Bernardo Euclides Faria Gomes, Luiz Henrique Gariglio dos Santos, Naiara Barcelos Silveira

PROJETO MISTURADOR DE TINTA

Belo Horizonte
2023

Bernardo Euclides Faria Gomes, Luiz Henrique Gariglio dos Santos, Naiara Barcelos Silveira

PROJETO MISTURADOR DE TINTA

Descrição do desenvolvimento do projeto de um misturador de tinta para a disciplina de Laboratório de Sistemas Digitais.

Professor: Hermes Aguiar Magalhães

Belo Horizonte
2023

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

O intuito deste trabalho é viabilizar uma máquina de mistura de tintas a partir dos conhecimentos construídos durante a disciplina de Laboratório de Sistemas Digitais. A ideia é que o usuário possa entrar com o código CMYK (Cyan - ciano, Magenta, Yellow - amarelo, Black - preto) na máquina e, a partir disso, receba uma tinta com a cor correspondente deste código. A simulação do modelo será feita em laboratório a partir do Kit Altera DE2, com um chip da família Cyclone II. Todo o código será feito em VHDL e as partes desenvolvidas da máquina serão dispostas nesse documento.

Para essa construção, contamos com um diagrama de máquina de estados finitos de alto nível, um desenho do caminho de dados e os códigos. O resultado esperado é possibilitar uma simulação de como funcionaria uma máquina desse tipo, a partir de botões e leds disponibilizados no kit Altera DE2.

2 PROJETO

2.1 Funcionamento Desejado

A máquina (figura 2) deverá receber do usuário um código referente à cor desejada por ele, e então, por meio da liberação de pigmentos e mistura da tinta, deverá manipular uma lata de tinta inicialmente branca a fim de transformá-la em uma lata de tinta da cor desejada.

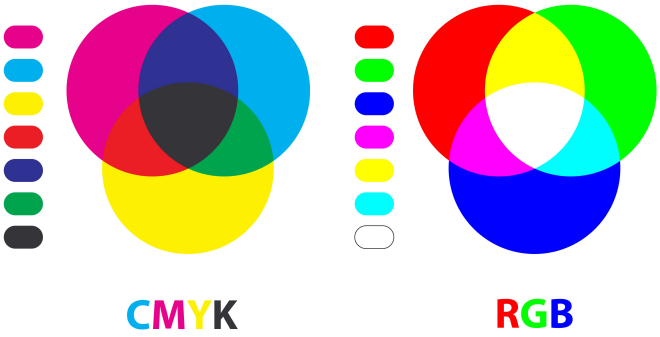

Figura 1 – Figura 1: Diferença entre CMYK e RGB.

O código recebido deve estar de acordo com a escala CMYK (Cyan - ciano, Magenta, Yellow - amarelo, Black - preto). Essa escala é um sistema subtrativo, sendo assim, os pigmentos são adicionados de maneira a remover cor até chegar ao tom pretendido. Dessa forma, os pigmentos servem como filtro, absorvendo determinados comprimentos de onda e manipulando qual cor será visualizada. A diferença entre o CMYK e o RGB se dá pois o primeiro é composto por cor pigmento enquanto o segundo é composto por cor luz.

Dessa forma, para o CMYK, a injeção de cores da máquina será feita por 4 válvulas, uma para cada pigmento, de forma a injetar as cores na base branca da lata. Será considerado que 1 gota de pigmento equivale a 0,05ml, e são necessários 100 milissegundos com a válvula aberta para que 1 gota caia.

O primeiro passo de funcionamento, é um estado de início. Nesse estado, os registradores são zerados, assim como os contadores. Isso evita que alguma informação de ”lixo”entre no processo que será iniciado, garantindo maior segurança de que o resultado será o desejado.

O segundo passo, é um estado de espera. Nele, a máquina está aguardando que o usuário entre com um código válido de cor CMYK e posicione a lata no local correto. O código é chamado de ”Codigo_cor”e é uma entrada de 32 bits, enquanto a posição da lata será tratada como uma entrada de um único bit.

A partir disso, a máquina avança para um estado de validação. Nesse estado, o código inserido será validado, ou seja, o programa deve retornar se o que o usuário forneceu como código de cor é de fato um código válido. Isso resulta em uma bifurcação no diagrama, já que, dependendo se o código for válido ou não, a tratativa sobre ele será diferente. Se for inválido, a máquina passa para o estado de erro. No caso do código ser válido, o próximo passo é o estado de pigmentação.

No caso de erro, o próximo estado passa a ser o início, a partir de um reset. Esse estado emite um código de erro para que o usuário possa recomeçar o processo depois de corrigir o que havia de errado da primeira vez.

No caso onde o código é válido e o processo segue para a pigmentação, Nesse estado, será calculado quanto tempo cada válvula deve ficar aberta para que a tinta resultante tenha a cor desejada. Após isso, as válvulas são energizadas, o que nesse trabalho será representado por leds acesos.

Após a adição de pigmento, o processo entra no estado de mistura. A partir da agitação da lata, os pigmentos são misturados na tinta. Esse processo também será representado com um led aceso.

Seguinte à mistura, a máquina deve verificar se o resultado obtido do processo é igual ao resultado desejado pelo usuário, sendo assim, inicia-se o estado de verificação. Na prática, isso seria feito a partir de um sensor de cor e a comparação do código de entrada e o código retornado pelo sensor. Nesse trabalho, usaremos apenas para teste com códigos inseridos manualmente. Caso os códigos batam, a máquina segue para o estado de pronto. No caso de ter sido obtido um resultado indesejado, inicia-se o estado de erro, que dá a mesma tratativa já descrita após o processo de validação.

