



**Universidade de Brasília**

Departamento de Ciência da Computação

# Síntese de Hardware

Verilog

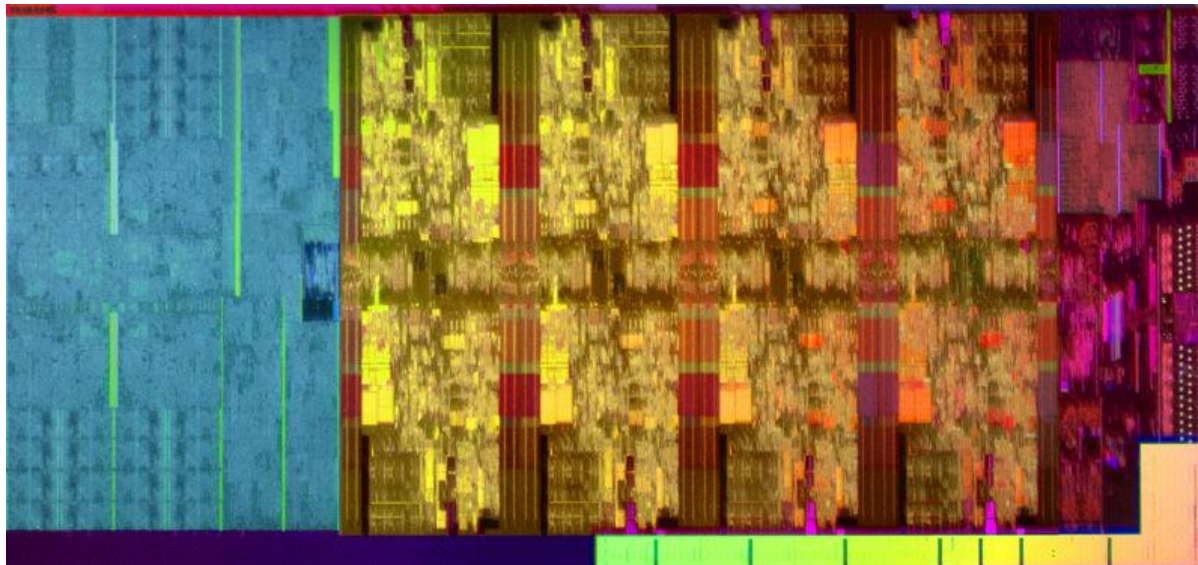
Altera DE1-SoC



# Linguagem de Descrição de Hardware (HDL)

## ■ Motivação:

Sistemas Digitais complexos são praticamente impossíveis de estudar e implementar através das técnicas tradicionais de síntese de circuitos digitais usando esquemático ao nível de transistores ou mesmo portas lógicas!

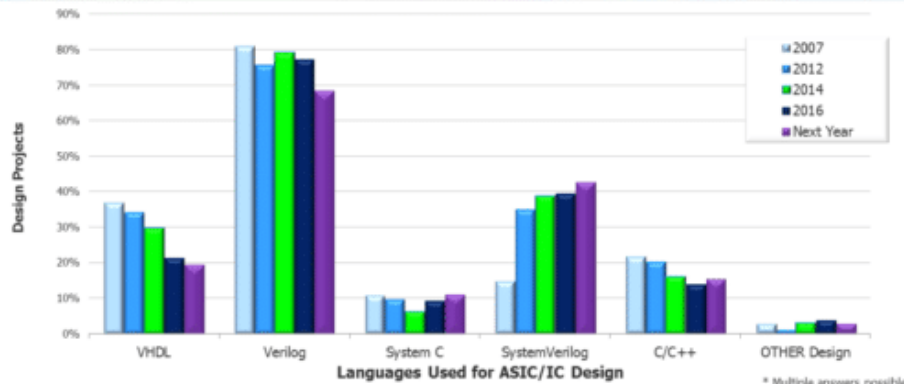




# Linguagem de Descrição de Hardware (HDL)

- Juntamente com VHDL, Verilog é uma das mais populares HDLs. Muito usada pela Intel (SystemVerilog).

## ASIC/IC Design Language Adoption Trends

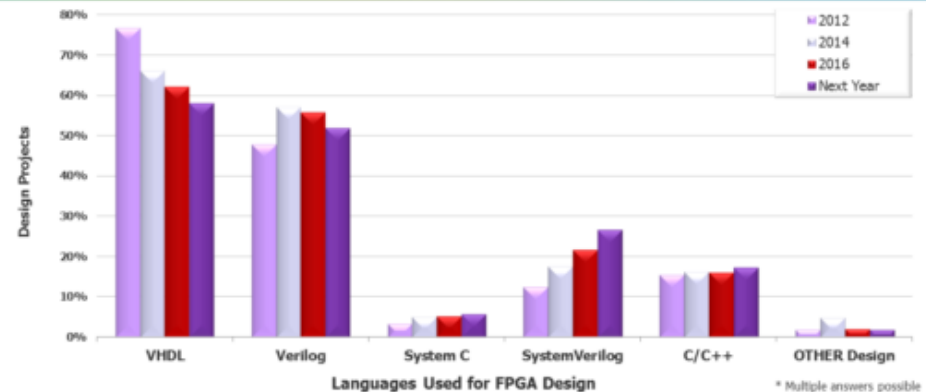


Source: Wilkon Research Group and Mentor Graphics, 2016 Functional Verification Study

© Mentor Graphics Corp. Company Confidential  
www.mentor.com



## FPGA Design Language Adoption Trends



Source: Wilkon Research Group and Mentor Graphics, 2016 Functional Verification Study

© Mentor Graphics Corp. Company Confidential  
www.mentor.com



Fonte: Mentor Graphics ([link](#))



# Verilog

## ■ Histórico

1984/85 – criada por Philip Moorby (Gateway Design System Co.)  
adquirida pela empresa Cadence

1990 – Open Verilog International : Domínio público

1995 – Padrão IEEE 1364

2005 – revisão e atualização do padrão, IEEE 1364

2012 – IEEE 1800, SystemVerilog, orientação a objetos - Superset

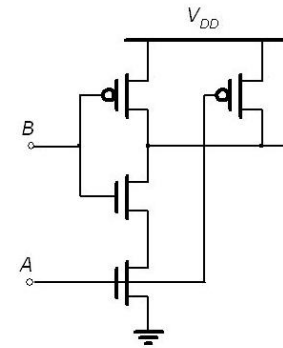
## ■ Usada para projetos de circuitos:

- ☐ ASIC (*Application Specific Integrated Circuit*)
- ☐ FPGA (*Field Programmable Gate Array*).

