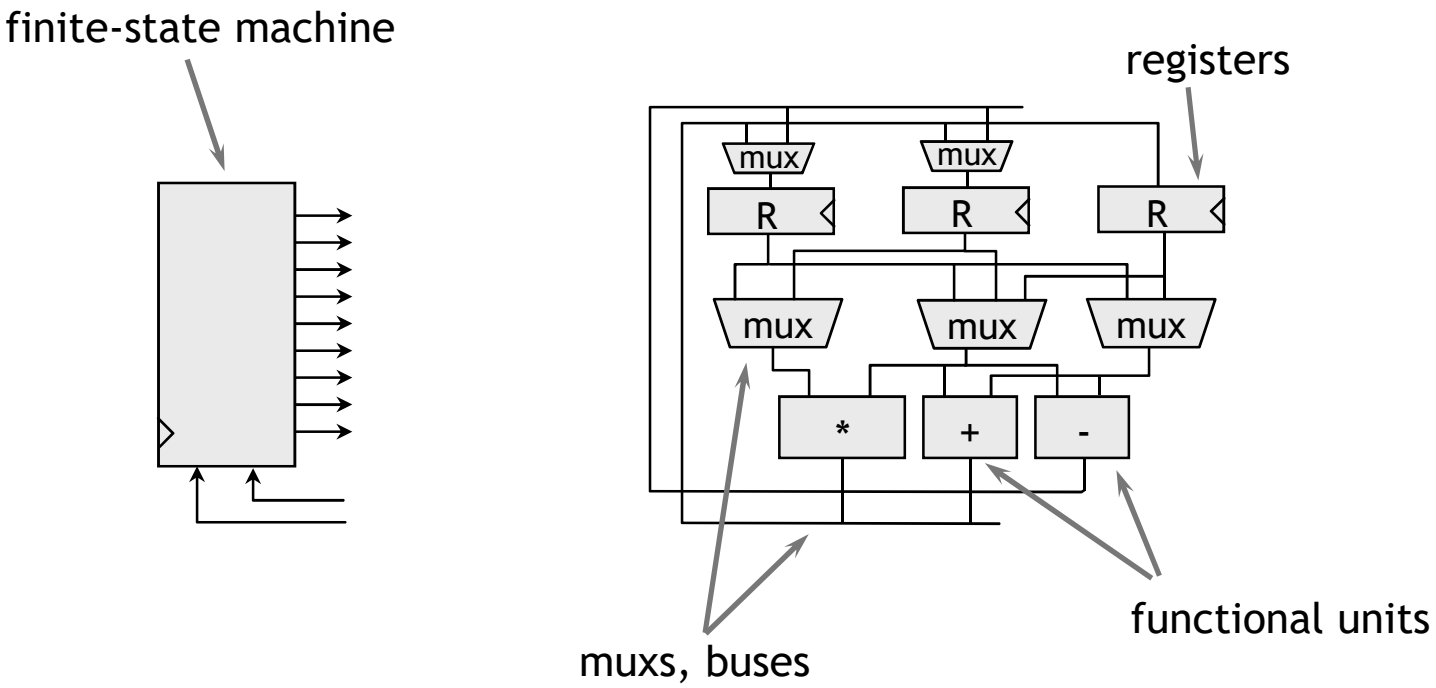
RTL digital system design

- Generic systems do control tasks and data processing
  - Processing unit or ***datapath***
    - Perform the operations among data data
    - Data enters the system and is stored in registers or memories
    - Functional units implement the operations (coded as Verilog *expressions*)
  - Control unit or ***control path***
    - Schedule the operations through the datapath
    - Selects the data to feed the functional units, select operations
    - Determine which registers or memories are loaded in each clock
- *Control dominated systems*
  - Most of the system function is control (e.g. traffic light controller)
- *Data flow dominated*
  - Most of the system is datapath, little or no control (FIR filter)