No último estágio do ciclo, tem-se a tinta pronta. Esse estado aguarda que a lata de tinta seja retirada do local reservado, para que o ciclo possa ser reiniciado.

2.2 Máquina de Estados Finitos

A FSM (figura3) foi projetada com 8 estados:

- Início - Responsável por enviar o sinal de ”reset”aos registradores e contadores;
- Espera - Neste estado, é realizada a leitura do código digitado pelo usuário. O máquina permanece em espera até que receba um sinal de que há um recipiente no local e o botão de iniciar seja pressionado;
- Verifica - Ocorre a verificação se o código é válido. Caso o código seja válido, a máquina segue para o estado de pigmentação. Caso contrário, vai para o estado de erro;
- Pigmento - É enviado o sinal para a adição dos respectivos pigmentos de acordo com o código digitado pelo usuário. Uma vez terminado o tempo, a máquina segue para o estado de mistura;

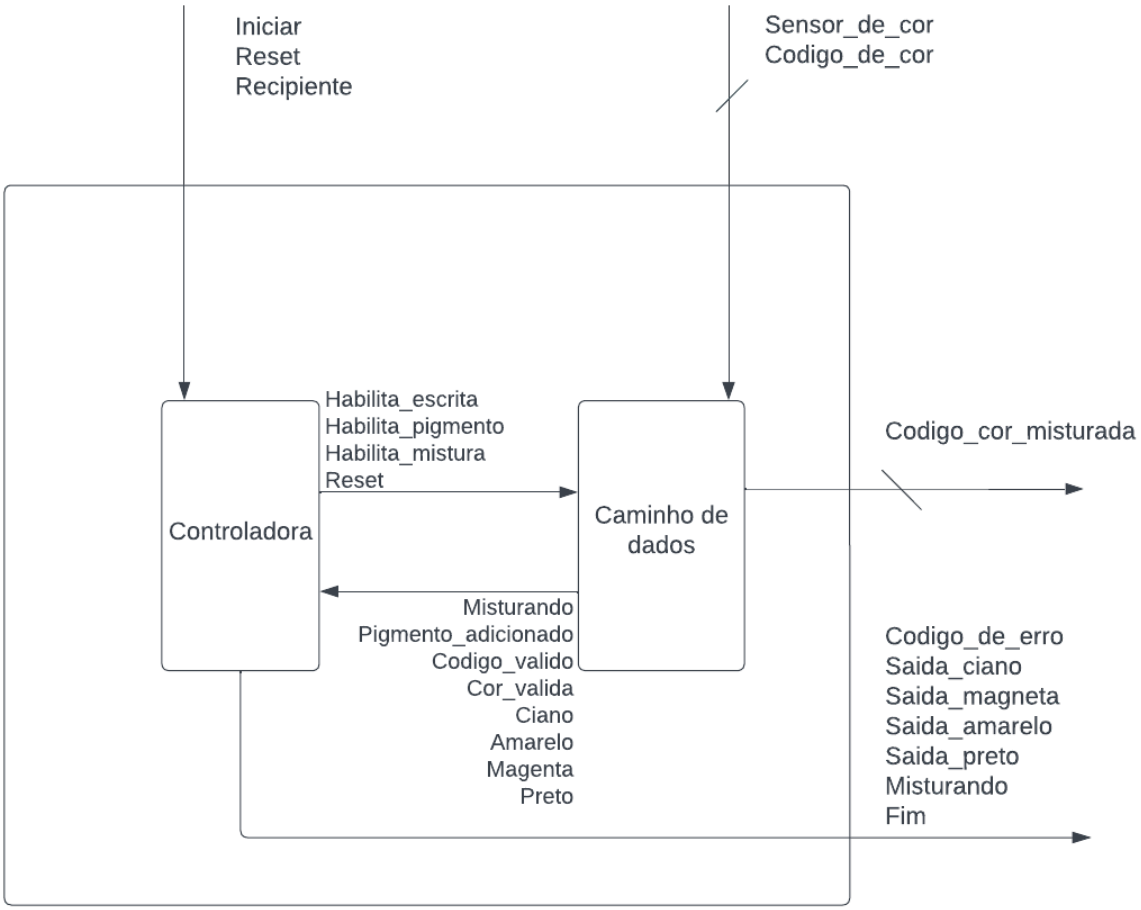

Figura 2 – Diagrama do misturador de tinta

- Mistura - É enviado o sinal para que a mistura da tinta seja realizada. Após o tempo determinado , a máquina segue para o próximo estado;
- Valida - Neste estado ocorre a validação da cor produzida. É comparado o código digitado pelo usuário com o sinal vindo de um sensor de cor. Caso a cor produzida esteja de acordo com a solicitada, segue para o estado de pronto. Se não, segue para o estado de erro;
- Pronto - Indica que a tinta está pronta e espera que o recipiente seja retirado para que uma nova cor possa ser produzida. O estado seguinte é o de início;
- Erro - Envia um sinal de erro e espera que a máquina seja reiniciada.

