

# Laboratório de Sistemas Digitais

## Aula Teórico-Prática 8

Ano Letivo 2019/20

Modelação de Máquinas de Estados Finitos

Modelo de Mealy

Sistemas Computacionais - *Controlpath e Datapath*

FSMs comunicantes

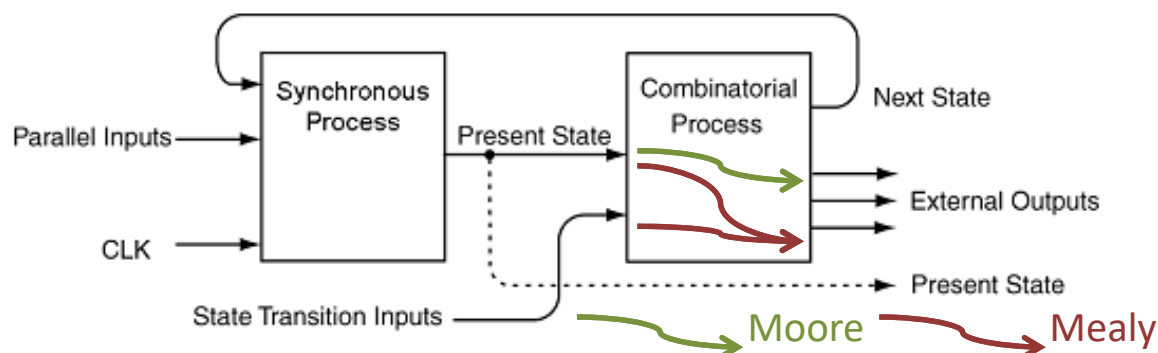
Guilherme Campos, Ioulia Skliarova

# Conteúdo

- Abordagens de modelação de Máquinas de Estados Finitos (MEFs) / *Finite State Machines* (FSMs) baseadas em VHDL
  - Modelo de *Mealy*
  - Observação externa do estado
- Sistemas computacionais
  - *Controlpath e datapath*
  - FSMs comunicantes
  - Exemplo de um multiplicador/divisor iterativo
- Codificações de FSMs alternativas em VHDL

# Máquinas de *Mealy*

- O estilo “2 processos” interdependentes pode facilmente adaptar-se às máquinas de *Mealy*
  - Dependendo da forma como as saídas são atribuídas (no processo combinatório)
    - *Moore* (dependem apenas do estado)
      - Atribuídas diretamente nos **when** de um **case**
    - *Mealy* (dependem do estado e das entradas)
      - Atribuídas em **if...then...else** (dependentes das entradas) dentro dos **when** de um **case**- Cuidado a ter com as saídas
  - Garantir sempre a atribuição (processo combinatório – sem *latches*!!!)
    - Tal como já acontecia no caso de *Moore*!!!



# Exemplo (*Mealy*) – Detetor de Sequência

(1101)

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

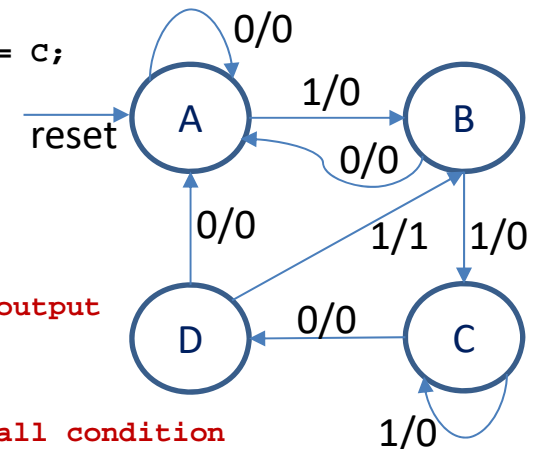
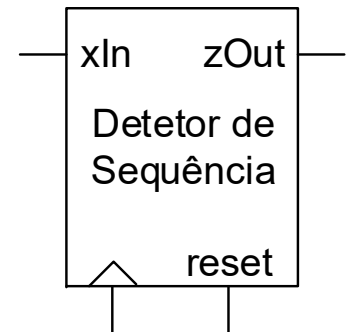
entity Seq1101Detector is
    port(reset : in  std_logic;
          clk   : in  std_logic;
          xIn   : in  std_logic;
          zOut  : out std_logic);
end Seq1101Detector;

architecture Behav of Seq1101Detector is
    type state is (A, B, C, D);
    signal PS, NS : state;
begin
    sync_proc: process(clk)
    begin
        if (rising_edge(clk)) then
            if (reset = '1') then
                PS <= A;
            else
                PS <= NS;
            end if;
        end if;
    end process;
```

## Exemplo:

```
xIn : 0110100101101101011010111010
zOut: 0000100000001001000010000010
```

```
comb_proc : process(PS, xIn)
begin
    zOut <= '0'; -- Frequent output value, could appear
    -- in all "when" statements, but would require more code
    case PS is
        when A =>
            if (xIn = '1') then NS <= B;
            else NS <= A;
            end if;
        when B =>
            if (xIn = '1') then NS <= C;
            else NS <= A;
            end if;
        when C =>
            if (xIn = '1') then NS <= C;
            else NS <= D;
            end if;
        when D =>
            if (xIn = '1') then
                NS <= B;
                zOut <= '1'; -- Mealy output
            else NS <= A;
            end if;
        when others => -- Catch all condition
            NS <= A;
    end case;
end process;
end Behav;
```



# Output do Estado Interno

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity Seq1101 is
  port(reset : in std_logic;
        clk   : in std_logic;
        xIn   : in std_logic;
        stout : out std_logic_vector(3 downto 0);
        zOut  : out std_logic);
end Seq1101;

architecture Behav of Seq1101 is
  type state is (A, B, C, D);
  signal PS, NS : state;
begin
  sync_proc: process(clk)
  begin
    if (rising_edge(clk)) then
      if (reset = '1') then
        PS <= A;
      else
        PS <= NS;
      end if;
    end if;
  end process;

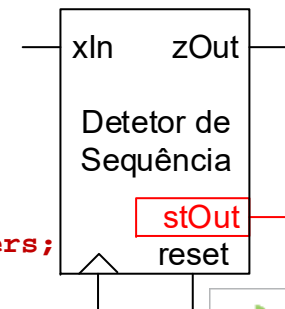
  comb_proc : process(PS, xIn)
  begin
    zOut <= '0'; -- Most frequent output value
```

```
    case PS is
      when A =>
        if (xIn = '1') then NS <= B;
        else NS <= A;
        end if;
        . . .
        . . .
      when D =>
        if (xIn = '1') then
          NS <= B;
          zOut <= '1'; -- Mealy output
        else NS <= A;
        end if;
      when others => -- Catch all condition
        NS <= A;
    end case;
  end process;
```

```
with PS select
  stOut <= "0001" when A,
          "0010" when B,
          "0100" when C,
          "1000" when D,
          "0000" when others;
```

```
end Behav;
```

