Clock and Synchronization

Outline

1. Why synchronous?
2. Clock distribution network and skew
3. Multiple-clock system
4. Meta-stability and synchronization failure
5. Synchronizer

1. Why synchronous

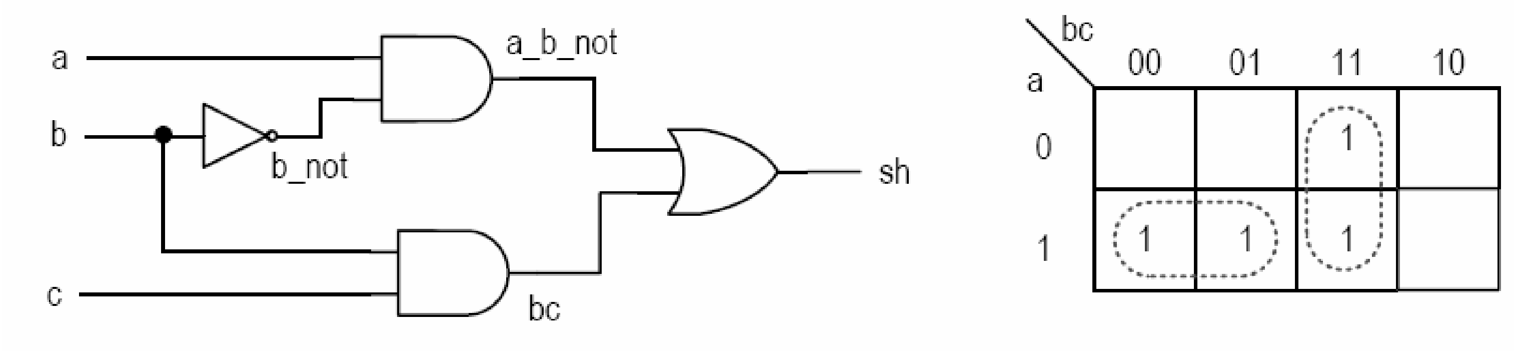
Timing of a combinational digital system

- Steady state
  - Signal reaches a stable value
  - Modeled by Boolean algebra
- Transient period
  - Signal may fluctuate
  - No simple model
- Propagation delay: time to reach the steady state

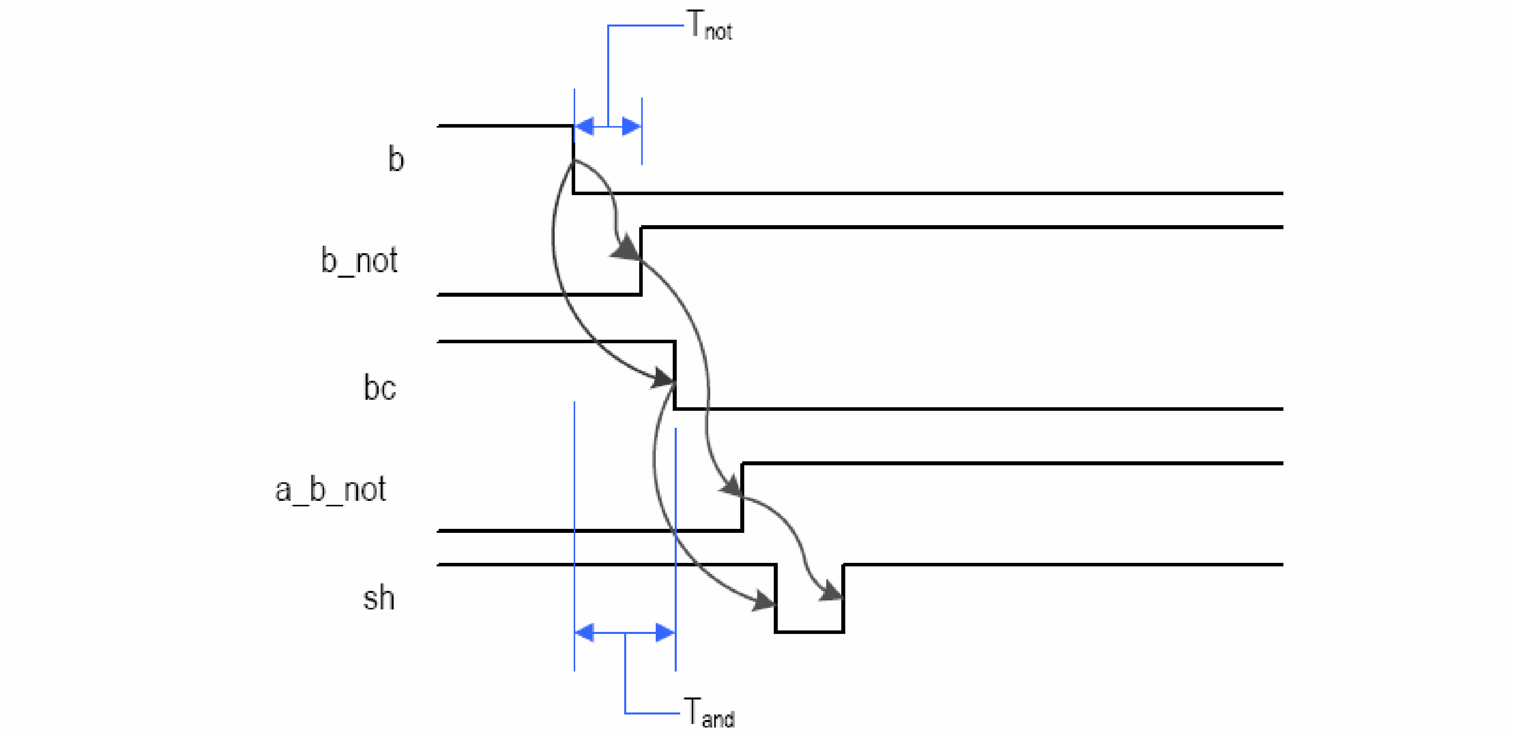
Timing Hazards

- Hazards: the fluctuation occurring during the transient period
  - – Static hazard: glitch when the signal should be stable
  - – Dynamic hazard: a glitch in transition
- Due to the multiple converging paths of an output port

- E.g., static-hazard (sh=ab’+bc; a=c=1)

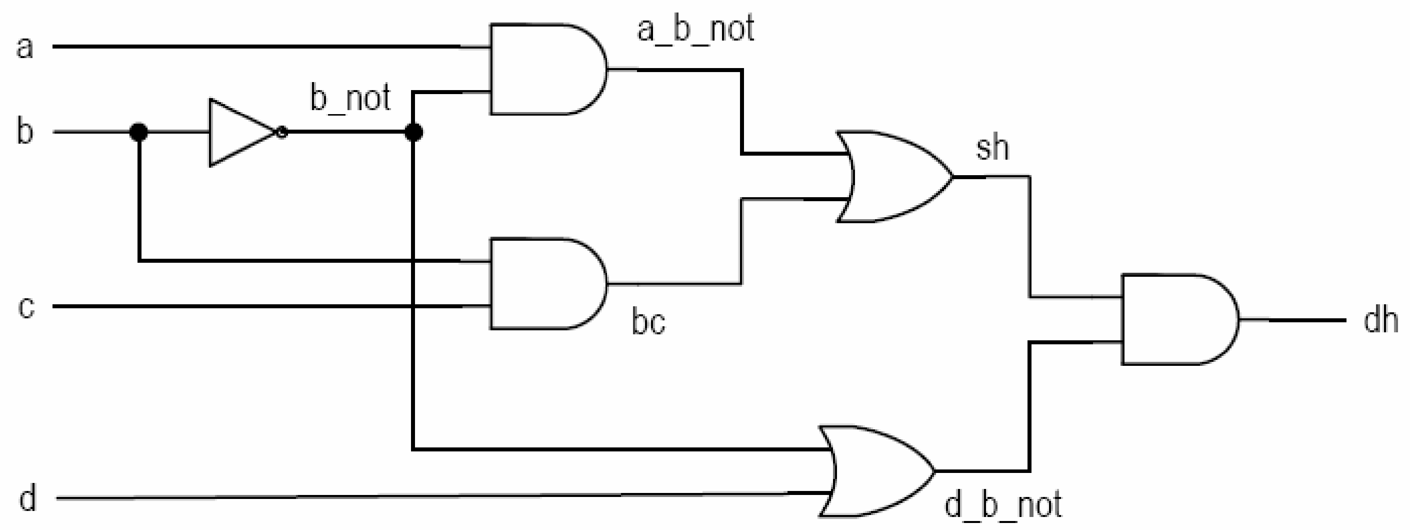


(a) Karnaugh map and schematic

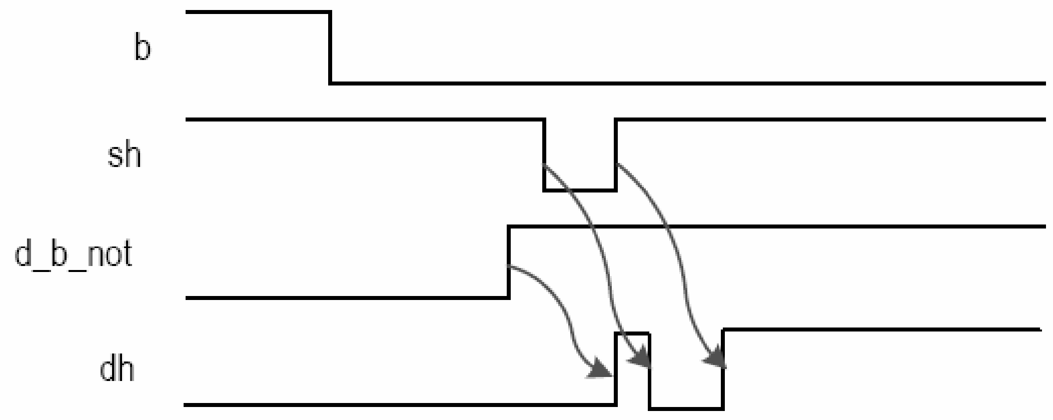


(b) Timing diagram

- E.g., dynamic hazard (a=c=d=1)

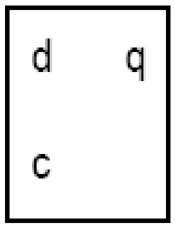


(a) Schematic



(b) Timing diagram

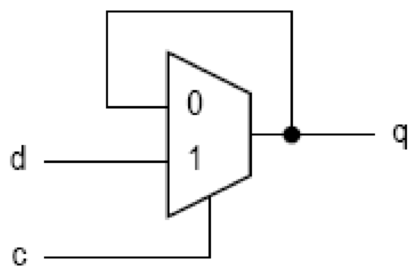
E.g., Hazard of circuit with closed feedback loop (async seq circuit)

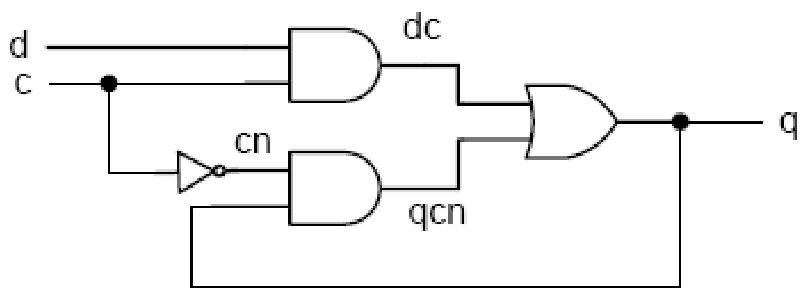


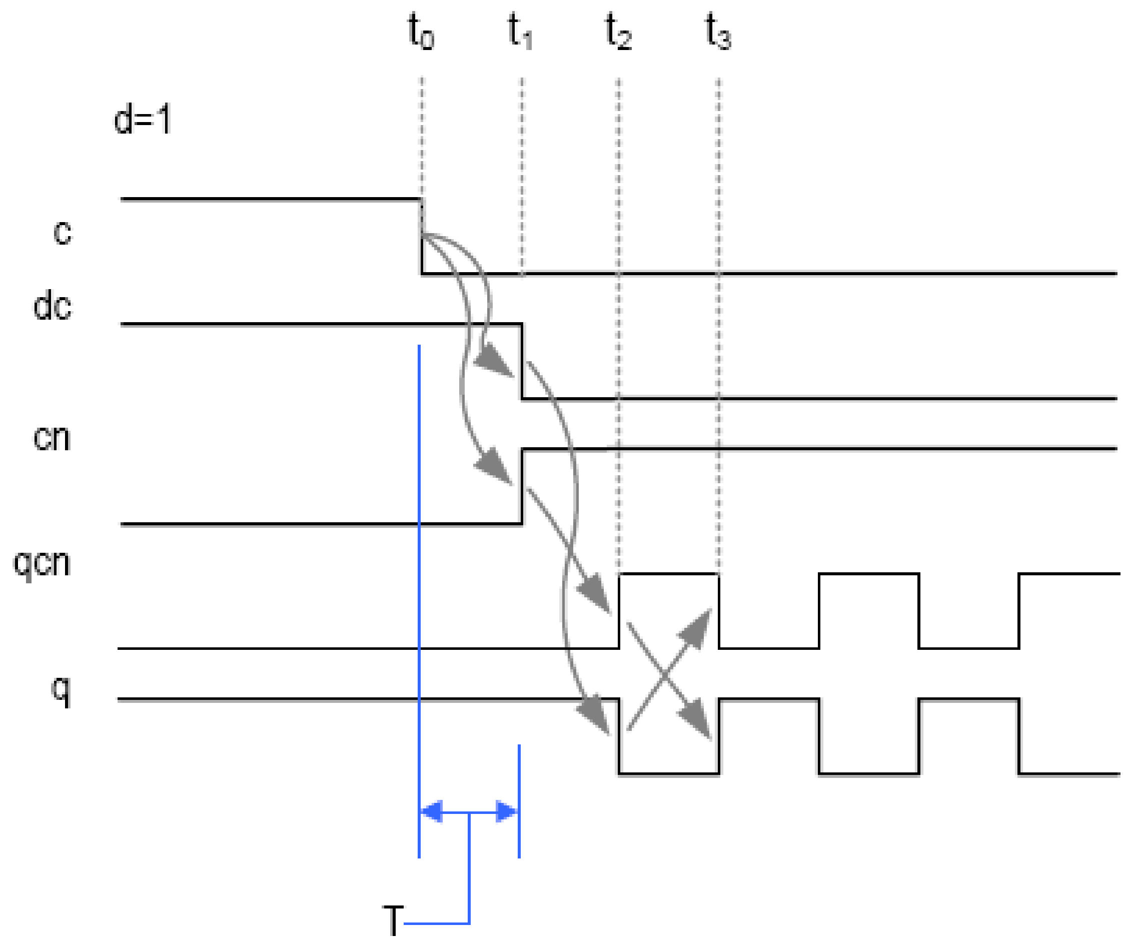
c	q*
0	q
1	d

(a) D latch

```
library ieee;
use ieee.std_logic_1164.all;
entity dlatch is
   port(
      c: in std_logic;
      d: in std_logic;
      q: out std_logic
   );
end dlatch;
architecture demo_arch of dlatch i
   signal q_latch: std_logic;
begin
   process (c, d, q_latch)
   begin
      if (c='1') then
         q_latch <= d;
      else
         q_latch <= q_latch;
      end if;
   end process;
   q <= q_latch;
end demo_arch;
```