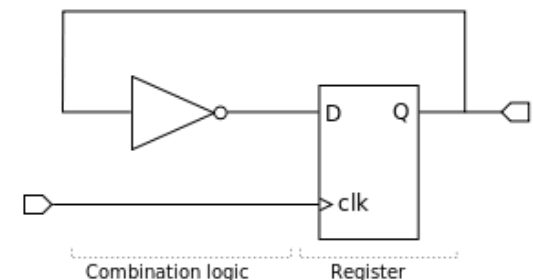
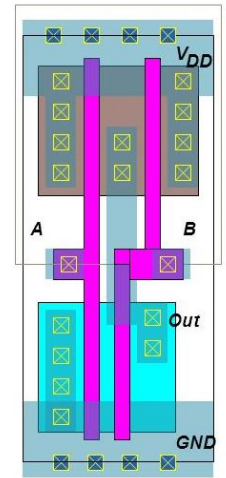
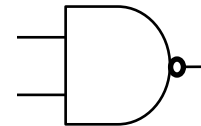
# Verilog

## ■ Níveis de Abstração

- Nível de Layout (*Layout Level*)
- Nível de Portas Lógicas (*Gate Level*)
- Nível de Transferência de Registradores (*Register Transfer Level*)
- Nível Comportamental (*Behaviour Level*)  
`assign a = b / c;`



2-input NAND gate





# SystemVerilog – aspectos básicos

## ■ Representação de Números:

	<tamanho em bits>'<base><número>	dado armazenado
Ex.: Decimal: ('d' ou 'D')	16'd255	0000000011111111
Hexadecimal ('h' ou 'H')	8'h9A	10011010
Binário ('b' ou 'B')	32'b1010	000000000000000000000000000000001010
Octal ('o' ou 'O')	8'o21	00_010_001

Números negativos: -8'd3 = -3 em 8 bits em complemento de 2 (11111101)

## ■ Valores:

níveis lógicos: 0 ou 1

desconhecido: x (nível lógico não conhecido)

alta impedância: z (fio não conectado)

Ex.: 32'bz 8'h0x 4'b1z0x

Obs: se o tamanho não for especificado o *default* é 32 bits!



# SystemVerilog – aspectos básicos

## ■ Tipos de Dados

- **wire**: define um ou conjunto de fios

```
Ex.: wire a;           // a é um fio  
      wire [7:0] b;    /* b é um conjunto de 8 fios {b[7],b[6],...,b[1],b[0]}*/
```

- **reg** : define um registrador (síncrono ou assíncrono)

```
Ex.: reg a;           // a é um latch ou flip-flop  
      reg [7:0] b;    /* b é um conjunto de 8 latches ou flip-flops */
```

- **logic** : define um registrador ou wire

```
Ex.: logic a;          // a é um elemento definido pelo sintetizador  
      logic [7:0] b;    /* b é um elemento de 8 bits definido pelo  
                        sintetizador */
```

- Tipos abstratos (32 bits): **integer**, **real** (IEEE 754)



# SystemVerilog

## ■ Características da sintaxe

### ❖ Case Sensitive :

`reset` é diferente de `Reset` e de `RESET`

### ❖ Nomes não podem iniciar por números:

`2mux` é um nome inválido! `mux2` é válido

### ❖ Espaços em branco são ignorados

### ❖ Comentários iguais à linguagem C

`// comentário de uma linha`

`/* comentário`

`em múltiplas linhas */`

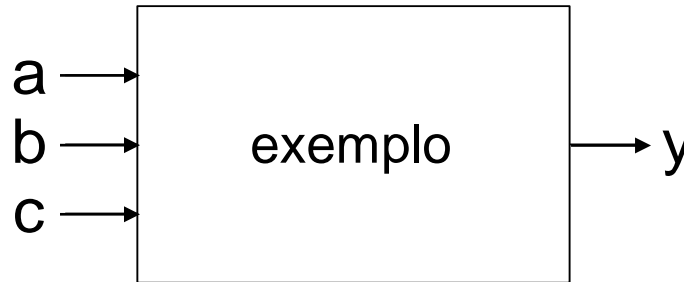




# SystemVerilog

## ■ Estrutura básica de uma descrição Verilog

```
module <nome>(<entradas>, <saídas>);  
    <descrição>;  
endmodule
```



```
module exemplo(input logic a, b, c,  
               output logic y);  
    assign y = ~a & ~b & ~c | a & ~b & ~c | a & ~b & c;  
endmodule
```



# SystemVerilog

## ■ Operações Básicas do Verilog e precedência

( )	Indica prioridade
{}, {{{}}	Concatenação
~	NOT
*, /, %	multiplicação, divisão, módulo
+, -	adição, subtração
<<, >>	deslocamento lógico
<<<, >>>	deslocamento aritmético
<, <=, >, >=	Comparações
==, !=	igual, diferente
&, ~&	AND, NAND
^, ~^	XOR, XNOR
, ~	OR, NOR
?:	operador ternário condicional



