The Abdus Salam International Centre for Theoretical Physics

2384-14

ICTP Latin-American Advanced Course on FPGADesign for Scientific Instrumentation

19 November - 7 December, 2012

Selected Topics on Logic Synthesis and FPGA Design

CICUTTIN Andres

ICTP Multidisciplinary Laboratory
Via Beirut 31
(34100) Trieste
ITALY

The Abdus Salam
International Centre
for Theoretical Physics

Selected Topics on

Logic Synthesis and FPGA Design

Andres Cicuttin

ICTP Multidisciplinary Laboratory

Advanced Training Course on
FPGA Design for Scientific Instrumentation

Havana, Cuba
Nov 19 – Dec 7, 2012

Part A. Logic Synthesis

- VHDL Coding Style for Synthesis
- Pipeline inference
- Resource sharing. The Area-Speed tradeoff
- Primitives and Macros
- Multiple driving. Buses and Multiplexers

Part B. FPGA Design

- Synchronous Design
- Unavoidable and Avoidable Asynchronous Designs
- Special Asynchronous Circuits
- Debugging Techniques
- Common Mistakes and Good Design Practices

Some References (used in this presentation)

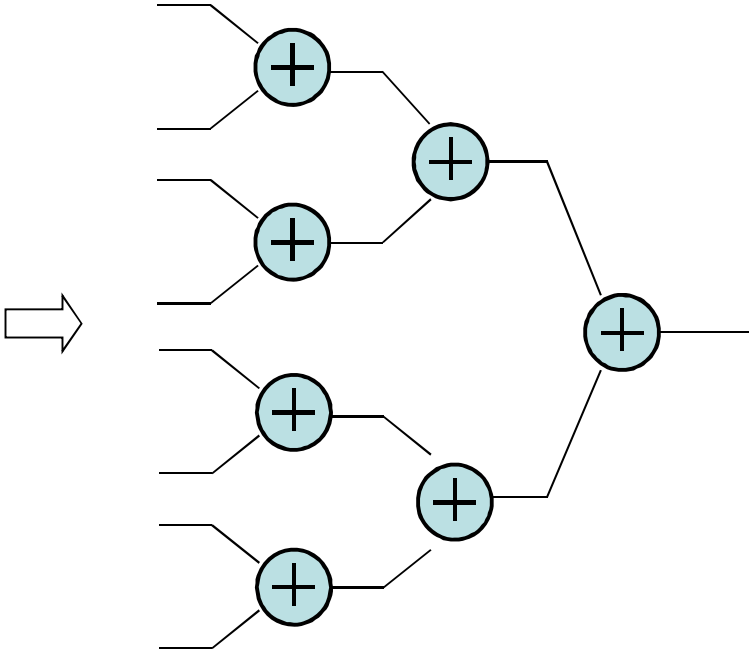
- Books on VHDL and FPGA design:
  - **Design Recipes for FPGAs**,
    Peter Wilson (*Elsevier, 2007*)
  - **RTL Hardware Design Using VHDL**,
    Pong P. Chu (*John Wiley & Sons, 2006*)
  - **The Design Warrior's guide to FPGAs**,
    Cleve "Max" Maxfield (*Elsevier, 2004*)
  - **Digital Signal Processing with Field Programmable Gate Arrays**,
    U. Meyer-Baese (*Springer, 2004*)
  - **Real Chip Design and Verification**,
    Ben Cohen, (*VhdlCohen Publishing, 2002*)
- Web sites:
  - www.opencores.org, http://asics.ws,
  - www.fpga4fun.com, www.us.design-reuse.com,
  - www.fpga-faq.org, http://www.andraka.com/papers.htm, …
  - Websites of FPGA companies (Actel, Altera, Xilinx,…)

VHDL for Synthesis

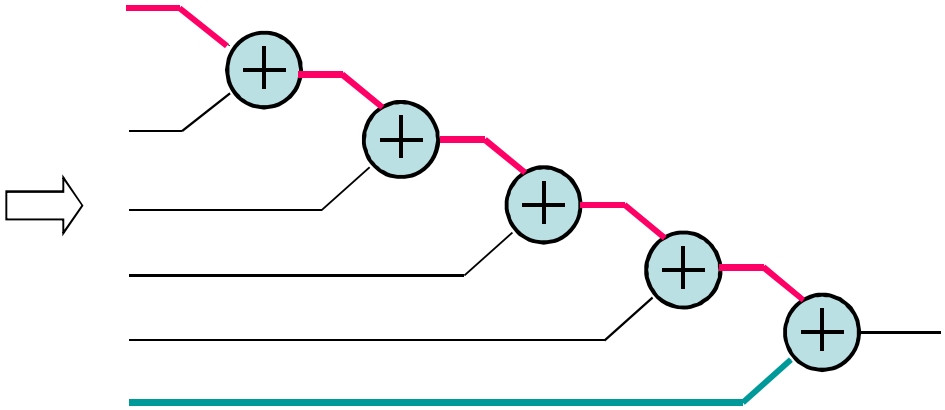
- Behavioral vs. Structural
- Coding Style and Synthesis
- Primitives and Macros
- Multiple Driving
  - Buses and Multiplexers
  - Potential conflicts (standard logic)
- The Area-Speed tradeoff

Different expression arrangement could determine different structures

```
result <= ((a+b)+(c+d))+((e+f)+(g+h)) ;
```



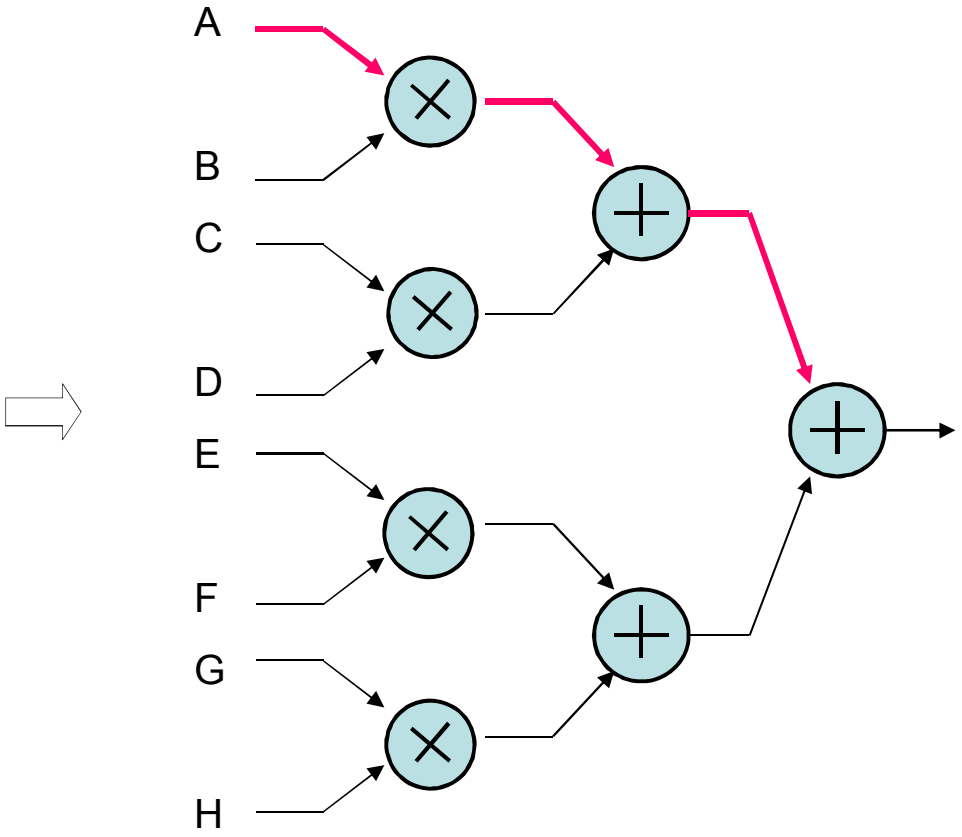
```
result <= (((((a+b)+c)+d)+e)+f) ;
```



Parallel implementation of

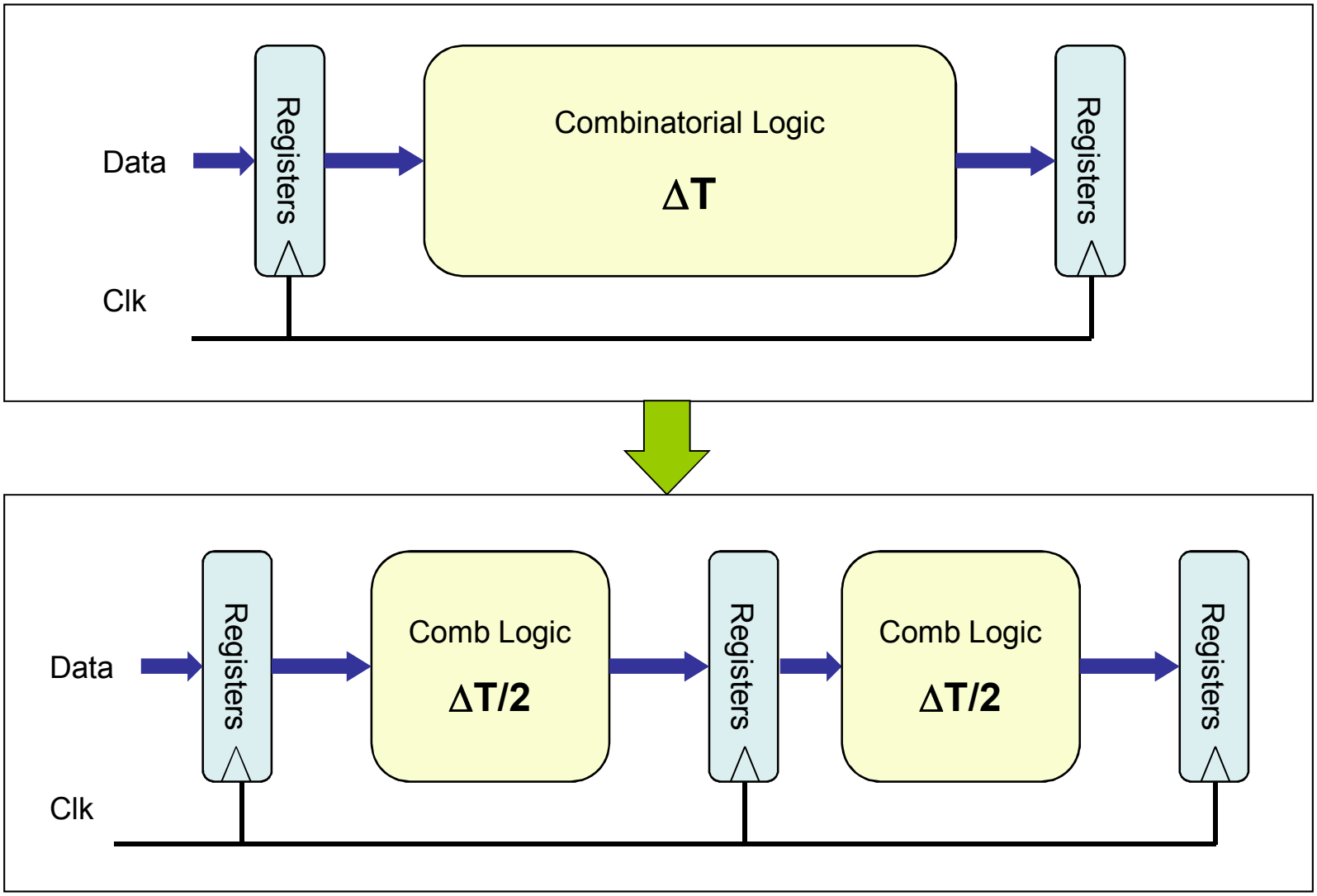
ABCDEFGH <= (a*b)+(c*d)+(e*f)+(g*h) after 10ns;

```
--Concurrent statements

AB   <= A * B ;
CD   <= C * D ;
EF   <= E * F ;
GH   <= G * H ;

ABCD <= AB + CD ;
EFGH <= EF + GH ;

ABCDEFGH <= ABCD + EFGH ;
```



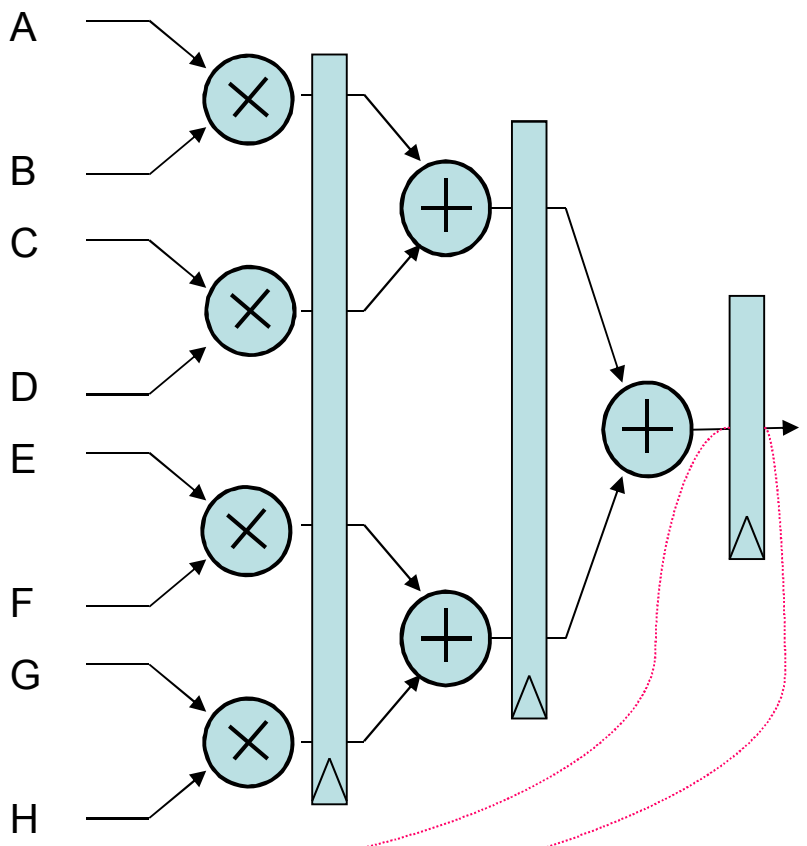
How to obtain more performance?