Atribuição concorrente  
com sync\_proc e  
comb\_proc



# Sistema Computacional

- *Datapath* (unidade de execução)

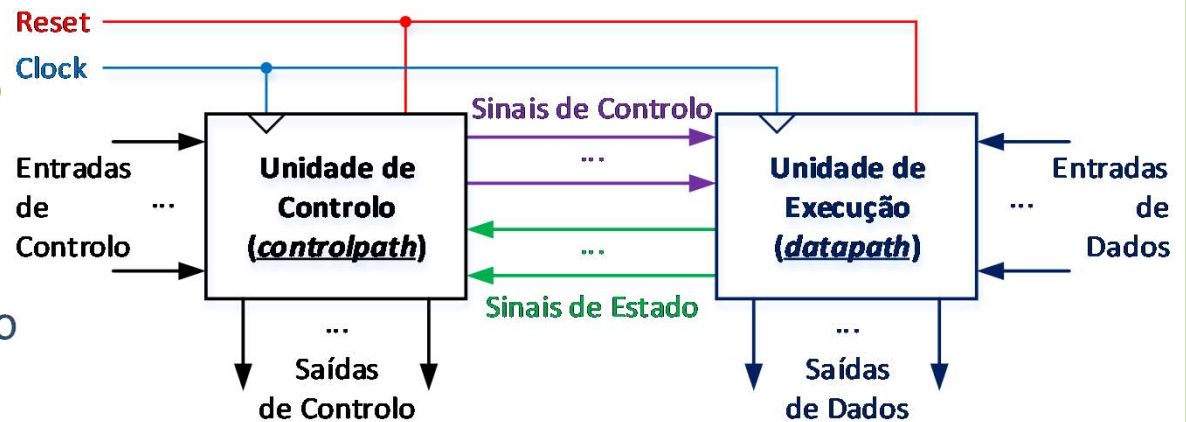
- Elementos

- Operativos
    - Encaminhamento
    - Armazenamento

- *Controlpath*

- Unidade de controlo

- FSM(s)

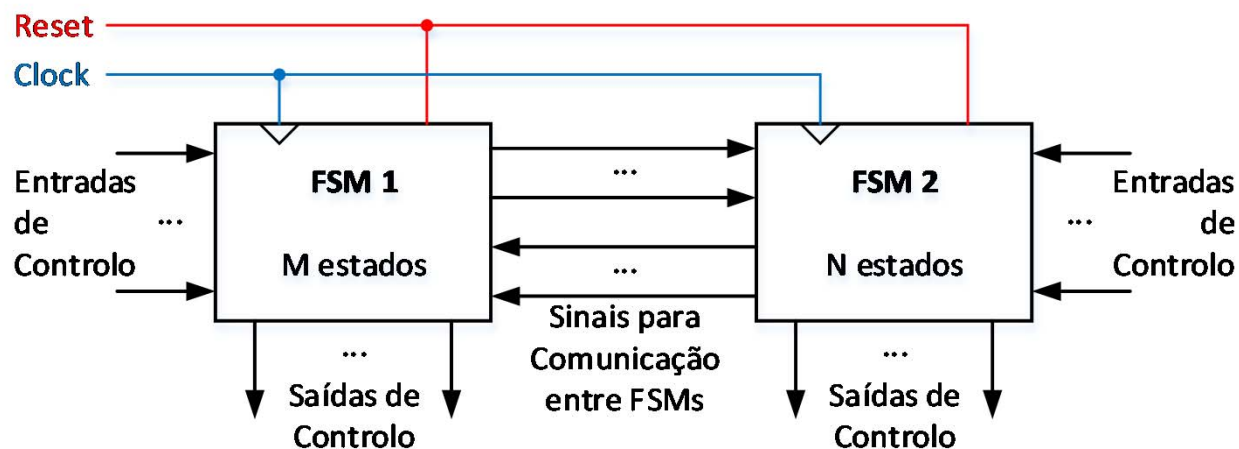


- Interligação entre *controlpath* e *datapath*

- Sinais de controlo (unidade de controlo → unidade de execução)
  - Sinais de estado (unidade de controlo ← unidade de execução)

# Máquinas de Estado Finitos Comunicantes

- Partição da funcionalidade do sistema em duas ou mais máquinas paralelas ou sequenciais
  - Decomposição visa facilitar o desenvolvimento e a validação
  - Partilha dos sinais de inicialização e sincronização
    - Frequentemente operam em diferentes flancos do mesmo sinal de *clock*
  - Sinais de entrada e de saída
    - Específicos de cada sub-máquina
    - Partilhados pelas sub-máquinas
    - Comunicação entre sub-máquinas



# Exemplo de um Sistema Computacional Trivial

- Multiplicador/Divisor *unsigned* de n bits
  - Implementação combinatória/paralela (em VHDL)

```
multResult    <= operand0 * operand1;  
divQuotient   <= operand0 / operand1;  
divRemainder  <= operand0 rem operand1;
```

- Implementação iterativa
  - Determinação do resultado em vários ciclos de relógio
  - Considera um sub-conjunto dos bits dos operandos de entrada em cada ciclo de relógio
- Implementação combinatória vs. iterativa
  - Compromisso entre desempenho/frequência de operação e recursos de implementação



# Multiplicação de Quantidades

## *Unsigned*

- A arquitetura de um multiplicador utiliza, em grande parte, o algoritmo da multiplicação que todos aprendemos a usar na escola primária
- Uma multiplicação que envolva **dois operandos de  $n$  bits** carece de um espaço de armazenamento, para o resultado, de  **$2n$  bits**

$$\begin{array}{r} 0101 \\ \times 0110 \\ \hline 0000 \\ 01010 \\ 010100 \\ +000000 \\ \hline 0011110 \end{array}$$



# Multiplicação de Quantidades

## *Unsigned*

- Esse algoritmo tira partido da propriedade distributiva em relação à adição, permitindo que a multiplicação seja decomposta numa sucessão de somas de produtos parciais
- Considere-se o seguinte produto, em que **M** **representa o multiplicando** e **m** **o multiplicador** representados com 4 bits:  $R = M \cdot m$

$$M \cdot m = M \cdot (m_3 \cdot 2^3 + m_2 \cdot 2^2 + m_1 \cdot 2^1 + m_0 \cdot 2^0)$$

$$M \cdot m = (M \cdot 2^3 \cdot m_3) + (M \cdot 2^2 \cdot m_2) + (M \cdot 2^1 \cdot m_1) + (M \cdot 2^0 \cdot m_0)$$

$$M \cdot m = ((M \cdot 2^3) \cdot m_3) + ((M \cdot 2^2) \cdot m_2) + ((M \cdot 2^1) \cdot m_1) + ((M \cdot 2^0) \cdot m_0)$$

