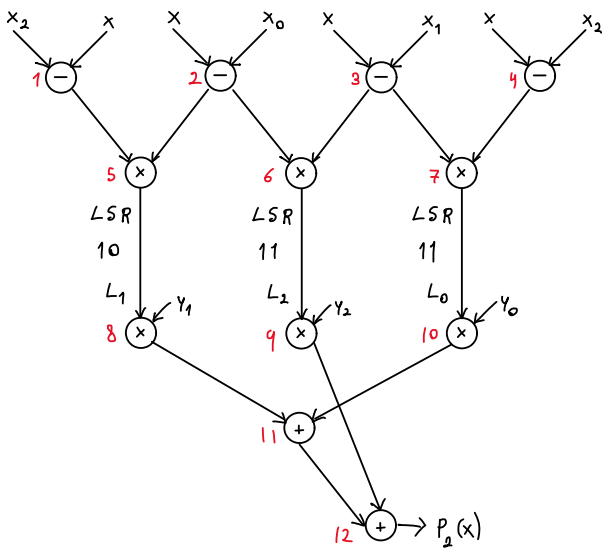
1)



$(x_0 - x_1)(x_0 - x_2) = (-32)(-64) = 2048 = 2^{11}$

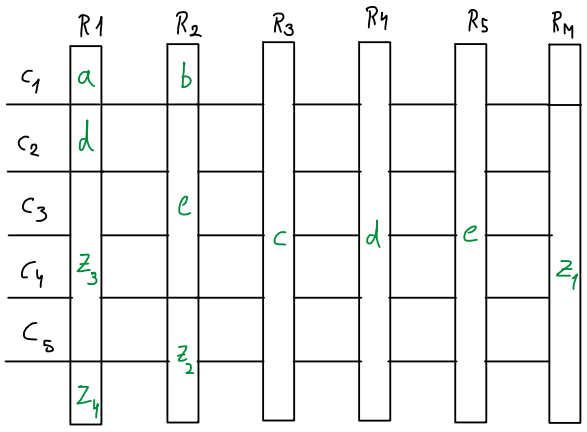
$(x_1 - x_0)(x_1 - x_2) = (32)(-32) = -1024 = -2^{10}$

$(x_2 - x_0)(x_2 - x_1) = (64)(32) = 2048 = 2^{11}$

1 (5) −
2 (5) −
3 (5) −
4 (5) −
5 (4) × ⇐ 1,2
6 (4) × ⇐ 2,3
7 (4) × ⇐ 3,4
8 (3) × ⇐ 5
9 (3) × ⇐ 6
10 (3) × ⇐ 7
11 (2) + ⇐ 8,10
12 (1) + ⇐ 11,9

Ciclo 1	1 (−) 2 (−)
Ciclo 2	3 (−) 4 (−) 5 (×)
Ciclo 3	6 (×)
Ciclo 4	7 (×)
Ciclo 5	8 (×)
Ciclo 6	10 (×)
Ciclo 7	11 (+) 9 (×)
Ciclo 8	12 (+)

2)



