



**INSTITUTO
FEDERAL**

Santa Catarina

Câmpus
São José

Contadores Genéricos para 18 bits

Eletrônica Digital II

Bernardo Souza Muniz

9 de Junho de 2025

Engenharia de Telecomunicações - IFSC-SJ

Sumário

1. Introdução	3
2. Configurações do projeto	3
3. Contador binário sequencial	4
4. Contador Gray	6
5. Contador em anel	9
6. Contador Johnson	12
7. Contador LFSR	14
8. Tabela de comparação de resultados	17
9. Conclusão	17

1. Introdução

O objetivo deste documento é apresentar 5 versões de contadores em **VHDL** (**VHSIC Hardware Description Language**) que realiza a contagem genérica para 18 bits. Todos os códigos foram estruturados em dois segmentos: um de lógica sequencial e outro de lógica combinacional. Além do desenvolvimento do código, são apresentados dados de verificação de desempenho, como a frequência máxima (Fmax), número de pinos utilizados e quantidade de elementos lógicos consumidos. A plataforma de desenvolvimento utilizada foi o Quartus Prime e a simulação foi realizada com o ModelSim.

2. Configurações do projeto

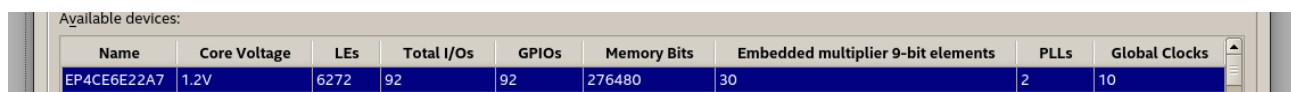
Os 5 contadores binários foram implementados contendo uma variável genérica presente na entidade do projeto que controla as variáveis de saída para terem 18 bits. Dessa forma, é possível contar de 0 zero até 262144 seguindo a seguinte lógica:

$$2^{18} - 1 = 262144 \quad (1)$$

Todos os contadores foram projetos em dois segmentos, contendo a parte sequencial (State Register) e a parte combinacional (Next State). Além disso, foi implementado em cada contador um sinal de enable para iniciar a contagem de bits.

As configurações relacionadas a família de FPGA's e código são semelhantes para todos os contadores. Foi utilizado o dispositivo FPGA da família **Cyclone IV E** com código **EP4CE6E22A7**. A figura 1 mostra alguns parâmetros relacionados ao dispositivo escolhido.

Figura 1: Device Family



Name	Core Voltage	LEs	Total I/Os	GPIOs	Memory Bits	Embedded multiplier 9-bit elements	PLLs	Global Clocks
EP4CE6E22A7	1.2V	6272	92	92	276480	30	2	10

Fonte: Elaborado pelo autor

Nota-se que o dispositivo em questão possui um **Core Voltage de 1.2V** e **6272 elementos lógicos configuráveis (LEs)**.

Além disso, com o objetivo de melhorar a frequência do Fmax para cada contador, foi utilizado um arquivo **.sdc** contendo a seguinte configuração:

```
1 create_clock -name CLK50MHz -period 50MHz [get_ports -no_case {clk*}]
```

O comando busca restringir a frequência máxima de clock, melhorando o desempenho do circuito.

A simulação para todos os contadores possui um arquivo com a extensão **.do** que tem o mesmo nome da entidade de cada circuito.

3. Contador binário sequencial

O contador binário sequencial genérico faz a contagem binária usual para 18 bits, fazendo o incremento do valor binário automaticamente a cada pulso de clock.

O código abaixo demonstra a declaração da entidade contendo a variável natural N que controla a quantidade de bits da saída para 18.

```
1 entity contador_binario is
2   generic(
3       N : natural := 18
4   );
5   port(
6       clk,rst,enable: std_logic;
7       c_out: out std_logic_vector(N-1 downto 0)
8   );
9 end entity;
```

Foi utilizado um código de dois segmentos, contendo a parte combinacional e a parte sequencial, além de um sinal de enable para início da contagem. Os dois códigos abaixo contêm a State Register e o Next State, respectivamente.

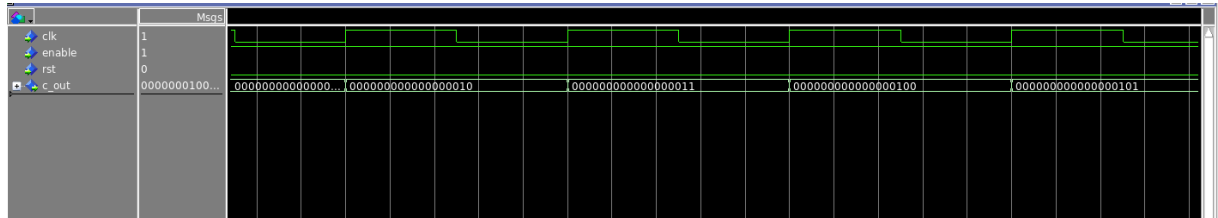
```
1 l1: process(clk, rst)
2 begin
3     if(rst = '1') then
4         c_reg <= (others => '0');
5     elsif rising_edge(clk) then
6         c_reg <= c_next;
7     end if;
8 end process;
```

```
1 l2: process(c_reg, enable)
2 begin
3     c_next <= c_reg;
4
5     if (enable = '1') then
6         c_next <= c_reg + 1;
7     end if;
8 end process;
```

O funcionamento do contador ocorre em duas etapas: o registrador de estado (State Register) atualiza o valor da contagem com base no próximo estado (c_next) a cada borda de subida do clock, desde que o sinal de rst esteja desativado. A lógica combinacional, por sua vez, define o valor de c_next com base no valor atual (c_reg) e no sinal enable, que permite o incremento apenas quando estiver em nível lógico alto. Dessa forma, o contador permanece estático enquanto enable = '0' e só conta quando enable = '1'.

