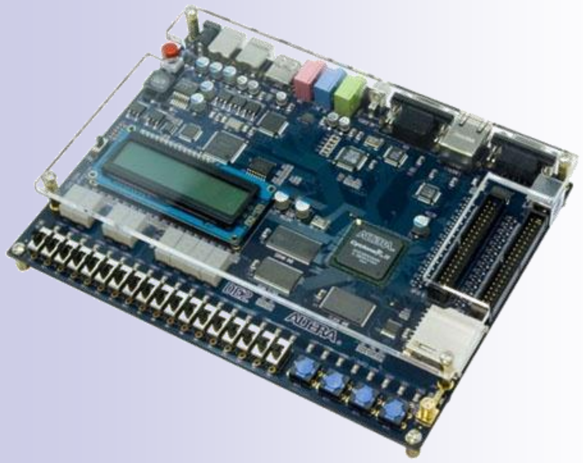
Logic Systems and Processors
cz:Logické systémy a procesory

Lecturer: Richard Šusta

richard@susta.cz, susta@fel.cvut.cz,
+420 2 2435 **7359**



Version V1.1 - redrawn slide 20

CTU-FEE in Prague, CR – subject BE5B35LSP

Metastability

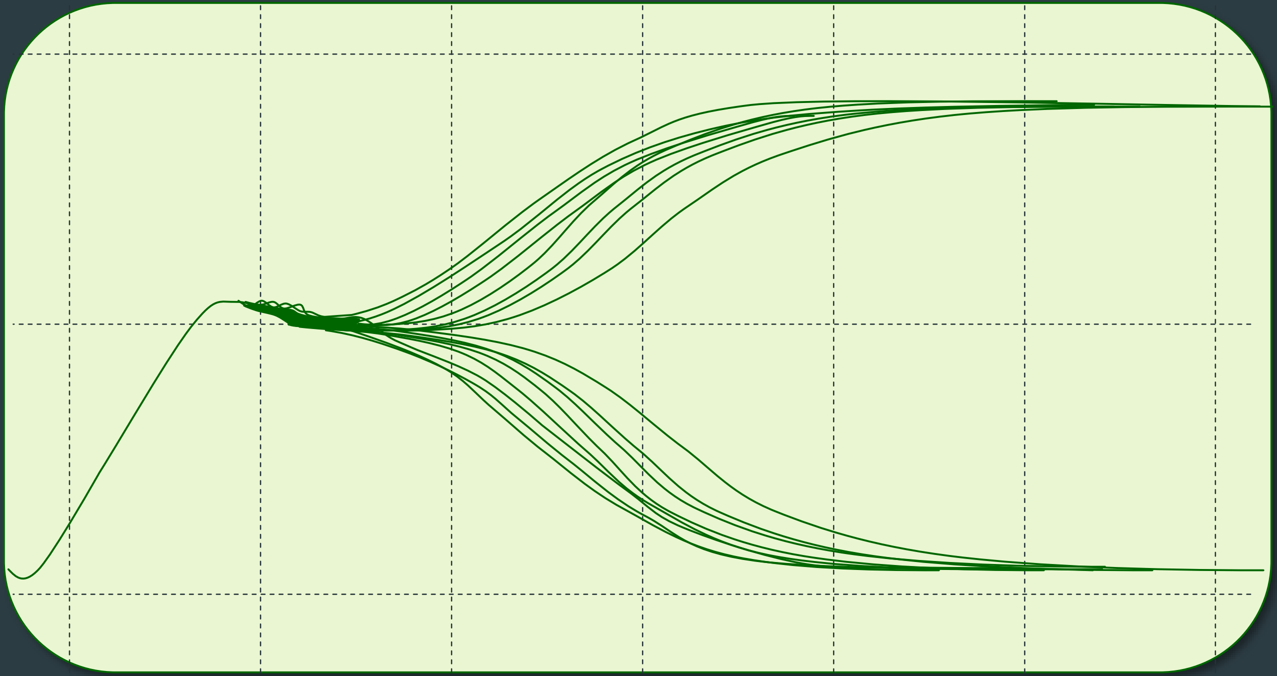
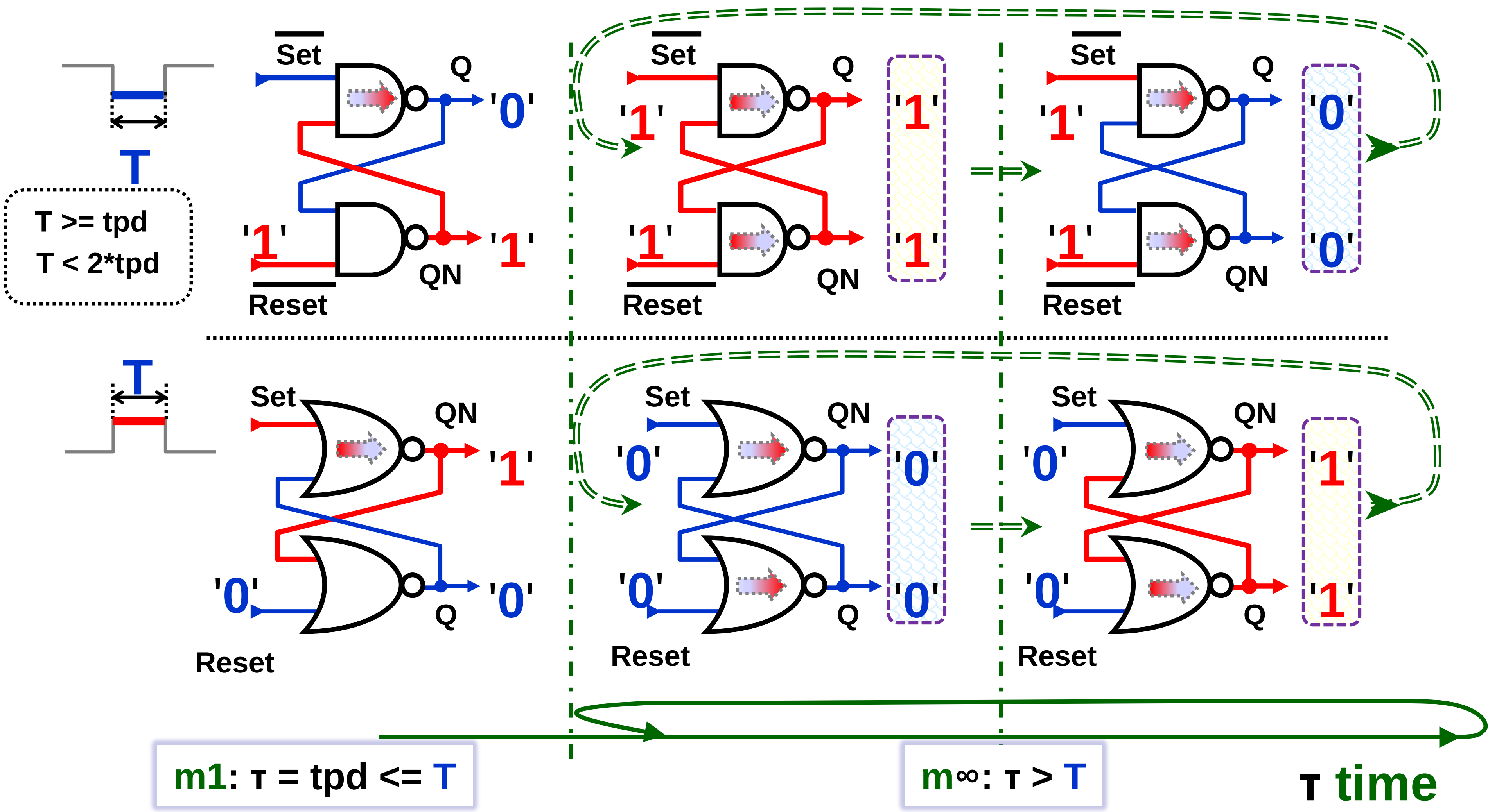


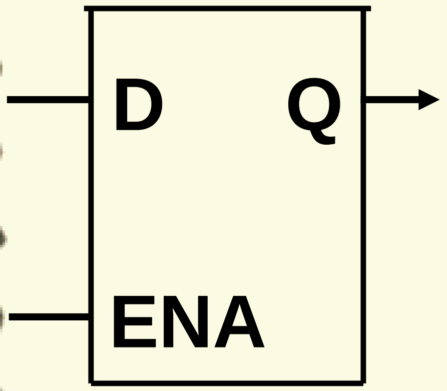
Image: https://simple.wikipedia.org/wiki/Latch

Short pulse-induced metastability

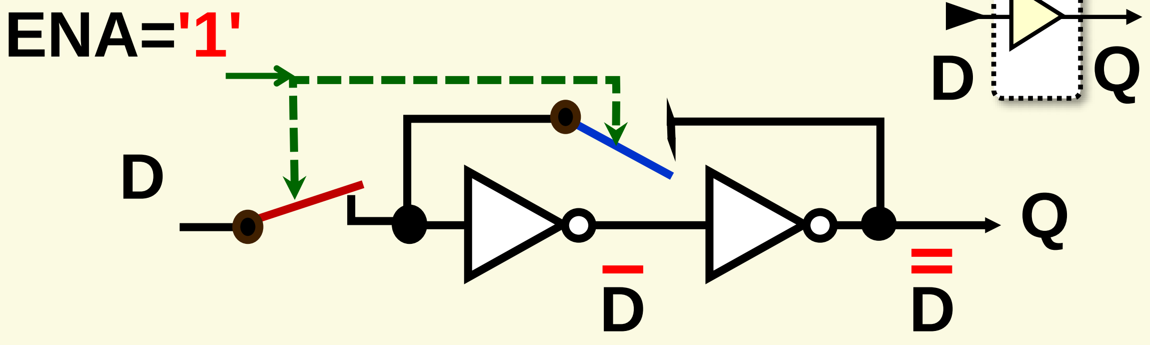


The output of the combination circuit must not be connected to the clock inputs!!!

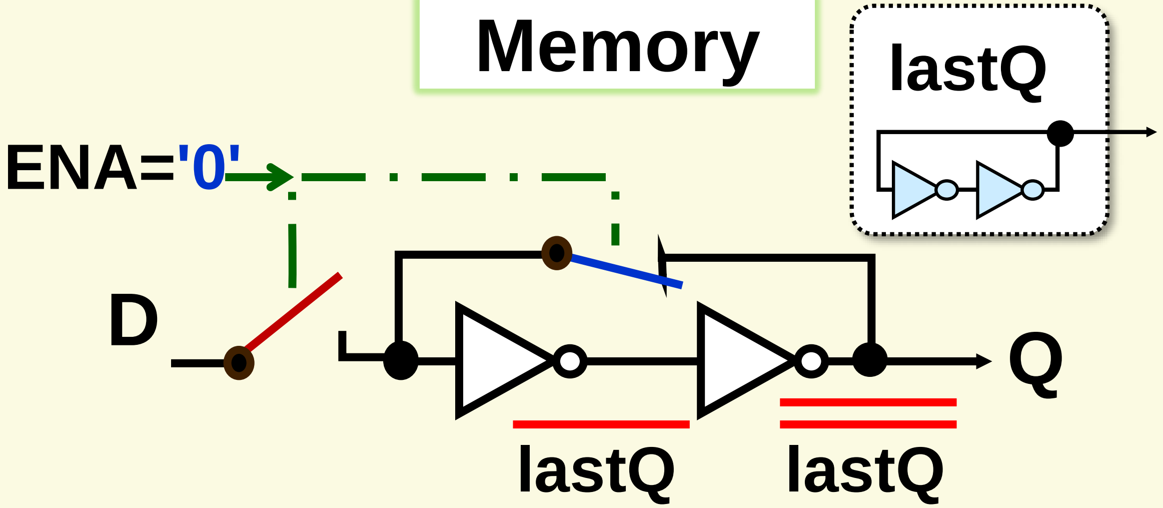
Stable Loop in D-Latch



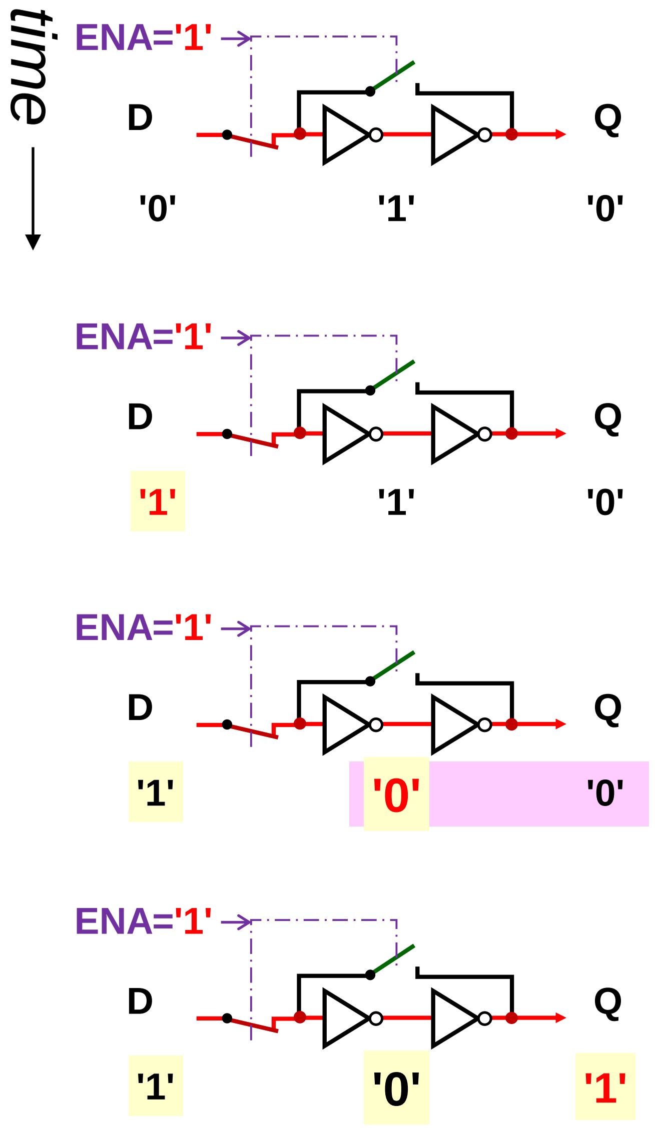
Transparent

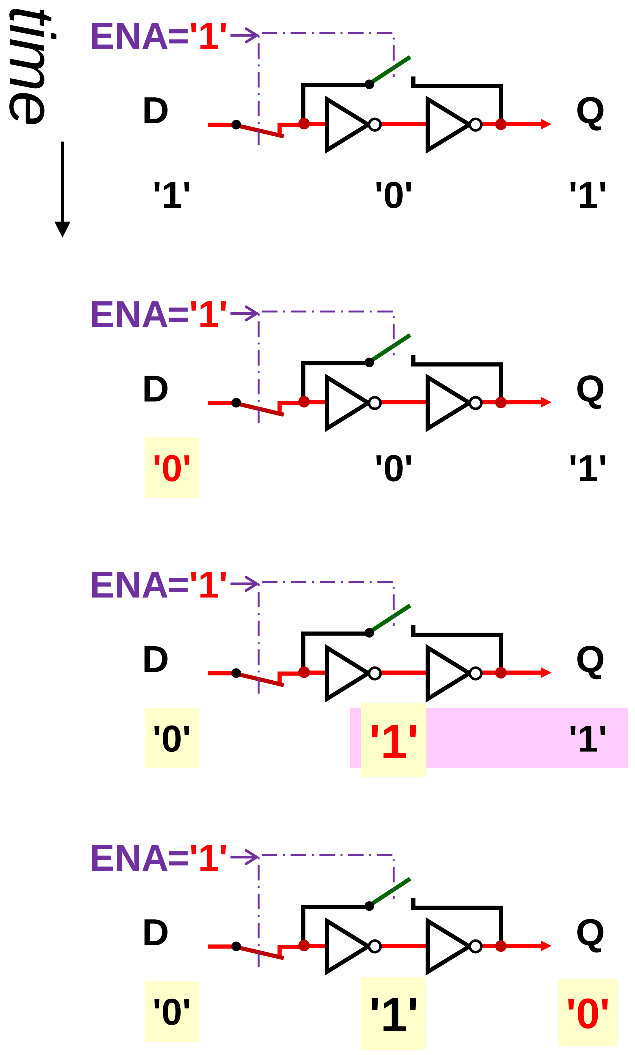


Memory

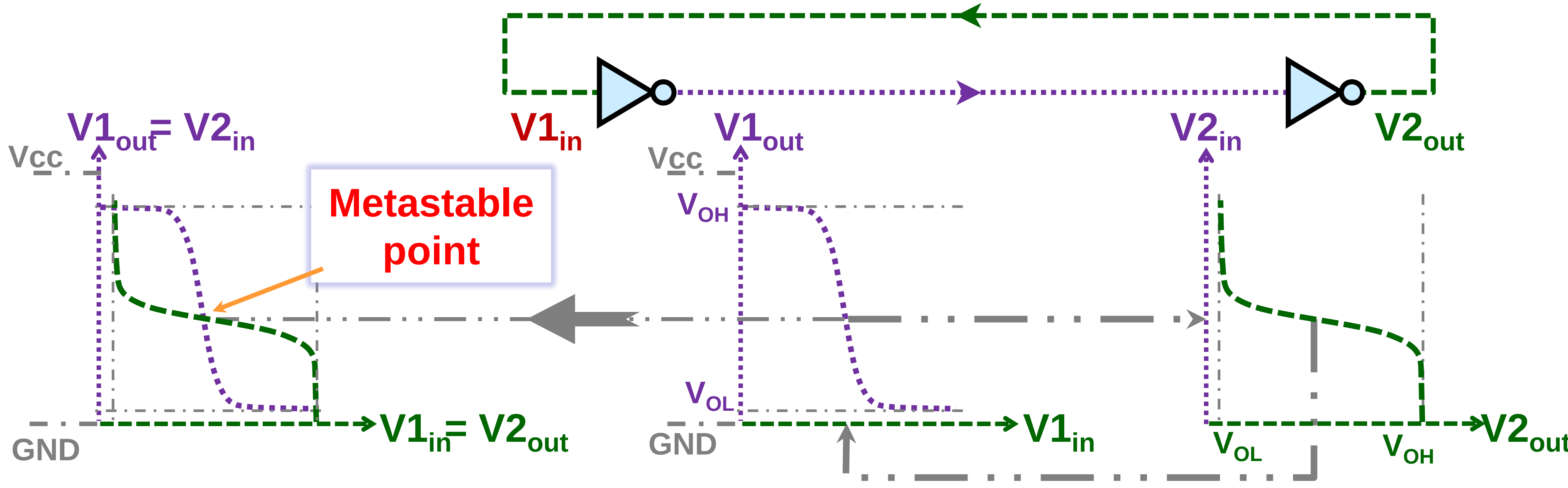

ENA='0'
D
Q
$\overline{\text{lastQ}}$
$\overline{\overline{\text{lastQ}}}$
lastQ

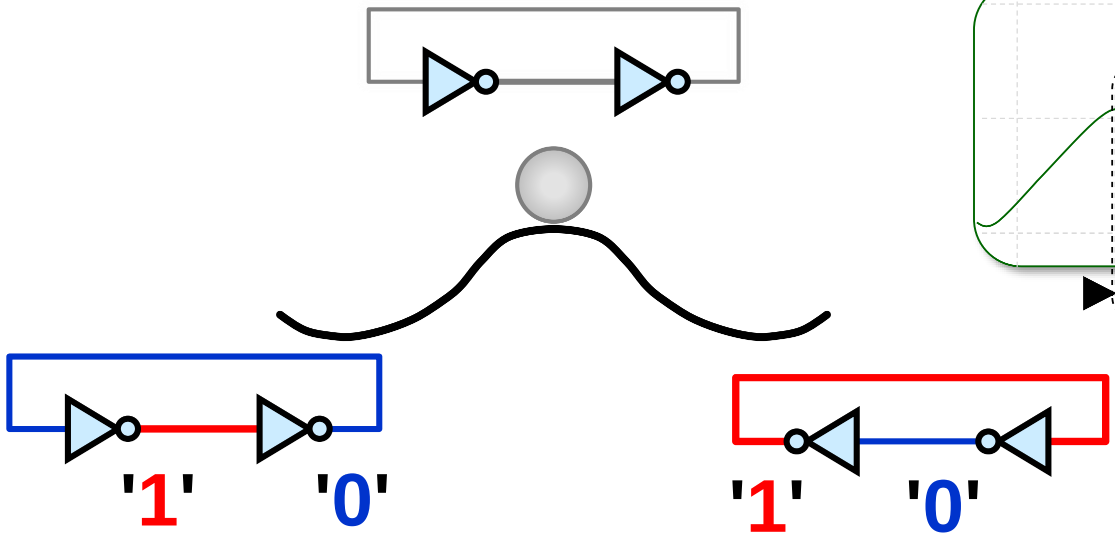
The risk of metastability has not been eliminated!

