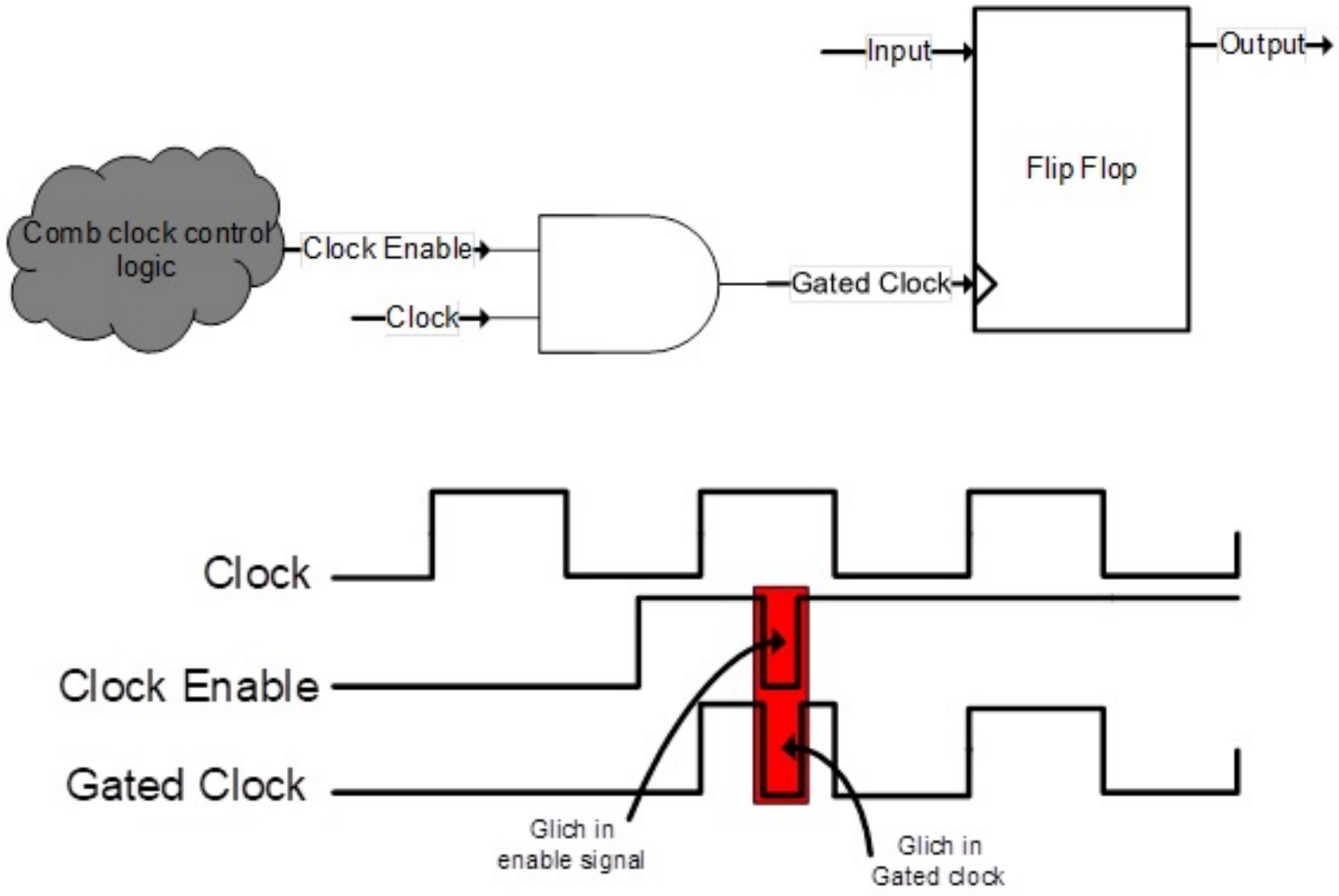
ECE260B Winter 22

Low-power & High-performance Techniques

Prof. Mingu Kang

Clock gating

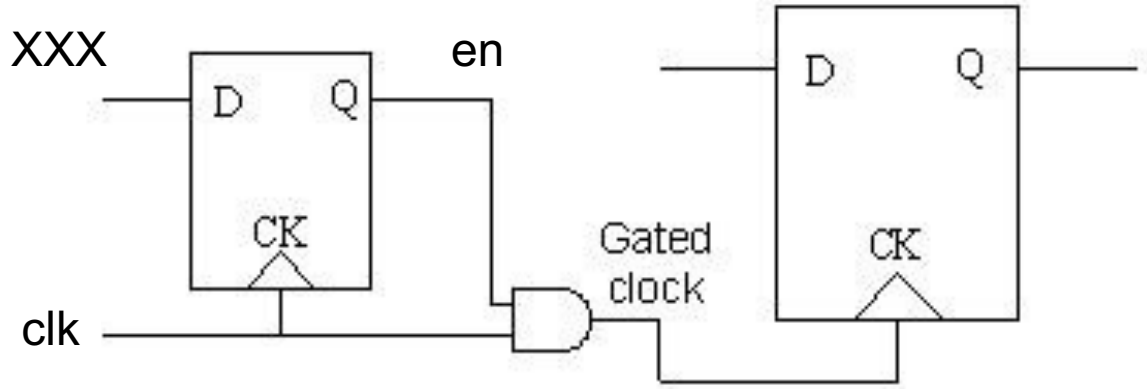
CLK Gating with Glitch Risk

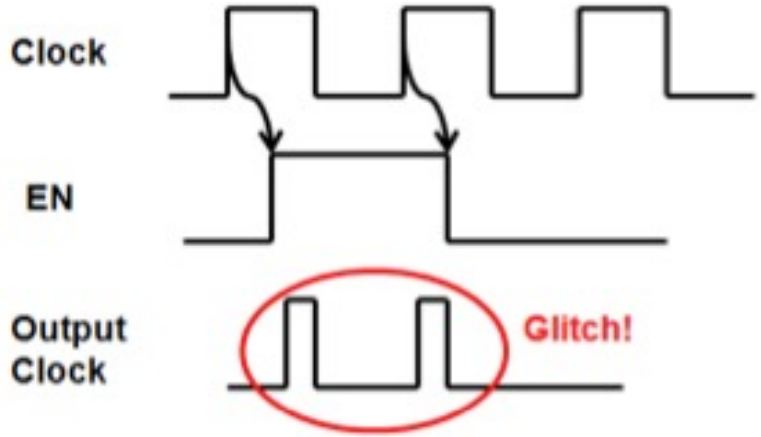


Latch-based Clock Gating

- Clock Gating RTL Code

```
assign gclk = en && clk;

always @ (posedge clk) begin
   en <= XXXXX;
end

always @ (posedge gclk) begin
   out_q = in;
end
```



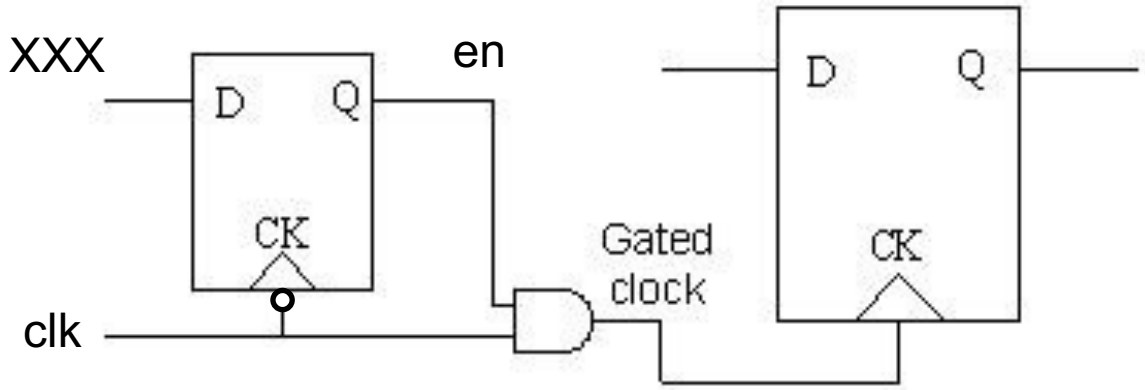


https://anysilicon.com/the-ultimate-guide-to-clock-gating/

Glitch-free Clock Gating

- Glitch-free Clock Gating

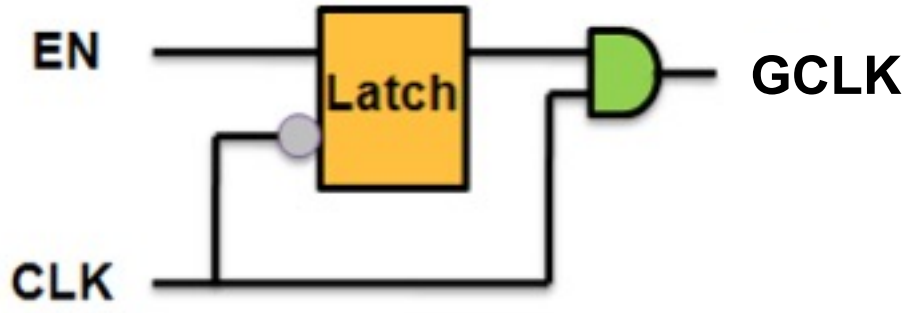
```
assign gclk = en_neg && clk;
always @ (negedge clk) begin
   en_neg <= XXXX;
end

always @ (posedge gclk) begin
   out_q <= in;
end
```



