Disciplina: PCS 3335 – Laboratório Digital A	
Prof.: *Anarosa*	**Data:** *10/04/2020*
Turma: *01*	**Bancada:** *A2*
Membros:	
Andre Devay Torres Gomes (10770089)	
FIZ O EXPERIMENTO SOZINHO	

Experiência *[N]*

[Título da Experiência]

1. Introdução

Esta experiência traz o conceito da estruturação de um projeto em sistemas digitais. Para projetos de grande escalas, comumente subdividi-se o projeto em sua unidade de controle e fluxo de dados, além de divisões com a parte combinatória/sequencial, etc. A ideia, portanto, é conseguirmos desenvolver sistemas digitais mais complexos, compostos por fluxo de dados e unidade de controle em um projeto sobre semáforos de trânsito.

2. Objetivo

O intuito desta experiência é que nós consigamos desenvolver e entender o conceito de Fluxo de Dados e Unidade de Controle; Máquina de estados e diagrama de transição de estados; Projeto Completo de um sistema digital.

3. Planejamento

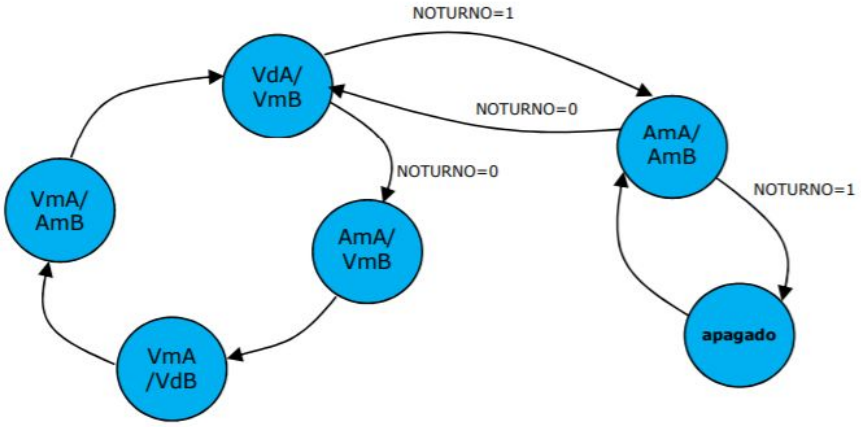
Projeto de um Sistema de Controle de Semáforos

a. Projeto

I. Descrição Funcional (e Máquina de Estados)

Para esta atividade fomos pedidos para construir um sistema digital de controle de semáforo que fosse constituído de uma lógica combinatória, uma lógica sequencial e um circuito temporizador. A ideia é trabalhar com variáveis do contexto como a presença ou não de veículos nas vias, as luzes dos semáforos, a indicação se está de madrugada e o reset do sistema.

Sabendo que A e B são as vias que se cruzam e que iremos trabalhar, temos o seguinte diagrama para de transição de estados:



Percebe-se que há dois modos de operações desses semáforos, um noturno e um diurno. Ao se acionar o noturno, ambos semáforos irão piscar amarelo e desligar simultaneamente em um ciclo infinito até que seja desativado o input NOTURNO. No modo diurno, temos 4 estados que se alternam em um ciclo contínuo. Onde se utiliza o verde de um com o vermelho de outro e, em seguida o amarelo de um com o mesmo vermelho do outro (na outra metade do ciclo se inverte os papéis, mas o funcionamento é igual).

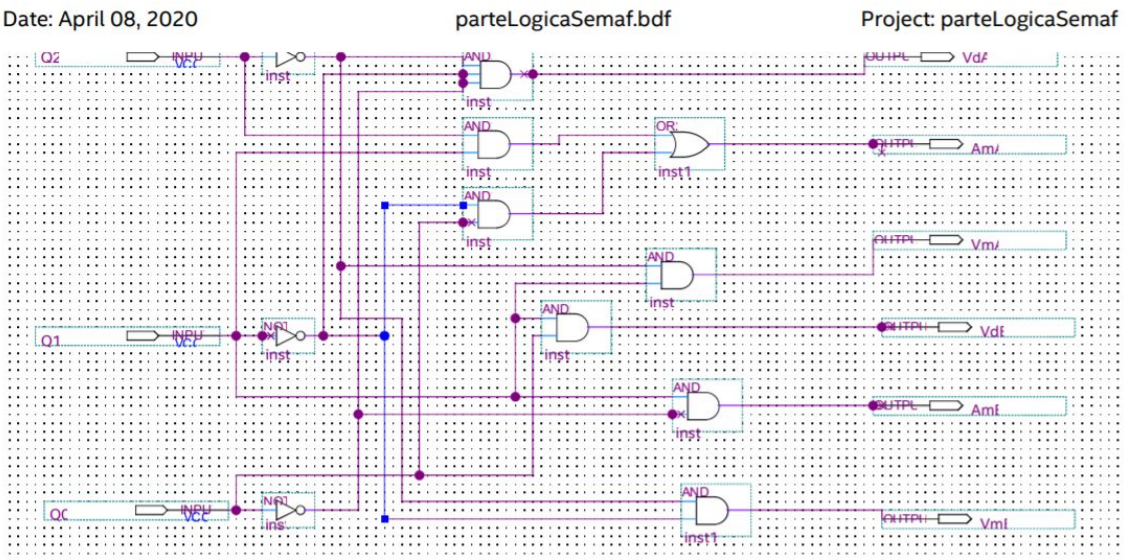
A seguir, temos a tabela de designação de estados e valores de cada lâmpada:

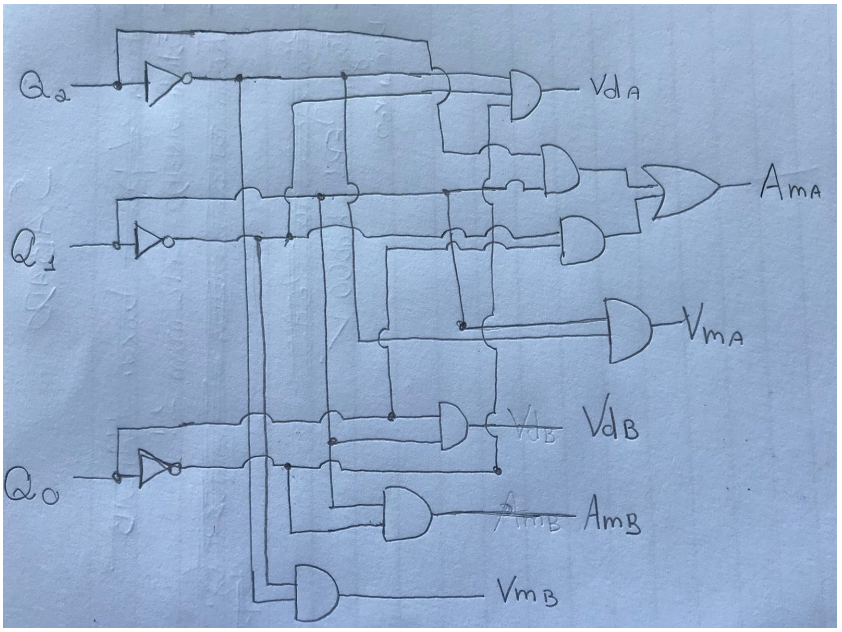
estado	Q_2	Q_1	Q_0	Vd_A	Am_A	Vm_A	Vd_B	Am_B	Vm_B
Vd_A/Vm_B	0	0	0	1	0	0	0	0	1
Am_A/Vm_B	0	0	1	0	1	0	0	0	1
Vm_A/Vd_B	0	1	1	0	0	1	1	0	0
Vm_A/Am_B	0	1	0	0	0	1	0	1	0
apagado	1	0	0	0	0	0	0	0	0
Am_A/Am_B	1	1	0	0	1	0	0	1	0

II. Projeto de Lógica Combinatória (para saídas Via A e Via B)

A partir da tabela acima, percebemos que podemos fazer um Karnaugh de entradas Q2, Q1 e Q0 para cada saída de cores dos semáforos para encontrar as lógicas combinatórias necessárias para esse bloco, vale ressaltar que nesse Karnaugh as entradas ‘101’ e ‘111’ serão saídas X, pois não são estados definidos em nosso projeto (isso ajudará a sintetizar nosso bloco de lógica combinatória).