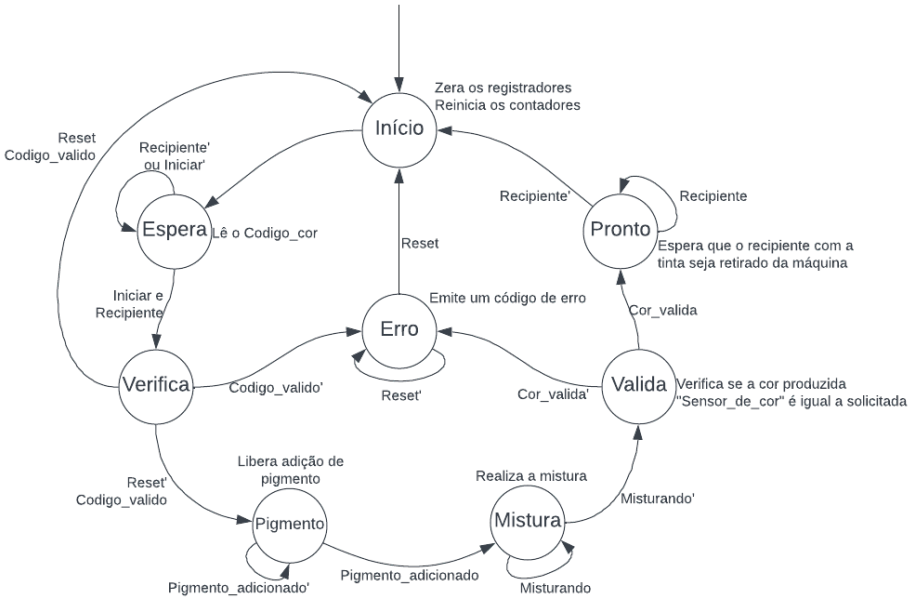

Figura 3 – Máquina de Estados Finitos de Alto Nível

2.3 Caminho de dados

A imagem abaixo (figura 4) mostra o caminho de dados projetado para essa aplicação. Os componentes utilizados no projeto são detalhados da seção ”Componentes”.

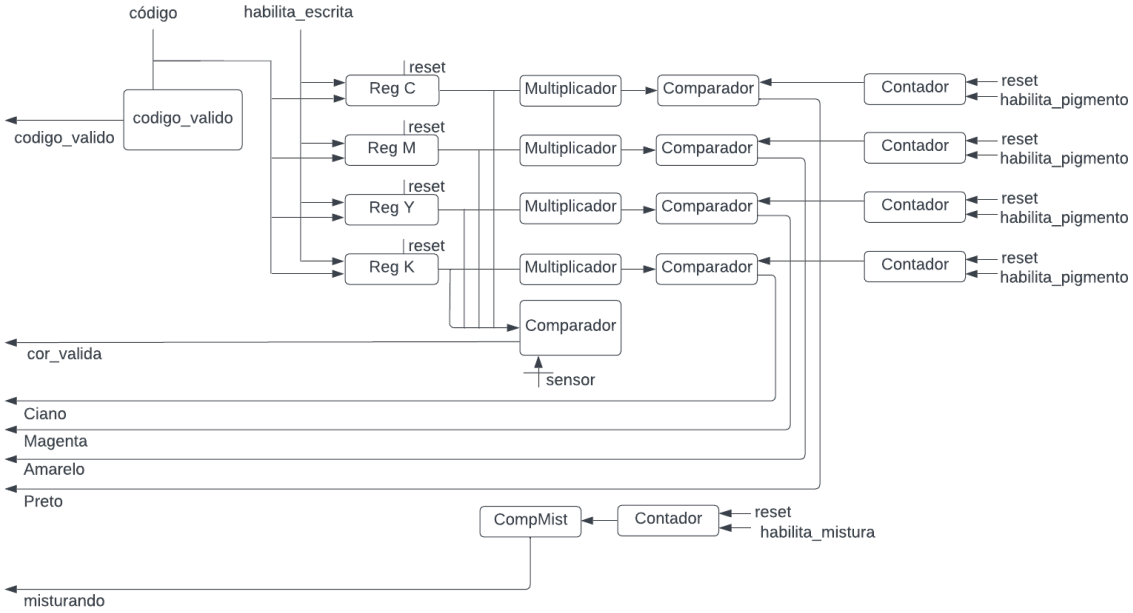

Figura 4 – Diagrama de ligação do caminho de dados

2.4 Componentes

2.4.1 Registrador

O registrador possui uma entrada load, que habilita que o dado seja salvo, uma entrada de reset, uma entrada de dados e uma saída de dados. O tamanho da entrada e da saída foi definido como ”generic”para possibilitar o uso do componente em instâncias com tamanhos diferentes.

```
1  LIBRARY IEEE;
2  use ieee.std_logic_1164.all;
3
4  entity Reg is
5
6      generic
7      (
8          W: integer:= 16
9      );
10     port
11     (
12         clock: in std_logic;
13         reset: in std_logic;
14         load: in std_logic;
15         D: in std_logic_vector ((W-1) downto 0);
16         Q: out std_logic_vector ((W-1) downto 0)
17     );
18 end Reg;
19
20 architecture RTL of Reg is
21 begin
22     process(clock, reset, load)
23     begin
24         if (reset = '1') then
25             Q <= (others => '0');
26         elsif (rising_edge(clock) and reset = '0' and load = '1') then
27             Q <= D;
28         end if;
29     end process;
30 end RTL;
```

Figura 5 – Código VHDL do registrador

```
1  LIBRARY IEEE;
2  use ieee.std_logic_1164.all;
3
4  entity tb_Reg is
5  end tb_Reg;
6
7  architecture teste of tb_Reg is
8     component Reg is
9        generic
10    (
11       W : integer:= 16
12    );
13    port
14    (
15       clock: in std_logic;
16       reset: in std_logic;
17       load: in std_logic;
18       D: in std_logic_vector ((W-1) downto 0);
19       Q: out std_logic_vector ((W-1) downto 0)
20     );
21    end component;
22    signal fio_clock: std_logic := '0';
23    signal fio_reset, fio_load: std_logic;
24    signal fio_D, fio_Q: std_logic_vector(3 downto 0);
25    begin
26       instance_regw: Reg generic map (W => 4)
27       port map (clock => fio_clock, D => fio_D, reset => fio_reset, load => fio_load, Q => fio_Q);
28       fio_clock <= not fio_clock after 1 ns;
29       fio_D <= x"8", x"3" after 2 ns, x"F" after 7 ns, x"2" after 12 ns;
30       fio_load <= '0', '1' after 4ns;
31       fio_reset <= '1', '0' after 2 ns, '1' after 7 ns, '0' after 10 ns;
32 end teste;
```