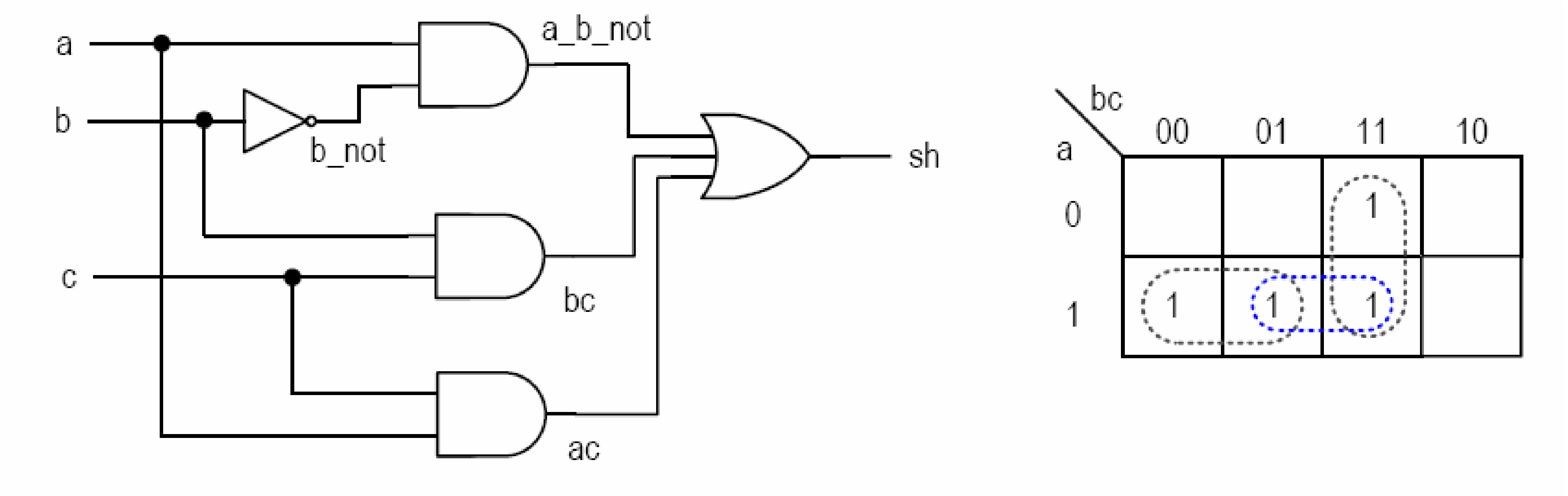

0
1
d
c
q


d
c
dc
cn
qcn
q


t_0
t_1
t_2
t_3
d=1
c
dc
cn
qcn
q
T

Dealing with hazards

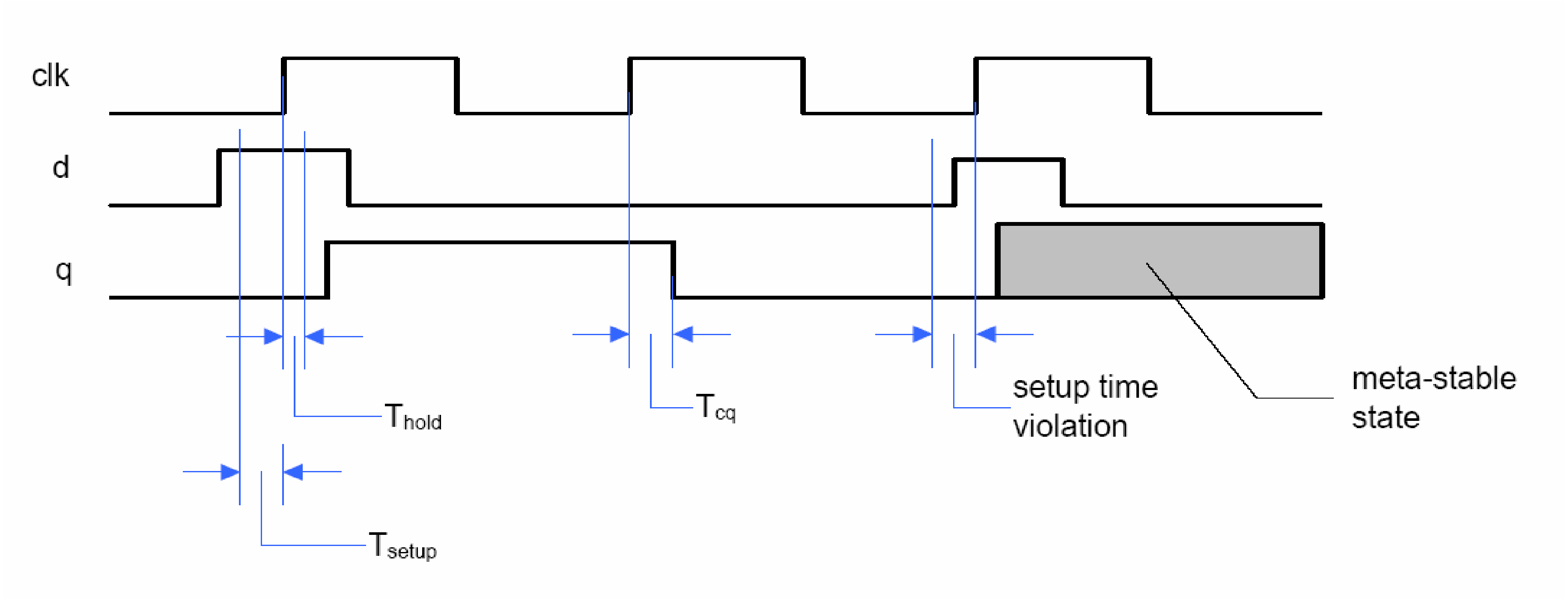
- In a small number of cases, additional logic can be added to eliminate race (and hazards).



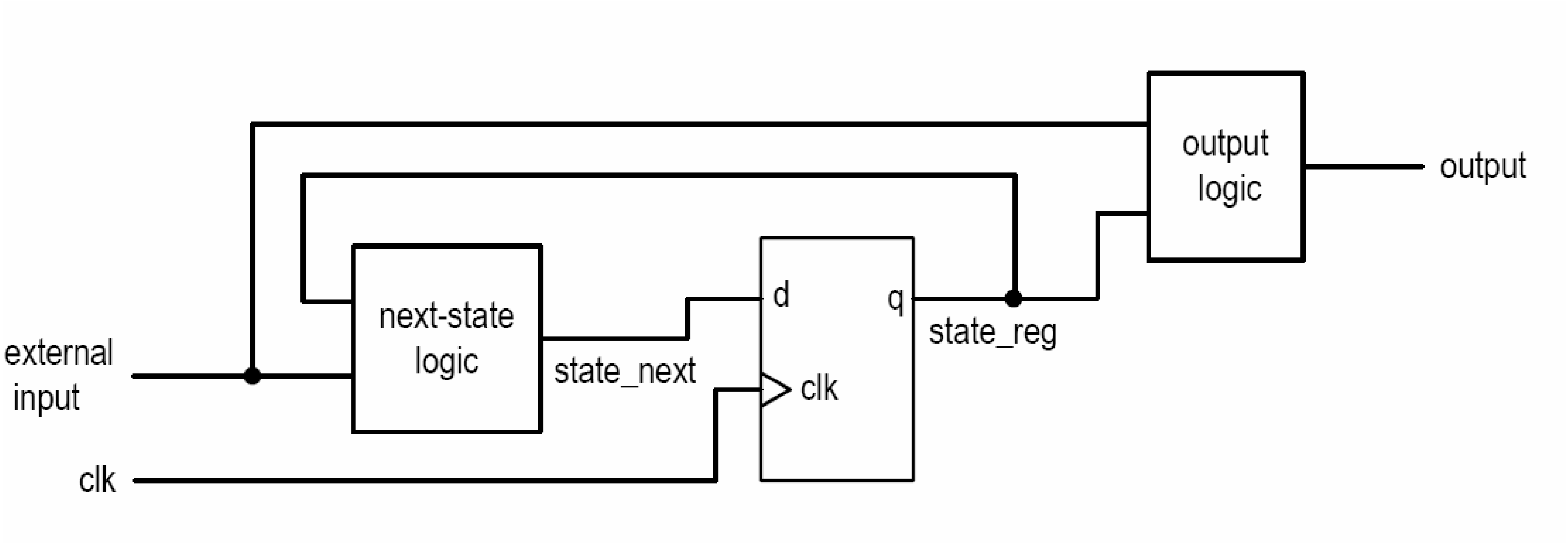
(c) Revised Karnaugh map and schematic to eliminate hazards

- This is not feasible for synthesis
- What’s can go wrong:
  - During logic synthesis, the logic expressions will be rearranged and optimized.
  - During technology mapping, generic gates will be re-mapped
  - During placement & routing, wire delays may change
  - It is bad for testing verification

- Better way to handle hazards
  - Ignore glitches in the transient period and retrieve the data after the signal is stabilized
- In a sequential circuit
  - Use a clock signal to sample the signal and store the stable value in a register.
  - But register introduces new timing constraint (setup time and hold time)



- Synchronous system:
  - group registers into a single group and drive them with the same clock
  - Timing analysis for a single feedback loop



Synchronous circuit and EDA

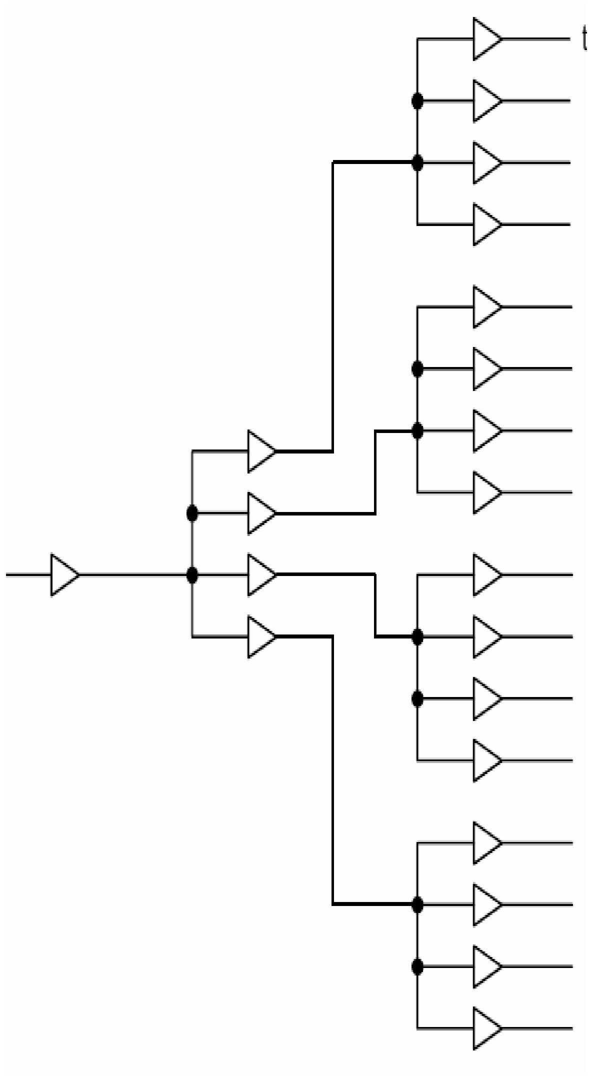
- Synthesis: reduce to combinational circuit synthesis
- Timing analysis: involve only a single closed feedback loop (others reduce to combinational circuit analysis)
- Simulation: support “cycle-based simulation”
- Testing: can facilitate scan-chain

2. Clock distribution network and skew

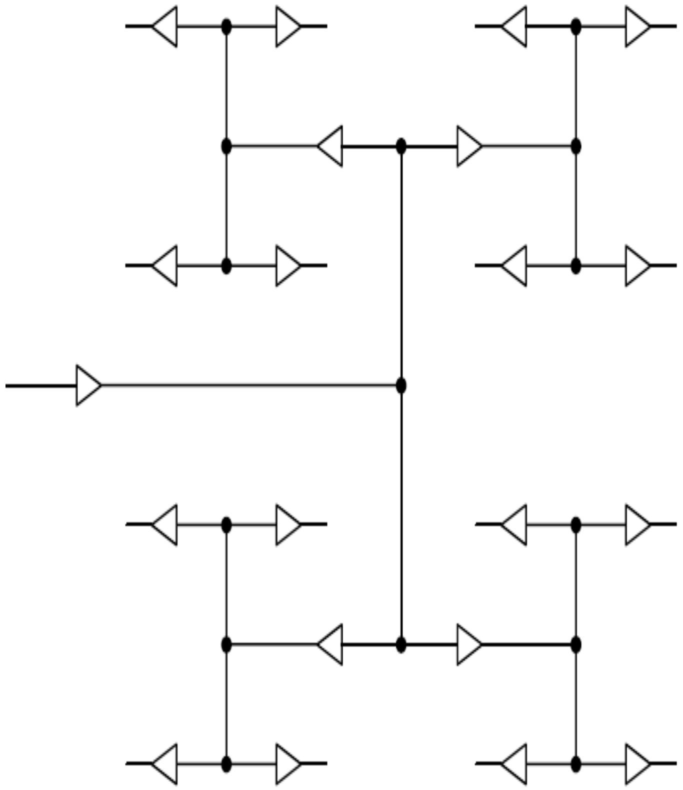
Clock distribution network

- Ideal clock: clock’s rising edges arrive at FFs at the same time
- Real implementation:
  – Driving capability of each cell is limited
  – Need a network of buffers to drive all FFs
  – In ASIC: done by clock synthesis (a step in physical synthesis)
  – In FPGA: pre-fabricated clock distribution network

