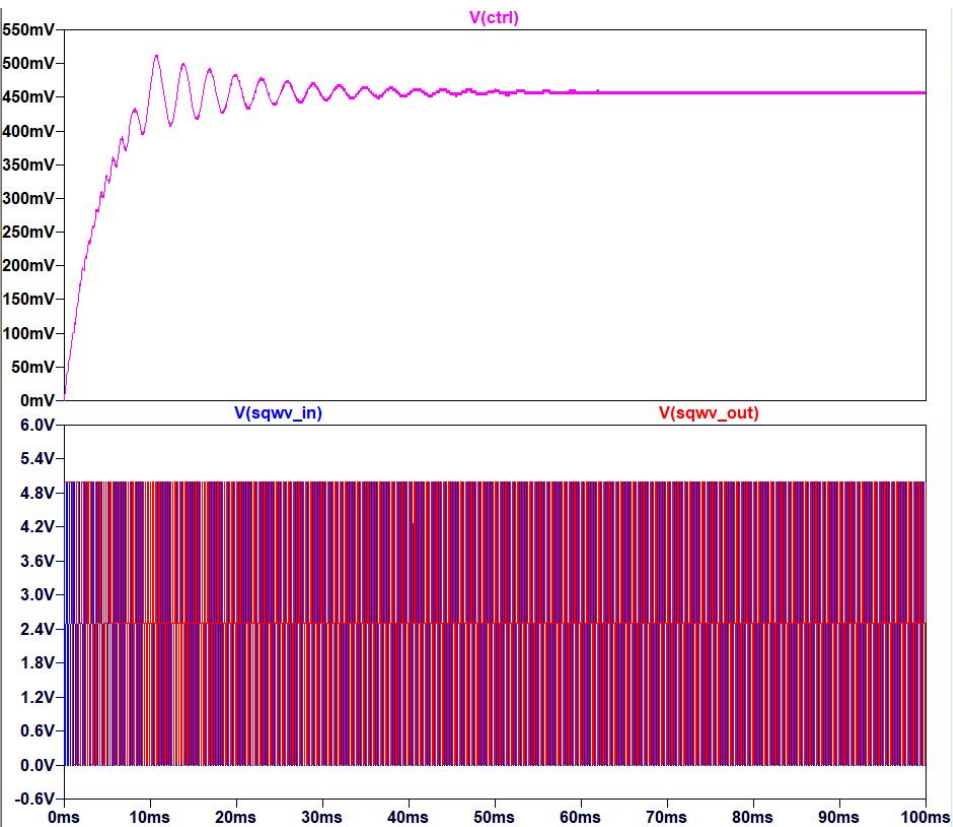
.backanno

end

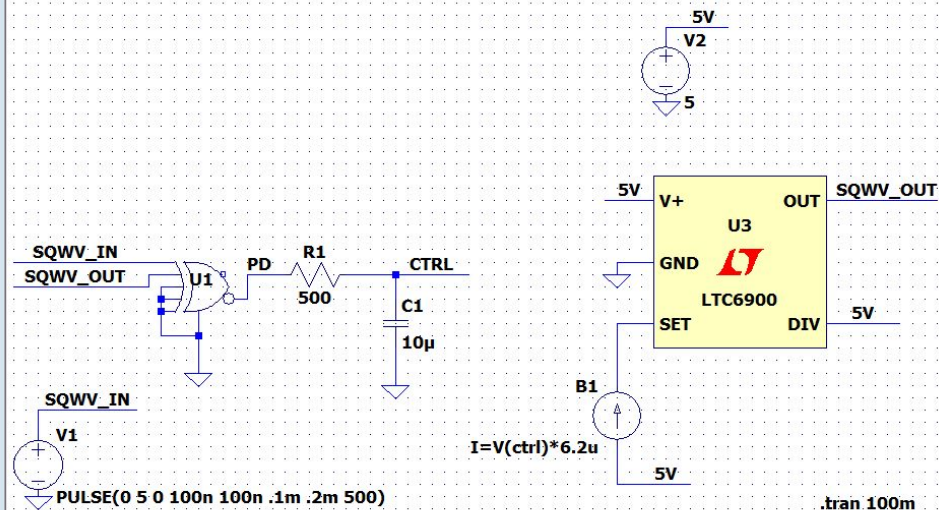
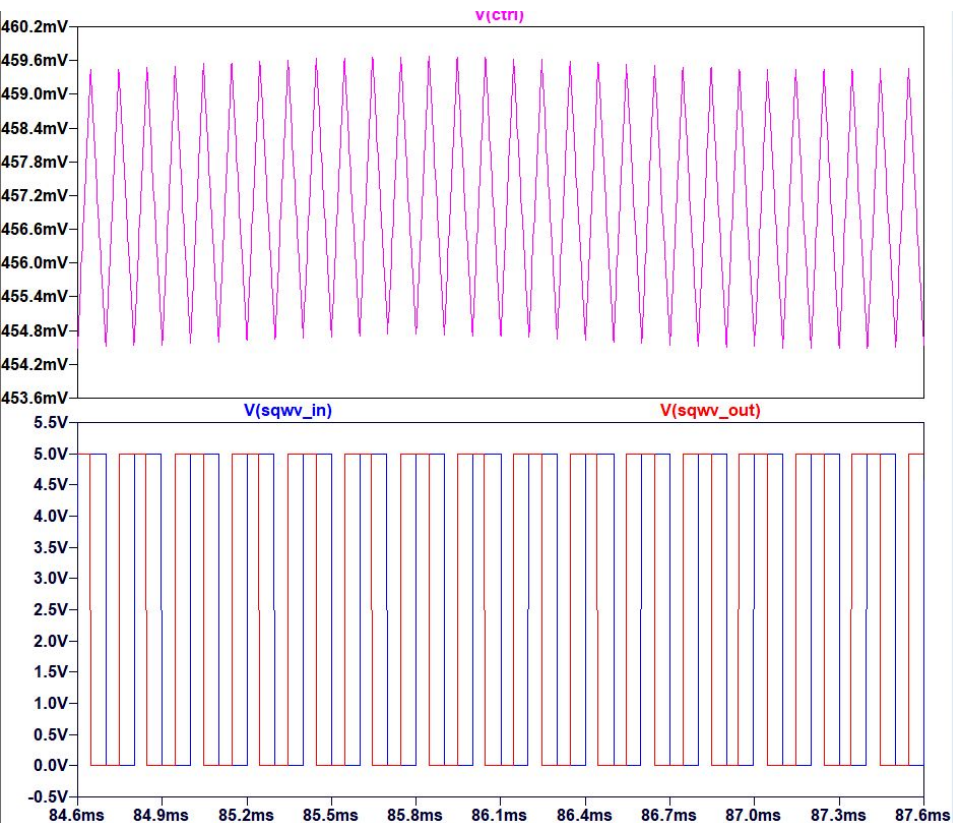




# PLL in LTspice



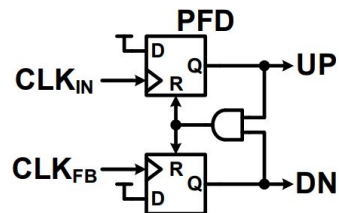
# PLL in LTspice



**SQWV\_OUT** of 5 kHz can be observed.



# Implement PFD with Verilog



```

1  module sync2 #(
2      parameter INIT = 1'b0
3  ) (
4      input wire clk,
5      input wire rstn, // syn
6      input wire async_in,
7      output reg sync_out
8  );
9
10     reg sync_r1;
11
12     always @(posedge clk) begin
13         if (!rstn) begin
14             sync_r1 <= INIT;
15             sync_out <= INIT;
16         end else begin
17             sync_r1 <= async_in;
18             sync_out <= sync_r1;
19         end
20     end
21
22     end
23
24     end
25
26 endmodule