Register-transfer level



Control unit

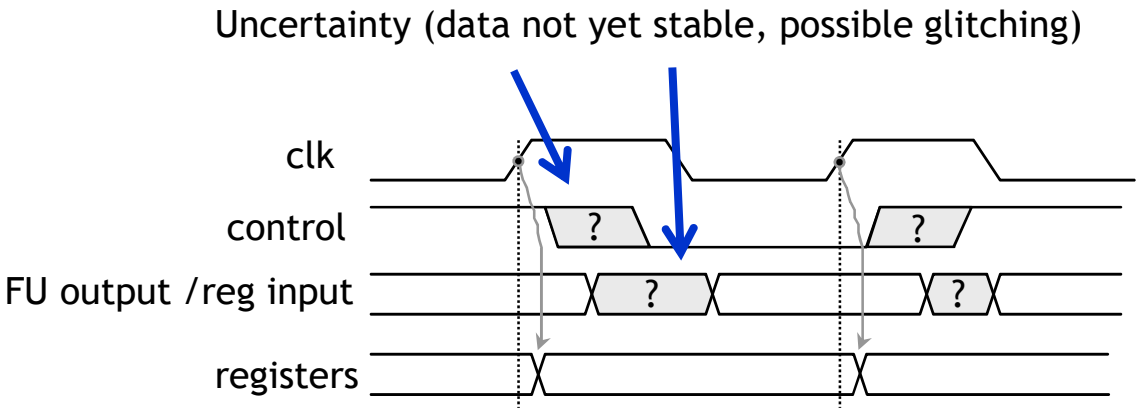
Datapath

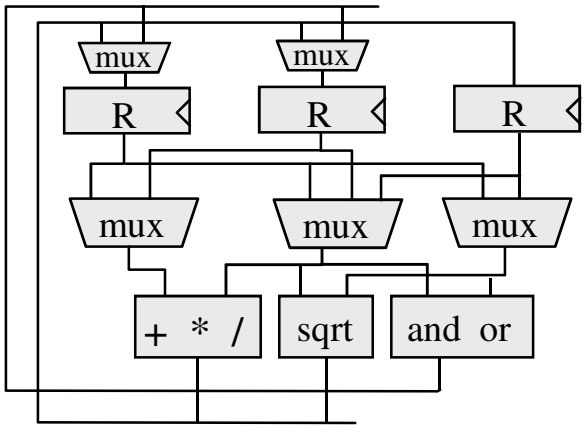
Elements of the datapath

- Registers and memories
  - Hold data, write is synchronous with a clock signal
- Buses, multiplexers, 3-state buffers
  - Interconnect components and steer data along the datapath
- Functional units (FU)
  - Basic model
    - Direct translation of a Verilog expression, fully combinational
    - `assign y1 = ( a + b * { y[31:16], z[15:0] } >>> 6 );`
  - More complex models for FUs (need to be built "by hand"!)
    - Perform the operation in more than a single clock cycle (sequential)
    - Perform operations in *pipeline*
    - Support for different operations sharing logic structures (add and mul)
    - Using dynamic reconfiguration (in FPGAs)

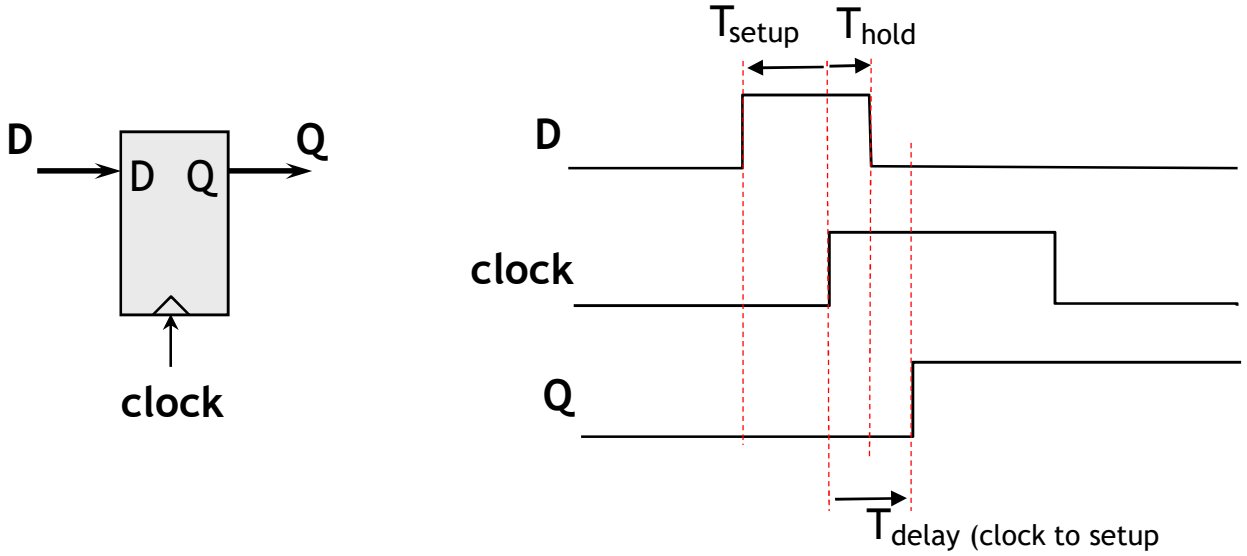
Clocked synchronous systems

- A clock signal triggers <u>all the registers</u>
  - The control path defines the next state of the registers
    - Which registers are loaded and with which data
  - The datapath performs the operations on the data
    - The registers/memories are loaded in the active clock edge
  - Digital circuits need time to propagate logic signals
    - Due to parasitic Rs and Cs and transistor dynamic behaviour