The Synchronous Pipeline Concept



Pipeline implementation of (a*b)+(c*d)+(e*f)+(g*h)

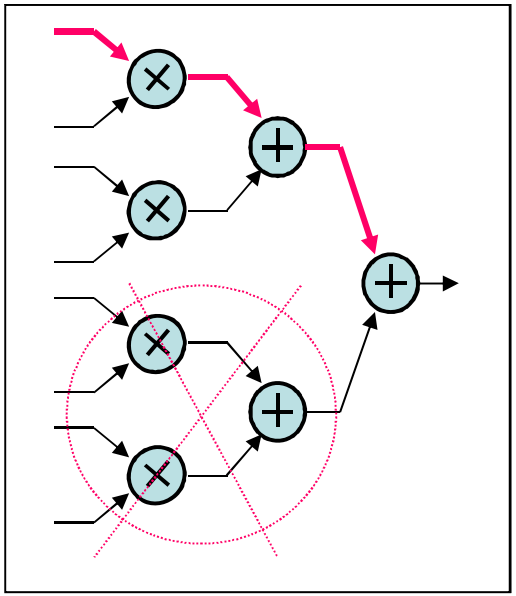
```
process (<clock>)
begin
 if <clock>'event and <clock>='1'
    then
        AB   <= A * B ;
        CD   <= C * D ;
        EF   <= E * F ;
        GH   <= G * H ;

        ABCD <= AB + CD ;
        EFGH <= EF + GH ;

        ABCDEFGH <= ABCD + EFGH ;
end if;
end process;
```

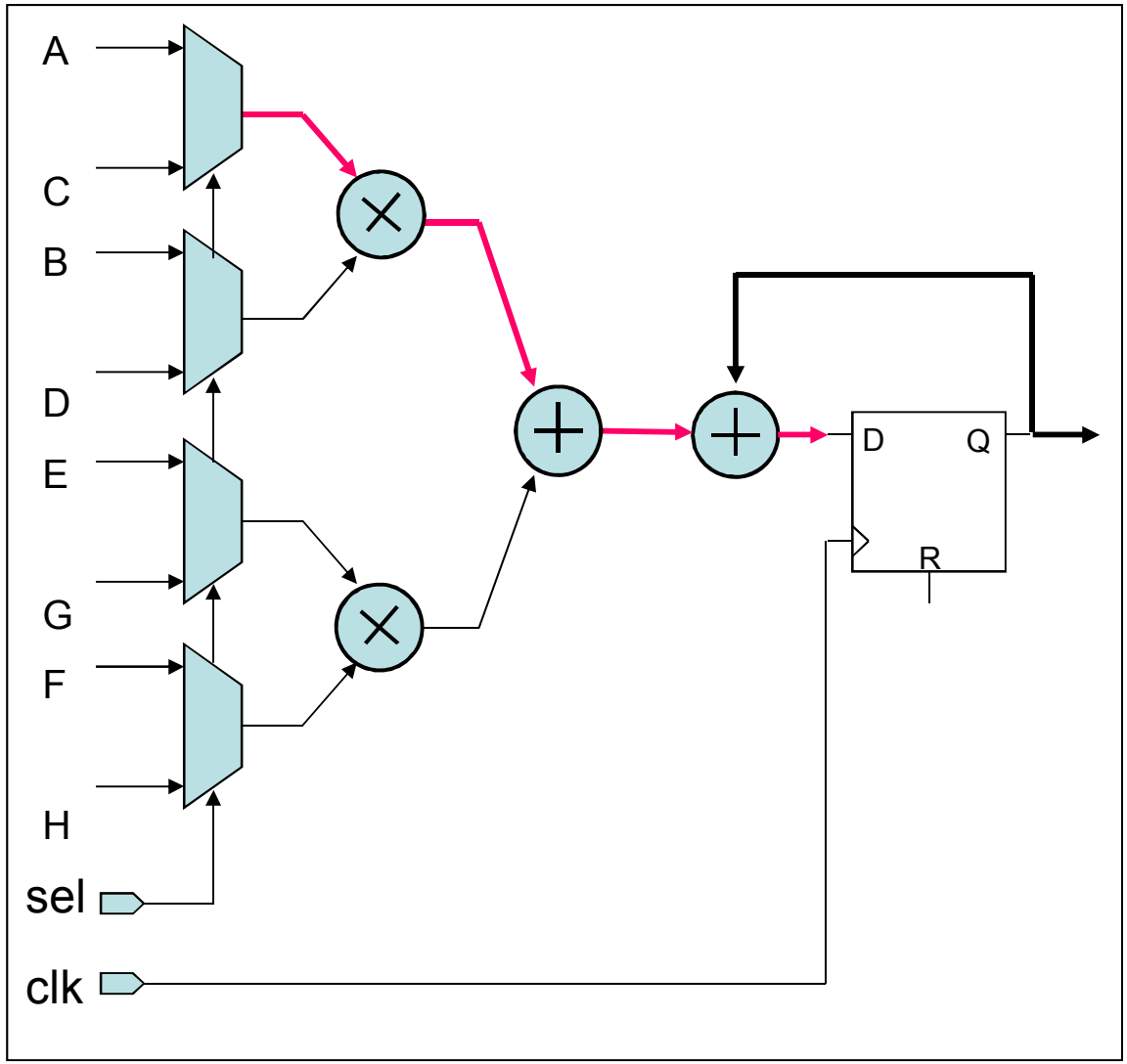


1) what about synchronizing the Input signals?

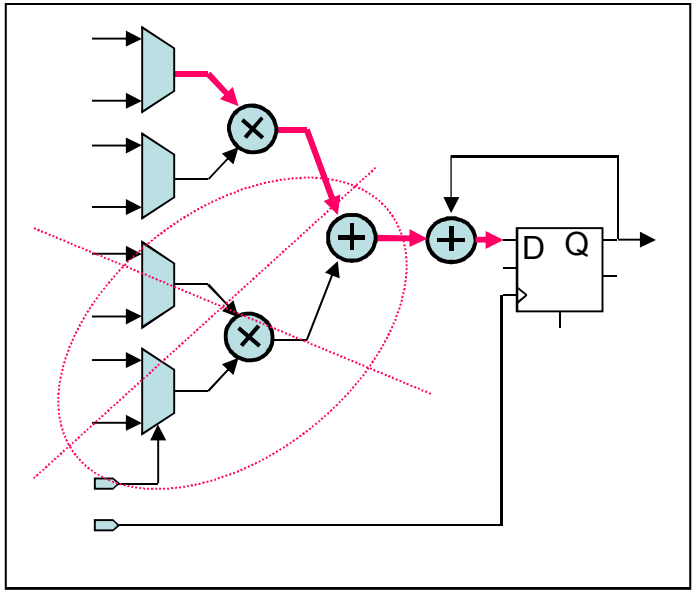
2) Can we use less resources for the same function?

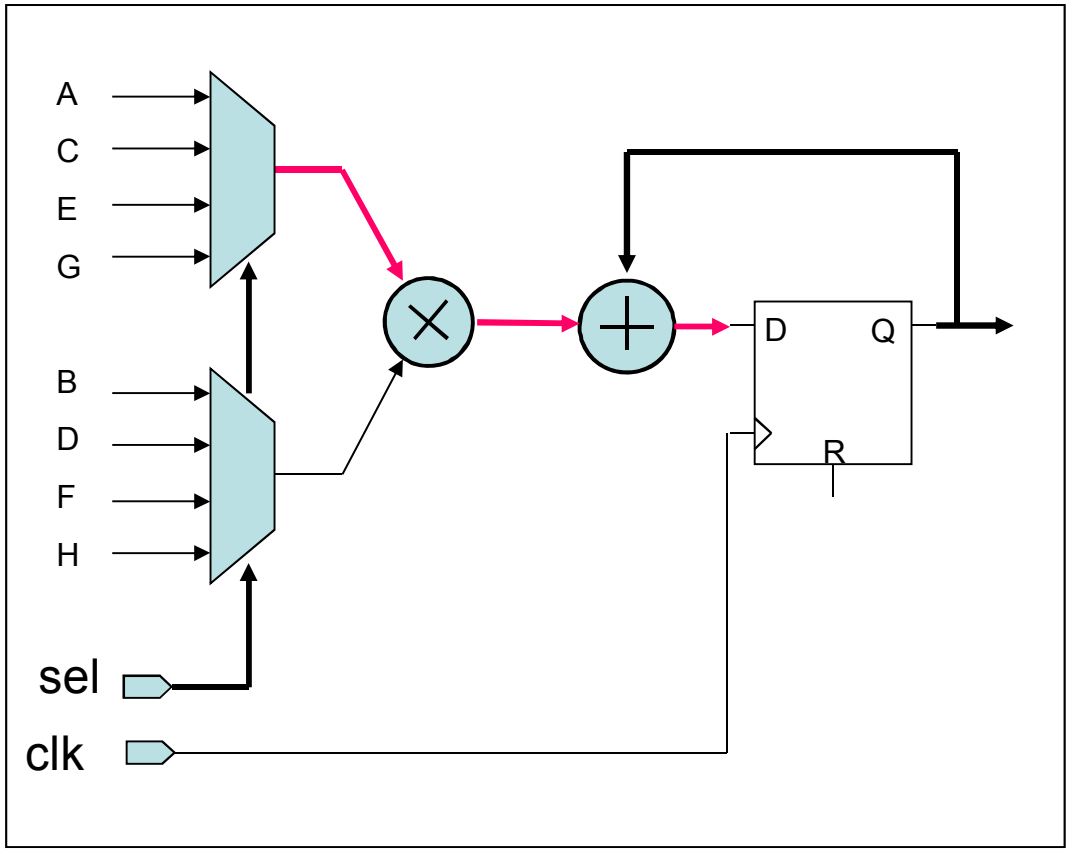
Resource Sharing





More Resource Sharing



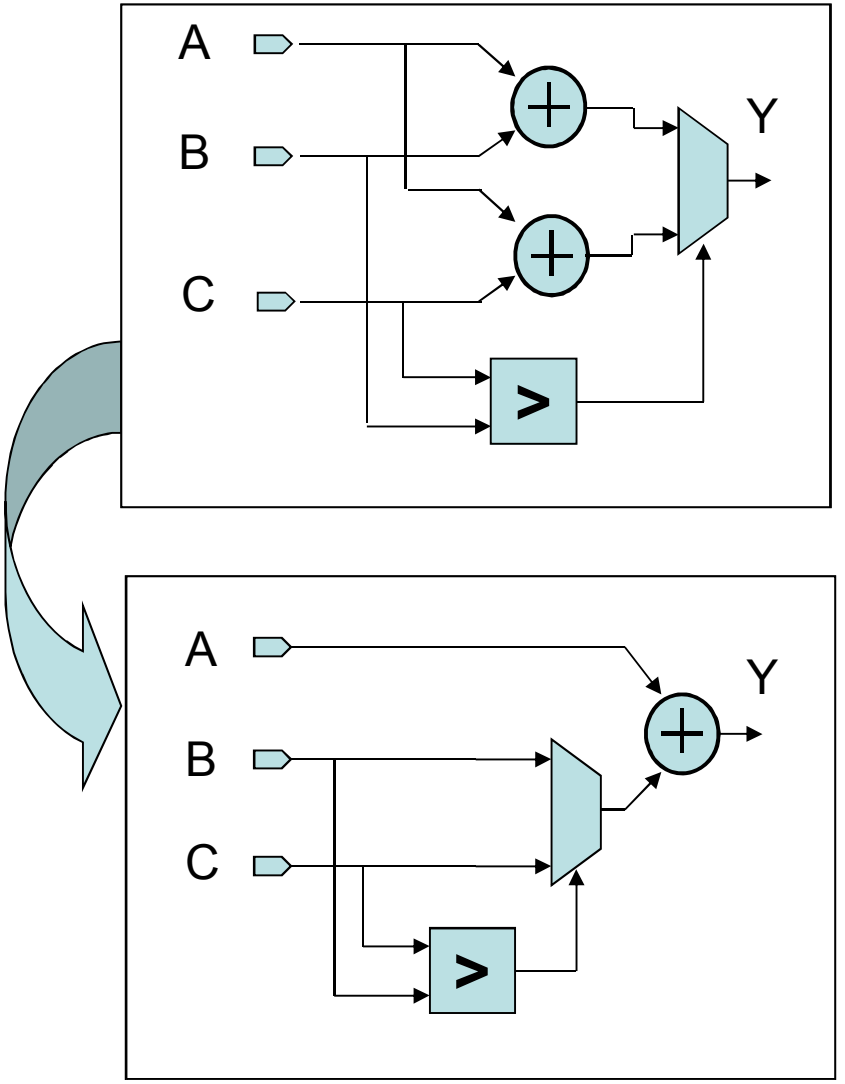


* VLSI <-> FPGA (use it or loose it!)
* Architectural decision

Automatic Resource Sharing

Conditional assignment

```
if (B > C)
  then  Y <= A + B;
  else  Y <= A + C;
end if;
```