```

```

5  module pfd_fsm #(
6      parameter integer PULSE_STRETCH = 4, // minimum pulse width in clk cycles (>=1)
7      parameter integer STRETCH_WIDTH = 8 // bit-width for stretch counters (must hold
8  ) (
9      input wire clk_fast,
10     input wire rstn, // synchronous active-low reset
11     input wire ref_i,
12     input wire fb_i,
13     output reg up_o,
14     output reg down_o
15 );
16
17     // Synchronizers
18     wire ref_sync, fb_sync;
19     sync2 u_sync_ref (
20         .clk(clk_fast), .rstn(rstn), .async_in(ref_i), .sync_out(ref_sync)
21     );
22     sync2 u_sync_fb (
23         .clk(clk_fast), .rstn(rstn), .async_in(fb_i), .sync_out(fb_sync)
24     );
25
26     // Edge-detection delay registers
27     reg ref_d, fb_d;
28     wire ref_rise = (~ref_d) & ref_sync;
29     wire fb_rise = (~fb_d) & fb_sync;
30
31     // FSM states (simple encoding)
32     localparam IDLE = 2'b00;
33     localparam UP_WAIT = 2'b01;
34     localparam DOWN_WAIT = 2'b10;
35
36     reg [1:0] state;
37
38     // Stretch counters (separate for up and down)
39     reg [STRETCH_WIDTH-1:0] up_cnt;
40     reg [STRETCH_WIDTH-1:0] down_cnt;
41
42     // Make an ARM value sized via addition to zero-vector to avoid truncation warnings
43     localparam integer ARM_INT = (PULSE_STRETCH > 0) ? (PULSE_STRETCH - 1) : 0;
44     wire [STRETCH_WIDTH-1:0] ARM_VAL = {{{(STRETCH_WIDTH){1'b0}}}} + ARM_INT;
45

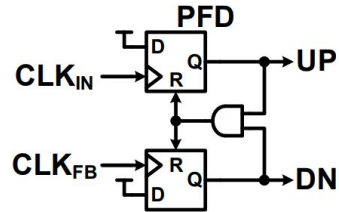
```

```

46     // Single synchronous block: update edge-delays, FSM, counters and outputs
47     always @(posedge clk_fast) begin
48         if (!rstn) begin
49             // synchronous reset
50             ref_d <= 1'b0;
51             fb_d <= 1'b0;
52             state <= IDLE;
53             up_o <= 1'b0;
54             down_o <= 1'b0;
55             up_cnt <= {(STRETCH_WIDTH){1'b0}};
56             down_cnt <= {(STRETCH_WIDTH){1'b0}};
57         end else begin
58             // Update edge detection history
59             ref_d <= ref_sync;
60             fb_d <= fb_sync;
61

