



UNIVERSIDADE CATÓLICA DE PELOTAS
ENGENHARIA DE COMPUTAÇÃO
Projetos de Circuitos Integrados

Gabriel Harter Zoppo

Desenvolvimento de um filtro FIR semi-paralelo

Pelotas, 30 de abril de 2022

1. Introdução:

Este projeto é referente ao segundo trabalho da primeira avaliação da disciplina de projeto de circuitos integrados ministrada pelo professor Eduardo Antonio Cesar da Costa e tem como objetivo desenvolver um filtro FIR semi-paralelo com 8 taps e 8 Bits na forma direta usando como base o código totalmente sequencial dado em aula.

O projeto foi totalmente elaborado e compilado na linguagem de descrição de hardware (VHDL) e compilada no programa *Quartus II*, um software de design de dispositivo lógico programável e o *modelsim*, um software de simulação dos códigos VHDL possibilitando ver os resultados forma gráfica.

A arquitetura do código utilizado no trabalho é mostrado na figura 1 e possui uma máquina de estados com um clock, um clear, uma carga, um apontador e dois seletores. Temos um registrador de deslocamento de 8 Bits, dois multiplexadores de 8 Bits, duas Roms, dois multiplicadores de 8 Bits, dois somadores de 16 Bits, um registrador de 16 Bits.

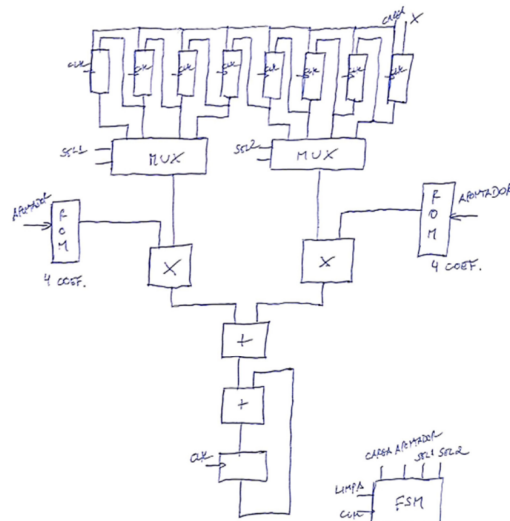


Figura 1: Filtro FIR semi-paralelo

2. Desenvolvimento:

Para o desenvolvimento do filtro FIR semi-paralelo devem ser declarados primeiramente os elementos utilizados nesse filtro, começa-se pelos dois somadores de 16 Bits, na qual as duas entradas e a saída são de 16 Bits conforme a figura 2.

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3  USE ieee.std_logic_unsigned.all;
4
5  ENTITY somador16bits IS
6  PORT ( a, b : IN STD_LOGIC_VECTOR (15 DOWNTO 0);
7        s : OUT STD_LOGIC_VECTOR (15 DOWNTO 0));
8  END somador16bits;
9
10 ARCHITECTURE dataflow OF somador16bits IS
11
12 BEGIN
13
14   s <= a + b;
15
16 END dataflow;
```

Figura 2: Somador de 16 Bits

O próximo passo é declarar o multiplicador de 8 Bits, na qual as duas entradas são de 8 Bits e a saída é de 16 Bits, após é declarado o registrador de 16 Bits com entrada e saída de 16 Bits, ambos elementos podem ser vistos na Figura 3.