Instantiating CLK Gating Cell from STDcell

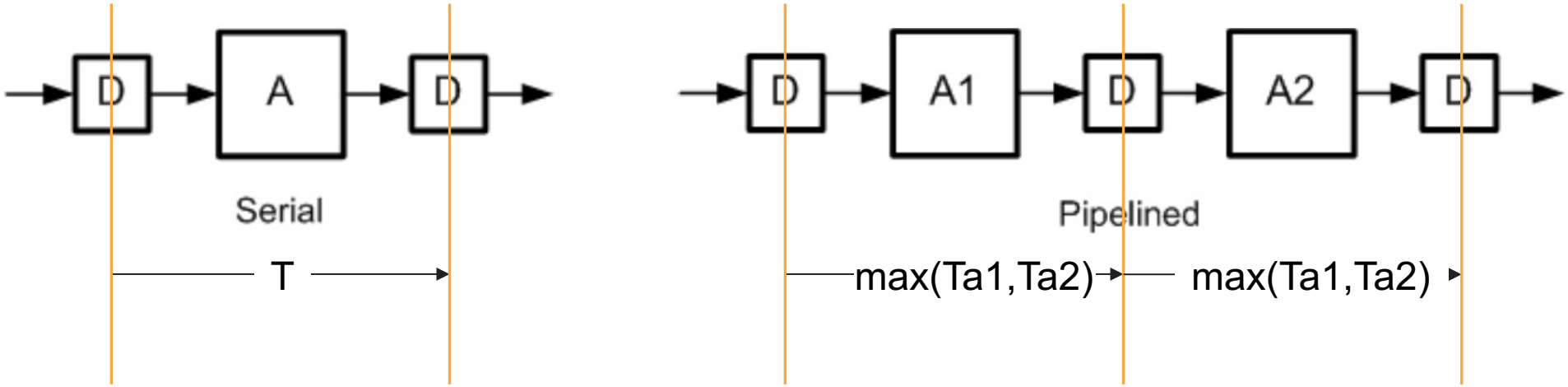
- Use of predefined clk gating cell

```
clkgate clkgate_instance .EN(enable), .CLK(clk), .GCLK(gclk);
```



Pipelining

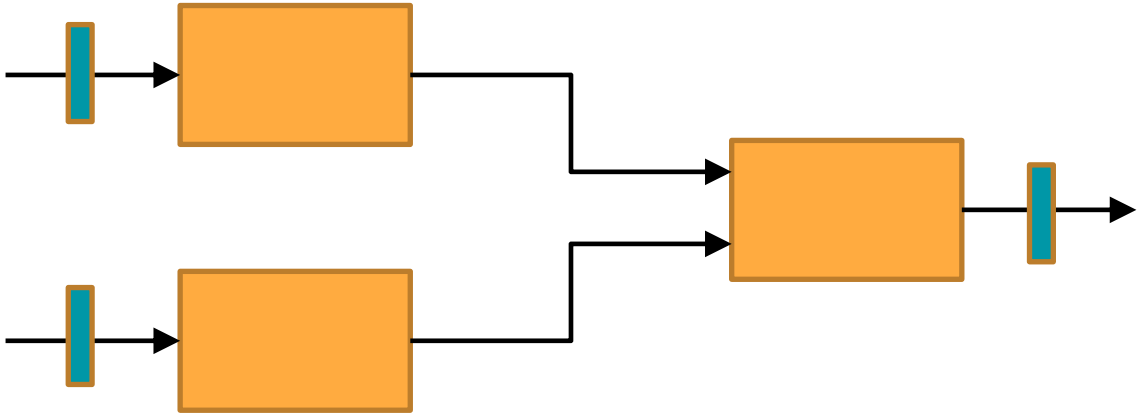
Pipeline

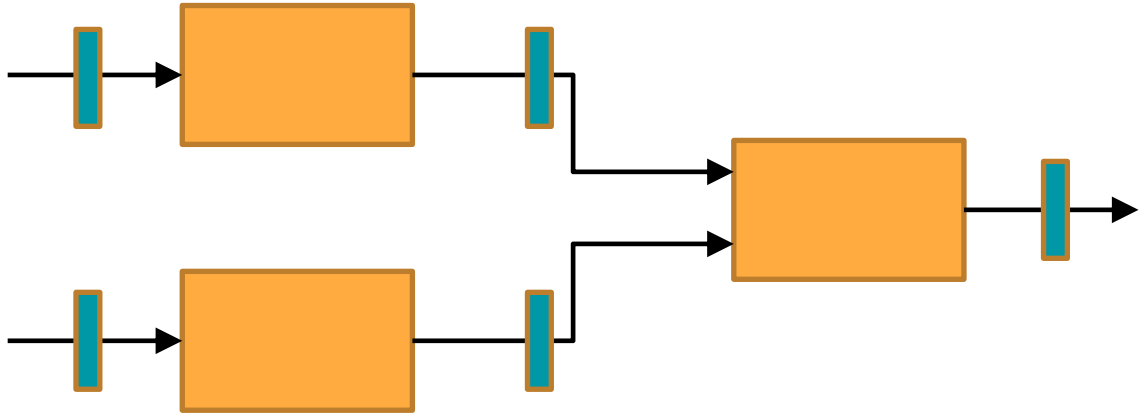


- Pipeline:
  - divide the combinational logic path into multiple stages (above example: 2 stage pipeline)
  - CLK period after pipeline: max(Ta1, Ta2) < T
  - For uniform pipeline, ideally new CLK period can be T/2

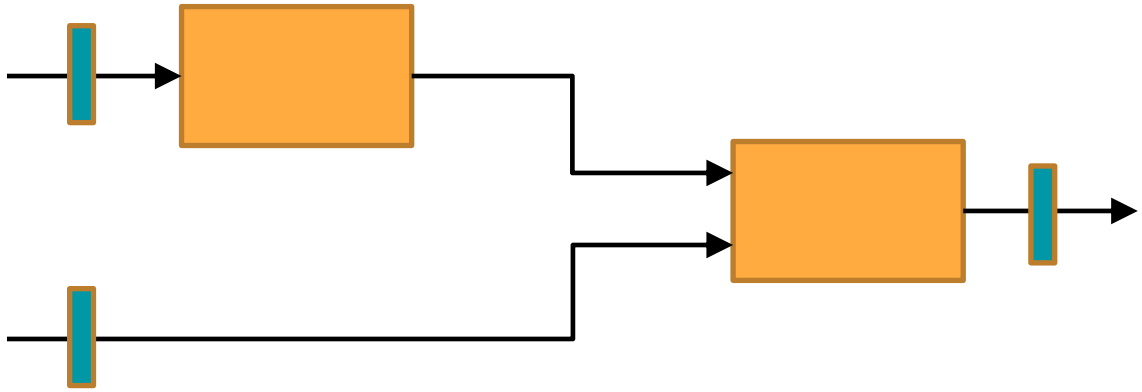
    -> 2X frequency up assuming no setup / hold time

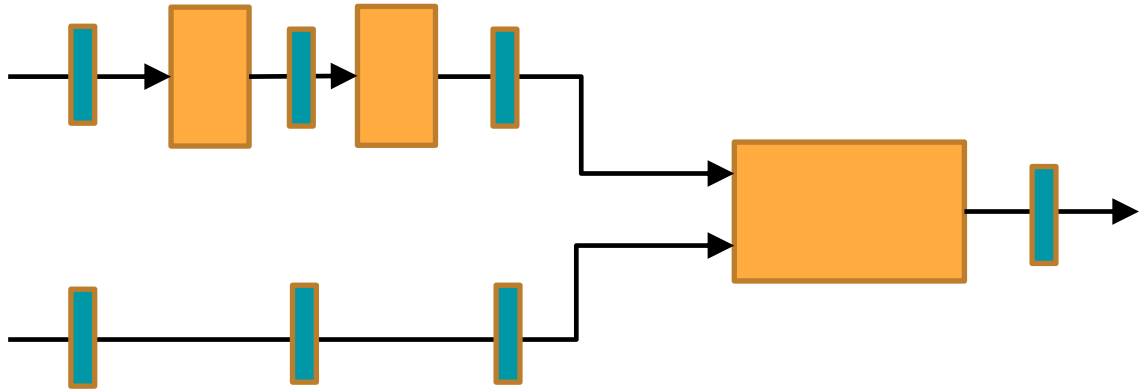
Pipeline in Different Cases





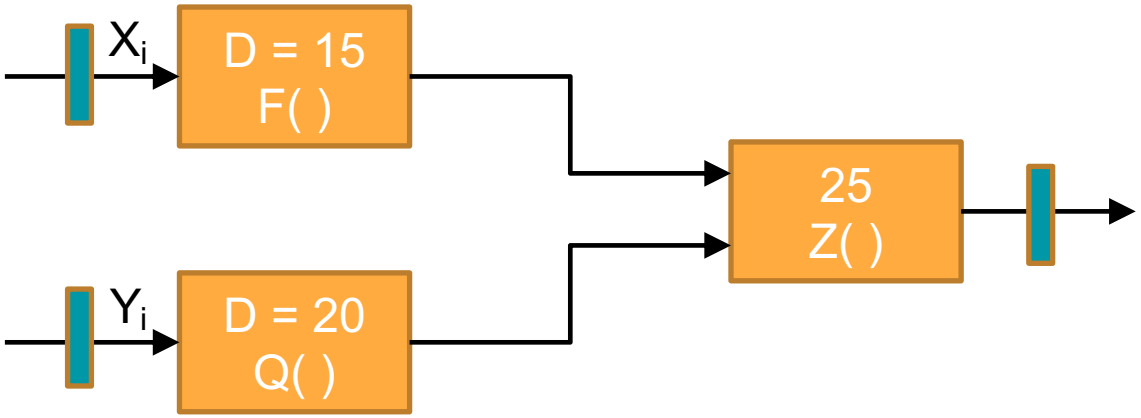
Pipeline in the dataflow with two branches