```

# Implement PFD with Verilog



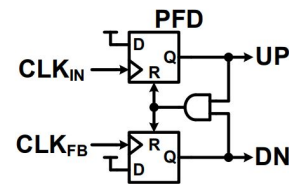
```

57 end else begin
58     // Update edge detection history
59     ref_d <= ref_sync;
60     fb_d <= fb_sync;
61
62     // Default next state = current state (we compute in
63     case (state)
64         IDLE: begin
65             // If both rise in same cycle -> IGNORE (pol
66             if (ref_rise && !fb_rise) begin
67                 state <= UP_WAIT;
68                 // arm UP counter
69                 up_cnt <= ARM_VAL;
70             end else if (fb_rise && !ref_rise) begin
71                 state <= DOWN_WAIT;
72                 // arm DOWN counter
73                 down_cnt <= ARM_VAL;
74             end else begin
75                 state <= IDLE;
76             end
77         end
78
79         UP_WAIT: begin
80             // While waiting for feedback edge, remain u
81             if (fb_rise) begin
82                 state <= IDLE;
83                 // do not modify up_cnt here; stretching
84             end else begin
85                 state <= UP_WAIT;
86             end
87         end
88
89         DOWN_WAIT: begin
90             if (ref_rise) begin
91                 state <= IDLE;
92             end else begin
93                 state <= DOWN_WAIT;
94             end
95         end
96
97         default: state <= IDLE;
98     endcase
99
100     // Counters and outputs (separate deterministic beha
101     if (up_cnt != {(STRETCH_WIDTH){1'b0}}) begin
102         up_cnt <= up_cnt - 1'b1;
103         up_o <= 1'b1;
104     end else begin
105         up_o <= 1'b0;
106     end
107
108     if (down_cnt != {(STRETCH_WIDTH){1'b0}}) begin
109         down_cnt <= down_cnt - 1'b1;
110         down_o <= 1'b1;
111     end else begin
112         down_o <= 1'b0;
113     end
114
115 end
116
117 endmodule

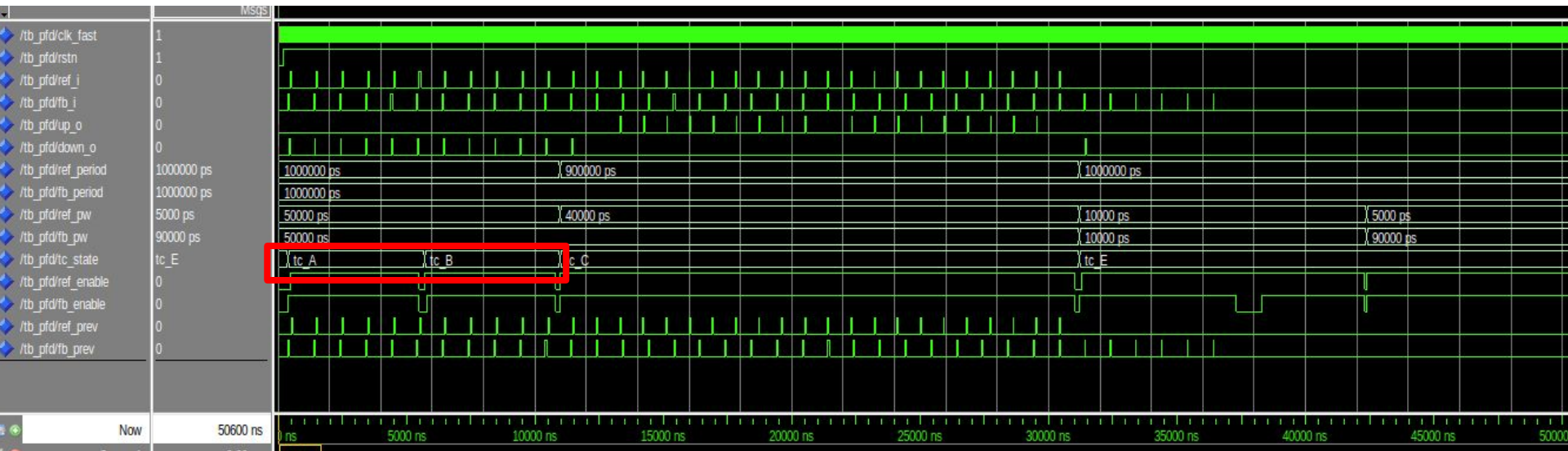
```

# Running testbenches in ModelSim:

## Case A and B: FB\_i leads REF\_i



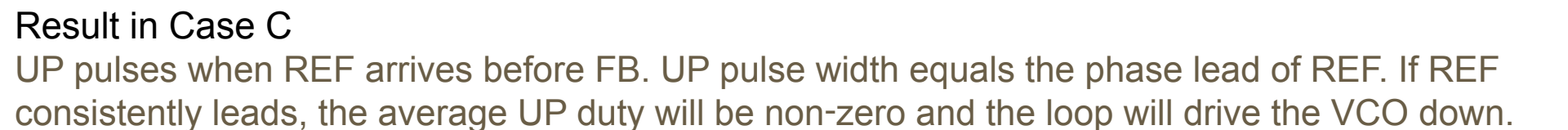
**ModelSim**  
Intel® Quartus® Prime



### Result in Case A and B

On each cycle where FB edge occurs before REF, the PFD asserts DN until the REF edge arrives; DN pulse width equals the phase lead (time between REF and FB edges).