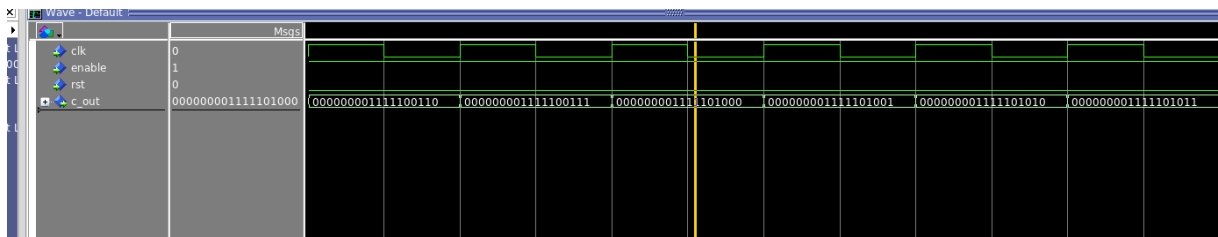
Uma vez compilado o projeto, foram feitas as devidas simulações no ModelSim com o intuito de verificar os resultados obtidos pelo contador. Foi feita a contagem até 1000 para demonstrar a eficiência do contador e a quantidade de bits de saída.

Figura 2: Simulação inicial do contador binário



Fonte: Elaborado pelo autor

Figura 3: Simulação de contagem até o número 1000

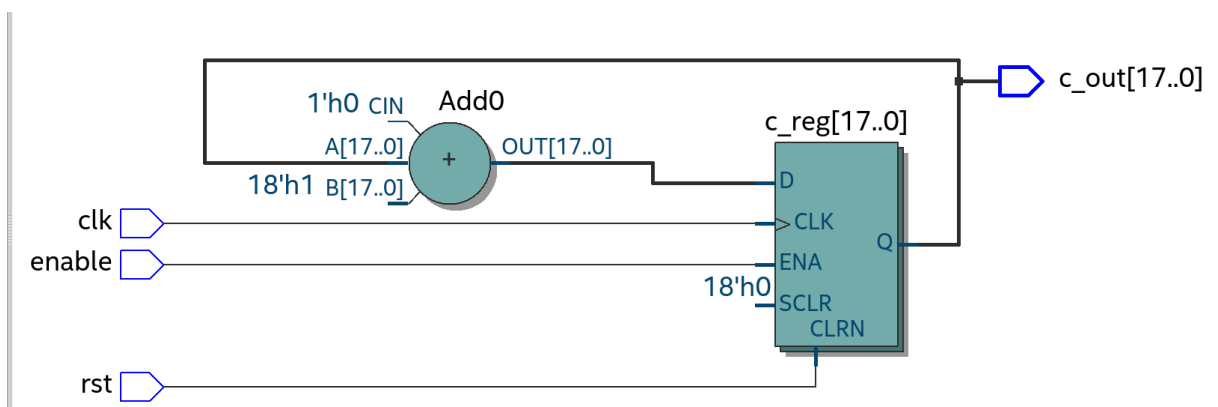


Fonte: Elaborado pelo autor

Nota-se que na figura 3 o contador chegou até o número 1111101000, 1000 em binário, comprovando o que se esperava do circuito.

As próximas figuras registram a análise feita dos parâmetros característicos do circuito, contendo tempo de Fmax, diagrama do RTL Viwer e número de elementos lógicos.

Figura 4: RTL Viwer contador binário



Fonte: Elaborado pelo autor

Figura 5: Sumário contendo a quantidade de elementos lógicos do contador binário

Table of Contents	
Flow Summary	
Flow Settings	
Flow Non-Default Global Settings	
Flow Elapsed Time	
Flow OS Summary	
Flow Log	
Analysis & Synthesis	
Flow Messages	
Flow Suppressed Messages	

Flow Summary	
<<Filter>>	
Flow Status	Successful - Tue Jun 10 19:34:29 2025
Quartus Prime Version	20.1.1 Build 720 11/11/2020 SJ Standard Edition
Revision Name	contador_binario
Top-level Entity Name	contador_binario
Family	Cyclone IV E
Device	EP4CE6E22A7
Timing Models	Final
Total logic elements	18
Total registers	18
Total pins	21
Total virtual pins	0
Total memory bits	0
Embedded Multiplier 9-bit elements	0
Total PLLs	0

Fonte: Elaborado pelo autor

Figura 6: FMAX do contador binário

Flow Summary

Flow Settings

Flow Non-Default Global Settings

Flow Elapsed Time

Flow OS Summary

Flow Log

Analysis & Synthesis

<<Filter>>

	Fmax	Restricted Fmax	Clock Name	Note
1	344.35 MHz	250.0 MHz	CLK50MHz	limit due to minimum period restriction (max I/O toggle rate)

Fonte: Elaborado pelo autor

4. Contador Gray

De maneira análoga ao contador binário sequencial, o contador gray faz a contagem genérica para 18 bits, porém em código Gray. Desta forma, ao fazer a contagem, a cada incremento apenas um bit é modificado, diferentemente da contagem binária aonde vários bits podem alterar com o incremento de um.