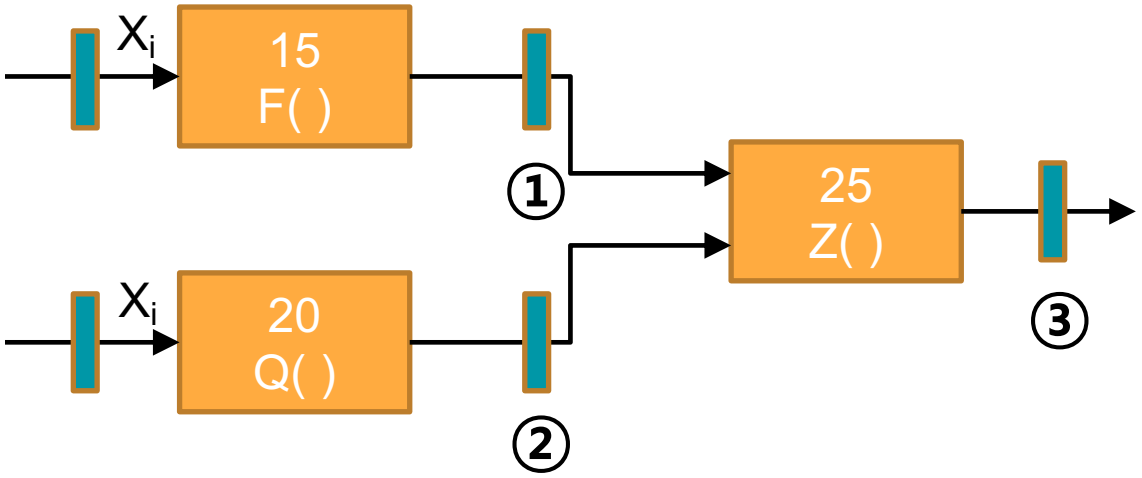


Pipeline with the control signals

Throughput and Latency in Pipelined System



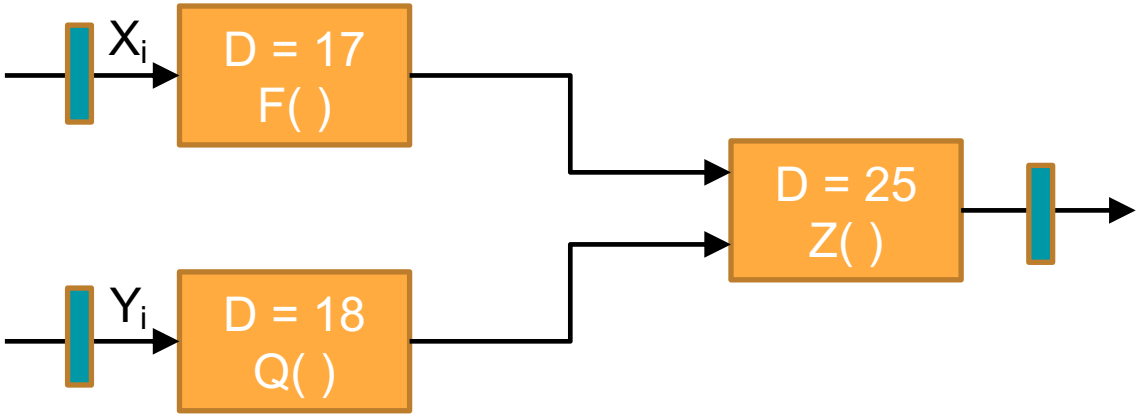
before pipeline



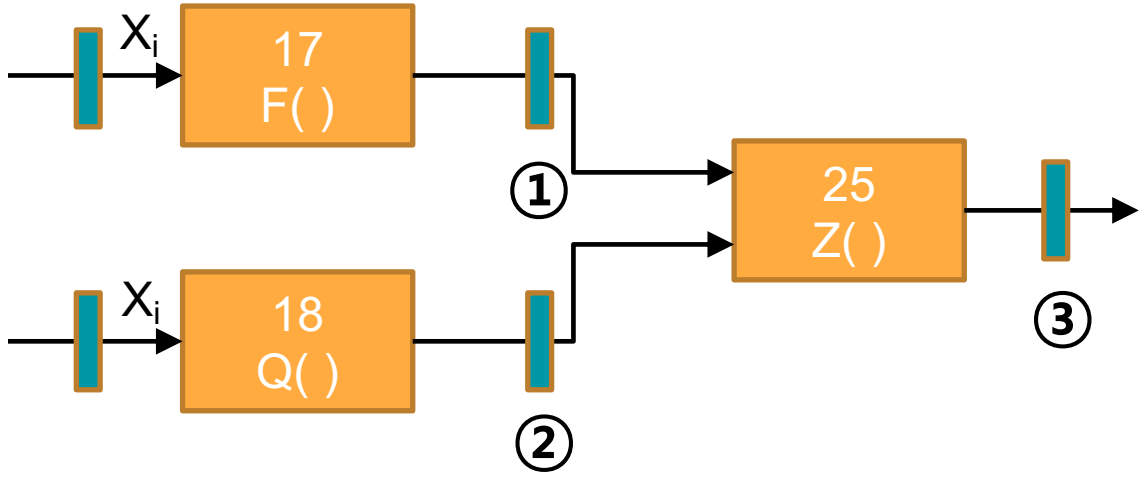
after pipeline

cycle	1	2	3	4
①	$F(X_i)$	$F(X_{i+1})$	$F(X_{i+2})$	…
②	$Q(Y_i)$	$Q(Y_{i+1})$	$Q(Y_{i+2})$	…
③		$Z(F(X_i), Q(Y_i))$	$Z(F(X_{i+1}), Q(Y_{i+1}))$	…

Throughput and Latency in Pipelined System



before pipeline

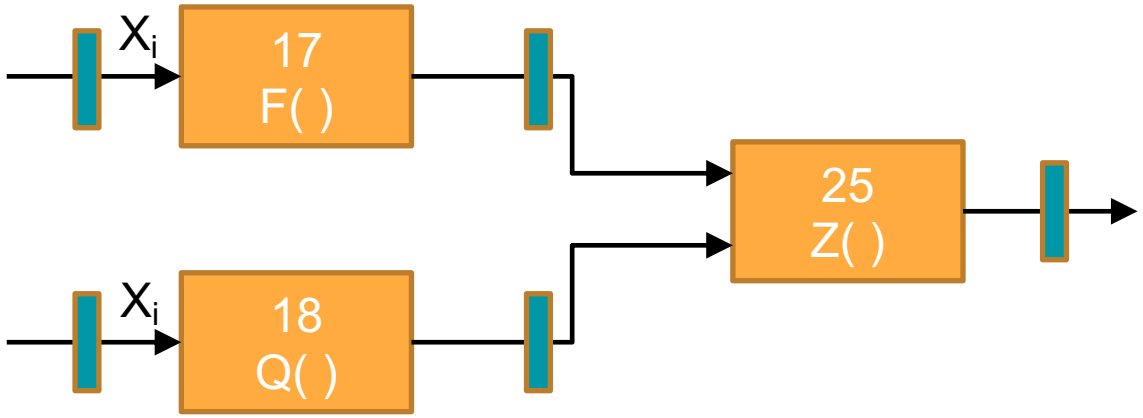


after pipeline

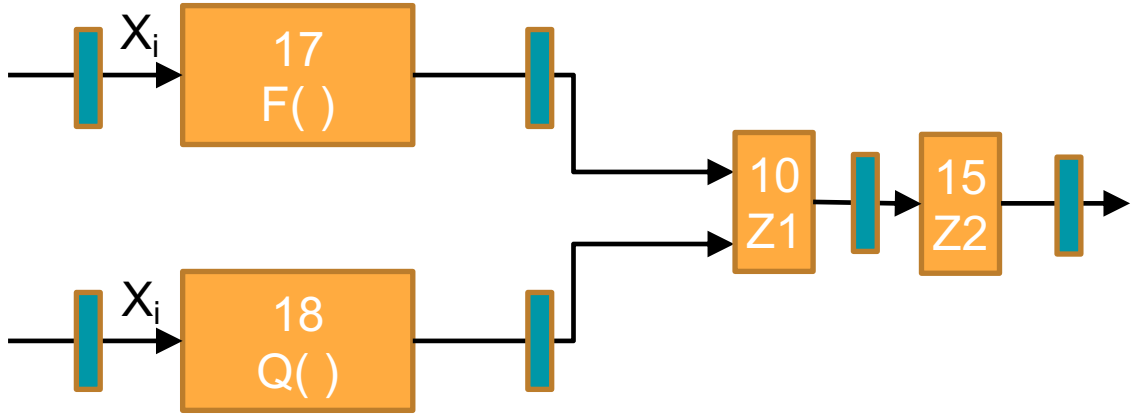
- Before pipeline
  - clk period: 43
  - Throughput: 1/43
  - Latency: 43

- After pipeline
  - clk period = 25
  - Throughput: 1/25
  - Latency: 25 + 25 = 50
  - assumption: ideal register
    e.g., no setup/hold time

Further Pipeline



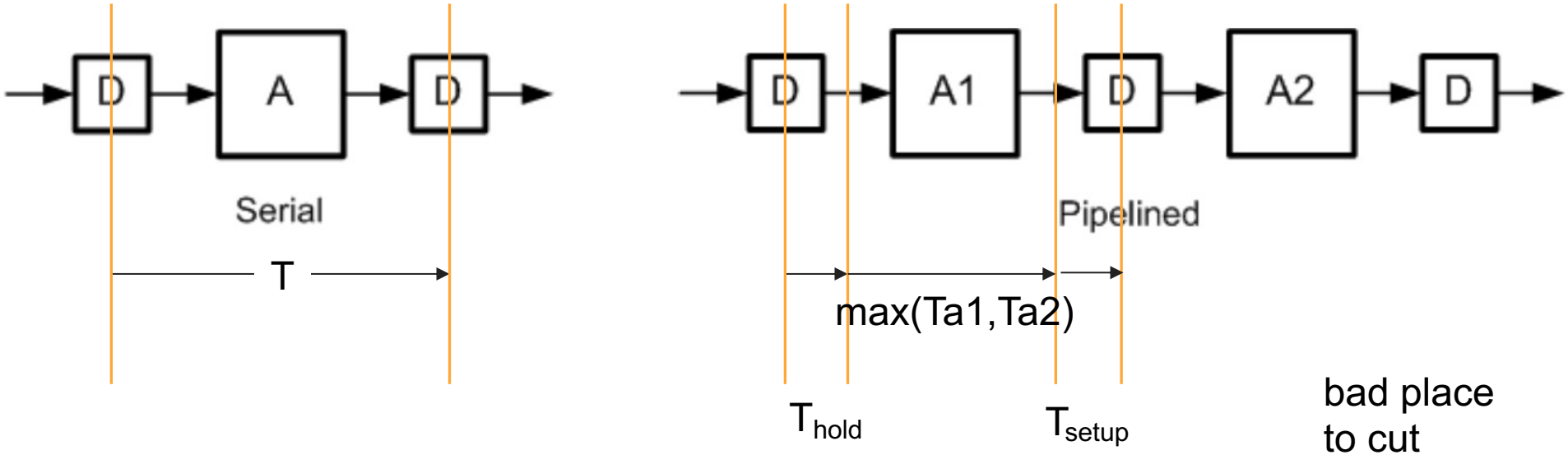
after pipeline

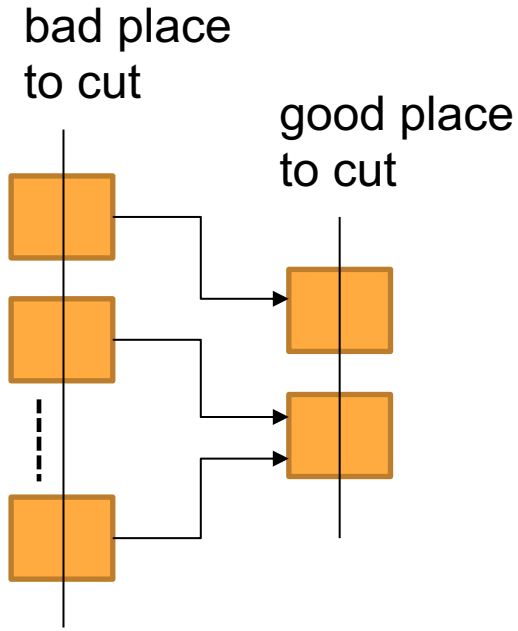


after more pipeline

- Further pipeline
  - If the target throughput / frequency hasn't met, apply further pipeline
  - Apply pipeline for slowest combination path
  - New throughput: 1/18, New latency: 18 * 3 =54