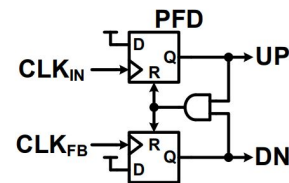
**ModelSim**  
Intel® Quartus® Prime



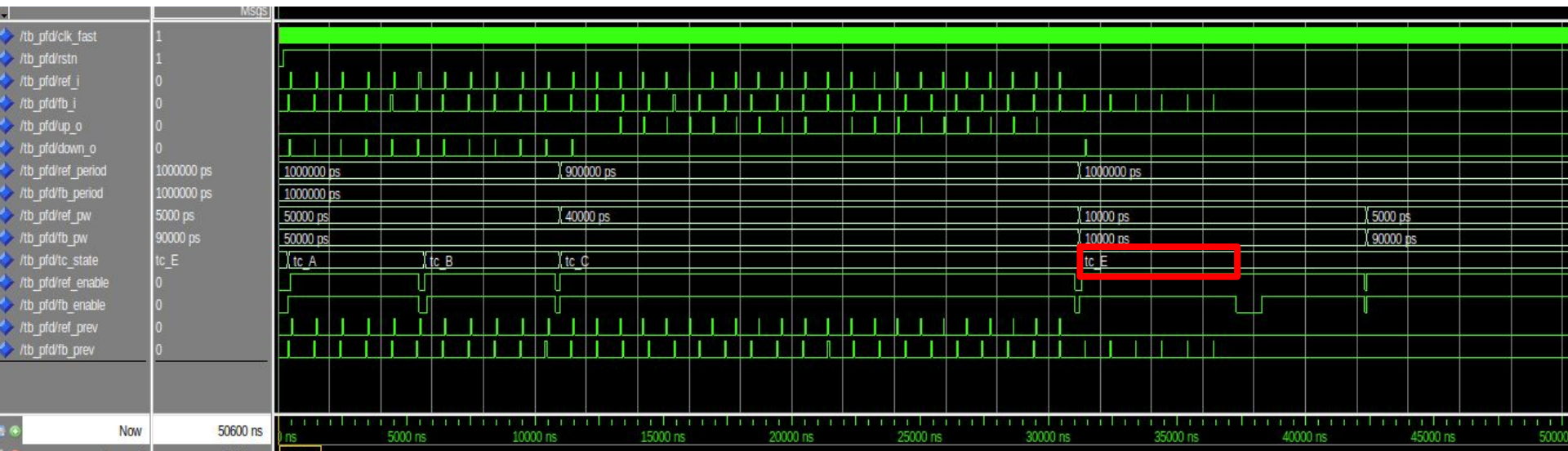
UP pulses when REF arrives before FB. UP pulse width equals the phase lead of REF. If REF consistently leads, the average UP duty will be non-zero and the loop will drive the VCO down.

# Running testbenches in ModelSim:

## Case E1: small offset with short



**ModelSim**  
Intel® Quartus® Prime



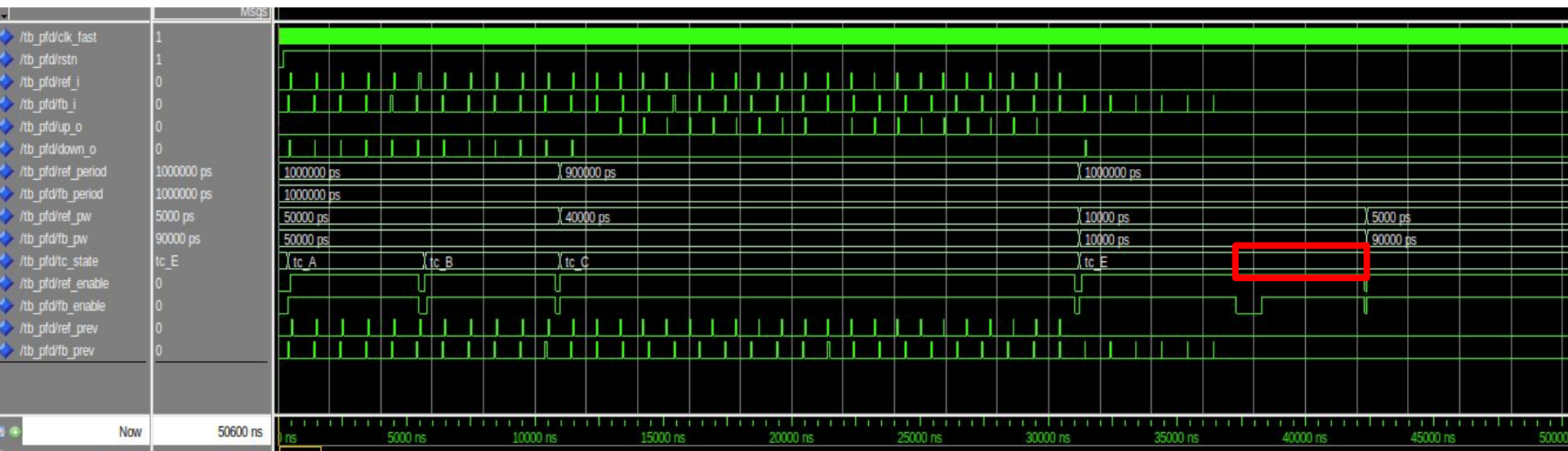
### Result in Case E1

Edge-triggered PFD is duty-cycle tolerant, so short pulses should still work if edges are clean. But if pulse widths approach or fall below internal timing margins, you may see missed edges or very narrow UP/DN pulses and possible metastability.