Por fim, chegamos ao seguinte bloco lógico:

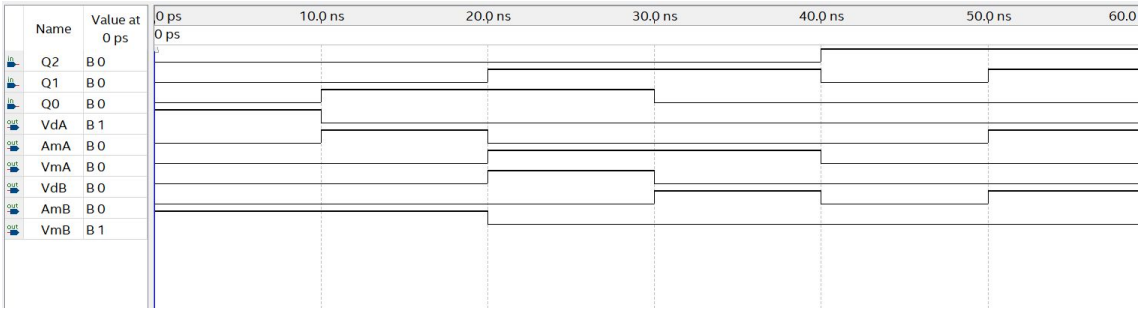




(É o mesmo desenho porém com os ‘inputs’ e ‘outputs’ mais visíveis)

III. Elaboração de Carta de Tempos para bloco de combinatória (para saídas Via A e Via B)

Criamos um Waveform para o nosso projeto no Quartus, setamos um end time de 60ns, utilizando cada 10ns para teste uma entrada do nosso bloco combinatório. Segue abaixo o resultado e como podemos perceber, nossa carta de tempo saiu como o esperado (equivalente a tabela de designação de estados e valores de cada lâmpada), isso demonstra que o bloco criado está correto.



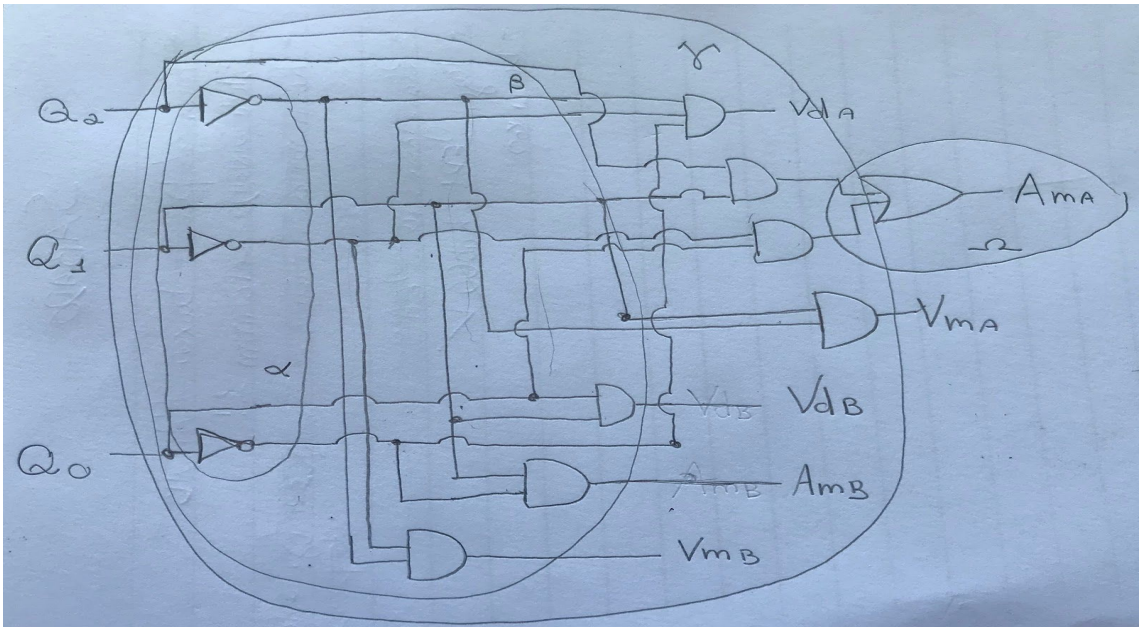
IV. Estratégia de Montagem, testes e depuração para bloco de combinatória (para saídas Via A e Via B)

Para realizar a montagem do circuito esperado, descrito anteriormente neste documento, serão necessários alguns passos:

- Selecionar CI's que executam operações lógicas pertinentes as partes do circuito digital;
- Realizar montagem de cada parte do circuito digital;
- Conectar partes montadas;
- Executar testes com placa auxiliar alimentada por 5V; Inspecionar tensões de saídas com o auxílio de um multímetro, aplicando diferentes entradas/saídas no circuito (que nesse caso, são Q2, Q1 e Q0 nas chaves C1, C2, C3 e as saídas VdA, AmA, VmA, VdB, AmB e VmB nos LED's);

O primeiro teste planejado para o nosso circuito é o teste de cada CI. Através de cada datasheet, olhamos as entradas e saídas das portas lógicas e verificamos se tudo ocorrerá como esperado, caso haja alguma irregularidade, faz-se necessário trocar o circuito integrado, pois provavelmente ele está estragado.

Em seguida, dividimos o nosso circuito em quatro sub-partes e, gradativamente, implementamos algumas portas lógicas e dispositivos, verificando se as saídas até aquele ponto estão conforme o esperado. Respectivamente, implementamos α , β, γ e Ω.



α: Q_2	$\overline{Q_2}$	Q_1	$\overline{Q_1}$	Q_0	$\overline{Q_0}$
0	1	0	1	0	1
1	0	1	0	1	0

β: Q_1	Q_0	VdB	Q_1 $\overline{Q_0}$	AmB	$\overline{Q_2}$ $\overline{Q_1}$	VmB
0	0	0	0 0	0	0 0	0
0	1	0	0 1	0	0 1	0
1	0	0	1 0	0	1 0	0
1	1	1	1 1	1	1 1	1

δ: $\bar{Q}_2\bar{Q}_1\bar{Q}_0$	VdA	Q_2Q_1	K'	$\bar{Q}_1Q_0$	K"	$\bar{Q}_2Q_1$	VmA
0 0 0	0	0 0	0	0 0	0	0 0	0
0 0 1	0	0 1	0	0 1	0	0 1	0
0 1 0	0	1 0	0	1 0	0	1 0	0
0 1 1	0	1 1	1	1 1	1	1 1	1
1 0 0	0						
1 0 1	0						
1 1 0	0						
1 1 1	1						