- Block diagram

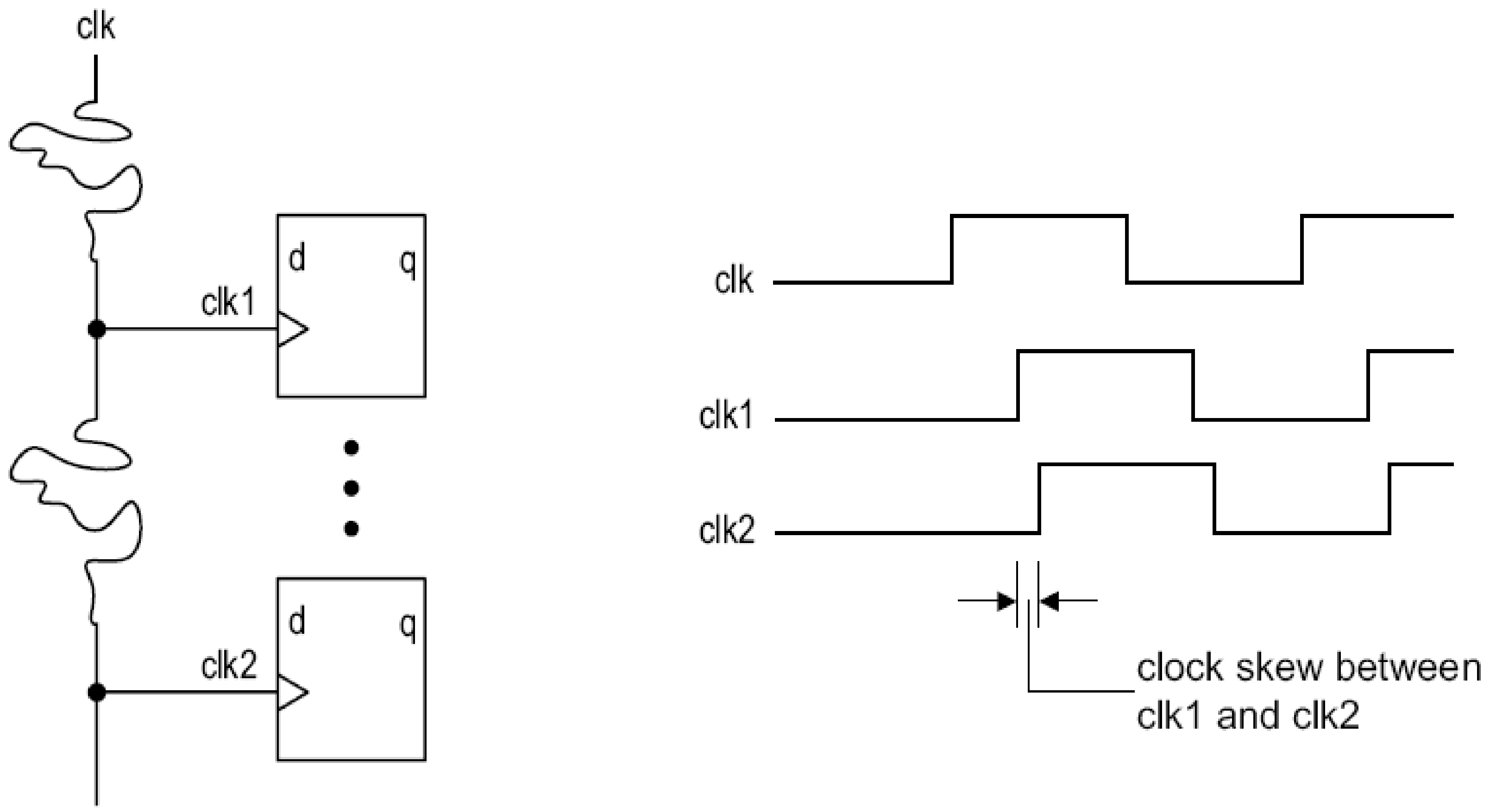


- Ideal H-routing



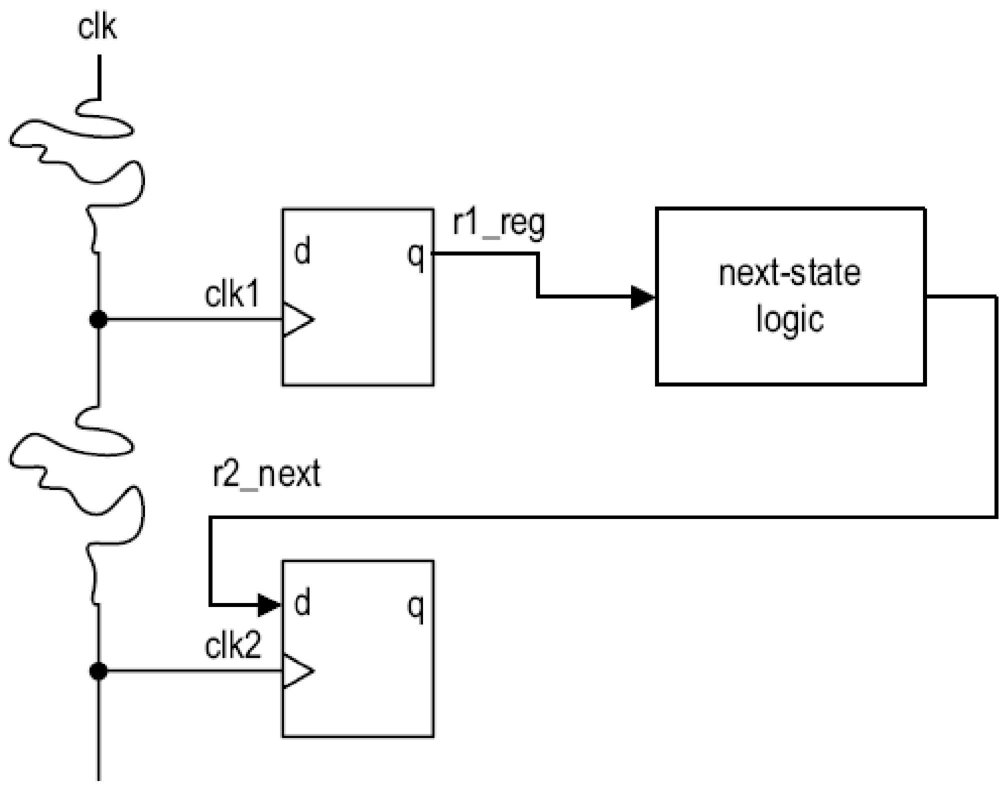
Clock skew

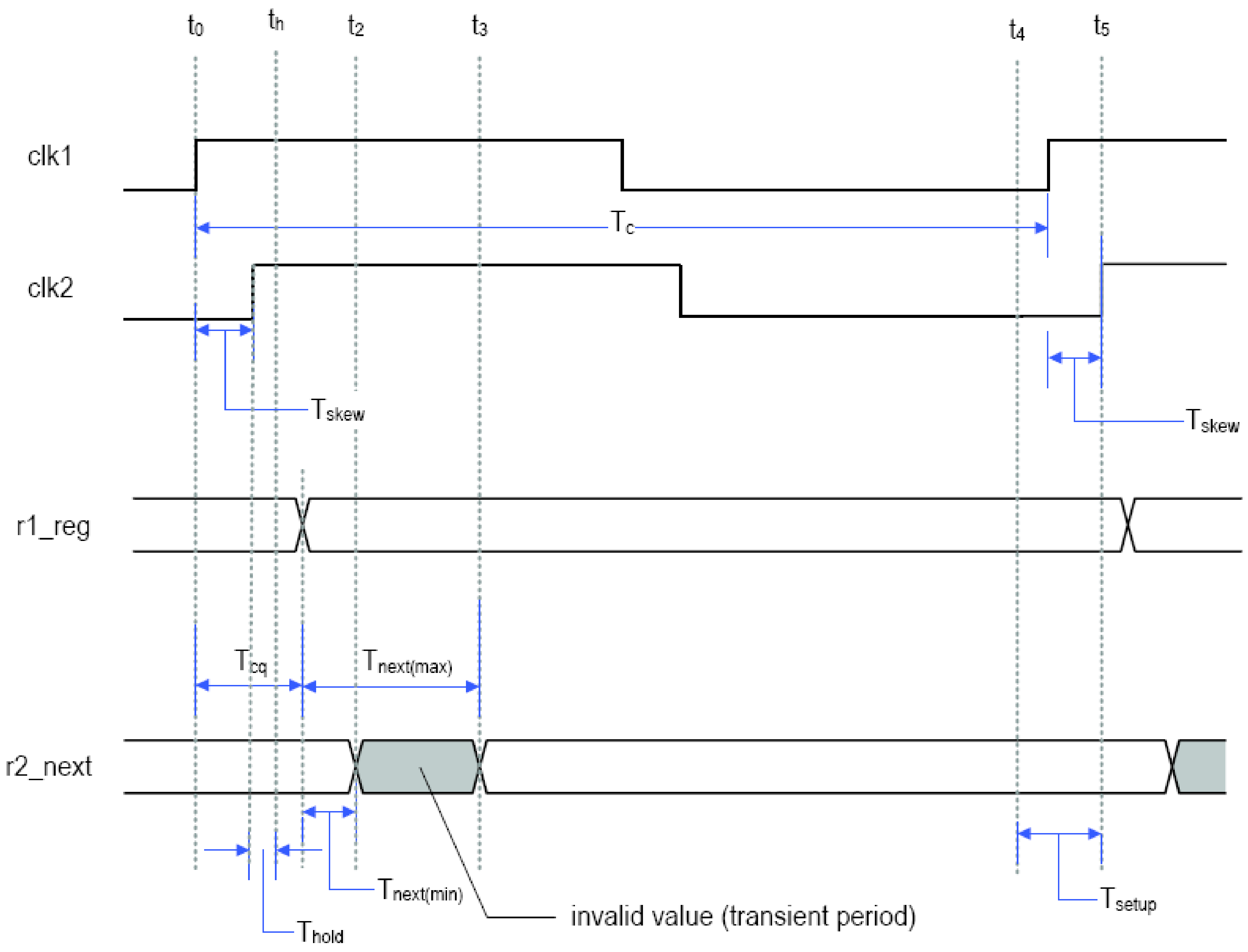
- Skew: time difference between two arriving clock edges



Timing analysis

- Setup time constraint (impact on max clock rate)
- Hold time constraint





t_0
t_h
t_2
t_3
t_4
t_5
clk1
T_c
clk2
T_{skew}
T_{skew}
r1_reg
T_{cq}
$T_{next(max)}$
r2_next
$T_{next(min)}$
T_{hold}
invalid value (transient period)
T_{setup}

$$t_3 < t_4$$

$$t_3 = t_0 + T_{cq} + T_{next(max)}$$

$$t_4 = t_5 - T_{setup} = (t_0 + T_c + T_{skew}) - T_{setup}$$

$$T_{cq} + T_{next(max)} + T_{setup} - T_{skew} < T_c$$

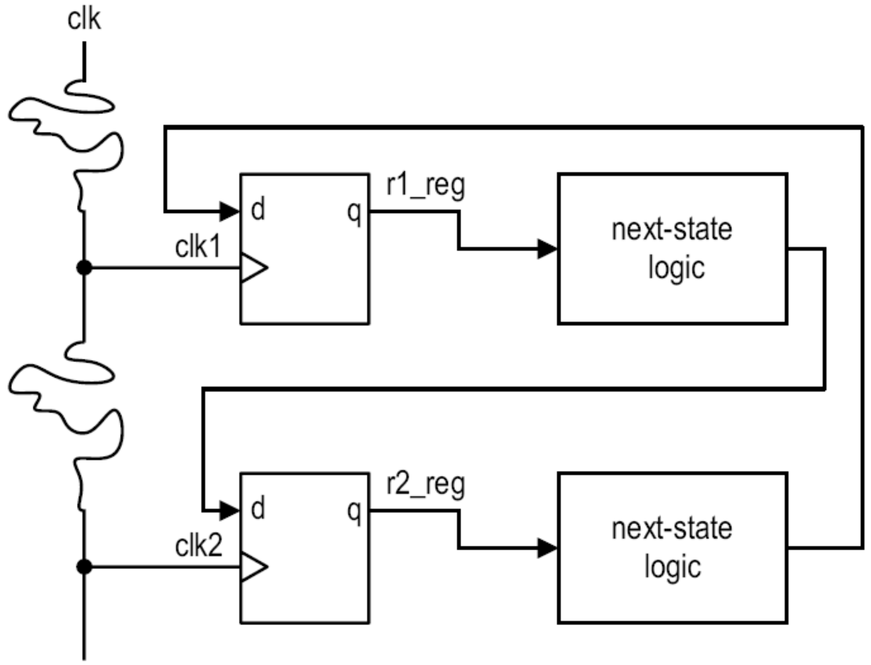
$$T_{c(min)} = T_{cq} + T_{next(max)} + T_{setup} - T_{skew}$$

- Clock skew actually helps increasing clock rate in this particular case

- If the clock signal travels from the opposite direction

$$T_{c(min)} = T_{cq} + T_{next(max)} + T_{setup} + T_{skew}$$

- Normally we have to consider the worst case since
  - No control on clock routing during synthesis
  - Multiple feedback paths