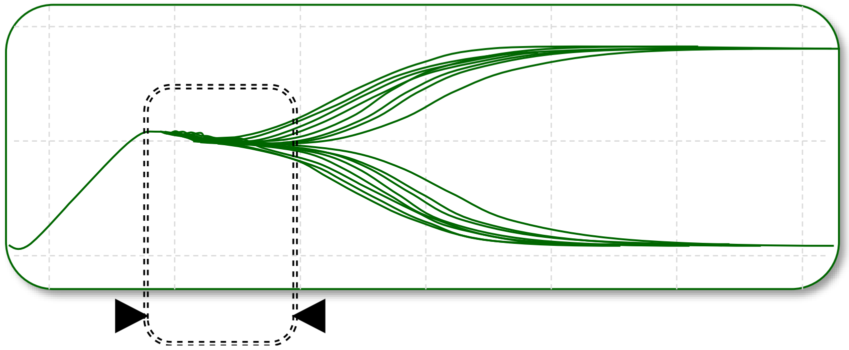




Metastability



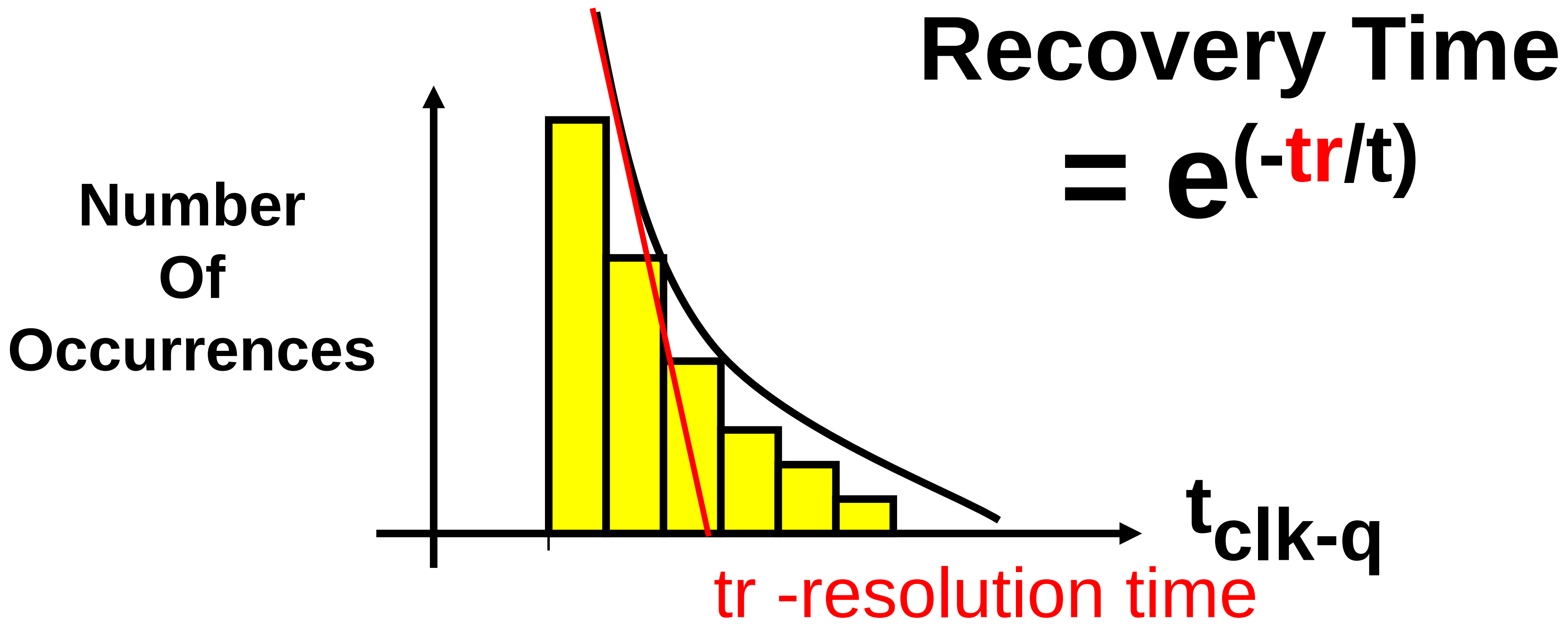




≈ Average Resolution Time

Recovery Time

- a random variable that depends on the chip technology and has an exponential distribution function;
- its parameter decision time t_r (resolution time) has typical values of about 2 ns for 250 nm technology and below 1 ns for < 60 nm technology.



Mean Time Between Failures

Manufacturers usually only specify

MTBF = Mean Time Between Failures

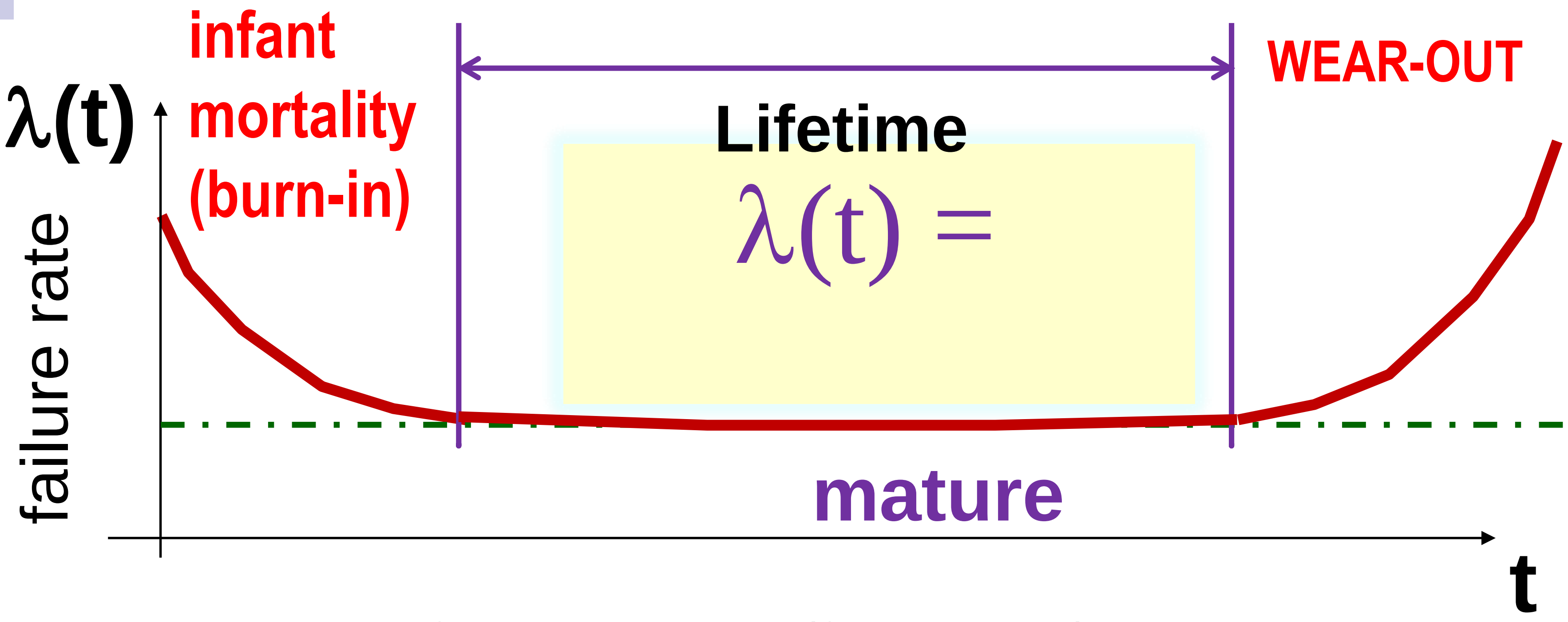
(mean time between failures)

which gives considerable value so it looks better, but is just an illusion for terrible designs :-(

The MTFB calculation is more complex and depends on the operating frequencies.

See e.g. Rogina, Branka & Škoda, Peter & Skala, Karolj & Michieli, Ivan. (2010). Metastability testing in FPGA circuit design using propagation time characterization. Radioelectronics & Informatics Journal. 51. 4 - 8. 10.1109/EWDTS.2010.5742050.

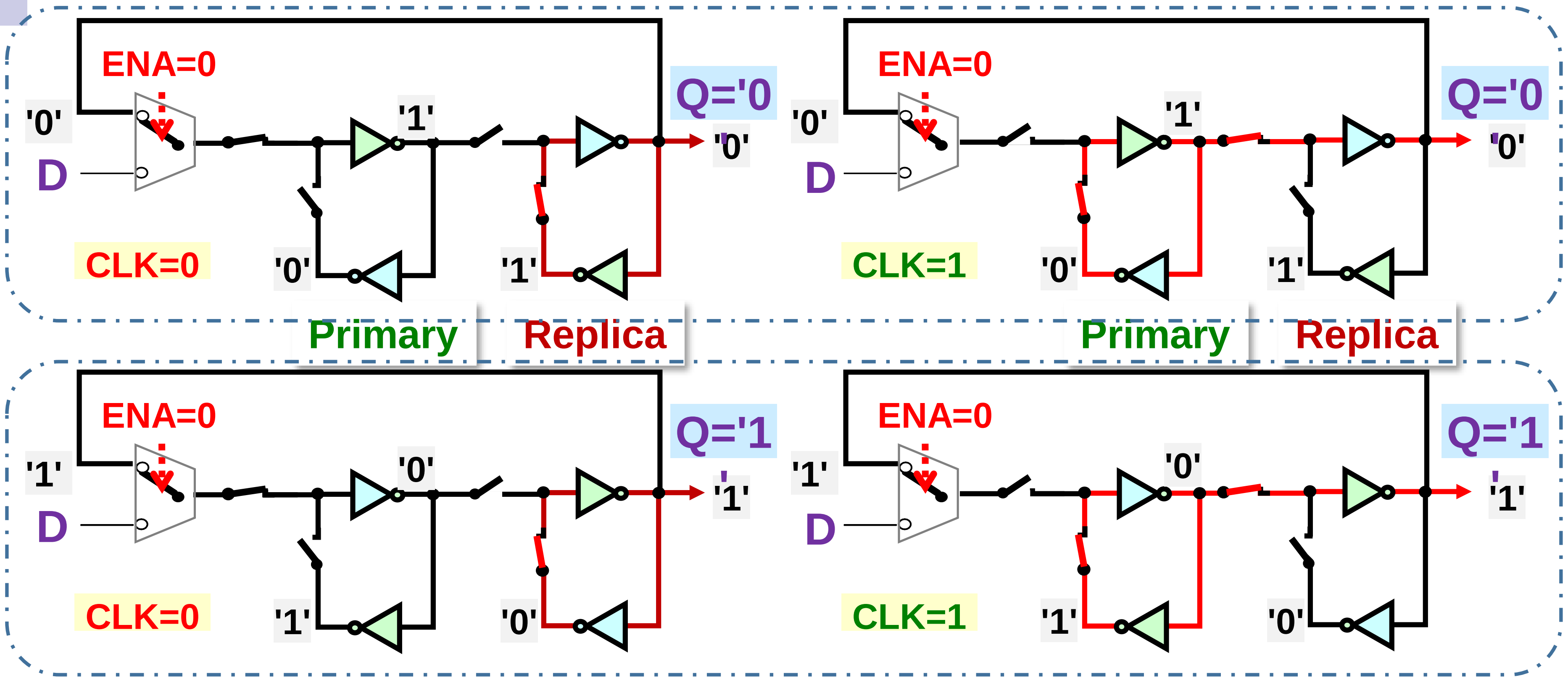
The Bathtub Curve

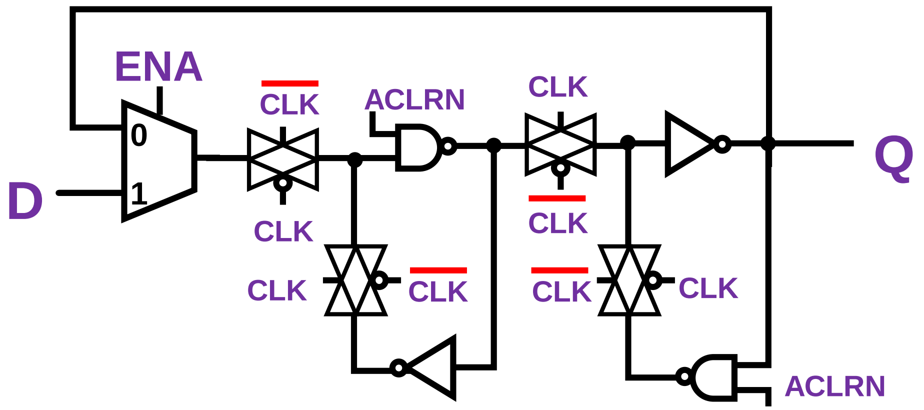


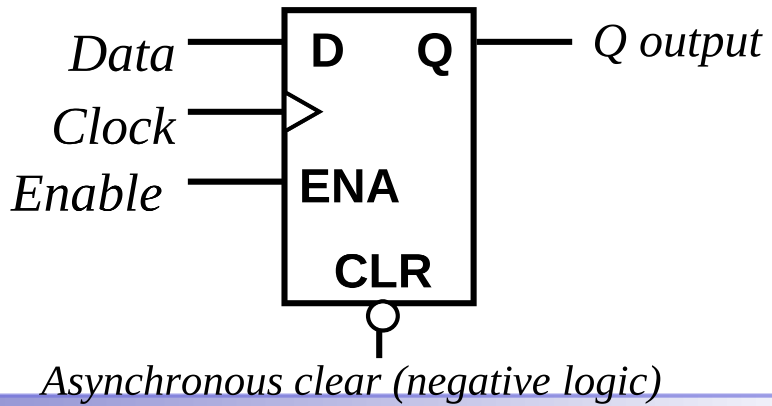
The assumption of constant **λ(t)** (failure rate) in the middle part is based on experience and simplifies the calculations.

Note: In complex systems where high reliability is required, components are first burned in to eliminate the initial part of the curve. Then, their frequent checks prevent the final increase, so their failure rate does not follow the bathtub curve. For a closer look, see, e.g., https://en.wikipedia.org/wiki/Physics_of_failure

DFF or DFFE contains 2 loops

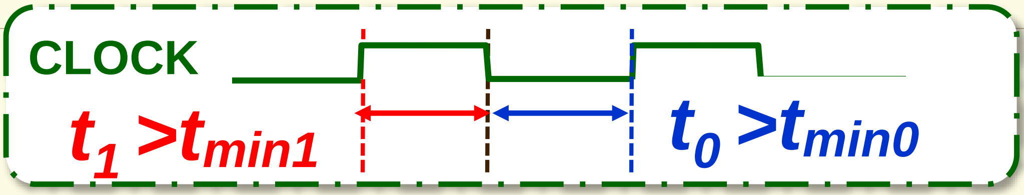




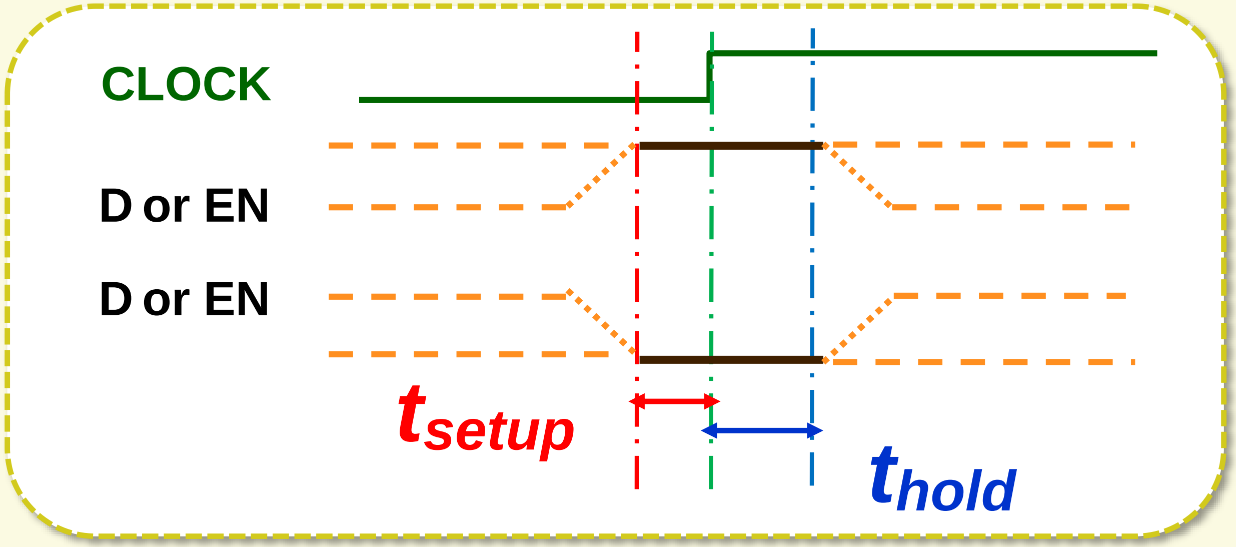


Asynchronous clear (negative logic) after power-up

To prevent metastability, we must meet the circuit operating conditions specified in the manufacturer's catalogs



- Clocks must be stable for minimum times.
- DFF and DFFE must have stable their D data and EN enable inputs around the rising edges of their clocks.



Some Hardware Abbreviations

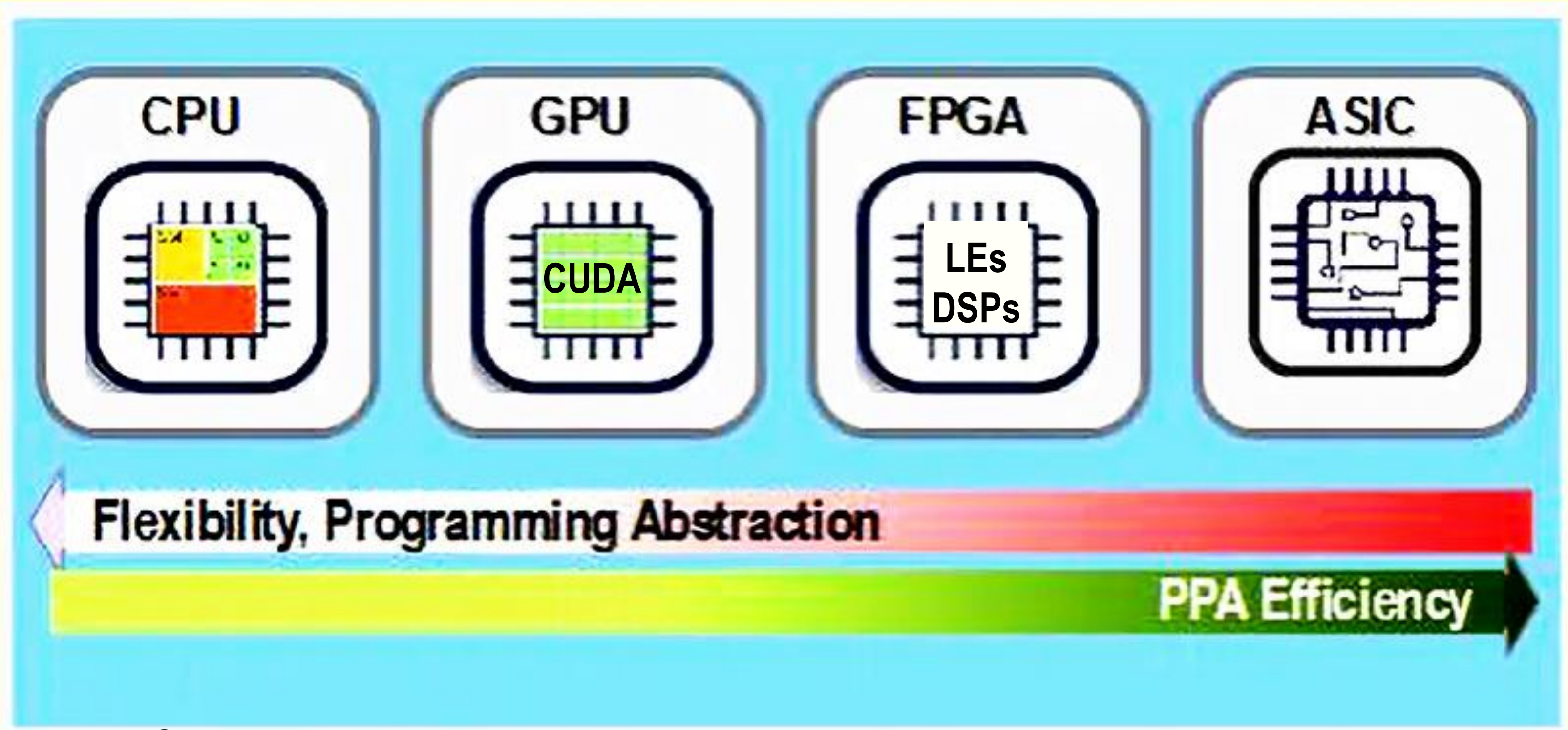


Image: pcbaa.com

- **CPU** - Central Processing Unit aka Processor
- **GPU** - Graphics Processing Unit
  - **CUDA** - Compute Unified Device Architecture
- **FPGA** - Field Programmable Gate Arrays
- **ASIC** - Application-Specific Integrated Circuit
- **PPA** - Power, Performance, and Area

DSP - Digital signal processor

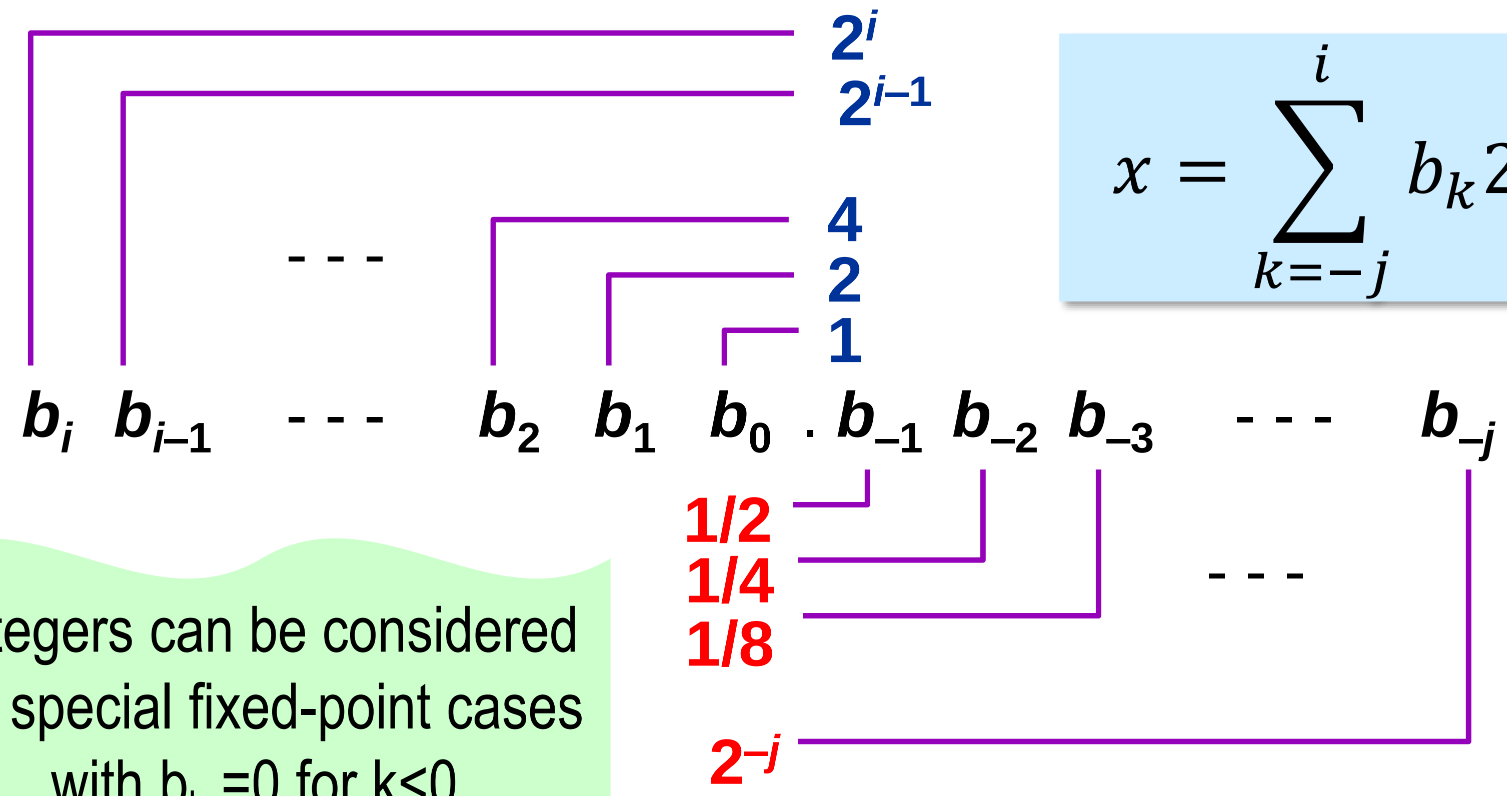
Float Points in Circuits

- Float point operations consist of fixed-point binary and an exponent.
- If we do not have float-point hardware support in DSP blocks of our FPGA, we should prefer faster fixed-point arithmetic.
- For float point, we can insert soft core float point support, e.g., https://github.com/hVHDL/hVHDL_floating_point

Fixed-points represent rational numbers

Rational numbers can be expressed as **quotients** or **fractions** = $\dfrac{\textbf{n} \text{ (numerator)}}{\textbf{q} \text{ (denominator)}}$ *where n and q are integers*

Binary fixed-point numbers:



2^i
2^{i-1}
4
2
1
- - -
b_i b_{i-1} - - - b_2 b_1 b_0 . b_{-1} b_{-2} b_{-3} - - - b_{-j}
1/2
1/4
1/8
- - -
2^{-j}

$$x = \sum_{k=-j}^{i} b_k 2^k$$

Integers can be considered as special fixed-point cases with $b_k = 0$ for $k<0$.