Ω: K'K"	AmA
0 0	0
0 1	1
1 0	1
1 1	1

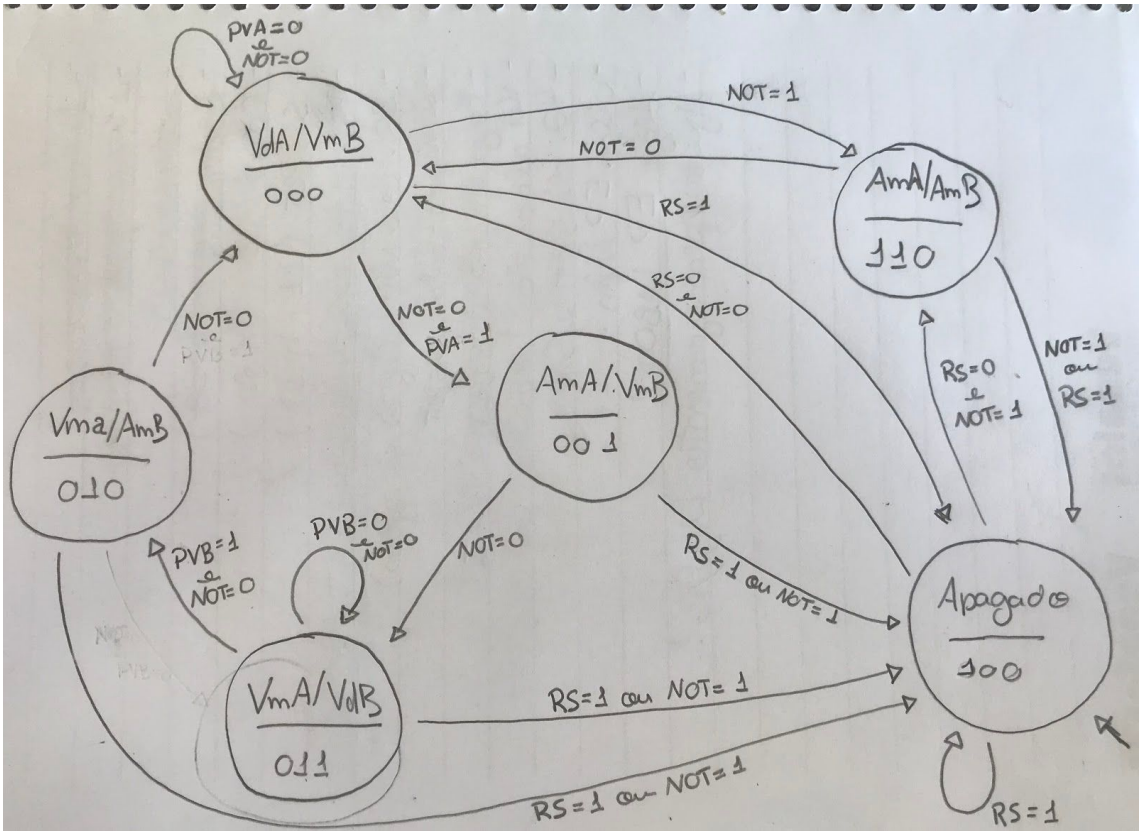
Ao terminarmos a montagem omega sem problemas, garantimos que nosso circuito do bloco combinatório não tem problemas nos fios, nos CI's e nem na nossa montagem até esse ponto. Portanto, caso haja algum problema, será nas etapas seguintes.

V. Diagrama de transição de estados completo da Unidade de Controle

Para a criação do diagrama de transição de estados completo da UC, faz-se necessário lembrar dos sensores de 'input' + RESET.

A não presença de veículos em uma via acarreta a suspensão do tempo de luz verde nesta via. Ao se detectar um veículo, a sequência de acionamentos das luzes volta ao normal. De madrugada, as luzes devem ser piscantes de forma intermitente para ambas as vias, e nesse caso a informação de presença de veículos não é utilizada. Com o sinal de RESET, a unidade de controle volta para o estado inicial.

Por fim, chamaremos esses 'input's de PvA, PvB, Noturno, Reset:



Para montarmos o conceito pedido, em primeiro momento, achamos que o RESET deveria ir ao estado ‘Apagado’, pois faria mais sentido ele entrar a partir desse estado e sair dele (quando reset = ‘0’) para um dos dois estados que iniciam os comportamentos diurno e noturno.

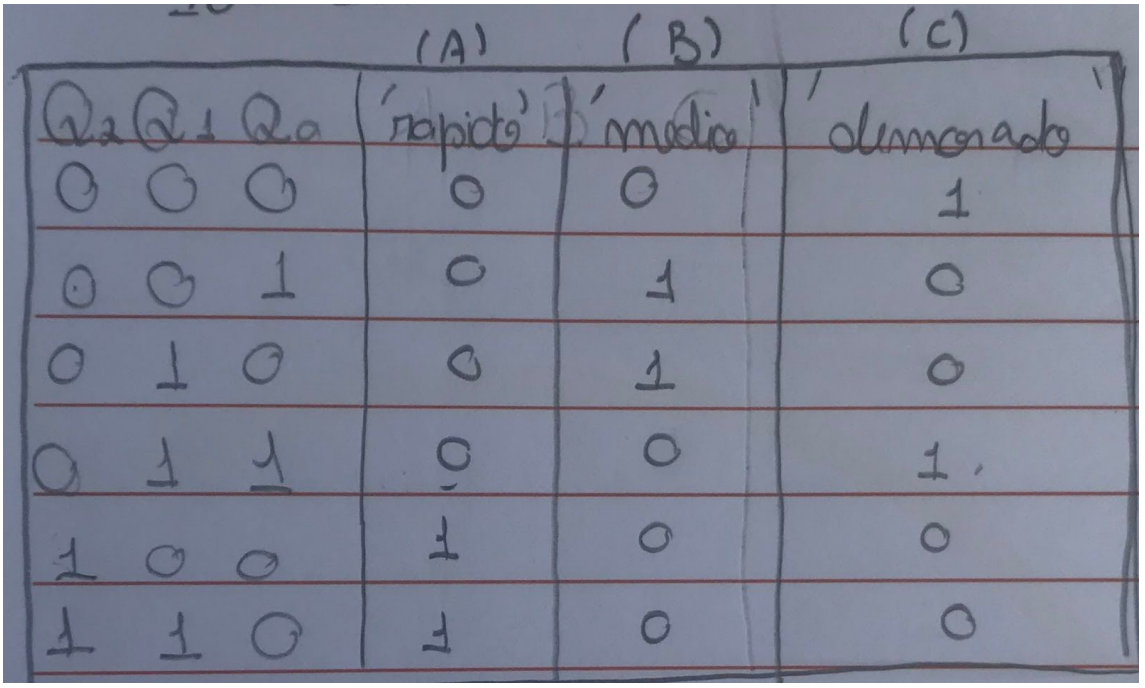
Em seguida, completamos com PvA/PvB para o sensor de veículos em cada via, note que ele só se faz necessário no comportamento diurno, pois a presença ou não de veículos no noturno é irrelevante para o comportamento do semáforo. Perceba também que quando não a presença de carros (ou seja, SVe = 0), nosso circuito tende a ficar no Verde mais próximo sem mudar de estado até que seja acionado a presença de carro. Perceba que há uma ordem de relevância dos ‘inputs’ e isso será repassado no código em VHDL.

Por fim, vale ressaltar o uso de ‘e’ e ‘ou’ trazendo, às vezes, a necessidade de dois ‘inputs’ para a transição de estados e em outras a necessidade de somente um dos ‘inputs’.

VI. Projeto de Lógica Combinatória (para saídas Triggers)

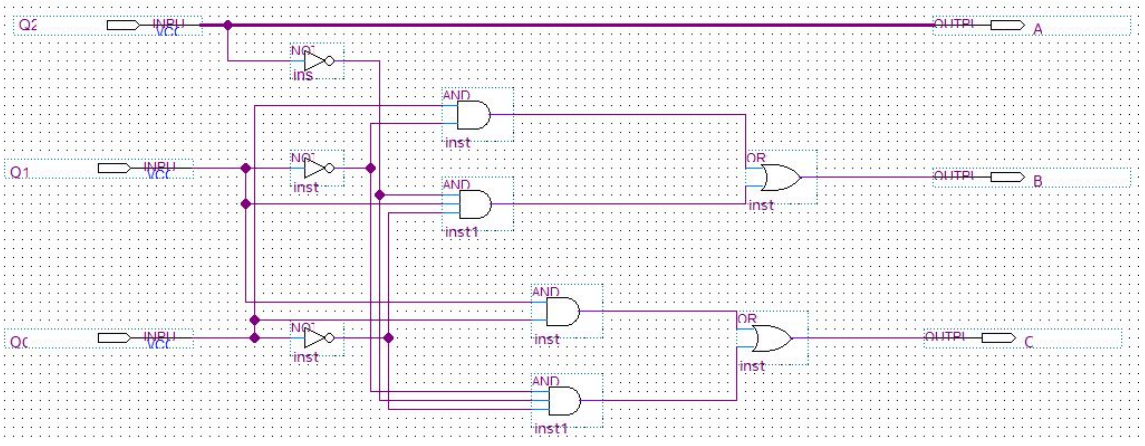
Para essa parte da lógica combinatória, teremos as mesmas entradas que representam cada estado da máquina de estados (‘Q2Q1Q0’). Contudo, as saídas devem ser três triggers, ou seja, três ‘outputs’ que informam ao circuito temporizador, quanto tempo ele deve contar até o próximo estado.