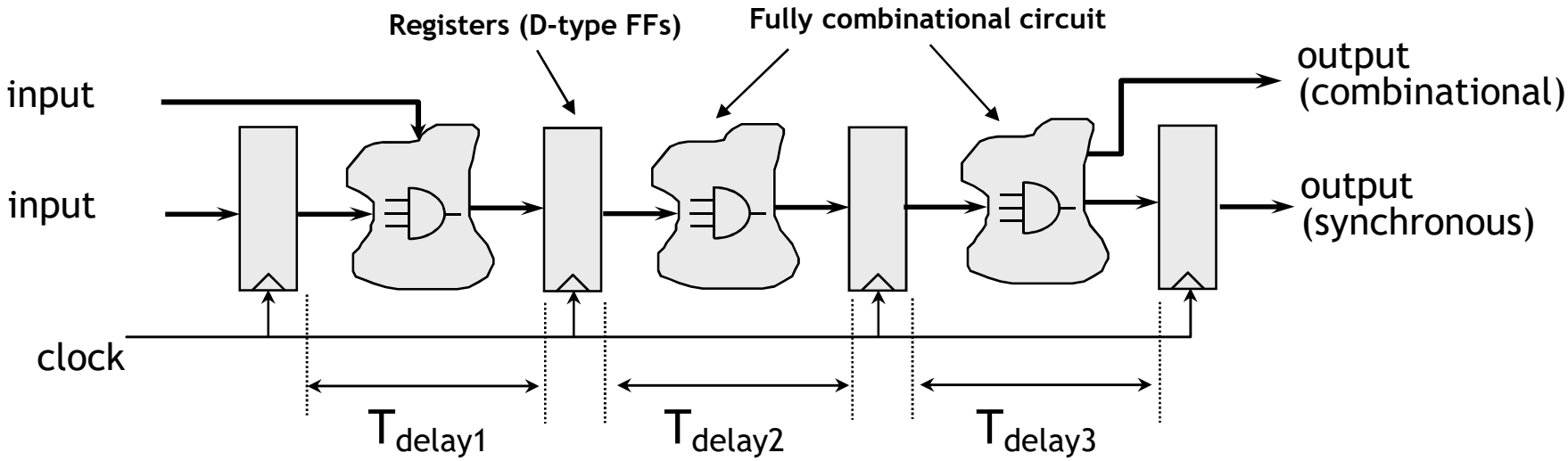


Flip-flop timing



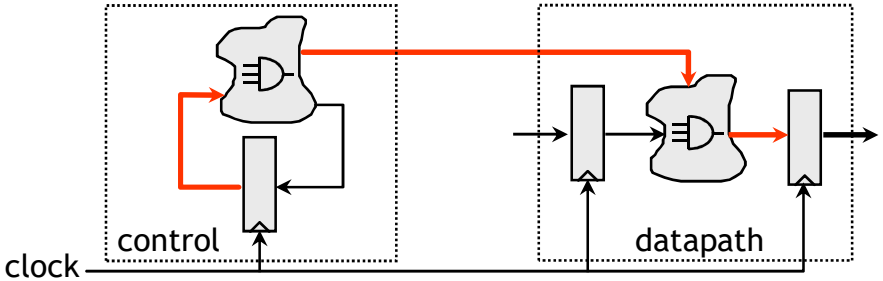
- Flip-flop timing constraints & characteristics
  - D input must be stable before the clock by T_{setup}
  - D input must remain stable for T_{hold} after the clock
  - The clock signal must have a minimum rise/fall time
  - Q output changes T_{delay} after the clock edge (not a constraint)
- What happens of these timing constraints are violated?

Clocked synchronous systems
(combinational logic between registers)



- The circuit delay determines the maximum clock frequency

Maximum clock frequency < 1 / max(T_{delay1}, T_{delay2}, T_{delay3})