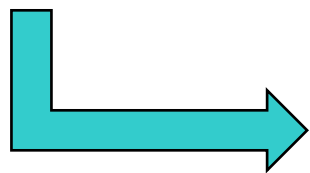
# SystemVerilog

## ■ Síntese

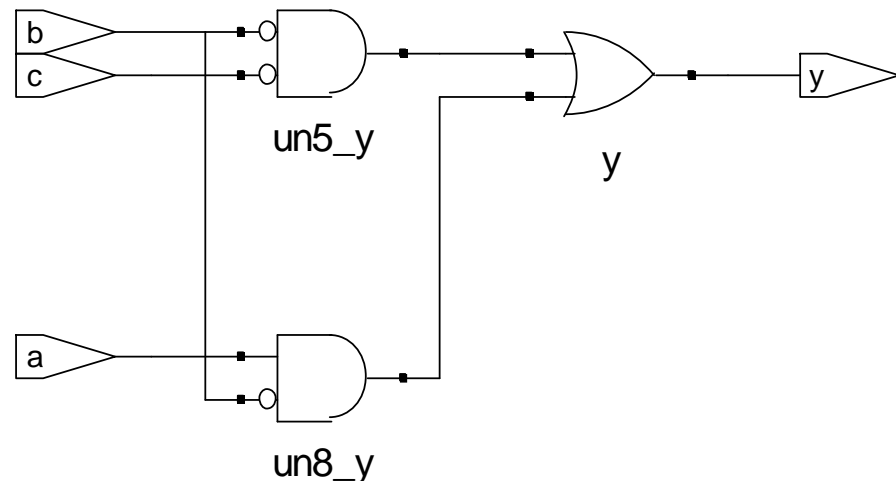
A síntese de um circuito corresponde à tradução (compilação) de uma descrição Verilog em uma netlist (descrição das interconexões dos circuitos) pelo uso de uma ferramenta de síntese (Quartus).

Geralmente, durante a compilação, diversas otimizações são realizadas.

```
module exemplo(input  logic a, b, c,  
                output logic y);  
    assign y = ~a & ~b & ~c | a & ~b & ~c | a & ~b & c;  
endmodule
```



síntese





# Atribuições

## 1) Definindo um circuito combinacional

Neste caso, geralmente  $y$  é do tipo `wire`.

```
assign y = ~a & ~b & ~c | a & ~b & ~c;
```

ou

```
always @(*) begin  
    y <= ~a & ~b & ~c | a & ~b & ~c; end;
```

**Operador atribuição Não-Blocante**

ou

```
always @(*) begin  
    y = ~a & ~b & ~c;  
    y = y | a & ~b & ~c; end;
```

**Operador atribuição Blocante**



# Atribuições

## 2) Definindo um circuito sequencial

Neste caso, geralmente  $y$  é do tipo `reg`.

```
always @ (posedge clock) begin
    y <= ~a & ~b & ~c | a & ~b & ~c; end;
```

Operador atribuição Não-Blocante

ou

```
always @ (posedge clock) begin
    y = ~a & ~b & ~c;
    y = y | a & ~b & ~c; end;
```

Operador atribuição Blocante



# FPGA: Intel Cyclone-V 5CSEMA5F31C6

## Field Programmable Gate Array

Chip capaz de implementar qualquer sistema digital através da definição da interconexão de seus elementos.

O modelo Cyclone V possui:

- 896 pinos
- 32.070 *Adaptive Logic Module* (ALM)
  - Função Lógica de 8 entradas
  - 4 registradores de 1 bit
  - Somadores encadeáveis (carry)
  - Roteamento
- Memória RAM 4.065.280 bits
- 87 DSP (multiplicador 18x18, 64 add)
- 6 PLL (*Phase Locked Loop*)
- 288 Pinos de IO para o usuário
- ARM Cortex-A9 dual core

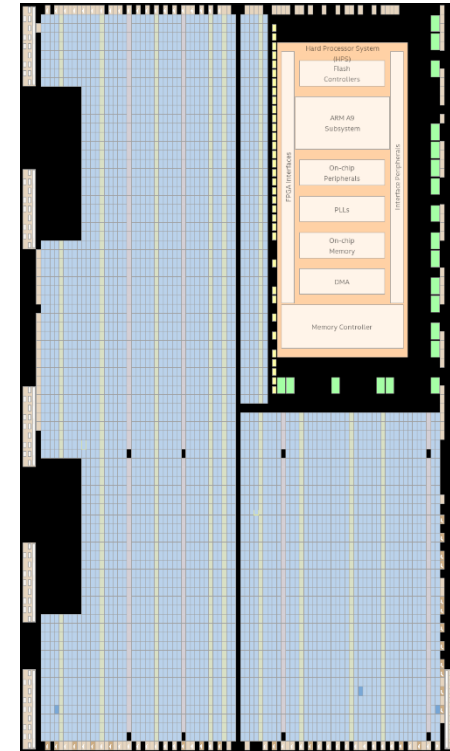
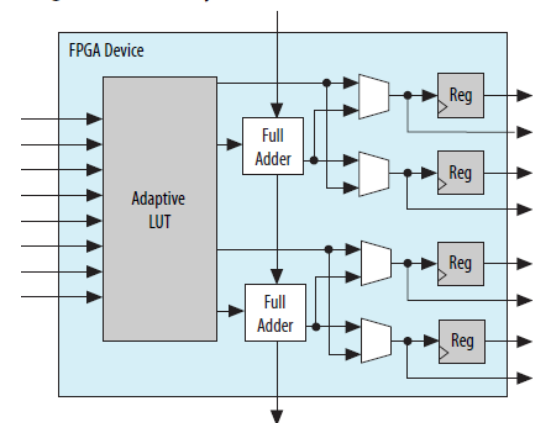