- Hold time constraint

$$t_h < t_2$$

$$t_2 = t_0 + T_{cq} + T_{next(min)}$$

$$t_h = t_0 + T_{hold} + T_{skew}$$

$$T_{hold} < T_{cq} + T_{next(min)} - T_{skew}$$

$$T_{hold} < T_{cq} - T_{skew}$$

- Skew may reduce hold time margin
- Hold time violation cannot be corrected in RT level

- Summary
  - Clock skew normally has negative impact on synchronous sequential circuit
  - Effect on setup time constraint: require to increase clock period (i.e., reduce clock rate)
  - Effect on hold time constraint: may introduce hold time violation
    - Can only be fixed during physical synthesis: re-route clock; re-place register and comb logic; add artificial delay logic
  - Skew within 10% of clock period tolerable

3. Multiple-clock system

Why multiple clocks

- Inherent multiple clock sources
  - E.g., external communication link
- Circuit size
  - Clock skew increases with the # FFs in a system
  - Current technology can support up to 10^4 FFs
- Design complexity
  - E.g., as sysetm w/ 16-bit 20 MHz processor, 1-bit 100 MHz serial interface, 1 MHz I/O controller
- Power consideration
  - Dynamic power proportional to switching freq

Derived vs Independent clocks

- Independent clocks:
  - Relationship between the clocks is unknown
- Derived clocks:
  - A clock is derived from another clock signals (e.g., different clock rate or phase)
  - Relationship is known
  - Logic for the derived clock should be separated from regular logic and manually synthesized (e.g., special delay line or PLL)
  - A system with derived clock can still be treated and analyzed as a synchronous system

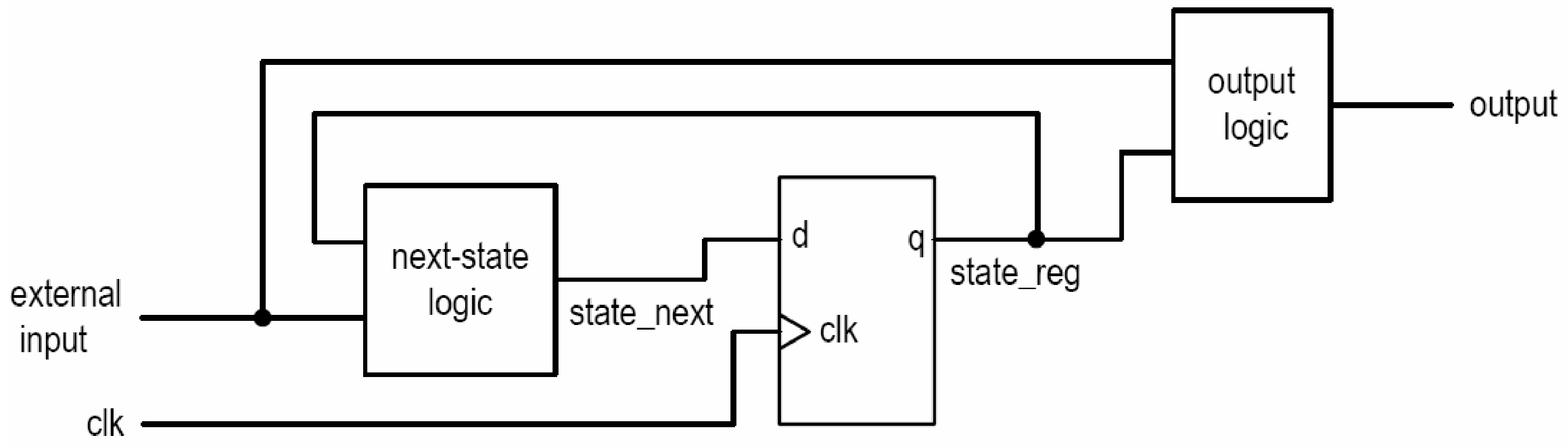
GALS

- Globally asynchronous locally synchronous system
  - Partition a system into multiple clock domains
  - Design and verify subsystem in same clock domain as a synchronous system
  - Design special interface between clock domains

4. Meta-stability and synchronization failure

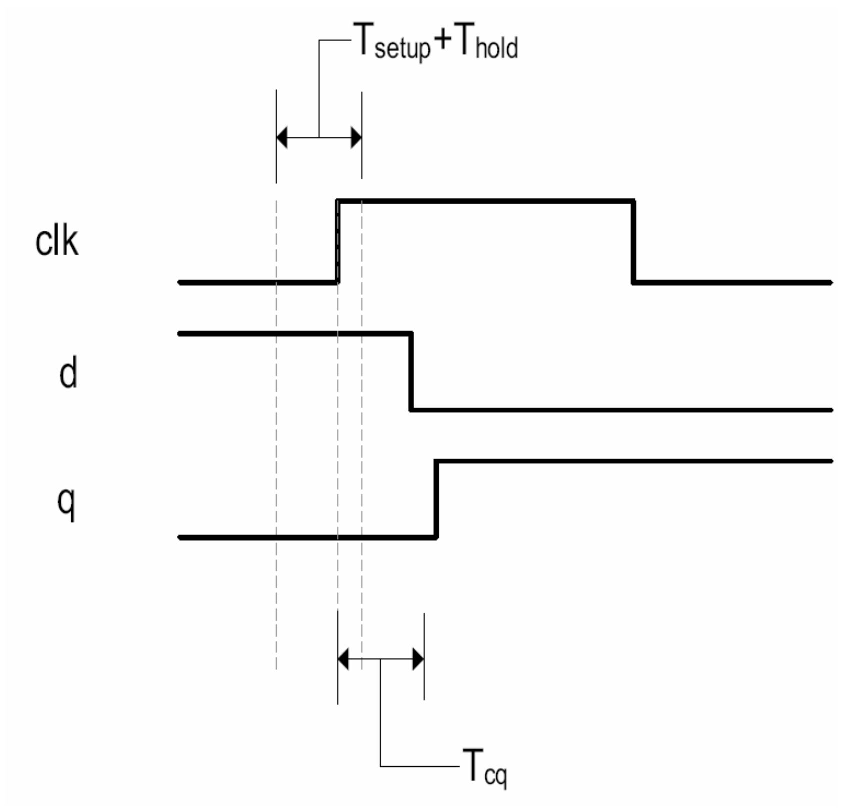
Timing analysis of a synchronous system

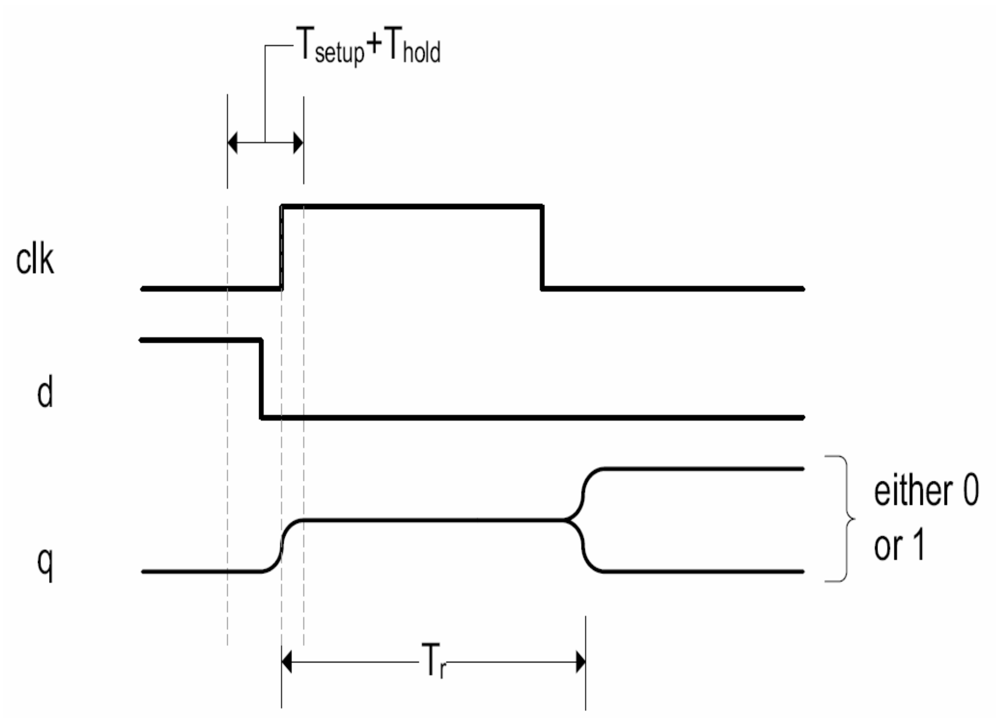
- To satisfy setup time constraint:
  - Signal from the state register
    - Controlled by clock
    - Adjust clock period to <u>avoid</u> setup time violation
  - Signal from external input
    - Same if the external input comes from another synchronous subsystem
    - Otherwise, have to <u>deal with the occurrence</u> of setup time violation.


output logic
output
next-state logic
d
q
clk
state_next
state_reg
external input
clk

Metastability

- What happens after timing violation?





- Output of FF becomes 1 (sampled old input value)
- Output of FF becomes 0 (sampled new input value)
- FF enters metastable state, the output exhibits an "in-between" value
  - FF eventually "resolves" to one of stable states
  - The resolution time is a random variable with distribution function (τ is decay constant)

$$P(T_r) = e^{-\frac{T_r}{\tau}}$$

  - The probability that metastability persists beyond Tr (i.e., cannot be resolved within Tr)

MTBF(Tr)

- Synchronization failure
  - an FF cannot resolve the metastable condition within the given time
- MTBF
  - Mean Time Between synchronization Failures
  - Basic criterion for metastability analysis
  - Frequently expressed as a function of Tr (resolution time provided)

- MTBF computation

- R_{meta}: The average rate at which an FF enters the metastable state.
- $P(T_r)$: The probability that an FF cannot resolve the metastable condition within T_r.

$$R_{meta} = w * f_{clk} * f_d$$

w is the *susceptible time window*.

$$P(T_r) = e^{-\frac{T_r}{\tau}}$$

$$AF(T_r) = R_{meta} * P(T_r) = w * f_{clk} * f_d * e^{-\frac{T_r}{\tau}}$$

$$\text{MTBF}(T_r) = \frac{1}{AF(T_r)} = \frac{e^{\frac{T_r}{\tau}}}{w * f_{clk} * f_d}$$

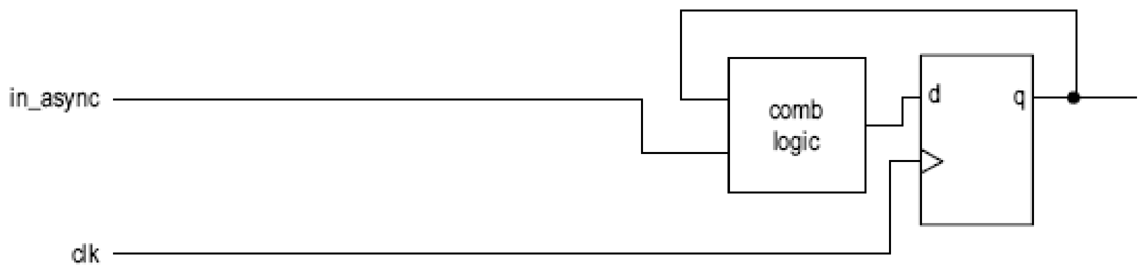
- E.g., w=0.1ns, τ=0.5ns, f_{clk}=50MHz, f_d=0.1f_{clk}

T_r	**MTBF**
0.0 ns	$4.00 * 10^{-05}$ sec (0.04 msec)
2.5 ns	$5.94 * 10^{-03}$ sec (5.94 msec)
5.0 ns	$8.81 * 10^{-01}$ sec (0.88 sec)
7.5 ns	$1.31 * 10^{+02}$ sec (131 sec)
10.0 ns	$1.94 * 10^{+04}$ sec (5.39 hours)
12.5 ns	$2.88 * 10^{+06}$ sec (3.33 days)
15.0 ns	$4.27 * 10^{+08}$ sec (1.36 years)
17.5 ns	$6.34 * 10^{+10}$ sec ($2.01 * 10^3$ years)
20.0 ns	$9.42 * 10^{+12}$ sec ($2.99 * 10^5$ years)
22.5 ns	$1.40 * 10^{+15}$ sec ($4.43 * 10^7$ years)
25.0 ns	$2.07 * 10^{+17}$ sec ($6.58 * 10^9$ years)
27.5 ns	$3.08 * 10^{+19}$ sec ($9.76 * 10^{11}$ years)
30.0 ns	$4.57 * 10^{+21}$ sec ($1.45 * 10^{14}$ years)
32.5 ns	$6.78 * 10^{+23}$ sec ($2.15 * 10^{16}$ years)
35.0 ns	$1.01 * 10^{+26}$ sec ($3.19 * 10^{18}$ years)

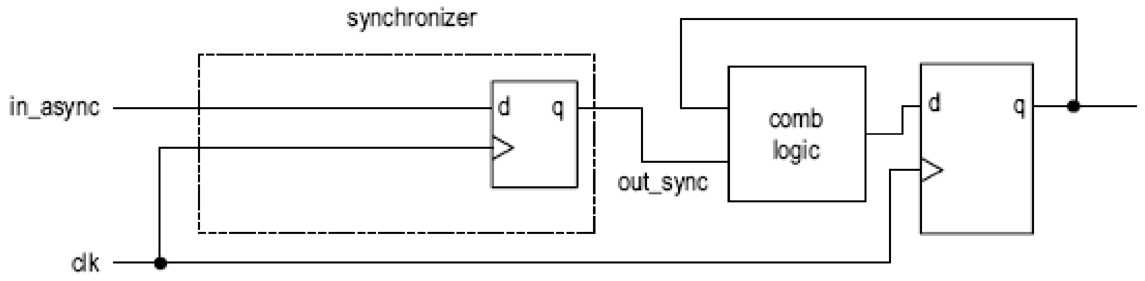
- Observations
  - MTBF is statistical average
  - Only Tr can be adjusted in practical design
  - MTBF is extremely sensitive to Tr
    - Good: synchronization failure can be easily avoided by providing additional resolution time
    - Bad: minor modification can introduce synchronization failure