Convert to Fixed-point

- 9/8
- 29/4
- 75.75

Solutions

- 9/8 = 1.125 = 1.001 = 1 + 1/8
- 29/4 = 7.25 = 111.01 = 7 +1/4
- 75.75 we multiply by 2
  = 151.5/2 = 303/4 = bin(303)/4
  - we convert faster 304 because it equals
    = 256+32+16 = 100110000
  - 303=304-1 = 100101111
  - 303/4 = 100101111. **>>2** =1001011.11
    right shift

❖ 0.1 by multiplying 2, we never obtain an exact integer
0.1, 0.2, 0.4, … have always inexact representations

Fixed-point Operations

- Add and subtract numerators with the same denominators:

$$\frac{X}{q} \pm \frac{Y}{q} = \frac{X \pm Y}{q}$$

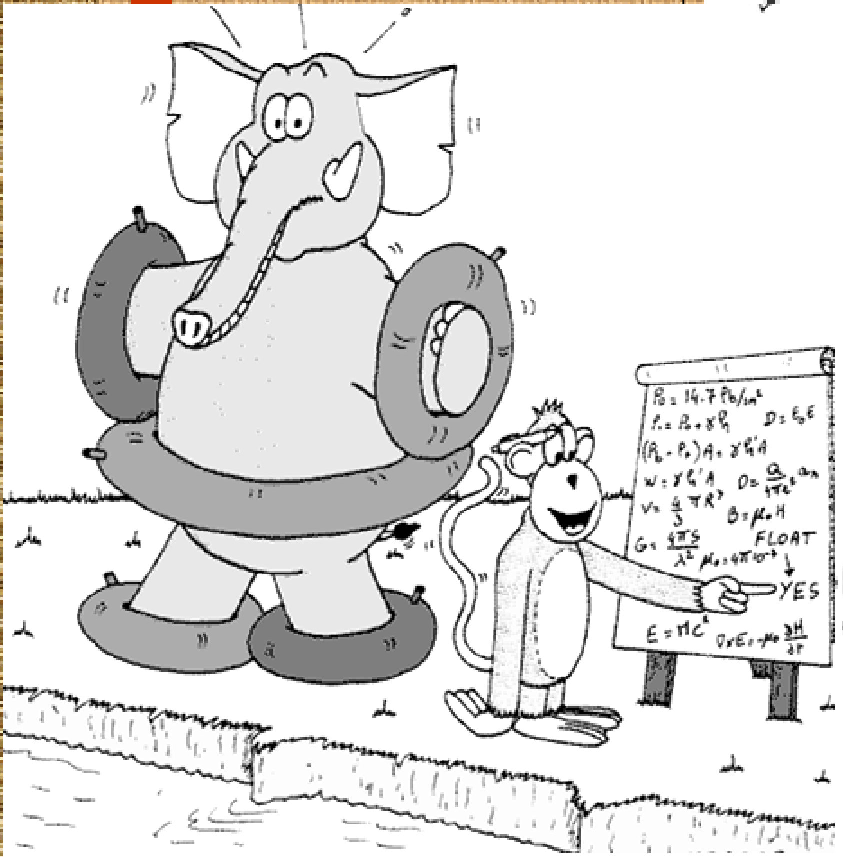
- Corrections are necessary after multiplications or divisions

- **Multiplication 2**: *shift left*, **Division 2**: *shift right*
- **Log$_2$ N:** can be found efficiently by shifts and squares,

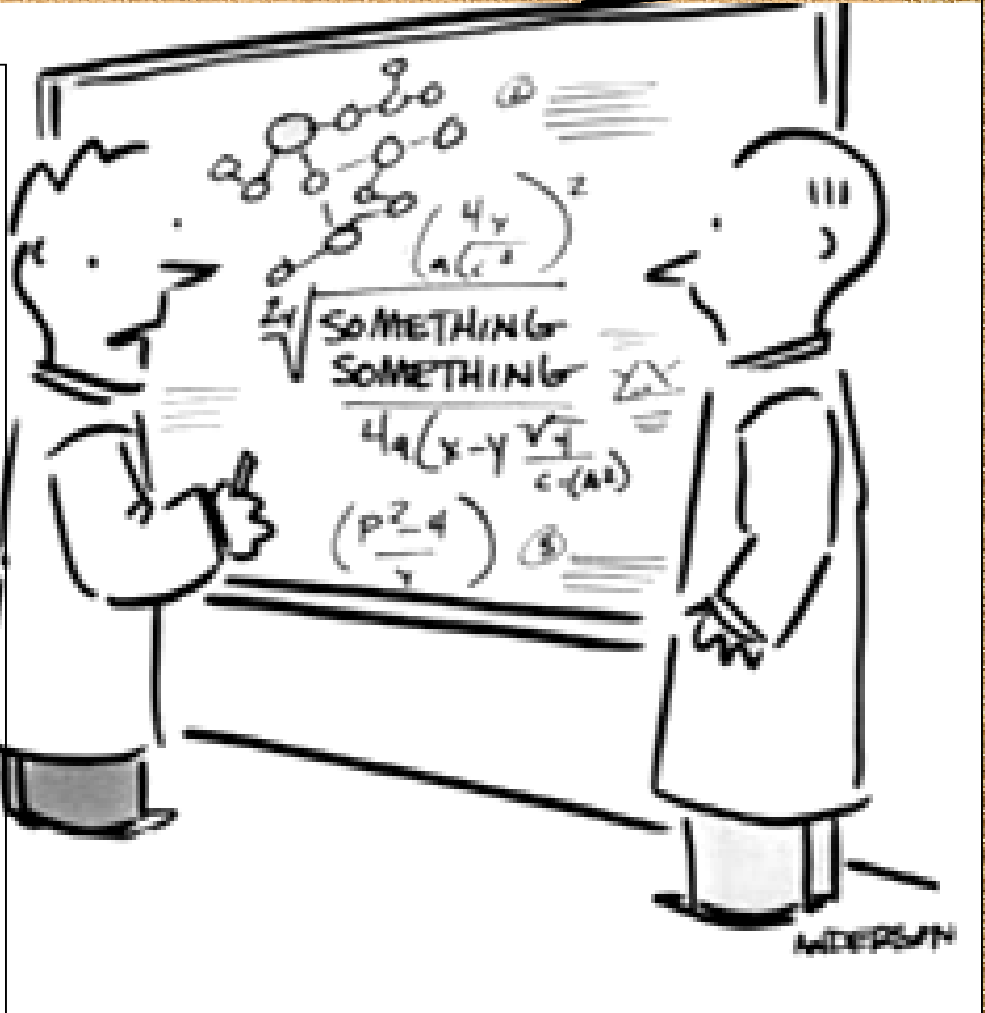
  https://en.wikipedia.org/wiki/Binary_logarithm

  (Paragraph: Iterative approximation)
- Sine, cosine, logarithm: *tables*

Fixed-Point Tricks in FPGA



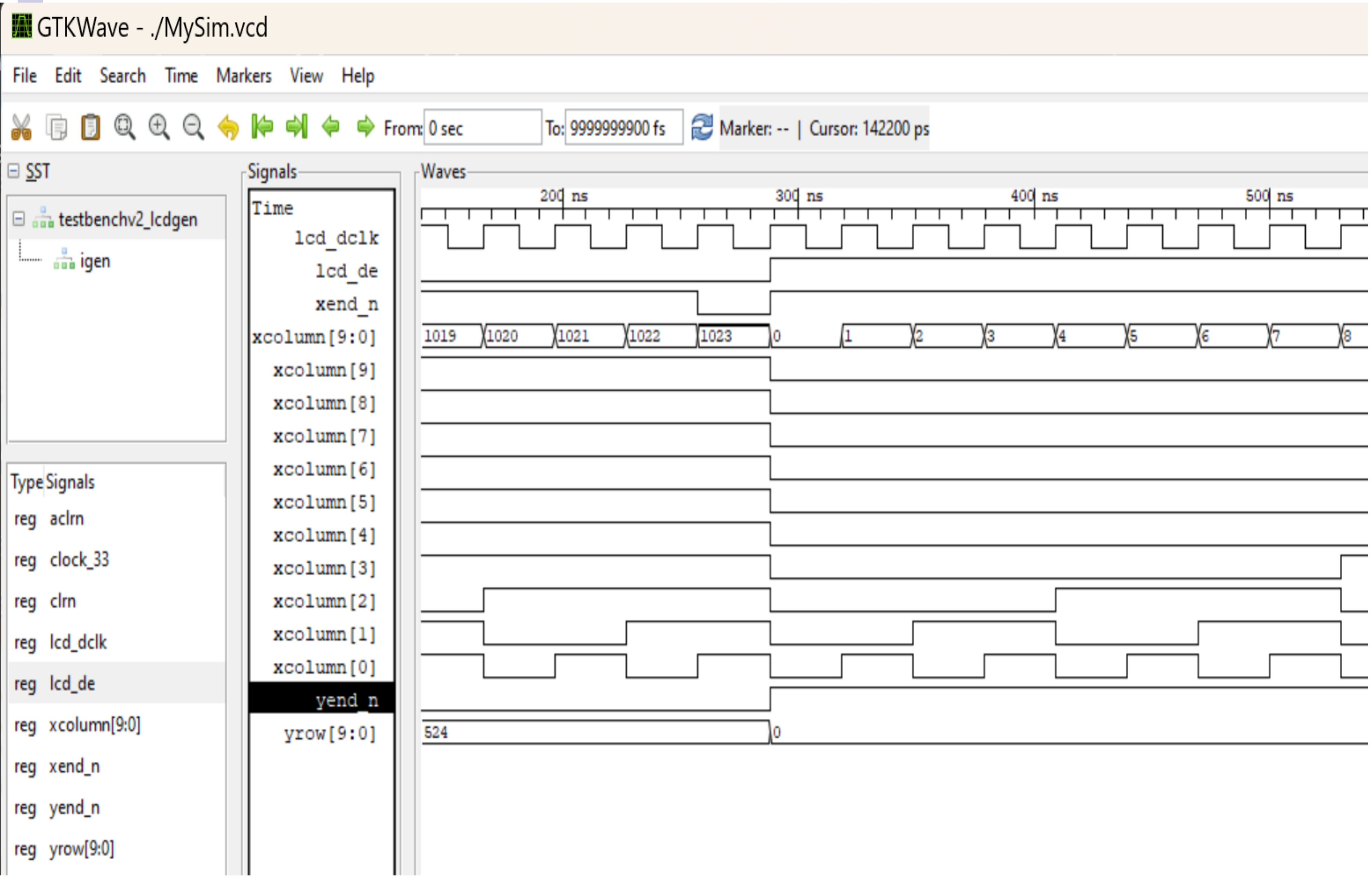
**"According to my calculation, you should not drown...
at least I think so"**



"It's an inexact science."

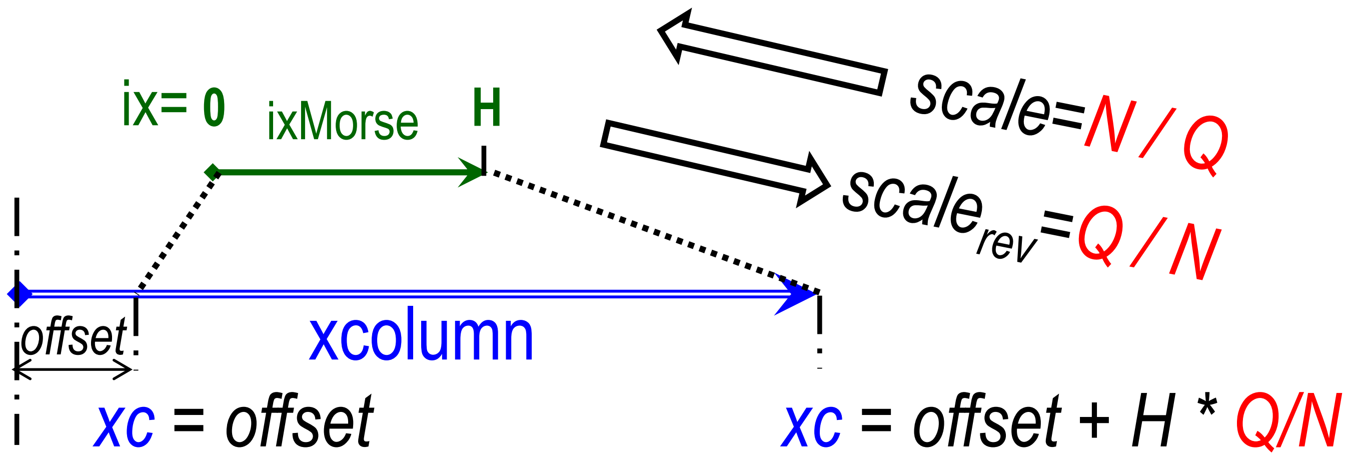
Obrázky z knihy: Barhani: Computer Arithmetic, Oxford 2012

VeekMT2_LCDgenV2



Type	Signals
reg	aclrn
reg	clock_33
reg	clrn
reg	lcd_dclk
reg	lcd_de
reg	xcolumn[9:0]
reg	xend_n
reg	yend_n
reg	yrow[9:0]

1D linear transformation: translate + scale



ix = ((xc-offset) * N) / Q

xc = (ix * Q) / N + offset

In circuits, we better divide by powers of 2,
so we select N and Q according to our calculations.

1D transformation

Let: offset=0; N = 7; Q = 16 (=24)**

1. By multiplication: x=(xcolumn*7)/16

2. By a continuous counter

xcolumn = xt	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	...
counter = 7*xt	0	7	14	21	28	35	42	49	56	63	70	77	84	91	98	...
counter / 16	0	0	0	1	1	2	2	3	3	3	4	4	5	5	6	...

3. By a limited counter

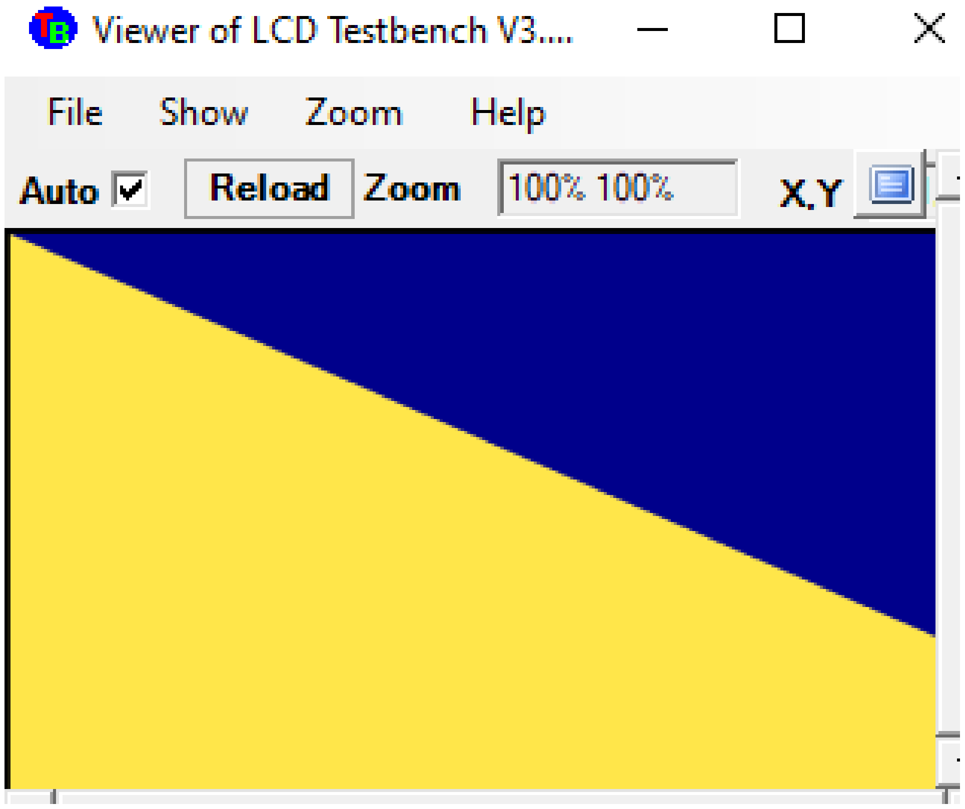
xcolumn = xt	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	...
counter = (7*xt) mod 16	0	7	14	5	12	3	10	1	8	15	6	13	4	11	2	...
(7*xt) / 16	0	0	0	1	1	2	2	3	3	3	4	4	5	5	6	...

3=19-16 6=22-16 2=20-16

5=14+7-16=21-16 1=17-16 4=20-16

N=7, Q=16

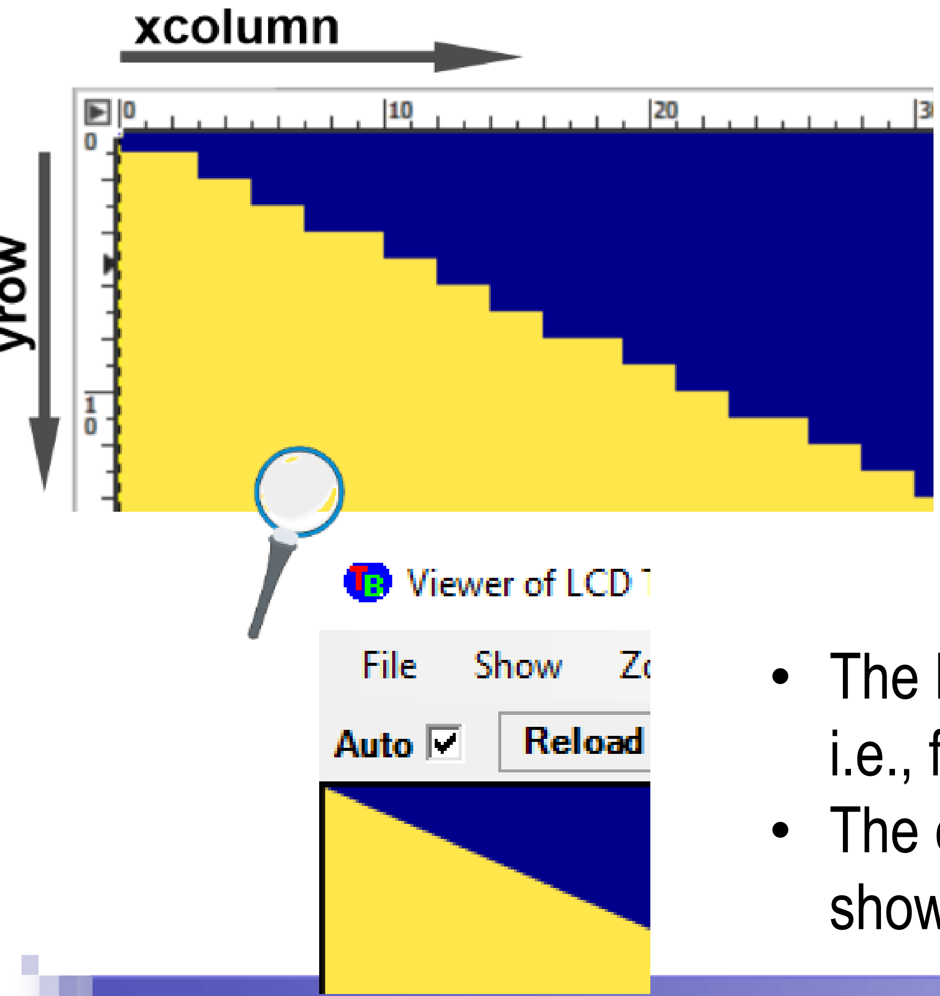
```
process(xcolumn,yrow)
variable RGB :
RGB_t:=BLACK;
begin -- process
 if 16*yrow > 7*xcolumn
    then RGB:=YELLOW;
    else RGB := DARKBLUE;
 end if;
 RGBcolor <=RGB;
end process;
```

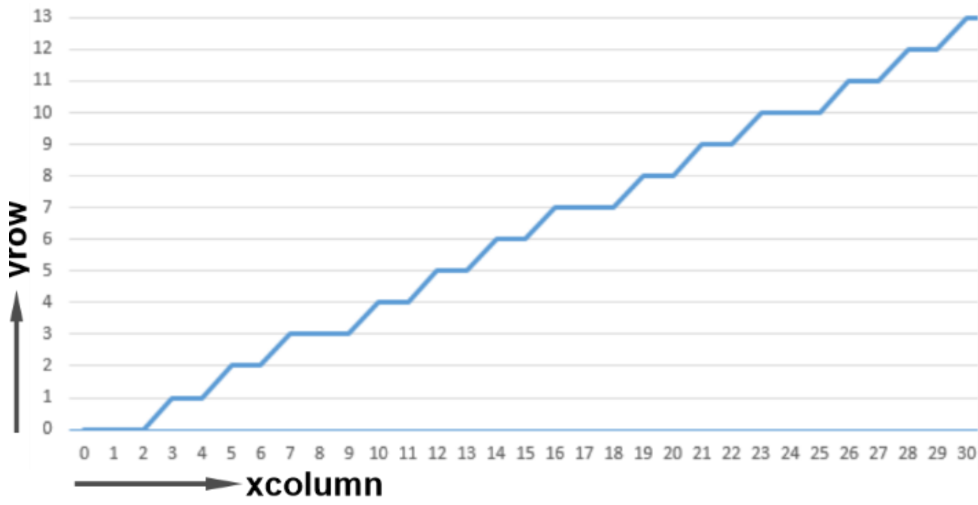


We do not see irregularity until we zoom the image.

