UNIVERSIDADE FEDERAL DO CEARÁ – CAMPUS QUIXADÁ

BACHARELADO EM ENGENHARIA DE COMPUTAÇÃO

DISCIPLINA DE SISTEMAS DIGITAIS

FRANCISCO DE ASSIS PAIVA NETO – 499740

ROBERT VINÍCIUS OLIVEIRA GONÇAVES – 495670

KÁSSIA CRISTINA DE SOUSA LOPES – 493657

HUGO SANTOS DA COSTA BESSA – 496870

OPERAÇÕES DE PROCESSAMENTO DIGITAL DE IMAGENS EM VHDL

QUIXADÁ, CEARÁ

2022

1. Introdução

Uma imagem pode ser conceituada como uma função bidimensional f(x, y), onde x e y representam as coordenadas espaciais do plano e f aplicada a esses pontos define a função mostrada. Desse modo, conseguimos estabelecer relações entre a forma como uma imagem é construída, possibilitando a manipulação de seus pixels/coordenadas.

O estudo disso é feito através do "Processamento Digital de Imagens", área que vem inovando e conseguindo seu espaço. Dessa forma, com intuito de promover um breve com ela, foi proposto o desenvolvimento de um algoritmo que configurasse a imagem em algumas formas.

Foi utilizado para a construção dessa atividade a programa Vivado 2020.1 em conjunto da placa Zybo, sendo sugerido a implementação dos seguintes módulos: Binarização, Tons de Cinza, Suavização, Rotação, Filtro de Sobel, Ajuste de Brilho, Sal e Pimenta, Redimensionamento e RGB.

2. Binarização

Esse processo se baseia nos níveis de cinza que compõem uma determinada imagem. Nas instruções para sua implementação, tivemos o threshhold definido como 100, com isso é possível isolar a imagem em dois grupos de pixels: valores acima e abaixo do limiar.

```
when exec_binar =>
  s_exec_start <= '1';
  ram_we <= '1';
  s_mem_addr <= pixel_addr;
  rom_addr <= pixel_addr;
  data_out <=  s_bin;

  if(s_done='1') then
    next_s <= init;
  else
    next_s <= exec_binar;
  end if;
```

```
s_din <= din;
dout <= X"FFFFFF" when s_din(15 downto 8) > 100 else X"000000";
```

3. Tons de cinza

Conseguimos a aplicação de tons de cinza ao realizarmos um novo balanceamento entre o RGB do pixel. Dada a circunstância, nosso pixel assume a seguinte estrutura: P = R(30%) + G(59%) + B(11%); onde iremos considerar 30% = 40/128, 59% = 74/12 e 11% = 14/128.

```
    mult_r: entity work.multiplicador
        generic map(N => 8)
        port map( A => s_mem_din(23 downto 16),
                  B => X"28",
                  S => s_red_big);


    mult_g: entity work.multiplicador
        generic map(N => 8)
        port map( A => s_mem_din(15 downto 8),
                  B => X"4A",
                  S => s_green_big);

    mult_b: entity work.multiplicador
        generic map(N => 8)
        port map( A => s_mem_din(7 downto 0),
                  B => X"0E",
                  S => s_blue_big);

        shift_r: entity work.shift_direita
            generic map (N => 16)
            port map( A => s_red_big,
                      B => x"0007",
                      S =>  s_red
                     );

        shift_g: entity work.shift_direita
            generic map (N => 16)
            port map( A => s_green_big,
                      B => x"0007",
                      S =>  s_green
                     );

        shift_b: entity work.shift_direita
            generic map (N => 16)
            port map( A => s_blue_big,
                      B => x"0007",
                      S =>  s_blue
                     );

s_res <= (s_red + s_green + s_blue);

dout <= (s_res(7 downto 0) & s_res(7 downto 0) & s_res(7 downto 0));
```

4. Suavização

São filtros que utilizam um algoritmo com uma função linear para sua aplicação, sendo utilizados para borramentos e redução de ruídos.

5. Rotação

Através dessa configuração, conseguimos girar a imagem em um determinado ângulo. Para a atividade sugerida, devemos rotacionar a imagem na tela 4 vezes, onde cada rotação corresponde a uma rotação de 90° e sempre realizando uma espera de 1 segundo entre cada rotação.

6. Negativo

Nesse filtro é feito um mapeamento inverso da imagem, onde é aplicado um contraste para que as áreas escuras se tornem claras e vice-versa.

```
when exec_negat =>
  s_exec_start <= '1';
  ram_we <= '1';
  rom_addr <= pixel_addr;
  s_mem_addr <= pixel_addr;
  data_out <=  s_neg;

  if(s_done='1') then
    next_s <= init;
  else
    next_s <= exec_negat;
  end if;

s_mem_din <= din;

s_red <= (X"FF" - s_mem_din(23 downto 16)) ;
s_green <= (x"FF" - s_mem_din(15 downto 8));
s_blue <= (X"FF" - s_mem_din(7 downto 0)) ;

dout <= (s_red & s_green & s_blue);
```

7. Filtro de Sobel

Esse filtro é aplicado em algoritmos de detecção de contornos, que se fundamenta num operador para o cálculo das diferenças finitas da imagem, aproximando o gradiente da intensidade dos pixels da imagem. Com isso, para cada ponto da imagem teremos o resultado da aplicação do Sobel devolvendo o gradiente ou a norma do vector.