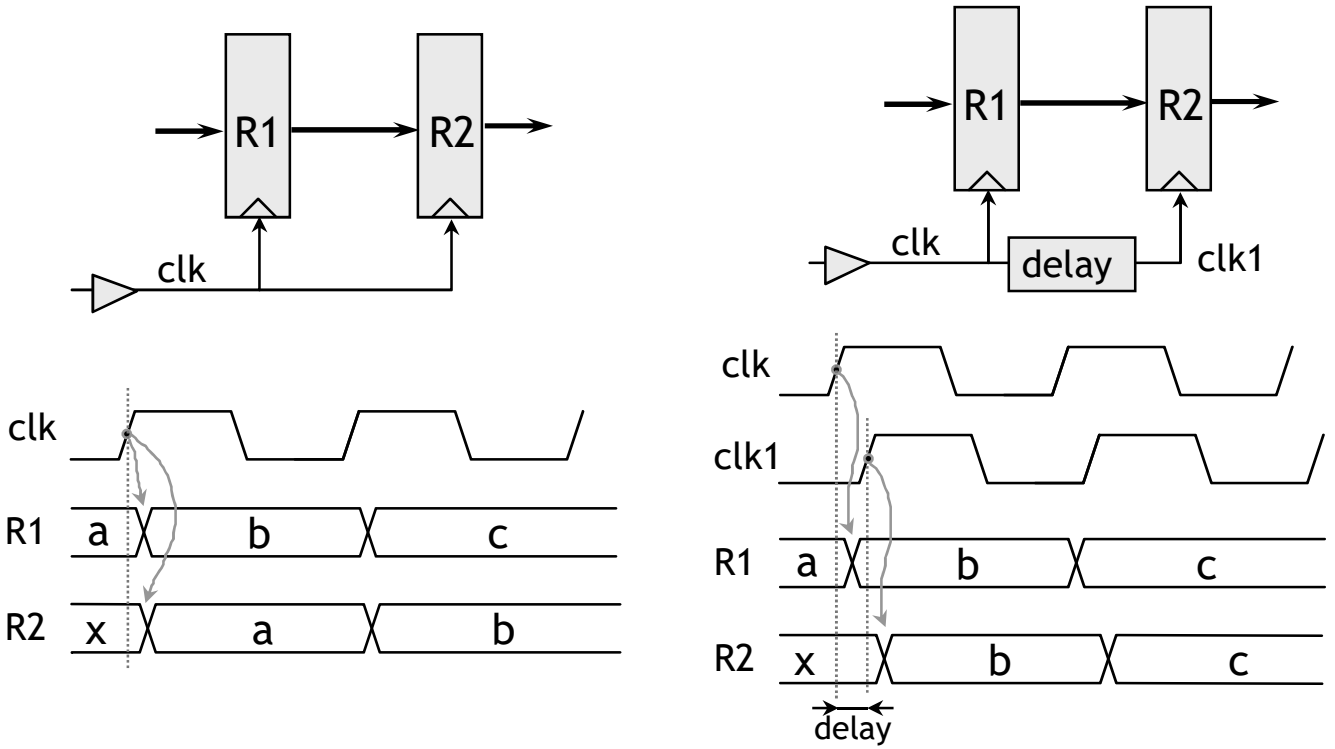


Clocked synchronous systems

- Simple to design, but…
  - All registers have the same clock [logic] signal
    - the clock must be in phase in all flop-flops (nor necessarily…)
    - It is necessary to guarantee T_{setup} and T_{hold} of all flip-flops
    - high fanout net: requires a dedicated network to distribute the clock
  - The clock is a free-running signal continuously switching
    - High power consumption (a significant part of power is due to the transitions)
    - "Clock gating" can switch off the clock but needs careful design
    - The real propagation delays are only known after place & route
  - Handling asynchronous inputs
    - Preventing metastability
  - Synchronizing data transfers among multiple clock domains
    - Need to guarantee correct data transport between different clock domains
    - Need for dedicated blocks to handle the clock domain crossing (*CDC*)

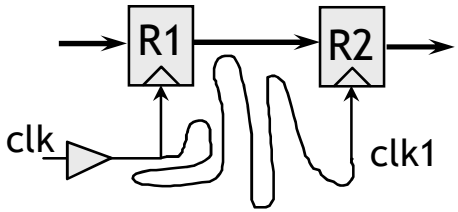
Clock skew

- Variation of the clock phase in different registers

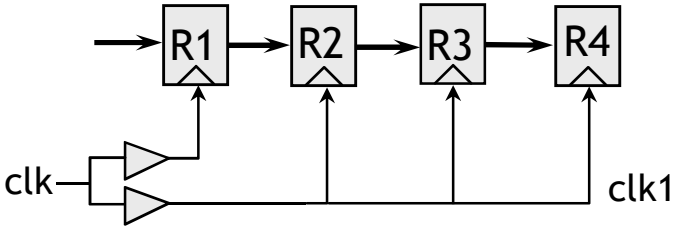


Clock skew

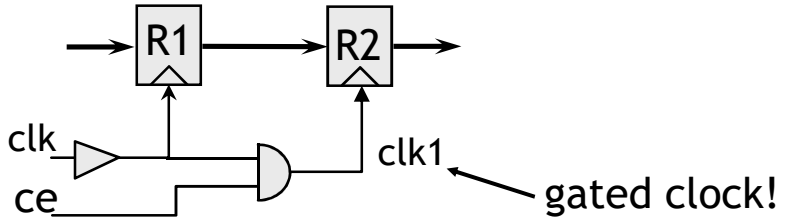
- Un-ballanced clock distribution network
  - Different physical characteristics (length, R, C, L)