# Multiplicação de Quantidades *Unsigned*

$$M \cdot m = ((M \cdot 2^3) \cdot m_3) + ((M \cdot 2^2) \cdot m_2) + ((M \cdot 2^1) \cdot m_1) + ((M \cdot 2^0) \cdot m_0)$$

- Multiplicar por dois (ou por uma potência de dois) corresponde a deslocar o número multiplicado à esquerda (**shift left**) tantos bits quantos a potência de dois envolvida
- Por outro lado, se  $m_n$  for igual a "0", o produto parcial correspondente também será zero, e se for "1", o mesmo produto parcial será igual ao multiplicando deslocado à esquerda de n bits

```
  0101
x0110
-----
  0000
 01010
010100
+0000000
-----
0011110
```

Para cada bit a "1" do multiplicador, o multiplicando é adicionado ao resultado deslocado à esquerda de um nº de bits igual à posição do "1" no multiplicador

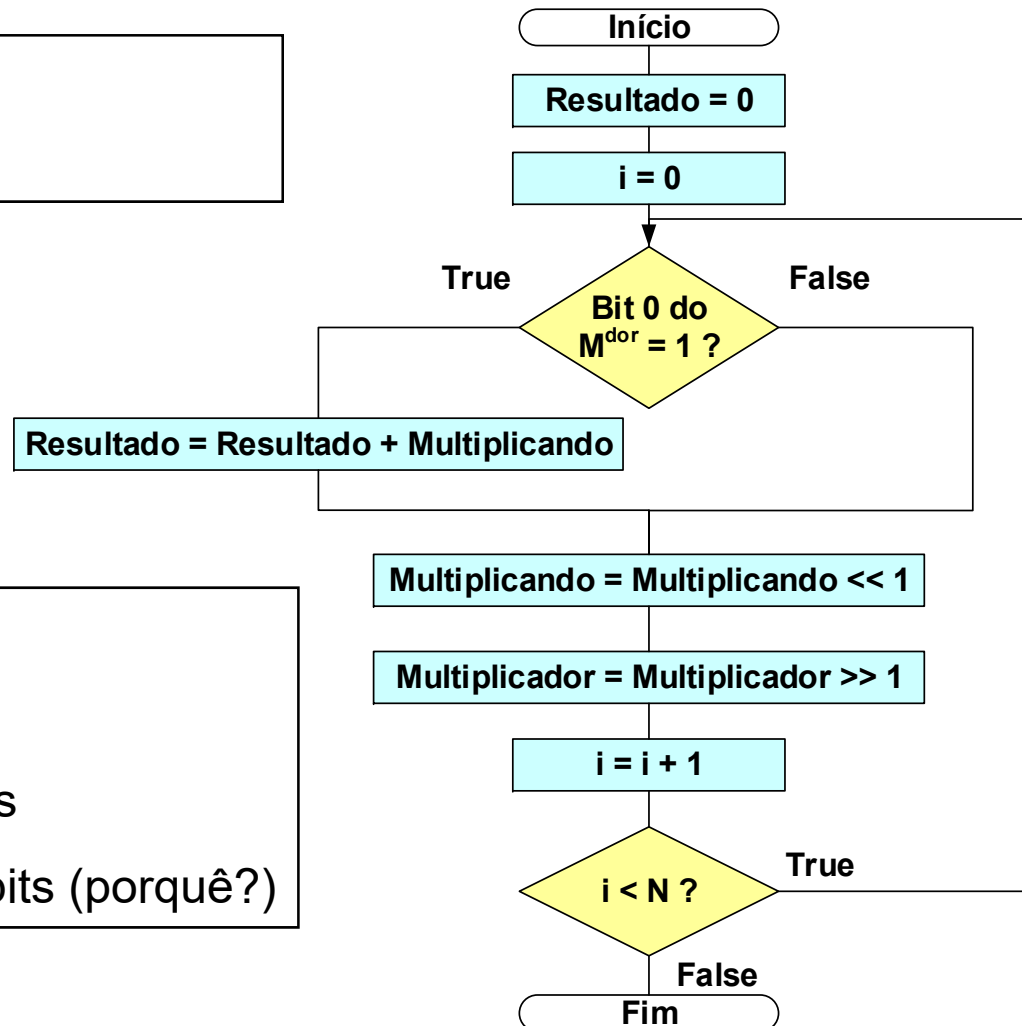
# Algoritmo Iterativo da Multiplicação de Inteiros *Unsigned* de N bits