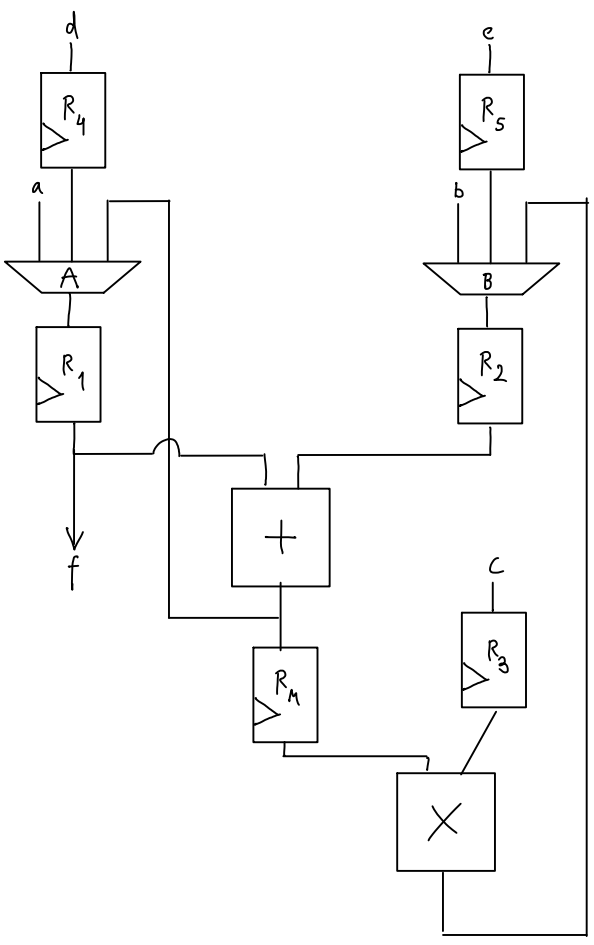
$z_1 = a + b$

$z_2 = c(a+b) = c \cdot z_1$

$z_3 = d + e$

$z_4 = c(a+b) + d + e = z_2 + z_3 \Rightarrow Res$

	En_{R1}	En_{R2}	En_{R3}	En_{R4}	En_{R4}	En_{RM}	Sel A	Sel B	Estado
Init	1	1	1	1	1	–	00	00	111000
c_1	1	1	0	—	–	1	01	01	110010
c_2	1	–	0	—	–	0	10	10	100100
c_3	0	–	0	—	–	0	10	10	000100
c_4	0	1	–	—	–	–	10	10	010100
c_5	1	–	–	—	–	–	10	10	110100
done	0	–	–	—	–	–	–	–	010101
	b5	b4	b3	b4	b4	b4	b2 b1		Valid



Registos : 6

4:1 MUX : 2

$A = Add + Mul + 6 \times Reg + 2 \times Mux\ 4:1 =$

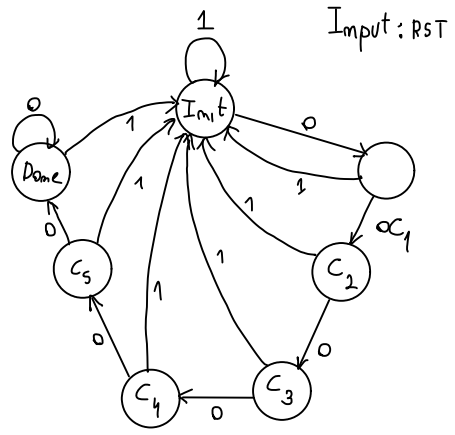
$= 16 + 120 + 6 \times 10 + 2 \times 32 =$

$= 260$

$t_{clk\,min} = t_{P_{Add}} + t_{P_{Mux}} + t_{P_{Reg}} + t_{set} = 5 + 4 + 2 + 1 = 12\,ms$

Não é pelo Mult pois este está distribuido por 3 ciclos!

Latencia → $5 \times 12\,ms = 60\,ms$



3)

$$Y_K = \sum_{i=0}^{\frac{N-1}{2}} h_i(X_i + X_{N-i})$$

$N = 31$

$x_i = 8\,bits \rightarrow Q_{5.3}$

$h_i = 16\,bits \rightarrow Q_{2.14}$

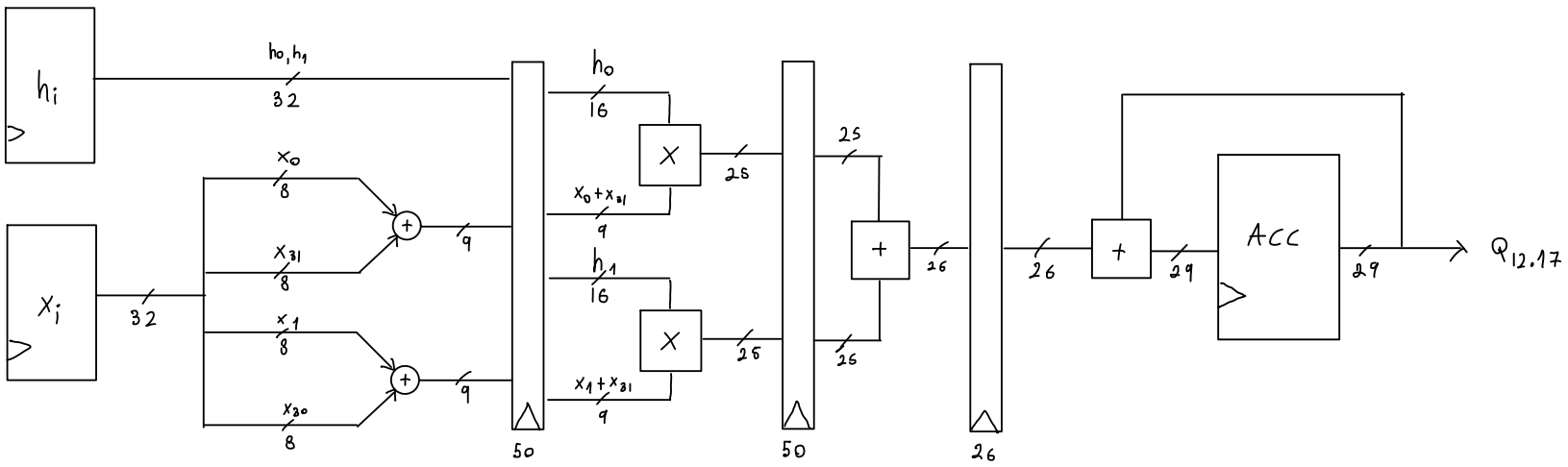
Simple dual port → 512 x 36 (512 linhas de 36 bits)
↳ Só lemos um port de cada vez