Figura 6 – Código VHDL do testbench do registrador

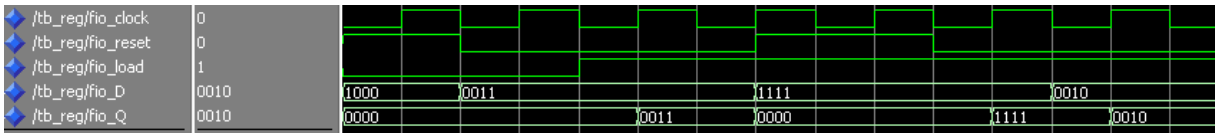

Figura 7 – Resultado da simulação do registrador

2.4.2 Comparador

O projeto do comparador, possui duas entradas de dados e três saídas, igual, maior e menor. As saídas são mantidas em nível lógico baixo e somente são alteradas caso o resultado da respectiva comparação seja verdadeiro. Do mesmo modo do registrador, o tamanhos da entrada de dados foi definido como ”generic”.

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use ieee.numeric_std.all;

entity comparador is
    generic
    (
        DATA_WIDTH : natural := 16
    );
    port
    (
        a   : in std_logic_vector    ((DATA_WIDTH-1) downto 0);
        b   : in std_logic_vector    ((DATA_WIDTH-1) downto 0);
        maior : out std_logic;
        menor : out std_logic;
        igual : out std_logic
    );
end comparador;
architecture comp of comparador is
begin
    maior <= '1' when unsigned(a) > unsigned(b) else
        '0' when unsigned(a) < unsigned(b) else
        '0';

    menor <= '1' when unsigned(a) < unsigned(b) else
        '0' when unsigned(a) > unsigned(b) else
        '0';

    igual <= '1' when unsigned(a) = unsigned(b) else
        '0' when unsigned(a) > unsigned(b) else
        '0';
end comp;
```

Figura 8 – Código VHDL do comparador

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.all;
3  use ieee.numeric_std.all;
4
5  entity tb_comparador is
6  end tb_comparador;
7
8  architecture teste of tb_comparador is
9     component comparador is
10       generic
11       (
12          DATA_WIDTH : natural := 16
13       );
14       port
15       (
16          a   : in std_logic_vector    ((DATA_WIDTH-1) downto 0);
17          b   : in std_logic_vector    ((DATA_WIDTH-1) downto 0);
18          maior : out std_logic;
19          menor : out std_logic;
20          igual : out std_logic
21       );
22    end component;
23
24    signal fio_A, fio_B: std_logic_vector(3 downto 0);
25    signal fio_maior, fio_menor, fio_igual: std_logic;
26
27    begin
28    instancia_comparador: comparador generic map (DATA_WIDTH => 4)
29    port map(a=>fio_A,b=>fio_B,maior=>fio_maior, menor=>fio_menor,igual=>fio_igual);
30
31    fio_A <= x"0", x"8" after 20 ns, x"7" after 40 ns, x"4" after 60 ns;
32    fio_B <= x"0", x"7" after 10 ns, x"8" after 30 ns, x"1" after 50 ns;
33 end teste;
```

Figura 9 – Código VHDL do testbench do comparador

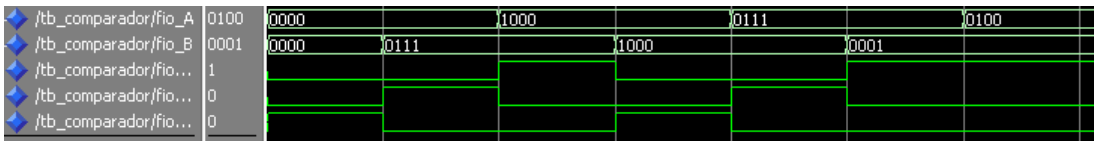

Figura 10 – Resultado da simulação do comparador

2.4.3 CodValido

Este componente é o responsável por verificar a validade do código. Possui uma entrada de 32 bits referente ao código da cor e uma saída que indica a validade. O código é no formato da escala CMYK, desse modo cada posição corresponde a uma cor e deve estar no intervalo de $0 \approx 100$, em decimal. Como a entrada do código é em hexadecimal, o valor deve ir de $0 \approx 64$. O componente então, verifica se a entrada encontra-se nesse intervalo.