No contador, foi utilizada a lógica de conversão de binário pra gray, onde se utiliza uma operação lógica **xor** para calcular o próximo bit. Tal relação é dada abaixo:

$$\begin{aligned}
 b_3 &= g_3 \oplus 0 \\
 b_2 &= g_2 \oplus b_3 \\
 b_1 &= g_1 \oplus b_2 \\
 b_0 &= g_0 \oplus b_1
 \end{aligned} \tag{2}$$

O código abaixo demonstra a declaração da entidade contendo a variável natural N que controla a quantidade de bits da saída para 18.

```
1 entity contador_gray is
```

```

2 generic(
3     N : natural := 18
4 );
5 port(
6     clk, reset, enable: in std_logic;
7     gray_out: out std_logic_vector(N - 1 downto 0)
8 );
9 end entity;

```

Foi utilizado um código de dois segmentos, contendo a parte combinacional e a parte sequencial, além de um sinal de enable para início da contagem. Os dois códigos abaixo contêm a State Register e o Next State, respectivamente.

```

1 process(clk,reset)
2 begin
3     if (reset='1') then
4         g_reg <= (others=>'0');
5     elsif rising_edge(clk) then
6         g_reg <= g_next;
7     end if;
8 end process;

```

```

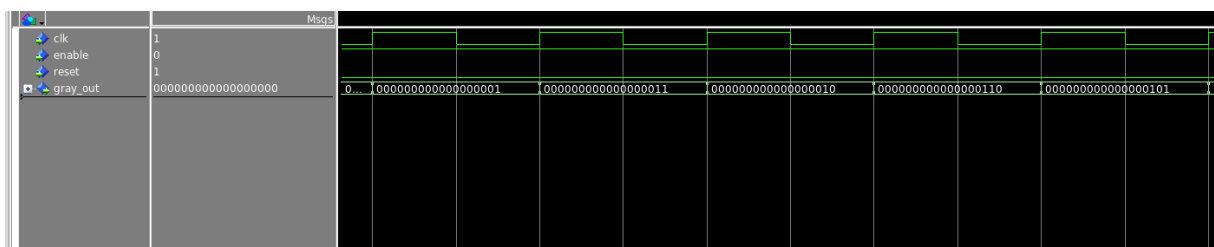
1 process(enable, g_reg)
2     variable b, b1: unsigned(N-1 downto 0);
3 begin
4     g_next <= g_reg;
5     if enable = '1' then
6         b := g_reg xor ('0' & g_reg(N-1 downto 1));
7         b1 := b+1;
8         g_next <= b1 xor ('0' & b1(N-1 downto 1));
9     end if;
10 end process;

```

O State Register atualiza o valor da contagem com base no próximo estado (g_next) a cada borda de subida do clock desde que o sinal de reset esteja desativado. Na lógica combinacional, foram utilizadas duas variáveis dentro do process do Next State b e b1 para auxiliar na contagem do próximo bit apenas quando o sinal enable estiver em nível lógico alto, caso contrário, as variável g_next apenas recebe g_reg.

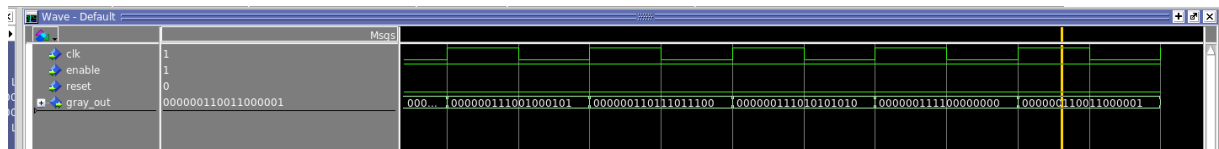
Uma vez compilado o projeto, foram feitas as devidas simulações no ModelSim com o intuito de verificar os resultados obtidos pelo contador.

Figura 7: Simulação inicial do contador gray



Fonte: Elaborado pelo autor

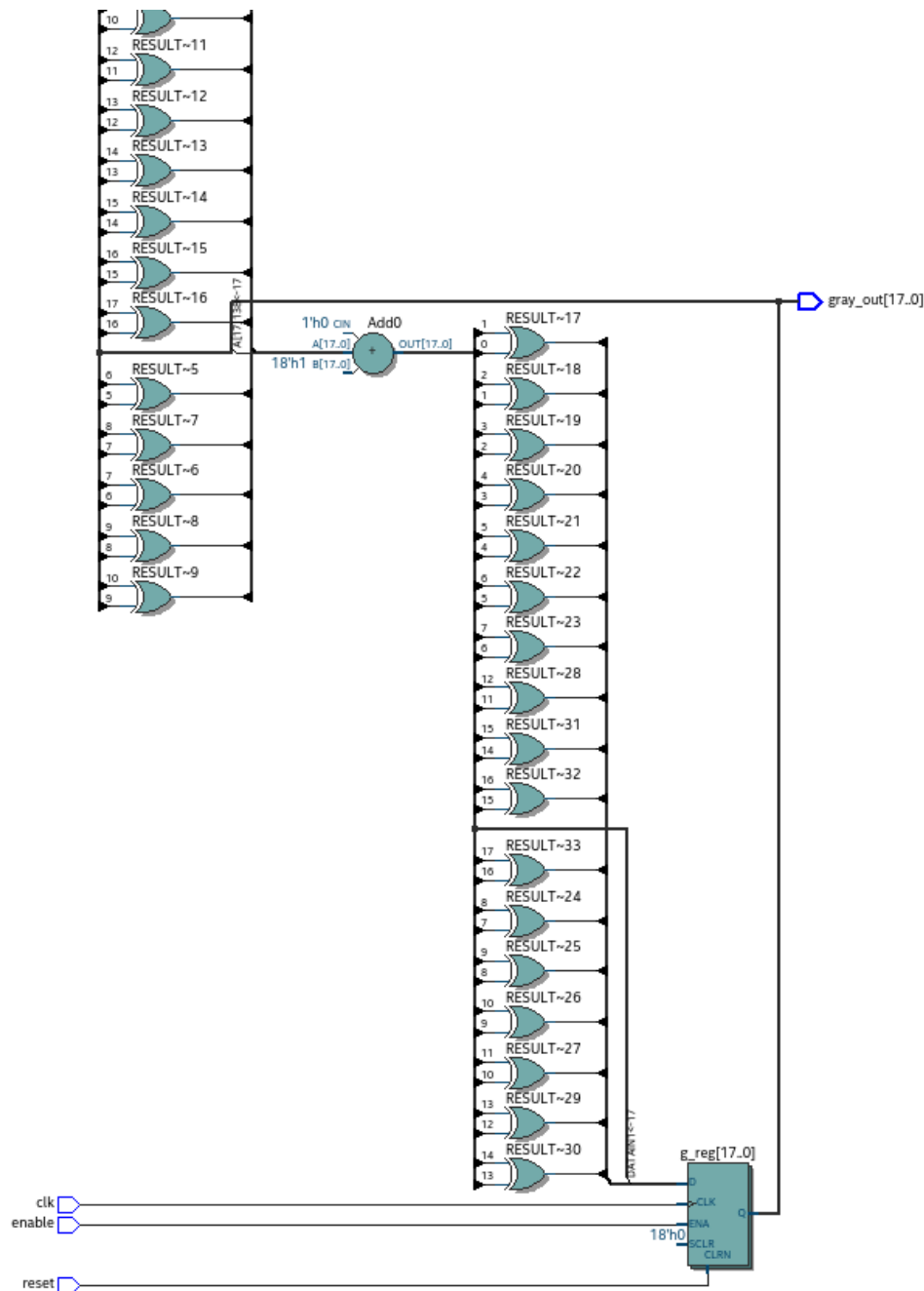
Figura 8: Simulação de contagem até o número 1000