Line and Graph

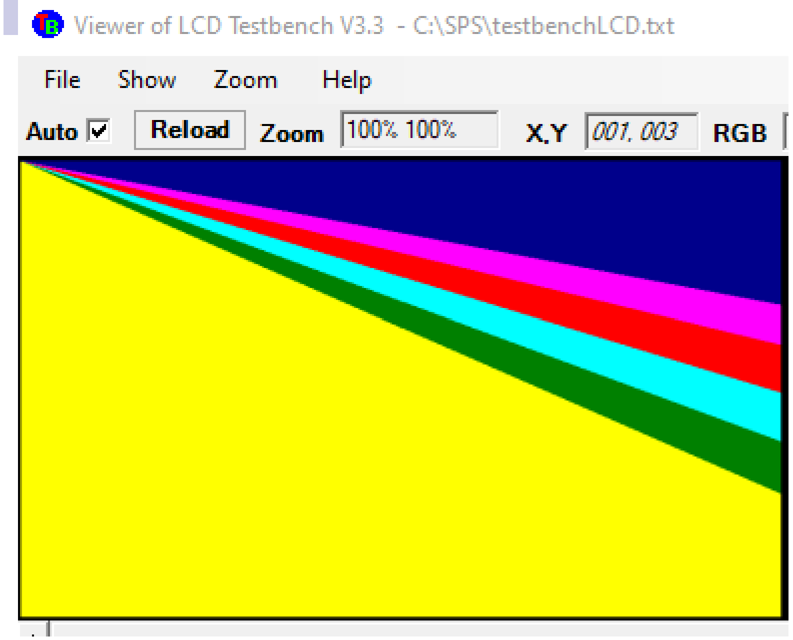
xcolumn	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	...
x	**0**	**0**	**0**	**1**	**1**	**2**	**2**	**3**	**3**	**3**	**4**	**4**	**5**	**5**	**6**	**...**





- The LCD has its y-axis reversed, i.e., from top to bottom.
- The classic graph has a rising y-axis when showing the counter above.

Q value



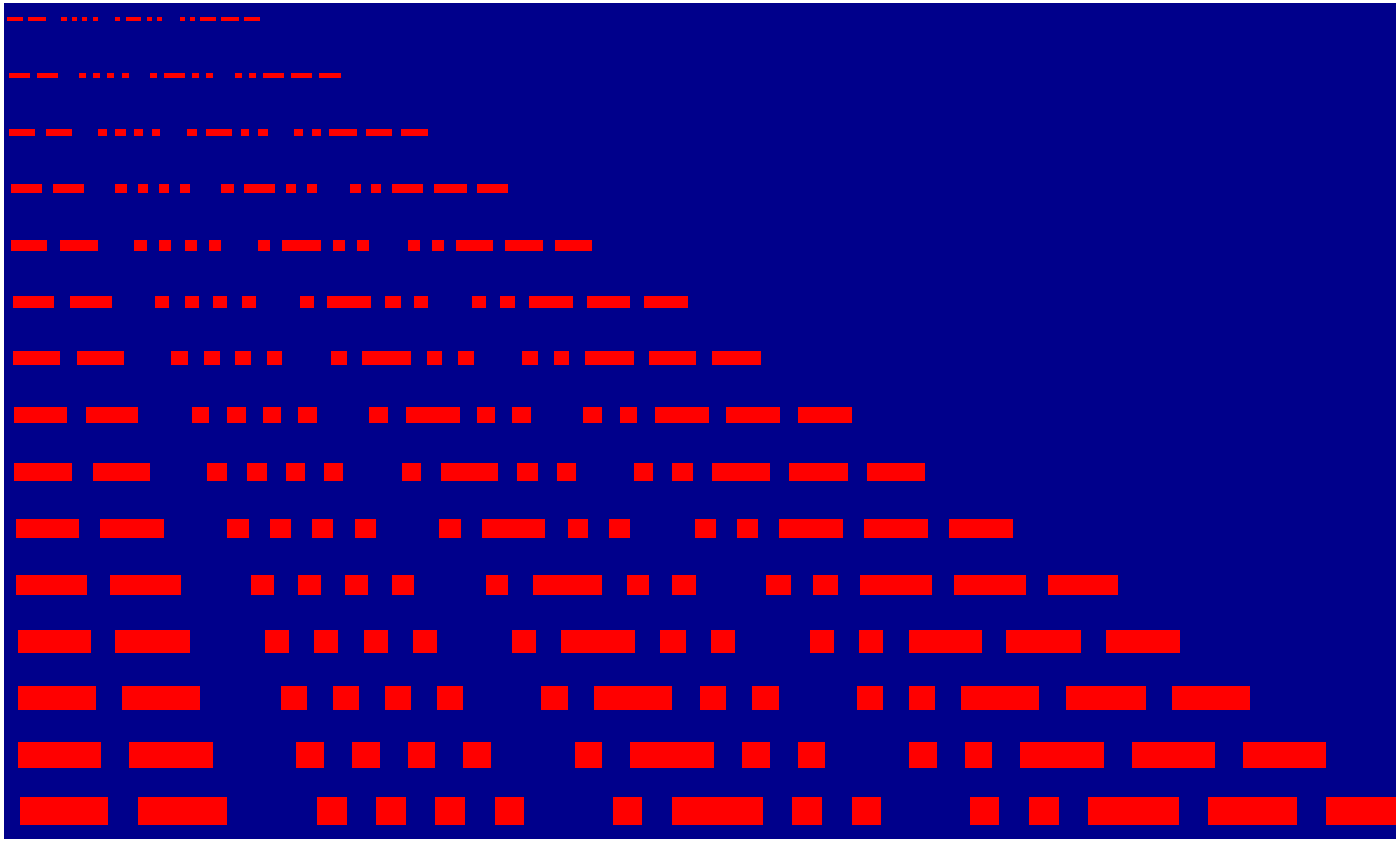
- Lines allow any Q>0.

```
x := to_integer(xcolumn); y := to_integer(yrow);
RGB := DARKBLUE;
if 37*y > 7*x then RGB:=VIOLET; end if;
if 29*y > 7*x then RGB:=RED; end if;
if 23*y > 7*x then RGB:=AQUA; end if;
if 19*y > 7*x then RGB:=GREEN; end if;
if 16*y > 7*x then RGB:=YELLOW; end if;
```

- If we do not need modulus then limited counters have lower complexities for any integer Q>0.
- But divisions and continuous fixed-point counters have acceptable complexities only for **Q=2**K**, where K>=**0.**

This example also discovers that the GHDL performs simulations much faster with integers than with unsigned or signed.

Morse Enlarged by Fixed-point Counter



0

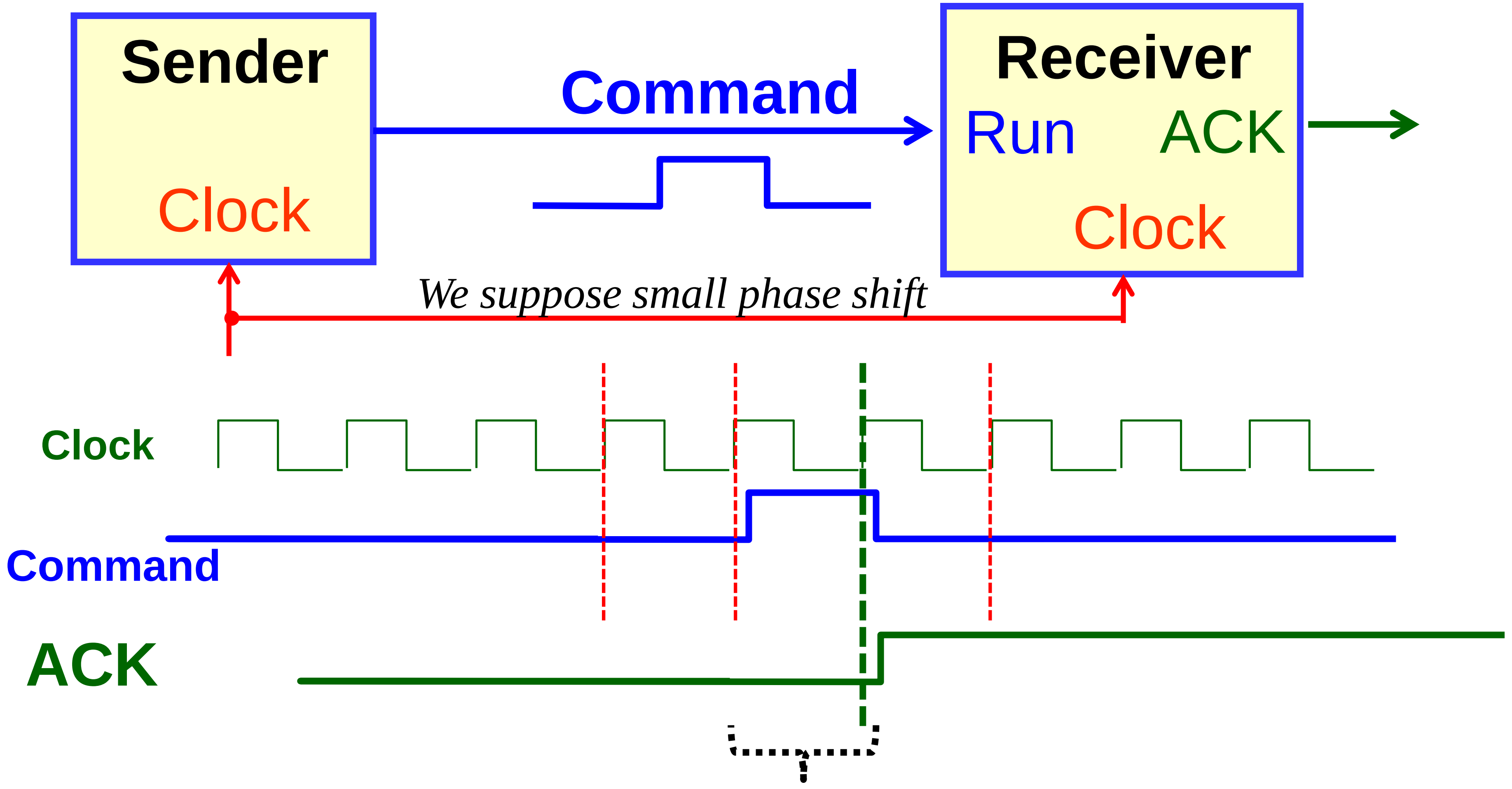
DECOMPOSITIONS

into more parts can require mutual communications

[Image: 9gag.com]

Clock synchronisation

We depend on two clocks, assuming that the receiver is always ready :-(



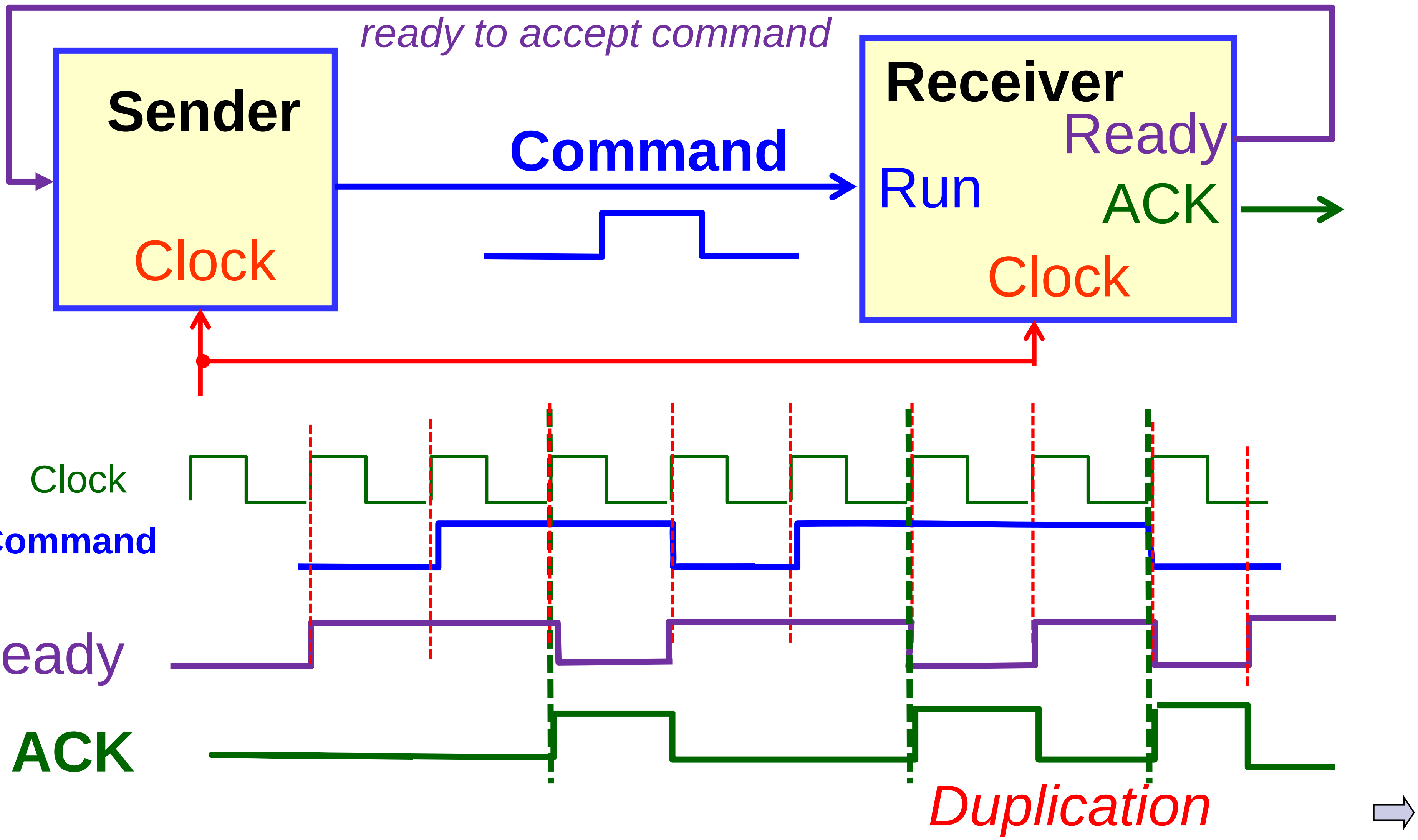
1 period clock delay

Synchronous command with clock input

- Advantages:
  - Simple.
  - Suitable for receivers with fast response times that always wait for commands.
- Disadvantages:
  - A small window for correct command recognition.
  - The solution depends on the exact phase of the clock.
  - Missing verification that the receiver is ready to accept the command.

READY

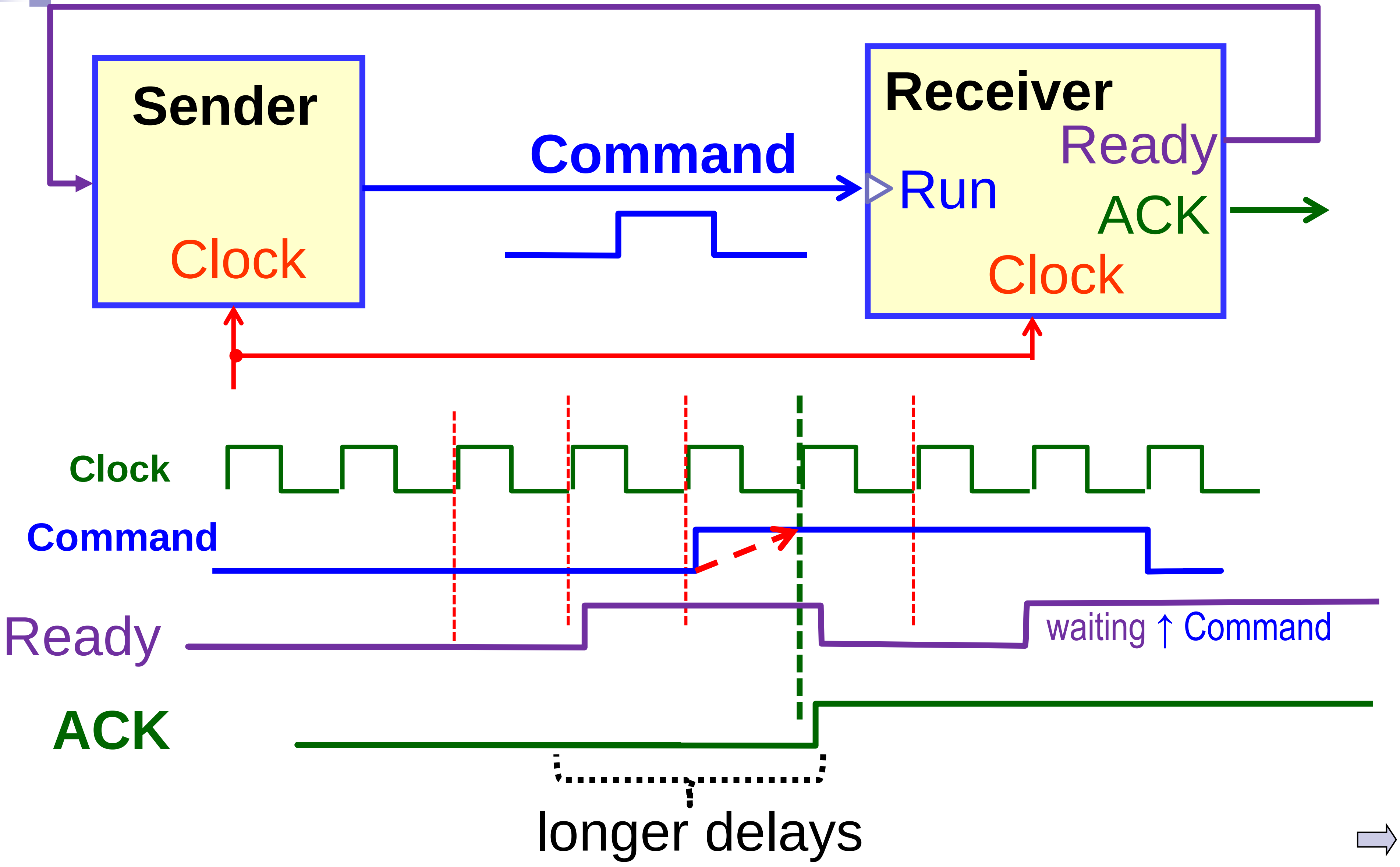
Again, we have a dependency on the same clock.
The command must last only an appropriate time:-(



Enhancement by READY

- Benefits
  - Reducing the risk of losing commands.
- Disadvantages
  - Only applicable when a sender and a receiver have the same clock.
  - If the sender deactivates the command too late, it is accepted multiple times.
  - Delay in the recever response by multiple clock pulses.