Logic Synthesis Option

Resource Sharing → OFF / ON

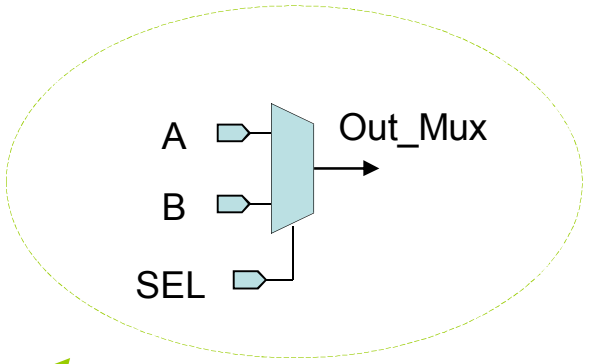
* VLSI <-> FPGA (use it or loose it!)
* Synthesis Tool decision

Primitives and Macros

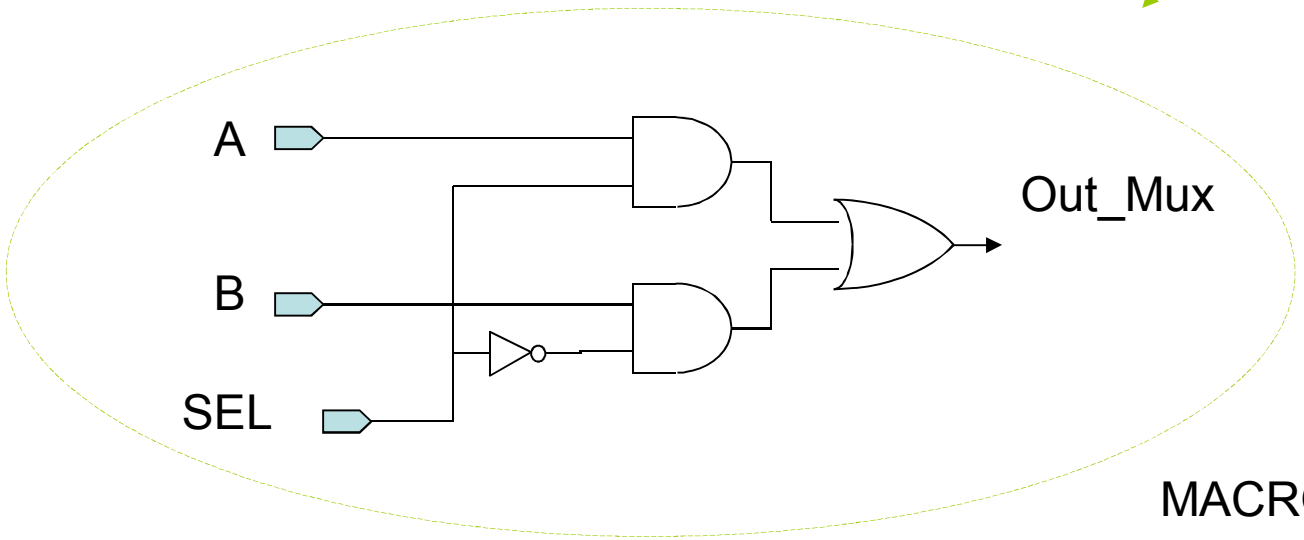
Primitives	Macros
Dedicated hardware	**Software implemented**
Memories	Adders
Multipliers	Multipliers
DLL, PLLs	Multiplexers
microprocessors	microprocessors

Buses and Multiplexers: *a simple 2_to_1 MUX*

```
--conditional assignment

Out_Mux <= A when SEL = '1' else B;
```



PRIMITIVE



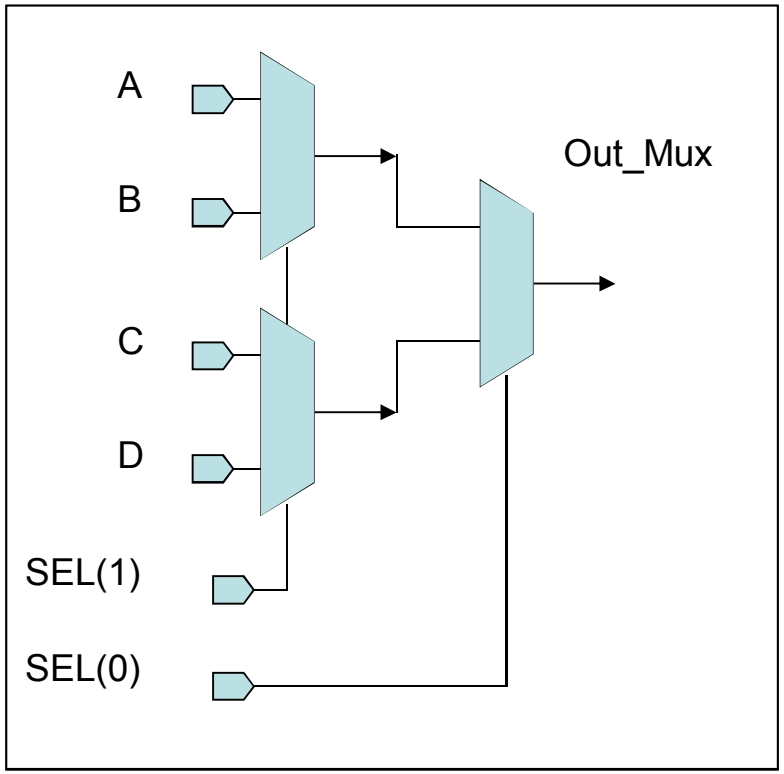
MACRO

Buses and Multiplexers: *a 4_to_1 MUX*

Behavioral VHDL

```
process (SEL, A, B, C, D)
begin
  case SEL is
    when "00" => Out_Mux <= A;
    when "01" => Out_Mux <= B;
    when "10" => Out_Mux <= C;
    when "11" => Out_Mux <= D;
    when others => Out_Mux <= A;
--(?)
  end case;
end process;
```

Schematic
(equivalent to structural VHDL)



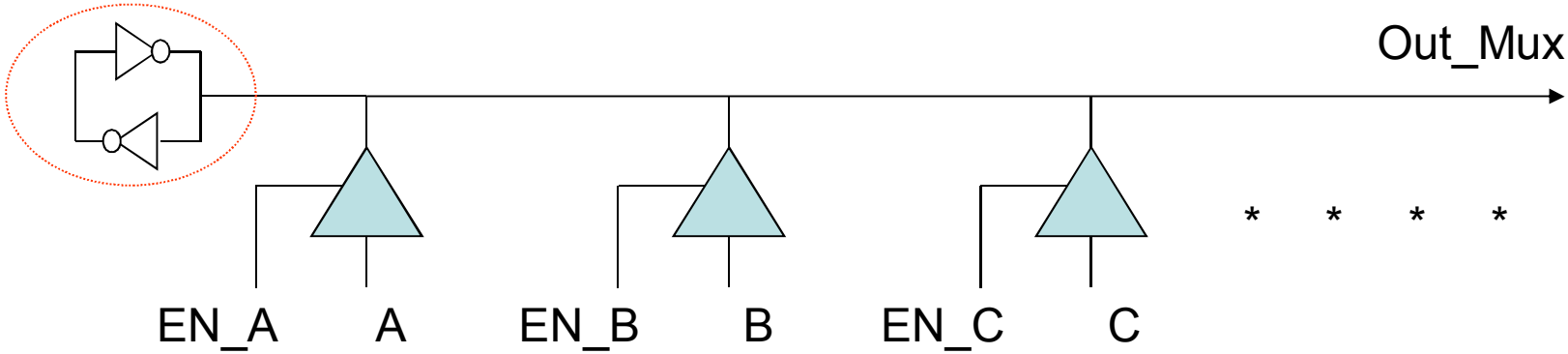
Buses and Multiplexers: *a many inputs MUX*

```
Out_Mux <= A when EN_A = '1' else 'Z';

Out_Mux <= B when EN_B = '1' else 'Z';

Out_Mux <= C when EN_C = '1' else 'Z';

                    *
                    *
                    *
```

Be careful !!!

(potential shorts

and floating lines)



Who takes care of the mutual exclusion of EN_X ?

Part B. FPGA Design

- The synchronous design paradigm
- Unavoidable and Avoidable asynchronous designs
- Special asynchronous circuits
- Debugging Techniques
- Common mistakes
- Good design practices

The Ideal Synchronous Design

- The timing of the whole design is referred to a single free running clock

(*or multiple clocks from a common source with perfectly controlled inter-clock phase*)

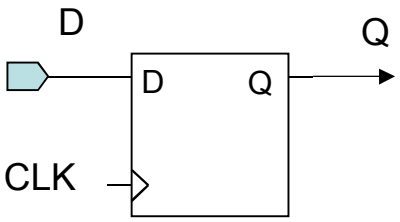
The synchronous design paradigm

	Synchronous	Asynchronous
Debugging	Easier	Very difficult
Predictability	Deterministic	Non deterministic
Interface with a sync. environment	Naturally interfaced	Requires special circuitry
Power:	Variable depending on architecture	Probably lower (?)
Speed	Essentially given by the clock frequency and architecture	Higher (?). Closer to the maximum for a given activity
Area	(?) Depends on architecture	(?) Depends on architecture
Design Time	Shorter time. A proved methodology exists. Mainly based on critical paths.	Longer time. It requires more physical information and detailed analog simulations for accurate delays determination
Reliability	Robust	Must be extensively checked

Unavoidable asynchronous designs

- Metastability
- Debouncing
- Multiple (unrelated) clock domains
  - Resynchronization of signals
  - Updating flags
  - Gray and Johnson codes

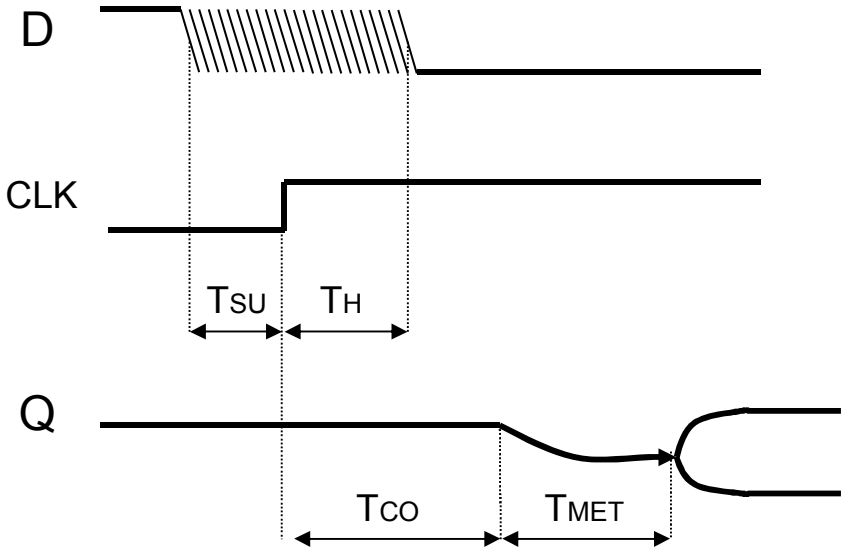
Metastability



$$MTBF = \frac{e^{(C_2 \times t_{MET})}}{C_1 \times f_{CLOCK} \times f_{DATA}}$$

C1: represents metastability-catching setup time windows (likelihood of going metastable)

C2: is an indication of the gain-bandwith product of the master lacth in the flip-flop