a) $Y_K = Q_{5.3} + Q_{2.14} + \log_2(16) + 1 = Q_{12.17}$

h_i: Basta guardar h_0 a h_{15} (é simétrico)
↳ h_i → 16 bits ⇝ 2 x h_i por linha (16 linhas) → 1 BRAM

x_i: Guardar x_0 a x_{31}
↳ x_i → 8 bits ⇝ 4 x x_i por linha (16 linhas) → 1 BRAM

$x_0\ x_{31}\ x_1\ x_{30}$
$x_2\ x_{29}\ x_3\ x_{28}$
...




```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity addm is
	Port (
	hi : in std_logic_vector ( 15 downto 0);
	xi0, xi1 : in std_logic_vector ( 7 downto 0);
	pi : out std_logic_vector ( 24 downto 0 )
	);
end addm;

architecture behavioral of addm is
signal Sigxi0, Sigxi1 : signed(7 downto 0);
signal Sighi : signed( 15 downto 0);
signal xAux : signed(8 downto 0);
signal pAux : signed(24 downto 0);

begin

Sigxi0 <= signed(xi0);
Sigxi1 <= signed(xi1);

xAux <= Sigxi0 + Sigxi1;
pAux <= xAux * Sighi;

pi <= std_logic_vector(pAux);

end behavioral;
```

b) Throughput → 2 MAC's por ciclo

Latency:

$$N^{\circ}ciclos = 3 + \frac{16}{2} = 11$$

$$t_{clk} = t_{p_{Mul}} + t_{p_{Reg}} + t_{Set} = 14 + 2 + 1 = 17\,ns$$

$$Latency = N^{\circ}ciclos \times t_{clk} = 187\,ns$$

e) Apenas 1 DSP

- 25bit pre-adder => aguenta com $x_i + x_{N-i}$
- 25 x 18 multiplier => aguenta com $h_i(x_1 + x_{N-i})$

4 a) $t_{Min} = t_{P_{FF}} + t_{SET} + Max\{t_{P_{Log}}\} = 4 + 2 + 11 = 17\,ns$

b) $\delta = t_{p\,buffer}$

$$t'_{Min} = t_{Min} - \delta = 14\,ns$$

c) $t'_{Hold} < t_{P_{FF}} + Min\{t_{P_{Log}}\} - \delta \iff t'_{Hold} < 4 + 2 - 3 \iff t'_{Hold} < 3\,ns$