Edge Detection

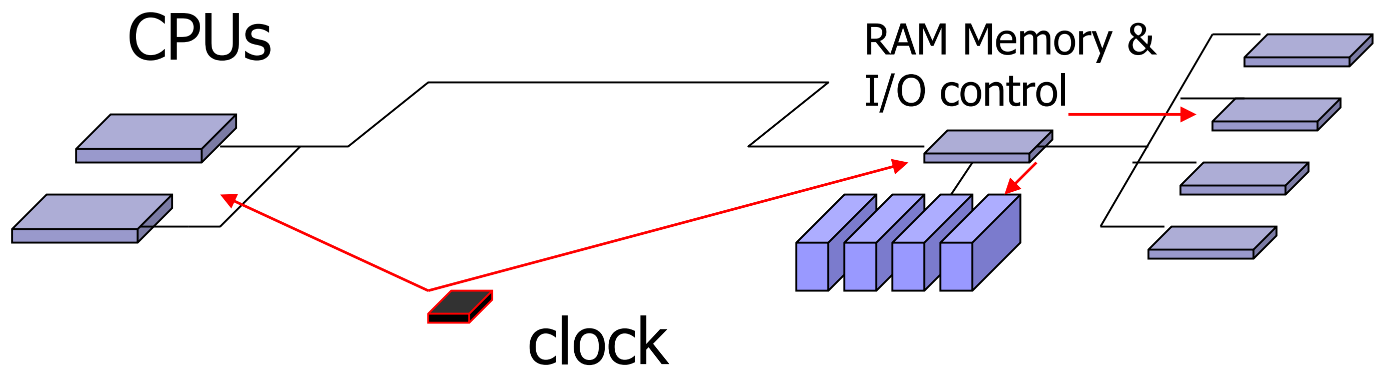


Enhancement by edge detection command

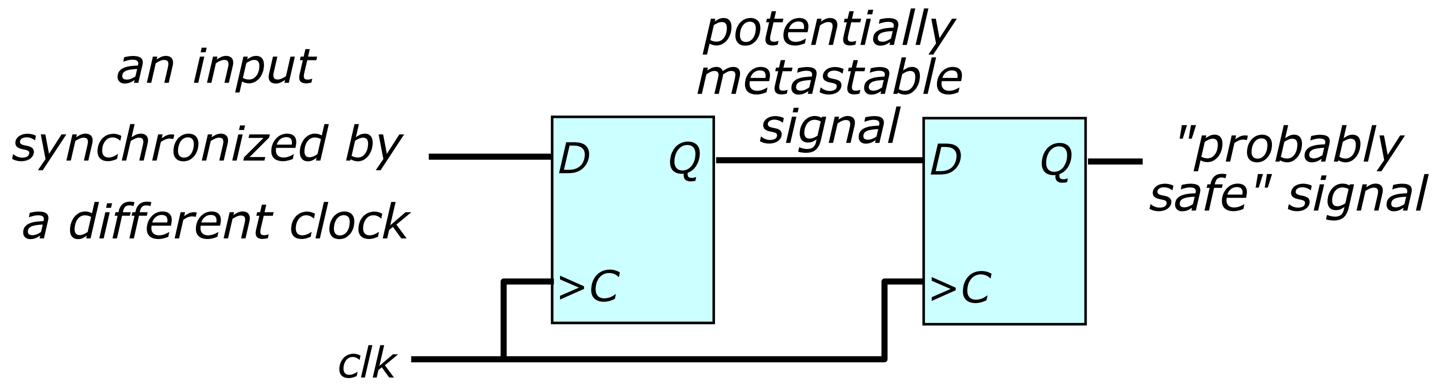
- Benefits
  - A very robust solution.
  - Senders transmit commands only when the receivers are ready to accept.
  - Receivers detect the rising edge of commands by comparing them with previous values, thus avoiding false repetitions.
- Disadvantages
  - We still need the same clock.

Clock domain

a set of synchronous signals with the same clock

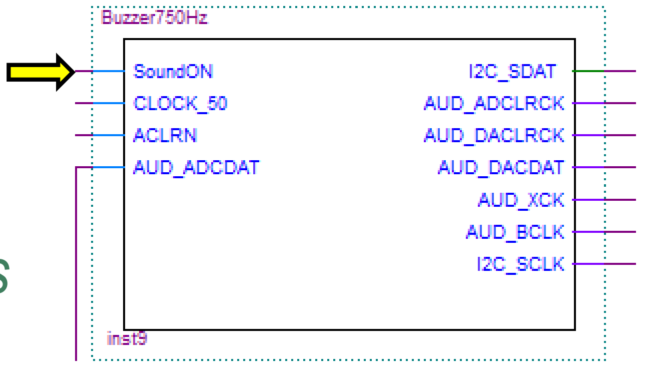


Crossing clock domain boundaries requires synchronizers for isolating metastability



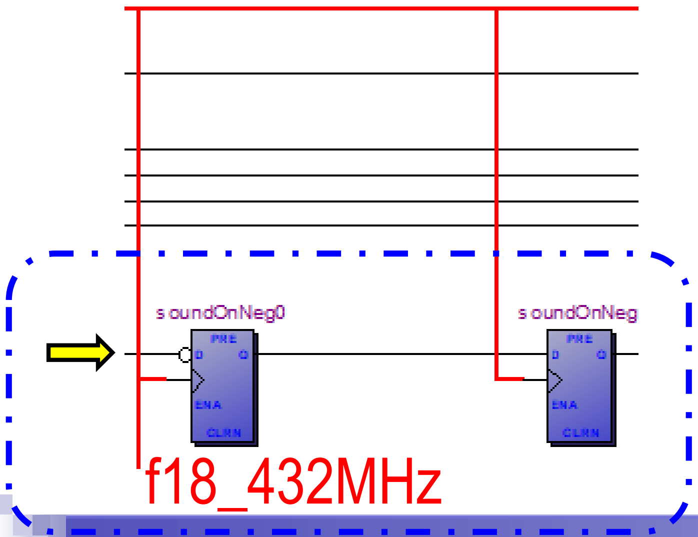
Even the Morse Code Buzzer contains a Synchronizer

```
signal soundOnNeg, soundOnNeg0 : std_logic;
begin --architecture
   process(f18_432MHz)
   begin -- Synchronizer for crossing clock domains
      if rising_edge(f18_432MHz) then
         soundOnNeg <= soundOnNeg0; soundOnNeg0 <= not SoundON;
      end if;
   end process;
```

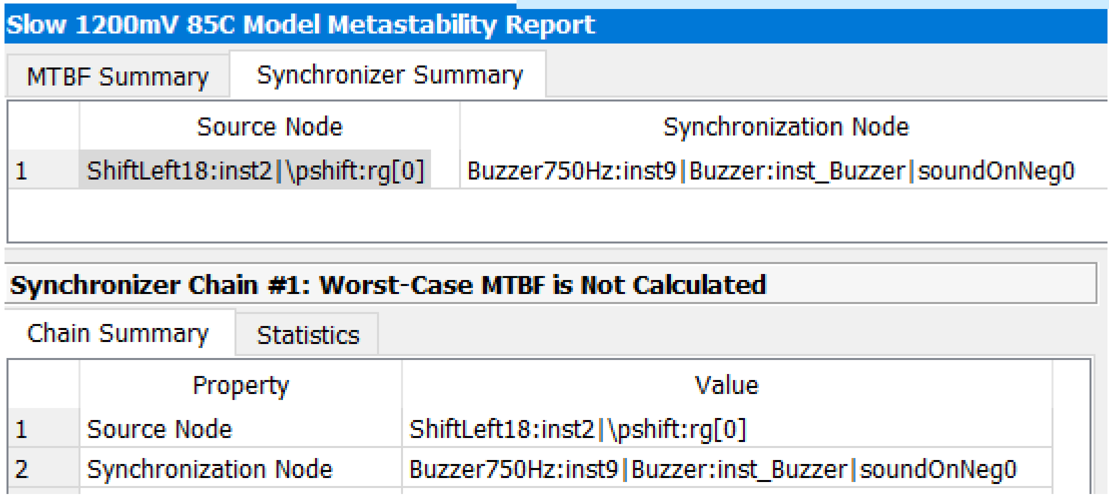


We synchronize here between CLOCK_50 domain (50 MHz) and the audio clock domain 18.432 MHz

RTL Viewer



Time Quest Analyzer Report



Slow 1200mV 85C Model Metastability Report

MTBF Summary | Synchronizer Summary

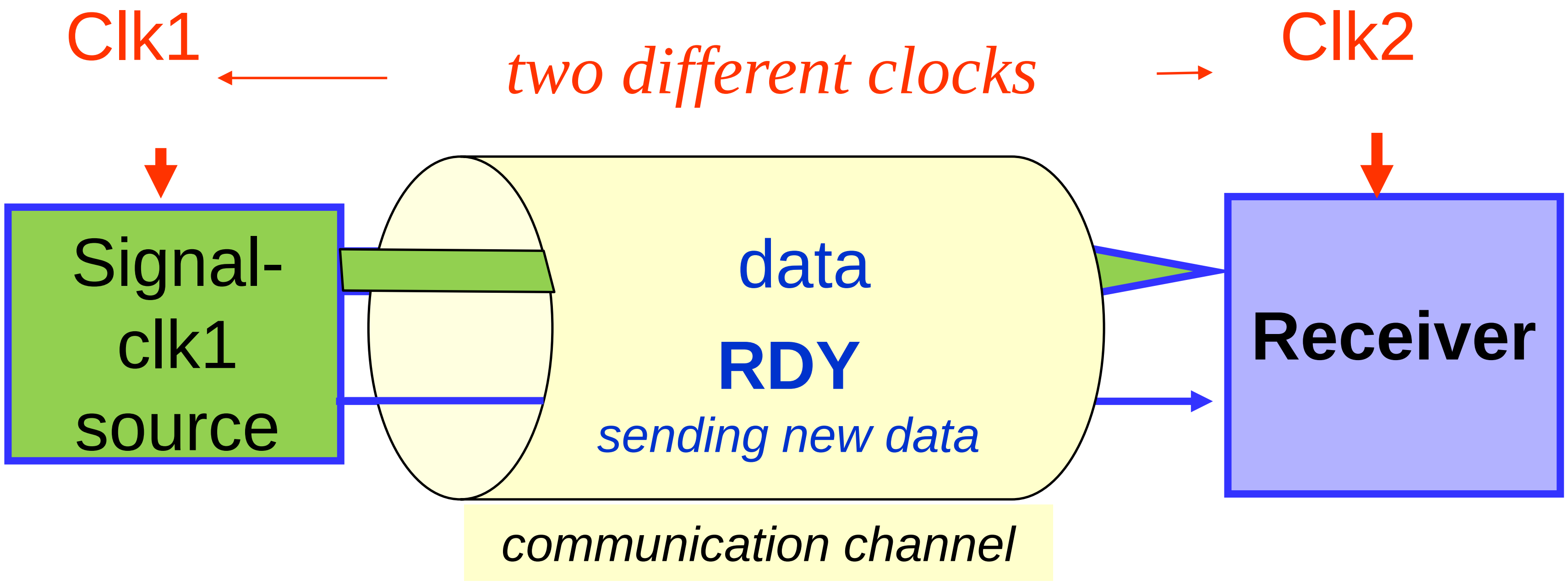
	Source Node	Synchronization Node
1	ShiftLeft18:inst2|\pshift:rg[0]	Buzzer750Hz:inst9|Buzzer:inst_Buzzer|soundOnNeg0

Synchronizer Chain #1: Worst-Case MTBF is Not Calculated

Chain Summary | Statistics

	Property	Value
1	Source Node	ShiftLeft18:inst2|\pshift:rg[0]
2	Synchronization Node	Buzzer750Hz:inst9|Buzzer:inst_Buzzer|soundOnNeg0

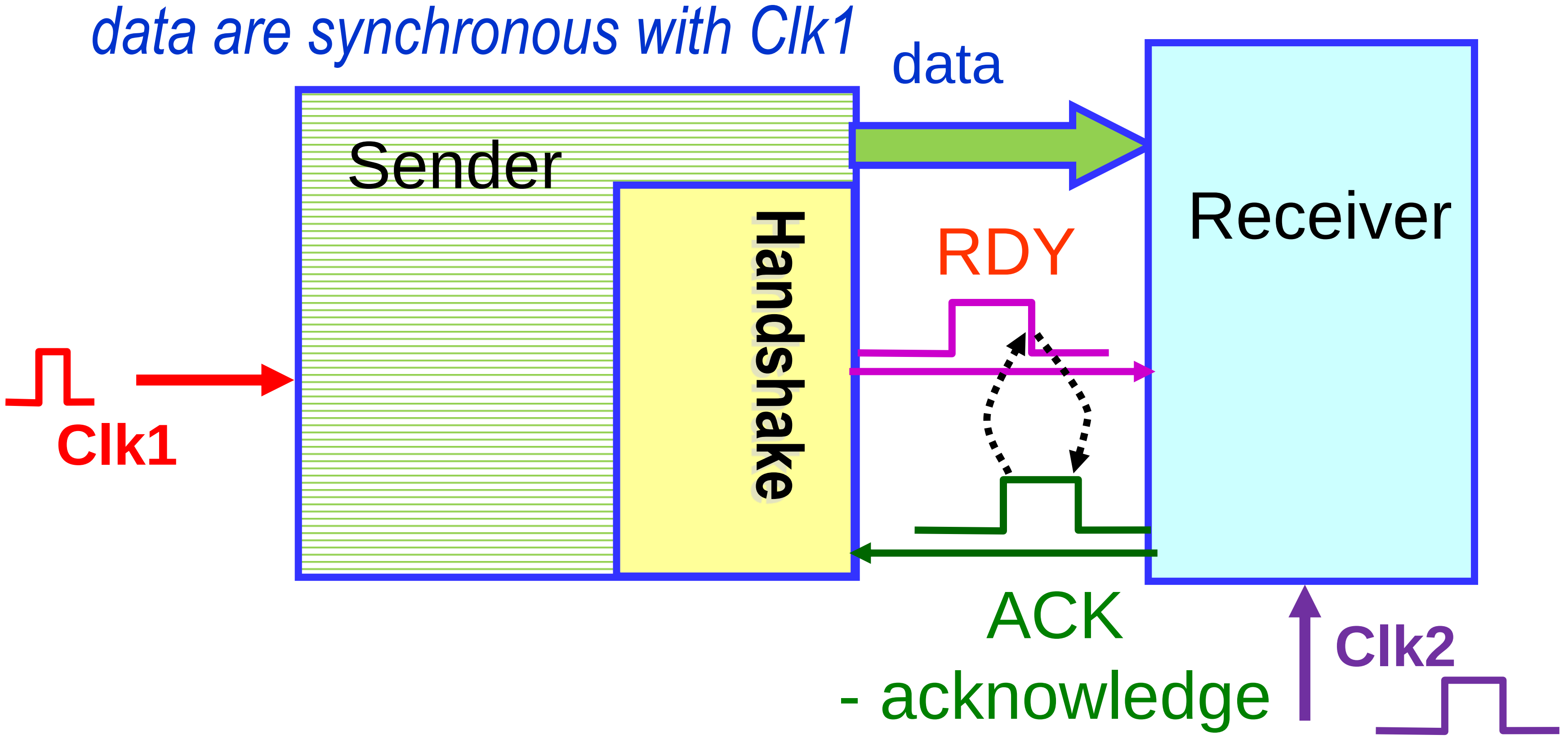
Transfer between devices with different clocks



Sending without implicit confirmation.
We assume either a fast response from the receiver or a command input queue (FIFO) implementation.

Similar to the UDP network protocol
(UDP - User Datagram Protocol)

Hand-shaking

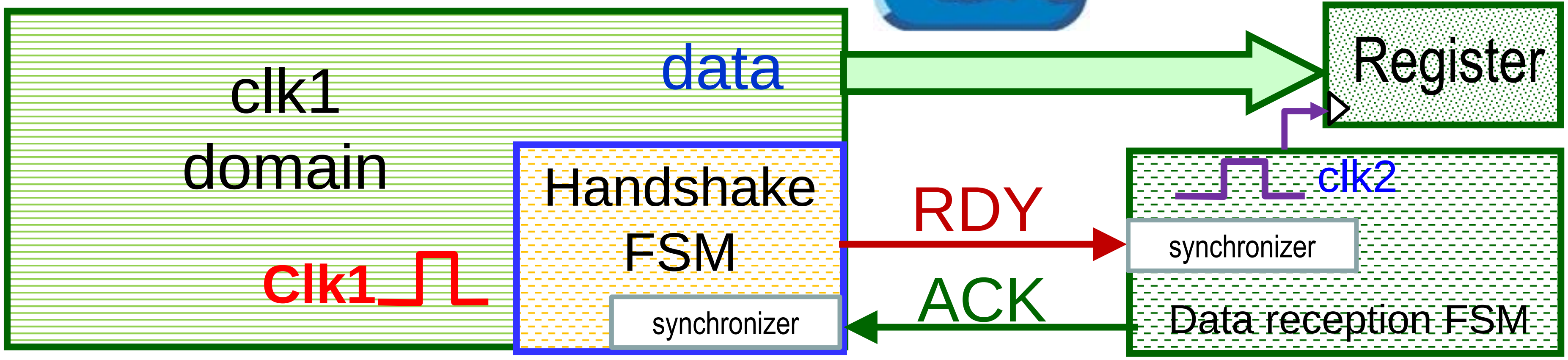


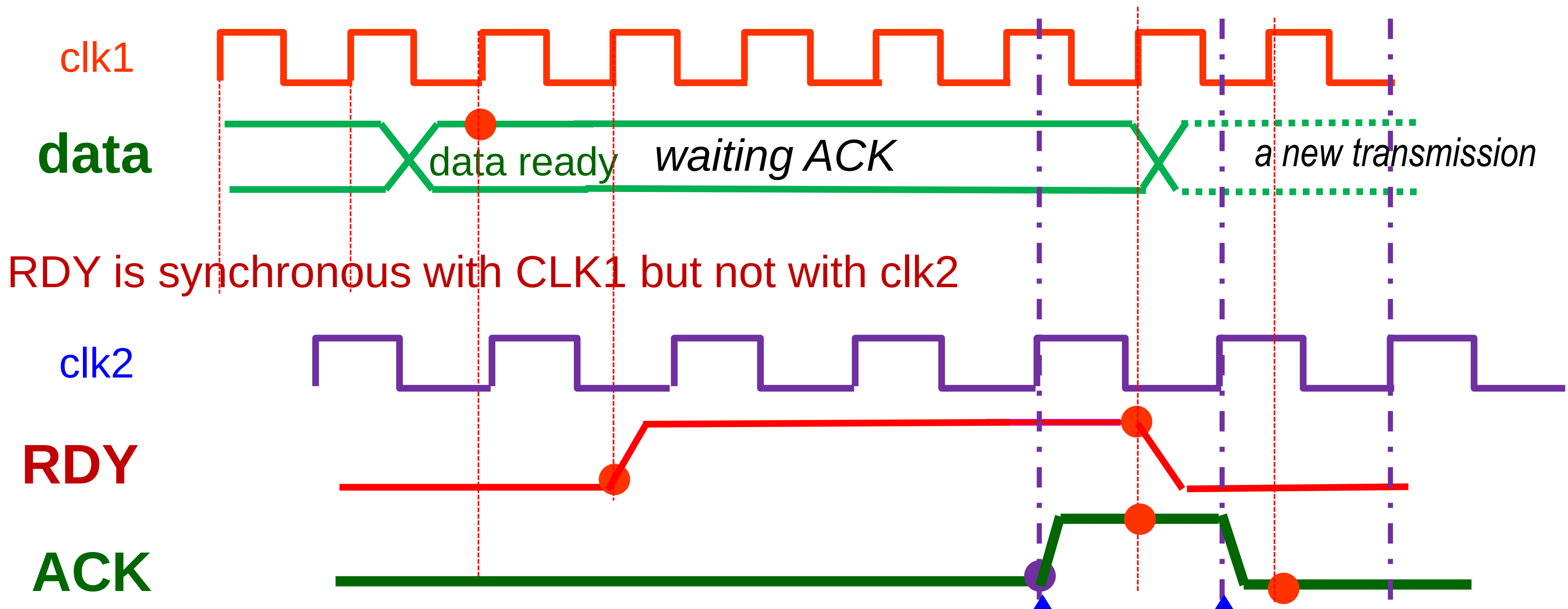
Positively confirmed transmission (hand-shaking)

analogy of the TCP network protocol

(Transmission Control Protocol)

Handshake





Receiver confirmed
by ACK that accepted data

Receiver confirmed
detection of RDY end

Forbidden Structures

- Asynchronous Counters (Ripple Counter)
- Asynchronous input priority violation
- Dependency on the double clock

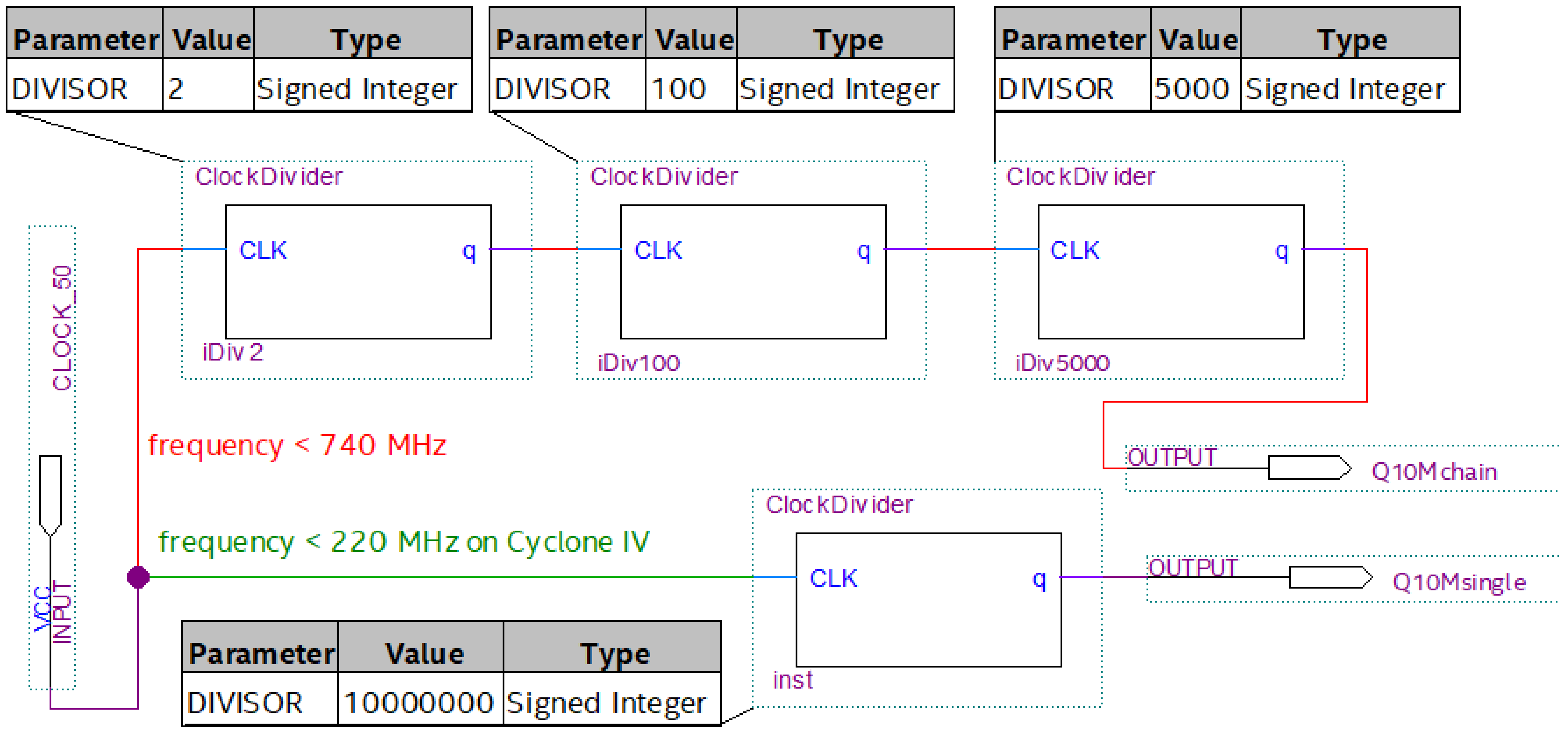
Ripple Counters 2^n
cz: asychronous counters