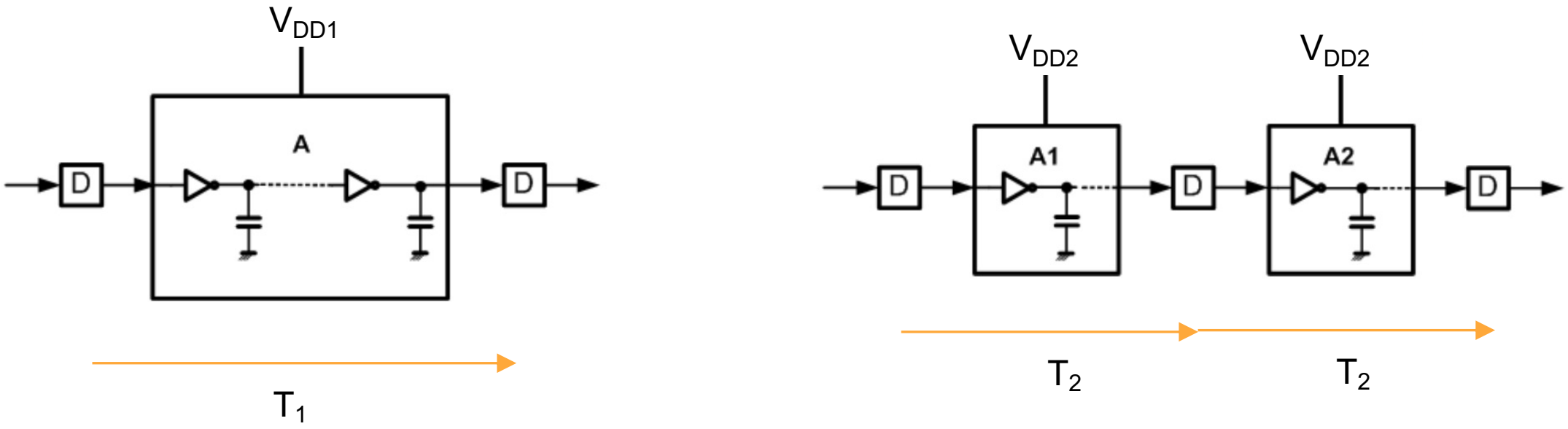
Pipeline in Reality





- Pipeline in reality:
  - Additional delay for T_{hold} and T_{setup}
  - Additional power / area consumption for registers
  - Good to add pipeline registers at reduced paths

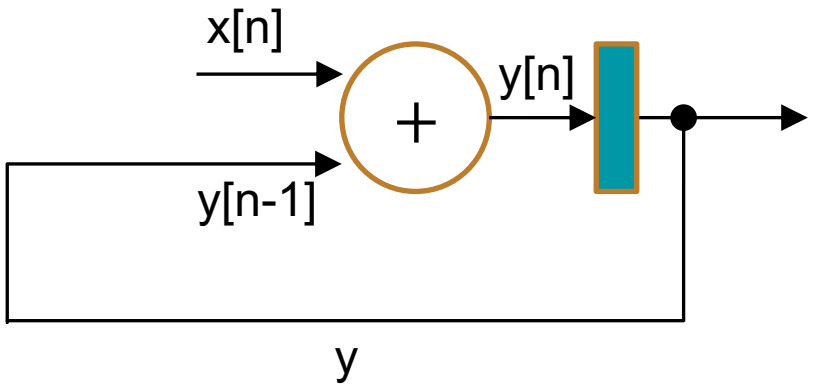
Pipeline for Low-power

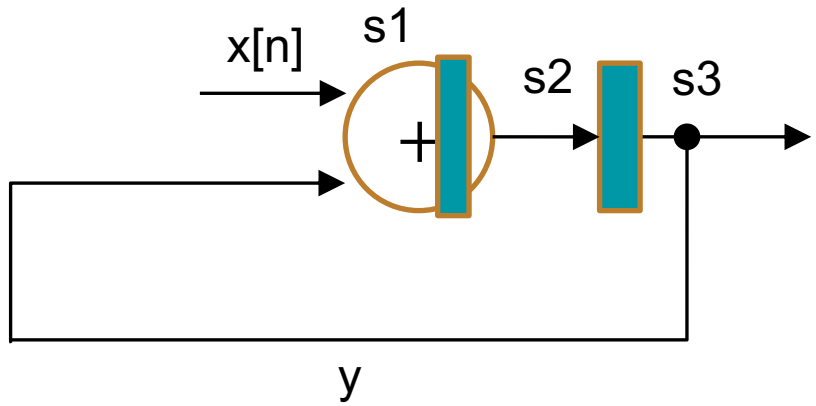


- Pipeline for low V_{DD}:
  - $T_2 < T_1$ at same supply voltage ($V_{DD1} = V_{DD2}$)
  - To achieve the same frequency, V_{DD2} can be reduced by a factor of α
  - Power saving by a factor of α^2

Pipeline in Cyclic Dataflow (Pipeline Hazard)

y[n] = x[n] + y[n-1]





needs input x[n] and y[n-1],

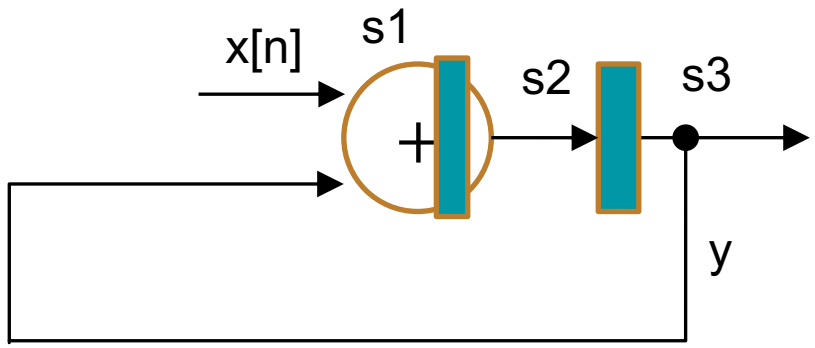
but we do not have y[n-1] yet

cycle	1	2	3	4	5	…
s1	n	n+1	n+2	n+3		…
s2	n-1	n	n+1	n+2	n+3	…
s3	n-2	n-1	n	n+1	n+2	…

Interleaving in Pipelined System

Two independent sets are interleaved

x2[n+1] x1[n+1] x2[n+1] x1[n+1] x2[n] x1[n] →



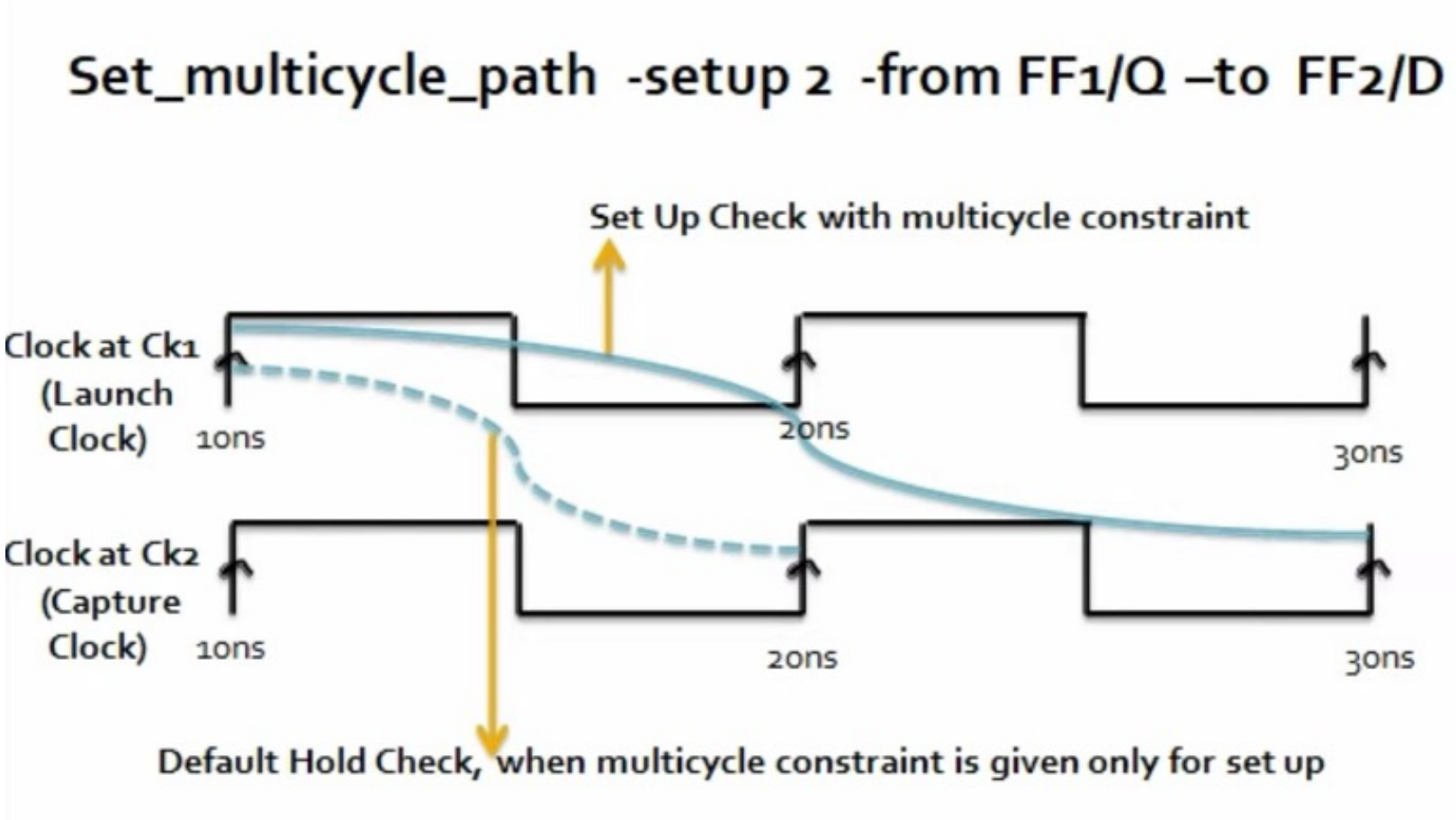
needs input x[n] and y[n-1],

Now, we have y[n-1]