Fonte: Elaborado pelo autor

As próximas figuras registram a análise feita dos parâmetros característicos do circuito, contendo tempo de Fmax, diagrama do RTL Viwer e número de elementos lógicos.

Figura 9: RTL Viwer do contador gray



Fonte: Elaborado pelo autor

Figura 10: Sumário contendo a quantidade de elementos lógicos do contador gray

Flow Summary	
<<Filter>>	
Flow Status	Successful - Tue Jun 10 22:03:31 2025
Quartus Prime Version	20.1.1 Build 720 11/11/2020 SJ Standard Edition
Revision Name	contador_gray
Top-level Entity Name	contador_gray
Family	Cyclone IV E
Device	EP4CE6E22A7
Timing Models	Final
Total logic elements	36
Total registers	18
Total pins	21
Total virtual pins	0
Total memory bits	0
Embedded Multiplier 9-bit elements	0
Total PLLs	0

Fonte: Elaborado pelo autor

Figura 11: FMAX do contador gray

Slow 1200mV 125C Model Fmax Summary				
<<Filter>>				
	Fmax	Restricted Fmax	Clock Name	Note
1	210.3 MHz	210.3 MHz	CLK50MHz	

Fonte: Elaborado pelo autor

5. Contador em anel

O contador em anel genérico para 18 bits faz o deslocamento de bits a cada pulso de clock.

O código abaixo demonstra a declaração da entidade contendo a variável natural N que controla a quantidade de bits da saída para 18.

```

1  entity contador_anel is
2  generic(
3      N : natural := 18
4  );
5  port(
6      clk,rst,enable: std_logic;
7      ring_out: out std_logic_vector(N-1 downto 0)
8  );
9  end entity;
```

Foi utilizado um código de dois segmentos, contendo a parte combinacional e a parte sequencial, além de um sinal de enable para início da contagem. Os dois códigos abaixo contêm a State Register e o Next State, respectivamente.

Nota-se que quando o sinal de reset está alto, apenas o último bit fica com sinal 1 e o resto fica zero, para que assim seja iniciado a contagem.

```

1  l1: process(clk, rst)
2  begin
3      if(rst = '1') then
4          ring_reg <= (0 => '1', others => '0');
5      elsif rising_edge(clk) then
6          ring_reg <= ring_next;
7      end if;
8  end process;

```

```

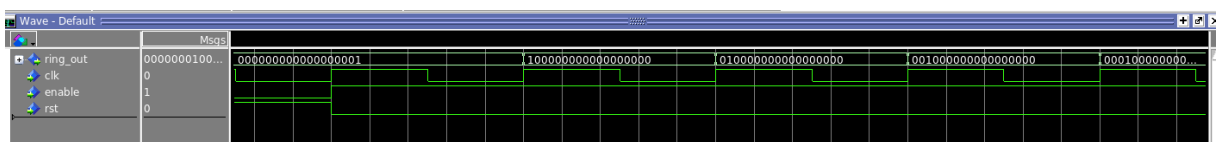
1  l2: process(enable, ring_reg)
2  begin
3      ring_next <= ring_reg;
4      if enable = '1' then
5          ring_next <= ring_reg(0) & ring_reg(N-1 downto 1);
6      end if;
7  end process;

```

O State Register atualiza o valor da contagem com base no próximo estado (ring_next) a cada borda de subida do clock desde que o sinal de rst esteja desativado. Na lógica combinacional, foram utilizadas dois sinais que fazem a mudança de bits apenas quando o sinal enable estiver em nível lógico alto, caso contrário, as variável ring_next apenas recebe ring_reg.

Uma vez compilado o projeto, foram feitas as devidas simulações no ModelSim com o intuito de verificar os resultados obtidos pelo contador.