  - Different fanout



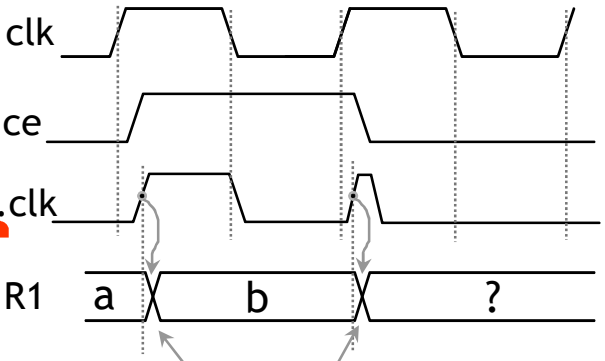
  - Logic in the clock path (for now this is illegal!)



Gated clock

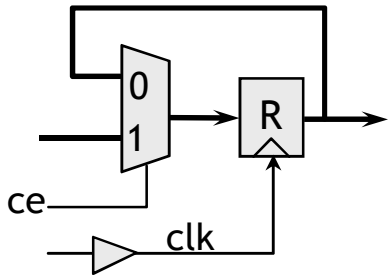
- Conditional register loading
  - Wrong solution: insert and AND in the clock path

```
assign clock_ce = clock & ce;
always @(posedge clock_ce)
...
```



R1 may be loaded twice
Depends on the propagation delays

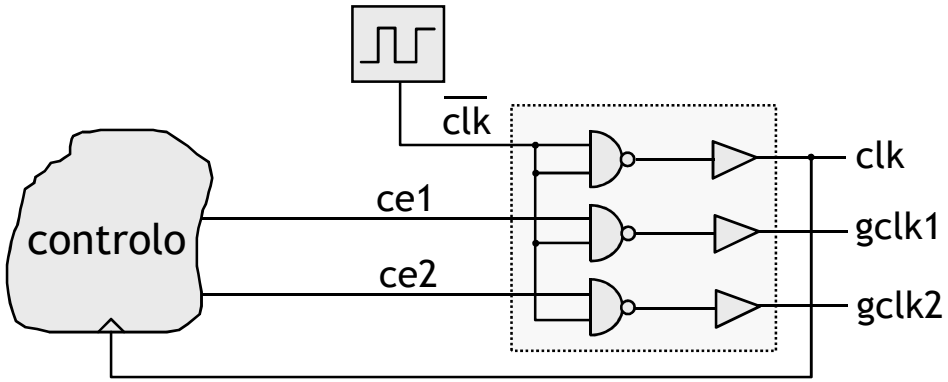
- *synchronous clock enable*

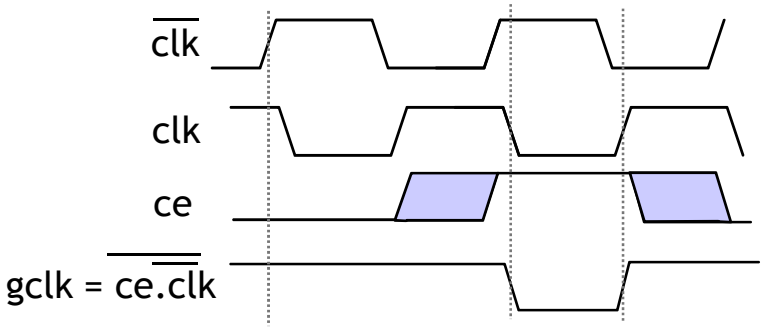


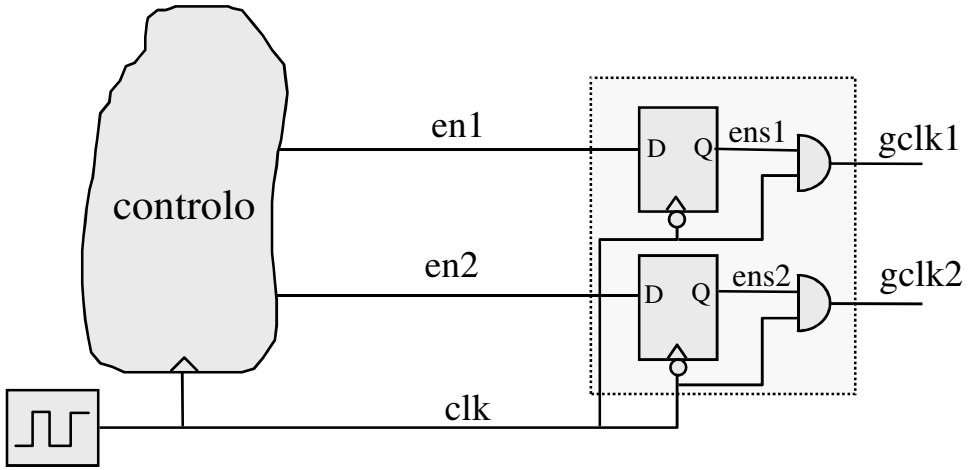
- *flip-flops* with native *clock enable*