```
22 ENTITY mult8bits IS
23 PORT ( a, b : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
24       s : OUT STD_LOGIC_VECTOR (15 DOWNTO 0));
25 END mult8bits;
26
27 ARCHITECTURE dataflow1 OF mult8bits IS
28
29 BEGIN
30
31   s <= a * b;
32
33 END dataflow1;
34
35 LIBRARY ieee;
36 USE ieee.std_logic_1164.all;
37
38 ENTITY reg16b IS
39 PORT ( clk : IN STD_LOGIC;
40       D : IN STD_LOGIC_VECTOR (15 DOWNTO 0);
41       Q : OUT STD_LOGIC_VECTOR (15 DOWNTO 0));
42 END reg16b;
43
44 ARCHITECTURE comportamento OF reg16b IS
45 BEGIN
46 PROCESS (clk)
47 BEGIN
48 IF clk'EVENT AND clk = '1' THEN
49   Q <= D;
50 END IF;
51 END PROCESS;
52 END comportamento;
```

Figura 3: Multiplicador e registrador de 16 Bits

Depois do registrador temos o multiplicador de 8 bits temos o multiplexador e um registrador de 8 Bits, o multiplexador possui 4 entradas de 8 bits, seletor de 2 Bits e uma saída de 8 Bits, já o registrador tem entrada e saída de 8 Bits conforme mostrado na figura 4.

```

54  LIBRARY ieee;
55  USE ieee.std_logic_1164.all;
56
57  ENTITY mux4para2 IS
58  PORT ( sel: IN STD_LOGIC_VECTOR (1 downto 0);
59        a, b, c, d: IN STD_LOGIC_VECTOR (7 downto 0);
60        Y : OUT STD_LOGIC_VECTOR (7 downto 0)
61        );
62  END mux4para2;
63
64  ARCHITECTURE dataflow OF mux4para2 IS
65  BEGIN
66  PROCESS (sel) -- lista de sensibiliza  o
67  BEGIN
68  CASE sel IS
69  WHEN "00" => Y <= a;
70  WHEN "01" => Y <= b;
71  WHEN "10" => Y <= c;
72  WHEN "11" => Y <= d;
73  END CASE;
74  END PROCESS;
75  END dataflow;
76
77  LIBRARY ieee;
78  USE ieee.std_logic_1164.all;
79  ENTITY regdes8b IS
80  PORT ( clk, ld : IN STD_LOGIC;
81        D : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
82        Q : OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
83  END regdes8b;
84
85  ARCHITECTURE comportamento OF regdes8b IS
86  BEGIN
87  PROCESS (ld, clk)
88  BEGIN
89
90  IF clk'EVENT AND clk = '1' THEN
91
92  IF ld = '1' THEN
93
94  Q <= D;
95
96  END IF;
97
98  END IF;
99
100 END PROCESS;
101
102 END comportamento;

```

Figura 4: Multiplexador e registrador de de 8 Bits.

Abaixo temos as duas roms que possuem 4 valores de 8 Bits cada, um apontador para quatro entradas conforme mostrado na figura 5. Os valores foram divididos nas duas roms e cada um possuem 4 deles, cada um possui um endere o que ser  definido apartir da posi  o do apontador.

```

104  LIBRARY ieee;
105  USE ieee.std_logic_1164.all;
106
107  ENTITY rom IS
108  GENERIC ( bits: INTEGER := 8; -- # of bits per word
109           words: INTEGER := 4); -- # of words in the memory
110  PORT ( addr: IN INTEGER RANGE 0 TO words-1;
111        data: OUT STD_LOGIC_VECTOR (bits-1 DOWNTO 0));
112  END rom;
113
114  ARCHITECTURE rom OF rom IS
115  TYPE vector_array IS ARRAY (0 TO words-1) OF
116  STD_LOGIC_VECTOR (bits-1 DOWNTO 0);
117  CONSTANT memory: vector_array := ( "00000001",
118                                     "00000010",
119                                     "00000100",
120                                     "00001000");
121
122  BEGIN
123  data <= memory(addr);
124  END rom;
125
126  LIBRARY ieee;
127  USE ieee.std_logic_1164.all;
128
129  ENTITY rom1 IS
130  GENERIC ( bits: INTEGER := 8; -- # of bits per word
131           words: INTEGER := 4); -- # of words in the memory
132  PORT ( addr: IN INTEGER RANGE 0 TO words-1;
133        data: OUT STD_LOGIC_VECTOR (bits-1 DOWNTO 0));
134  END rom1;
135
136  ARCHITECTURE rom1 OF rom1 IS
137  TYPE vector_array IS ARRAY (0 TO words-1) OF
138  STD_LOGIC_VECTOR (bits-1 DOWNTO 0);
139  CONSTANT memory: vector_array := ( "00010000",
140                                     "00100000",
141                                     "01000000",
142                                     "10000000");
143
144  BEGIN
145  data <= memory(addr);
146  END rom1;

```

Figura 5: Duas Rom de 8 Bits

No final precisamos fazer a máquina de estado que vai direcionar o filtro, precisa-se de um clock, um clear, uma carga, um seletor de 2 Bits e um apontador com 4 de tamanho, e cada um dos estados será lido um valor da rom. O código dessa máquina de estado está na figura 6 e 7.