Figura 12: Simulação de contagem do contador em anel



Fonte: Elaborado pelo autor

Nota-se que o contador com 18 bits segue a mesma lógica de sequência que um contador com 4 bits, por exemplo:

```

1  0001 -> 1000 -> 0100 -> 0010 -> 0001 -> 1000 -> 0100 -> ... -> (repete)

```

Figura 13: Simulação de contagem do contador em anel com repetição

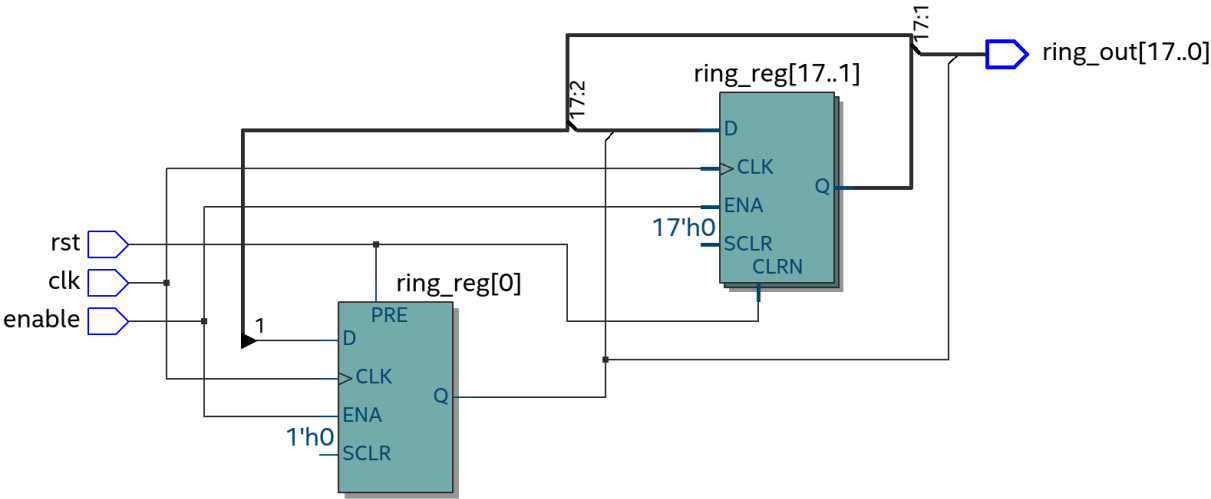


Fonte: Elaborado pelo autor

Na figura 13 foi registrado a repetição do contador em anel para 18 bits, onde o contador naturalmente volta para o seu estado original de 100000000000000000 para reiniciar a contagem, fazendo a mudança dos próximos bits

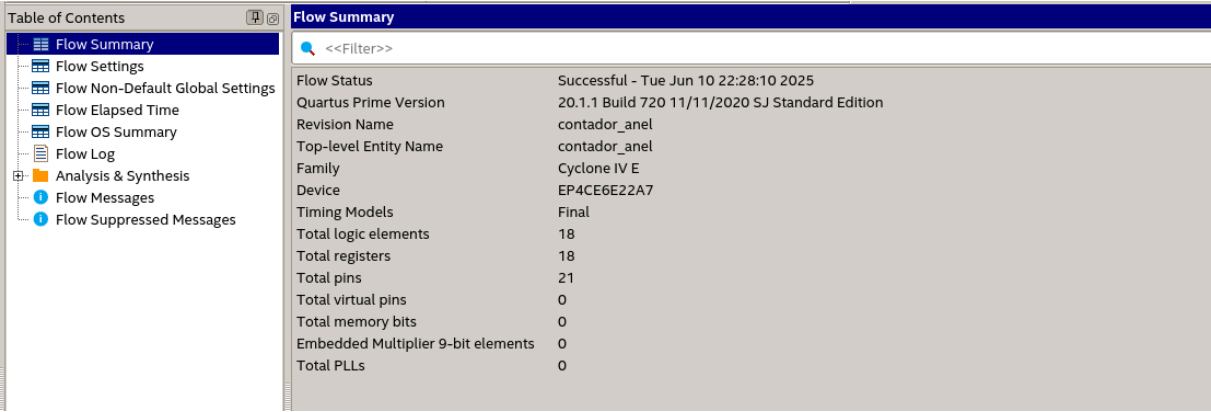
As próximas figuras registram a análise feita dos parâmetros característicos do circuito, contendo tempo de Fmax, diagrama do RTL Viwer e número de elementos lógicos.

Figura 14: RTL Viwer do contador em anel



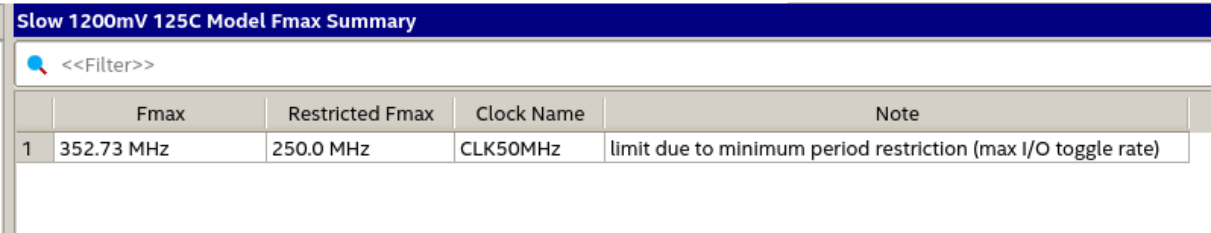
Fonte: Elaborado pelo autor

Figura 15: Sumário contendo a quantidade de elementos lógicos do contador em anel



Fonte: Elaborado pelo autor

Figura 16: FMAX do contador em anel



Fonte: Elaborado pelo autor

6. Contador Johnson

Semelhante ao contador em anel, o contador Johnson para 18 bits faz o uso de um inversor na realimentação do contador em anel.