5. Synchronizer

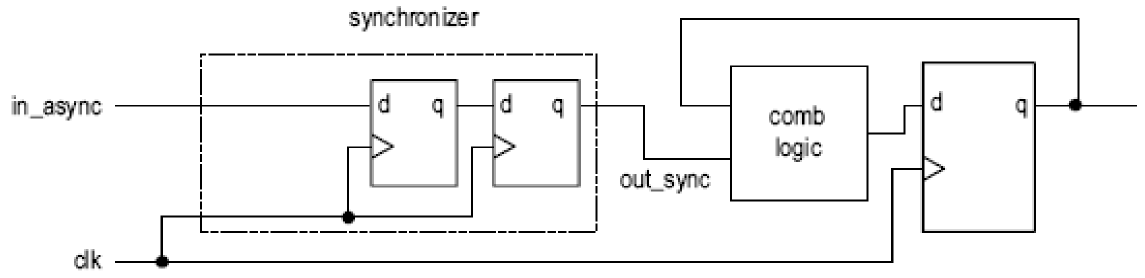
- Synchronization circuit:
  - Synchronize an asynchronous input with system clock
  - No physical circuit can prevent metastability
  - Synchronizer just provides enough time for the metastable condition to be “resolved”
- E.g.,
  - $w=0.1ns$, $\tau=0.5ns$, $f_{clk}=50MHz$, $f_d=0.1f_{clk}$
  - $T_{setup}=2.5s$



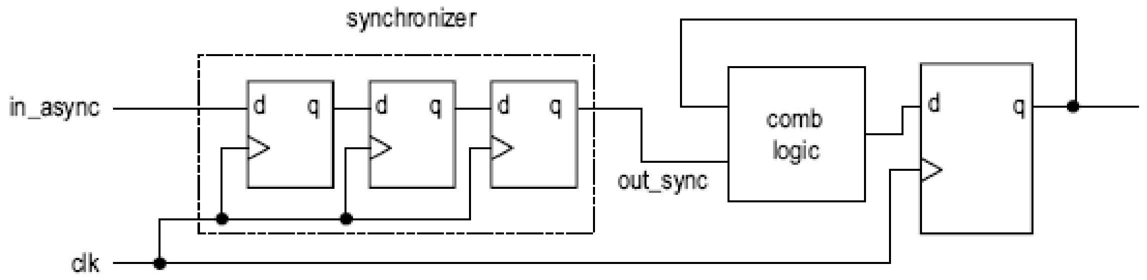
(a) No synchronizer



(b) One-FF synchronizer



(c) Two-FF synchronizer



(d) Three-FF synchronizer

No synchronizer

- $T_r = 0$
- MTBF(0) = 0.04 ms

One-FF synchronizer

- $T_r = T_c – (T_{comb} + T_{setup})$
- T_r depends on T_c , T_{setup} and T_{comb}
  - T_c: vary with system specification
  - T_{comb}: vary with circuit, synthesis (gate delay), placement & routing (wire delay)
- E.g.,
  - $T_r = 20 – (T_{comb} + 2.5) = 17.5 – T_{comb}$
  - T_{comb} = 1ns, T_r = 16.5ns; MTBF(16.5) = 272yr
  - T_{comb} = 12.5ns, T_r = 5ns; MTBF(5) = 0.88ns
- Not a reliable design

Two-FF synchronizer

- Add an extra FF to eliminate T_{comb}
  - $T_r = T_c – T_{setup}$
  - T_r depends on T_c only
  - Async input delayed by two clock cycles
- E.g.,
  - T_r=20 - 2.5=17.5; MTBF(17.5)=3000yr
- Most commonly used synchronizer
- In ASIC technology
  - May have “metastability-hardened” D FF cell (large area)

Three-FF synchronizer

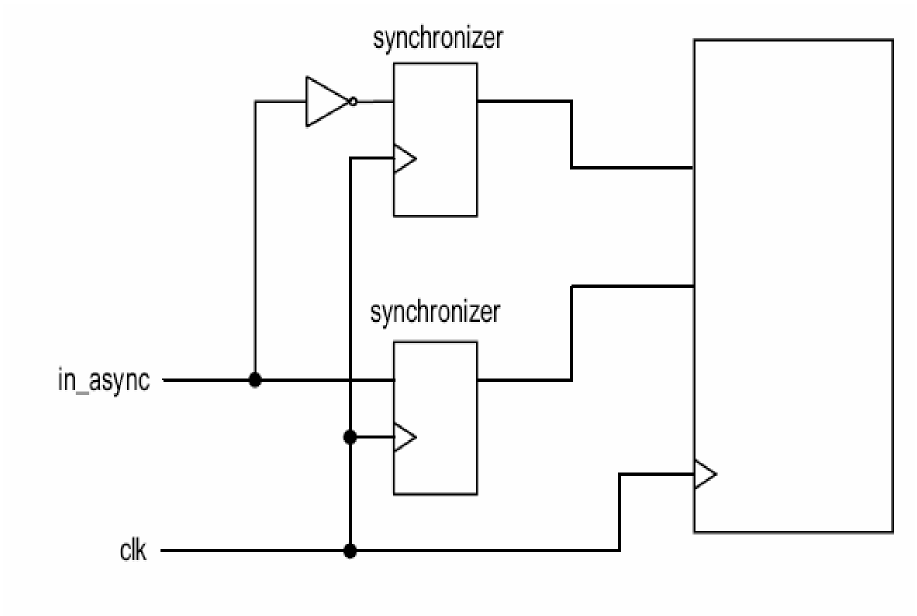
- Add an extra stage to increase resolution time
  - $T_r = 2(T_c – T_{setup)})$
  - Async input delayed by three clock cycles
- E.g.,
  - T_r =2*(20 - 2.5); MTBF(30)=6 billion yr
- Hardly needed

Observation

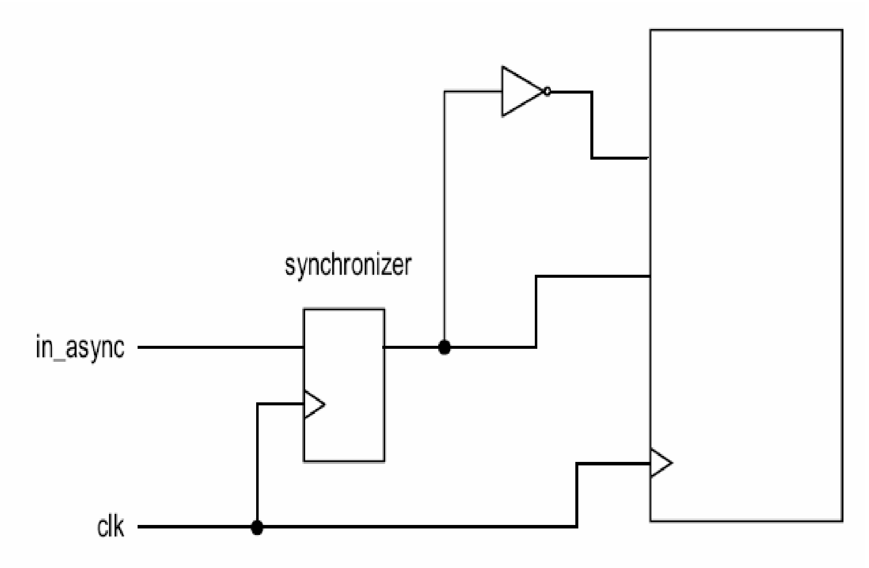
- T_r is in the exponent of MTBF equation
- Small variation in T_r can lead to large swing in MTBF

Proper use of synchronizer

- Use a glitch-free signal for synchronization
- Synchronize a signal in a single place
- Avoid synchronization multiple “related” signals.
- Reanalyze the synchronizer after each design change



(a) Synchronizing a signal in two places



(b) Synchronizing a signal in one place

Why synchronization is a “tricky” issue

- Metastability is basically an “analog” phenomena
- Metastability behavior is described by random variable
- Metastability cannot be easily modeled or simulated in gate level (only ‘X’)
- Metastability cannot be easily observed or measured in physical circuit (e.g., MTBF = 3 months)
- MTBF is very sensitive to circuit revision

6. Enable tick crossing clock domain

Signals crossing clock domains

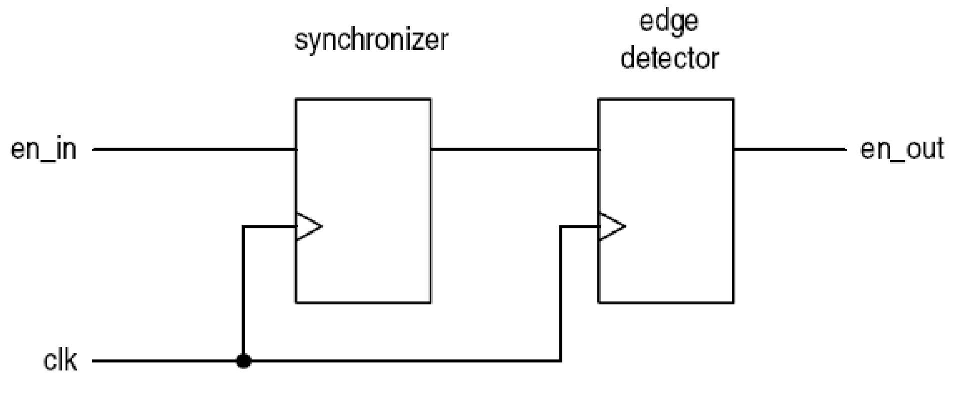
- Synchronizer
  - Just ensures that the receiving system does not enter a metastable state
  - Not guarantee the “function” of the received signal
- Consideration
  - One signal
  - Multiple signals (“bundled data”)

Domain-crossing of an enable signal

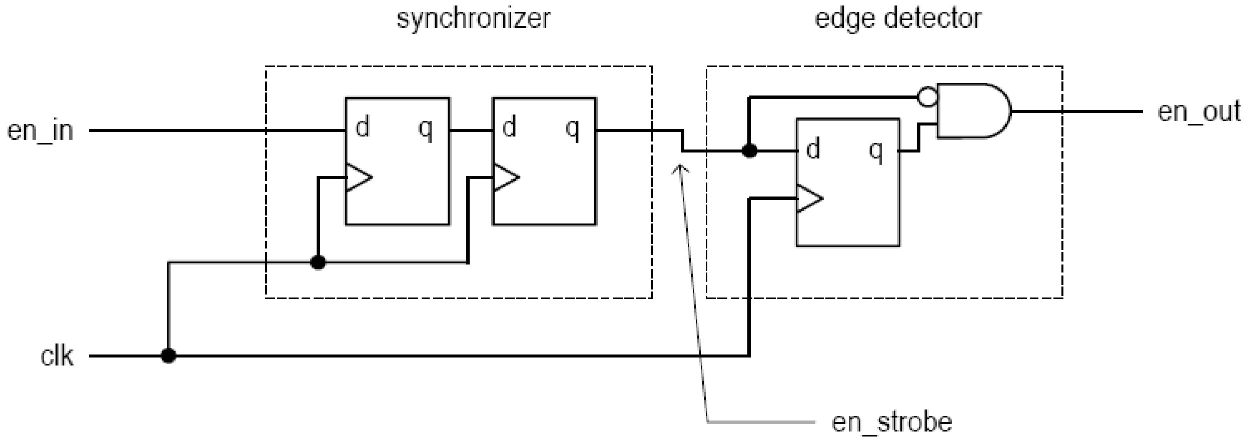
- An enable tick
  - One-clock-cycle wide
  - To be sample in a single clock edge
  - E.g., enable input of a counter; read/write signal of a FIFO buffer
  - Can also be used to retrieve bundled data

“Wide” enable signal

- From a slow clock domain to a fast clock domain (e.g., 1 MHz to 10 MHz)