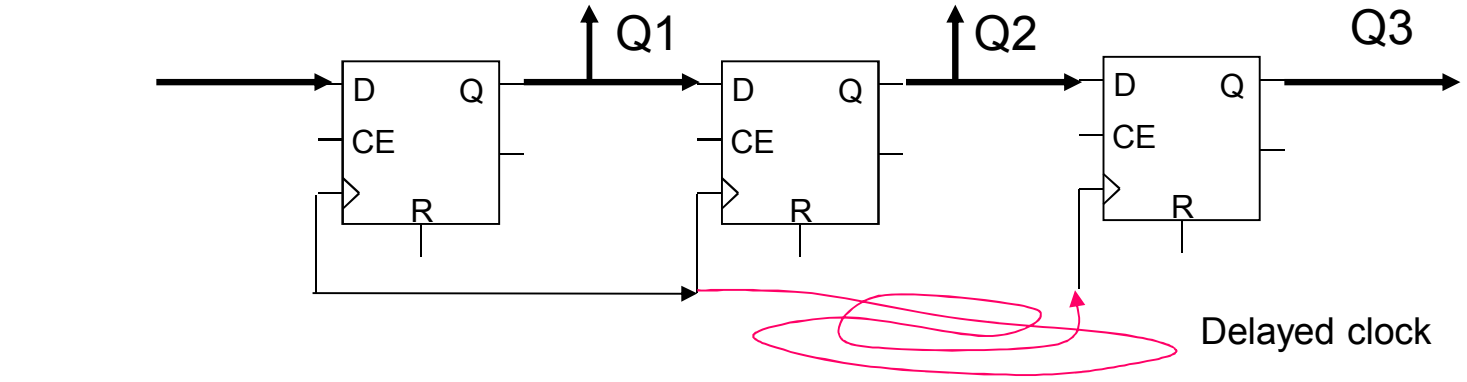
T_{SU}: set up time

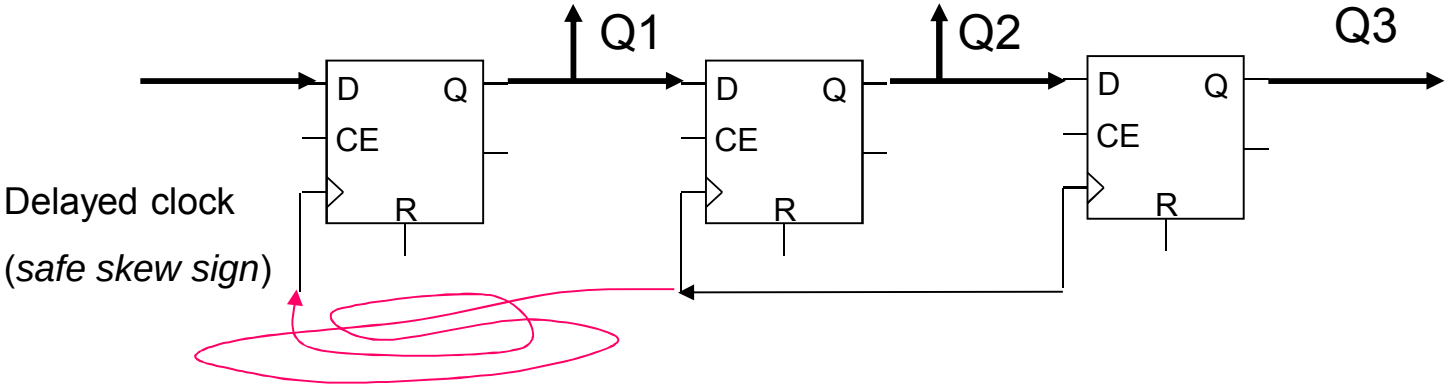
T_H: hold time

T_{CO}: clock to output delay

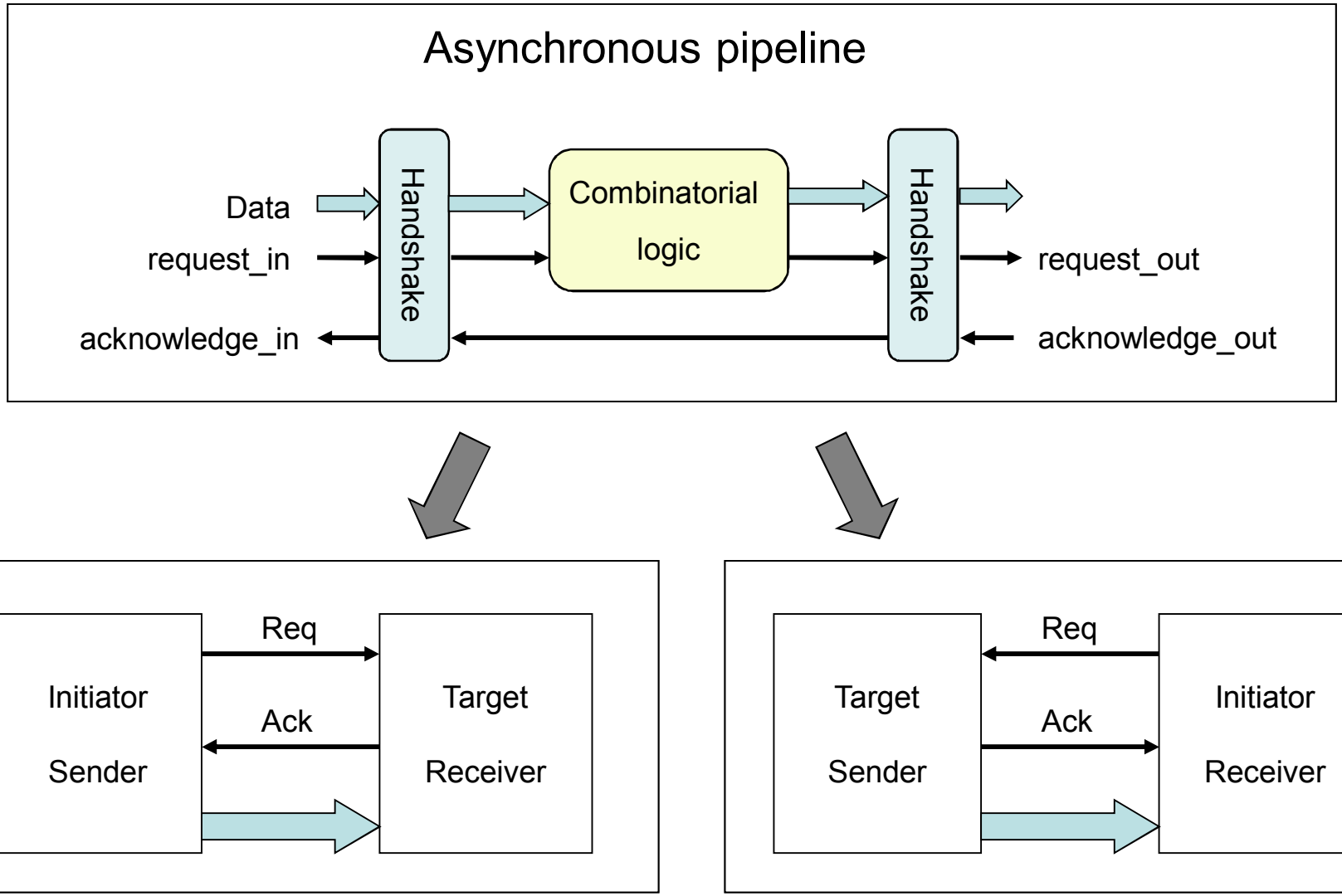
T_{met}: settling time

The clock skew problem

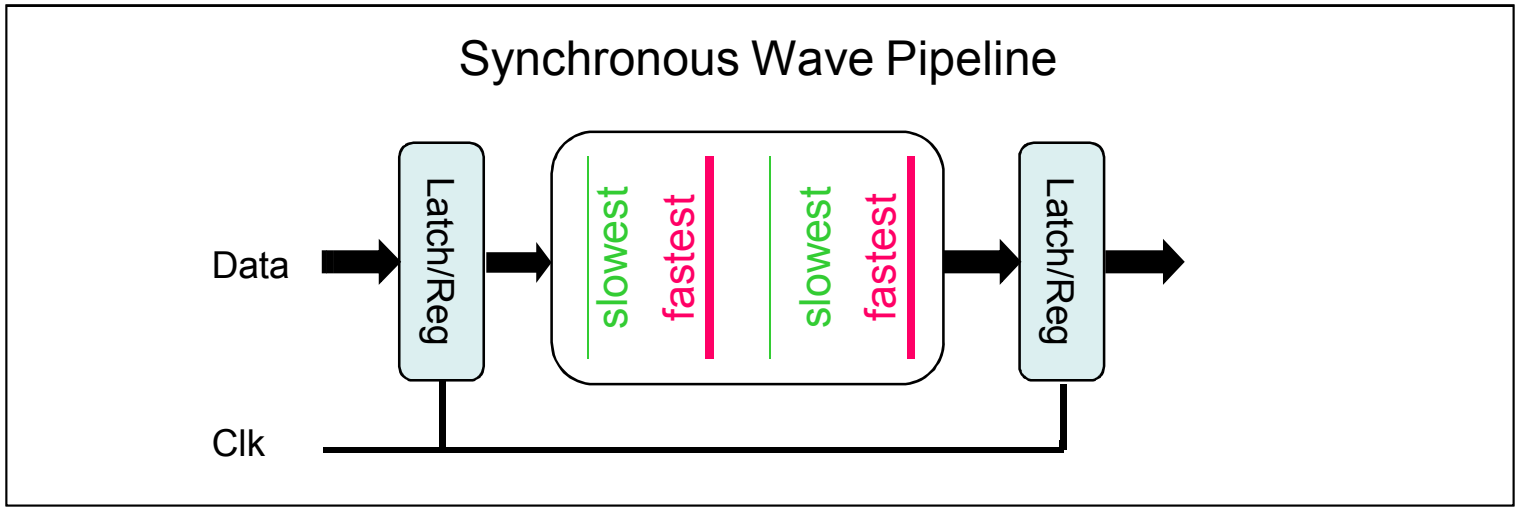


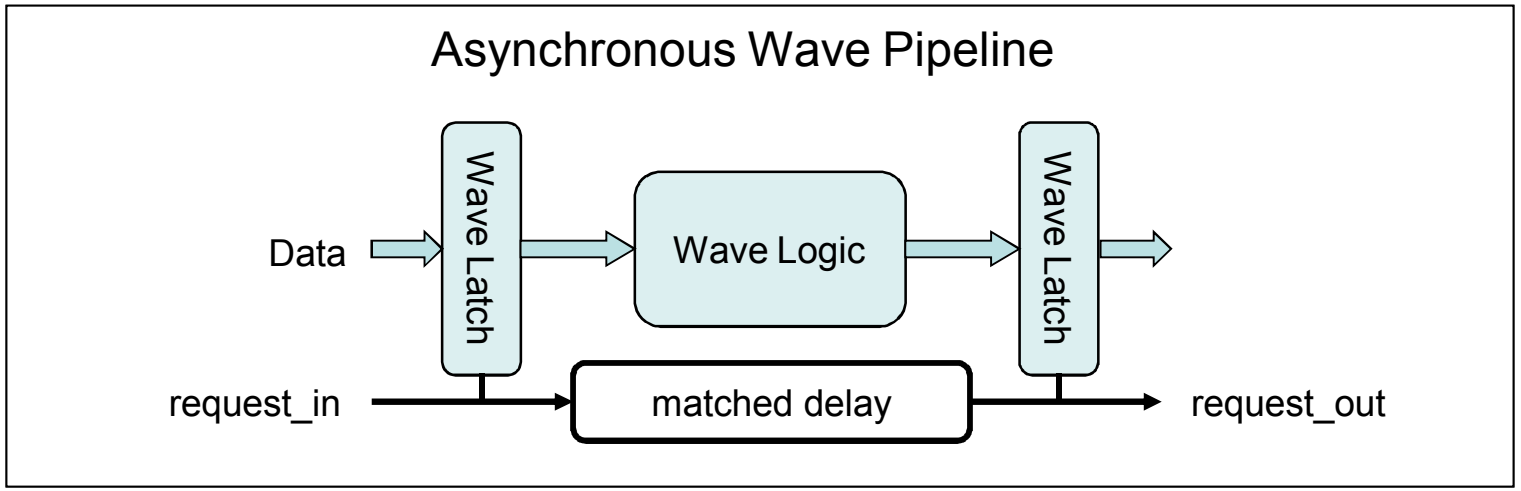


Examples of **avoidable** asynchronous circuits



More examples of **avoidable** asynchronous circuits



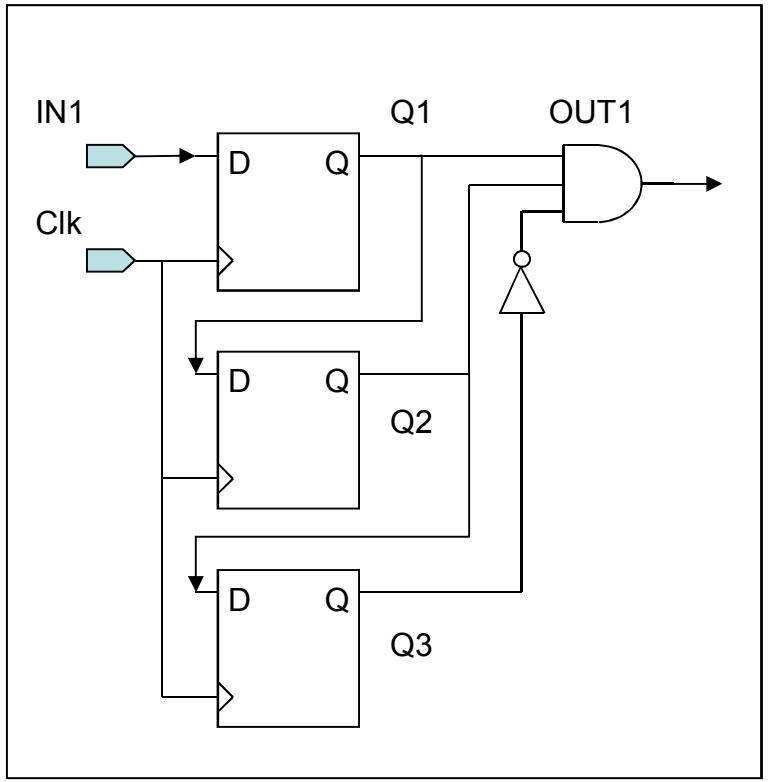


Common examples of **unavoidable** asynchronous circuits

Context	Examples	useful circuits
External inputs	Push buttons, Switches, Interrupts, etc	**I.** Debouncing **II.** Stabilizer
Microprocessor interface	Co-processing with DSP, Microc., GPP	**III.** Flancter
Multiple clock domains	Glue logic in complex systems	**IV.** Clock switching **V.** Gray codes