O código abaixo demonstra a declaração da entidade contendo a variável natural N que controla a quantidade de bits da saída para 18.

```
1 entity contador_johnson is
2   generic(
3       N : natural := 18
4   );
5   port(
6       clk,rst,enable: std_logic;
7       ring_out: out std_logic_vector(N-1 downto 0)
8   );
9 end entity;
```

Foi utilizado um código de dois segmentos, contendo a parte combinacional e a parte sequencial, além de um sinal de enable para início da contagem. Os dois códigos abaixo contêm a State Register e o Next State, respectivamente.

```
1 l1: process(clk, rst)
2 begin
3     if(rst = '1') then
4         ring_reg <= (0 => '1', others => '0');
5     elsif rising_edge(clk) then
6         ring_reg <= ring_next;
7     end if;
8 end process;
```

```
1 l2: process(enable, ring_reg)
2 begin
3     ring_next <= ring_reg;
4     if enable = '1' then
5         ring_next <= (not ring_reg(0)) & ring_reg(N-1 downto 1);
6     end if;
7 end process;
```

Nota-se a inversão na realimentação com o uso do **not** para atualizar o contador.

Uma vez compilado o projeto, foram feitas as devidas simulações no ModelSim com o intuito de verificar os resultados obtidos pelo contador.

Figura 17: Simulação de contagem do contador em Johnson

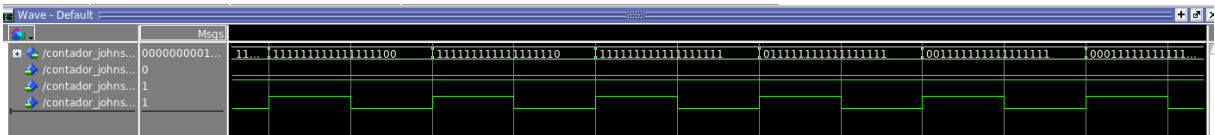


Fonte: Elaborado pelo autor

Nota-se que o contador Johnson com 18 bits segue a mesma lógica de sequência que um contador com 4 bits, por exemplo:

```
1  0001 -> 0000 -> 1000 -> 1100 -> 1110 -> 1111 -> 0111 -> 0011 -> 0001 ->
    0000 -> ... -> (repete)
```

Figura 18: Simulação de contagem do contador Johnson com repetição

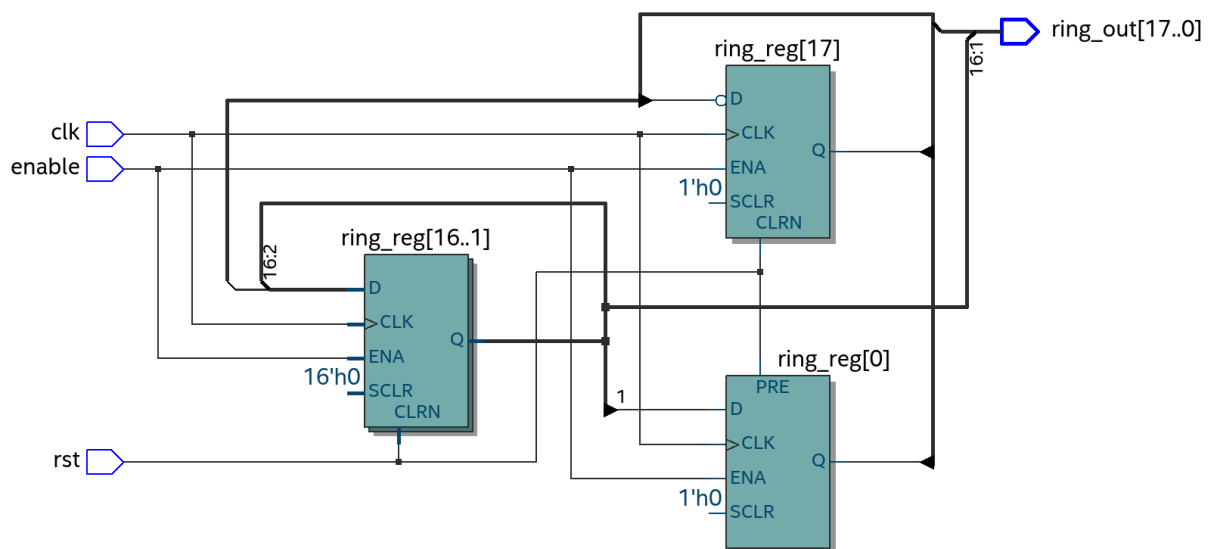


Fonte: Elaborado pelo autor

Na figura 18 foi registrado a repetição do contador em Johnson para 18 bits, onde o contador estoura a quantidade de bits com '1' e volta para completar com zeros, reiniciando novamnte o ciclo.

As próximas figuras registram a análise feita dos parâmetros característicos do circuito, contendo tempo de Fmax, diagrama do RTL Viwer e número de elementos lógicos.

Figura 19: RTL Viwer do contador Johnson



Fonte: Elaborado pelo autor

Figura 20: Sumário contendo a quantidade de elementos lógicos do contador Johnson

Table of Contents		Flow Summary	
<ul style="list-style-type: none"> Flow Summary Flow Settings Flow Non-Default Global Settings Flow Elapsed Time Flow OS Summary Flow Log Analysis & Synthesis Flow Messages Flow Suppressed Messages 		<<Filter>>	
		Flow Status	Successful - Mon Jun 9 22:53:51 2025
		Quartus Prime Version	20.1.1 Build 720 11/11/2020 SJ Standard Edition
		Revision Name	contador_johnson
		Top-level Entity Name	contador_johnson
		Family	Cyclone IV E
		Device	EP4CE6E22A7
		Timing Models	Final
		Total logic elements	18
		Total registers	18
		Total pins	21
		Total virtual pins	0
		Total memory bits	0
		Embedded Multiplier 9-bit elements	0
		Total PLLs	0

Fonte: Elaborado pelo autor

sim:/timer_seg/SSD_UNIDADE

7. Contador LFSR

O contador LFSR (Linear-feedback shift register) foi projetado para gerar números aleatórios.