```

148 LIBRARY ieee;
149 USE ieee.std_logic_1164.all;
150
151 ENTITY mef IS
152 PORT (clk,clr: IN STD_LOGIC;
153       ld: OUT STD_LOGIC;
154       C: OUT STD_LOGIC_VECTOR (1 downto 0); -- selecao
155       app: OUT INTEGER RANGE 0 TO 3 -- Apontador
156       );
157 END mef;
158
159 ARCHITECTURE Behave OF mef IS
160 TYPE estados is (n0, n1, n2, n3);
161 SIGNAL estado:estados;
162
163 BEGIN
164
165 PROCESS(clk,clr)
166 BEGIN
167
168 IF clr ='0'then
169 estado <= n0;
170
171 ELSE
172 IF (clk 'EVENT AND clk = '1') then

```

Figura 6: Máquina de estados

```

174 CASE estado is
175 WHEN n0 =>
176 estado<=n1;
177 ld <= '1';
178 C<= ("00");
179 app <= 0;
180 WHEN n1 =>
181 estado<=n2;
182 ld <= '0';
183 C<= ("01");
184 app <= 1;
185 WHEN n2 =>
186 estado<=n3;
187 ld <= '0';
188 C<= ("10");
189 app <= 2;
190 WHEN n3 =>
191 estado<=n0;
192 ld <= '0';
193 C<= ("11");
194 app <= 3;
195
196 END CASE;
197 END IF;
198 END IF;
199 END PROCESS;
200 END Behave;

```

Figura 7: Máquina de estados

Posteriormente será feito a declaração do My Componentes.vhd que é quando declaramos nossos componentes.Conforme mostrado nas próximas duas figuras e na figura 9.

```

223 ----- File my_components.vhd: -----
224 LIBRARY ieee;
225 USE ieee.std_logic_1164.all;
226
227 PACKAGE my_components IS
228 |
229 | COMPONENT somador16bits IS
230 | PORT ( a, b : IN STD_LOGIC_VECTOR (15 DOWNTO 0);
231 |       s : OUT STD_LOGIC_VECTOR (15 DOWNTO 0));
232 | END COMPONENT;
233 |
234 | COMPONENT mult8bits IS
235 | PORT ( a, b : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
236 |       s : OUT STD_LOGIC_VECTOR (15 DOWNTO 0));
237 | END COMPONENT;
238 |
239 | COMPONENT reg16b IS
240 | PORT ( clk : IN STD_LOGIC;
241 |       D : IN STD_LOGIC_VECTOR (15 DOWNTO 0);
242 |       Q : OUT STD_LOGIC_VECTOR (15 DOWNTO 0));
243 | END COMPONENT;
244 |
245 | COMPONENT mux4para2 IS
246 | PORT ( sel: IN STD_LOGIC_VECTOR (1 downto 0);
247 |       a, b, c, d : IN STD_LOGIC_VECTOR (7 downto 0);
248 |       Y : OUT STD_LOGIC_VECTOR (7 downto 0)
249 |       );
250 | END COMPONENT;
251 |

```

Figura 8: my_components