De forma a simplificar o nosso circuito chamarem as saídas de ‘rapido’, ‘medio’, ‘demorado’ que serão respectivamente para timers de 1 segundo, 1 minuto e 5 minutos. Perceba que como não temos acesso a ‘inputs’ de sensores, por exemplo, teremos que designar uma tabela de comportamento que só relacione um tempo a cada estado. O que nos traz a necessidade de adicionar do seguinte comportamento :



	(A)	(B)	(C)
$Q_2 Q_1 Q_0$	'rapido'	'medio'	'demorado'
0 0 0	0	0	1
0 0 1	0	1	0
0 1 0	0	1	0
0 1 1	0	0	1
1 0 0	1	0	0
1 1 0	1	0	0

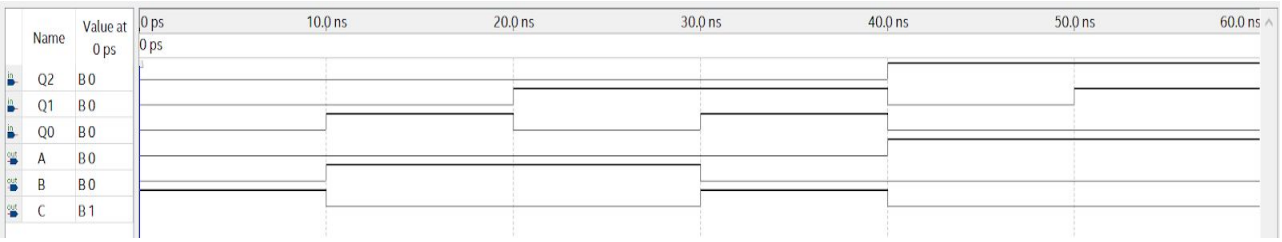
Assim, através da tabela de comportamento acima, chegamos na seguinte parte lógica para a saída dos triggers:



VII. Elaboração de Carta de Tempos para bloco de combinatória (para saídas Triggers)

Criamos um Waveform para o nosso projeto no Quartus, setamos um end time de 60ns, utilizando cada 10ns para teste uma entrada do nosso bloco combinatório. Segue abaixo o resultado e como podemos perceber, nossa carta

de tempo saiu como o esperado (equivalente a tabela de designação de estados e valores de cada trigger), isso demonstra que o bloco criado está correto.



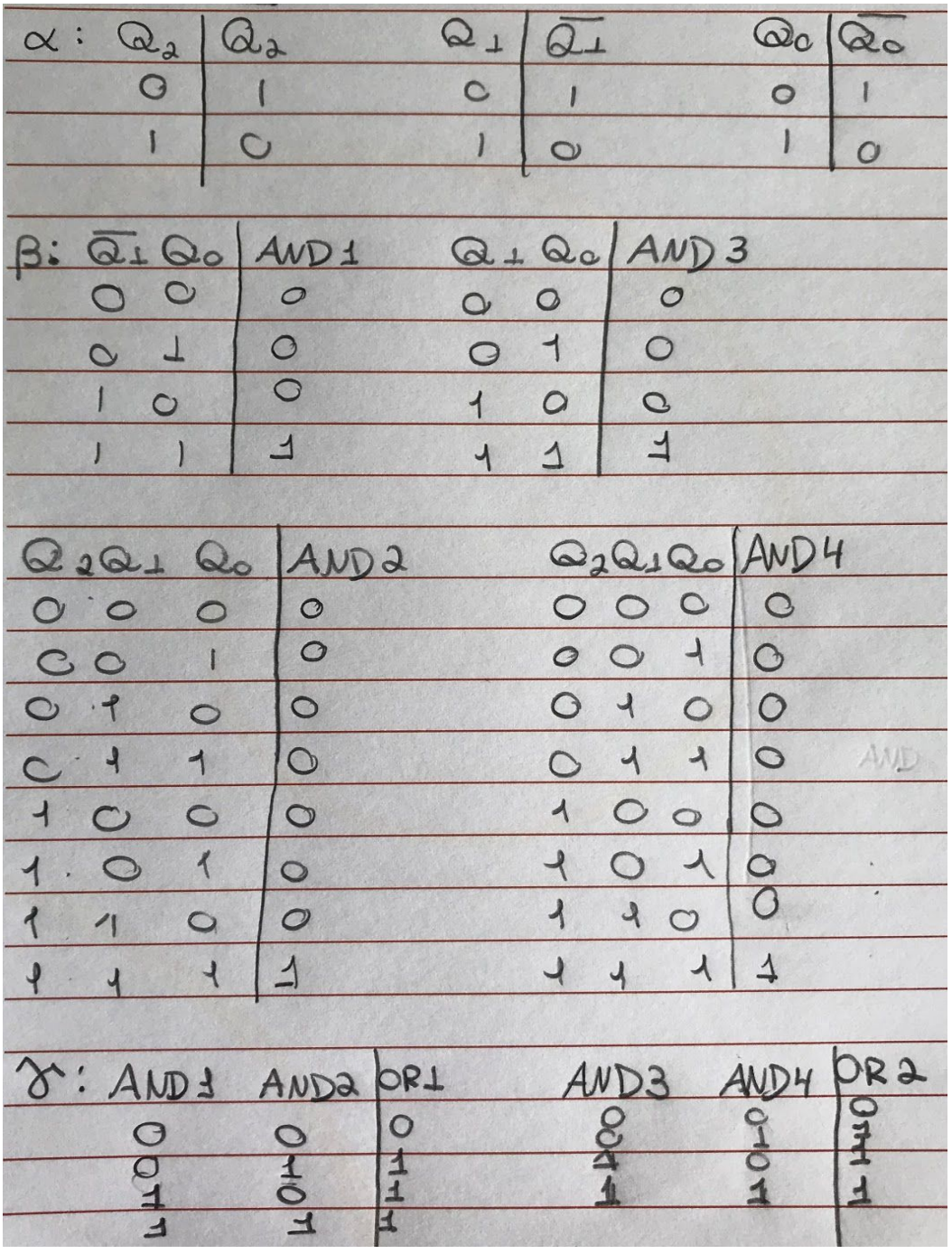
VIII. Estratégia de Montagem, testes e depuração para bloco de combinatória (para saídas Triggers)

Para realizar a montagem do circuito esperado, descrito anteriormente neste documento, serão necessários alguns passos:

- Selecionar CI's que executam operações lógicas pertinentes as partes do circuito digital;
- Realizar montagem de cada parte do circuito digital;
- Conectar partes montadas;
- Executar testes com placa auxiliar alimentada por 5V; Inspecionar tensões de saídas com o auxílio de um multímetro, aplicando diferentes entradas/saídas no circuito (que nesse caso, são Q2, Q1 e Q0 nas chaves C1, C2, C3 e as saídas A, B e C nas entradas do circuito temporizador na placa FPGA);

O primeiro teste planejado para o nosso circuito é o teste de cada CI. Através de cada datasheet, olhamos as entradas e saídas das portas lógicas e verificamos se tudo ocorrerá como esperado, caso haja alguma irregularidade, faz-se necessário trocar o circuito integrado, pois provavelmente ele está estragado.

Em seguida, dividimos o nosso circuito em quatro sub-partes e, gradativamente, implementamos algumas portas lógicas e dispositivos, verificando se as saídas até aquele ponto estão conforme o esperado. Respectivamente, implementamos α , β e γ . Sendo alpha as saídas dos NOT's, beta as saídas dos AND's e, por fim, gama as saídas dos OR's.