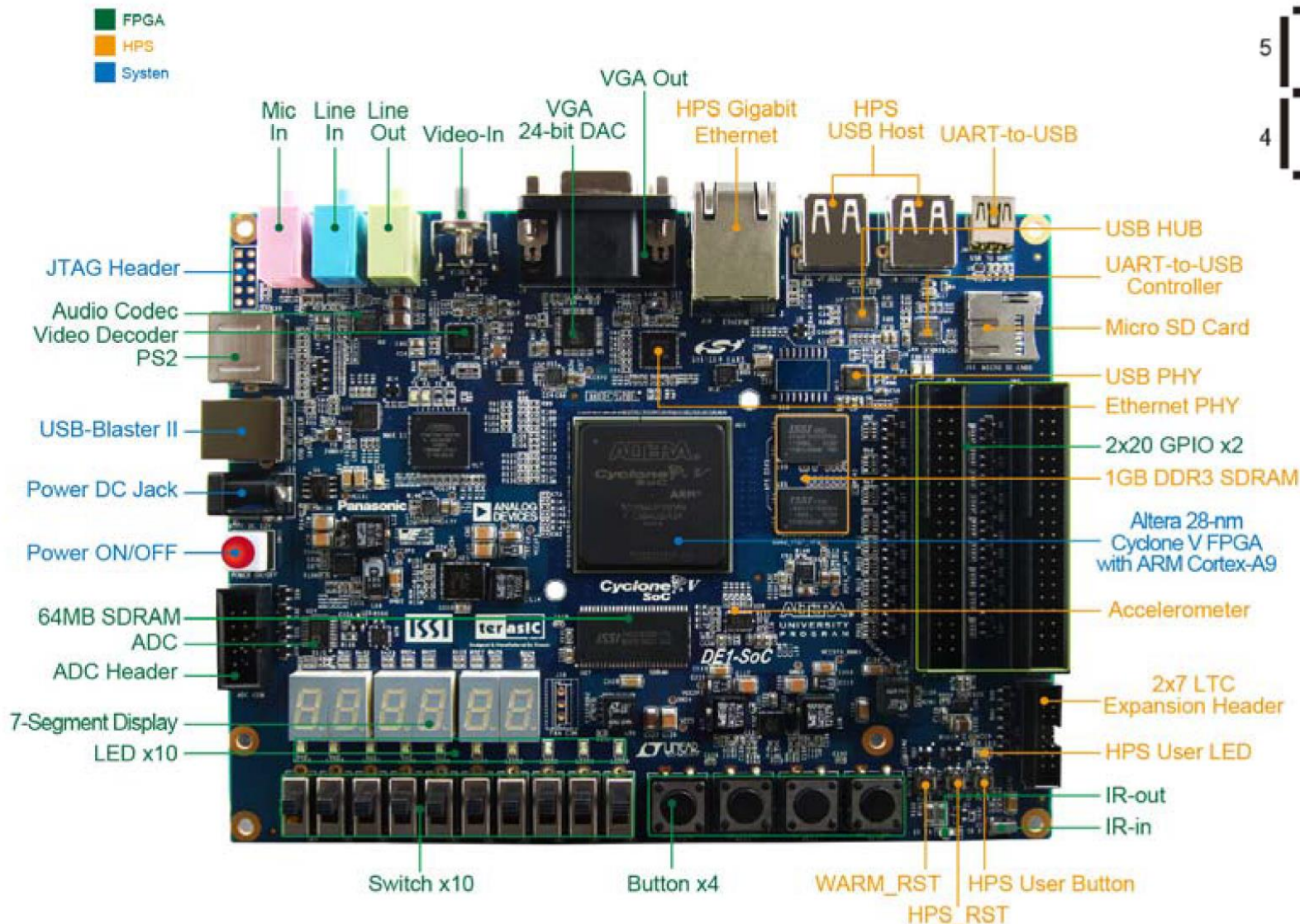
Figure 8: ALM for Cyclone V Devices





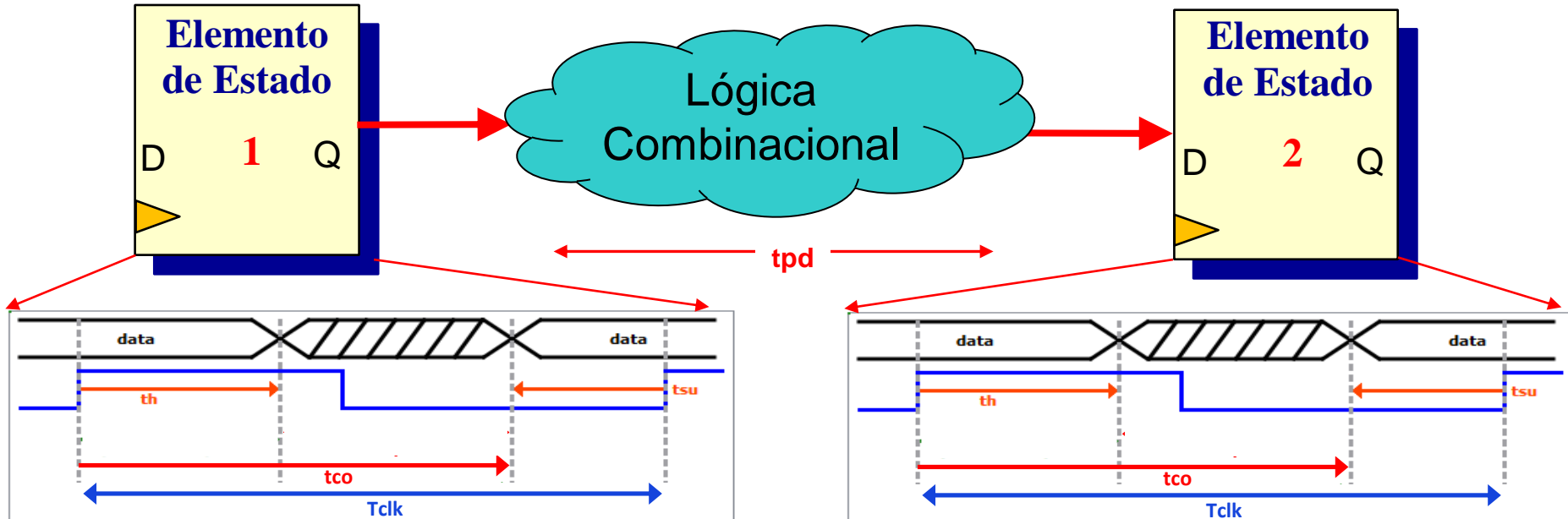
# Plataforma Intel DE1-SoC

- Diversos dispositivos de IO já ligados aos pinos do FPGA





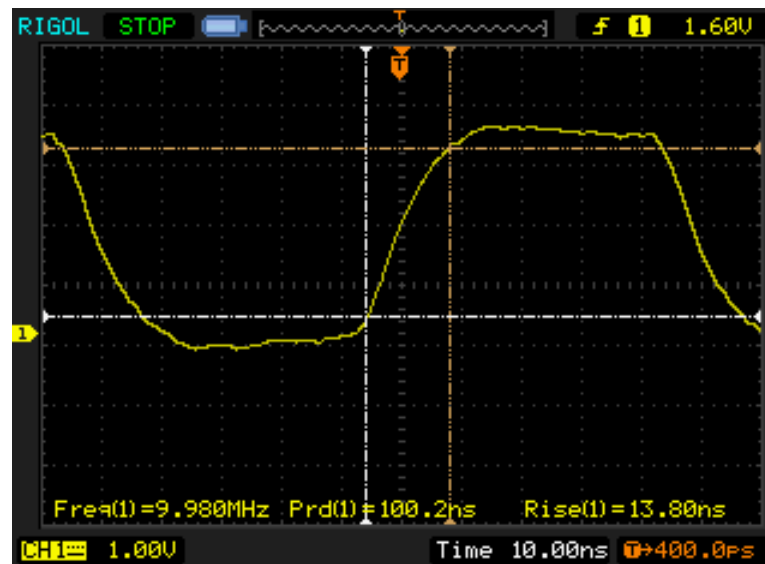
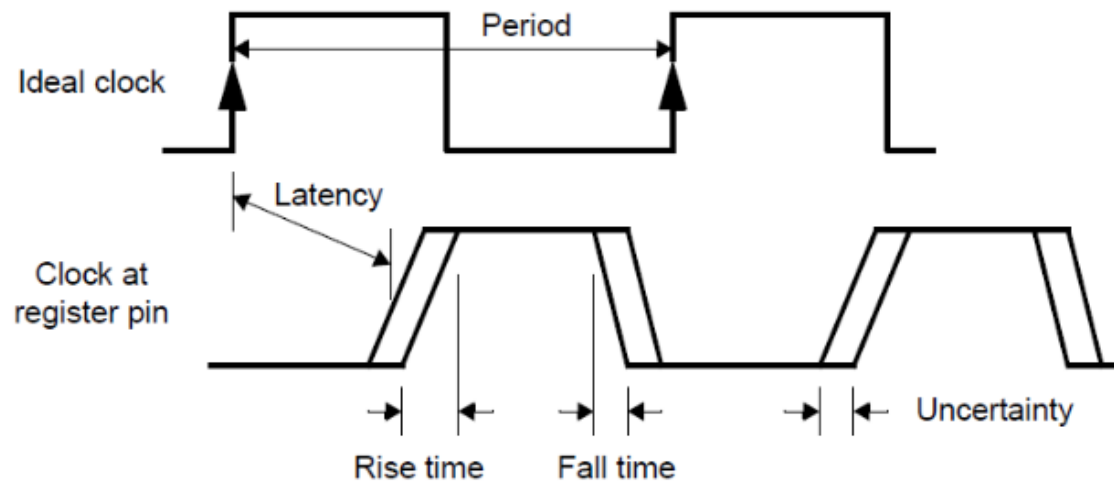
# Revisão: Temporização em circuitos digitais



- **A escrita no elemento de estado 1 deve respeitar os tempos de:**
  - tempo de pré-carga (*setup time*,  $t_{su}$ ) do elemento 1
  - tempo de hold (*hold time*,  $t_h$ ) do elemento 1
- **O elemento de estado 2 só pode ser escrito depois que os dados em sua entrada estarem estáveis**
  - atraso de propagação da saída dado o clock (*clock to output*,  $t_{co}$ ) do elemento 1
  - atraso da lógica combinacional (*propagation delay*,  $t_{pd} = \max(t_{p_{HL}}, t_{p_{LH}})$ )
  - tempo de pré-carga (*setup time*,  $t_{su}$ ) do elemento de estado 2