```

232 | COMPONENT regdes8b IS
233 | PORT ( clk, ld : IN STD_LOGIC;
234 |       D : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
235 |       Q : OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
236 | END COMPONENT;
237 |
238 | COMPONENT rom IS
239 | GENERIC ( bits: INTEGER := 8; -- # of bits per word
240 |          words: INTEGER := 4); -- # of words in the memory
241 | PORT ( addr: IN INTEGER RANGE 0 TO words-1;
242 |       data: OUT STD_LOGIC_VECTOR (bits-1 DOWNTO 0));
243 | END COMPONENT;
244 |
245 | COMPONENT rom1 IS
246 | GENERIC ( bits: INTEGER := 8; -- # of bits per word
247 |          words: INTEGER := 4); -- # of words in the memory
248 | PORT ( addr: IN INTEGER RANGE 0 TO words-1;
249 |       data: OUT STD_LOGIC_VECTOR (bits-1 DOWNTO 0));
250 | END COMPONENT;
251 |
252 | COMPONENT mef IS
253 | PORT (clk, clr: IN STD_LOGIC;
254 |       ld: OUT STD_LOGIC;
255 |       C: OUT STD_LOGIC_VECTOR (1 downto 0);
256 |       app: OUT INTEGER RANGE 0 TO 3
257 |       );
258 | END COMPONENT;
259
260 END my_components;

```

Figura 9: my_components

No registrador deslocador foi pego o valor X e colocado num registrador de 8 Bits que fora chamado 8 vezes e a sua saída vai para os dois multiplicadores conforme mostrado na figura 10.

```

288 ENTITY registradordeslocamento IS
289 PORT (clk, load: IN STD_LOGIC;
290       X: STD_LOGIC_VECTOR (7 downto 0);
291       Sa, Sb, Sc, Sd, Se, Sf, Sg, Sh: OUT STD_LOGIC_VECTOR (7 downto 0)
292       );
293 END registradordeslocamento;
294
295 ARCHITECTURE comportamento OF registradordeslocamento IS
296
297     SIGNAL Ta, Tb, Tc, Td, Te, Tf, Tg: STD_LOGIC_VECTOR (7 downto 0);
298
299 BEGIN
300
301     stage_0: regdes8b port map (clk, load, X, Ta);
302     stage_1: regdes8b port map (clk, load, Ta, Tb);
303     stage_2: regdes8b port map (clk, load, Tb, Tc);
304     stage_3: regdes8b port map (clk, load, Tc, Td);
305     stage_4: regdes8b port map (clk, load, Td, Te);
306     stage_5: regdes8b port map (clk, load, Te, Tf);
307     stage_6: regdes8b port map (clk, load, Tf, Tg);
308     stage_7: regdes8b port map (clk, load, Tg, Sh);
309
310     Sa <= Ta;
311     Sb <= Tb;
312     Sc <= Tc;
313     Sd <= Td;
314     Se <= Te;
315     Sf <= Tf;
316     Sg <= Tg;
317

```

Figura 10: Registrador de Deslocamento

```

322 ----- File my_components.vhd: -----
323 LIBRARY ieee;
324 USE ieee.std_logic_1164.all;
325
326 PACKAGE my_components1 IS
327
328 COMPONENT registradordeslocamento IS
329 PORT (clk, load: IN STD_LOGIC;
330       X: STD_LOGIC_VECTOR (7 downto 0);
331       Sa, Sb, Sc, Sd, Se, Sf, Sg, Sh: OUT STD_LOGIC_VECTOR (7 downto 0)
332       );
333 END COMPONENT;
334
335 end my_components1;
336

```

Figura 11: my_components1

Na última parte tem-se o código do filtro FIR, com a inicialização de todos os elementos necessários para ele, conforme mostra a figura 12 e 13. Criamos sinais de 8 Bits para os 8 valores de entrada, saída dos multiplexadores e das duas rom, também foram criados valores de 16 Bits para as saídas dos multiplicadores, dos somadores e do registrador, além dos sinais referentes a carga, o seletore e o apontador.