α:

Q_2	$\overline{Q_2}$
0	1
1	0

Q_1	$\overline{Q_1}$
0	1
1	0

Q_0	$\overline{Q_0}$
0	1
1	0

β:

$\overline{Q_1}$	Q_0	AND1
0	0	0
0	1	0
1	0	0
1	1	1

Q_1	Q_0	AND3
0	0	0
0	1	0
1	0	0
1	1	1

Q_2	Q_1	Q_0	AND2
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Q_2	Q_1	Q_0	AND4
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

γ:

AND1	AND2	OR1
0	0	0
0	1	1
1	0	1
1	1	1

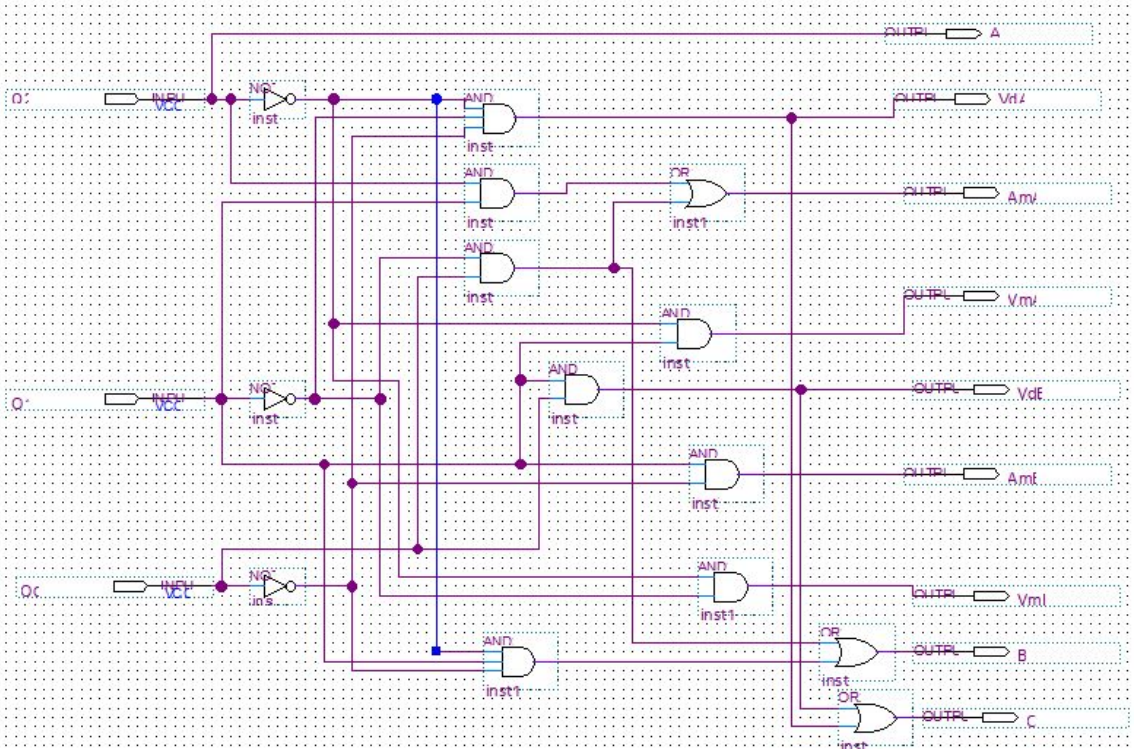
AND3	AND4	OR2
0	0	0
0	1	1
1	0	1
1	1	1

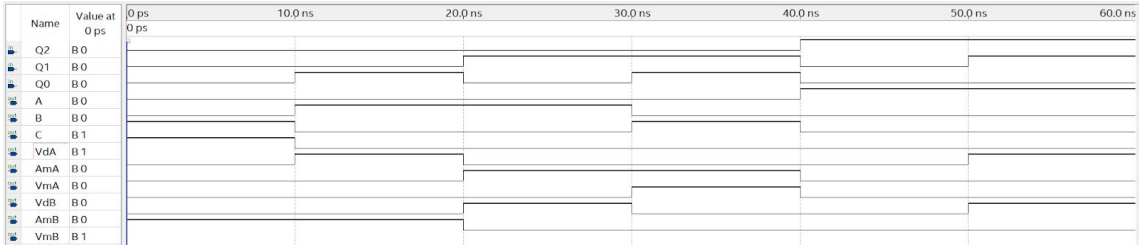
Ao terminarmos a montagem gama sem problemas, garantimos que nosso circuito do bloco combinatório não tem problemas nos fios, nos CI's e nem na nossa montagem até esse ponto. Portanto, caso haja algum problema, será nas etapas seguintes.

IX. Junção de partes lógicas combinatórias (Sub bloco H)

Agora que já compreendemos e analisamos separadamente o funcionamento da parte combinatória e seus outputs que irão para os LED's e os que irão para o circuito temporizador, podemos juntar em um único bloco combinatório com

todas as saídas. Perceba que, temos o AND1 = K’’; AND3 = VdB; AND4 = VdA e basta adicionarmos ao programa inicial das Vias A e vias B uma porta AND e duas portas OR’s que teremos nosso projeto completo. Segue abaixo layout do sub bloco H e carta de tempos:





Podemos ver que nossa junção ocorreu corretamente, pois a carta de tempos é análoga às duas outras demonstradas anteriormente.

X. Projeto em VHDL para bloco “Lógica sequencial” (com Waveform)

Assim, como visto e desenvolvido acima, temos que novos ‘inputs’ foram adicionados ao nosso diagrama de transição de estados e, portanto, nosso código em VHDL será:

```
library IEEE;
use IEEE.std_logic_1164.all;

entity parteSeq is
   port ( clock, reset :              in  std_logic;
          iniciar, noturno, pva, pvb: in  std_logic;
          db_estado:                  out std_logic_vector(2 downto 0)
        );
end parteSeq;

architecture comportamentalseq of parteSeq is
   type Tipo_estado is (apagado, vdA_vmB, amA_vmB, vmA_amB, vmA_vdB, amA_amB);
   signal Eatual, Eprox: Tipo_estado;

begin

   process (reset, clock)
   begin
      if reset = '1' then
         Eatual <= apagado;

      elsif clock'event and clock = '1' then
         Eatual <= Eprox;
      end if;
   end process;


   process (noturno, pva, pvb)
   begin
      case Eatual is
         when apagado =>          if noturno = '1'
                                  then Eprox <= amA_amB;
                                  else Eprox <= vdA_vmB;
                                  end if;

         when vdA_vmB =>          if noturno = '0'then
                                     if pva = '0' then
                                        Eprox <= vdA_vmB;
                                     else Eprox <= amA_vmB
                                     end if;
                                  else Eprox <= amA_amB;
                                  end if;

         when amA_vmB =>          if noturno = '1'
                                  then Eprox <= apagado;
                                  else Eprox <= vmA_vdB;
                                  end if;

         when vmA_amB =>          if noturno = '1' then
                                    Eprox <= apagado;
                                  else Eprox <= vdA_vmB;
                                  end if;

         when vmA_vdB =>          if noturno = '0'then
                                     if pvb = '0' then
                                        Eprox <= vmA_vdB;
                                    else Eprox <= vmA_amB;
                                    end if;
                                  else Eprox <= apagado;
                                  end if;

          when amA_amB =>         if noturno = '1'
                                  then Eprox <= apagado;
                                  else Eprox <= vdA_vmB;
                                  end if;

         when others =>           Eprox <= apagado;
      end case;
   end process;


   with Eatual select
      db_estado <= "000" when vdA_vmB,
                   "001" when amA_vmB,
                   "010" when vmA_amB,
                   "011" when vmA_vdB,
                   "100" when apagado,
                   "110" when amA_amB,
                   "111" when others;

end comportamentalseq;
```

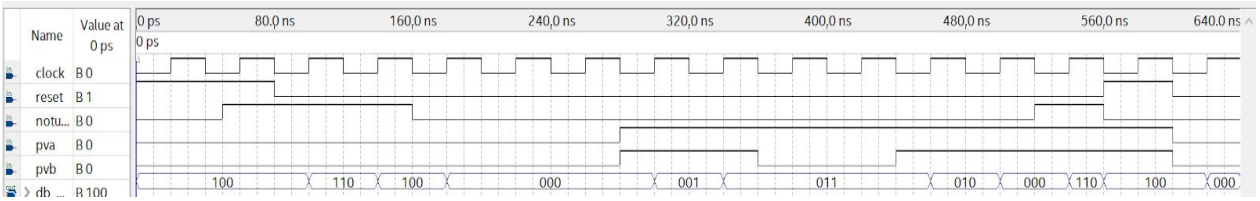
A partir da nossa análise completa do comportamento que desejamos desse bloco do nosso projeto, podemos criar um plano de testes e a carta de tempo para esse circuito.