8. Ajuste de brilho

Para a aplicação desse filtro, consideramos dois possíveis estados para a imagem: original e preto. Assim, para sua aplicação temos o ajuste do brilho até que a imagem fique em um estado totalmente escuro.

```
when exec_baixa_brilho =>
          s_init_brilho <= '1';
          s_exec_start <= '1';
          ram_we <= '1';
          s_mem_addr <= pixel_addr;
          rom_addr <= pixel_addr;
          data_out <=  s_brilho;
          s_brilho_op <= '0';
          control_brilho <= std_logic_vector(to_unsigned(brilho, 16));

      if (rst = '1') then
          next_s <= init;
       end if;

       if (s_done_baixa_brilho = '1') then
          next_s <= exec_sobe_brilho;
       else
          next_s <= exec_baixa_brilho;
       end if;

when exec_sobe_brilho =>
       s_init_brilho <= '1';
       s_mem_addr <= pixel_addr;
       rom_addr <= pixel_addr;
       data_out <=  s_brilho;
       s_exec_start <= '1';
       ram_we <= '1';
       s_brilho_op <= '1';
       control_brilho <= std_logic_vector(to_unsigned(brilho, 16));

       if (rst = '1') then
           next_s <= init;
       end if;

       if (s_done_aumenta_brilho = '1') then
              next_s <= init;
       else
              next_s <= exec_sobe_brilho;
       end if;

     begin
         if (rising_edge(clk)) then
            if(s_init_brilho = '1') then
                 contador_de_pulsos_brilho <= contador_de_pulsos_brilho + 1;
                 if (contador_de_pulsos_brilho >= 5000000) then
                      contador_de_pulsos_brilho <= 0;
                         if(brilho > 1 and s_brilho_op = '0') then
                             s_done_baixa_brilho <= '0';
                             brilho <= brilho - 2;
                         elsif(s_brilho_op = '0') then
                             s_done_baixa_brilho <= '1';
                         end if;
                         if(brilho < 256 and s_brilho_op = '1') then
                            s_done_aumenta_brilho <= '0';
                            brilho <= brilho + 2;
                         elsif(s_brilho_op = '1') then
                            s_done_aumenta_brilho <= '1';
                         end if;
                 end if;
              else
                 contador_de_pulsos_brilho <= 0;
                 s_done_baixa_brilho <= '0';
                 s_done_aumenta_brilho <= '0';
              end if;
           end if;
     end process;
```

9. Sal e pimenta

Esse filtro mostra ruídos encontrados na imagem em formato circular, nas cores preto e branco.

```
        when exec_sal_pimenta =>
          s_exec_start <= '1';
          s_mem_addr <= pixel_addr;
          s_init_count <= '0';
          ram_we <= '1';
          rom_addr <= pixel_addr;
          data_out <=  din;
          if(s_done='1') then
            next_s <= derrama_sal;
          else
            next_s <= exec_sal_pimenta;
          end if;
when derrama_sal =>
  s_exec_start <= '0';
  s_init_count <= '1';
  data_out <= x"ffffff";
  ram_we <= '1';
  s_rand <= ((s_rand xor x"ECECECEC") + s_count_rand);
  s_mem_addr <= s_rand(14 downto 0);
  next_s <= derrama_pimenta;

when derrama_pimenta =>
  s_exec_start <= '0';
  s_init_count <= '1';
  data_out <= x"000000";
  s_rand <= ((s_rand xor x"DF0F0FDF") + s_count_rand);
  s_mem_addr <= s_rand(14 downto 0);
  ram_we <= '1';

  if(s_count_clocks < 1000) then
      next_s <= derrama_sal;
    else
      next_s <= init;
    end if;
```

10. Redimensionamento

Para esse filtro, teremos a redução do tamanho da imagem em 50%.

11. RGB

Para sua aplicação, alternaremos os canis entre R, G e B com um segundo intervalo até o botão de start ser iniciado.

```
when exec_red =>
 s_exec_start <= '1';
 s_mem_addr <= pixel_addr;
 sel_mux_rgb <= "00";
 ram_we <= '1';
 s_init_count <= '0';
 rom_addr <= pixel_addr;
 data_out <=  s_rgb;
 s_loop <= '0';
if(start = '1') then
     next_s <= espera0;
     s_init_count <= '0';
 elsif(s_count_clocks < 100000000) then
     s_init_count <= '1';
     next_s <= exec_red;
 else
     s_init_count <= '0';
     next_s <= exec_green;
 end if;

when exec_green =>
   s_exec_start <= '1';
   s_mem_addr <= pixel_addr;
   ram_we <= '1';
   sel_mux_rgb <= "01";

   rom_addr <= pixel_addr;
   data_out <=  s_rgb;
   s_loop <= '0';
 if(start = '1') then
    next_s <= espera0;
    s_init_count <= '0';
 elsif(s_count_clocks < 100000000) then
    s_init_count <= '1';
    next_s <= exec_green;
 else
    s_init_count <= '0';
    next_s <= exec_blue;
 end if;
```

```
        when exec_blue =>
            s_exec_start <= '1';
            s_mem_addr <= pixel_addr;
            s_loop <= '0';
            ram_we <= '1';
            sel_mux_rgb <= "10";

            rom_addr <= pixel_addr;
            data_out <=  s_rgb;

            if(start = '1') then
              next_s <= espera0;
              s_init_count <= '0';
            elsif(s_count_clocks < 100000000) then
              s_init_count <= '1';
              next_s <= exec_blue;
            else
              s_init_count <= '0';
              next_s <= exec_red;
            end if;

with s_mux_rgb select
s_dataout <= s_datain(23 downto 16)&x"0000" when "00",
             x"00"&s_datain(15 downto 8)&x"00" when "01",
             x"0000"&s_datain(7 downto 0) when "10",
             s_datain when others;

dout <= s_dataout;
```