Simple counters,

but with phase delays…

Asynchronous and Synchronous

asynchronous chain



Parameter	Value	Type
DIVISOR	2	Signed Integer

Parameter	Value	Type
DIVISOR	100	Signed Integer

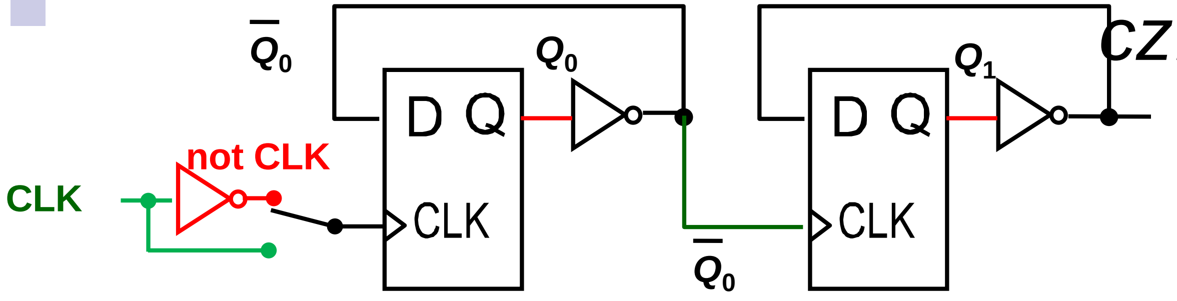
Parameter	Value	Type
DIVISOR	5000	Signed Integer

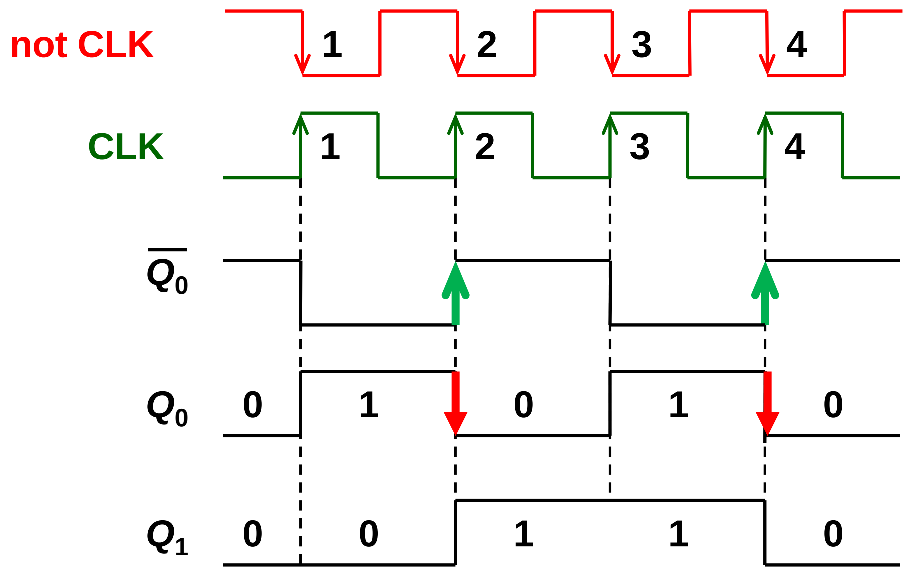
Parameter	Value	Type
DIVISOR	10000000	Signed Integer

synchronous

4-bit Ripple counter

cz: asynchronous counte





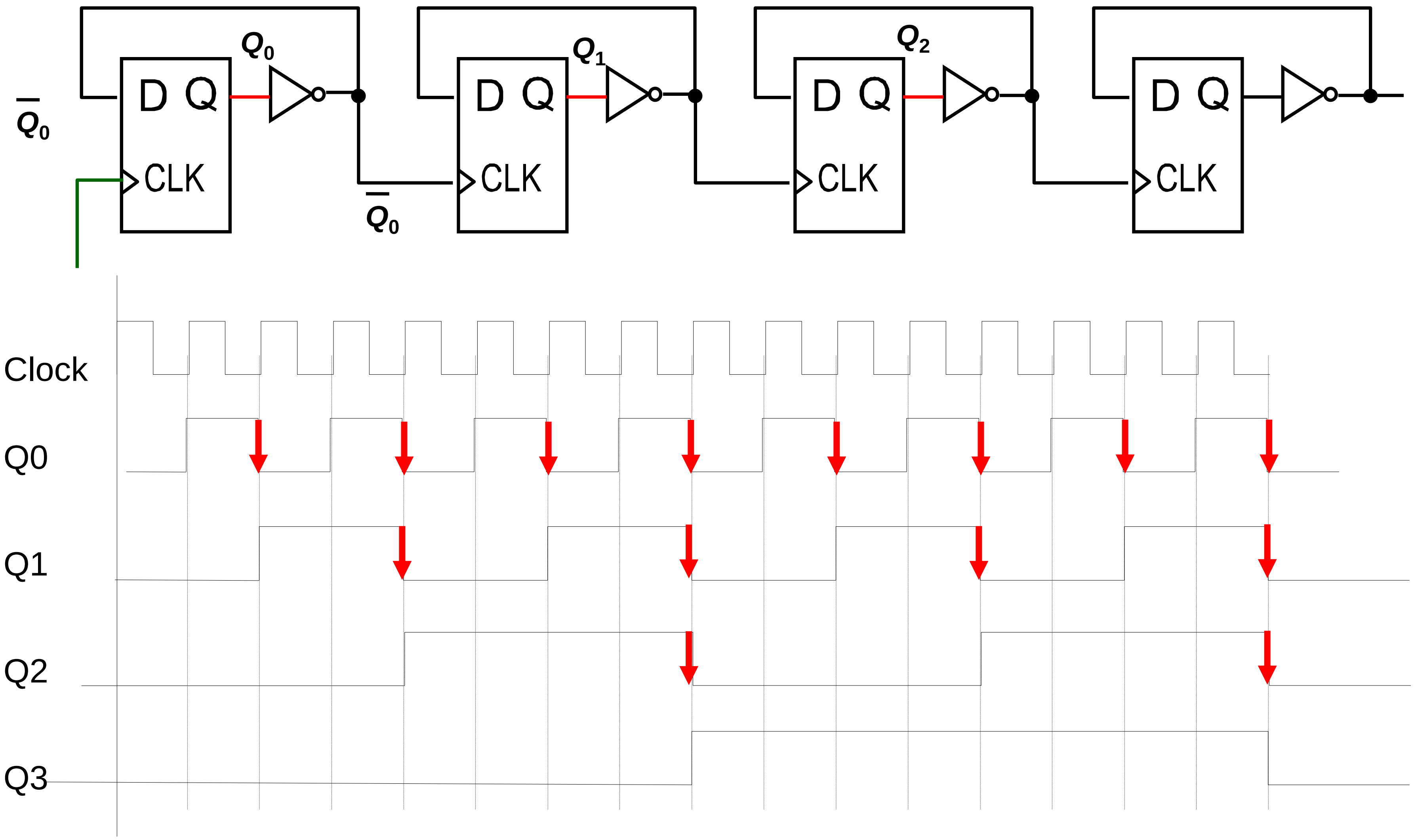
Timing diagram

00 → 01 → 10 → 11 → 00 ...

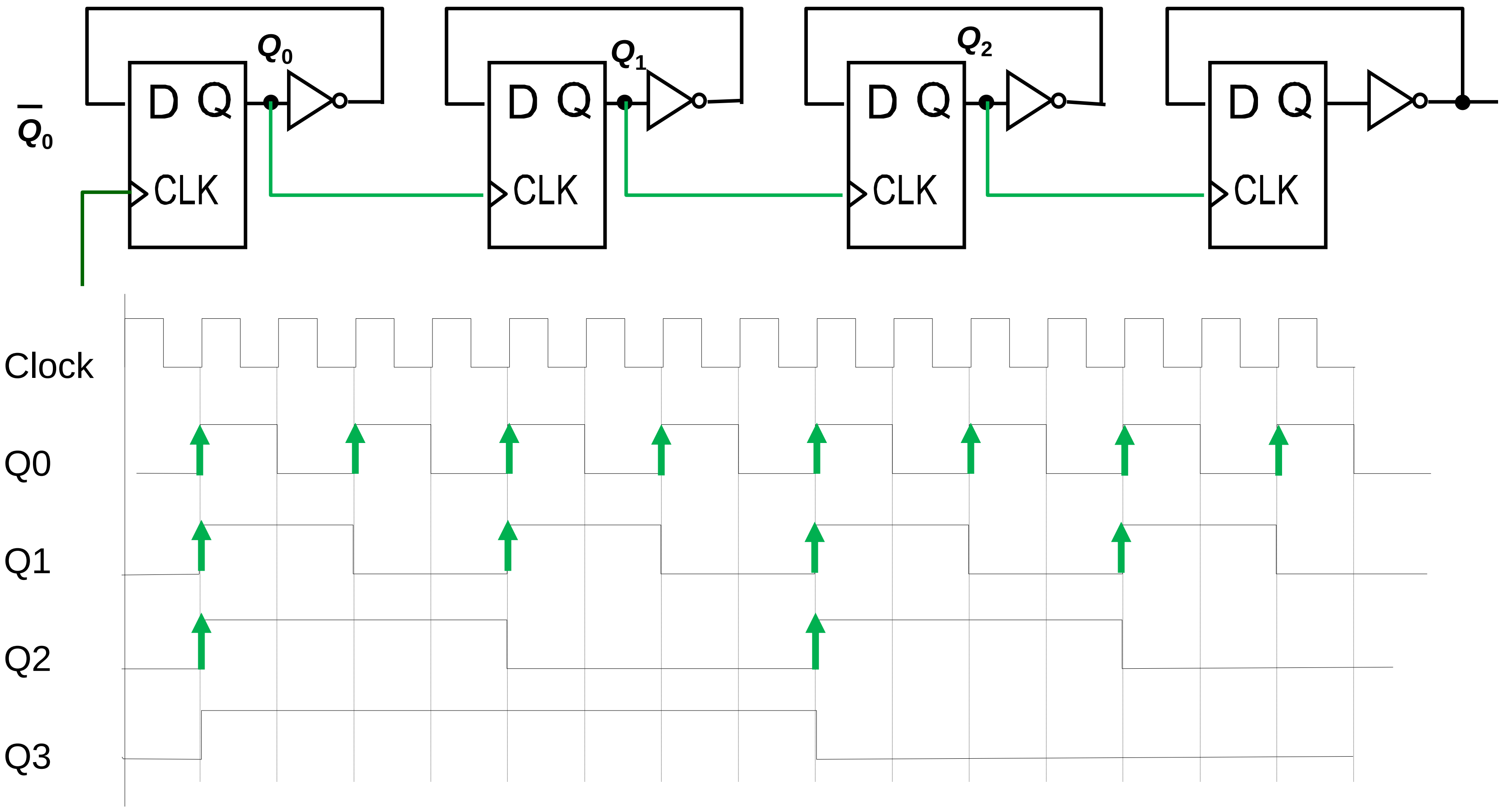
The first DFF can operate on the rising or falling edges of the clock signal, depending on our needs, but subsequent DFFs must knock on the falling edges to count up. This follows from the binary sequence = its high bit changes to the opposite just as the low bit goes from 1 to 0.

Frequency counter/gunner 16

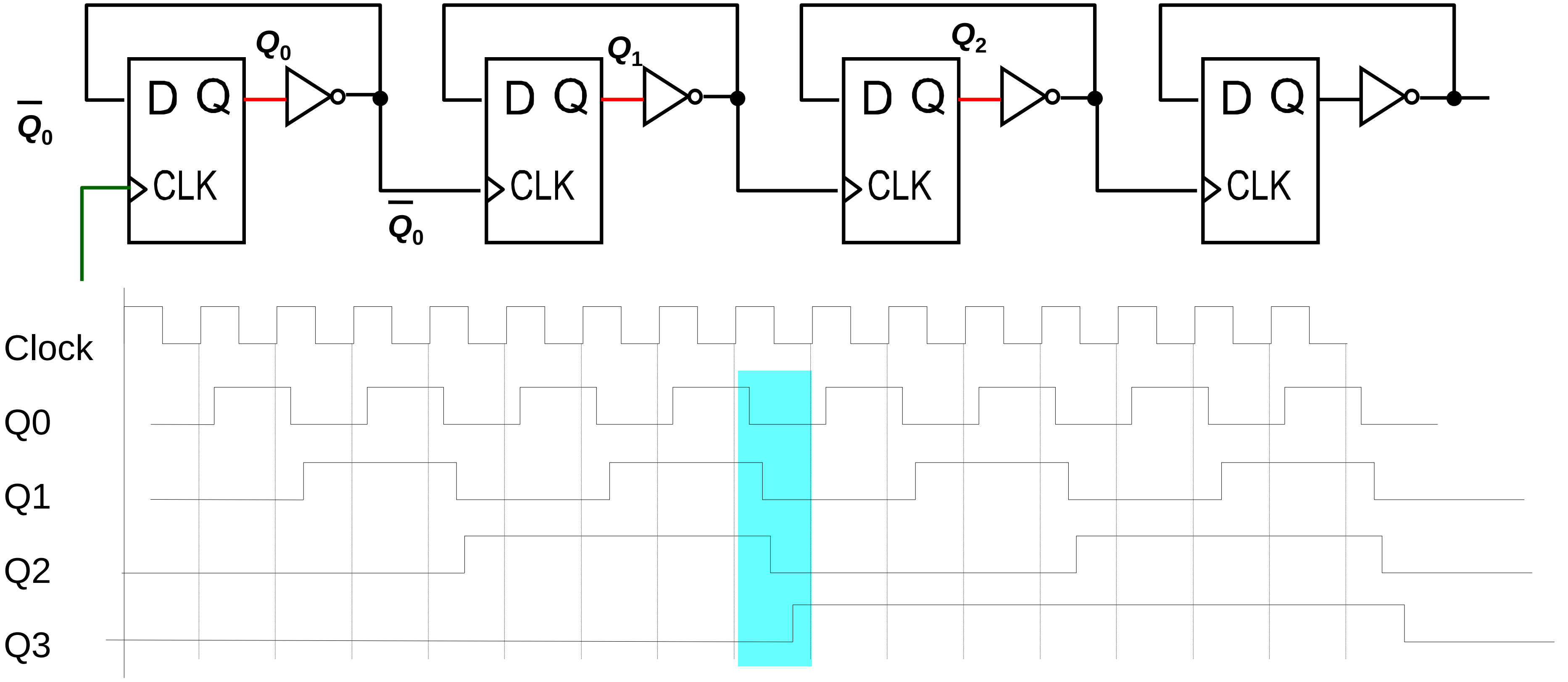
Reading up



Counter down



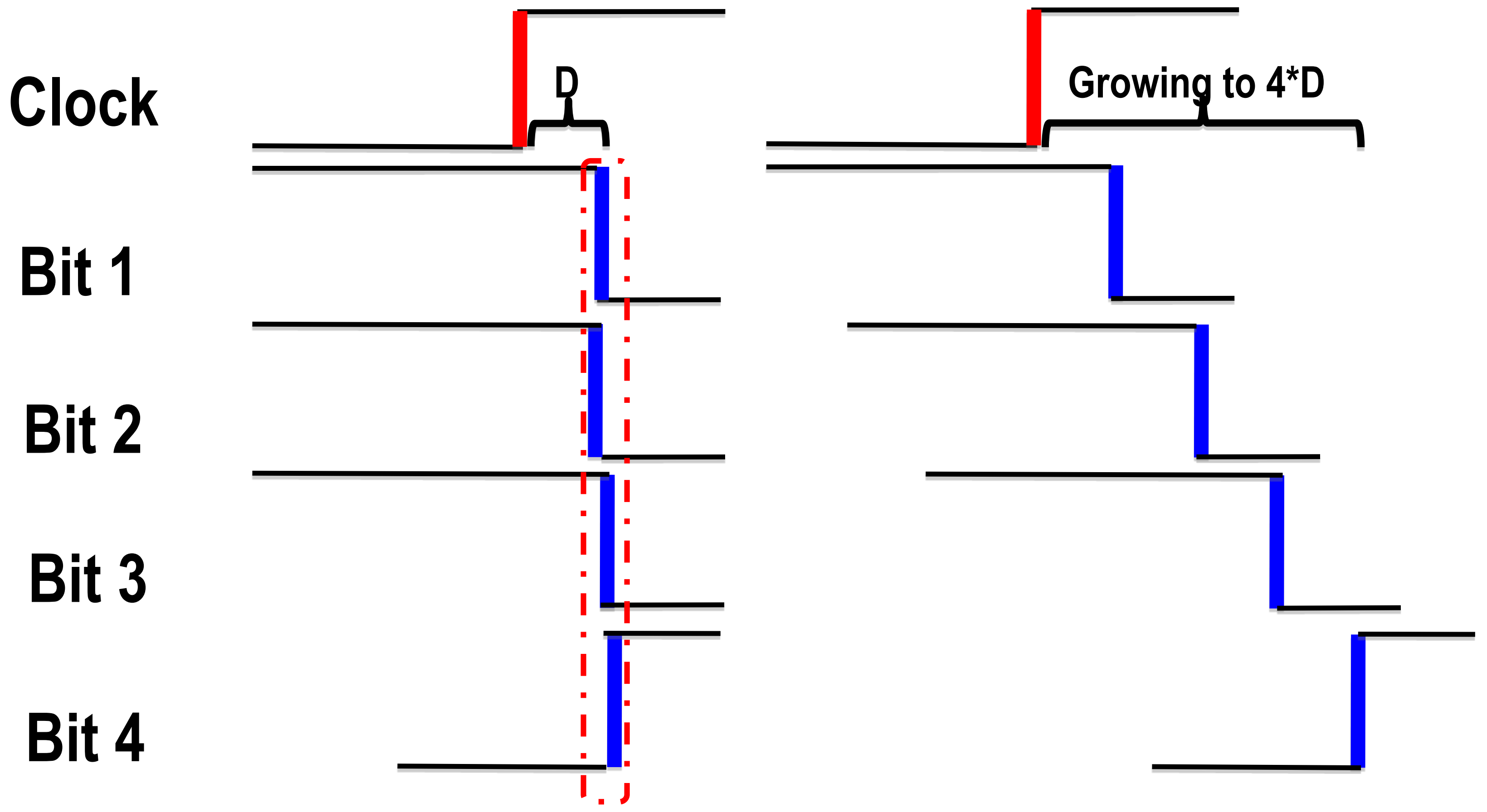
The disadvantage is the increasing delay of outputs



Intermediates - GRAY'S CODE 0111→0110→0100→0000→1000

0111 > 0110 > 0100 > 0000 < 1000

Synchronous versus asynchronous counter

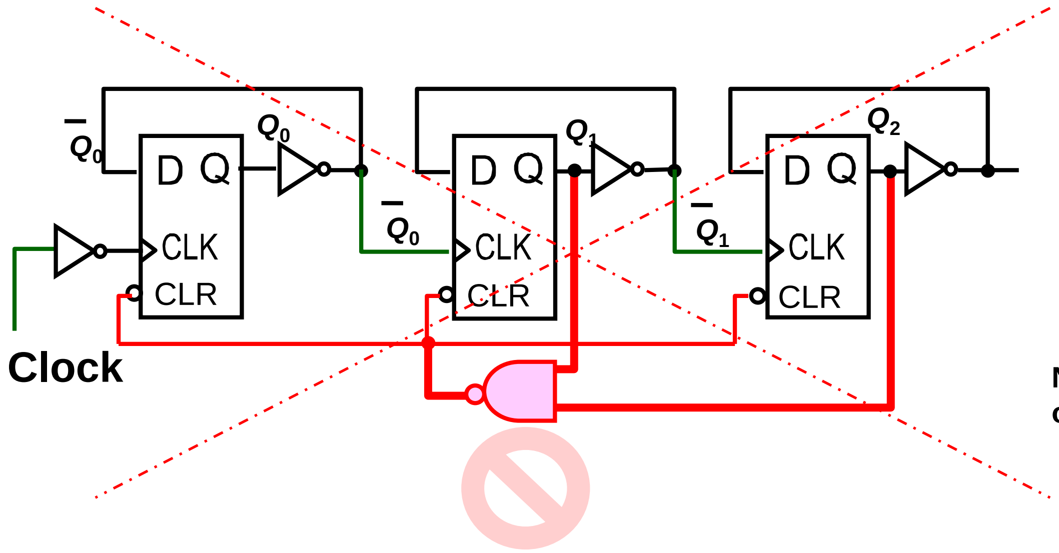


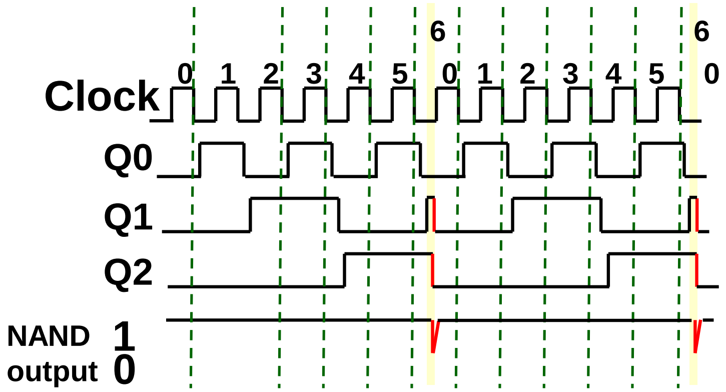
Slight phase shift **Significant phase shift**

The outputs of the synchronous counter are admittedly knocking
in a short time interval,
but ***not all at once!!!***
They are slowed down by their load (fan-out).