Para isso, foi utilizado um polinômio primitivo para 18 bits com o seguinte formato:

$$x^{18} + x^{11} + 1 \quad (3)$$

Além disso, foi utilizado um Tap correspondente com o polinômio utilizado no formato:

$$100000010000000000 \quad (4)$$

O código abaixo demonstra a declaração da entidade contendo a variável natural N que controla a quantidade de bits da saída para 18.

```

1  entity contador_lfsr is
2  generic(
3      N : natural := 18
4  );
5  port(
6      clk, reset, enable: in std_logic;
7      lfsr_out: out std_logic_vector(N-1 downto 0)
8  );
9  end entity;
```

Foi utilizado um código de dois segmentos, contendo a parte combinacional e a parte sequencial, além de um sinal de enable para início da contagem. Os dois códigos abaixo contêm a State Register e o Next State, respectivamente.

```

1  ll: process(clk,reset)
2  begin
3      if (reset='1') then
```

```

4     r_reg <= SEED;
5     elsif rising_edge(clk) then
6         r_reg <= r_next;
7     end if;
8 end process;

```

Dentro da arquitetura do contador, foi utilizado uma constante **SEED** que utiliza o valor do Tap referente ao polinômio primitivo para 18 bits:

```

1 constant SEED: std_logic_vector(N-1 downto 0) := "100000010000000000";

```

```

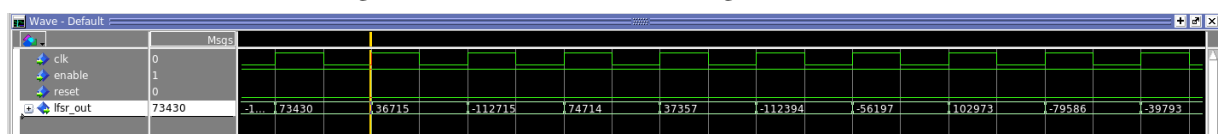
1 l2: process(enable, r_reg)
2     variable fb: std_logic;
3 begin
4     r_next <= r_reg;
5     if enable = '1' then
6         fb := r_reg(N-1) xor r_reg(10);
7         r_next <= fb & r_reg(N-1 downto 1);
8     end if;
9 end process;

```

Neste caso quando o reset é ativado, temos que $r_reg \leq SEED$ e dentro do process temos que $fb := r_reg(N-1) \text{ xor } r_reg(10)$, fazendo a lógica inicial de $fb := '0' \text{ xor } '1' = '1'$, onde o feedback fica com valor inicial 1 e dá início ao contador.

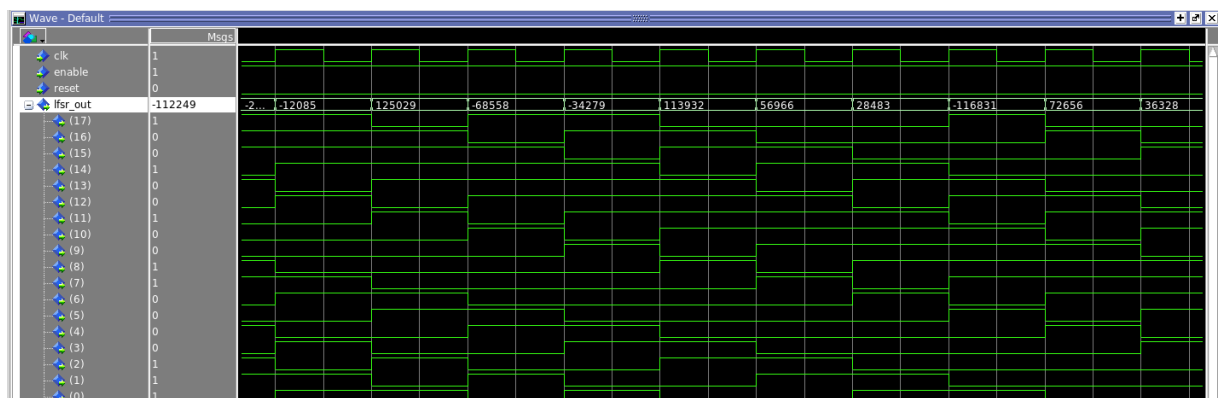
Uma vez compilado o projeto, foram feitas as devidas simulações no ModelSim com o intuito de verificar os resultados obtidos pelo contador.