I. Debouncing (For *edge* sensitive signals)

```
-- Single shot pulse generator
Process (clk, reset)
begin
 if (reset = '1') then
    Q1 <= '0';
    Q2 <= '0';
    Q3 <= '0';
 elsif (clk'event and clk = '1') then
    Q1 <= IN1;
    Q2 <= Q1;
    Q3 <= Q2;
 end if;
end process;

OUT1 <= Q1 and Q2 and (not Q3);
```

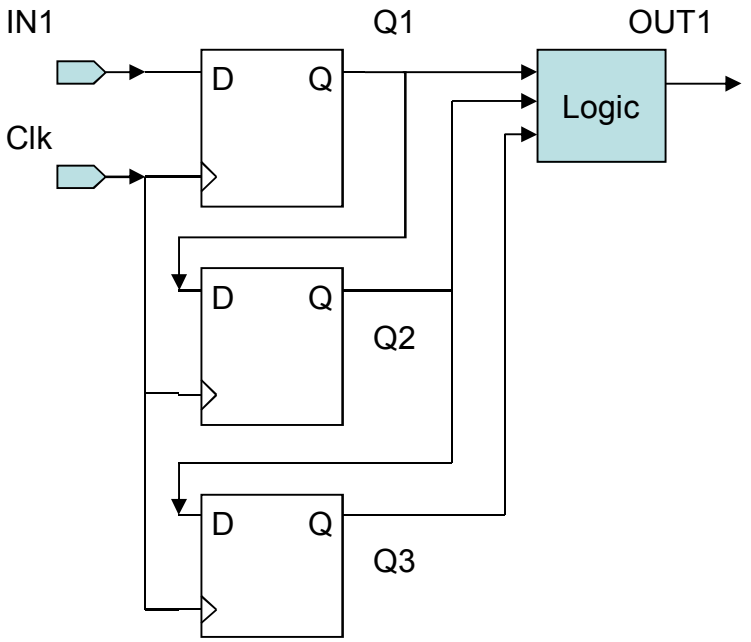


What about glitches ?

What happens if the ***out1*** assignment is done inside the process?

II. Stabilizer (For *level* sensitive signals)

```
Process (clk, reset)
begin
 if (reset = '1') then
    Q1 <= '0';
    Q2 <= '0';
    Q3 <= '0';
 elsif (clk'event and clk = '1') then
    Q1 <= IN1;
    Q2 <= Q1;
    Q3 <= Q2;
 end if;

end process;

OUT1 <= Q1 when ((Q1=Q2) and (Q2=Q3));
```

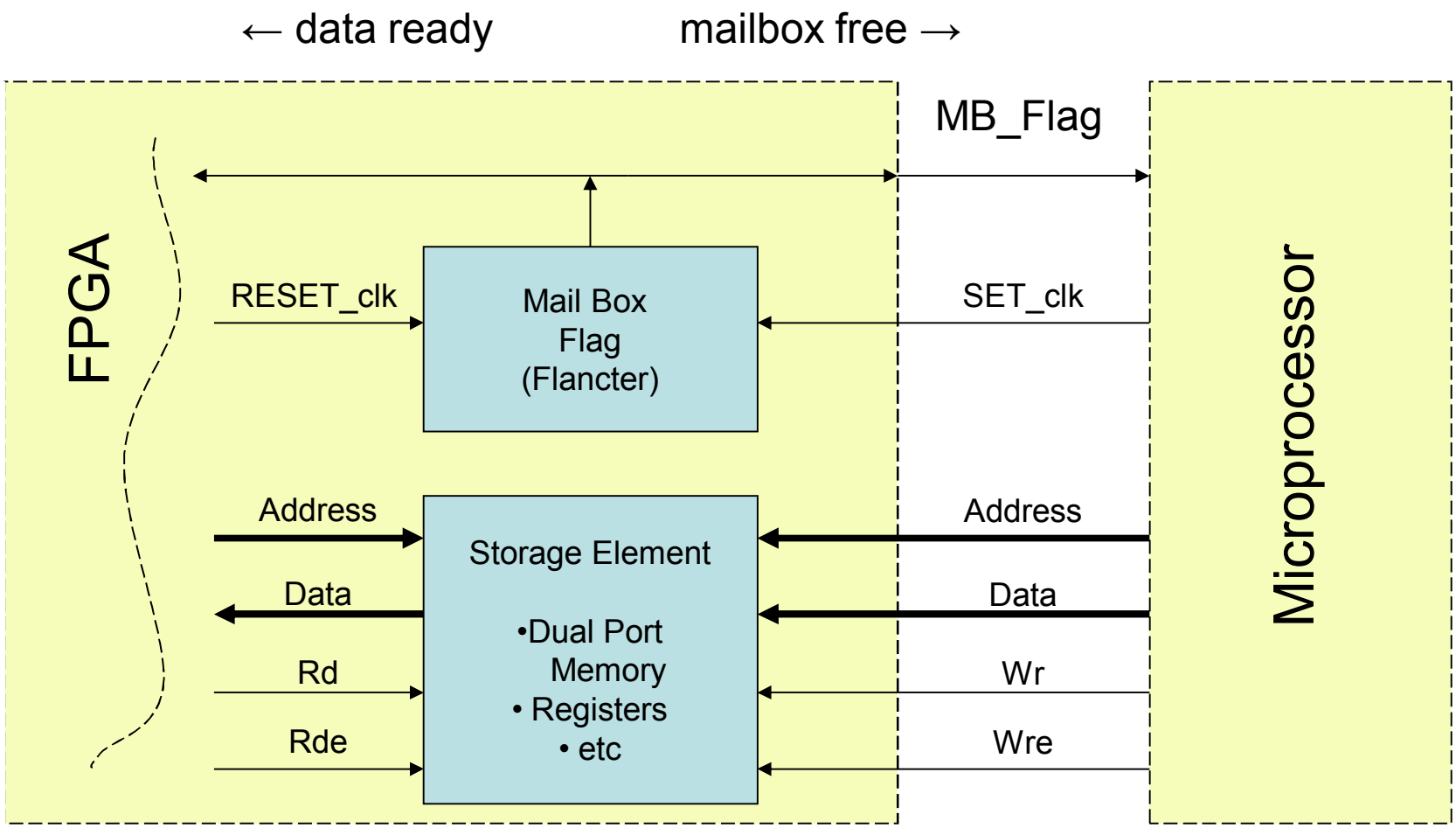


What about glitches ?

What about latch inference on OUT1 ?

What happens if the *out1* assignment is done inside the process?

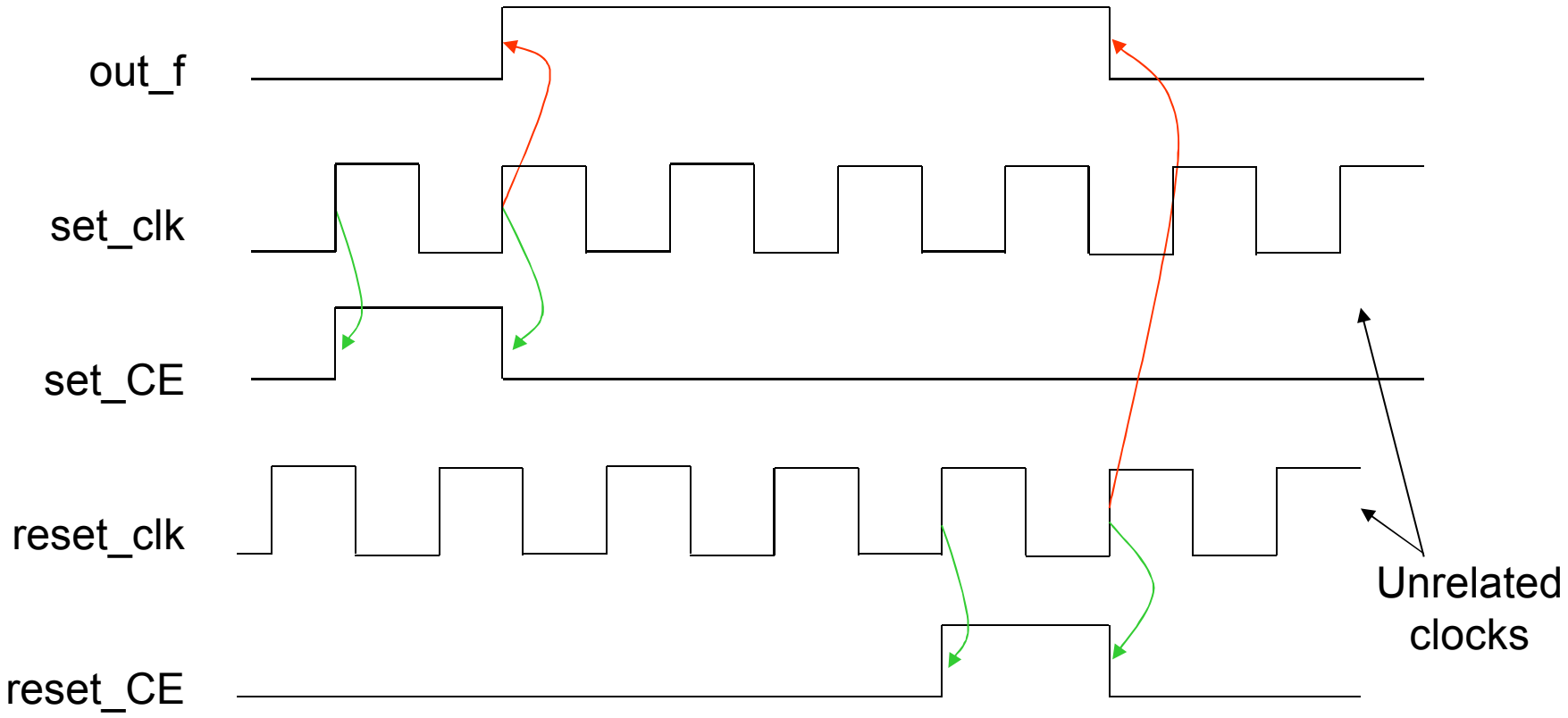
A possible FPGA ←→ Microprocessor handshaking mechanism



III. Flancter

(setting and clearing a flag with the edges of two signals)

Timing diagram

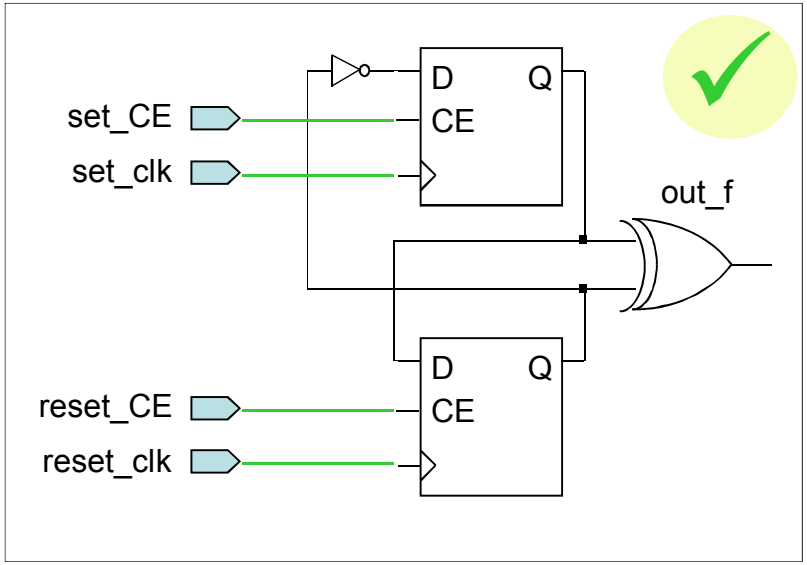


III. Flancter

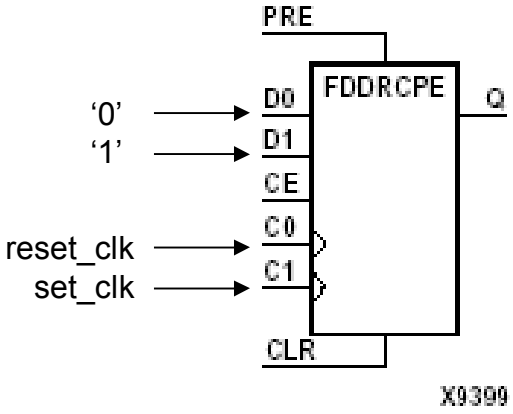
(setting and clearing a flag with the edges of two signals)

```
-- SET PROCESS:
Process (set_clk, set_CE)
begin
 if (set_CE = '1') then
  elsif (set_clk'event and set_clk = '1')
    then
    out_f <= '1';
 end if;
end process;

-- RESET PROCESS:
Process (reset_clk, reset_CE)
begin
 if (reset_CE = '1') then
    elsif (reset_clk'event and reset_clk = '1')
    then
    out_f <= '0';
 end if;
end process;
```