Operandos: N bits

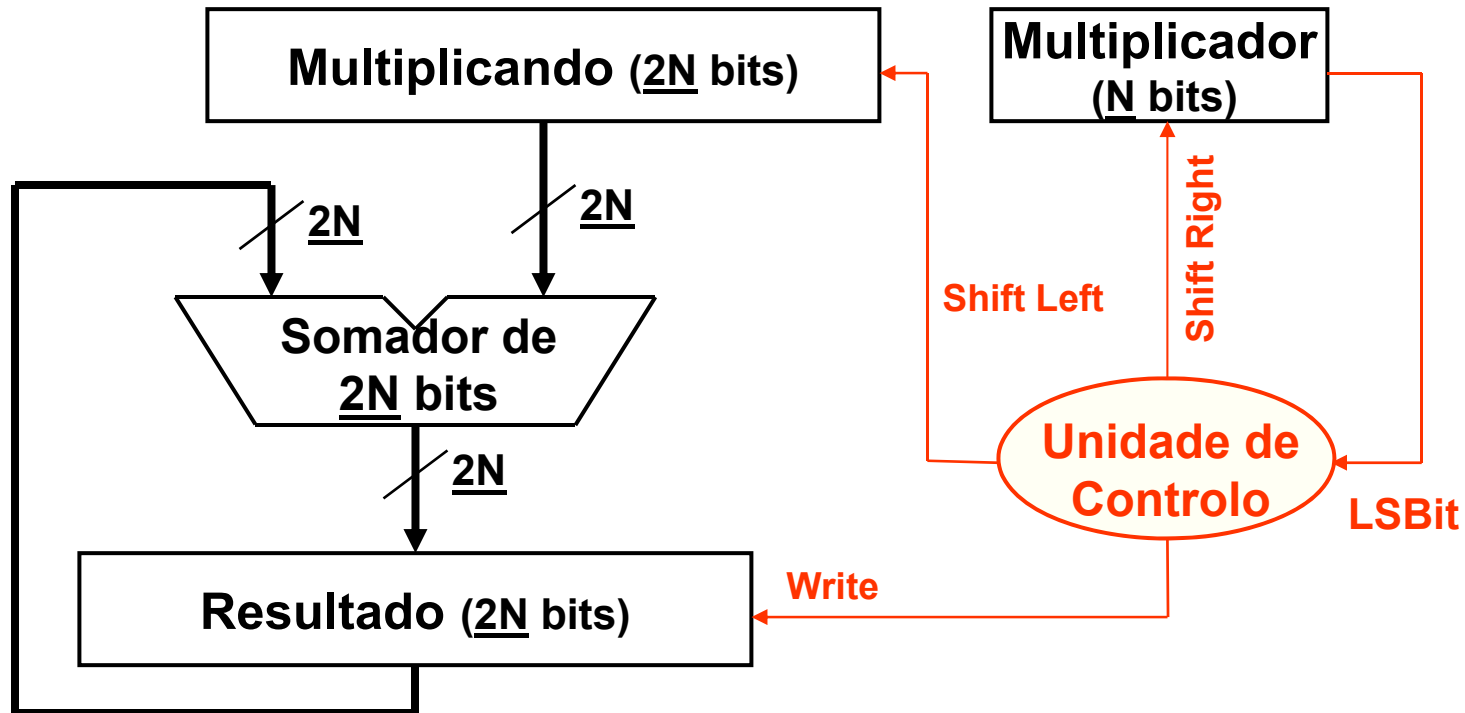
Resultado: 2N bits

Registos necessários:

- **Resultado**: 2N bits
- **Multiplicador**: N bits
- **Multiplicando**: 2N bits (porquê?)



# Arquitetura de um Multiplicador *Unsigned* Iterativo de N bits



- O **Somador** e os registos **Multiplicando** e **Resultado** operam com **2N** bits
- O sinal de relógio não está representado (é implícito) e sincroniza
  - escritas nos registos multiplicando, multiplicador e resultado
  - transições de estado da unidade de controlo