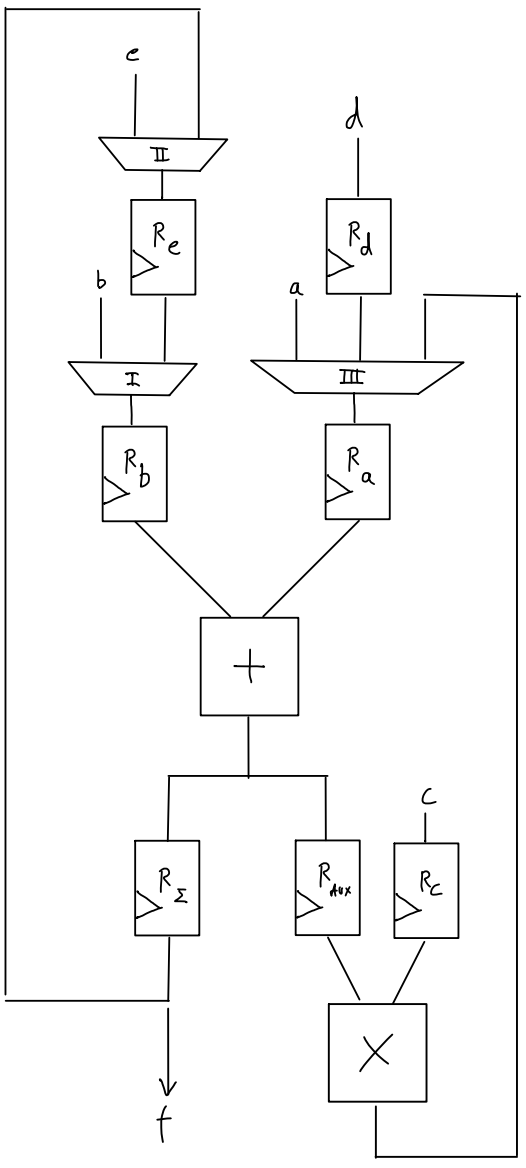
$$WHS = t'_{Hold} - t_{Hold} = 2\,ns$$

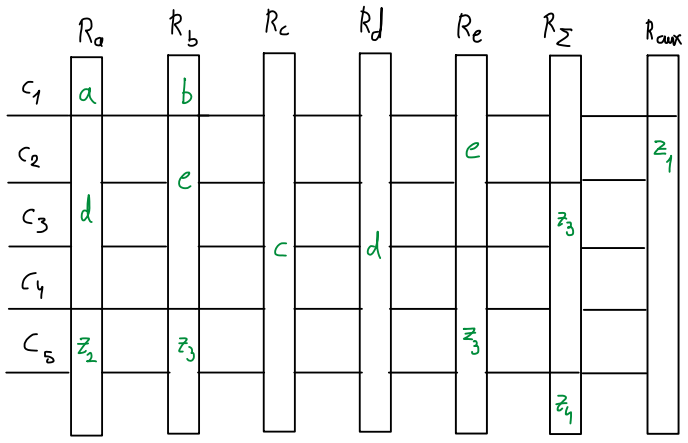
5 1 LUTRAM → 64 bits

$$\left\lceil \frac{112 \times 2}{64} \right\rceil = 4\ LUTs$$

Solução alternativa

- Como o mul está dividido por 3 ciclos o melhor possível é só passar por um somador!





$z_1 = a + b$

$z_2 = c(a+b) = c \cdot z_1$

$z_3 = d + e$

$z_4 = c(a+b) + d + e = z_2 + z_3 \Rightarrow Res$

Registos : 7

Mux 4:1 : 1

Mux 2:1 : 2

$A = Add + Mul + 7 \times Reg + 2 \times Mux\,2:1 + Mux\,4:1 =$

$= 16 + 120 + 70 + 2 \times 16 + 32$

$= 270$

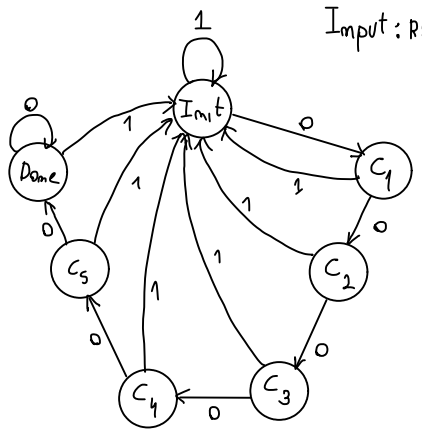
$t_{clk_{mim}} = t_{p_{Add}} + t_{p_{Reg}} + t_{set} = 5 + 2 + 1 = 8\,ms$

Não é pelo Mult pois este está distribuido por 3 ciclos!

$t_{p_{MUX}} + t_{p_{Mul}} + t_{p_{Reg}} + t_{set} = 4 + 14 + 2 + 1 = 21\,ms$

$\frac{21}{3} = 7ms \quad 7 < 8ms$! Dá !!

Latencia $\rightarrow 5 \times 8\,ms = 40\,ms$



	em_{R_a}	em_{R_b}	em_{R_e}	em_{R_Σ}	$em_{R_{aux}}$	Sel I	Sel II	Sel III	Estado
Init	1	1	1	—	—	0	0	00	11111 00000
c_1	1	1	0	0	1	1	—	01	11001 10010
c_2	—	—	—	1	0	—	—	—	11110 00000
c_3	—	—	1	—	0	—	1	—	11100 01110
c_4	1	1	—	—	0	1	—	10	11110 11100
c_5	—	—	—	1	—	—	—	—	11111 11111
done	—	—	—	0	—	—	—	—	00000 00000