Primeiramente, ajustamos o ‘clock’ (com um ciclo de 40ns), setamos um end time para 0.64 micro segundos (tempo suficiente para o nosso teste) e colocamos o ‘reset’ em ‘1’ para garantir que nosso circuito não tem resquícios que possa atrapalhar nosso teste.

Em seguida, vamos introduzir o estado inicial aplicando a condição ‘1’ e ‘0’ de Reset e verificando se nosso programa se mantém no estado ‘apagado’ e vai para o estado ‘AmA/AmB’, respectivamente. Retornamos ao estado apagado e forçamos um ‘nortuno’ = 0 para ver se ele passará ao estado “VdA/VmB”.

Logo, deixaremos o circuito percorrer todos os 4 estados do seu funcionamento de forma diurna, somente parando por 2 clocks em “VdA/VmB” e “VmA/VdB” para testar os ‘inputs’ ‘PvA’ e ‘PvB’.

Após isso, iremos acionar o ‘noturno’ para a passagem ao estado “AmA/AmB”’. E, para terminar a nossa simulação, saíremos do ‘AmA/AmB’ desligando o ‘noturno’ e acionando o ‘reset’ para chegarmos ao ‘apagado’ de um formato diferente.



XI. Projeto em VHDL para bloco “Circuito temporizador” (com Waveform)

Para a elaboração deste circuito temos que lembrar que esse deve gerar clocks em segundos e minutos. E a ideia é que esse circuito envie os clocks prontos para o circuito sequencial a partir de entradas encaminhadas do circuito lógico combinatório.

Os clocks devem ser de 1 segundo para as transições de estados entre luzes amarela acesa e apagada, 5 segundos entre amarela e vermelha, 1 min entre verde e amarela e vermelha e verde. Ou seja, teremos três valores de tempo diferente e estes se intercalam a medida das transições de estados do circuito sequencial.

Nossa placa FPGA (DE0-CV) fornece um clock-base de 50MHz. Para se obter clocks da ordem de Hz, por exemplo, precisamos dividir essa frequência.

Assim, podemos dividir nosso circuito em três funcionamentos fundamentais A, B e C, temporizando 1 segundo, 5 segundos e 1 minuto, respectivamente. Para o funcionamento A, temos duas situações:

Se Q2Q1Q0 = ‘100’;

Se Q2Q1Q0 = ‘110’ + NOTURNO = ‘1’;

Para o funcionamento B, temos duas situações:

Se Q2Q1Q0 = ‘001’;

Se Q2Q1Q0 = ‘010’;

Para o funcionamento C, temos duas situações:

Se Q2Q1Q0 = ‘000’ + NOTURNO = ‘0’;

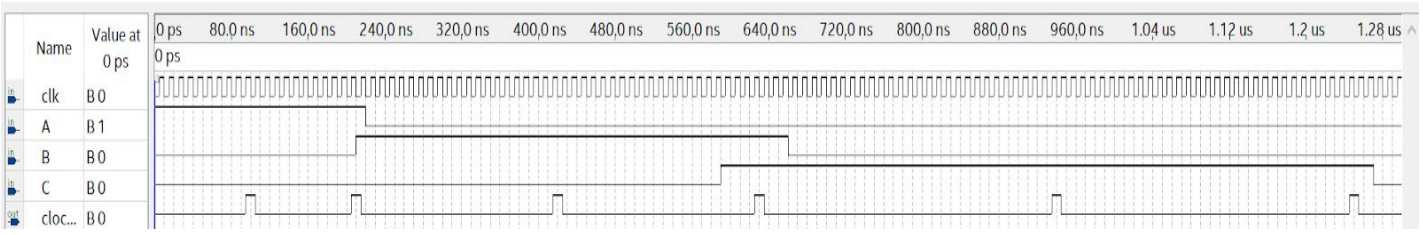
Se Q2Q1Q0 = ‘011’;

Perceba que para os demais casos, o circuito temporizador não deve ser acionado e, portanto, a entrada e saída serão dadas em um clock do nossa placa FPGA (que é extremamente rápido, pois tem 50MHz).

Abaixo segue o código em VHDL para o nosso circuito temporizador:

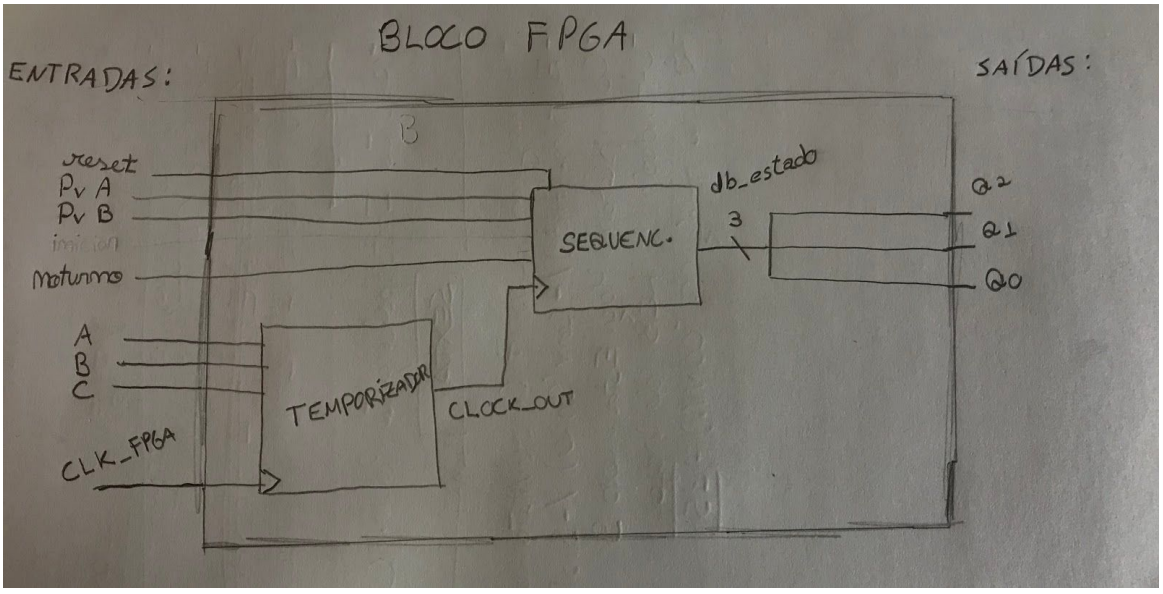
```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.numeric_std.ALL;
4
5  entity circuitoTemporizador is
6  port ( clk: in std_logic;
7         A, B, C: in std_logic;
8         clock_out: out std_logic);
9  end circuitoTemporizador;
10
11 architecture bhv of circuitoTemporizador is
12
13 signal count: integer := 1;
14 signal tmp : std_logic := '0';
15
16 begin
17
18 process(clk)
19 begin
20
21 if(clk'event and clk='1') then
22    if count = 0 then
23       tmp <= NOT tmp;
24       count <= 2;
25    end if;
26 count <=count+1;
27    if (A = '1' and count = 50000000) then
28       tmp <= NOT tmp;
29       count <= 0;
30    elsif (B = '1' and count = 250000000) then
31       tmp <= NOT tmp;
32       count <= 0;
33    elsif (C = '1' and count = 3000000000) then
34       tmp <= NOT tmp;
35       count <= 0;
36    end if;
37 end if;
38 clock_out <= tmp;
39 end process;
40
41
42 end bhv;
43
```

Para o nosso teste, utilizamos o mesmo código que tínhamos feito acima, porém, alteramos os valores para não termos um experimento muito extenso (que demoraria muitos minutos de cartas de tempo). Assim, ajustamos o ‘clock’ (com um ciclo de 10ns), setamos um end time para 1.3 microssegundos e mudamos os valores de ‘count’ para A, B e C, colocando 10, 20 e 30 contagens respectivamente. Ou seja, em nosso teste, um clock A, por exemplo, espera 10 contagens e faz um clock original (10ns * 10 contagens = clock em 100ns) . Perceba que utilizamos a mesma lógica do código, porém, trouxemos ordem de grandeza mais palpáveis para o nosso teste.