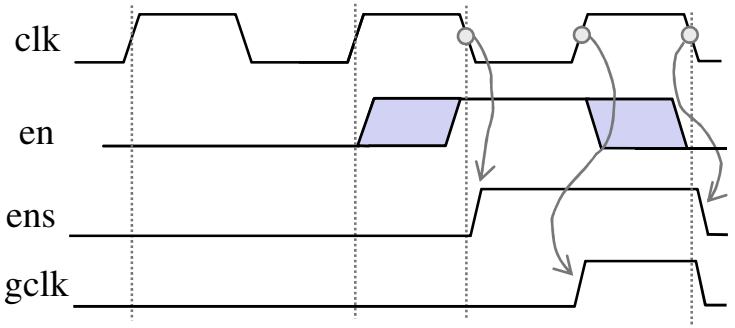
```
always @(posedge clock)
if ( ce )
...
```

Safe gated clocking



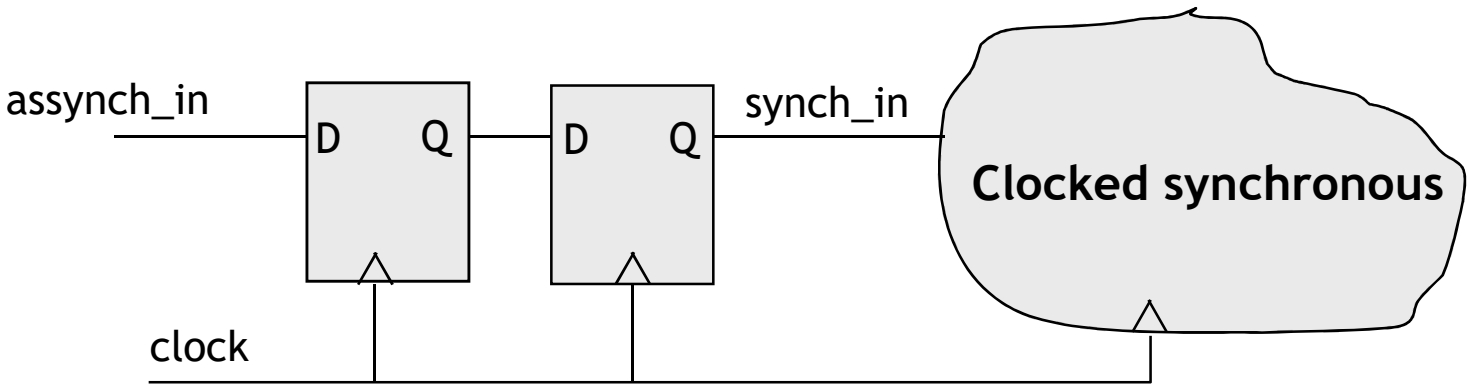






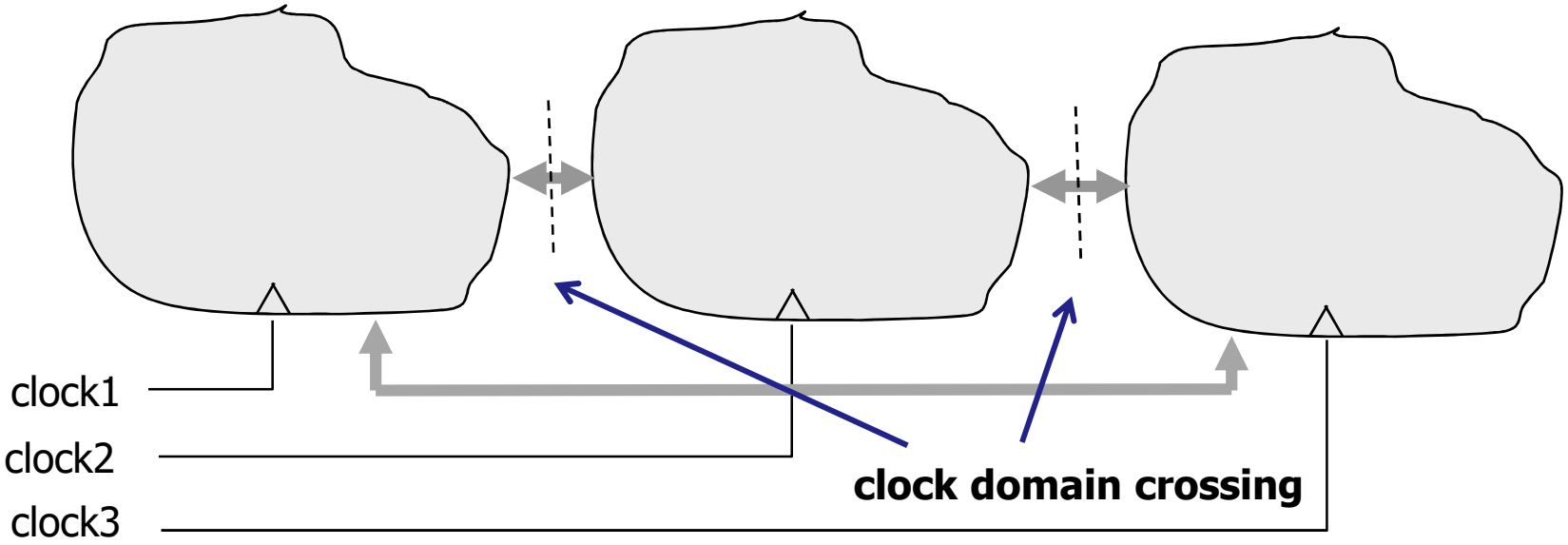
Asynchronous inputs

- What happens if the FF timing is violated?