cycle	1	2	3	4	5	…
s1	n	n	n+1	n+1		…
s2	n-1	n	n	n+1	n+1	…
s3	n-1	n-1	n	n	n+1	…

Multi-path Design

Multicycle Setup without Hold Check

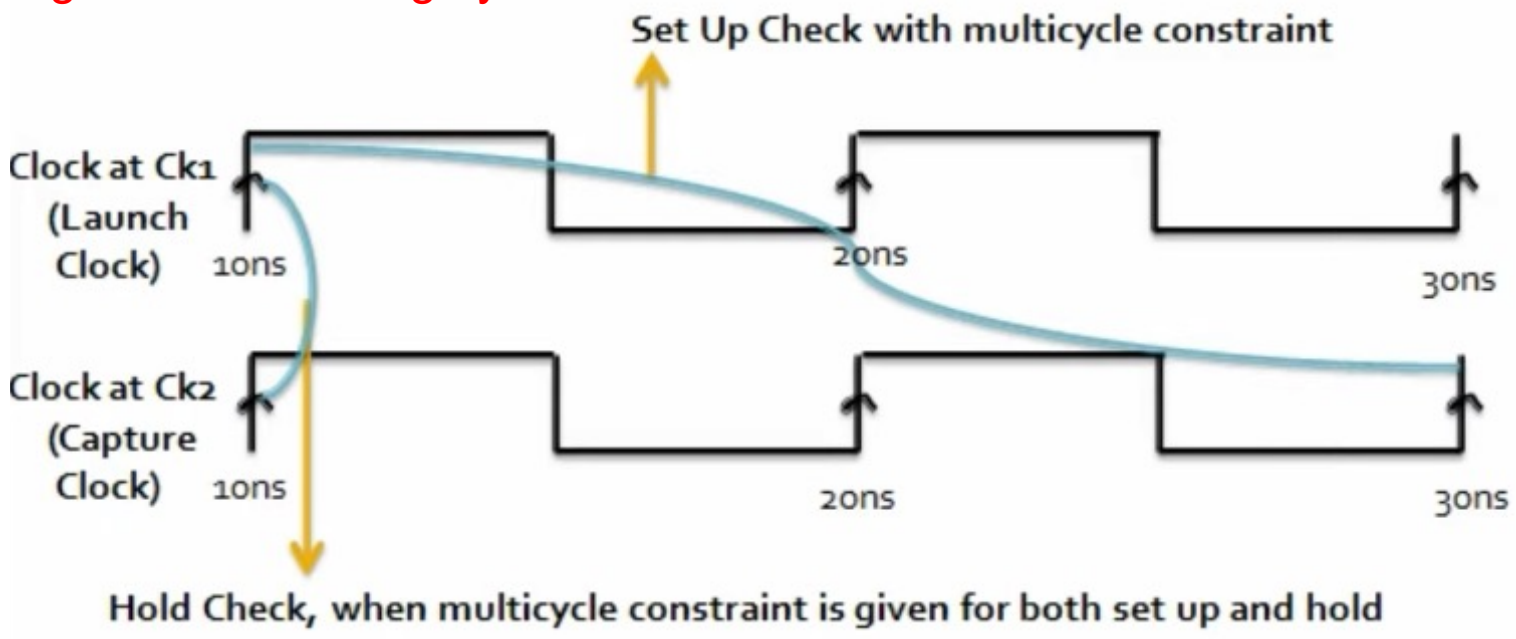


https://www.youtube.com/watch?v=lqIAUK9r14w

Multicycle Setup with Hold Check Correction

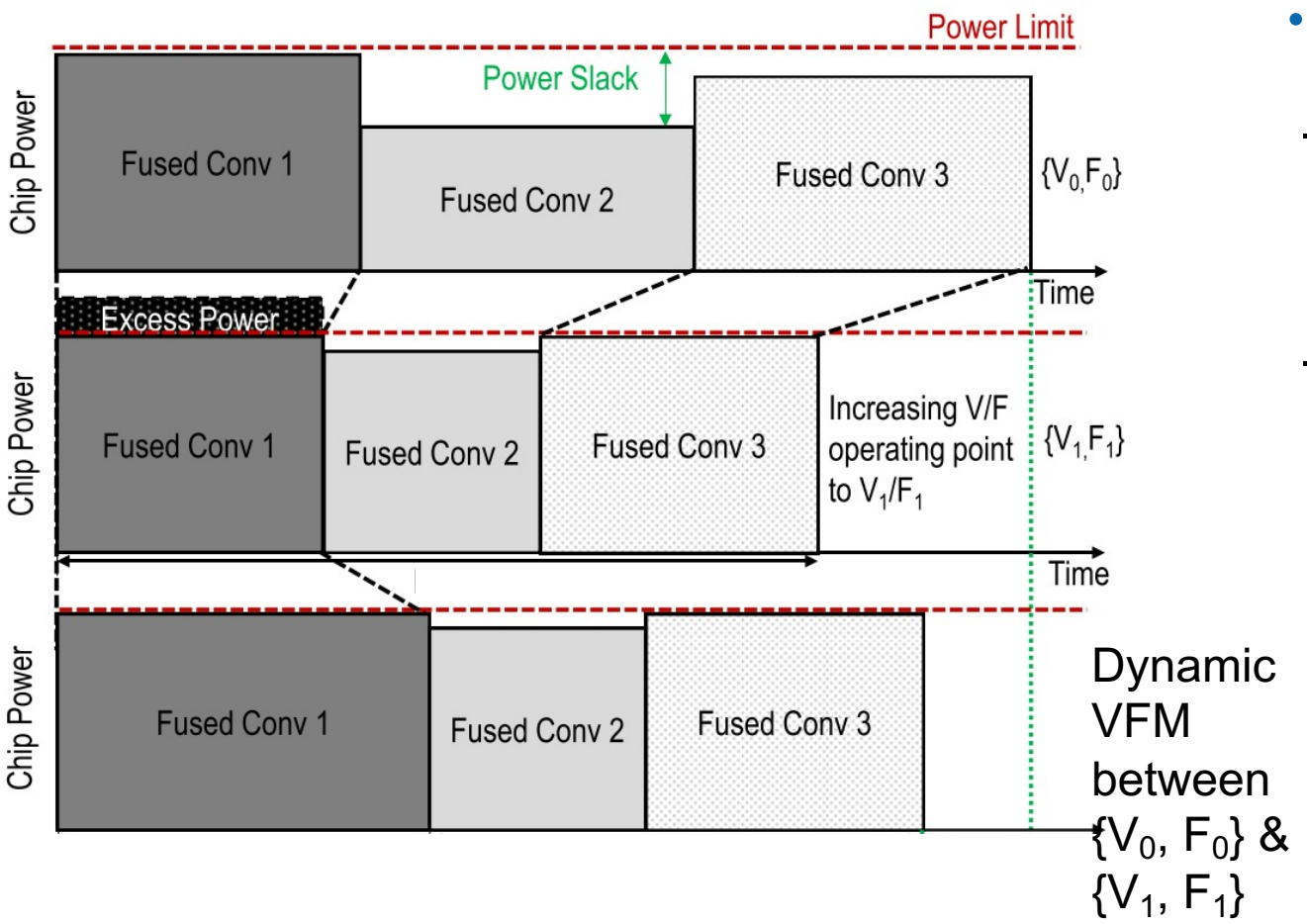
```
Set_multicycle_path  -setup 2  -from FF1/Q –to  FF2/D
Set_multicycle_path  -hold 1   -from FF1/Q –to  FF2/D
```

bring back hold timing by 1



Dynamic Voltage Frequency Scaling

DVFM (Dynamic Voltage Frequency Modulation)