Figura 21: Simulação de contagem do LFSR



Fonte: Elaborado pelo autor

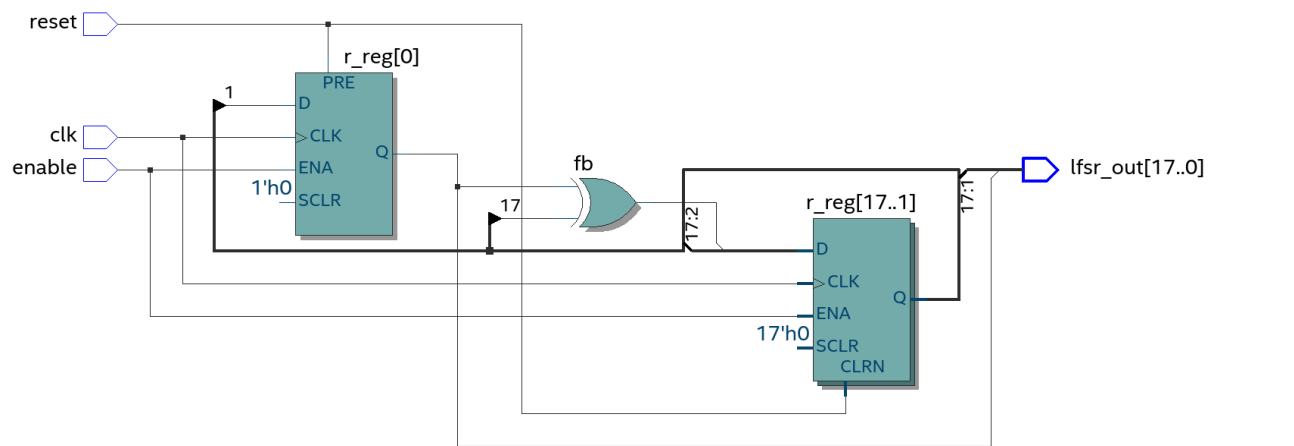
Figura 22: Simulação de contagem do LFSR - Visualização dos bits de saída



Fonte: Elaborado pelo autor

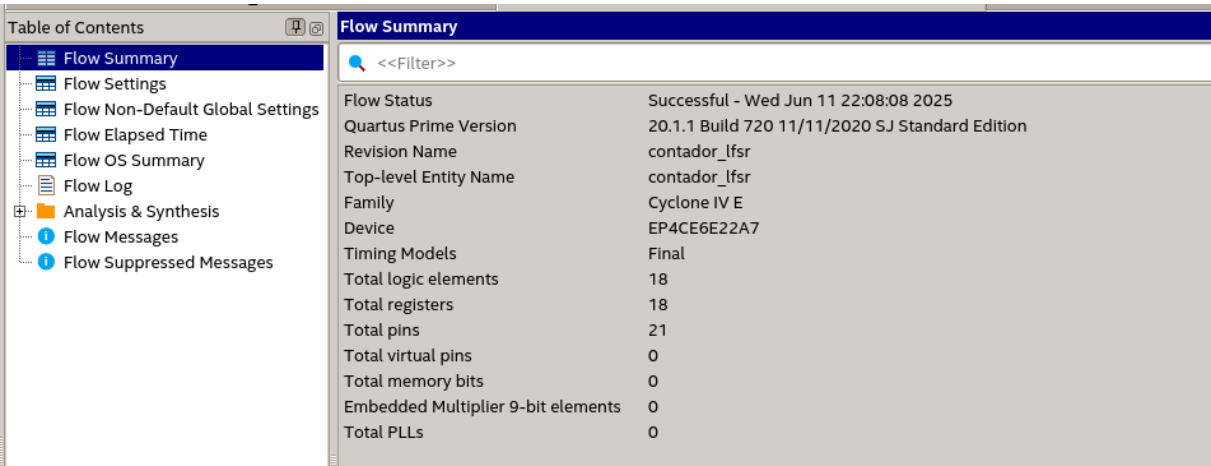
As próximas figuras registram a análise feita dos parâmetros característicos do circuito, contendo tempo de Fmax, diagrama do RTL Viwer e número de elementos lógicos.

Figura 23: RTL Viwer do contador LFSR



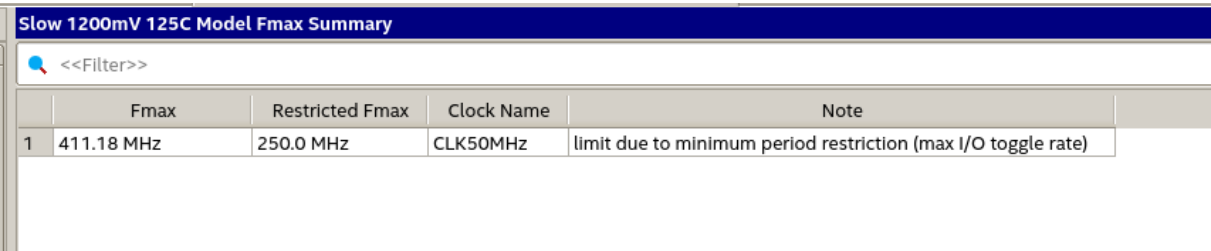
Fonte: Elaborado pelo autor

Figura 24: Sumário do contador LFSR



Fonte: Elaborado pelo autor

Figura 25: FMAX do contador LFSR



Fonte: Elaborado pelo autor

8. Tabela de comparação de resultados

Para fins de comparação de resultados, foi montado uma tabela contendo as principais informações técnicas de cada contador.

Tabela 1: Elaborada pelo Autor

Parâmetros	Contador Binário	Contador Gray	Contador em anel	Contador Johnson	Contador LFSR
Elementos lógicos	18	36	18	18	18
Pinos	21	21	21	21	21
Registers	10	18	18	18	18
Fmax (MHz)	344,35	210,30	352,73	352,73	411,18

Tabela de resultados de compilação para os 5 contadores

9. Conclusão

Após a implementação dos 5 contadores BCD para 18 bits, foi possível concluir que a utilização de cada contador tem uma funcionalidade específica para cada cenário de uso. Nota-se que a utilização de tais circuitos podem estar presentes em diversas partes de indústrias e empresas, sendo possível fazer a reutilização para diferentes tipos de projetos. Algumas das implementações possíveis que foram notadas são: timer, sorteador de números, contagens em displays de 7 segmentos e etc.

Os objetivos definidos no início do projeto foram alcançados, resultando na implementação bem-sucedida dos contadores genéricos para 18 bits.