Try a solution based on standard cells for design portability



Synthesis ERROR:Xst:528 - Multi-source on signal <out_f>



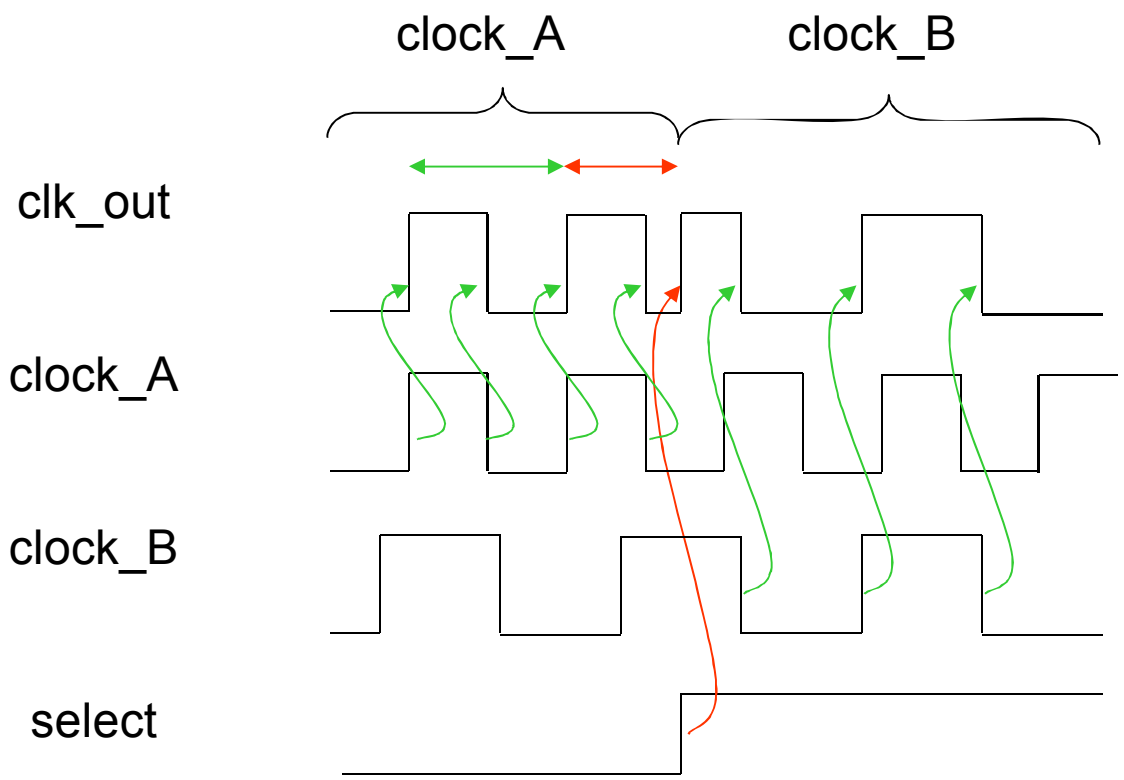
Inputs							Outputs
C0	C1	CE	D0	D1	CLR	PRE	Q
X	X	X	X	X	1	0	0
X	X	X	X	X	0	1	1
X	X	X	X	X	1	1	0
X	X	0	X	X	0	0	No Chg
↑	X	1	D0	X	0	0	D0
X	↑	1	X	D1	0	0	D1

Usage: For HDL, this design element is instantiated rather than inferred.

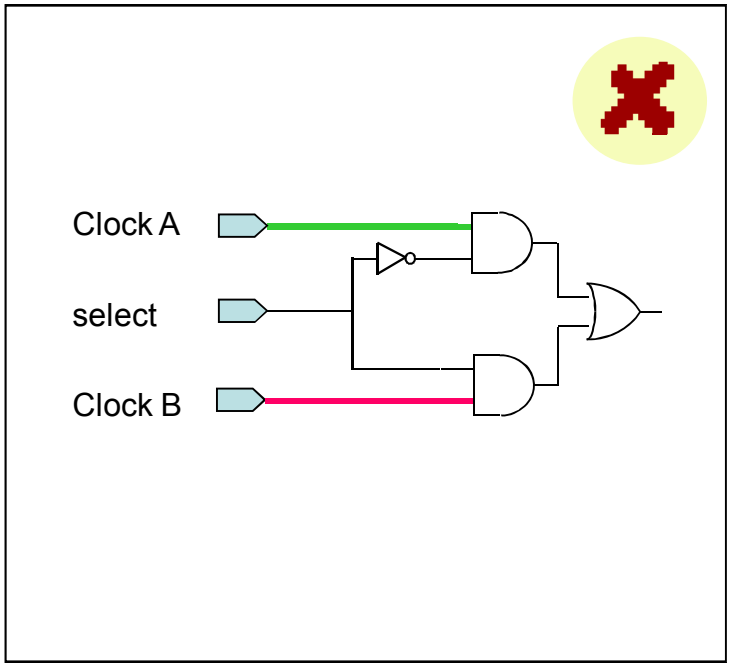
--VHDL Instantiation Template

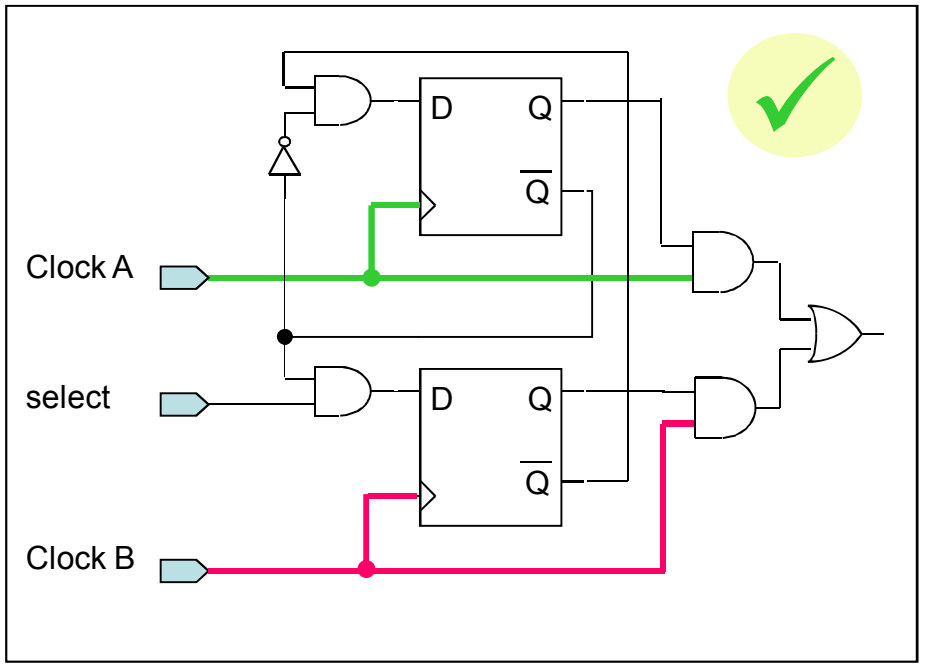
```
FDDRCPE_INSTANCE_NAME : FDDRCPE
  port map (Q => user_Q,
            C0 => user_C0,
            C1 => user_C1,
            CE => user_CE,
           CLR => user_CLR,
            D0 => user_D0,
            D1 => user_D1,
           PRE => user_PRE);
```

1) A suitable primitive could exist but could not be inferred from the HDL code. Synthesis Tools may not be mature enough for this.
2) Explore special resources (primitives) and manually instantiate them wherever is possible for max performance


clock_A
clock_B
clk_out
clock_A
clock_B
select

IV. Clock switching

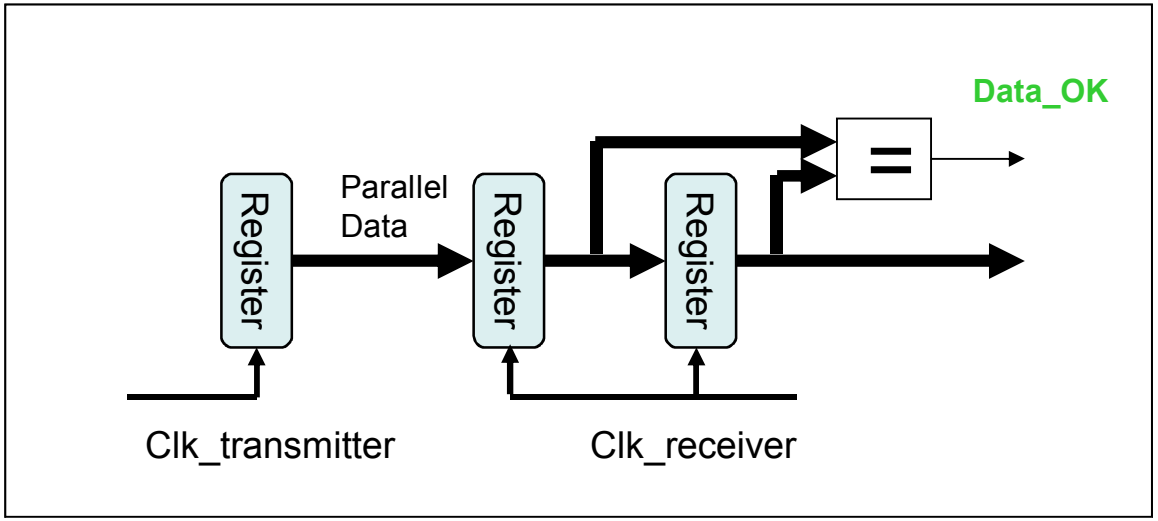




Transmitting parallel data with unrelated clocks

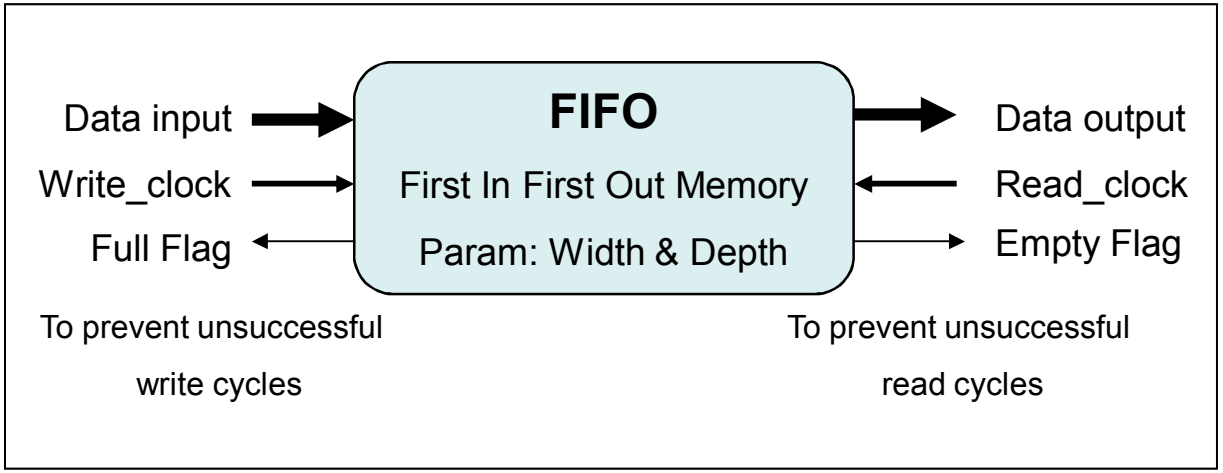
If the receiver clock is always faster than the transmitter clock
(e.g. freceiver>3*ftransmitter)