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use ieee.numeric_std.all;

entity CodValido is

    port
    (
        cod    : in std_logic_vector    (31 downto 0);
        valido    : out std_logic
    );

end CodValido;

architecture codvalid of CodValido is
begin
    process (cod)
        variable limite : std_logic_vector(31 downto 0);
    begin
        limite := x"64646464";
        if  to_integer(unsigned(cod)) <= to_integer(unsigned(limite)) then
            valido <= '1';
        else
            valido <= '0';
        end if;
    end process;
end codvalid;
```

Figura 11 – Código VHDL do CodValido

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use ieee.numeric_std.all;

entity tb_CodValido is
end tb_CodValido;

architecture teste of tb_CodValido is

    component CodValido is
        port
        (
            cod    : in std_logic_vector    (31 downto 0);
            valido    : out std_logic
        );
    end component;

    signal cod : std_logic_vector (31 downto 0);
    signal valido : std_logic;

    begin

        instance_codvalido: CodValido port map (cod, valido);

        cod <= x"00000000", x"64646465" after 2 ns, x"64646464" after 4 ns, x"641a643f" after 6ns;

end teste;
```

Figura 12 – Código VHDL do testbench do CodValido

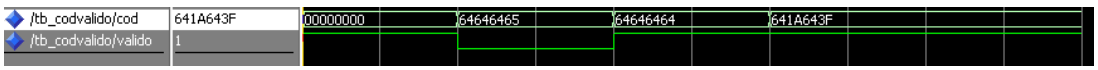

Figura 13 – Resultado da simulação do CodValido

2.4.4 Multiplicador

O multiplicador é responsável por realizar operações de multiplicação de valores, para calcular a quantidade necessária de tempo de cada válvula de cor no processo de mistura da tinta. O funcionamento do multiplicador é baseado no princípio de que cada 1 gota de cor primária equivale ao volume de 0,05 (que é o incremento padrão de pigmento da máquina) de cor primária que é obtida pela abertura da válvula solenóide correspondente a 100 milissegundos.

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use IEEE.NUMERIC_STD.ALL;
4
5   entity multiplicador is
6       Port (
7           A : in  STD_LOGIC_VECTOR (15 downto 0);
8           Produto : out  STD_LOGIC_VECTOR (31 downto 0)
9       );
10  end multiplicador;
11
12  architecture Behavioral of multiplicador is
13  begin
14      Produto <= std_logic_vector(unsigned(A) * 100);
15  end Behavioral;
```

Figura 14 – Código VHDL do multiplicador

```
1   LIBRARY IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use IEEE.NUMERIC_STD.ALL;
4
5   entity tb_multiplicador is
6   end tb_multiplicador;
7
8   architecture teste of tb_multiplicador is
9       component multiplicador is
10          Port (
11              A : in  STD_LOGIC_VECTOR (15 downto 0);
12              Produto : out  STD_LOGIC_VECTOR (31 downto 0)
13          );
14      end component;
15
16      signal fio_A: std_logic_vector(15 downto 0);
17      signal fio_Produto: std_logic_vector(31 downto 0);
18
19  begin
20      instance_mult: multiplicador port map (A => fio_A, Produto => fio_Produto);
21
22      fio_A <= "0000000000000010", "0000000000000011" after 8 ns;
23
24  end teste;
```

Figura 15 – Código VHDL do testbench do multiplicador

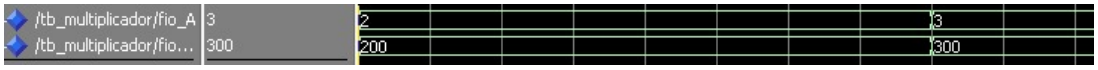

Figura 16 – Resultado da simulação do multiplicador

2.4.5 Contador

O contador é utilizado para rastrear a quantidade de ciclos. No contexto do nosso misturador de tinta, ele é necessário para controlar a duração da abertura das válvulas durante o processo de pigmentação, garantindo precisão na mistura das cores.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity contador is
    port (
        Clock, Reset, E : in STD_LOGIC;
        Q : out STD_LOGIC_VECTOR (15 downto 0)
    );
end contador;

architecture Behavior of contador is
    signal Count : STD_LOGIC_VECTOR (15 downto 0);
begin
    process (Clock, Reset)
    begin
        if (Reset = '1') then
            Count <= (others => '0');
        elsif (Clock'event AND Clock = '1') then
            if E = '1' then
                Count <= Count + 1;
            else
                Count <= Count;
            end if;
        end if;
    end process;
    Q <= Count;
end Behavior;
```

Figura 17 – Código VHDL do contador

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

entity tb_contador is
end tb_contador;

architecture behavior of tb_contador is
    component contador
    port(
         Clock : in  std_logic;
         Reset : in  std_logic;
         E : in  std_logic;
         Q : out  std_logic_vector(15 downto 0)
         );
    end component;

    signal clk : std_logic := '0';
    signal rst : std_logic;
    signal fioE : std_logic;
    signal fioQ : std_logic_vector(15 downto 0);
begin
    instance_cont: contador port map (
          Clock => clk,
          Reset => rst,
          E => fioE,
          Q => fioQ
        );

   clk <= not clk after 1 ms;
   rst <= '1', '0' after 5 ms, '1' after 12 ms;
   fioE <= '0', '1' after 3 ms, '0' after 10 ms;
end;
```

Figura 18 – Código VHDL do testbench do contador

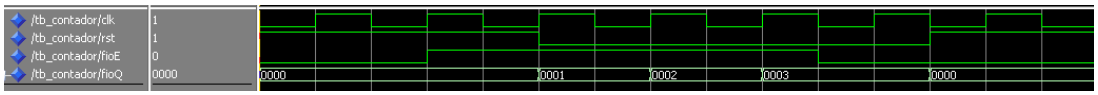

Figura 19 – Resultado da simulação do contador

2.4.6 CompMist

Este componente é uma adaptação do comparador, responsável por garantir que a máquina misture a tinta produzida por um tempo determinado, neste caso, 100s. O comparador recebe um sinal de entrada b e emite ’1’ em sua saída enquanto o sinal é menor que 100.

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use ieee.numeric_std.all;

entity CompMist is

    port
    (
       b   : in std_logic_vector    (7 downto 0);
       menor : out std_logic
    );

end CompMist;

architecture comp of CompMist is
begin

       menor <= '1' when unsigned(b) < 100 else
       '0';

end comp;
```