Obsolete asynchronous dividers

- Beware of schematics on sites where asynchronous inputs are used as working inputs. All such circuits refer to older TTL logic that was slower and did not respond to short gambling pulses.



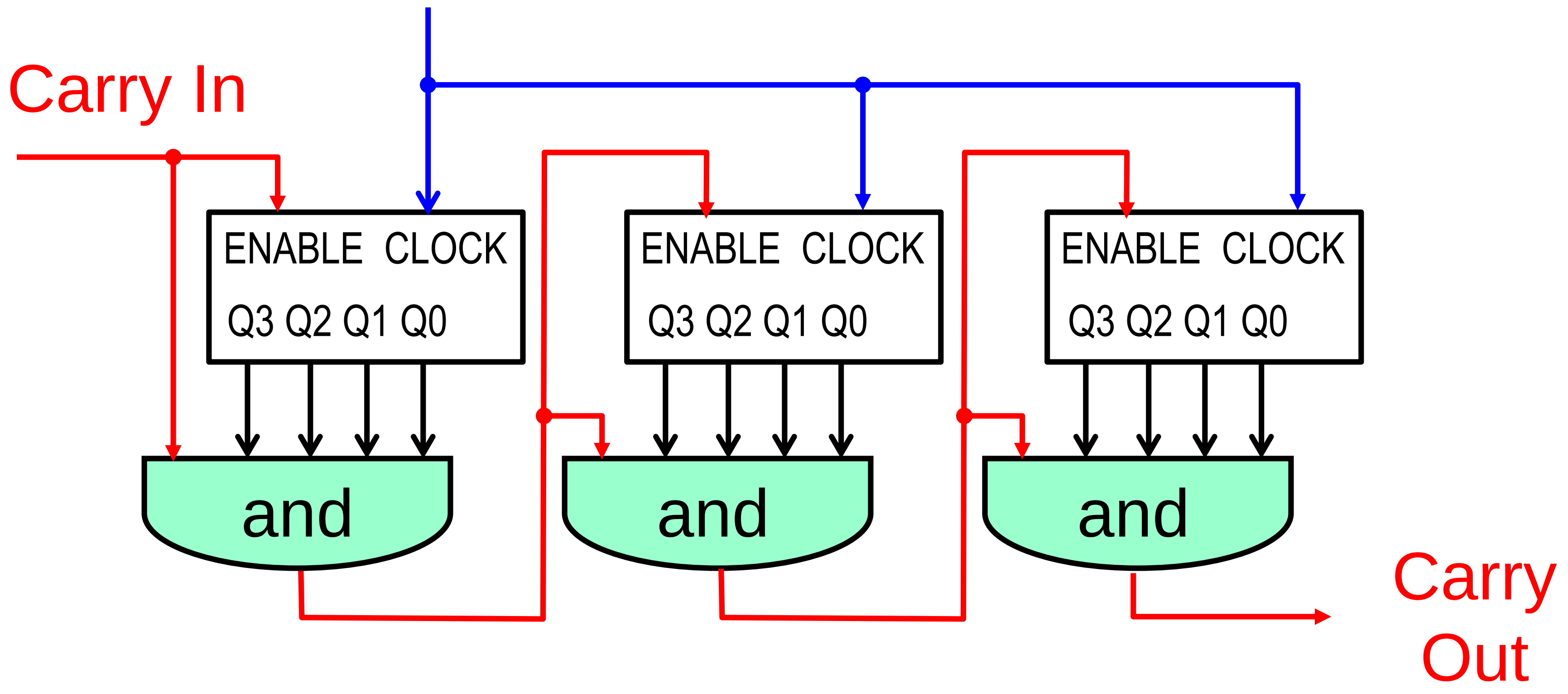


Older design of TTL divider 6 - nowadays risky !

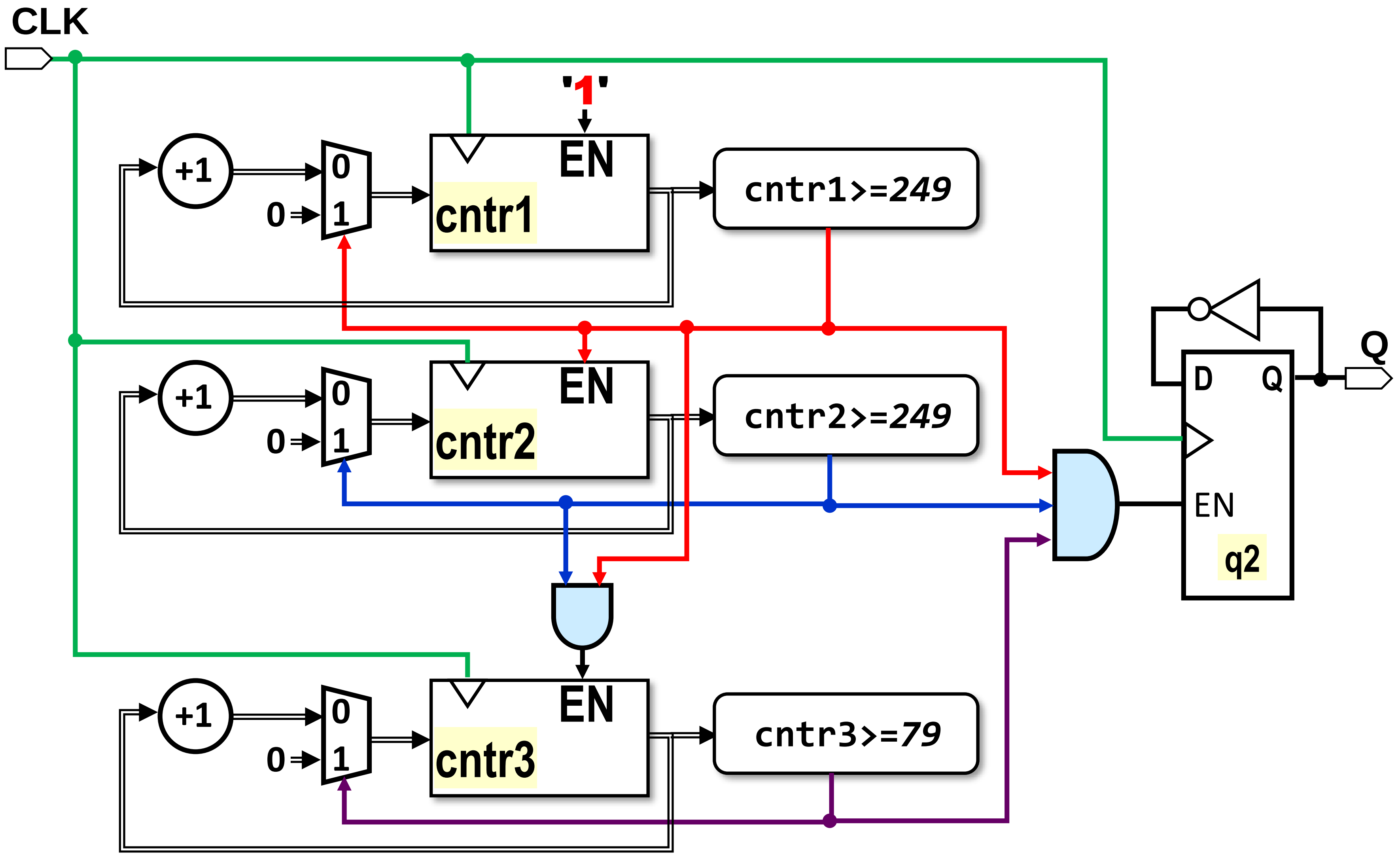
- On fast logic, i.e. all of today's, inputs with asynchronous behavior are used exclusively for initialization after power-up.
  On VEEK-MT2 boards, they are connected to the **KEY[0]** signal.

Proper Propagation Logic for Counters

Propagation Logic



High-frequency divider by 10M (1/4)



Explanation

- The block structure uses the EN (enable) inputs of the DFF flip-flop circuits. All internal counters receive the clock signal permanently, they only determine whether to respond to it or not.
- The first counter cntr1 is enabled permanently in '1', so it always counts. When it reaches 249, its comparator switches the input multiplexer so that after the next clock edge it loads '0', starting from the beginning. The output of its comparator is also sent to the enable input of the next counter that adds +1. It counts the number of times cntr1 has been reset.
- The last counter, cntr3, is incremented only when both upper counters are zeroed. The output flip-flop circuit q2 flips when all three counters are zeroed at once, which occurs after 250*250*80=5 million, so there will be a CLK divided by 10 million on Q.
- Note that all counters and comparators work in parallel and do not wait for their results. Their cascade has nearly the same complexity as one single divider. Much more important is the fact that the cascade handles almost 50 percent higher frequency on its CLK input.

High-frequency divider by 10M 2/4

```
library ieee; use ieee.std_logic_1164.all; use ieee.numeric_std.all;
entity DivBy10M is
  port (  CLK : in std_logic;
       Q : out std_logic);
end entity;
architecture rtlEdu of DivBy10M is
begin
  process(CLK)
  constant DivBy1: integer:=250; constant DivBy2: integer:=250;
  constant DivBy3: integer:=80;
  variable cntr1 : integer range 0 to DivBy1-1:=0;
  variable cntr2 : integer range 0 to DivBy2-1:=0;
  variable cntr3 : integer range 0 to DivBy3-1:=0;
  variable q2 : std_logic:='0';
  variable clear1, clear2, clear3 : boolean;
  begin --process
```

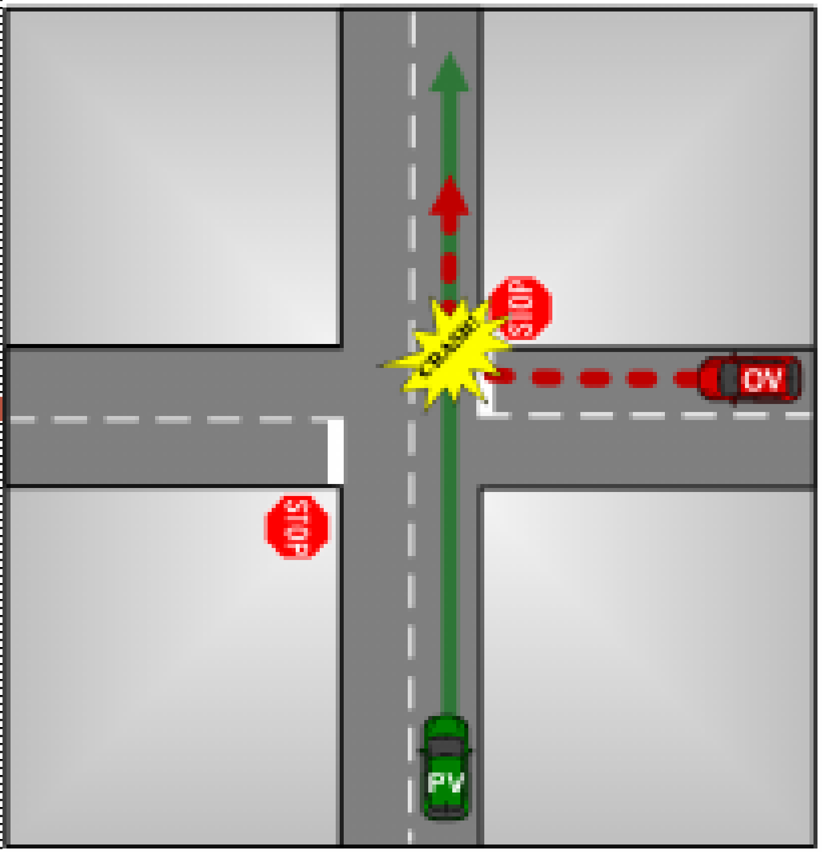
High-frequency divider by 10M 3/4

```
begin
      if rising_edge(CLK) then
            clear1:=cntr1>=DivBy1-1; clear2:=cntr2>=DivBy2-1; clear3:=cntr3>=DivBy3-1;
            if TRUE then  -- cntr1 is always enabled
                  if clear1 then cntr1:=0; else cntr1:=cntr1+1; end if;
            end if;
            if clear1 then  --now, cntr2 is enabled
                  if clear2 then cntr2:=0; else cntr2:=cntr2+1; end if;
            end if;
            if clear1 and clear2 then  -- now, cntr3 is enabled
                  if clear3 then cntr3:=0; else cntr3:=cntr3+1; end if;
            end if;
            if clear1 and clear2 and clear3 then  -- div2 is enabled
                  q2:=not q2;
            end if;
      end if;
      Q<=q2;
   end process; end architecture;
```

High-frequency divider by 10M 4/4

```
architecture rtlShort of DivBy10M is
begin
   process(CLK)
   constant DivBy1_2: integer:=250;
   constant DivBy3: integer:=80;
   variable cntr1, cntr2 : integer range 0 to DivBy1_2-1:=0;
   variable cntr3 : integer range 0 to DivBy3-1:=0;
   variable q2 : std_logic:='0';
   begin
         if rising_edge(CLK) then
               if cntr1<DivBy1_2-1 then cntr1:=cntr1+1;
               else cntr1:=0;
                     if cntr2<DivBy1_2-1 then cntr2:=cntr2+1;
                     else cntr2:=0;
                           if cntr3<DivBy3-1 then cntr3:=cntr3+1;
                           else  cntr3:=0; q2:=not q2;
                           end if;
                     end if;
               end if;
         end if;
         Q<=q2;
         end process; end architecture;
```

Not giving priority to asynchronous input



Scenario 1 (priority violation)
2 (stop violation)

Image: Stephanie Lefevre, Collision scenarios

Violated asynchronous input priority

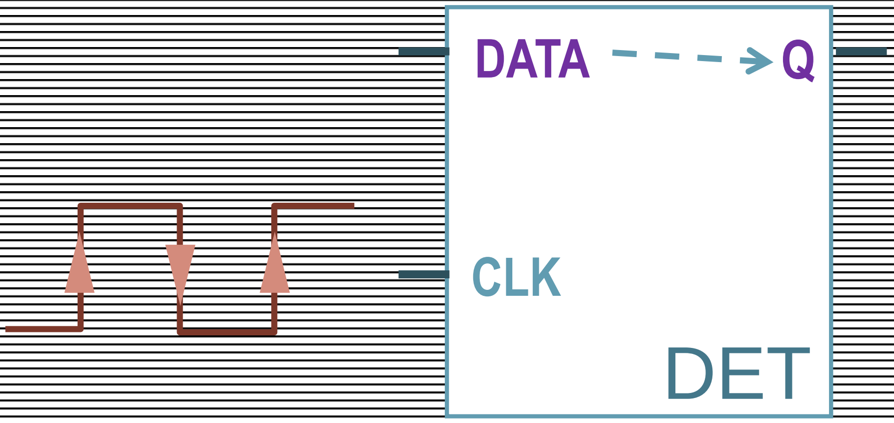
```
library ieee; use ieee.std_logic_1164.all;
 entity Nonsense is
  port (clock, aclrn : in std_logic;
         D : in std_logic;
         q : out std_logic);
  end;
architecture rtl of Nonsense is
begin process(clock, aclrn)
  begin
       if rising_edge(clock) then
             q <= D;
       elsif aclrn = '0' then
             q <='0';
       end if;
    end if;
   end process;
 end rtl;
```

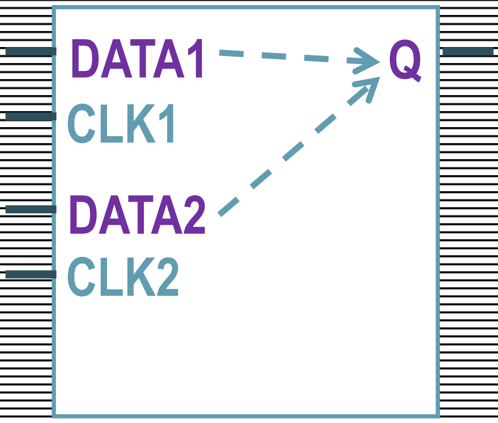
Asynchronous clear (aclrn)
always has a higher priority,
so it must appear in the code before the clock edge detection of the synchronous part - otherwise it cannot be implemented!

Error (10818): Can't infer register for "q" at Nonsense.vhd(14) because it does not hold its value outside the clock edge
Error (10822): HDL error at Nonsense.vhd (14): couldn't implement registers for assignments on this clock edge

Wrong Dependency on the Two Clocks

- DET or DETFF - Dual Edge Trigged Flip-Flop
- Double Clock D type Flip-Flop





Faulty Dependence on 2 Clocks

```
library ieee; use ieee.std_logic_1164.all;
 entity NotInOurFPGA is
  port (clock : in std_logic;
         D : in std_logic; q : out std_logic);
  end;
architecture rtl of NotInOurFPGA is
begin
process(clock)
 begin
   if rising_edge(clock) then
         q <= D;
   elsif falling_edge(clock) then
         q <= not D;
   end if;
end process; end architecture;
```

Our Cyclone does not include flip-flop circuits sensitive to multiple clock inputs.

- similar codes are not directly a bug; we just don't have them to implement.

Error: Netlist error at reg4.vhd(13): can't infer register for q because it changes value on both rising and falling edges of the clock

Error: HDL error at NotInOurFPGA .vhd(13): couldn't implement registers for assignments on this clock edge

Violated value in the next function

Impossible operation

```
library ieee; use ieee.std_logic_1164.all;
 entity DeltaWrong1 is
  port (clock : in std_logic;
         D, A : in std_logic; q : out std_logic);
  end;
architecture rtl of DeltaWrong1 is
begin
process(clock, A)
  begin
    q <= A;
    if rising_edge(clock) then
          q <= D;
    end if;
end process; end architecture;
```

Error (10818): Can't infer register for "q" at reg_DeltaWrong1.vhd(11) because it does not hold its value outside the clock edge

Incorrect value change in the output function

```
library ieee; use ieee.std_logic_1164.all;
 entity OmegaLatch is
  port (clock : in std_logic;
         D, A : in std_logic; q : out std_logic);
  end;
architecture rtl of OmegaLatch is
begin
process(clock, A)
  begin
    if rising_edge(clock) then
          q <= D;
    end if;
    if A='1' then q <= not D;
   end if;
end process; end architecture;
```

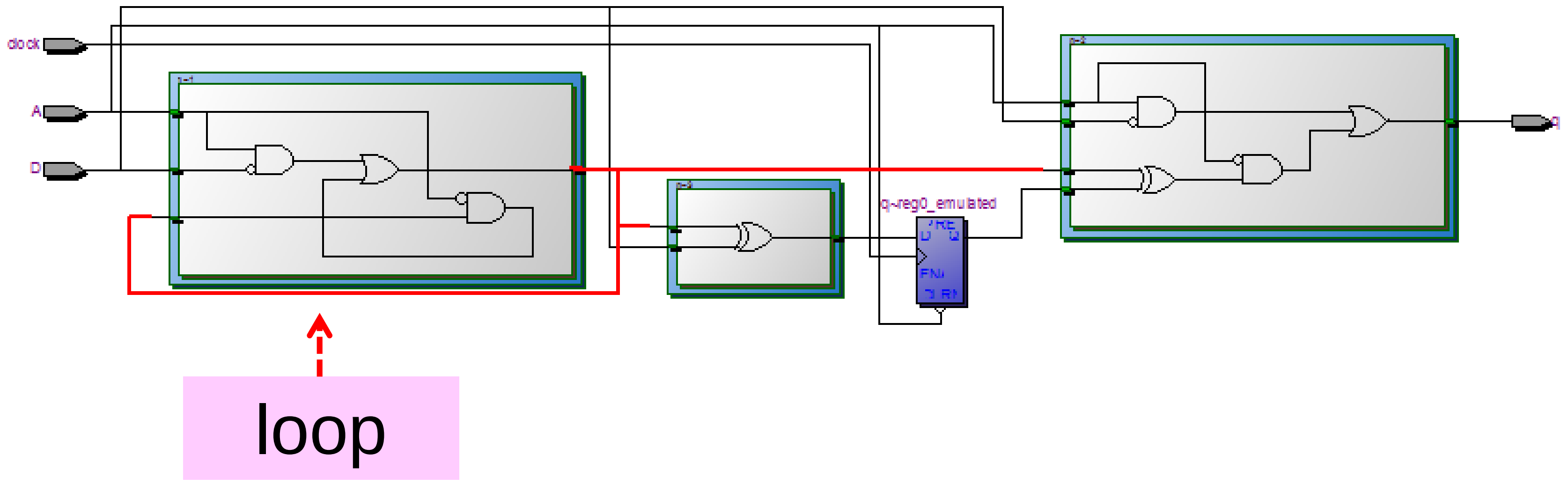
Warning (335093): TimeQuest Timing Analyzer is analyzing 1 combinational loops as latches.

Input A now belongs to the sensitivity list

Here we have the combination circuit, incomplete assignment creates Latch !

Changing the value in the next value function - Technology Map

```
process(clock, A)
  begin if rising_edge(clock) then q <= D; end if;
          if A='1' then q <= not D; end if;
end process;
```



Incorrect value change

```
library ieee; use ieee.std_logic_1164.all;
 entity OmegaLatch2 is
  port (clock : in std_logic;
         D, A : in std_logic; q : out std_logic);
  end;
architecture rtl of OmegaLatch2 is
begin
process(clock, A)
  variable x : std_logic;
  begin
    if rising_edge(clock) then x := D;
   end if;
    if A='1' then x := not D; end if;
    q<= x;
end process; end architecture;
```

Warning (335093): TimeQuest Timing Analyzer is analyzing 1 combinational loops as latches.

Entry A again belongs to the sensitivity list

The variable won't help. Latch again! We require conditional storing x value!!!

Incorrect value change in the output function

```
library ieee; use ieee.std_logic_1164.all;
 entity OmegaLatch2 is
  port (clock : in std_logic;
         D, A : in std_logic; q : out std_logic);
  end;
architecture rtl of OmegaLatch2 is
begin
process(clock, A)
  variable x : std_logic;
  begin
    if rising_edge(clock) then x := D;
   end if;
    if A='1' then x := not D; else x := x; end if;
    q<= x;
end process; end architecture;
```

Warning (335093): TimeQuest Timing Analyzer is analyzing 1 combinational loops as latches.