# Sinal de relógio: clock

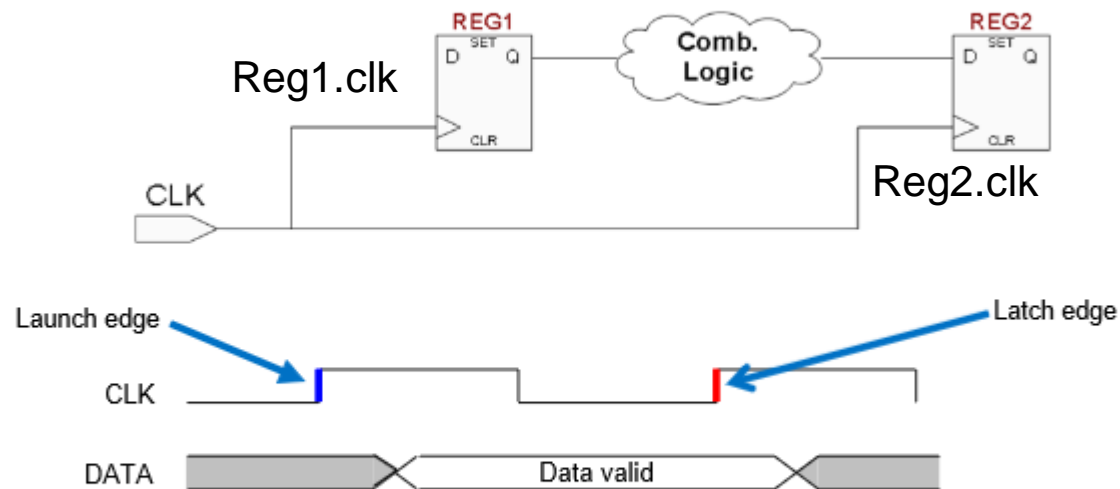




# Temporização em circuitos digitais

[https://www.intel.com/content/www/us/en/programmable/customertraining/OLT/TQ\\_Intro\\_p1/presentation\\_html5.html](https://www.intel.com/content/www/us/en/programmable/customertraining/OLT/TQ_Intro_p1/presentation_html5.html)

## Launch & Latch Edges



Launch edge: the edge which “launches” the data from the source register

Latch edge: the edge which “latches” the data at the destination register (with respect to the launch edge, selected by timing analyzer; typically 1 clock cycle, i.e. rising to rising edge)

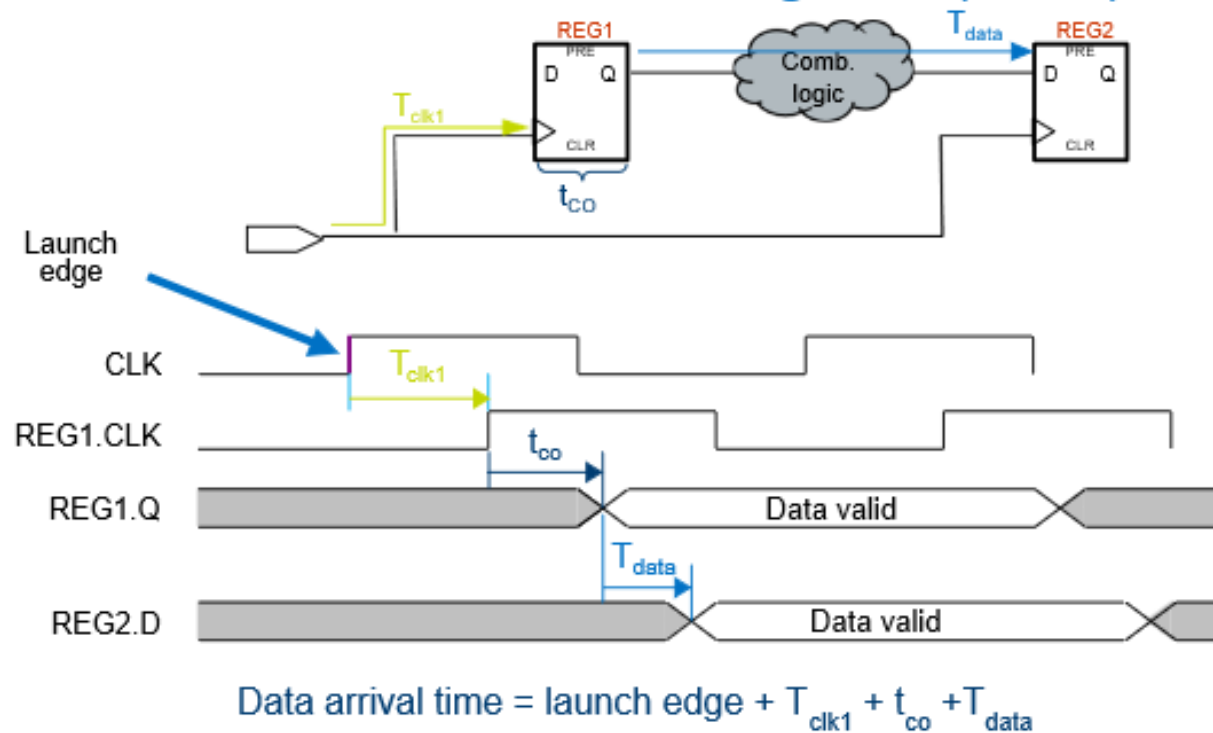
O tempo de chegada do clock nas entradas dos regs geralmente, não são o mesmo! Dependem do caminho! Clock Skew.



# Tempo de chegada do dado ( $t_{c\_data}$ )

## Data Arrival Time

The time for data to arrive at the destination register's (REG2) D input



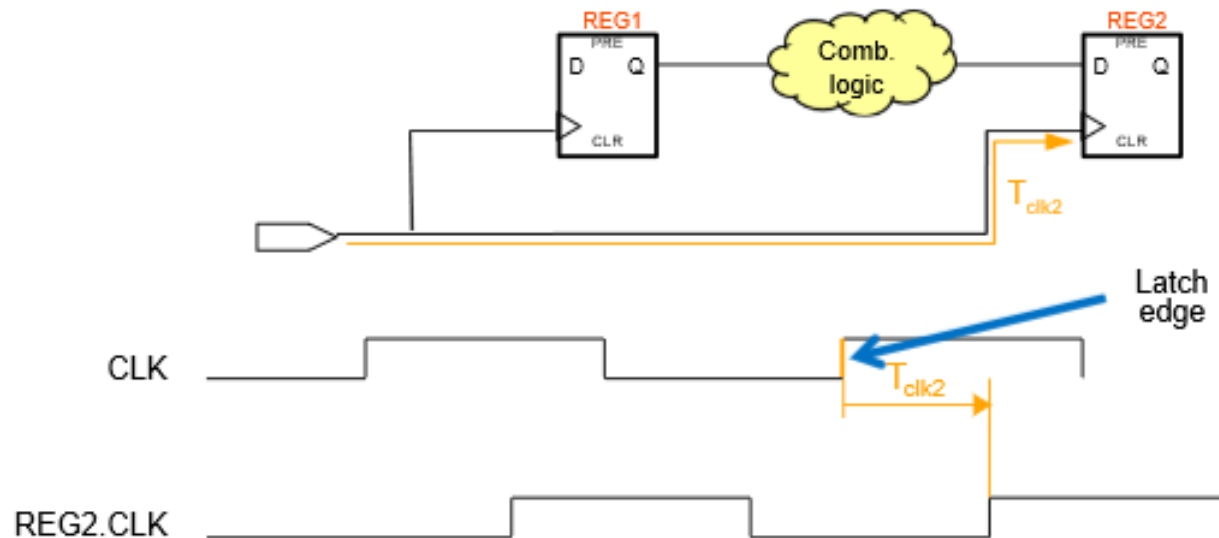
$$t_{c\_data} = t_{clk1} + t_{co} + t_{pd}$$