  - FFs can stay for some time in a metastable state (not representing 1 or 0)
  - FFs are bistable, if left in an unstable state they will fall to 1 or 0
- How to synchronize assynchronous inputs
  - Basic solution: use a two-stage shift-register to sync external inputs



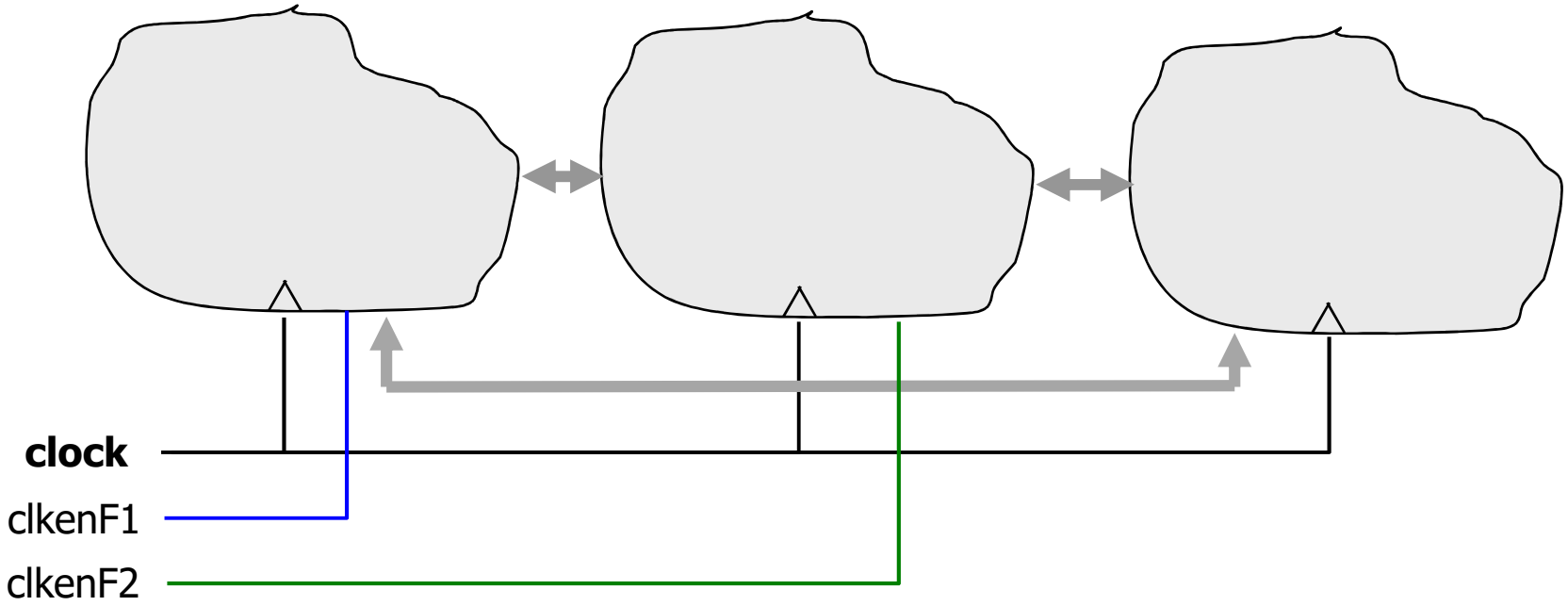
Explain how this works!