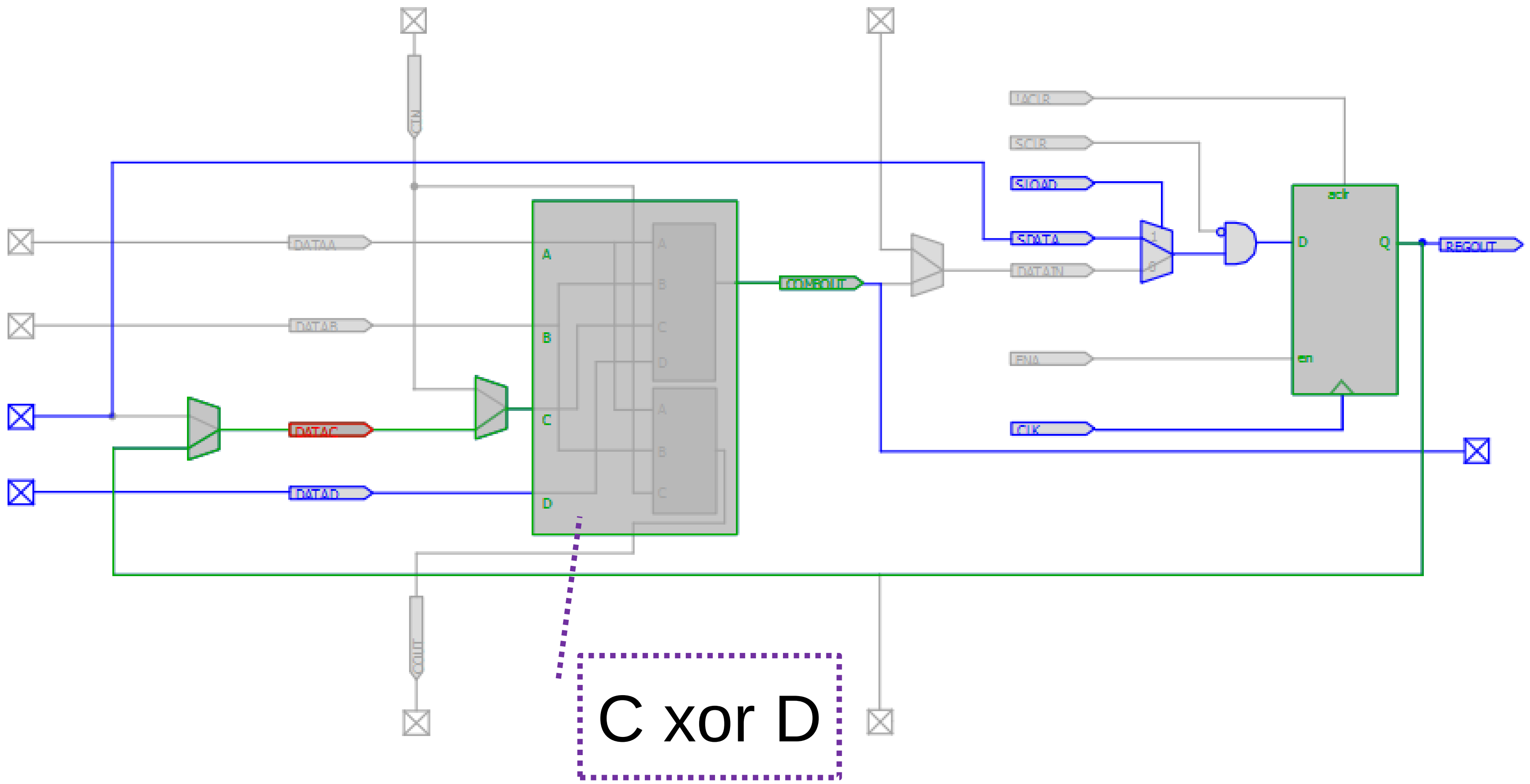
Entry A again belongs now to the sensitivity list

Latch again!
And why is that?

Correct now - we no longer change x

```
library ieee; use ieee.std_logic_1164.all;
 entity OmegaNot is
  port (clock : in std_logic;
         D, A : in std_logic; q : out std_logic);
  end;
architecture rtl of OmegaNot is
begin
process(clock, A)
  variable x : std_logic;
  begin
    if rising_edge(clock) then x := D;
   end if;
    if A='1' then q<= x; else q<= not x; end if;
  end process;
end architecture;
```

Resource Property Editor



Quartus has cleverly created an output function the only logical element, in which he used his fastest inputs C and D.

Suicide Structures

Jidai Geki, Harakiri (1962)

- **Non-constant asynchronous initialization**

Initialization to a Non-constant Value

```
port (clock, Init, D : in std_logic;
        q: out std_logic); ...
*******
begin process(clock, aclrn, Init)
begin if aclrn = '0' then

                q <= Init;

        elsif rising_edge(clock) then
                q <= D;
         end if;

end process;
```

Never perform asynchronous initialization to non-constant values if you have only DFF circuits with one asynchronous input. *Cyclone II and IV FPGAs have only flip-flop circuits with one asynchronous input, so it can only asynchronously initialize to one constant value.*

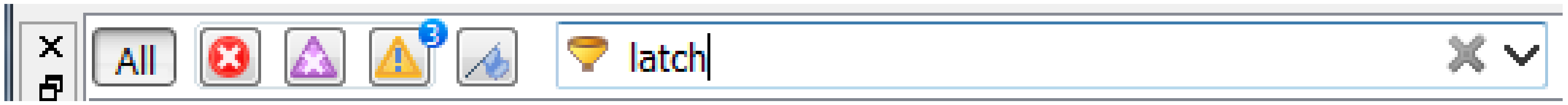
2 asynchronous settings = constant initialization

```
port(clock, D, aclrn, apren : in std_logic;
        q: out std_logic); ...
*******
begin process(clock, aclrn, apren)
begin
                if apren = '0' then q <= '1';
                elsif aclrn = '0' then q <= '0';

        elsif rising_edge(clock) then
                q <= D;
        end if;

end process;
```

Do not use two independent asynchronous influences on a synchronous variable.

DFF in our FPGA has only one asynchronous input.

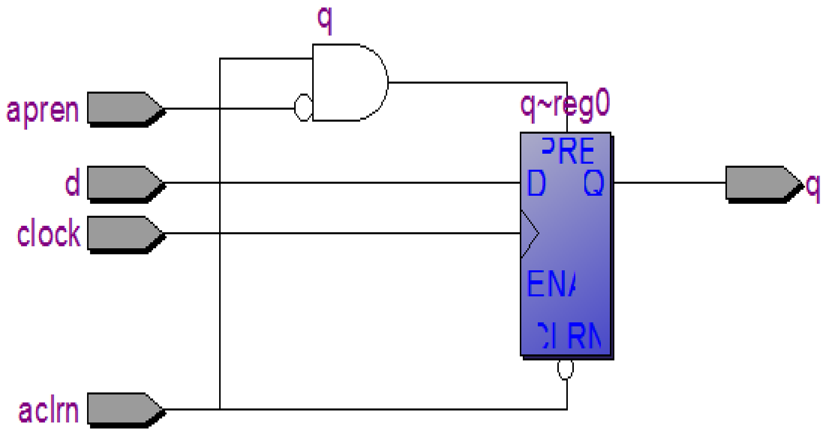
Compiler Warning



- Warning (13004): Presettable and clearable registers converted to equivalent circuits with latches. Registers power-up to an undefined state, and DEVCLRn places the registers in an undefined state.
  - Warning (13310): Register "q~reg0" is converted into an equivalent circuit using register "q~reg0_emulated" and latch "q~1"
- Warning (335093): TimeQuest Timing Analyzer is analyzing 1 combinational loops as latches.
- Warning (335093): TimeQuest Timing Analyzer is analyzing 1 combinational loops as latches.

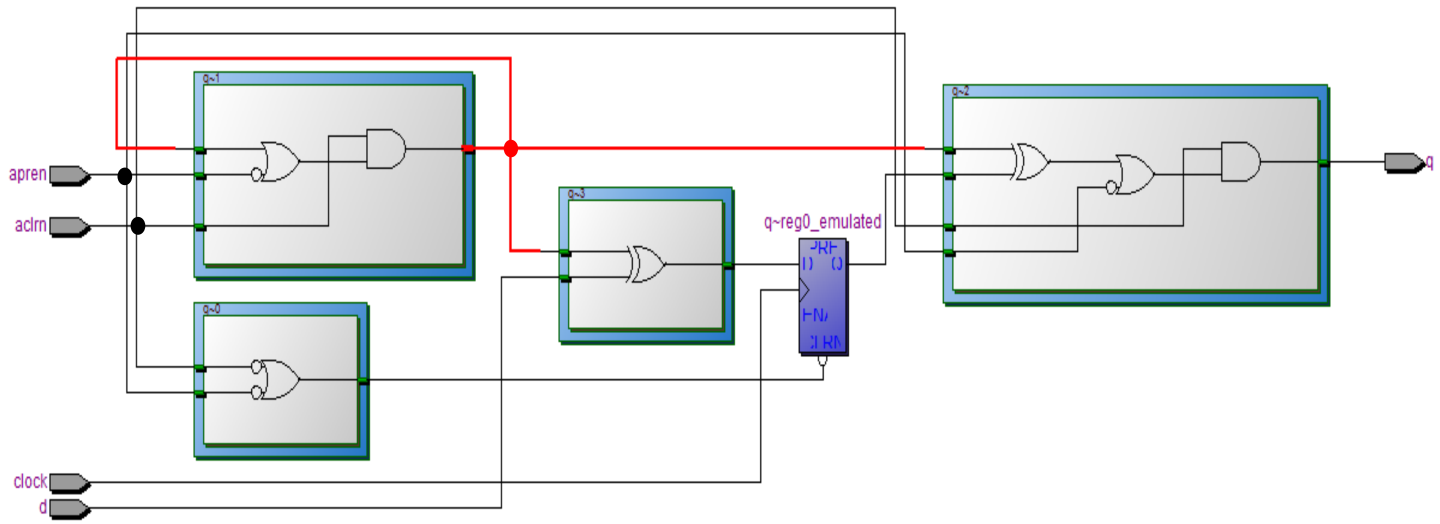
The structure appears only in the Technology Map

RTL Map

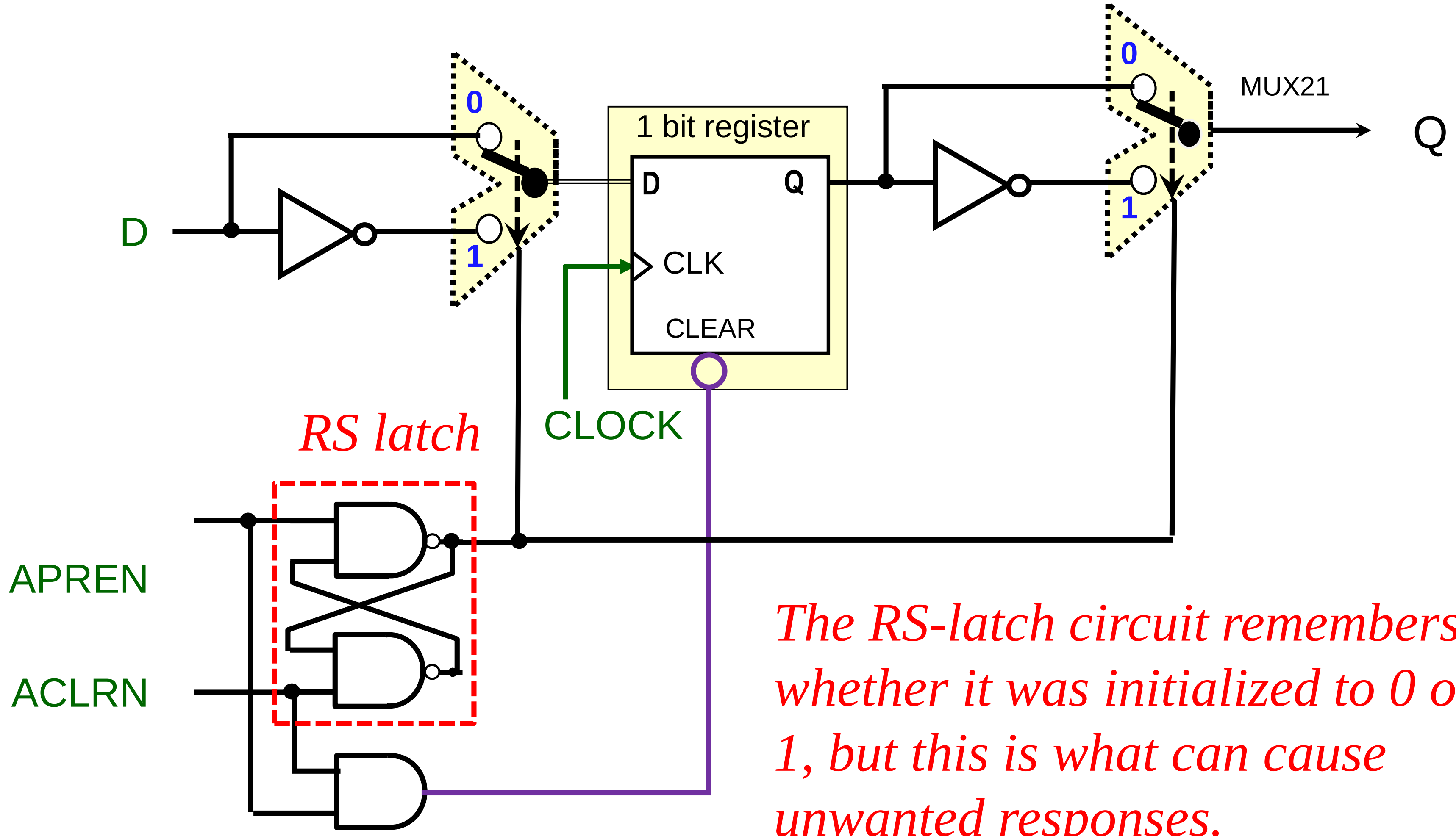


Technology Map

uartus found here that our FPGA does not have a DFF with two asynchronous inpu



Redrawn Implementation



The RS-latch circuit remembers whether it was initialized to 0 or 1, but this is what can cause unwanted responses.

Quartus implements, but it must use unreliable wiring.

Incomplete Assignment in Combinational Parts

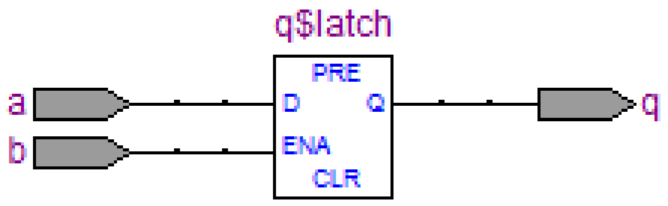
Quartus warning:

Combinational loops, latches

Incomplete assignment (1/2)

in the **combination** circuit

```
entity assignment1 is
port(a, b : in std_logic;
 q : out std_logic);
end;
architecture dataflow of assignment1 is
begin
    test : process(a, b)
    begin
     if b='1' then q<=a;
     end if;
    end process;

end architecture;
```



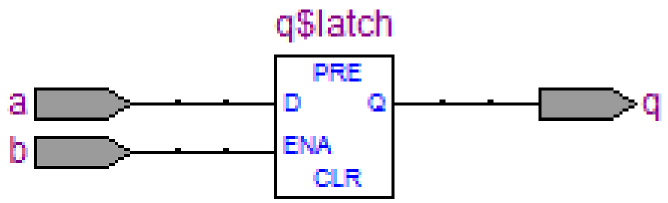
*The VHDL specification specifies that a signal that has not been changed retains its previous value, which can only be solved by inserting a latch memory element. Thus, **the combination circuit** must always define the value of the signal, not be generated.*

However, incomplete assignments are commonly used in sequential circuits,

which already contain memory elements and do not need to be inserted.

Incomplete Assignments(2/2)

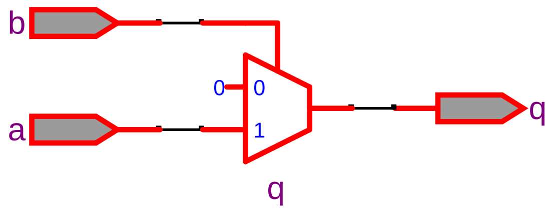
- Warning (10631): VHDL Process Statement warning at assignment1.vhd(13): inferring latch(es) for signal or variable "q", which holds its previous value in one or more paths through the process
- Warning (335093): TimeQuest Timing Analyzer is analyzing 1 combinational loops as latches.
- Warning (335093): TimeQuest Timing Analyzer is analyzing 1 combinational loops as latches.



*Always look for combinational "**loop**" and "**latch**" warnings in compiler messages.*

Correction

```
entity assignment1 is
port(a, b : in std_logic;
 q : out std_logic);
end;
architecture dataflow of assignment1 is
begin
    test : process(a, b)
    begin
     if b='1' then q<=a;
     else q<='0';
     end if;
    end process;

end architecture dataflow;
```



The if-then-else statement ***must*** *in the combinational circuit*
always specify a value on all possible branches
-> deterministic behavior.

Why points down, it works!!!

- *I will always look for your suggestions*

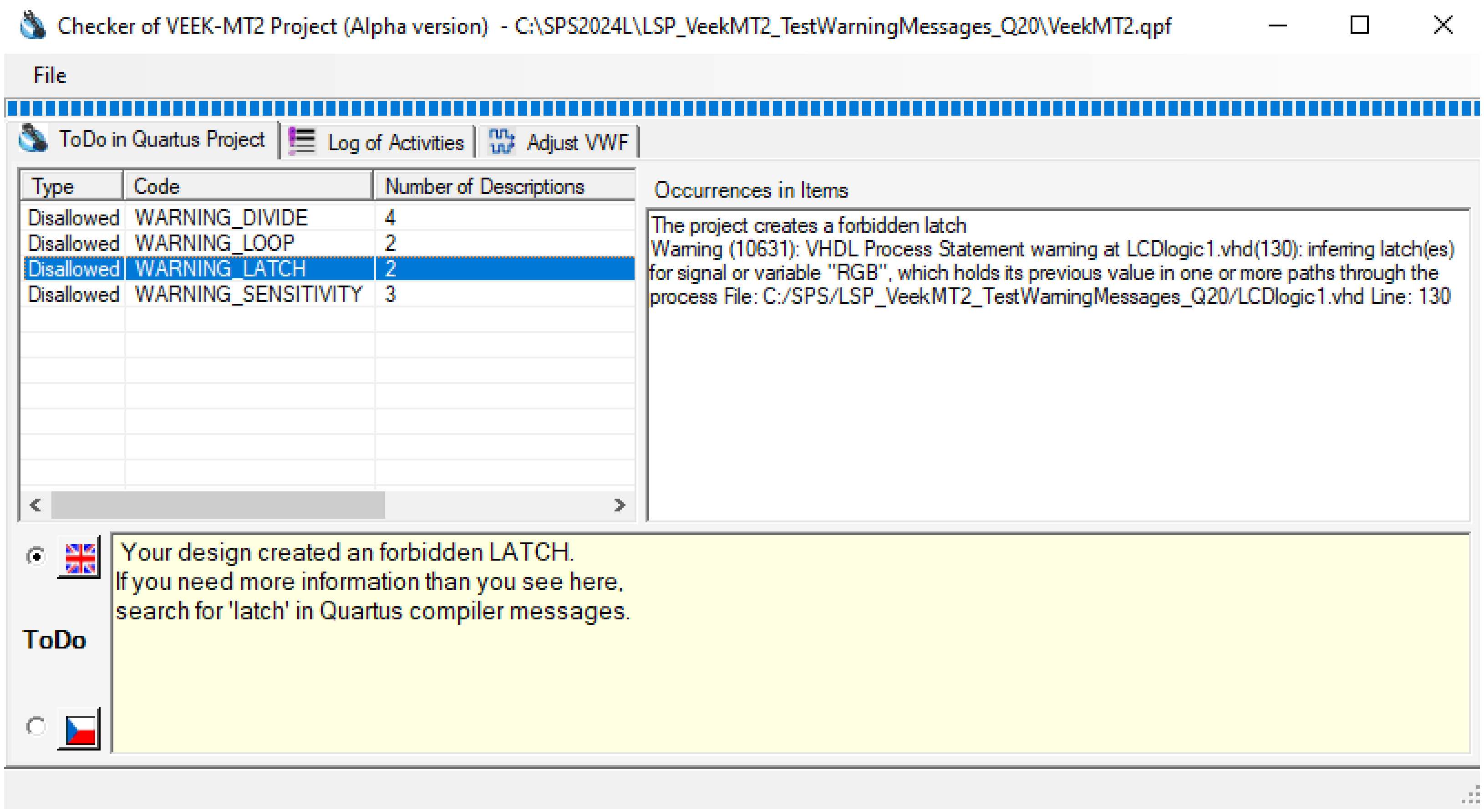
  "loop" and "latch",

  If your solution contains them,
  then it cannot be accepted :-(

[Image source wordpress.com]

- *Latch copies faults from input to output for the entire active clock period.*
- *In FPGAs it is not implemented by gates but by LUT tables, so it exhibits unexpected unwanted behavior similar to a hidden bomb - if it hasn't exploded yet, it doesn't mean it won't in the near future....*

Use Quartus Checker



Checker of VEEK-MT2 Project (Alpha version) - C:\SPS2024L\LSP_VeekMT2_TestWarningMessages_Q20\VeekMT2.qpf

Type	Code	Number of Descriptions
Disallowed	WARNING_DIVIDE	4
Disallowed	WARNING_LOOP	2
Disallowed	WARNING_LATCH	2
Disallowed	WARNING_SENSITIVITY	3

Occurrences in Items

```
The project creates a forbidden latch
Warning (10631): VHDL Process Statement warning at LCDlogic1.vhd(130): inferring latch(es)
for signal or variable "RGB", which holds its previous value in one or more paths through the
process File: C:/SPS/LSP_VeekMT2_TestWarningMessages_Q20/LCDlogic1.vhd Line: 130
```

ToDo

Your design created an forbidden LATCH.
If you need more information than you see here,
search for 'latch' in Quartus compiler messages.