Figura 20 – Código VHDL do CompMist

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use ieee.numeric_std.all;

entity tb_compmist is
end tb_compmist;

architecture teste of tb_compmist is

   component CompMist is

      port
      (
         b   : in std_logic_vector    (7 downto 0);
         menor : out std_logic
      );

   end component;

   signal fio_B: std_logic_vector(7 downto 0);
   signal fio_menor: std_logic;

   begin

   instancia_compmist: CompMist port map(b=>fio_B,menor=>fio_menor);

   fio_B <= x"00", x"65" after 10 ns, x"08" after 30 ns, x"5f" after 50 ns;
end teste;
```

Figura 21 – Código VHDL do testbench do CompMist

Figura 22 – Resultado da simulação do CompMist

3 SIMULAÇÃO

Após a realização do projeto, a fase de testes foi iniciada. O intuito é comparar os resultados da simulação com o objetivo proposto. Ao compilarmos os códigos propostos nesse documento no Quartus, pudemos verificar que os resultados obtidos foram coerentes ao desejado.

A figura 23 demonstra o diagrama do misturador de tinta gerado a partir dos componentes desenvolvidos no trabalho. É notável que a comunicação entre o bloco do caminho de dados e o bloco da controladora está condizente com o esperado e demonstrado na figura 2.

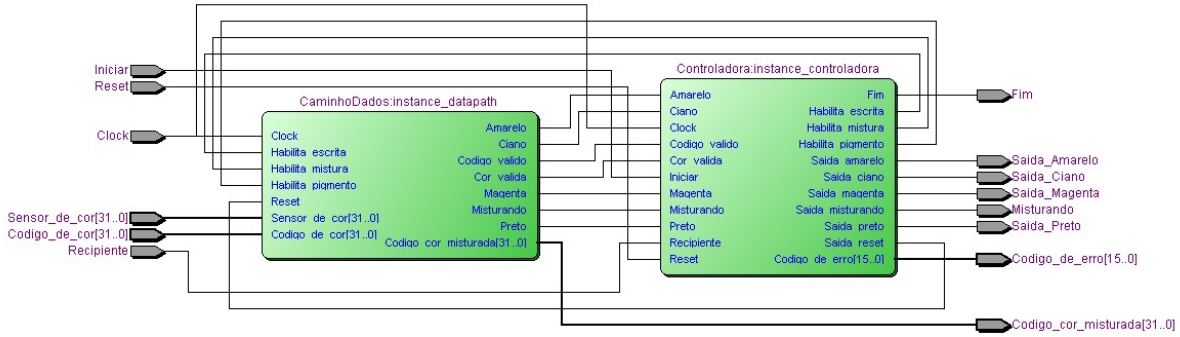

Figura 23 – Diagrama do misturador de tinta pelo Quartus.

A partir da compilação, também é gerado um diagrama para a máquina de estados finitos proposta (24). Comparando com a máquina de estados finitos de alto nível da figura 3, fica evidente que a ordem proposta é a que está sendo seguida pela máquina real.