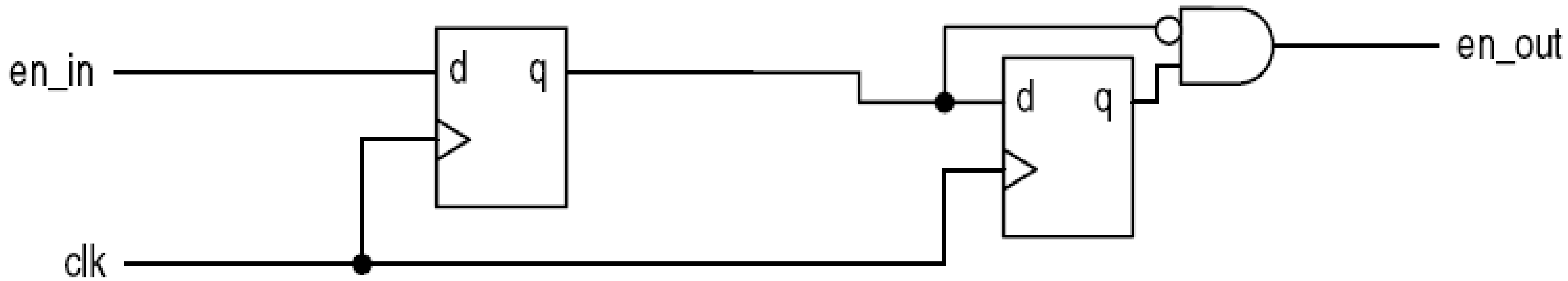


(a) Block diagram



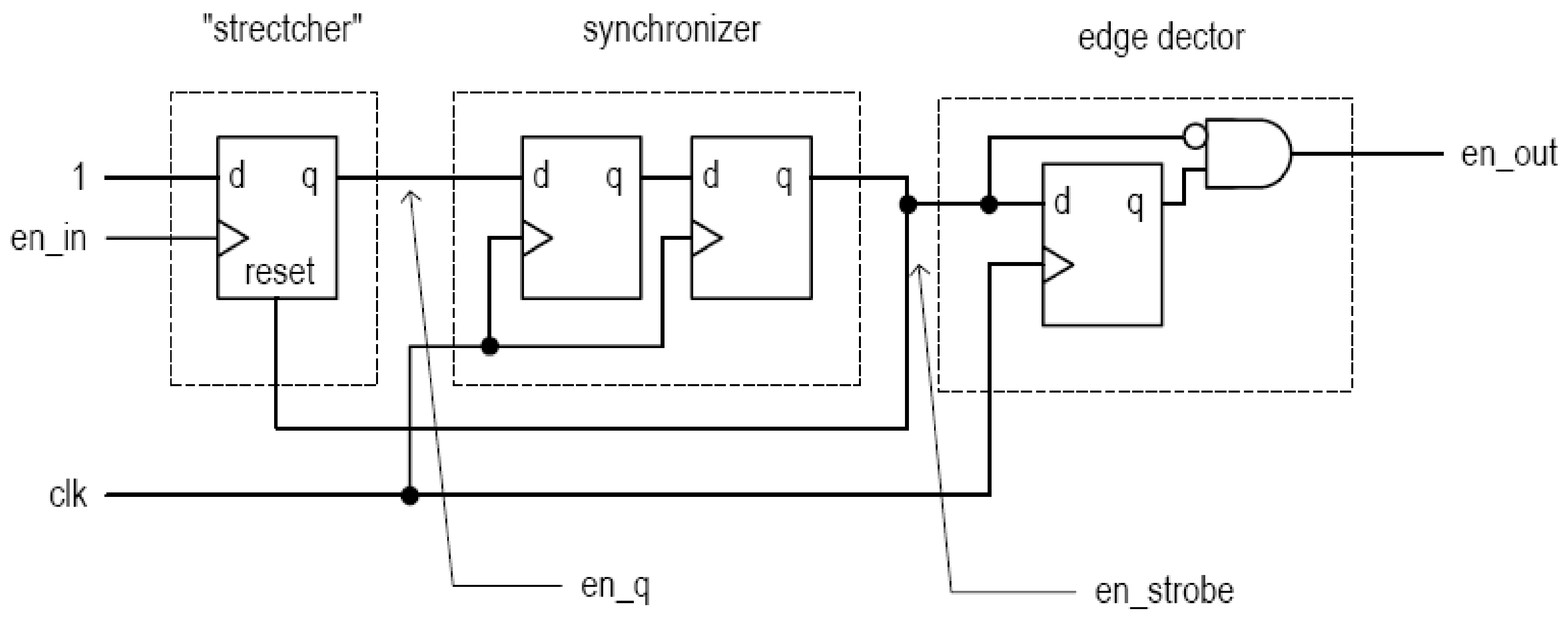
(b) Correct circuit diagram

- Will this work?



“Narrow” enable signal

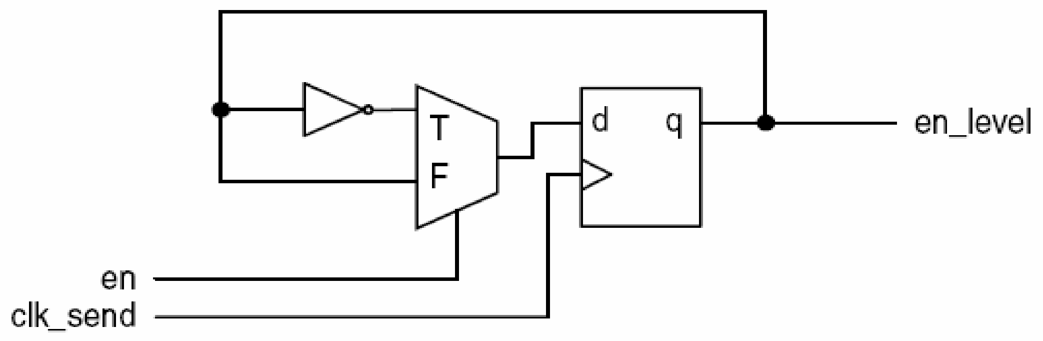
- From a fast clock domain to a slow clock domain (e.g., 10 MHz to 1 MHz)
- The enable pulse is probably to narrow to be detected
- Need to “stretch” the pulse
  - Cannot be done by a normal sequential circuit
  - Need to use “tricks”



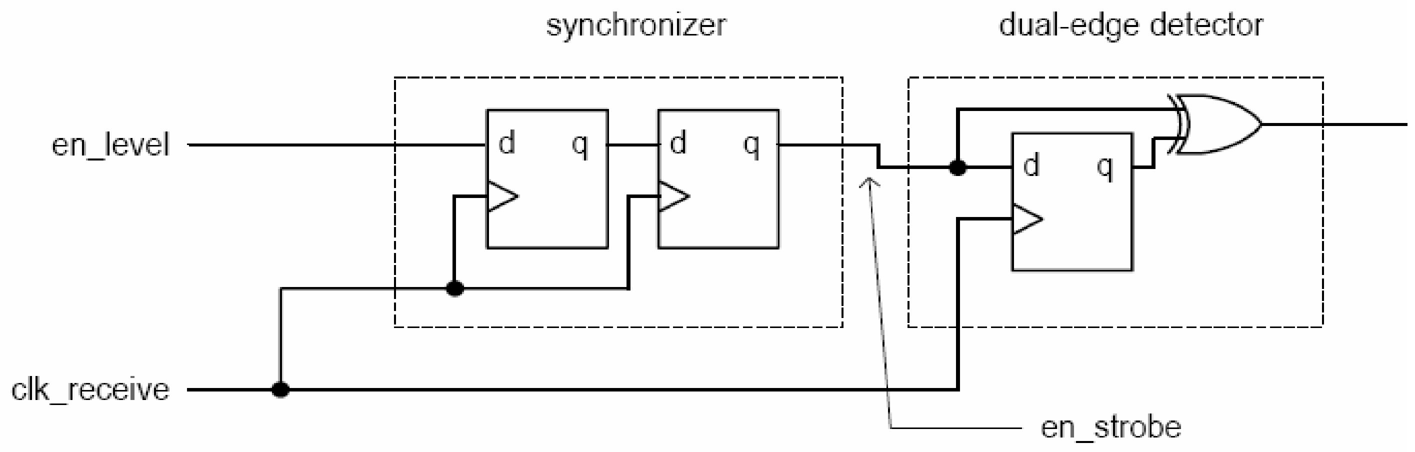
- en_q asserted at the rising edge of en_in
- en_q then synchronized
- en_strobe then clears stretcher
- en_q may last over two clock cycles and thus an edge-detector is needed
- Can this scheme be used for wide-pulse?

Level-alternating scheme

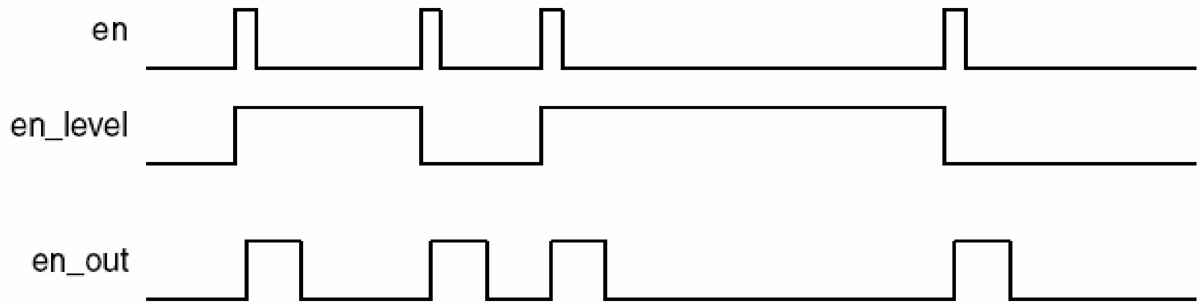
- Output interface of sender and input interface of receiver modified for domain crossing
- Output interface converts an “edge-sensitive” enable pulse to a level-alternating signal
  - Use a T-FF
- Input interface converts the level-alternating signal back to “edge-sensitive” enable pulse
  - Use a dual-edge detector
- Eliminate the ad-hoc stretcher and follow the synchronous design methodology



(a) Block diagram of the sending subsystem



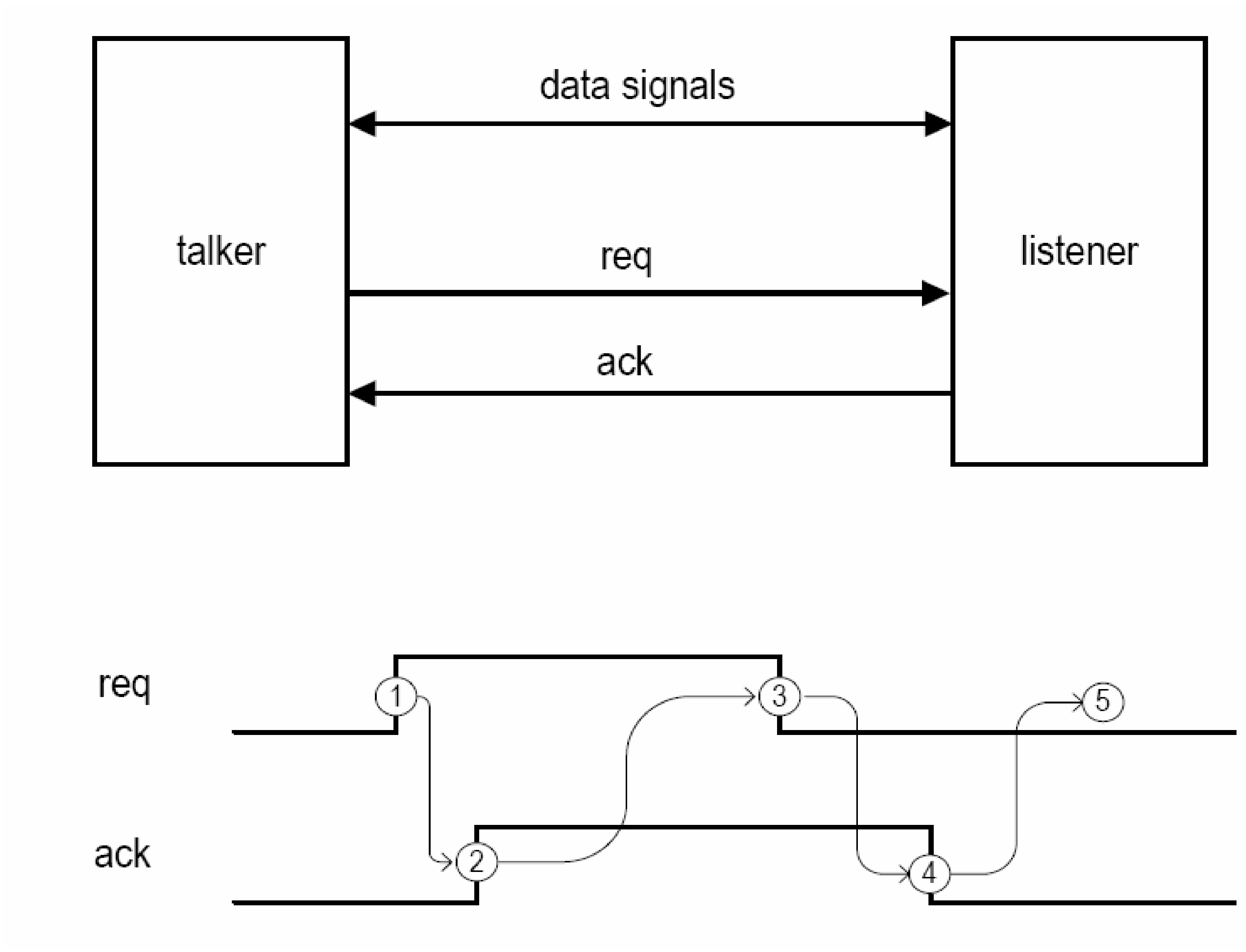
(b) Block diagram of the receiving subsystem