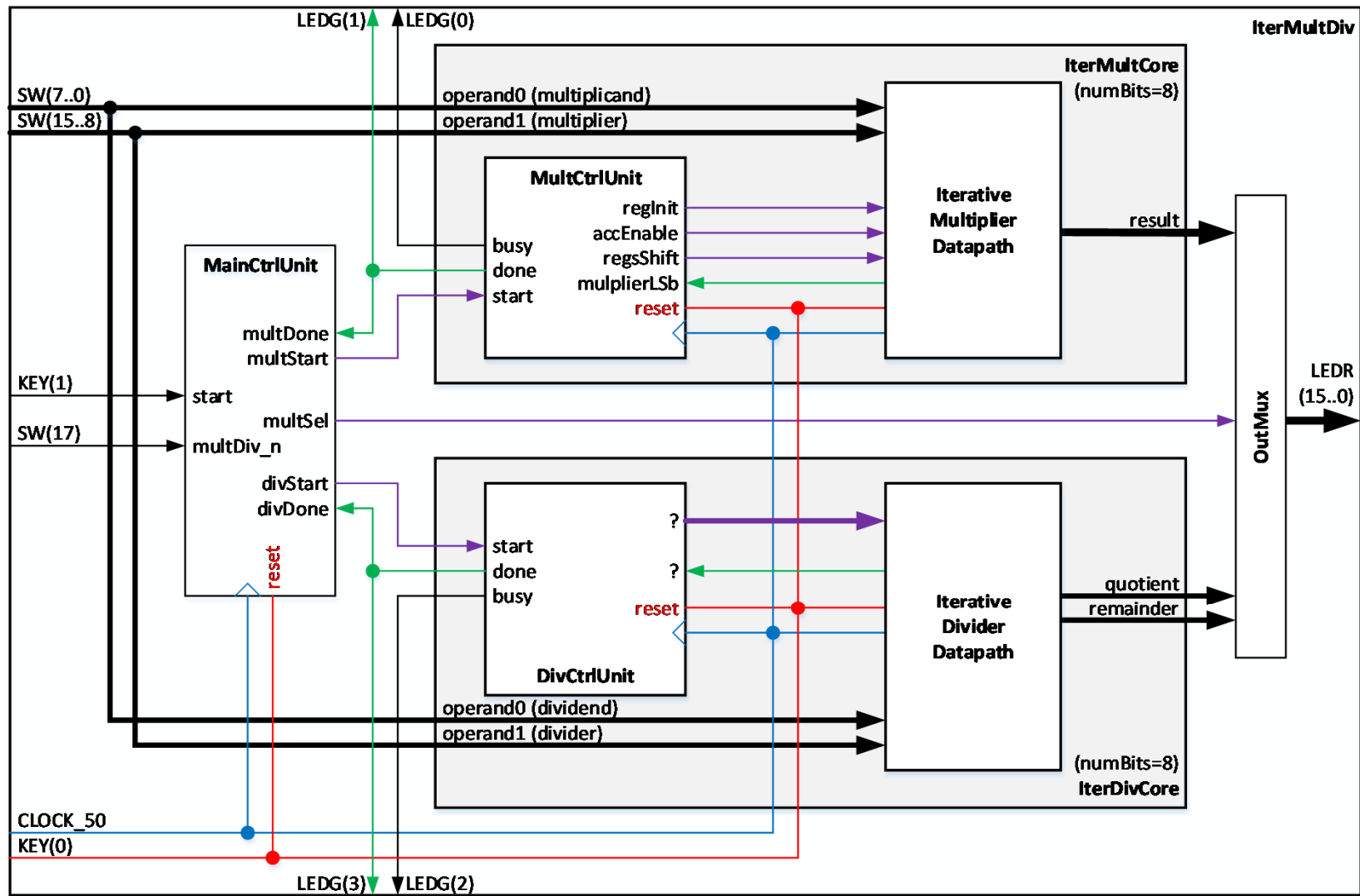
# Multiplicação Iterativa de Inteiros

## *Unsigned* Exemplo com 4 bits

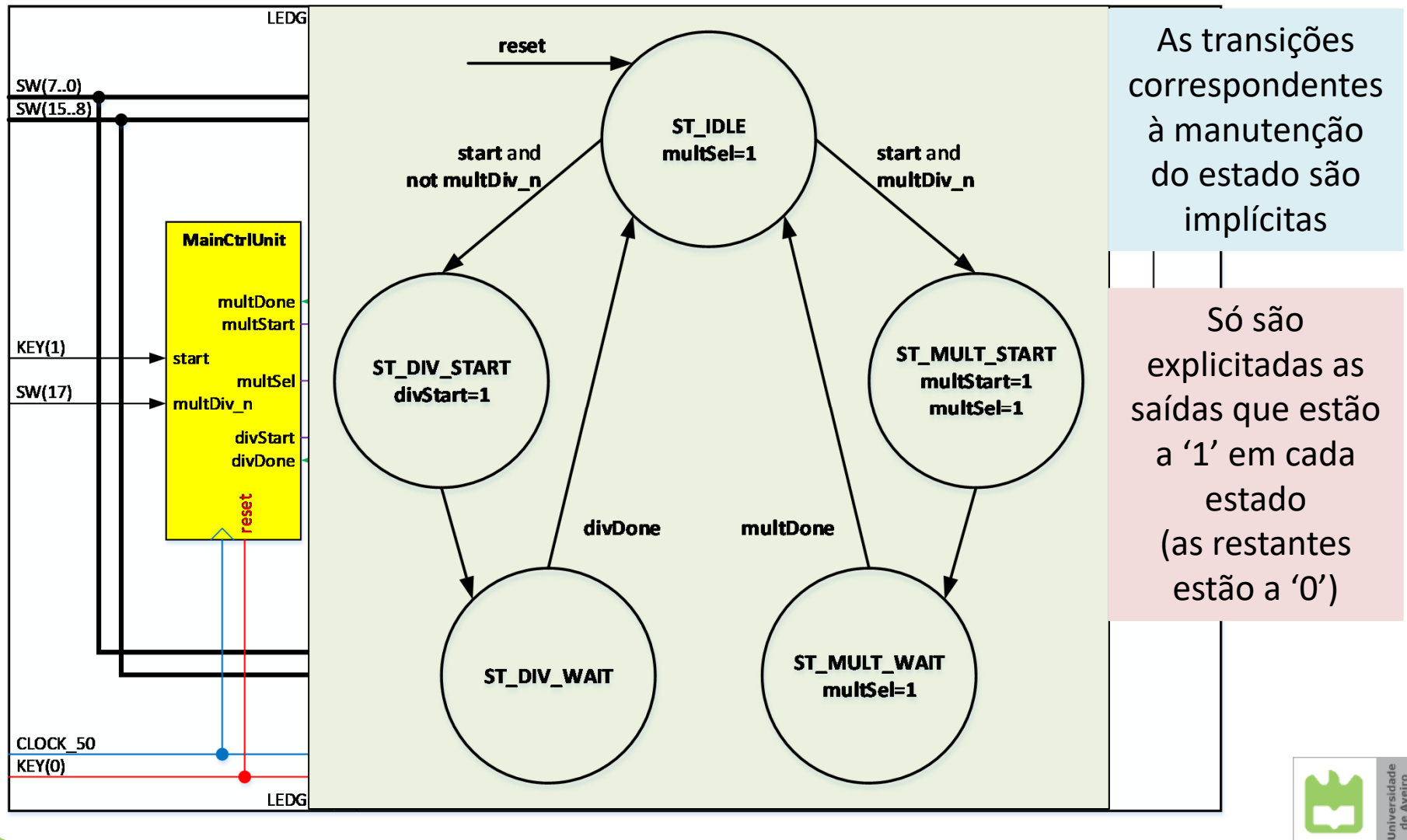
- Com operandos de 4 bits, o resultado terá uma dimensão máxima de 8 bits
- Para a implementação de um multiplicador de 4 bits, que aplique o algoritmo do slide anterior, os registos necessários são:
  - **resultado**: registo de 8 bits
  - **multiplicando**: registo de 8 bits (inicialmente os bits mais significativos são colocados a 0000)
  - **multiplicador**: registo de 4 bits

					<b>0</b>	<b>1</b>	<b>0</b>	<b>1</b>	
			x	<b>0</b>	<b>1</b>	<b>1</b>	<b>0</b>		
<hr/>									
	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>Res. Inicial</b>
<b>+</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0.mdo.<math>2^0</math></b>
<hr/>									
	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	
<b>+</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>1</b>	<b>0</b>	<b>1</b>	<b>0</b>	<b>1.mdo.<math>2^1</math></b>
<hr/>									
	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>1</b>	<b>0</b>	<b>1</b>	<b>0</b>	
<b>+</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>1</b>	<b>0</b>	<b>1</b>	<b>0</b>	<b>0</b>	<b>1.mdo.<math>2^2</math></b>
<hr/>									
	<b>0</b>	<b>0</b>	<b>0</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>0</b>	
<b>+</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0.mdo.<math>2^3</math></b>
<hr/>									
	<b>0</b>	<b>0</b>	<b>0</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>0</b>	
<hr/>									
	<b>0</b>	<b>0</b>	<b>0</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>0</b>	<b>Res. FINAL</b>