This only grants data integrity.
Doesn't prevent multiple reading or overwriting

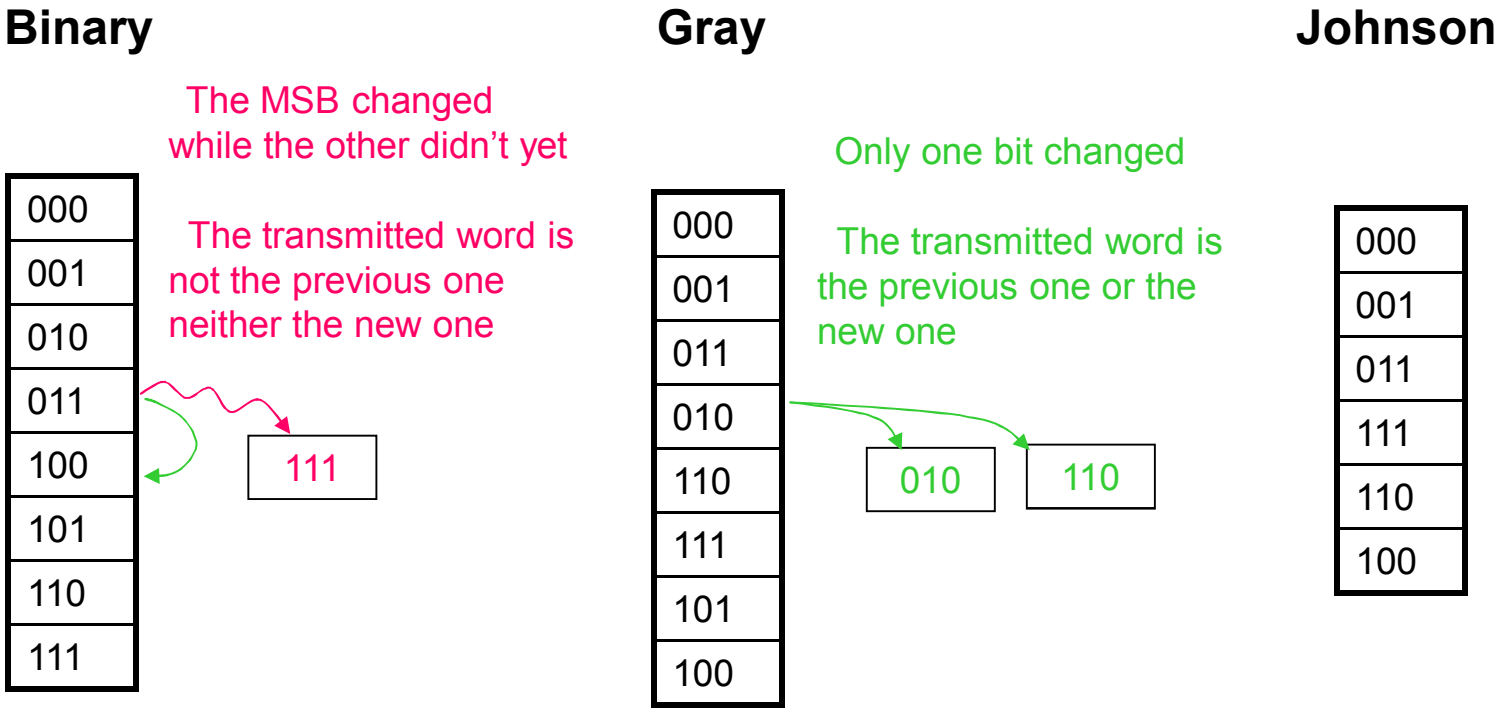


For completely unrelated clocks and sequential data transfer use Asynchronous FIFOs

Be sure the reading average frequency is equal or higher than the writing frequency.
(ave. freq fluctuations vs fifo length)

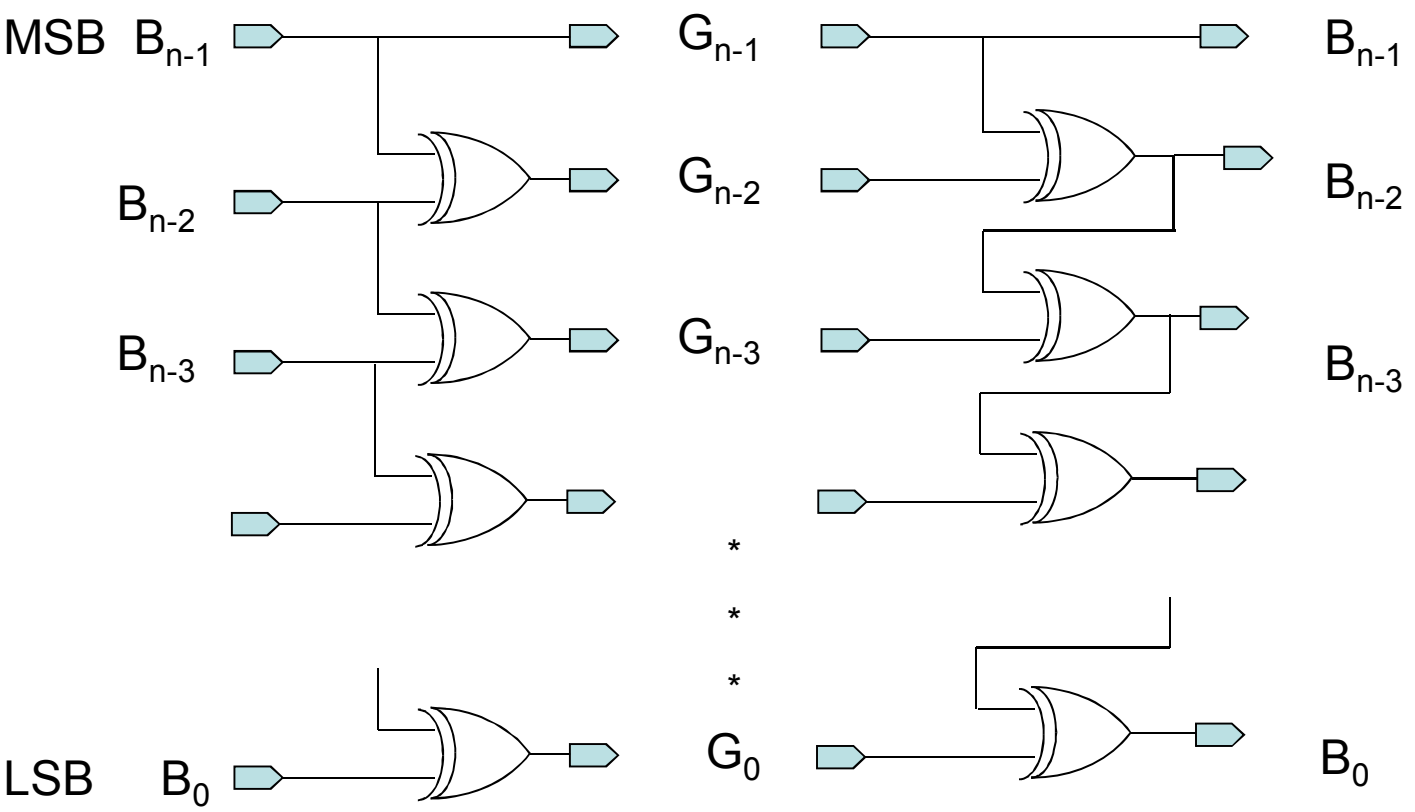


VI. Gray code



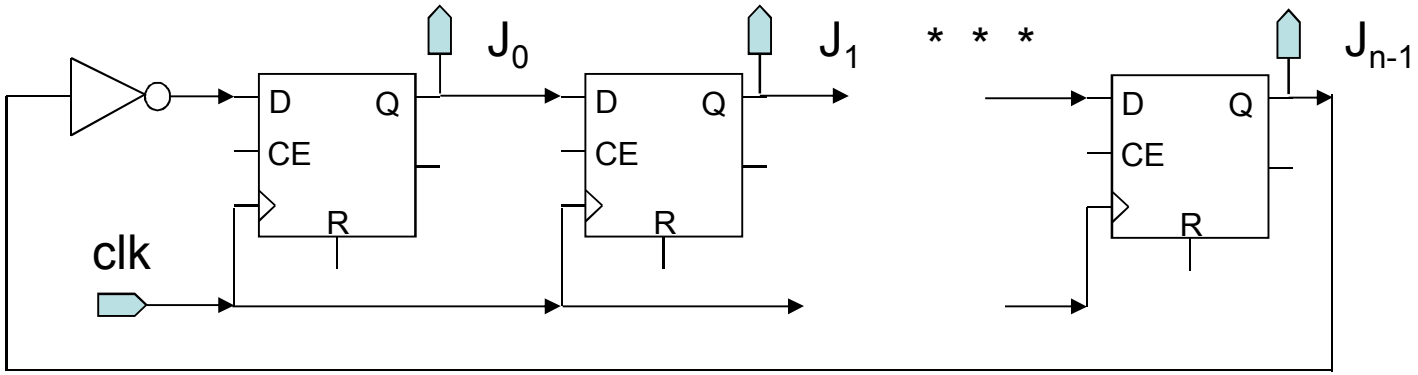
* Useful for sequential data as in counters, sequencers, etc
* All bits are stable except the only one that changes.

Binary ↔ Gray Conversion



Which are the critical paths?

Johnson counter (*twisted-ring counter*)

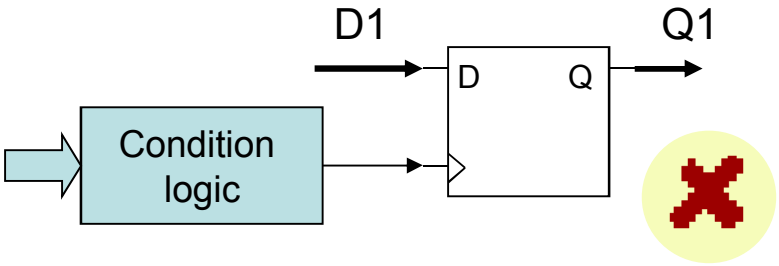



```
Process (clk, reset)
begin
  if (clk'event and clk = '1') then

    J <= J(WIDTH-2 downto 1) & not J(0);

  end if;
end process;
```

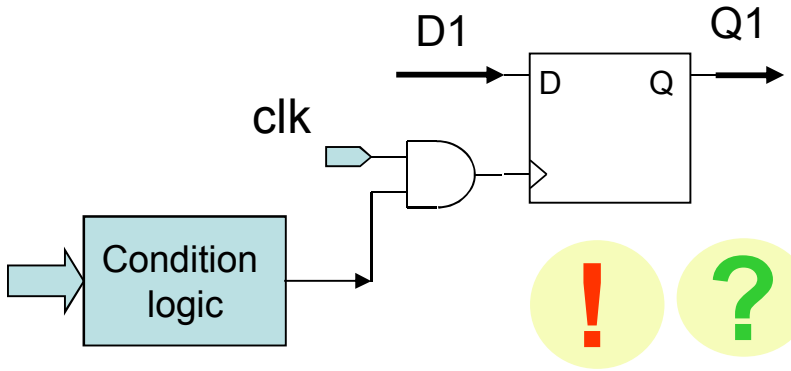
Fast, simple and glitch-free

Clocking Strategies (VERY IMPORTANT)

