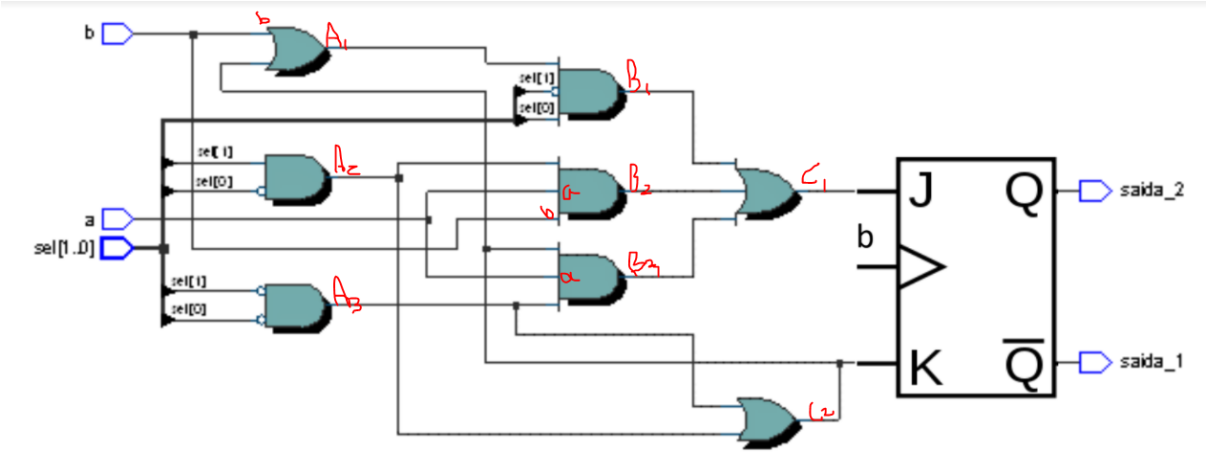
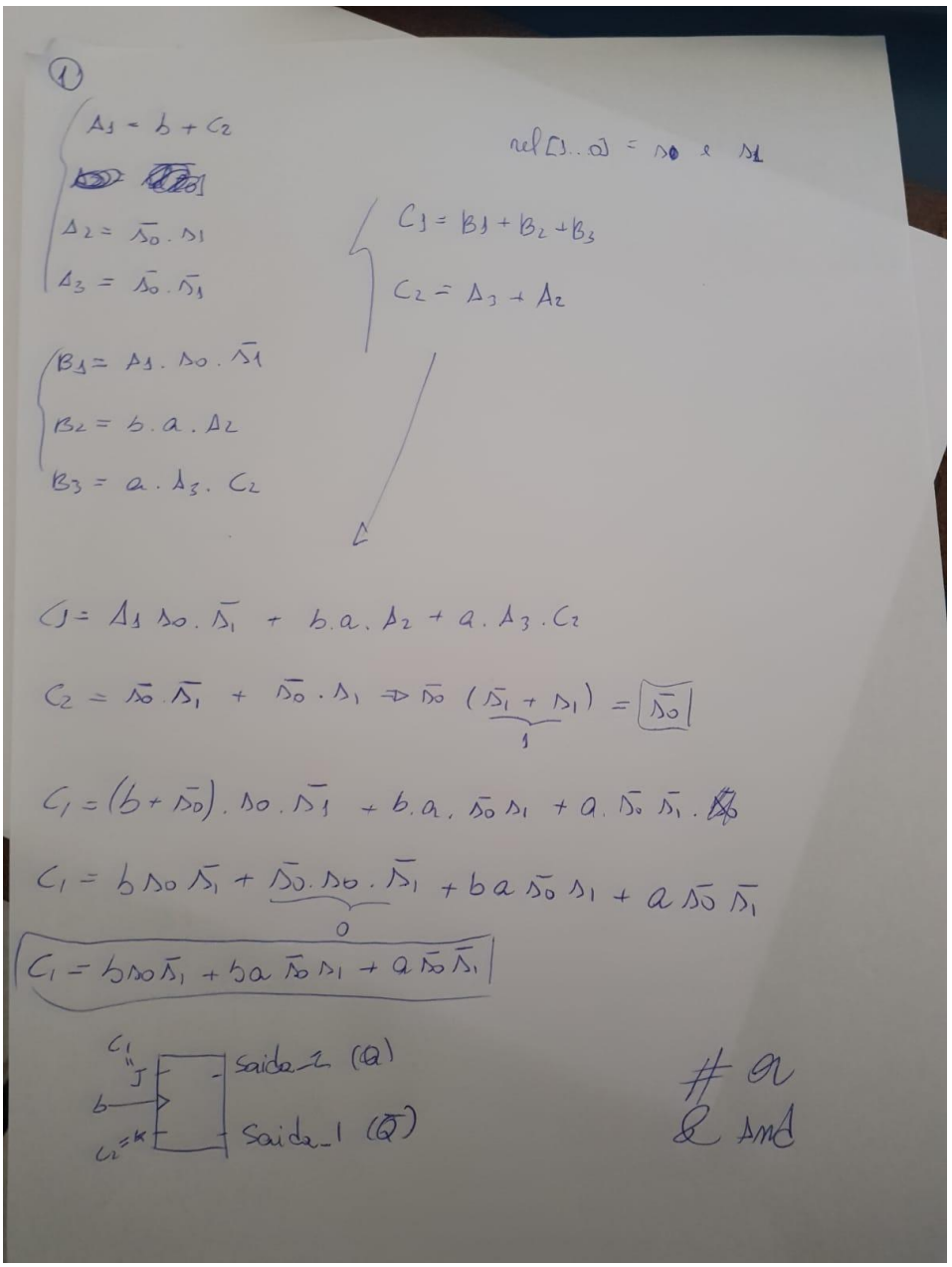
Aluno: Pedro Gomes