# Projeto Exemplo de um Multiplicador/Divisor Iterativo

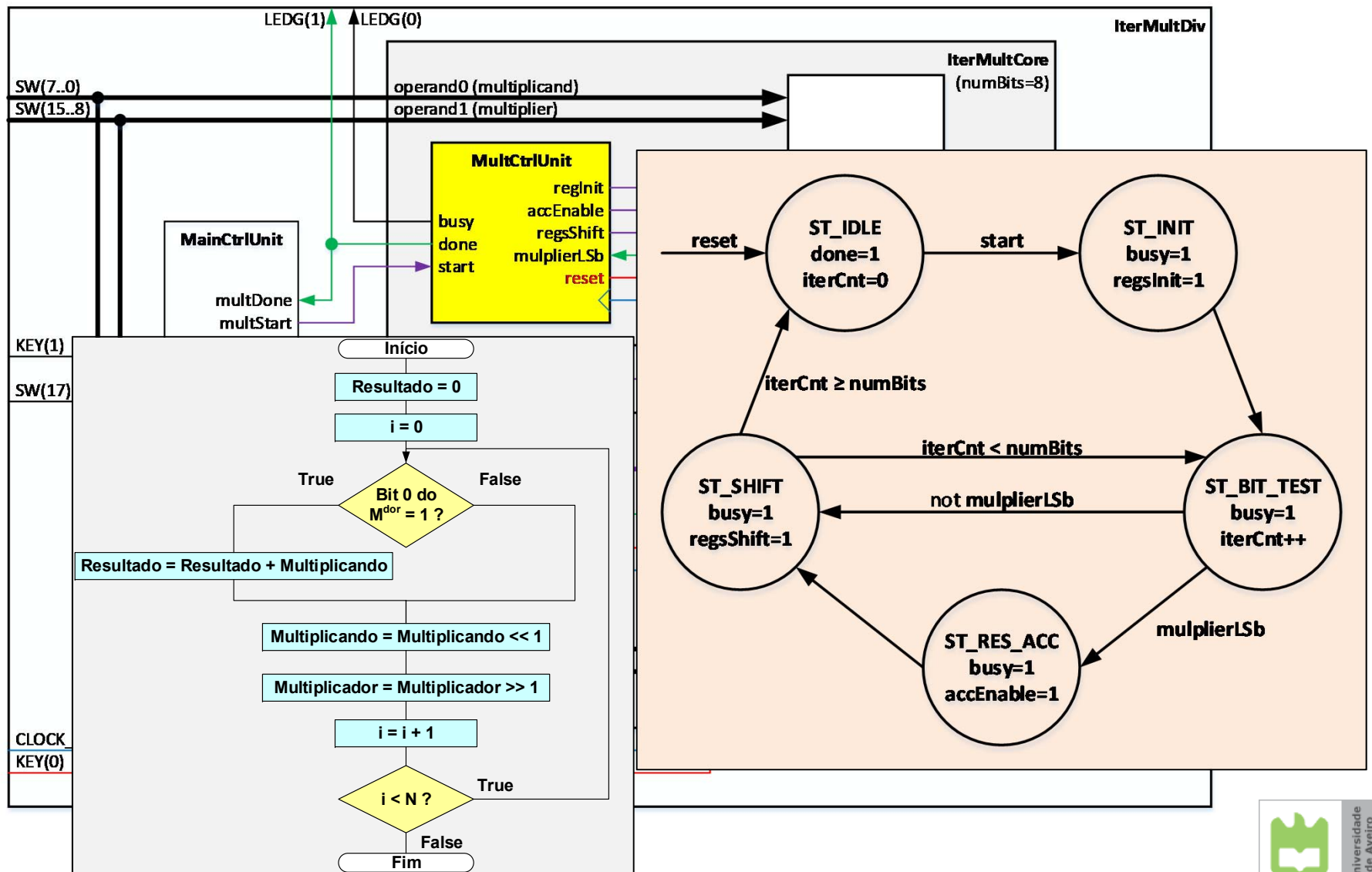


# Unidade de Controlo Principal

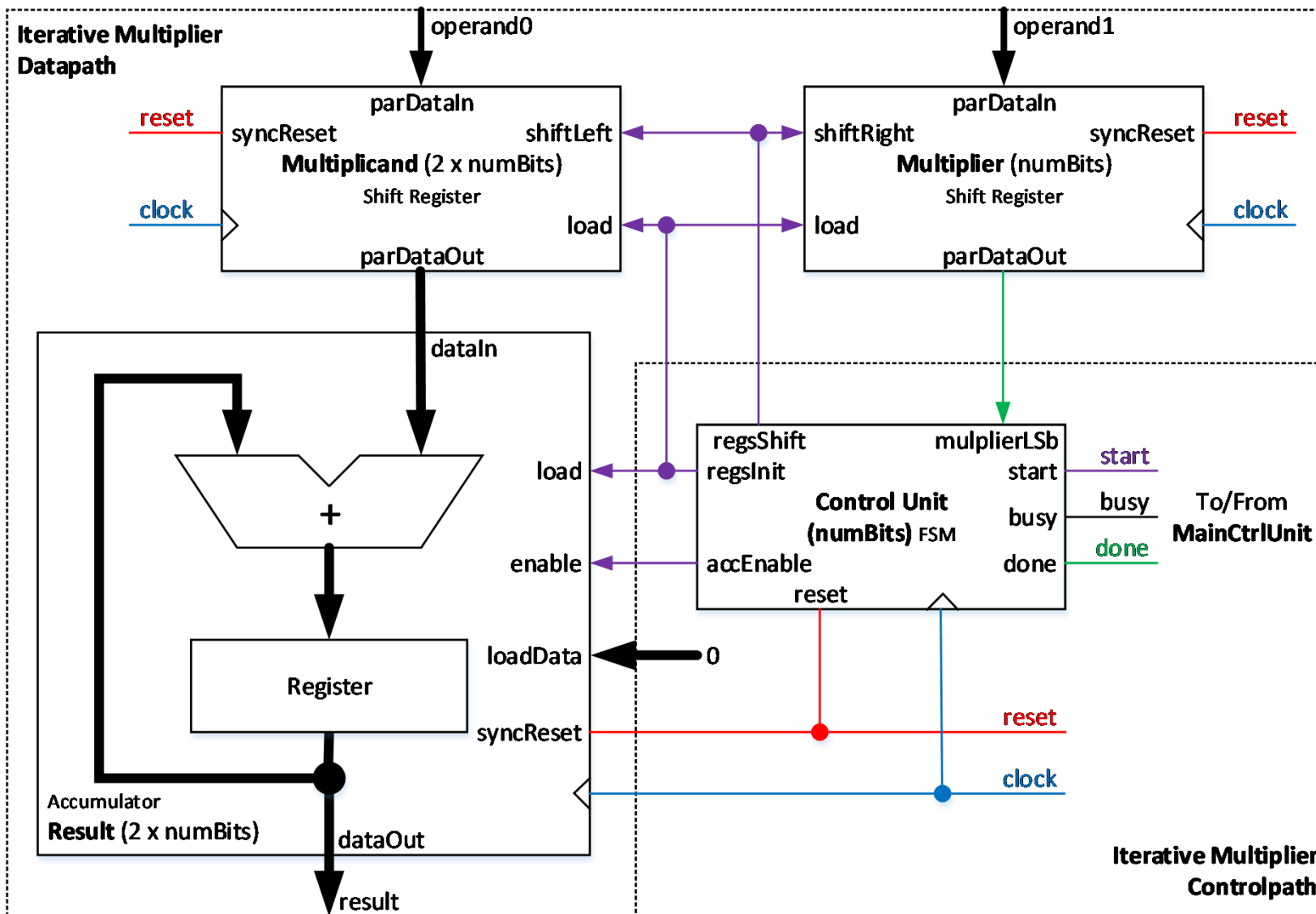




# Unidade de Controlo da Multiplicação



# Controlpath+Datapath do Multiplicador

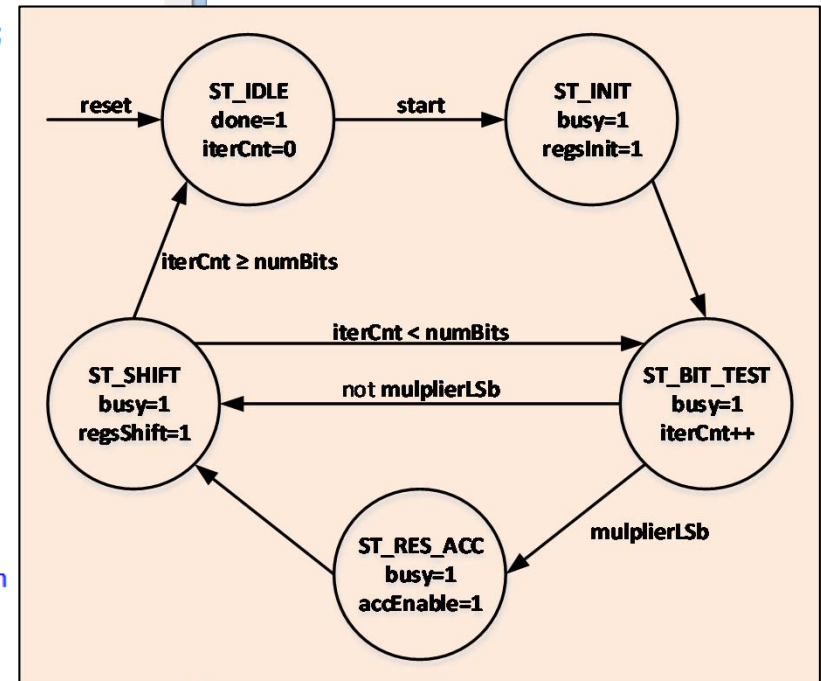


# MultCtrlUnit (FSM com 4 Processos VHDL)

```
188
189 architecture Behavioral_4Proc_AsyncOut of MultCtrlUnit is
190
191     type TState is (ST_IDLE, ST_INIT, ST_BIT_TEST,
192                     ST_RES_ACC, ST_SHIFT);
193     signal s_currentState, s_nextState : TState;
194
195     subtype TCounter is natural range 0 to numBits;
196     signal s_iterCnt : TCounter;
197
198 begin
199     process(clk)
200     begin
201         if (rising_edge(clk)) then
202             if (reset = '1') then
203                 s_currentState <= ST_IDLE;
204             else
205                 s_currentState <= s_nextState;
206             end if;
207         end if;
208     end process;
209
210     process(clk)
211     begin
212         if (rising_edge(clk)) then
213             if (s_currentState = ST_IDLE) then
214                 s_iterCnt <= 0;
215             elsif (s_currentState = ST_BIT_TEST) then
216                 s_iterCnt <= s_iterCnt + 1;
217             end if;
218         end if;
219     end process;
220
```