# Tempo de chegada do clock ( $t_{\text{clock2}}$ )

## Clock Arrival Time

The time for the clock to arrive at destination register's (REG2) clock input



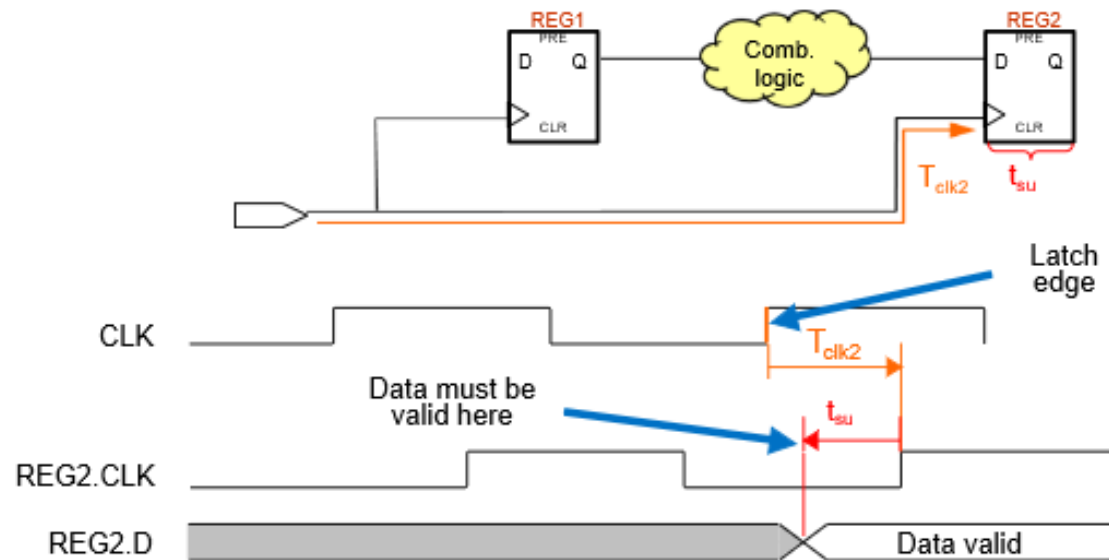
$$\text{Clock arrival time} = \text{latch edge} + T_{\text{clk2}}$$



# Tempo requerido de setup ( $t_{r\_setup}$ )

## Data Required Time (Setup)

The minimum time required for the data to be valid before the latch edge so the data can be successfully latched into the destination register (REG2)



Data required time (Setup) = Clock arrival time -  $t_{su}$  - Setup uncertainty (clock Jitter)

$$t_{r\_setup} = t_{clk2} - t_{su} - \text{incerteza}$$

## Data Required Time (Hold)

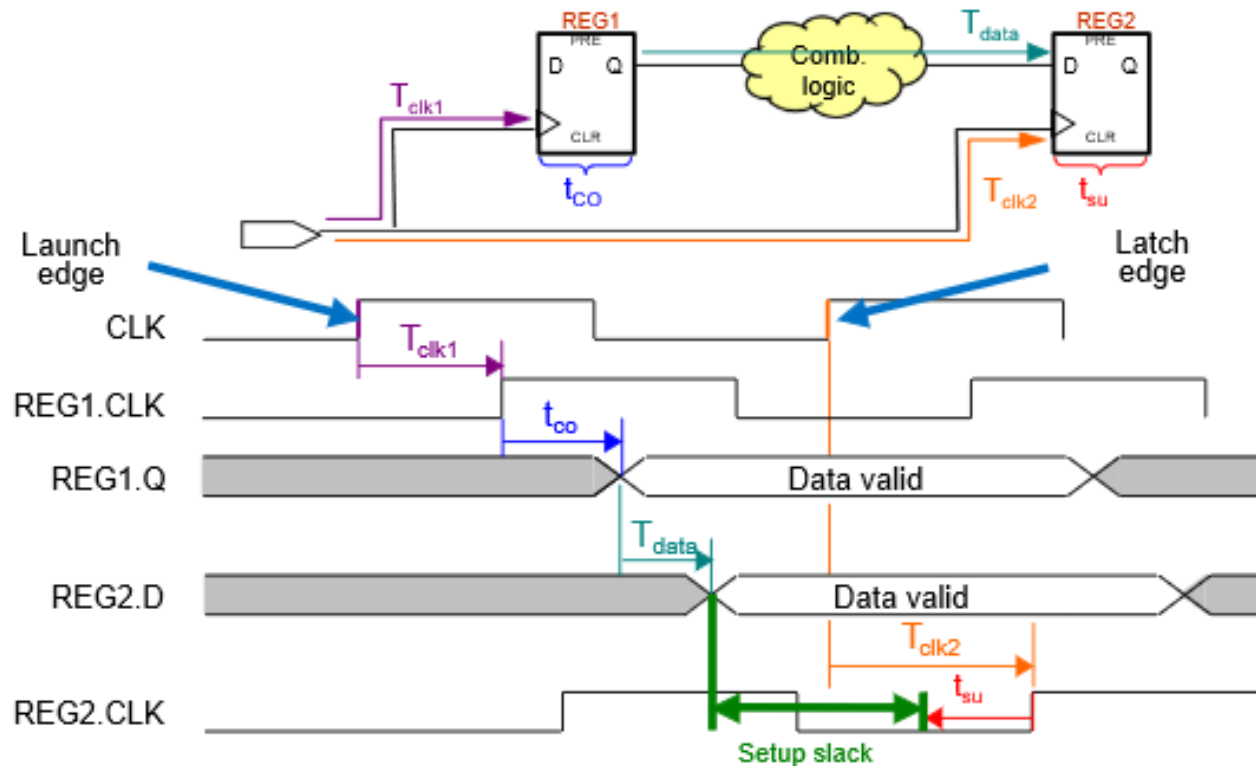
The diagram illustrates the timing requirements for a crossbar switch. It features three signals: CLK, REG2.CLK, and REG2.D. The CLK signal is a periodic square wave. The REG2.CLK signal is a single clock pulse. The REG2.D signal is a data bus that is valid during the REG2.CLK pulse. The timing parameters shown are  $T_{clk2}$  (the period between the falling edge of CLK and the rising edge of REG2.CLK) and  $t_h$  (the hold time after the falling edge of REG2.CLK). A blue arrow points to the rising edge of REG2.CLK, labeled "Latch edge". Another blue arrow points to the falling edge of REG2.CLK, labeled "Data must remain valid to here". The data valid period is indicated by a grey bar on the REG2.D signal.