Dynamic VFM between $\{V_0, F_0\}$ & $\{V_1, F_1\}$

- Dynamic frequency scaling

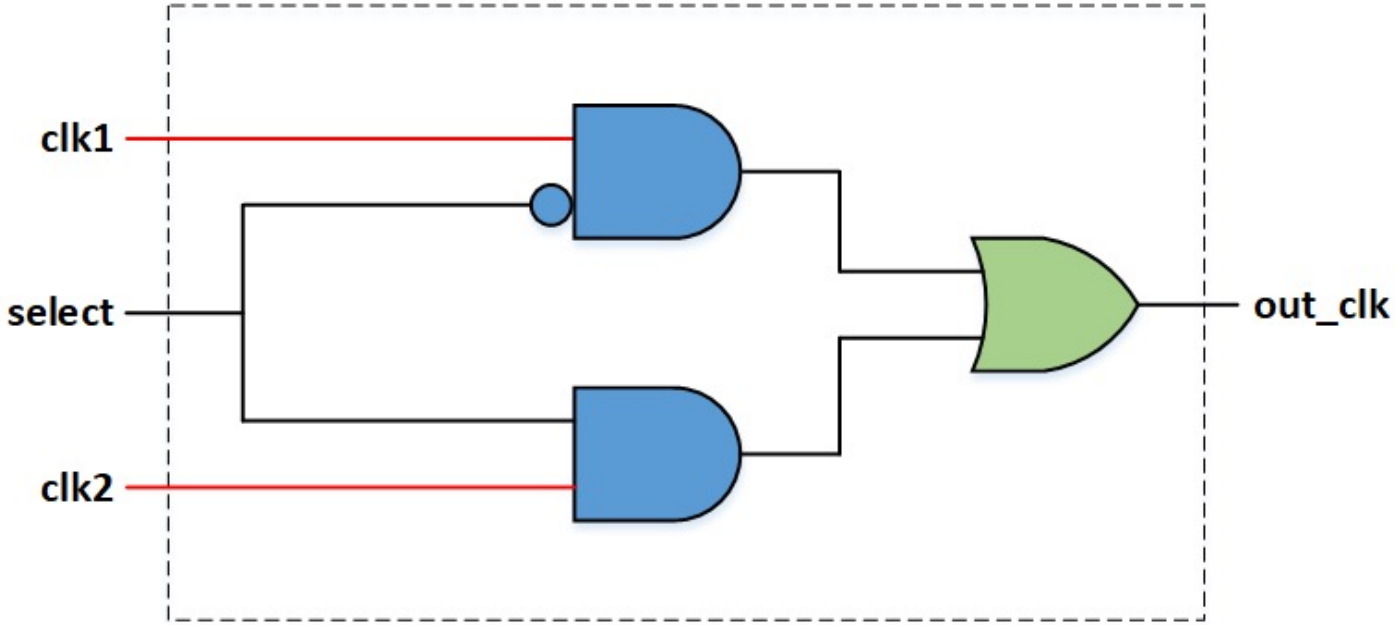
- almost always appear with dynamic voltage scaling

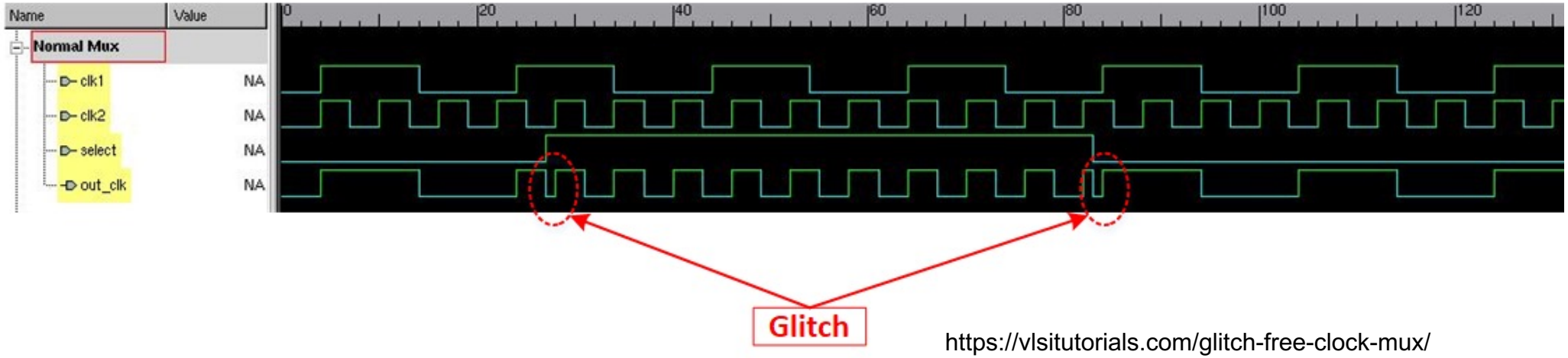
- known as **dynamic voltage and frequency scaling** (**DVFS**).

- when frequency is reduced, it is also called as “throttling”