Processos síncronos  
(registos/variáveis de estado da FSM):

- Estado “convencional”
- Contador

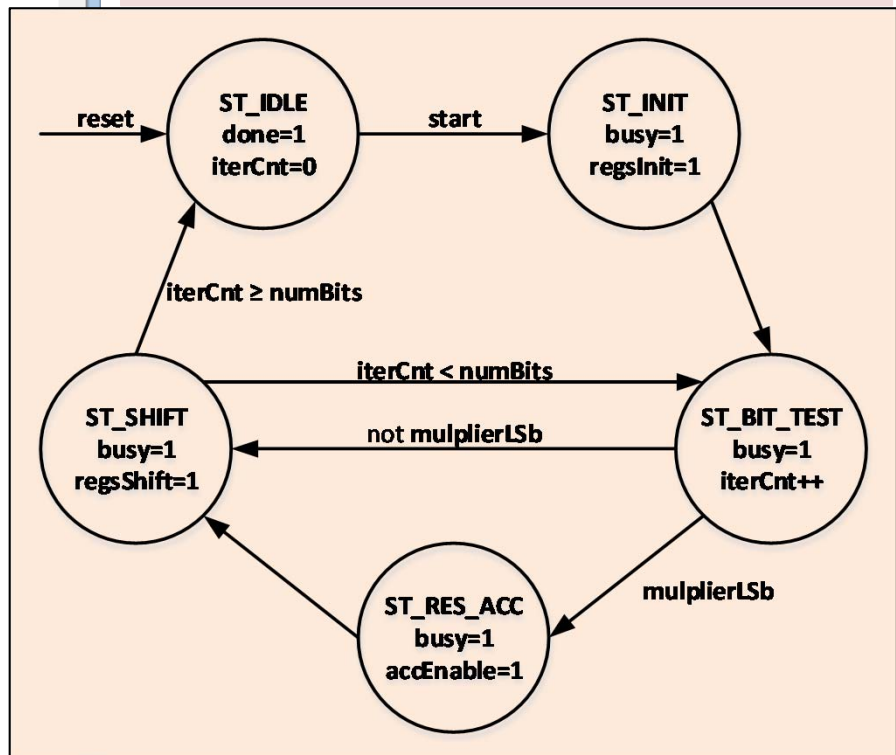


# MultCtrlUnit (FSM com 4 Processos VHDL)

```
Text Editor - C:/Users/asoliveira/CloudStation/LSDig2017/SlidesT...
File Edit View Project Processing Tools Window Help Search altera.com
220
221 process(s_currentState, start, multiplierLsb)
222 begin
223   s_nextState <= s_currentState;
224
225   case s_currentState is
226   when ST_IDLE =>
227     if (start = '1') then
228       s_nextState <= ST_INIT;
229     end if;
230
231   when ST_INIT =>
232     s_nextState <= ST_BIT_TEST;
233
234   when ST_BIT_TEST =>
235     if (multiplierLsb = '1') then
236       s_nextState <= ST_RES_ACC;
237     else
238       s_nextState <= ST_SHIFT;
239     end if;
240
241   when ST_RES_ACC =>
242     s_nextState <= ST_SHIFT;
243
244   when ST_SHIFT =>
245     if (s_iterCnt < numBits) then
246       s_nextState <= ST_BIT_TEST;
247     else
248       s_nextState <= ST_IDLE;
249     end if;
250   end case;
251 end process;
252
```

Circuito combinatório para  
determinação do estado seguinte

Codificação da manutenção de estado:  
**s\_nextState <= s\_currentState**



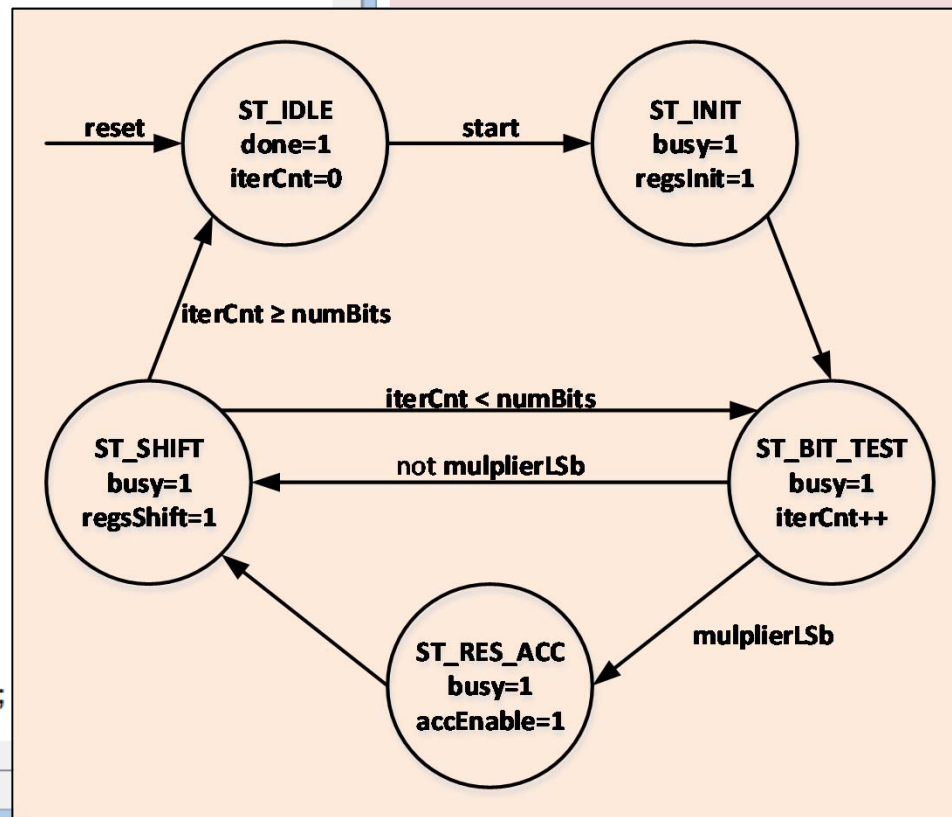
Podem ser usadas expressões Booleanas,  
aritméticas e relacionais nas condições