Q1a) Dado o sistema abaixo, foi realizado uma otimização.





Após isso foi implementado em AHDL:

```
SUBDESIGN questao_1
(  sel[1..0] : INPUT;
   b, a : INPUT;
   saida_2, saida_1 : OUTPUT;

)
VARIABLE
   ff_jk : JKFF;
   nsel[1..0] : NODE; --seletor negado

BEGIN

   ff_jk.clk=b;

   nsel[0]=!(sel[0]);
   nsel[1]=!(sel[1]);

   saida_2=ff_jk.q;
   saida_1=!(ff_jk.q);

   ff_jk.k= nsel[0];

   IF (b & sel[0] & nsel[1]) # (b & a & nsel[0] & sel[1]) # ( a & nsel[0] & nsel[1]) THEN

      ff_jk.j=VCC;
   ELSE
      ff_jk.j=GND;

   END IF;
END;
```

Q1b)

```
SUBDESIGN questao_1b
(  A, B, C : INPUT; --select
   G1, G2A, G2B : INPUT; --enable
    Y0,Y1,Y2,Y3,Y4,Y5,Y6,Y7 : OUTPUT;
)
VARIABLE
   select[2..0] : NODE;
   ys[7..0] :  NODE;

BEGIN
   select[0]=A;
   select[1]=B;
   select[2]=C;

   Y0=ys[0];
   Y1=ys[1];
   Y2=ys[2];
   Y3=ys[3];
   Y4=ys[4];
   Y5=ys[5];
   Y6=ys[6];
   Y7=ys[7];

   IF !(G2A) & !(G2B) & G1 THEN

   CASE select[] IS

      WHEN B"000" => ys[]=B"01111111";
      WHEN B"001" => ys[]=B"10111111";
      WHEN B"010" => ys[]=B"11011111";
      WHEN B"011" => ys[]=B"11101111";
      WHEN B"100" => ys[]=B"11110111";
      WHEN B"101" => ys[]=B"11111011";
      WHEN B"110" => ys[]=B"11111101";
      WHEN B"111" => ys[]=B"11111110";

   END CASE;
   ELSE

      ys[]=B"11111111";

   END IF;
END;
```