XII. Junção bloco “Lógica sequencial” e “Circuito temporizador” em único arquivo

Nesta seção do nosso projeto, faremos a junção de ambos blocos que irão a placa FPGA. Para isso, vamos utilizar o ‘portmap’ de forma similar ao que foi construído no experimento anterior e criaremos uma tabela de pinagem de forma coerente e possível de se interligar ao bloco de lógica combinatória que será montado no painel de montagem.



Através da esquematização acima de como queremos interligar esses blocos de VHDL, a junção de códigos VHDL correspondente será:

```
library IEEE;
use IEEE.std_logic_1164.all;

entity parteSeq is
   port ( clock, reset :             in  std_logic;
          noturno, pva, pvb: in  std_logic;
          db_estado:                 out std_logic_vector(2 downto 0)
        );
end parteSeq;

architecture comportamentalseq of parteSeq is
   type Tipo_estado is (apagado, vdA_vmB, amA_vmB, vmA_amB, vmA_vdB, amA_amB);
   signal Eatual, Eprox: Tipo_estado;

begin

   process (reset, clock)
   begin
      if reset = '1' then
         Eatual <= apagado;

      elsif clock'event and clock = '1' then
         Eatual <= Eprox;
      end if;
   end process;


   process (pva, pvb)
   begin
      case Eatual is
         when apagado =>          if noturno = '1'
                                  then Eprox <= amA_amB;
                                  else Eprox <= vdA_vmB;
                                  end if;

         when vdA_vmB =>          if noturno = '0'then
                                     if pva = '0' then
                                        Eprox <= vdA_vmB;
                                     else Eprox <= amA_vmB;
                                     end if;
                                  else Eprox <= amA_amB;
                                  end if;

         when amA_vmB =>          if noturno = '1'
                                  then Eprox <= apagado;
                                  else Eprox <= vmA_vdB;
                                  end if;

         when vmA_amB =>          if noturno = '1' then
                                    Eprox <= apagado;
                                  else Eprox <= vdA_vmB;
                                  end if;

         when vmA_vdB =>          if noturno = '0'then
                                     if pvb = '0' then
                                        Eprox <= vmA_vdB;
                                     else Eprox <= vmA_amB;
                                     end if;

         when amA_amB =>          if noturno = '1'
                                  then Eprox <= apaga
                                  else Eprox <= vdA_v
                                  end if;

         when others =>           Eprox <= apagado;
      end case;
   end process;


   with Eatual select
      db_estado <= "000" when vdA_vmB,
                   "001" when amA_vmB,
                   "010" when vmA_amB,
                   "011" when vmA_vdB,
                   "100" when apagado,
                   "110" when amA_amB,
                   "111" when others;

end comportamentalseq;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;

entity circuitoTemporizador is
port ( clk: in std_logic;
       A, B, C: in std_logic;
       clock_out: out std_logic);
```

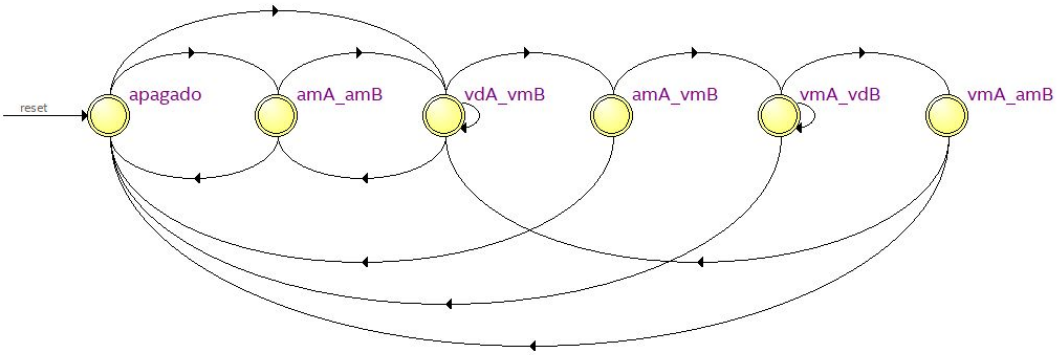
```
91  end circuitoTemporizador;
92
93  architecture bhv of circuitoTemporizador is
94
95  signal count: integer := 1;
96  signal tmp : std_logic := '0';
97
98  begin
99
00  process(clk)
01  begin
02
03  if(clk'event and clk='1') then
04     if count = 0 then
05        tmp <= NOT tmp;
06        count <= 2;
07     end if;
08  count <=count+1;
09     if (A = '1' and count = 50000000) then
10        tmp <= NOT tmp;
11        count <= 0;
12     elsif (B = '1' and count = 250000000) then
13        tmp <= NOT tmp;
14        count <= 0;
15     elsif (C = '1' and count = 3000000000) then
16        tmp <= NOT tmp;
17        count <= 0;
18     end if;
19  end if;
```

```
120 clock_out <= tmp;
121 end process;
122
123
124 end bhv;
125
126
127 library IEEE;
128 use IEEE.STD_LOGIC_1164.ALL;
129 use IEEE.numeric_std.ALL;
130
131 entity blocoFPGA is
132    port ( clk_fpga, reset_fpga :  in  std_logic;
133           noturno_fpga, pva_fpga, pvb_fpga: in  std_logic;
134           A, B, C:     in  std_logic;
135           db_estado_fpga:  out std_logic_vector(2 downto 0)
136         );
137 end blocoFPGA;
138
139 architecture comp of blocoFPGA is
140
141 component parteSeq is
142    port ( clock, reset :              in  std_logic;
143           noturno, pva, pvb: in  std_logic;
144           db_estado:                 out std_logic_vector(2 downto 0)
145         );
146 end component;
```

```
148 component circuitoTemporizador is
149 port ( clk: in std_logic;
150       A, B, C: in std_logic;
151       clock_out: out std_logic);
152 end component;
153
154 signal S_clock: std_logic;
155
156 begin
157
158 sequencial1: parteSeq port map ( s_clock, reset_fpga, noturno_fpga, pva_fpga,
159 pvb_fpga, db_estado_fpga);
160
161 temporizador1: circuitoTemporizador port map (clk_fpga, A, B, C, s_clock);
162
163 end comp;
```

Após compilarmos, conseguimos ver que o diagrama de estados está correto conforme planejamos.