```

317 LIBRARY ieee;
318 USE ieee.std_logic_1164.all;
319 USE work.my_components.all;
320 USE work.my_components1.all;
321
322 ENTITY Trabalho2PCI IS
323 PORT (clk, limpa: IN STD_LOGIC;
324       X: IN STD_LOGIC_VECTOR (7 downto 0);
325       S: OUT STD_LOGIC_VECTOR (15 downto 0);
326       S1: OUT STD_LOGIC
327 );
328 END Trabalho2PCI;
329
330 ARCHITECTURE comportamento OF Trabalho2PCI IS
331
332 SIGNAL RP00, RP01, RP02, RP08, RP09: STD_LOGIC_VECTOR (15 downto 0);
333 SIGNAL RP03, RP04, RP06, RP07, A, B, C, D, E, F, G, H: STD_LOGIC_VECTOR (7 downto 0);
334 SIGNAL apontador: INTEGER RANGE 0 TO 3;
335 SIGNAL selecao: STD_LOGIC_VECTOR (1 downto 0);
336 SIGNAL carga: STD_LOGIC;

```

Figura 12: Filtro FIR

```

338 BEGIN
339     stage_0: registradordeslocamento port map (clk, carga, X, A, B, C, D, E, F, G, H);
340
341     stage_1: mux4para2 port map (selecao, A, B, C, D, RP04);
342     stage_2: mux4para2 port map (selecao, E, F, G, H, RP06);
343
344     stage_3: rom port map (apontador, RP03);
345     stage_4: rom1 port map (apontador, RP07);
346
347     stage_5: mult8bits port map (RP04, RP03, RP00);
348     stage_6: mult8bits port map (RP06, RP07, RP08);
349
350     stage_7: somador16bits port map (RP00, RP08, RP09);
351     stage_8: somador16bits port map (RP09, RP01, RP02);
352
353     stage_9: reg16b port map (clk, RP02, RP01);
354     stage_10: mef port map (clk, limpa, carga, selecao, apontador);
355
356     S1 <= carga;
357
358     S <= RP02;

```

Figura 13: Filtro FIR

3. Discussões e Resultados:

Abaixo temos a arquitetura do filtro FIR gerada pelo quartus II , conforme figura 14 , e o resultado da quantidade de registradores e pins de ambos os filtros, o semi-paralelo na figura 15 e o totalmente sequencial dado em aula na figura 16 .

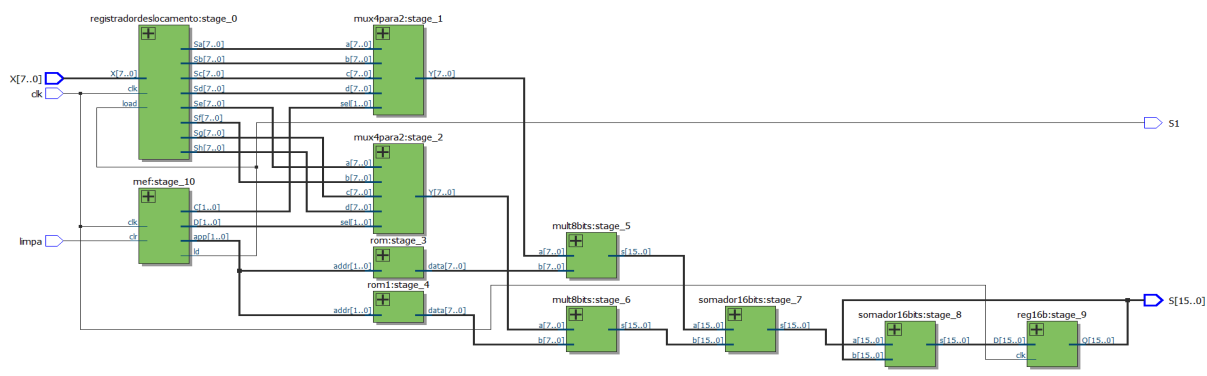


Figura 14: Arquitetura do filtro FIR Semi-Paralelo

Flow Status	Successful - Wed Apr 27 20:52:47 2022
Quartus II 64-Bit Version	13.1.0 Build 162 10/23/2013 SJ Web Edition
Revision Name	Trabalho2PCI
Top-level Entity Name	Trabalho2PCI
Family	Cyclone IV GX
Total logic elements	216 / 14,400 (2 %)
Total combinational functions	187 / 14,400 (1 %)
Dedicated logic registers	87 / 14,400 (< 1 %)
Total registers	87