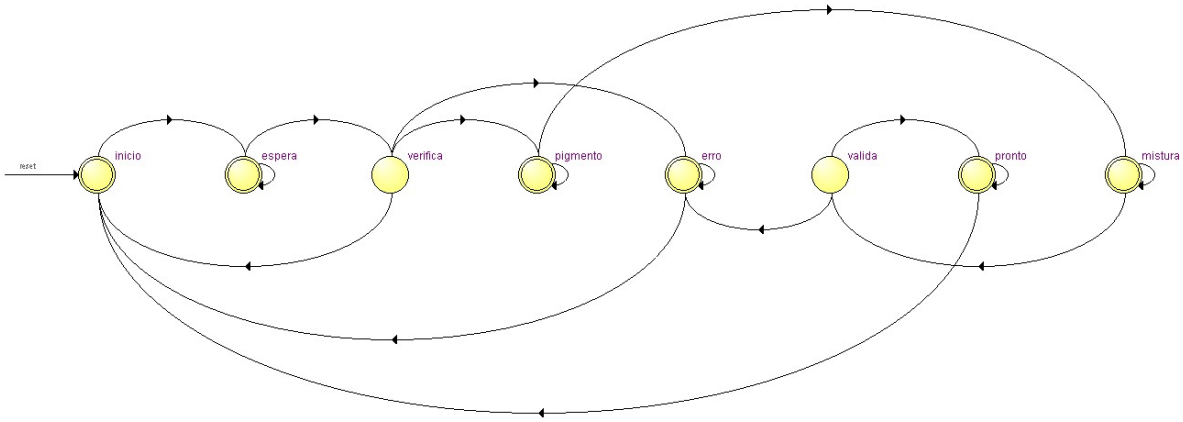

Figura 24 – Diagrama da máquina de estados finitas pelo Quartus.

Nesse caso, também resulta em uma lista de sinais (figura 25) envolvidos pela FSM durante as transições. Nessa lista estão relacionados o estado fonte do sinal, o estado de destino do sinal e em qual condição o a transição da máquina de estados de fato aconteceria. A conferência com o diagrama da figura 3 também mostra que o funcionamento está de acordo com o esperado.

	Source State	Destination State	Condition
1	erro	inicio	(Reset)
2	erro	erro	(!Reset)
3	espera	verifica	(Iniciar).(Recipiente)
4	espera	espera	(!Iniciar) + (Iniciar).(!Recipiente)
5	inicio	espera	
6	mistura	valida	(!Misturando)
7	mistura	mistura	(Misturando)
8	pigmento	pigmento	(!Ciano) + (Ciano).(!Magenta) + (Ciano).(Magenta).(!Amarelo) + (Ciano).(Magenta).(Amarelo).(!Preto)
9	pigmento	mistura	(Ciano).(Magenta).(Amarelo).(Preto)
10	pronto	pronto	(Recipiente)
11	pronto	inicio	(!Recipiente)
12	valida	pronto	(Cor_valida)
13	valida	erro	(!Cor_valida)
14	verifica	pigmento	(!Reset).(Codigo_valido)
15	verifica	inicio	(Reset).(Codigo_valido)
16	verifica	erro	(!Codigo_valido)

Figura 25 – Lista de sinais da FSM pelo Quartus.

A figura 26 é o resultado do RTL Viewer dos códigos desenvolvidos. A comparação é com base na figura 4.

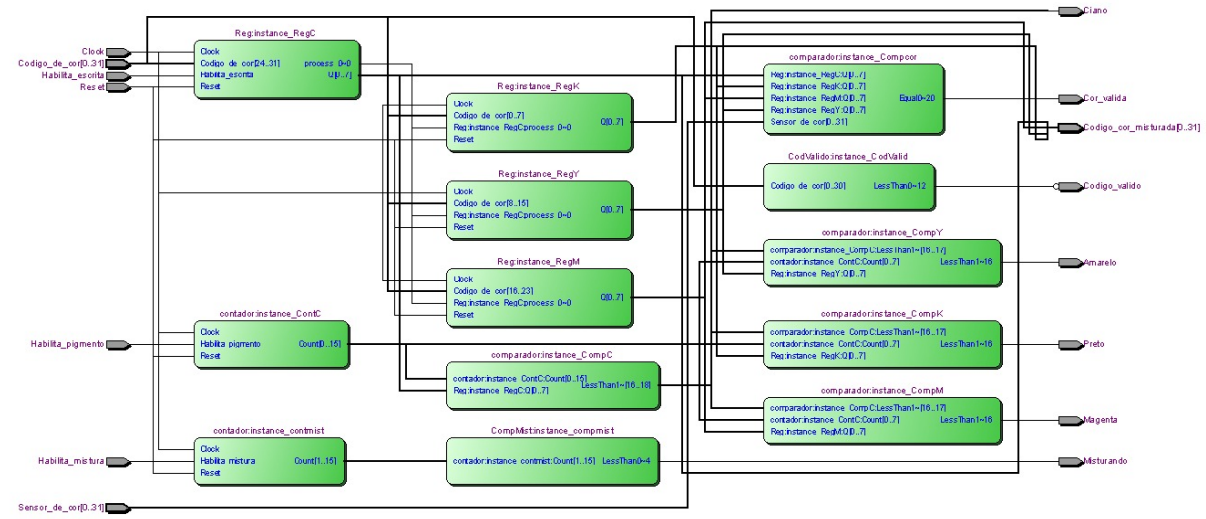

Figura 26 – Diagrama do caminho de dados pelo Quartus.

A figura abaixo (figura 27) demonstra o diagrama do bloco da controladora.

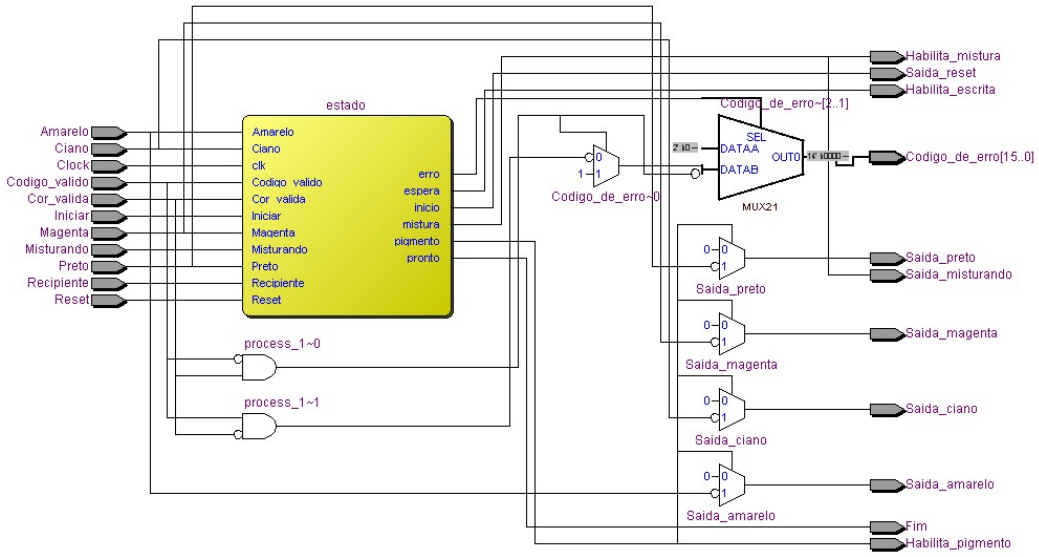

Figura 27 – Diagrama da controladora pelo Quartus.