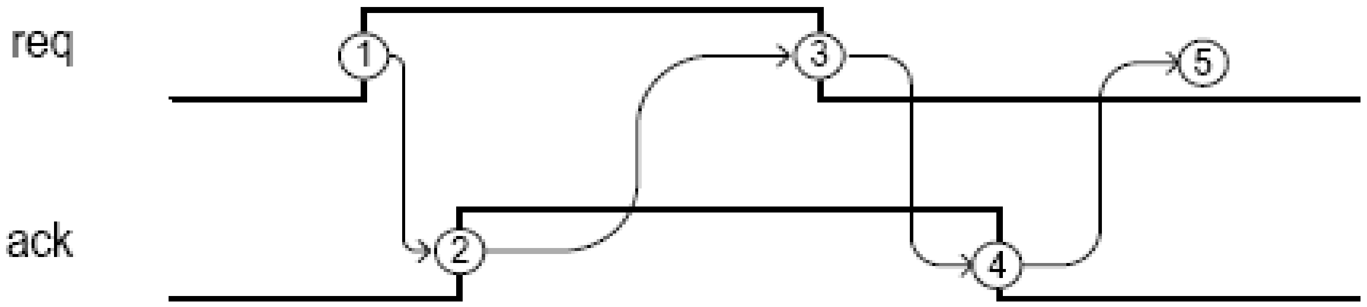


(c) Simplified timing diagram

6. Handshaking

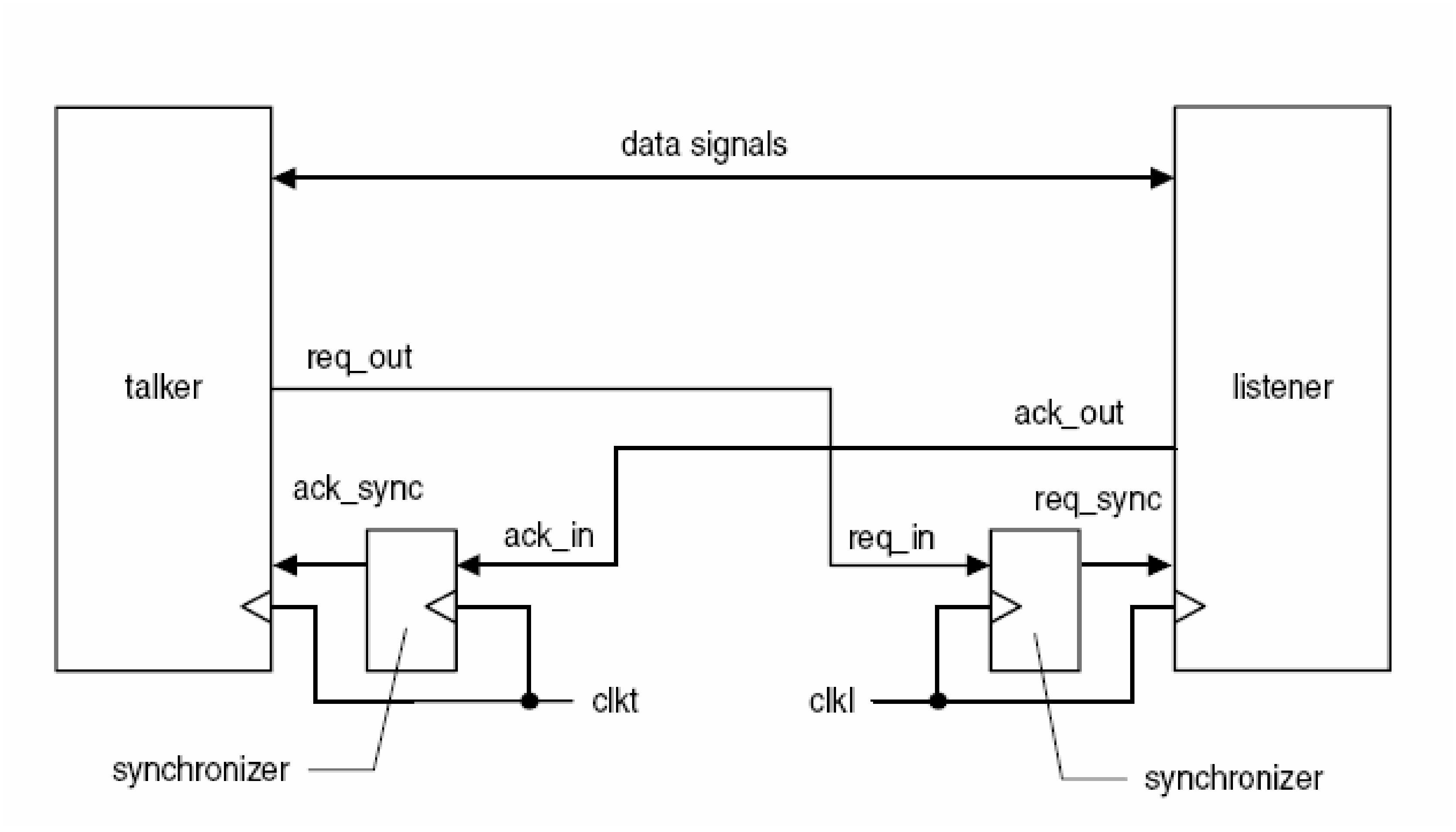
- How to control the rate of data (or number of enable ticks) between two clock domains? (e.g., 10 MHz system to 1 MHz system)
- Does the sending system have prior knowledge about the processing speed of receiving system?
- Handshaking scheme
  - Use a feedback signal
  - Make minimal assumption about the receiving system


data signals
talker
req
ack
listener
req
ack
1
2
3
4
5

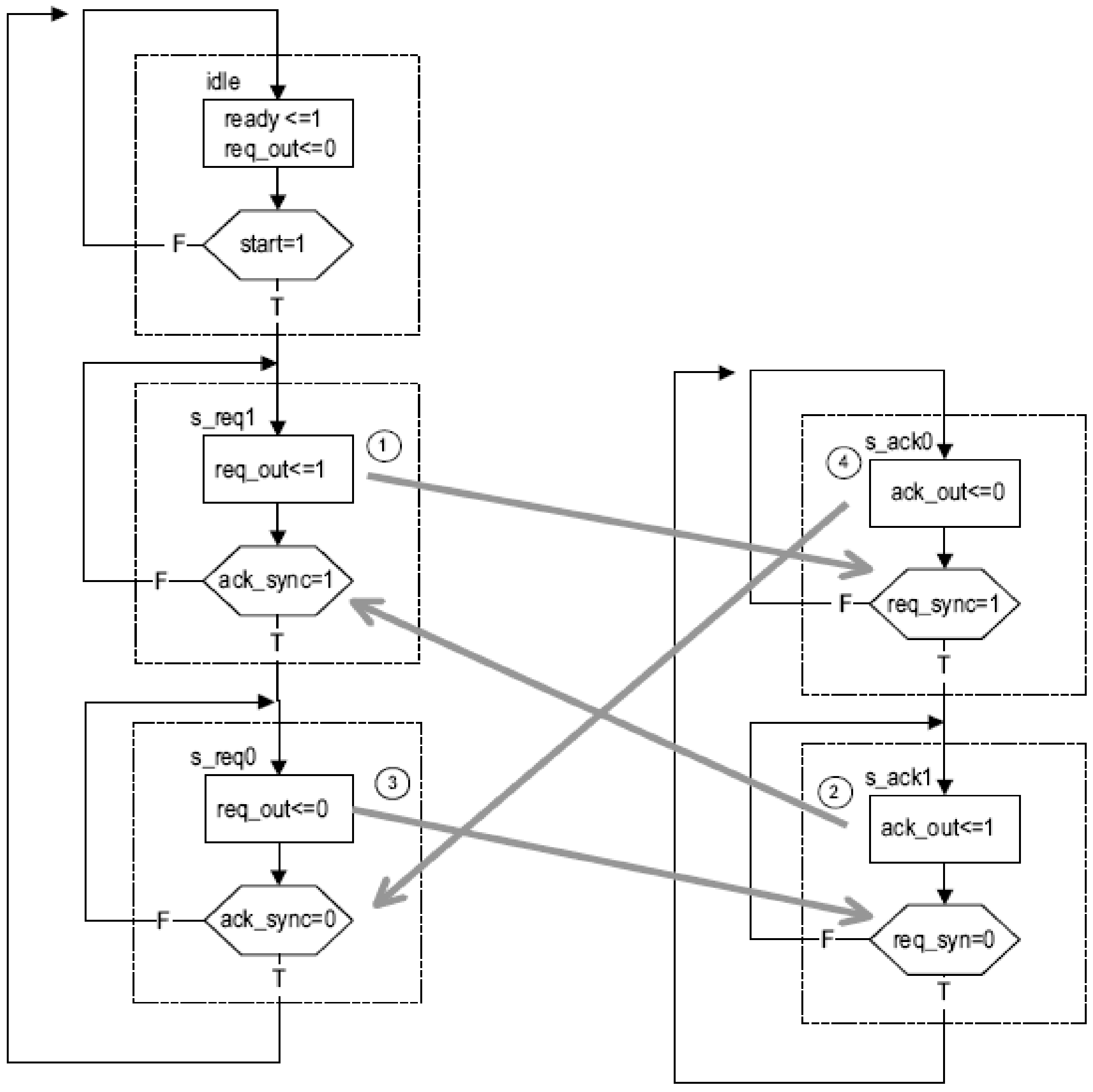


- Four phases:
  - Phase 1: talker activates req
  - Phase 2: listener activates ack
  - Phase 3: talker de-activates req
  - Phase 4: listener de-activates ack
  - Talker can start a new request

- Need synchronizer if talker listener in different clock domains



- Talker FSM and listener FSM

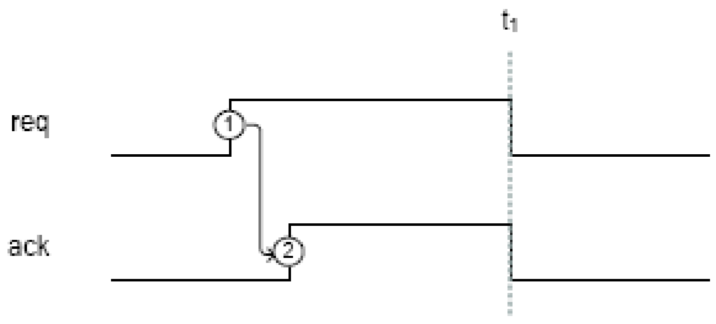


Talker FSM

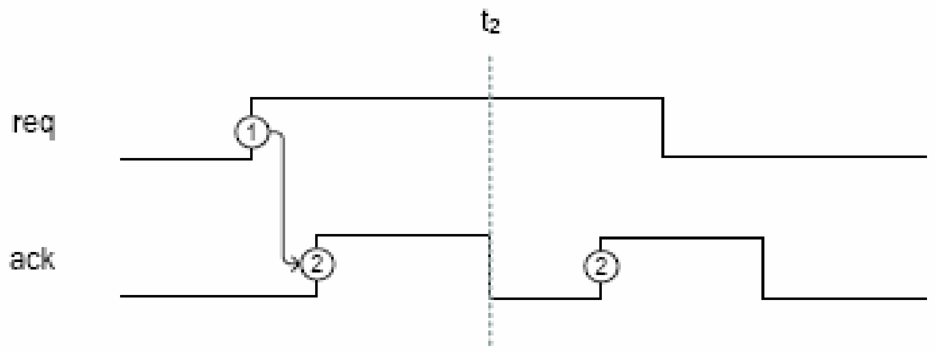
Listener FSM

- Implementation:
  - Talker: FSM and synchronizer for ack_out
  - Listener: FSM and synchronizer for req_out
- Pass an enable tick using handshaking
  - The enable tick functions as the start signal in talker
  - The listener generates a Mealy output which is asserted when req_sync is asserted in the s_ack0 state (i.e., a rising-edge detection circuit for req_sync)

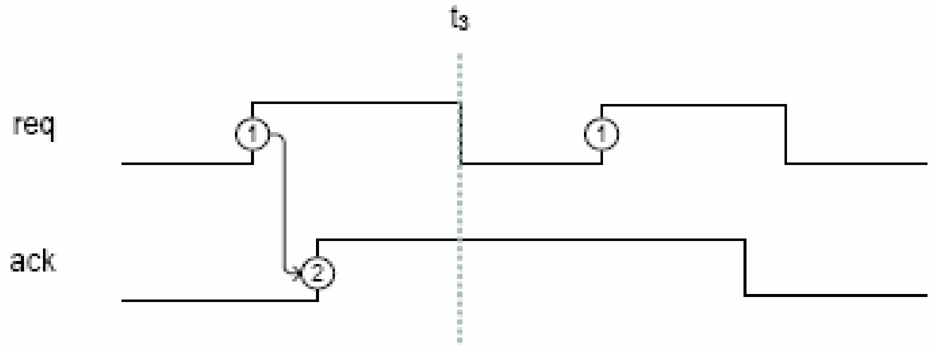
- Can we remove the second part of handshaking?