# Running testbenches in ModelSim:

## Case E2: drop one FB cycle while REF keeps running

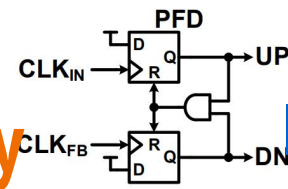


### Result in Case E2

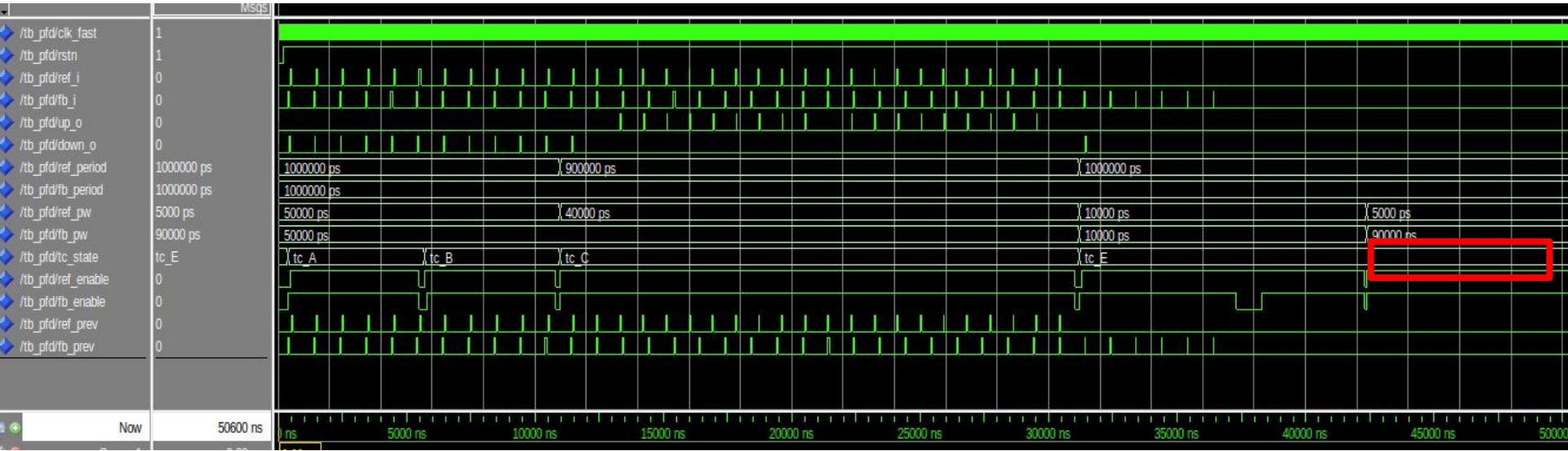
When FB is disabled for one cycle, the PFD sees consecutive REF edges without FB.  
→ continuous UP pulses for that interval (large integrator step), then recovery when FB returns;  
this simulates a slip or transient that the loop must correct → NOT seen in ModelSim ...

# Running testbenches in ModelSim:

## Case E3: extreme narrow/broad duty



**ModelSim**  
Intel® Quartus® Prime



### Result in Case E3

Very narrow REF pulses or very wide FB pulses still produce edge events, but overlap of pulses and internal reset timing can create spurious short UP/DN pulses or pulse cancellation if edges fall inside the PFD reset window. Watch for glitches, shortened pulses, or missed resets.

→ NOT seen in ModelSim ...

# Conclusion

- **XOR Phase Detector in LTspice**
  - LTspice simulation controls the frequency at 5 kHz with square wave inputs.
- **Phase-Frequency Discriminator (PFD) in ModelSim**
  - ModelSim fails to generate UP/DN signals in some edge cases.



**If you have any feedback,  
please contact me at**

— twwang97@gmail.com —



twwang97

---

Electronics and AI Training Program (3rd session in 2025)

# Thanks for your attention



NYCU

國立陽明交通大學



陽明交大雷射系統研究中心