A partir disso, o testbench foi compilado e a partir do Modelsim vemos os valores dos sinais gerados ao longo do tempo. A primeira parte do teste, mostrada na figura 28, representa um teste de erro no código de cor da tinta. O código de entrada, ”6F7260F9”é um código inválido pois ultrapassa o valor máximo do sistema CMYK. No terceiro quadrante da imagem, é possível ver que a máquina passa para o estado de verificação, e ao ver que o código é inválido, passa para o estado de erro. Nisso, a máquina aguarda a subida do sinal de reset para voltar ao estado de início.

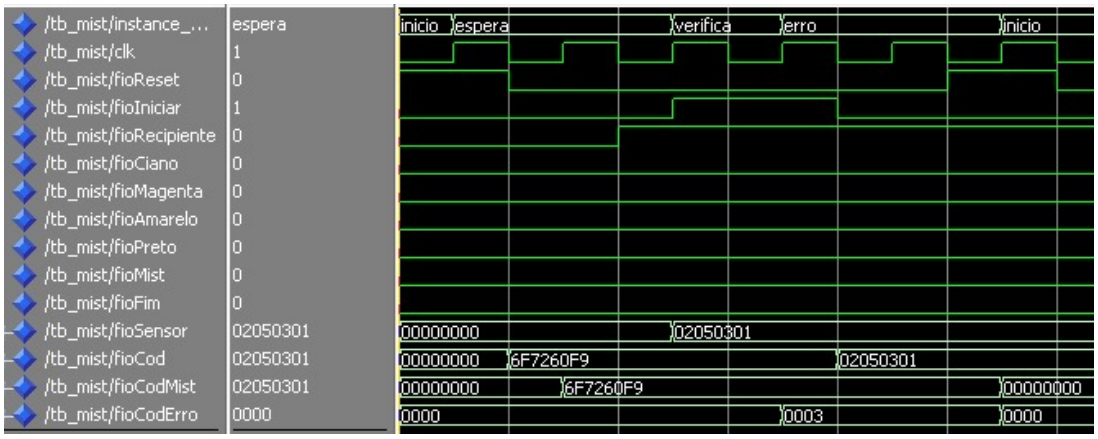

Figura 28 – Parte de verificação do código.

Com o reinício do processo, um novo código é inserido na máquina, mas dessa vez sendo válido. Sendo assim, o código passa pela verificação e em seguida o processo de pigmentação é iniciado. Cada cor é liberada pelo tempo necessário para alcançar o código de entrada por seu respectivo sinal. Com o acréscimo dos pigmentos, a tinta é misturada.

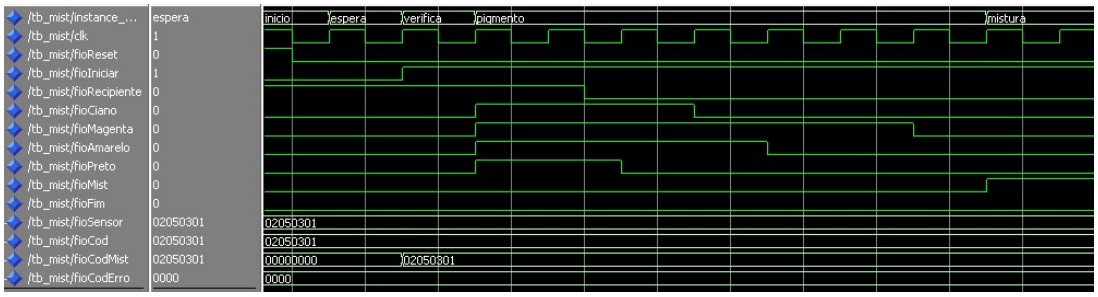

Figura 29 – Parte de verificação do funcionamento da máquina.

Após o tempo de mistura, a cor é validada em comparação ao código inserido e passa para o estado de pronto. A partir desse momento, é esperado que a lata de tinta seja retirada para retornar ao estado de início e recomeçar o ciclo.

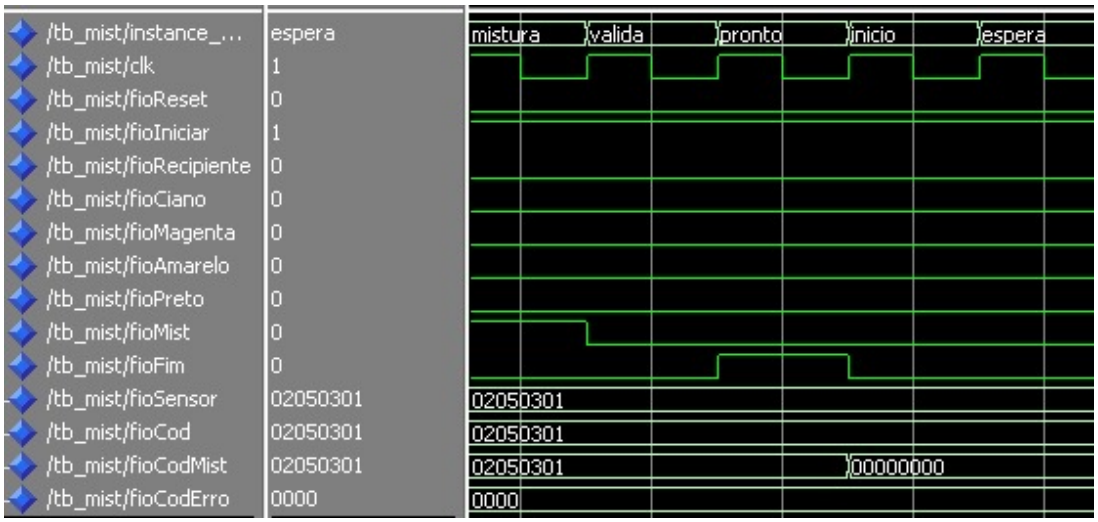

Figura 30 – Parte final do teste.

Referências

[1] ≪ https://www.printi.com.br/blog/cores-cmyk-no-sistema-de-impressao ≫

[2] ≪ https://tecnoblog.net/responde/o-que-sao-os-padroes-de-cores-rgb-e-cmyk/ ≫

[3] ≪ https://www.people.com.br/noticias/design/qual-a-diferenca-entre-cmyk-e-rgb ≫