(a) Ideal scenario

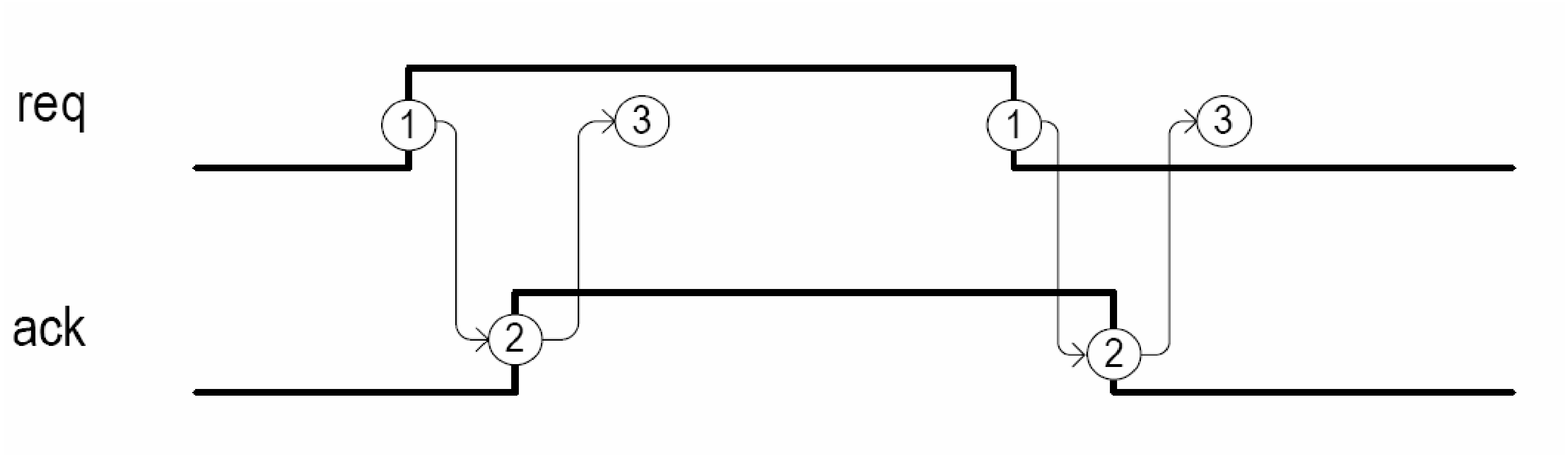


(b) Error due to slow talker response



(c) Error due to slow listener response

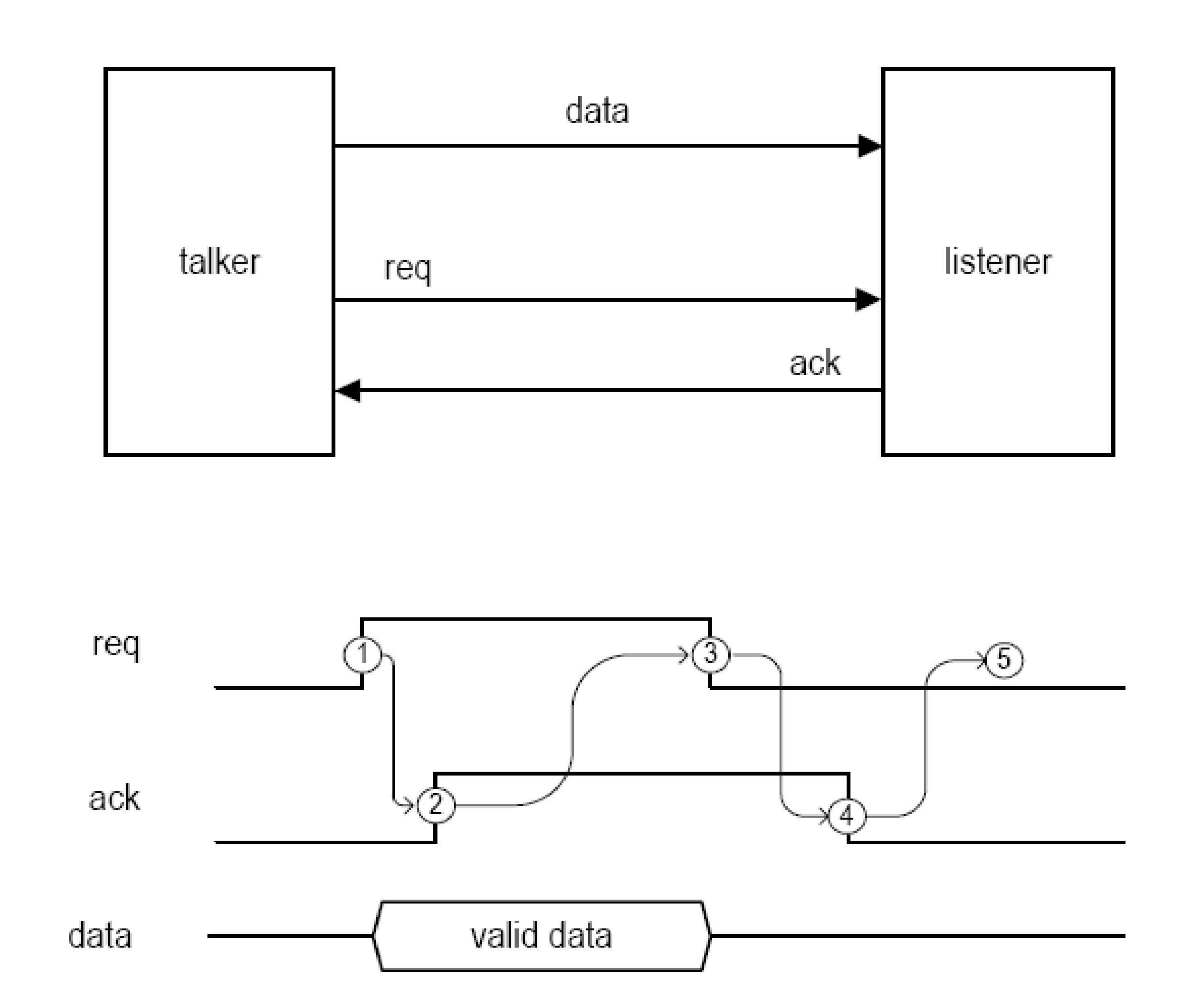
- Two-phase handshaking protocol
  - We can modify the 4-phase protocol so that talker/listener not returning to 0
  - May not be proper for certain applications



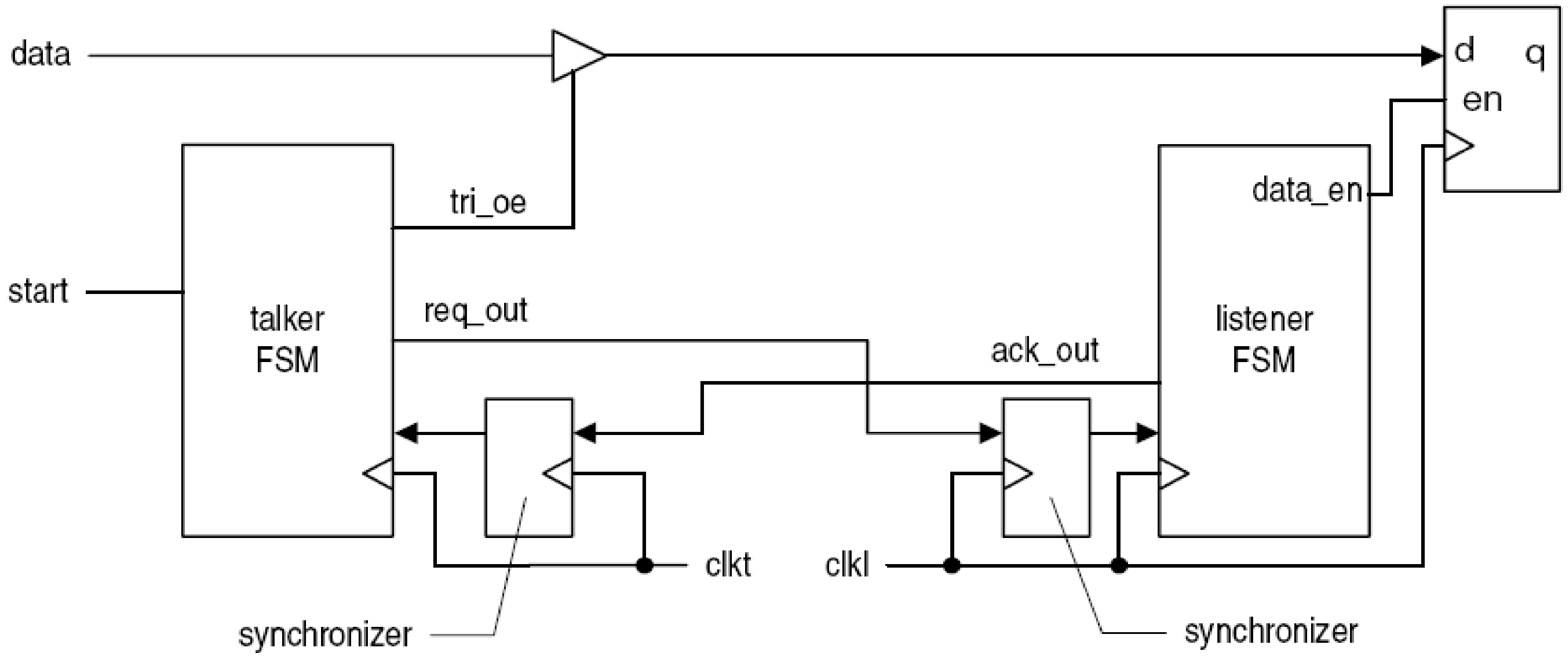
6. Data transfer crossing clock domains

- It is difficult to synchronize a multiple-bit signal (e.g., signal changes from 11 to 00)
- Use req/ack and handshaking protocol to coordinate data transfer
  - Only one signal needs to be synchronized in each domain
  - All other signals are bundled as “data”

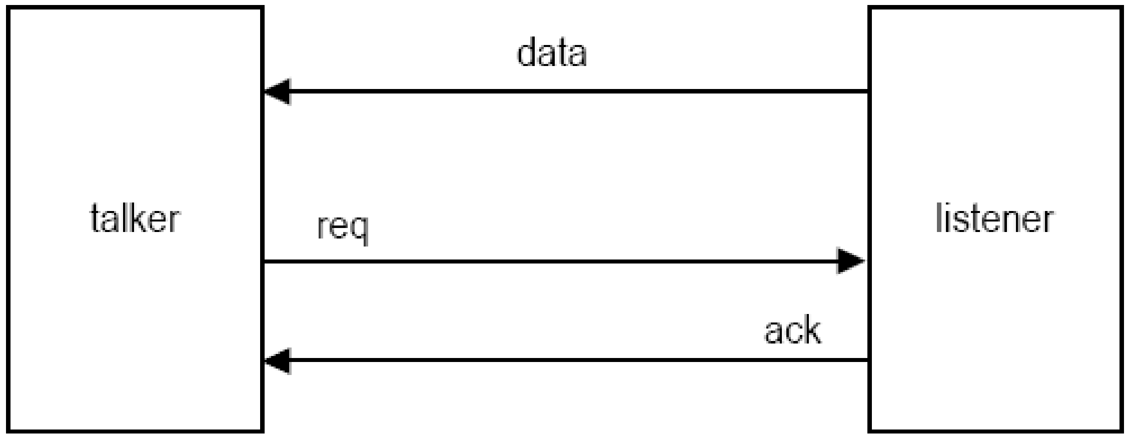
- Push operation (talker sending data)
  - Conceptual diagrams

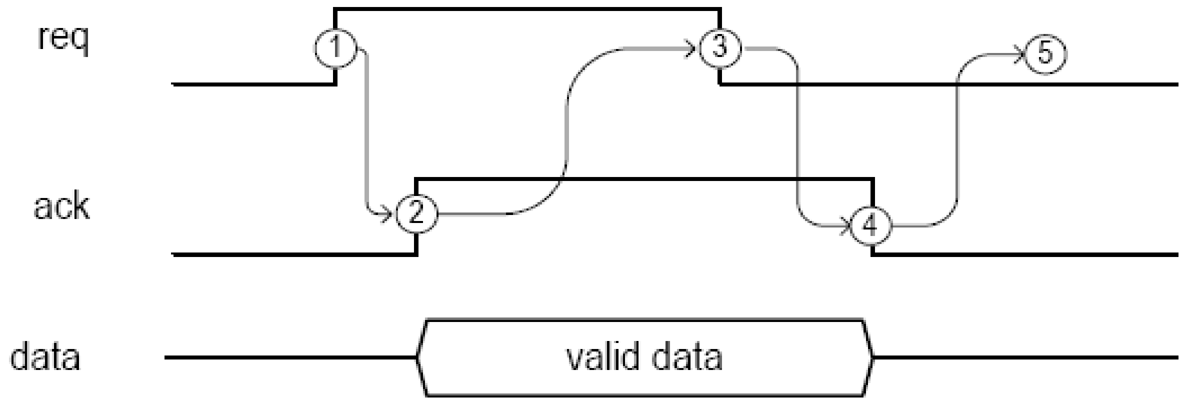


- More detailed diagram
  - Talker activates req_out and tri_en (i.e., placing data on data bus) at the same time.
  - req_out is delayed one or two clocks when synchronized in listener
  - data is stabilized when data_en is asserted (i.e., no timing violation)

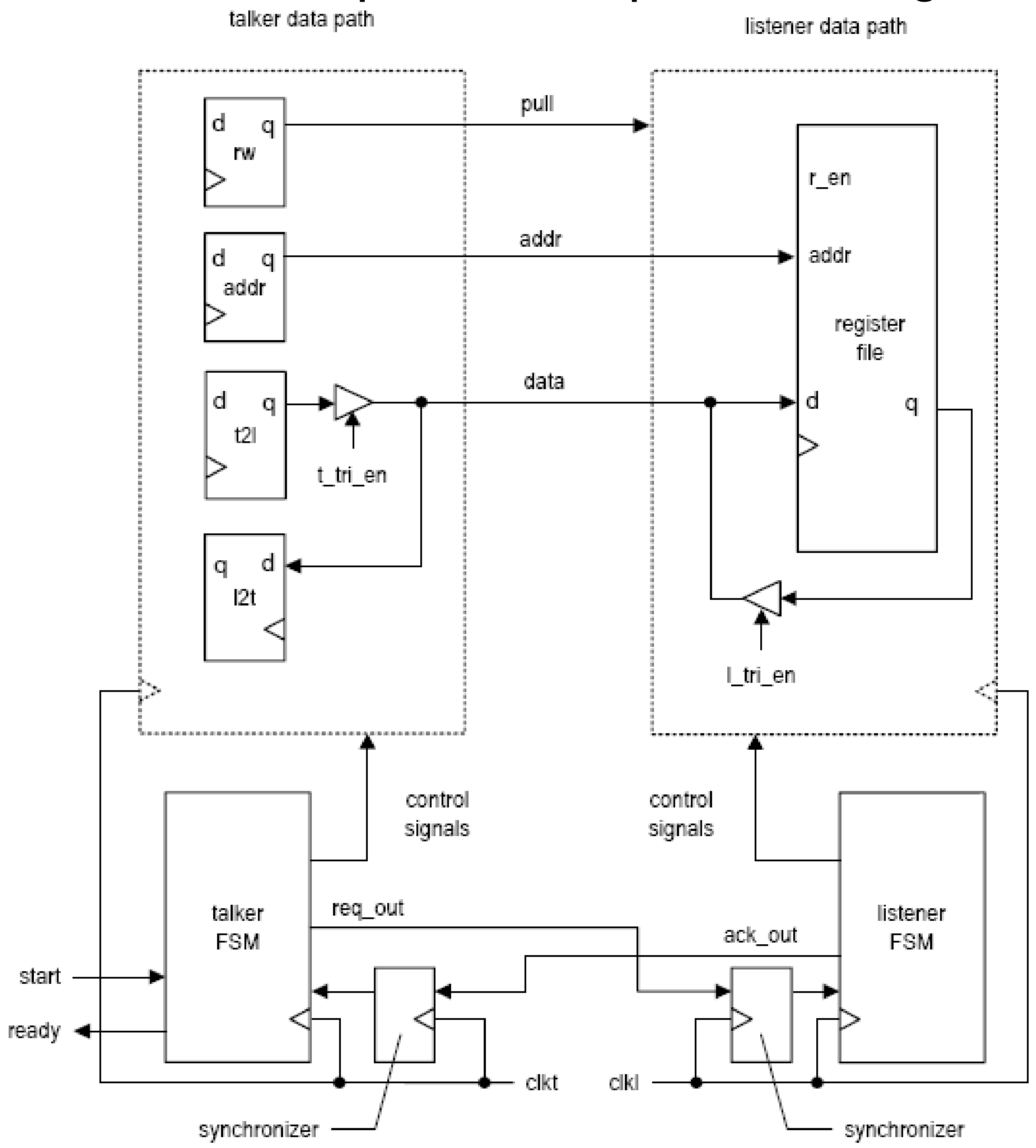


- Pull operation (taller retrieving data)
  - Conceptual diagrams





- Bidirectional operation is possible; e.g.,



- Performance:
  - How many clock cycle for one data transfer?
- Other methods for data transfer
  - FIFO (synchronization needed for empty and full status signal)
  - Shared memory (synchronization needed for arbitration circuit)
  - Dual-port memory (meta-stable condition may occur in the internal arbitration circuit)