Total pins	27 / 81 (33 %)
Total virtual pins	0
Total memory bits	0 / 552,960 (0 %)
Embedded Multiplier 9-bit elements	0
Total GXB Receiver Channel PCS	0 / 2 (0 %)
Total GXB Receiver Channel PMA	0 / 2 (0 %)
Total GXB Transmitter Channel PCS	0 / 2 (0 %)
Total GXB Transmitter Channel PMA	0 / 2 (0 %)
Total PLLs	0 / 3 (0 %)
Device	EP4CGX158F14C6
Timing Models	Final

Figura 15: Flow Status Filtro Fir Semi-Paralelo

Flow Status	Successful - Tue Apr 26 18:37:47 2022
Quartus II 64-Bit Version	13.1.0 Build 162 10/23/2013 SJ Web Edition
Revision Name	filtro_FIR5
Top-level Entity Name	filtro_FIR5
Family	Cyclone IV GX
Total logic elements	199 / 14,400 (1 %)
Total combinational functions	163 / 14,400 (1 %)
Dedicated logic registers	92 / 14,400 (< 1 %)
Total registers	92
Total pins	27 / 81 (33 %)
Total virtual pins	0
Total memory bits	0 / 552,960 (0 %)
Embedded Multiplier 9-bit elements	0
Total GXB Receiver Channel PCS	0 / 2 (0 %)
Total GXB Receiver Channel PMA	0 / 2 (0 %)
Total GXB Transmitter Channel PCS	0 / 2 (0 %)
Total GXB Transmitter Channel PMA	0 / 2 (0 %)
Total PLLs	0 / 3 (0 %)
Device	EP4CGX15BF14C6
Timing Models	Final

Figura 16: Flow Status Filtro Fir totalmente sequencial

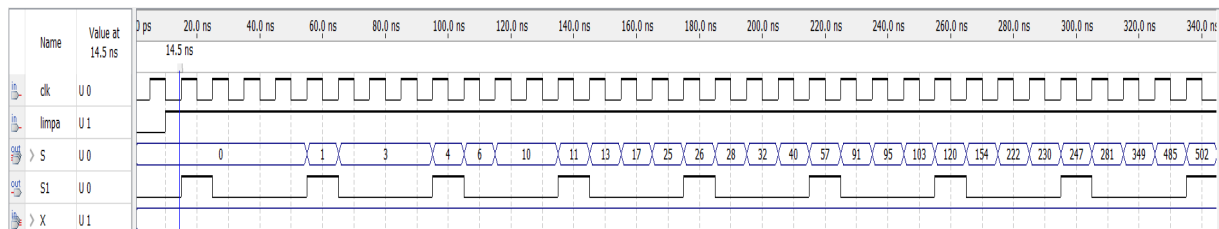


Figura 17: Testbench gerado

Conforme mostra a figura 17 temos o somatório da multiplicação dos valores da rom com o X, que vale 1 no teste, os valores finais são um pouco diferente do filtro fir original pelo fato da arquitetura ser diferente, ser feito por duas cargas em 8 ciclos de relógio diferente do sequencial que é uma carga a cada 8 ciclos de relógio.

4. Conclusão:

Teve-se um resultado bem positivo no filtro FIR semi-paralelo em relação ao filtro FIR totalmente sequencial dado em aula com a diminuição dos ciclos de relógios necessários para chegar no final de 64 no totalmente sequencial para 36 no semi-paralelo. Teve-se diferença no número de total de elementos lógicos, de registradores e pins, sendo que 216,87 e 27 respectivamente no semi-paralelo e 199,92 e 27 no totalmente sequencial.

Entende-se que o o filtro FIR semi-paralelo é mais custoso em termos de recursos mas obtém uma melhor performance em relação a velocidade do filtro, sendo comprovado quando comparamos o testbench de ambos.