$$tr_{hold} = t_{clk2} + t_h + \text{incerteza}$$



# Tempo de folga de setup ( $S_{\text{setup}}$ )

## Setup Slack (2)

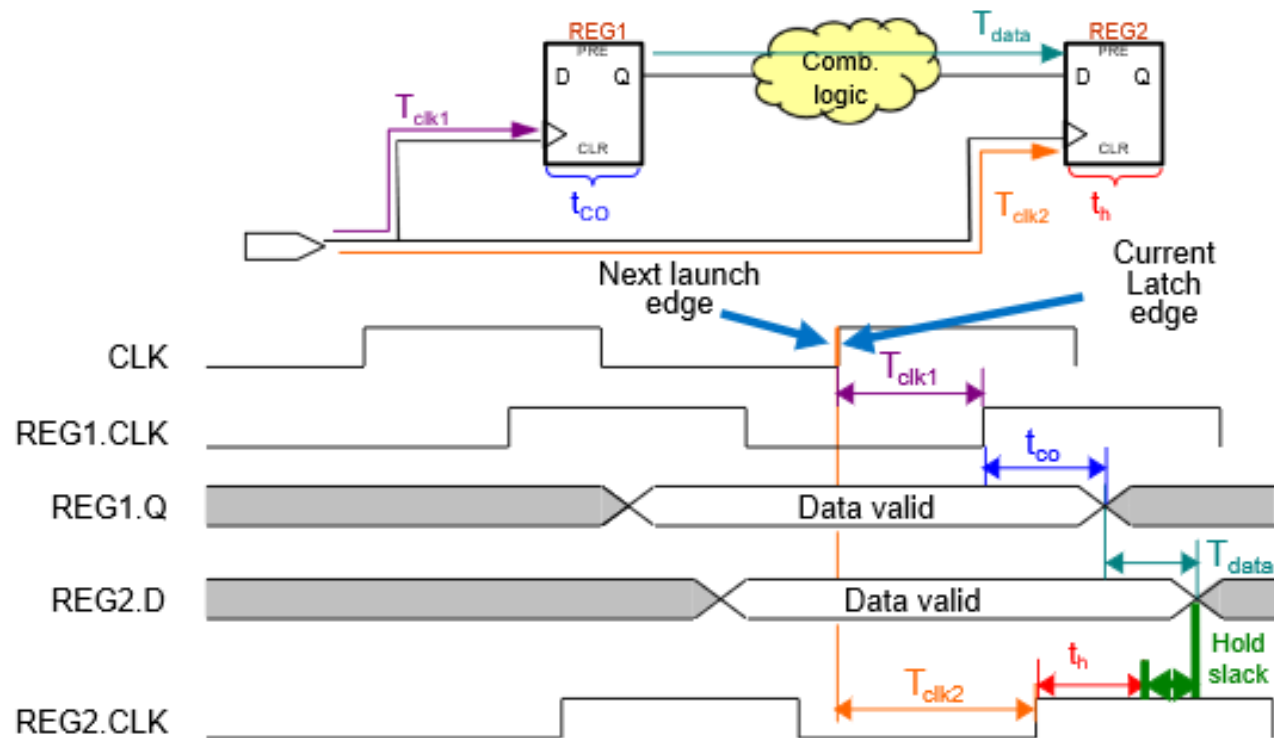


$$S_{\text{setup}} = \text{Min}\{tr_{\text{setup}}\} - \text{Max}\{tc_{\text{data}}\}$$



# Tempo de folga de hold ( $S_{\text{hold}}$ )

## Hold Slack (2)





$$S_{\text{hold}} = \text{Min}\{t_{c_{\text{data}}}\} - \text{Max}\{t_{r_{\text{hold}}}\}$$





# Análise dos tempos de folga

- Se os tempos de folga forem positivos → OK! 
  - ☐ Requerimentos foram atendidos
  
- Se os tempos de folga forem negativos → Problema! 
  - ☐ Requerimentos não foram atendidos
  - ☐ Precisa rever o projeto e/ou os requerimentos temporais