- used for power or heat management

CLK Mux with Glitch Risk

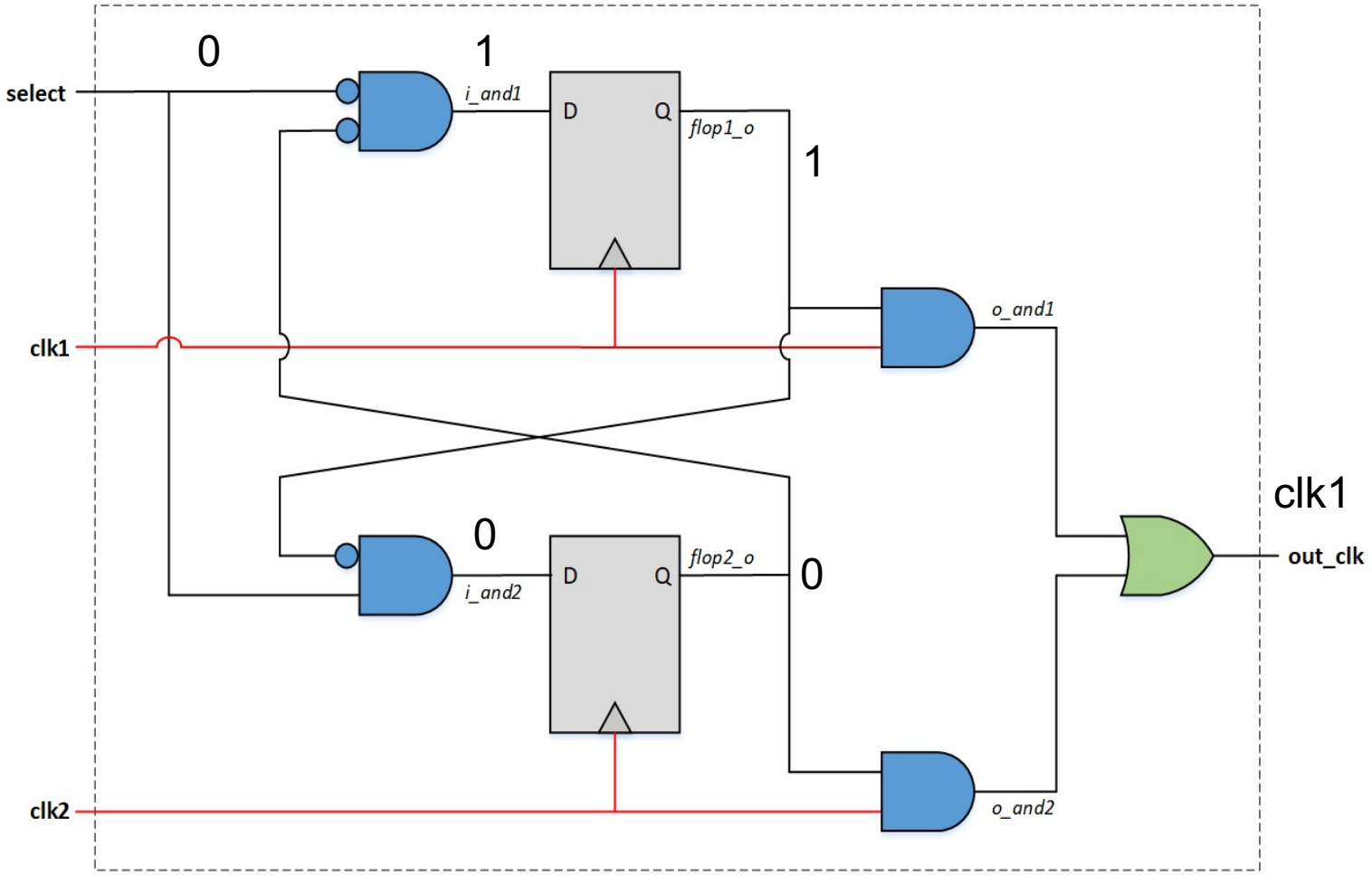




https://vlsitutorials.com/glitch-free-clock-mux/

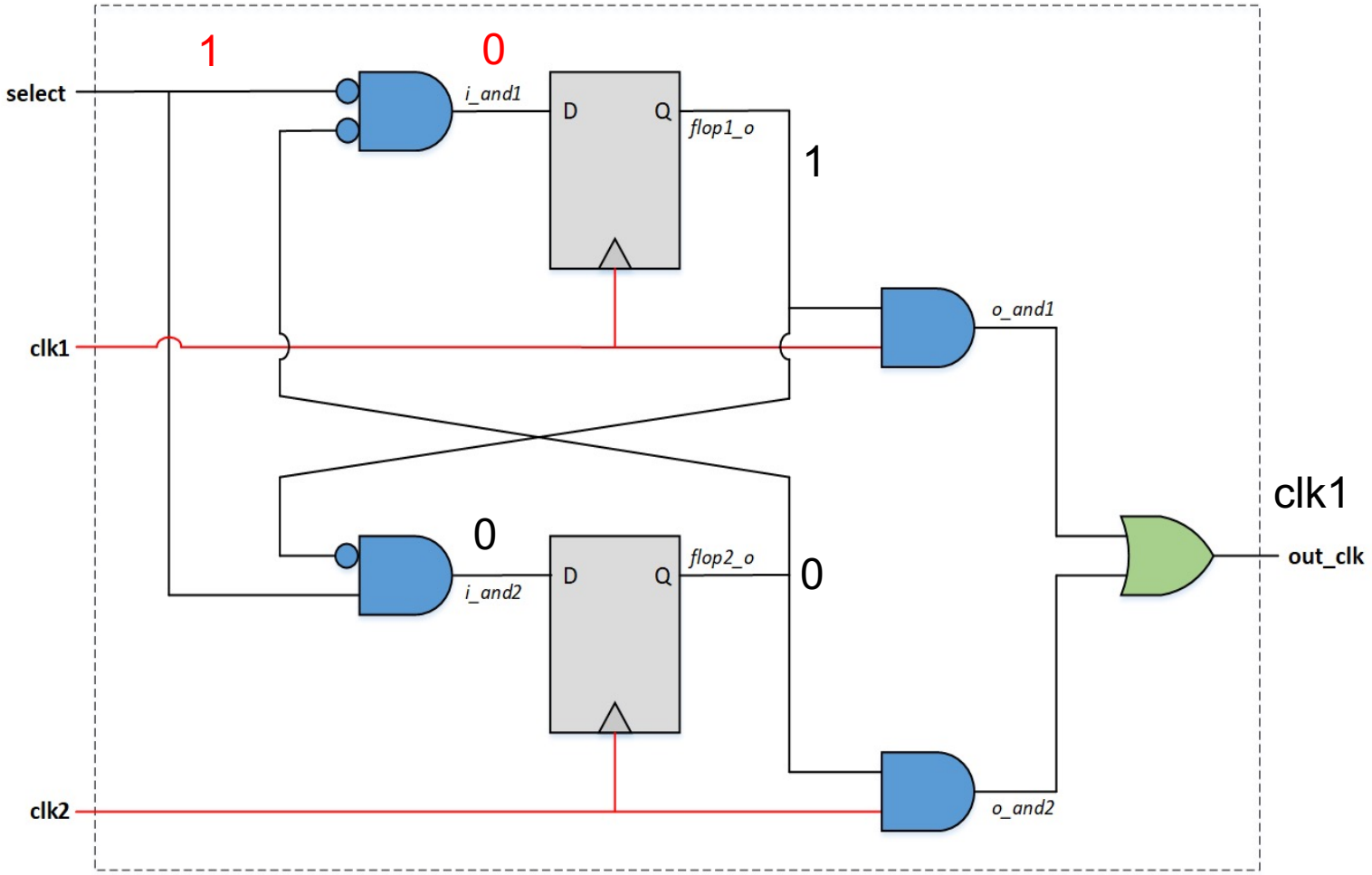
Glitch-free Clock Mux

"select" signal is 0



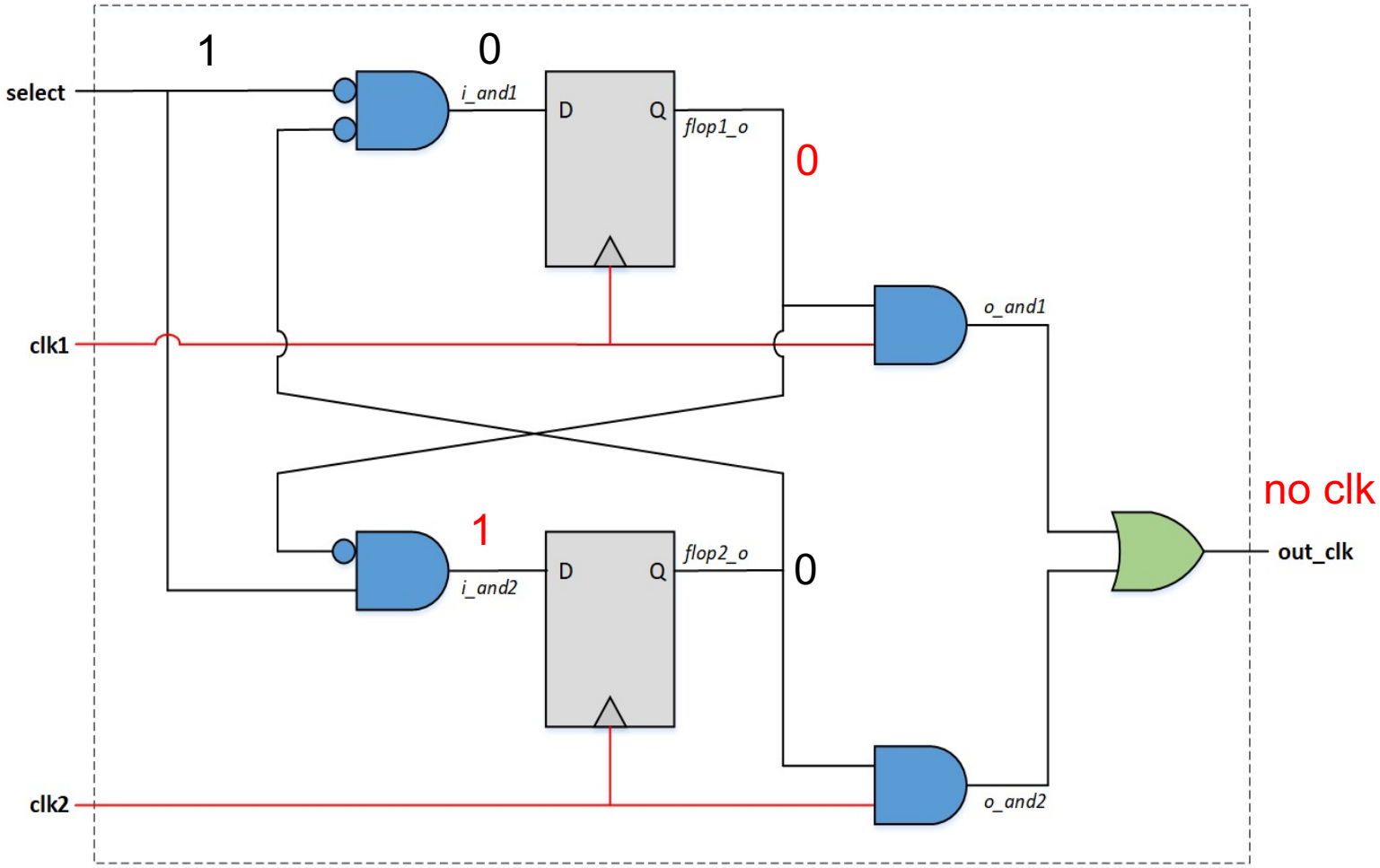
Glitch-free Clock Mux

"select" changed:
0 -> 1
But clk rising edge hasn't come yet



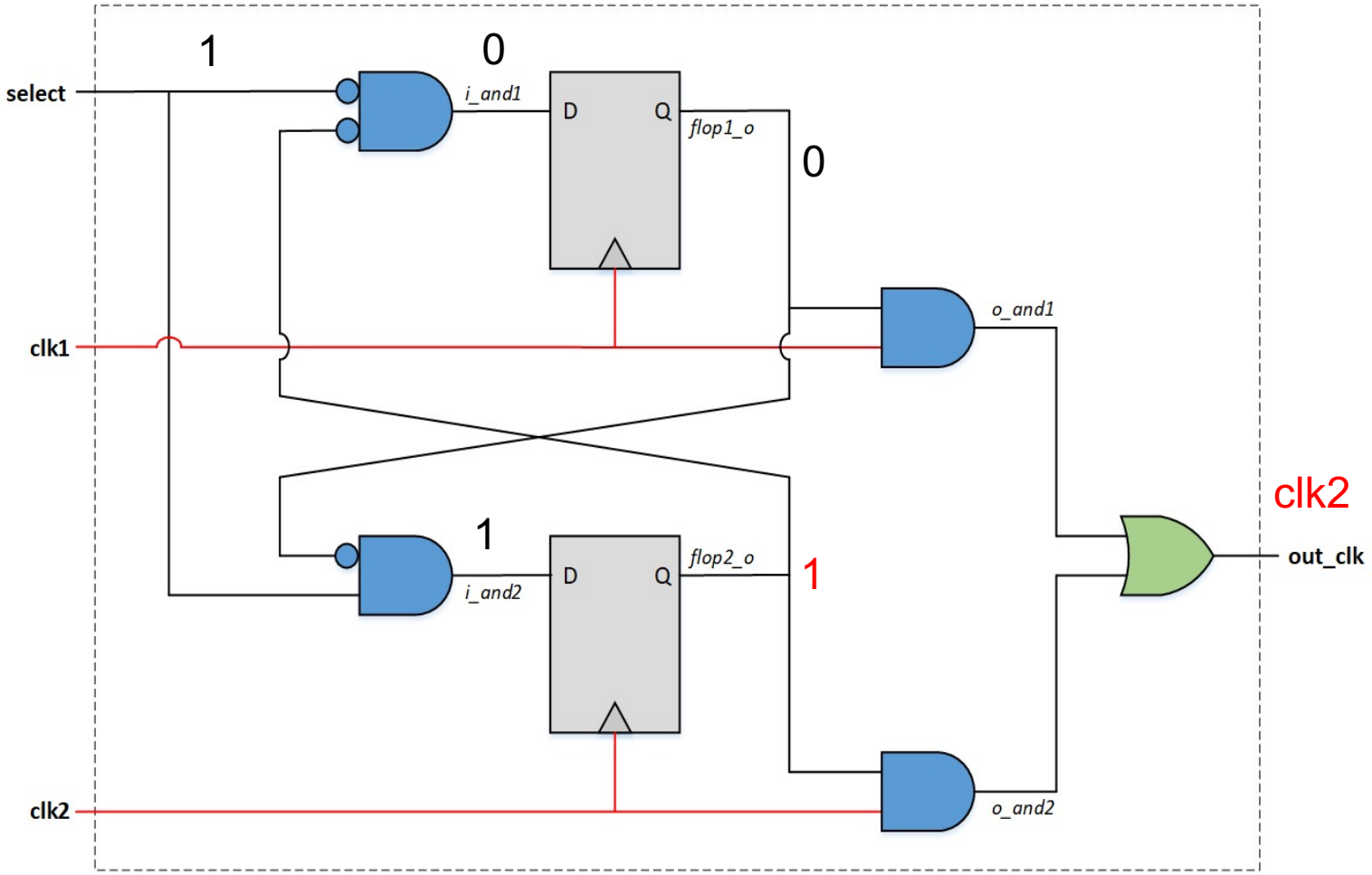
Glitch-free Clock Mux

clk rising edge has come



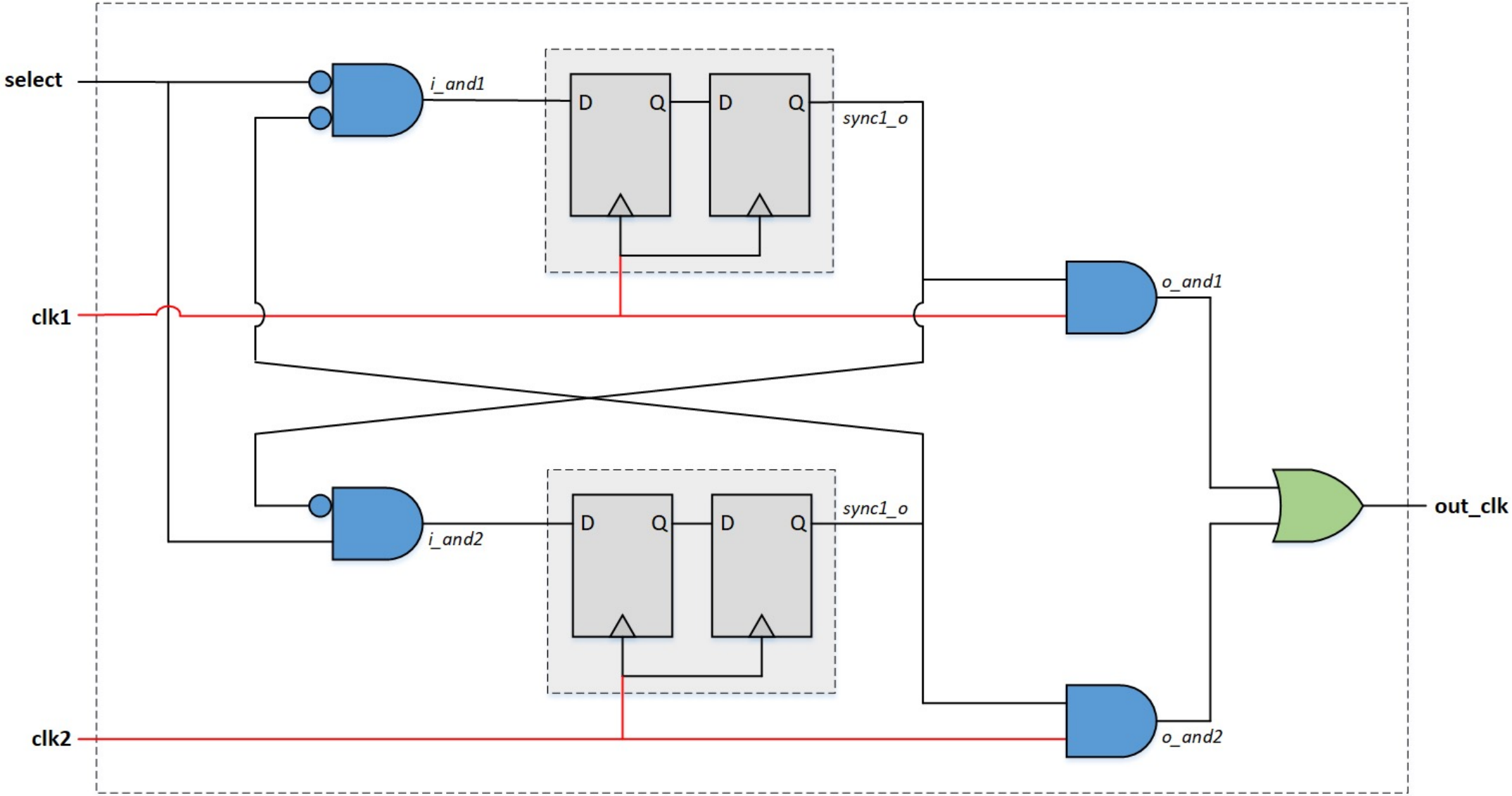
Glitch-free Clock Mux

Another clk rising edge has come



But, still there is an issue of asynchronous interface between select and clk1 / clk2

Glitch-free Clock Mux with Synchronizer



Loop Unrolling

Loop Unrolling (Unfolding)

```
for (i = 0; i < N; i++) {
 a[i]=b[i]*c[i];
}
```

```
for (i = 0; i < N/2; i++) {
 a[i*2] = b[i*2]*c[i*2];
 a[i*2 + 1] = b[i*2 + 1]*c[i*2 + 1];
 }
```

- Loop unrolling

- when there is some resource (time or area) budget, compute multiple terms in a single stage

- frequency can be reduced with lower number of loop iterations