# MultCtrlUnit (FSM com 4 Processos VHDL)

```
Text Editor - C:/Users/asoliveira/CloudStation/LSDig2017/SlidesT...
File Edit View Project Processing Tools Window Help Search altera.com
252
253 process(s_currentState)
254 begin
255     busy      <= '0';
256     done      <= '0';
257     regsInit  <= '0';
258     accEnable <= '0';
259     regsShift <= '0';
260
261     case s_currentState is
262     when ST_IDLE =>
263         done      <= '1';
264
265     when ST_INIT =>
266         busy      <= '1';
267         regsInit  <= '1';
268
269     when ST_BIT_TEST =>
270         busy      <= '1';
271
272     when ST_RES_ACC =>
273         busy      <= '1';
274         accEnable <= '1';
275
276     when ST_SHIFT =>
277         busy      <= '1';
278         regsShift <= '1';
279     end case;
280 end process;
281 end Behavioral_4Proc_AsyncOut;
282
```

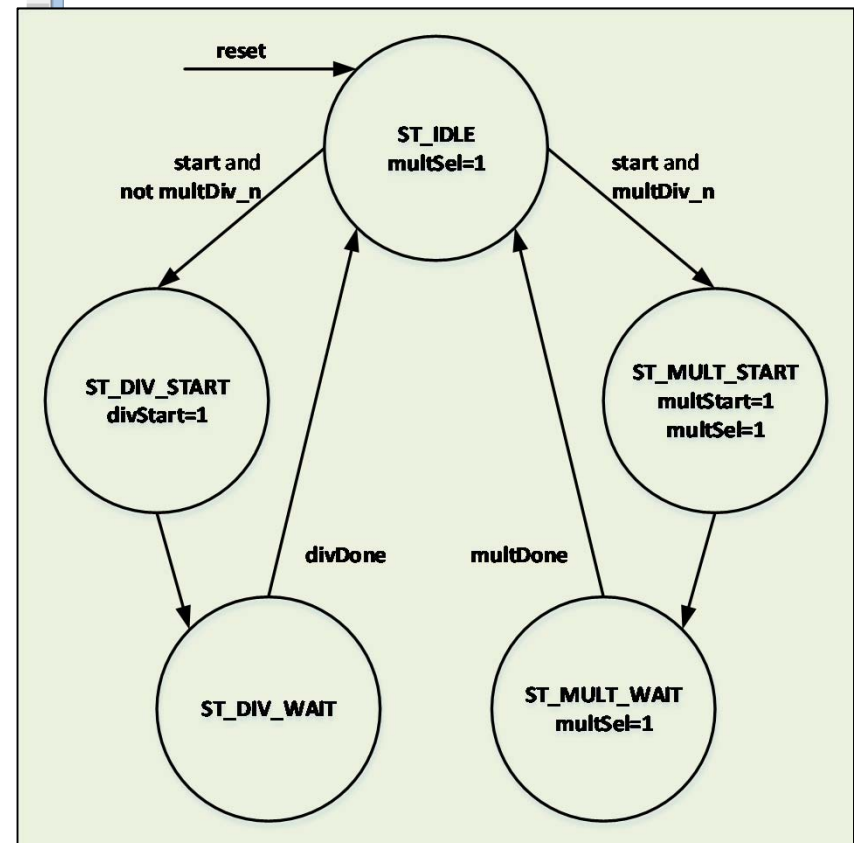
Circuito combinatório para  
determinação das saídas

Por omissão todas as saídas são '0'  
(caso mais frequente); explicitadas  
nos estados em que são '1'



# MainCtrlUnit (FSM codificada com 1 Processo VHDL)

```
17 architecture Behavioral of MainCtrlUnit is
18
19     type TState is (ST_IDLE, ST_MULT_START, ST_MULT_WAIT,
20                     ST_DIV_START, ST_DIV_WAIT);
21     signal s_state : TState;
22
23 begin
24     process(clk)
25     begin
26         if (rising_edge(clk)) then
27             if (reset = '1') then
28                 s_state <= ST_IDLE;
29                 multStart <= '0';
30                 divStart <= '0';
31                 multSel <= '1';
32             else
33                 case s_state is
34                     when ST_IDLE =>
35                         if (start = '1') then
36                             if (multDiv_n = '1') then
37                                 s_state <= ST_MULT_START;
38                                 multStart <= '1';
39                                 divStart <= '0';
40                                 multSel <= '1';
41                             else
42                                 s_state <= ST_DIV_START;
43                                 multStart <= '0';
44                                 divStart <= '1';
45                                 multSel <= '0';
46                             end if;
47                         end if;
48                     end if;
49
50                     when ST_MULT_START =>
51                         s_state <= ST_MULT_WAIT;
52                         multStart <= '0';
53                         divStart <= '0';
54                         multSel <= '1';
55
56                     when ST_MULT_WAIT =>
57                         if (multDone = '1') then
58                             s_state <= ST_IDLE;
59                             multStart <= '0';
60                             divStart <= '0';
61                             multSel <= '1';
62                         end if;
63
```





# Comentários Finais

- No final desta aula e do trabalho prático 9 de LSD, deverá ser capaz de:
  - Conhecer os passos necessários para a modelação, simulação e síntese de máquinas de estados finitos
  - Recorrer a descrições comportamentais VHDL próximas do diagrama de estados saídas
    - Modelo de *Mealy*
    - Modelo de *Moore*
  - Conceber *testbenches* para a simulação funcional das FSMs
  - Desenvolver sistemas computacionais constituídos por um *controlpath* (com uma ou mais FSMs) e um *datapath*
- ... bom trabalho prático 9, disponível no site da UC
  - [elearning.ua.pt](http://elearning.ua.pt)