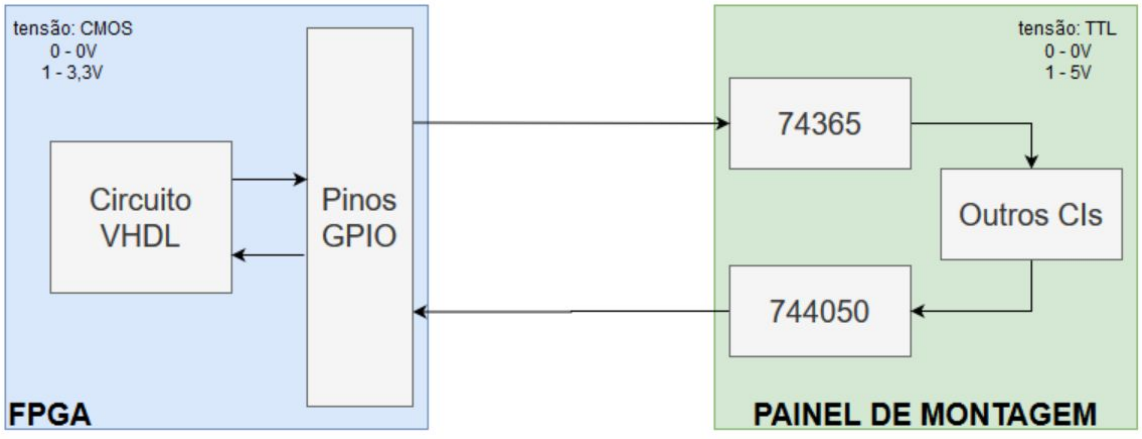
	Name	amA_amB	vmA_vdB	vmA_amB	amA_vmB	vdA_vmB	apagado
1	apagado	0	0	0	0	0	0
2	vdA_vmB	0	0	0	0	1	1
3	amA_vmB	0	0	0	1	0	1
4	vmA_amB	0	0	1	0	0	1
5	vmA_vdB	0	1	0	0	0	1
6	amA_amB	1	0	0	0	0	1



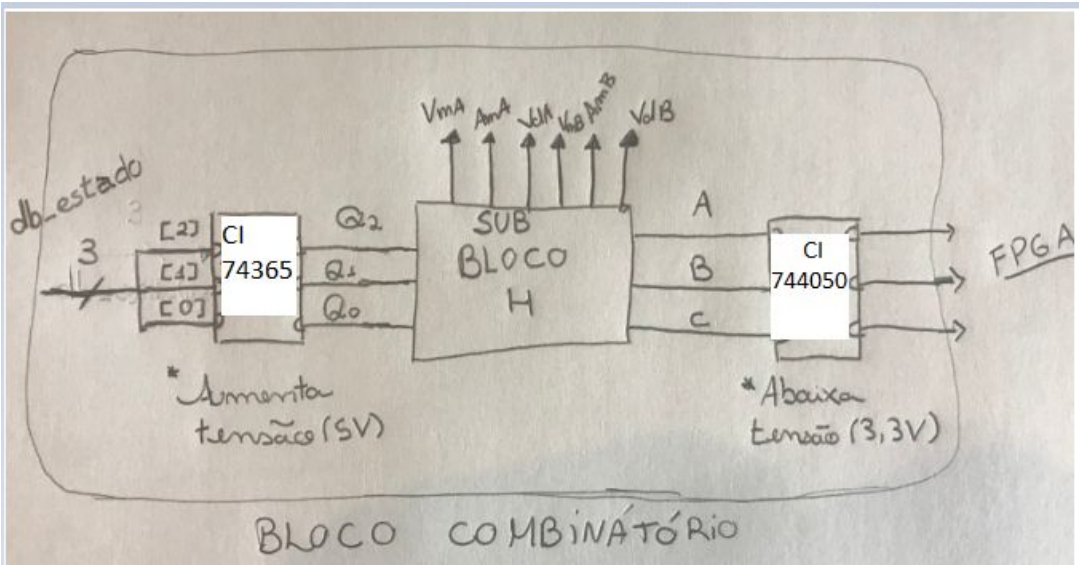
	Source State	Destination State	Condition
1	amA_amB	vdA_vmB	(!noturno)
2	amA_amB	apagado	(noturno)
3	amA_vmB	vmA_vdB	(!noturno)
4	amA_vmB	apagado	(noturno)
5	apagado	vdA_vmB	(!noturno)
6	apagado	amA_amB	(noturno)
7	vdA_vmB	vdA_vmB	(!noturno). (!pva)
8	vdA_vmB	amA_amB	(noturno)
9	vdA_vmB	amA_vmB	(pva). (!noturno)
10	vmA_amB	vdA_vmB	(!noturno)
11	vmA_amB	apagado	(noturno)
12	vmA_vdB	vmA_vdB	(!pvb). (!noturno)
13	vmA_vdB	vmA_amB	(pvb). (!noturno)
14	vmA_vdB	apagado	(noturno)

XIII. Diagrama de blocos detalhado para bloco lógico

Vale ressaltar que há diferenças de tensão de trabalho da FPGA (3,3V) e do painel de montagens (5V), portanto, isso deve ser levado em conta na hora de interligarmos entre si. Para resolvermos esta problemática, devemos adicionar dois buffers de tensão.



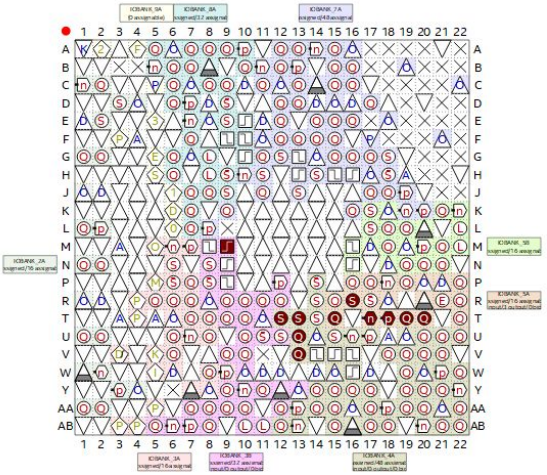
Um que levará a transformação do “db_estado_fpga” de 3,3V a 5V (o CI 74365). E um outro que levará a transformação dos sinais A, B, C de 5V a 3,3V (o CI 744050). Assim, o seguinte bloco:



XIV. Tabela de Pinagem do Projeto + designação de pinos dentro do Quartus

Sinal	Ligação na placa FPGA	Pino na FPGA
clk_fpga	CLOCK_50	PIN_M9
reset_fpga	SW0	PIN_U13
noturno_fpga	SW1	PIN_V13
pva_fpga	SW2	PIN_T13
pvb_fpga	SW3	PIN_T12
A	GPIO_0_D30	PIN_R16
B	GPIO_0_D32	PIN_T19
C	GPIO_0_D34	PIN_T17
db_estado_fpga[0]	GPIO_0_D31	PIN_T20
db_estado_fpga[1]	GPIO_0_D33	PIN_T18
db_estado_fpga[2]	GPIO_0_D35	PIN_T15

Node Name	Direction	Location	I/O Bank	/REF Group	tter Locatic	'O Standar	Reserved	rent Stren	Slew Rate	f
A	Input	PIN_R16	5A	B5A_N0	PIN_R16	2.5 V		12mA...ult)		
B	Input	PIN_T19	5A	B5A_N0	PIN_T19	2.5 V		12mA...ult)		
C	Input	PIN_T17	5A	B5A_N0	PIN_T17	2.5 V		12mA...ult)		
clk_fpga	Input	PIN_M9	3B	B3B_N0	PIN_M9	2.5 V		12mA...ult)		
db_est...pga[2]	Output	PIN_T15	5A	B5A_N0	PIN_T15	2.5 V		12mA...ult)	1 (default)	
db_est...pga[1]	Output	PIN_T18	5A	B5A_N0	PIN_T18	2.5 V		12mA...ult)	1 (default)	
db_est...pga[0]	Output	PIN_T20	5A	B5A_N0	PIN_T20	2.5 V		12mA...ult)	1 (default)	
noturno_fpga	Input	PIN_V13	4A	B4A_N0	PIN_V13	2.5 V		12mA...ult)		
pva_fpga	Input	PIN_T13	4A	B4A_N0	PIN_T13	2.5 V		12mA...ult)		
pvb_fpga	Input	PIN_T12	4A	B4A_N0	PIN_T12	2.5 V		12mA...ult)		
reset_fpga	Input	PIN_U13	4A	B4A_N0	PIN_U13	2.5 V		12mA...ult)		



4. **Relatório**

Apêndices

Referências

1. Apostilas e documentos de apoio do site ~/labdig do PCS.
2. Apostilas disponíveis na plataforma e-Disciplinas.

[1] https://balbertini.github.io/vhdl_fsm-pt_BR.html