Q2)
Dividi o processo em interpretar os digitos digitados, organizar os digitos na tela, colocando o digito inserido na direita, decodificadores dos digitos para o 7seg, e o funcionamento estrutural do microondas (contagem regressiva, checagem de porta fechada, botao de stop e clear).

```vhdl
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
use IEEE.numeric_std.all;

ENTITY questao_2 IS
    PORT(
    teclado : in std_logic_vector(9 DOWNTO 0);
    clock_in : in std_logic;
    startn, stopn, clearn, door_closed : in std_logic;

    mag_on : out std_logic;
    min_segs, sec_tens_segs, sec_ones_segs : out std_logic_vector(6 downto 0)

    );
END questao_2;

ARCHITECTURE vhd1 OF questao_2 IS
VARIABLE dig1, dig2, dig3: INTEGER RANGE 0 TO 9;
BEGIN


    PROCESS (teclado, dig1, dig2, dig3, dig)
    VARIABLE dig: INTEGER RANGE 0 TO 9;
    signal en: std_logic;
    BEGIN

     --interpretador do digito do teclado --
       CASE teclado IS
          WHEN "1000000000" => dig <=0;
          WHEN "0100000000" => dig <=1;
          WHEN "0010000000" => dig <=2;
          WHEN "0001000000" => dig <=3;
          WHEN "0000100000" => dig <=4;
          WHEN "0000010000" => dig <=5;
          WHEN "0000001000" => dig <=6;
          WHEN "0000000100" => dig <=7;
          WHEN "0000000010" => dig <=8;
          WHEN "0000000001" => dig <=9;
          WHEN OTHERS => dig <= 0;
       END CASE;

       --faz o digito andar pra frente e insere outro no mais a direita
       IF not(dig='0') and en='1' THEN
          en:=0;
          IF not(dig1='0') THEN
             IF not(dig2='0') THEN
                dig3:=dig2;
                dig2:=dig1;
                dig1:=dig;

             ELSE
                dig2:=dig1;
                dig1:=dig;

             END IF;
          ELSE
          dig1:=dig;
          dig:=0;
          END IF;
       ELSE
          en:=1;
       END IF;
```

```
--decodificador dos digitos--
CASE dig1 IS
    WHEN 0 => sec_ones_segs <="0000001";
    WHEN 1 => sec_ones_segs <="1001111";
    WHEN 2 => sec_ones_segs <="0010010";
    WHEN 3 => sec_ones_segs <="0000110";
    WHEN 4 => sec_ones_segs <="1001100";
    WHEN 5 => sec_ones_segs <="0100100";
    WHEN 6 => sec_ones_segs <="0100000";
    WHEN 7 => sec_ones_segs <="0001111";
    WHEN 8 => sec_ones_segs <="0000000";
    WHEN 9 => sec_ones_segs <="0000100";
END CASE;

CASE dig2 IS
    WHEN 0 => sec_tens_segs <="0000001";
    WHEN 1 => sec_tens_segs <="1001111";
    WHEN 2 => sec_tens_segs <="0010010";
    WHEN 3 => sec_tens_segs <="0000110";
    WHEN 4 => sec_tens_segs <="1001100";
    WHEN 5 => sec_tens_segs <="0100100";
    WHEN 6 => sec_tens_segs <="0100000";
    WHEN 7 => sec_tens_segs <="0001111";
    WHEN 8 => sec_tens_segs <="0000000";
    WHEN 9 => sec_tens_segs <="0000100";
END CASE;

CASE dig3 IS
    WHEN 0 => min_segs <="0000001";
    WHEN 1 => min_segs <="1001111";
    WHEN 2 => min_segs <="0010010";
    WHEN 3 => min_segs <="0000110";
    WHEN 4 => min_segs <="1001100";
    WHEN 5 => min_segs <="0100100";
    WHEN 6 => min_segs <="0100000";
    WHEN 7 => min_segs <="0001111";
    WHEN 8 => min_segs <="0000000";
    WHEN 9 => min_segs <="0000100";
END CASE;
END PROCESS;
```

```
    --funcionamento do microondas

PROCESS (cllock_in, dig1, dig2, dig3, startn, stopn, clearn, door_closed)
signal aux: std_logic;
BEGIN
    IF startn and door_closed and (not stopn) THEN
        mag_on<=1;
        IF rising_edge(clock_in) THEN --contagem regressiva
              IF not dig1='0' THEN
                 dig1:=dig1-1;
              ELSE
                 IF not dig2='0' THEN
                    dig2:=dig2-1;
                 ELSE
                    IF not dig3='0' THEN
                       dig3:=dig3-1;
                    ELSE
                    END IF;
                 END IF;
              END IF;
        END IF;
    ELSE
        mag_on<=0;
    END IF;
    IF clearn THEN
        dig1:=0;
        dig2:=0;
        dig3:=0;
    END IF;
END vhd1;
```

Q3)

Declaração de entradas e saidas:

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
use IEEE.numeric_std.all;

ENTITY questao_3 IS
PORT( nickel_in, dime_in, quarter_in : in std_logic;
      clk, rst : in std_logic;
      nickel_out, dime_out, candy_out : out std_logic

   );
END questao_3;
```

Calculo do dinheiro inserido:

- os aux's estão sendo utilizados para garantir somente uma entrada no laço por moeda inserida

```
ARCHITECTURE vhd2 OF questao_3 IS
VARIABLE din : INTEGER RANGE 45 TO 0;
BEGIN
    PROCESS(nickel_in, dime_in, quarter_in, rst) --calcula o dinheiro inserido
    signal aux1, aux2, aux3 : std_logic;
    BEGIN
       IF quarter_in='1' THEN
          aux1:=0;
          din:=din+25;
       ELSE
          aux1:=1;
       END IF;
       IF dime_in='1' THEN
          aux2:=0;
          din:=din+10;
       ELSE
          aux2:=1;
       END IF;
       IF nickel_in='1' THEN
          aux3:=0;
          din:=din+5;
       ELSE
          aux3:=1;
       END IF;
       IF rst=1 THEN
          din:=0;
       END IF;
    END PROCESS;
```

Cálculo do troco e compra do candy:

```
PROCESS (clk)
signal count : integer range 0 to 1000;
signal troco5, troco10, troco15, troco20 : std_logic;
BEGIN
      IF din>=25 THEN
         din:=din-25;
         candy_out<=1;

         IF din>=10 THEN --troco pra 10

            din:=din-10;
            troco10:=1;

            IF din>=10 THEN --troco pra 20
               din:=din-10;
               troco20:=1;
               troco10:=0;
            END IF;

            IF din>=5 THEN --troco pra 15
               din:=din-5;
               troco15:=1;
               troco10:=0;
            END IF;
         ELSE
            IF din>=5 THEN --troco pra 5
               troco5:=1;
            END IF;
         END IF;
      END IF;
```

Há uma contagem de 1000 subidas de clock para temporizar a saída das moedas e do candy e tornar possível dar trocos utilizando mais de uma moeda do mesmo tipo.

```
        IF rising_edge(clk) AND candy_out=1 THEN
            IF count=1000 THEN --time do candy
                candy_out<=0;
                count:=0;
            ELSE
                count:=count+1;
            END IF;
        END IF;
        IF rising_edge(clk) AND troco5=1 THEN
            IF count=1000 THEN --troco de 5
                candy_out<=0;
                count:=0;
                nickel_out<=0;
                troco5:=0;
            ELSE
                count:=count+1;
                nickel_out<=1;
            END IF;
        END IF;
        IF rising_edge(clk) AND troco10=1 THEN
            IF count=1000 THEN --troco de 10
                candy_out<=0;
                count:=0;
                dime_out<=0;
                troco10:=0;
            ELSE
                count:=count+1;
                dime_out<=1;
            END IF;
        END IF;
        IF rising_edge(clk) AND troco20=1 THEN
            IF count=1000 THEN --troco de 20
                candy_out<=0;
                count:=0;
                dime_out<=0;
                troco20:=0;
                troco10:=1;
            ELSE
                count:=count+1;
                dime_out<=1;
            END IF;
        END IF;
    END PROCESS;
END vhd2;
```

Q4)

Um sinal analogico representa infinitos valores dentro de um tempo relativamente curto, para amostrar estes sinais e trabalhar com eles, normalmente utilizamos um sinal digital quantizado e amostrado a uma taxa de amostragem que a depender de qual a necessidade iremos usar a taxa pode ser definida de modo que não altere as características básicas do sinal.
Para modificar a taxa de amostragem de um sinal, devemos diminuir a frequência com que retiramos as amostras do sinal principal, mas deve-se tomar cuidado com essa diminuição, pois de acordo com o critério de nyquist a menor frequência que o sinal pode ser amostrado é 2x a frequência do sinal, para sinais limitados no tempo, temos sinais aperiódicos, e consequentemente difíceis de estimar um valor de frequência mínima, mas podemos aproximá-los para sinais periódicos e estimá-los.

Sinais digitais são quantizados, visto que apresentam uma taxa de amostragem finita, basicamente o período entre amostras é constante e se aproximarmos o suficiente do

gráfico conseguimos observar uma espécie de escada, essa escada representa a quantização do sinal.

Q5)
Utilizei um always para fazer a interpretação do estado para a saída e outro always para a mudança de estado como o pulso de clock.
Como não foi especificado nenhuma entrada para controle de meio passo ou passo completo, o projeto foi dimensionado para meio passo, visto que o controle é mais preciso.

```
module questao_5 (clk, sentido, out);

input clk, sentido;
output reg[3:0]out;

//sentido=1 ~> rotação sentido horario
//sentido=0 ~> rotação sentido antihorario

reg[3:0]count;

parameter s1=1, s2=2, s3=3, s4=4, s5=5, s6=6, s7=7, s8=8;

always@(count) begin

case(count)
    s1:out=4'b1000;
    s2:out=4'b1100;
    s3:out=4'b0100;
    s4:out=4'b0110;
    s5:out=4'b0010;
    s6:out=4'b0011;
    s7:out=4'b0001;
    s8:out=4'b1001;
endcase
end
```

```
always @ (posedge clk) begin

   case(count)

   s1:if(sentido)
         count<=s2;
      else
         count<=s8;
   s2:if(sentido)
         count<=s3;
      else
         count<=s1;
   s3:if(sentido)
         count<=s4;
      else
         count<=s2;
   s4:if(sentido)
         count<=s5;
      else
         count<=s3;
   s5:if(sentido)
         count<=s6;
      else
         count<=s4;
   s6:if(sentido)
         count<=s7;
      else
         count<=s5;
   s7:if(sentido)
         count<=s8;
      else
         count<=s6;
   s8:if(sentido)
         count<=s1;
      else
         count<=s7;
   endcase
end

endmodule
```