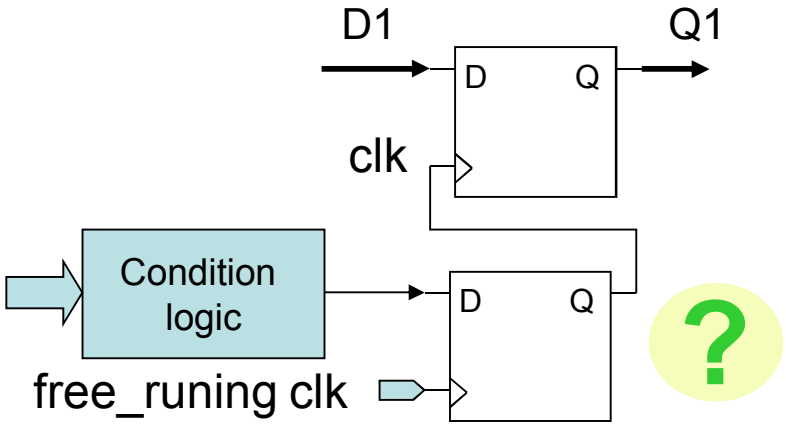
Combinational Synthetic Clock



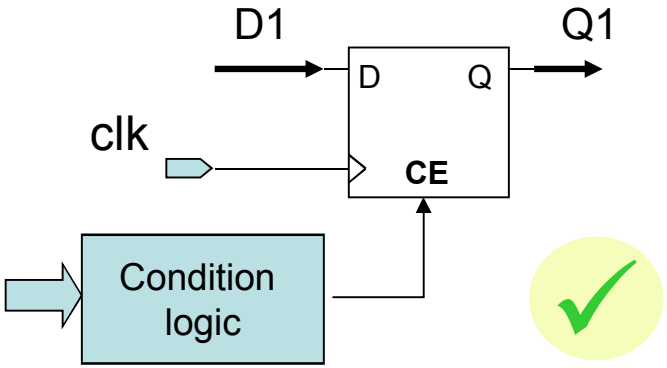
Free running clock + Gating



Registered Synthetic Clock



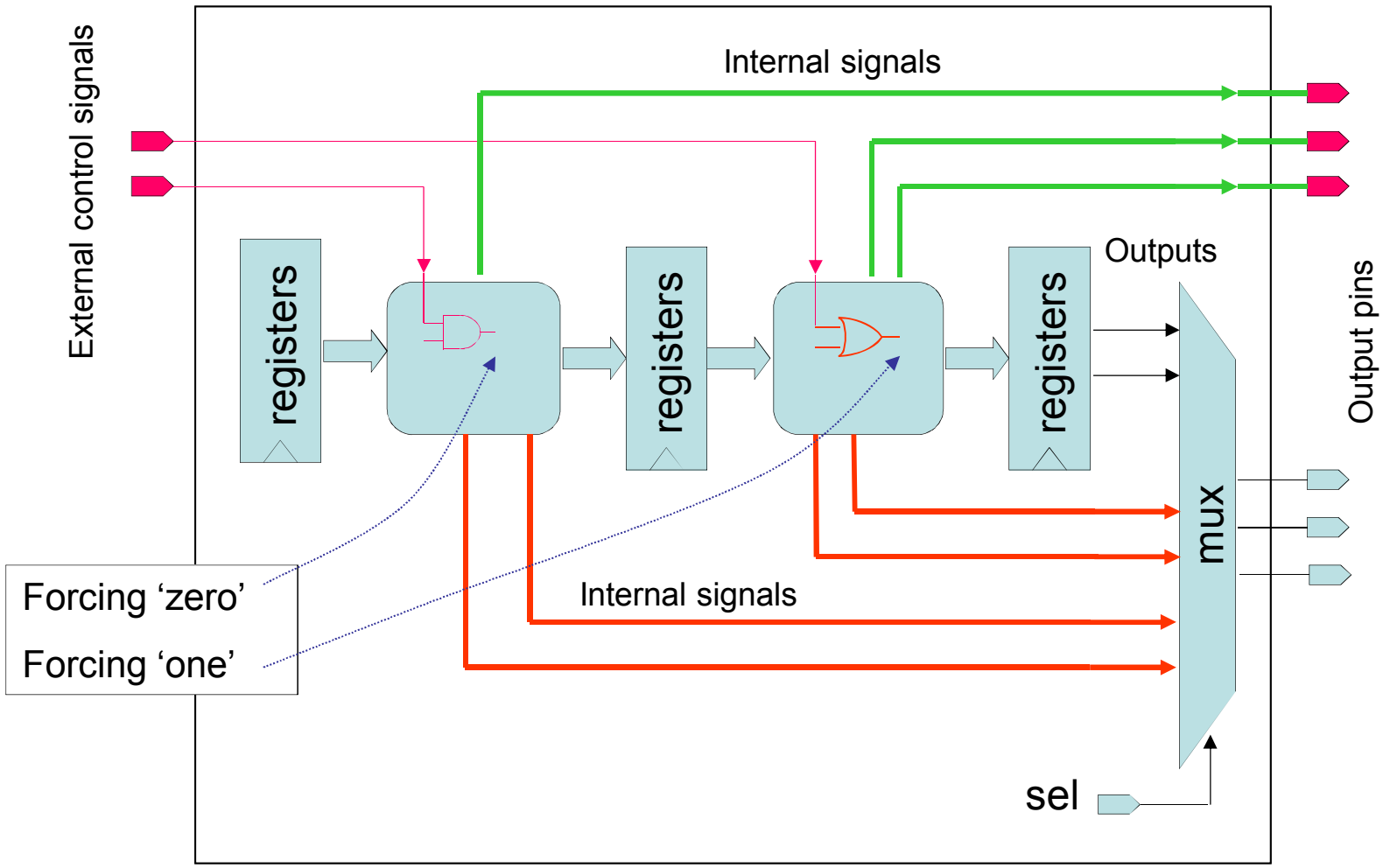
Free running clock + Clock enable



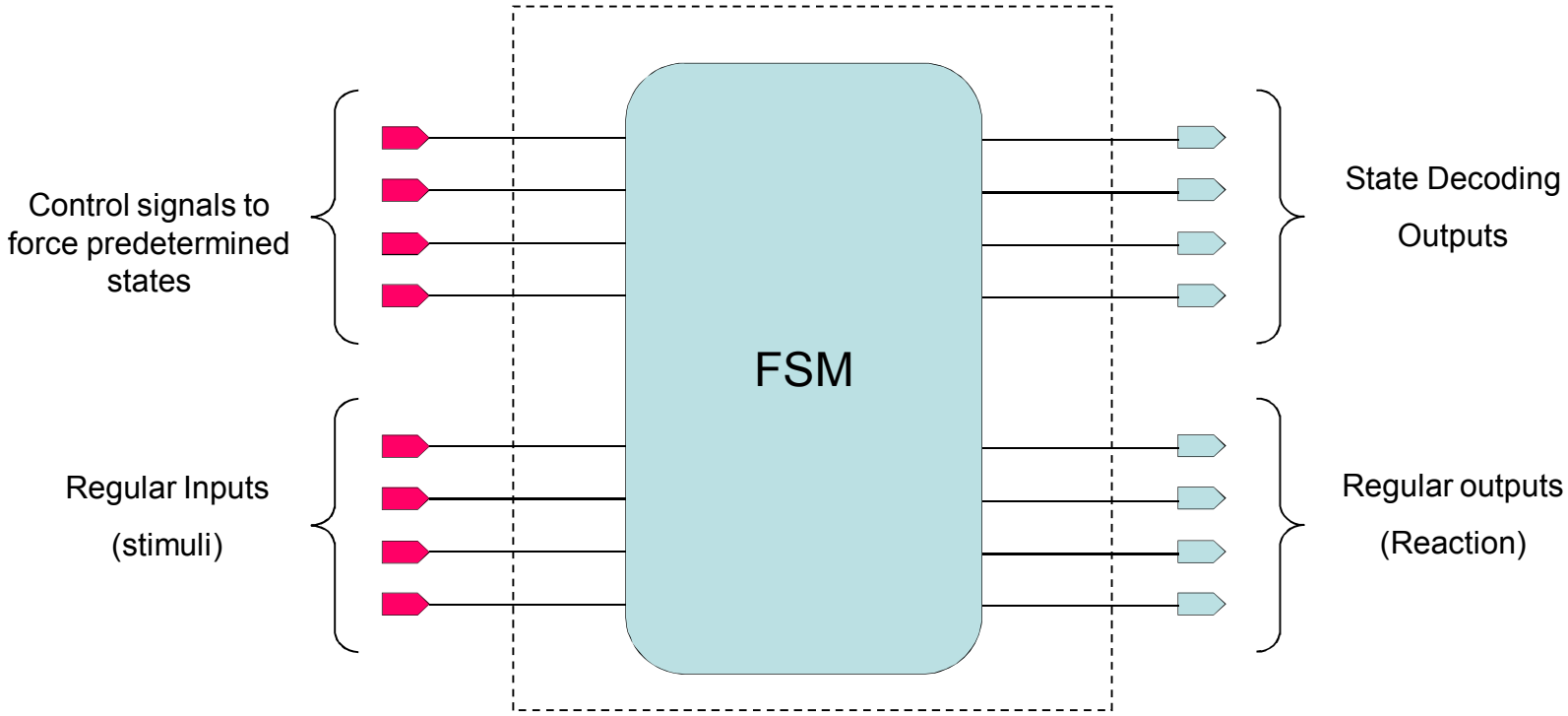
Debugging Techniques

- Seeing and controlling internal signals
- FSM debugging
- In chip logic analysis

Seeing and controlling internal signals



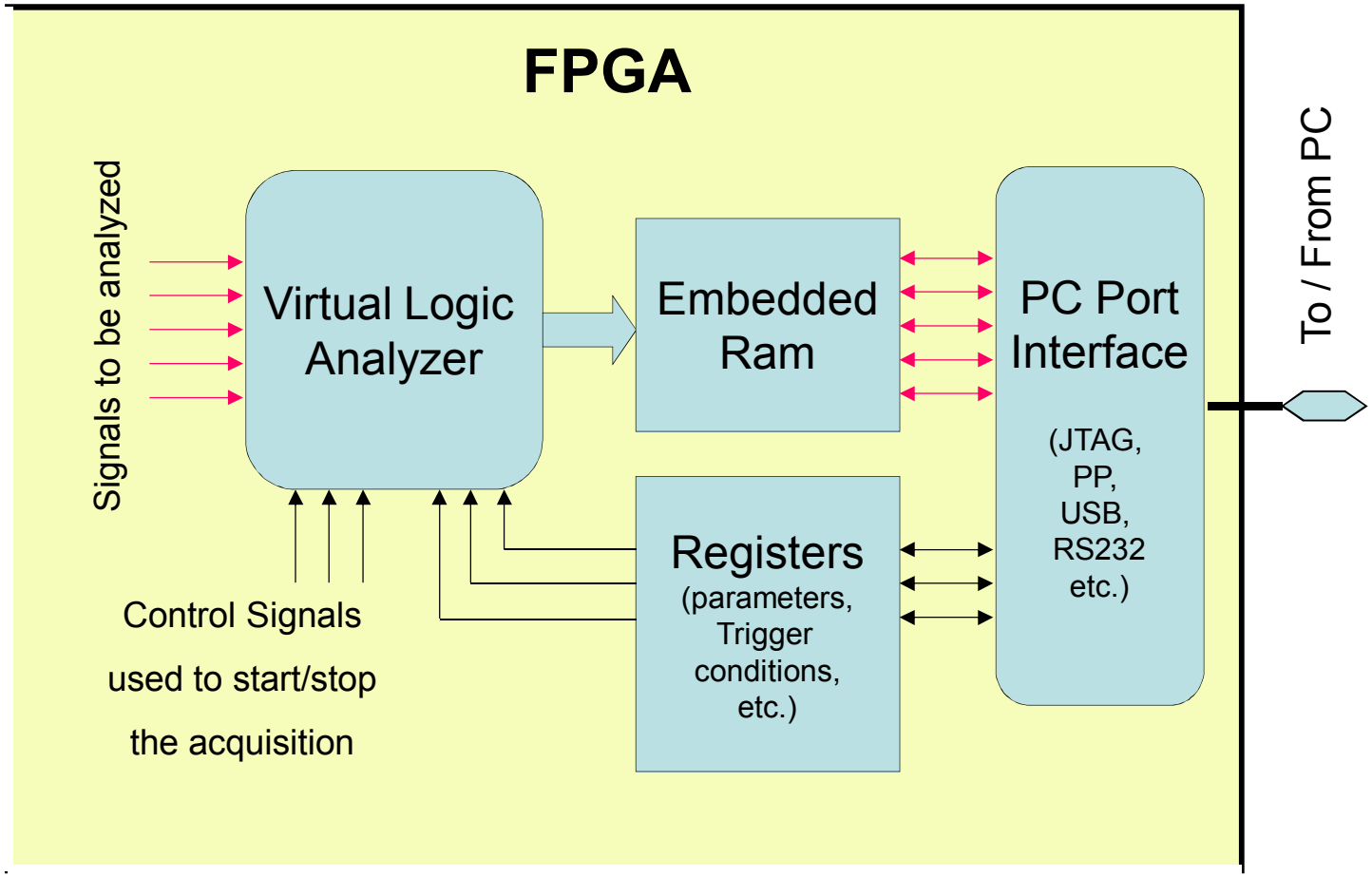
Debugging a FSM



-Design a hardware mechanism to force predetermined states (reset)

-Foresee outputs to decode and recognize those states and eventually "others"

On Chip Logic Analysis
- Virtual Logic Analyzer -



Most common mistakes

- Incomplete/Unclear/Wrong Specifications. Poor documentation
- Lack of a verification plan
- Debugging not foreseen
  - No visibility of internal signals/states
  - No hardware initialization mechanism
- Clocking
  - Asynchronous approach
  - Skew not well controlled
  - Clock enabling
- Metastability
  - Bouncing/dirty input signals
  - Asynchronous input signals
  - Multiple unrelated clock domains

Good design practices 1

- Prepare a good documentation: precise, exhaustive and easy readable
- Adopt a rigorous fully synchronous design approach whenever possible (clock enabling, better if only one free running clock, FIFOS, synchronous pipelining)
- Adopt a clear modular and hierarchical design approach to facilitate optimization, verification and reusability of functional blocks
- Design keeping in mind hardware verification. Provide external control and visibility of internal signals for debugging
- Use primitives for performance
- Don’t use primitives for portability

Good design practices 2

Use safe FSM (remember what "*others*" states means)

Use specific resources for clocks (dll, pll, clock buffers)

Synchronize all external inputs (and debounce and stabilize them if necessary)

Resynchronize internal signals between unrelated clock domains

Provide a hardware mechanism to port the system to a well known initial state (resets)

Check carefully the pad assignment report after implementation !

- **Y lo mas importante…**


Cruce los dedos antes de aplicar
voltaje a su FPGA !!!