Multiple clock domains



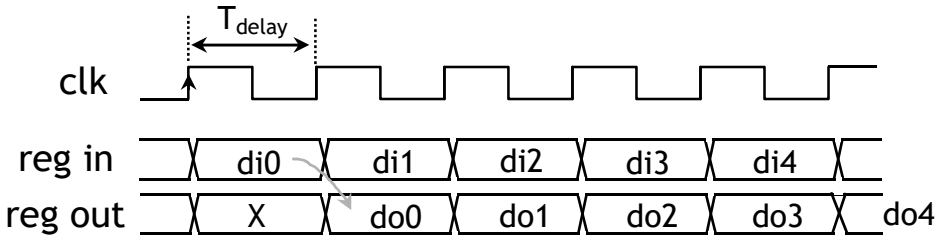
- GALS – *Globally Asynchronous Locally Synchronous*
  - Different clock signals from independent sources (for exemple, diferente oscillators)
- Problems
  - How to generate and distribute the different clock signals ?
  - How to implement a correct clock domain crossing (or CDC) ?

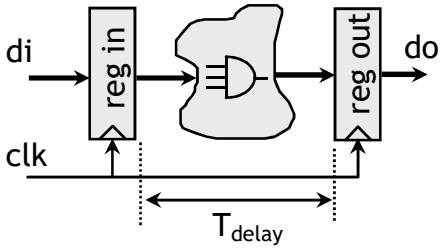
Multiple operation rate

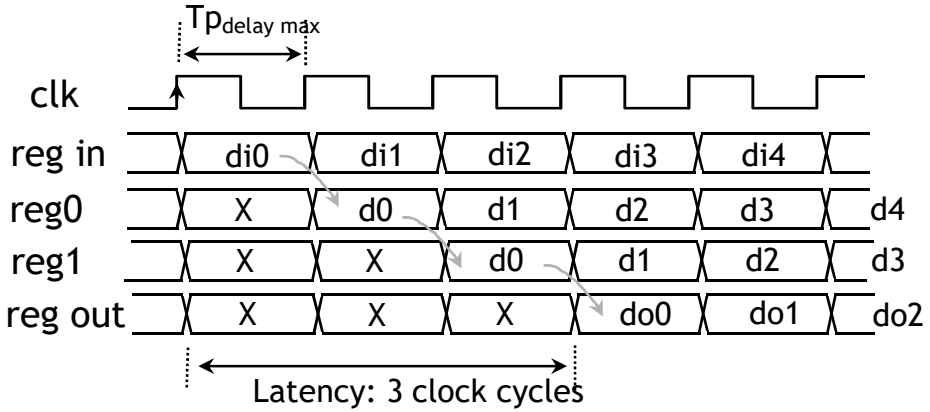


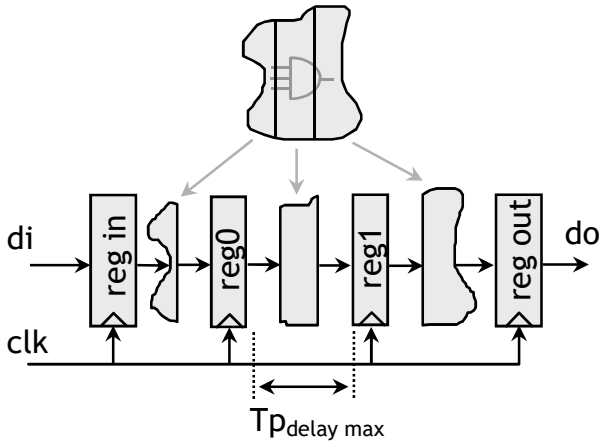
- Use a single clock for all the flip-flops
  - The whole circuit is synchronous with the (single) main clock
  - The clock frequency must be the minimum common multiple of all rates
- Use clock enables derived from the main clock
  - A clock enable pulse must last only for one main clock period
  - Easily generated with counters and comparators

Pipelining


T delay
clk
reg in
reg out
di0
di1
di2
di3
di4
X
do0
do1
do2
do3
do4


di
reg in
reg out
do
clk
T delay


Tp delay max
clk
reg in
reg0
reg1
reg out
di0
di1
di2
di3
di4
X
d0
d1
d2
d3
d4
do0
do1
do2
Latency: 3 clock cycles


di
reg in
reg0
reg1
reg out
do
clk
Tp delay max