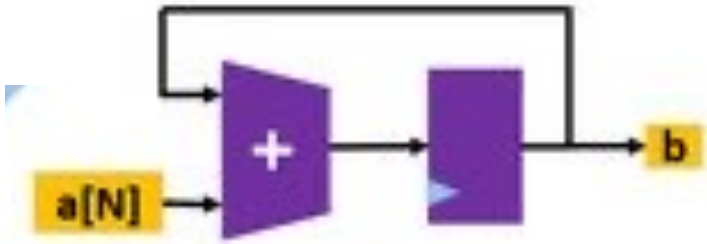
- reduced loop management overhead, e.g., incrementing counter, checking condition

  -> potential power savings

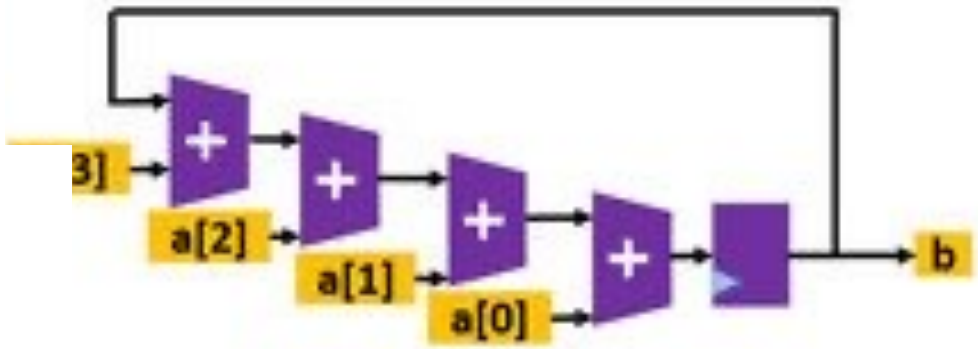
- opposite (folding) is also possible for frequency increase

Hardware Example for Loop Unrolling

```
for (i=0; i<N; i++) {
  b[i] = a[i] + b[i]
}
```




```
for (i=0; i<N/4; i++) {
  b = a[4*i]+a[4*i+1]+a[4*i+2]+a[4*i+3]+b
}
```

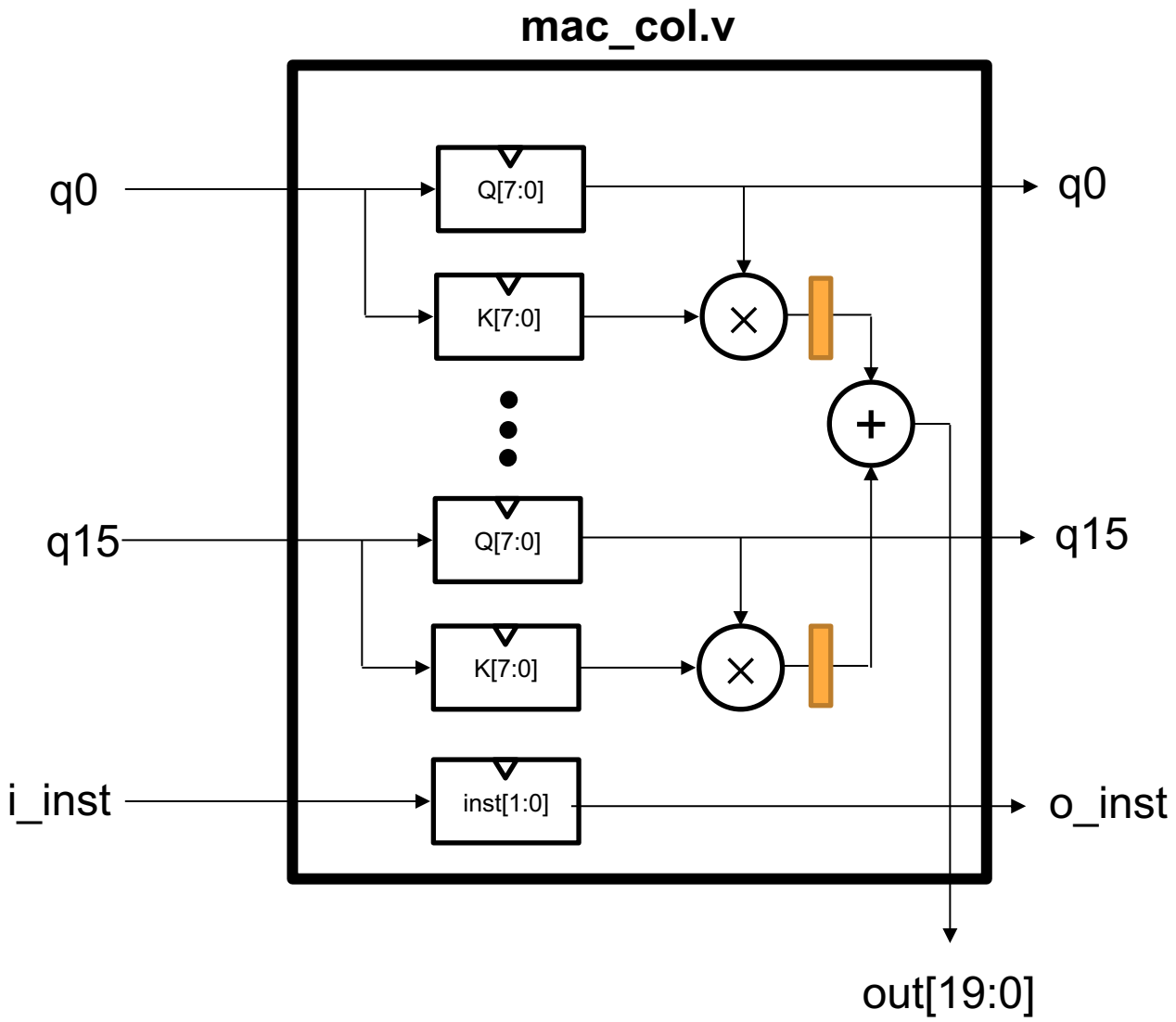


https://www.mdpi.com/2079-9292/9/3/449/htm

[HW] Pipeline of 16-tap Inner Product Computing

- 16 tap inner product computing with 8-bit for each operand
- e.g., out = a[0]*b[0] + … a[15]+b[15]
- Synthesis with typical corner targeting 1GHz freq with io_delay 0.2 ns
- Measure the worst negative slack, area, and power
- Now, create a pipeline stage right after multiplication.
- Run the iverilog simulation to verify your pipelined version’s functionality
- Synthesize again the pipelined version with the same condition
- Compare the worst negative slack, area, and power

mac_col.v



Inst[0]: K loading into register

Inst[1]: execute