

<b>Disciplina:</b> PCS 3335 – Laboratório Digital A	
<b>Prof.:</b> Anarosa	<b>Data:</b> 15/07/2020
<b>Turma:</b> 01	<b>Bancada:</b> A2
<b>Membros:</b>	
Andre Devay Torres Gomes (10770089)	
FIZ O EXPERIMENTO SOZINHO	



## Planejamento P2

## 1. Introdução

Neste experimento iremos implementar a hierarquização de prioridades em um circuito digital, pois geralmente estes são acionados por eventos externos e cada tipo de evento reconhecido, há uma função mais prioritária ou não. A aplicação mais comum deste tipo de circuito é o acionamento de alarmes sonoros diferentes de acordo com a sua prioridade. Sistemas de controle de reatores nucleares, trens, prédios etc. são exemplos onde os alarmes fazem parte da sua implementação.

## 2. Objetivo

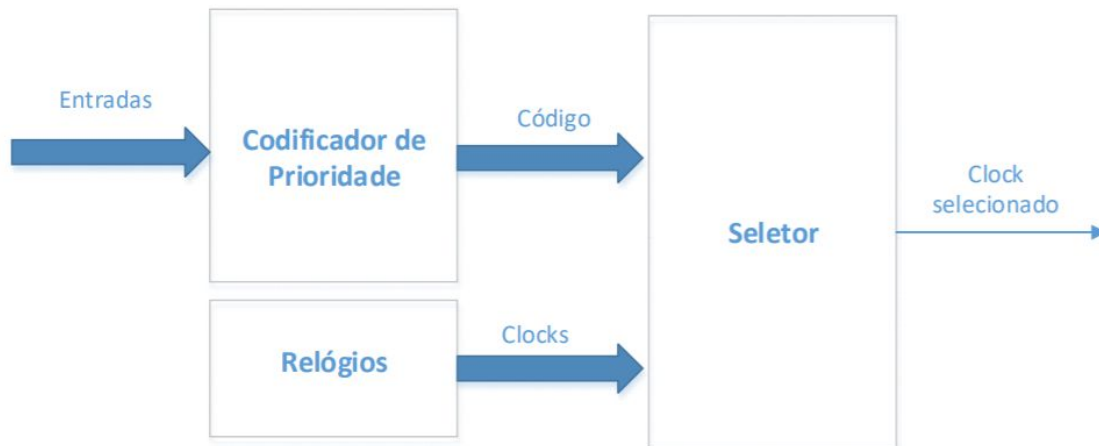
Conceituar e desenvolver um módulo digital muito útil em que se seleciona diferentes frequências a partir de entradas específicas e de níveis mais ou menos prioritários.

## 3. Planejamento

### I. Descrição Funcional projeto completo

A ideia deste experimento é a projeção de um seletor de prioridade para sensores (ou fatores externos), que determine um clock a ser utilizado. Apesar de genérico, este módulo digital pode ser muito útil, pois tem diversos tipos de projetos que podem utilizar desta aplicabilidade para economia de energia, por exemplo, entre outras melhorias de circuito.

Como mencionado no enunciado, queremos selecionar dentre 4 frequências de clock, portanto, iremos inferir 4 tipos de entradas diferentes (uma para cada frequência) e, cada uma delas, terá uma prioridade diferente no escopo digital. Assim, aquela entrada de maior prioridade irá acionar o clock de maior frequência e vice-versa.



O codificador de prioridade será responsável por elencar qual é a entrada prioritária, ou seja, ele receberá os quatro tipos de informações dos sensores (as entradas do módulo). E, a partir disso, ele definirá o “código” (uma saída de 2 bits que corresponde a qual entrada foi elencada para o clock).

O bloco dos relógios estará dentro da FPGA e, portanto, apesar de não ter sido explicitado sua entrada (pois não é externa), ela receberá o clock próprio da FPGA como sua entrada (de frequência de 50 MHz) e montará os 4 clocks de frequências distintas a partir do clock de entrada.

Por fim, o bloco seletor recebe como entrada ambas as saídas dos outros 2 blocos. Ou seja, este tem 6 entradas (os 4 clocks e 2 bits do “código”). Seu comportamento é equivalente ao de um MUX 4:1 e, portanto, ele seleciona um das 4 entradas que será a saída Y naquele instante a partir do código no seletor.

A ideia é montarmos o bloco “codificador de prioridade” e o bloco “seletor” como circuitos de CIs e o bloco “relógios” em código VHDL.

## II. Comportamento dos Relógios

Para este item, devemos desenvolver um código em VHDL, que produza 4 clocks distintos a partir do clock genérico de nossa FPGA. Ou seja, este bloco VHDL terá como entrada o clock de 50MHz e quatro saídas com os clocks de 20Hz, 200Hz, 2000Hz e 20000Hz.

De forma análoga ao código desenvolvido por nós no experimento dos semáforos (para o bloco temporizador), podemos utilizar uma contagem do clock original para que seja invertido o pulso da saída a cada meia contagem. Assim, basta ajustarmos a razão para cada frequência desejada.

Para 20Hz, teremos  $50000000/20 = 2500000$  contagens/clock.

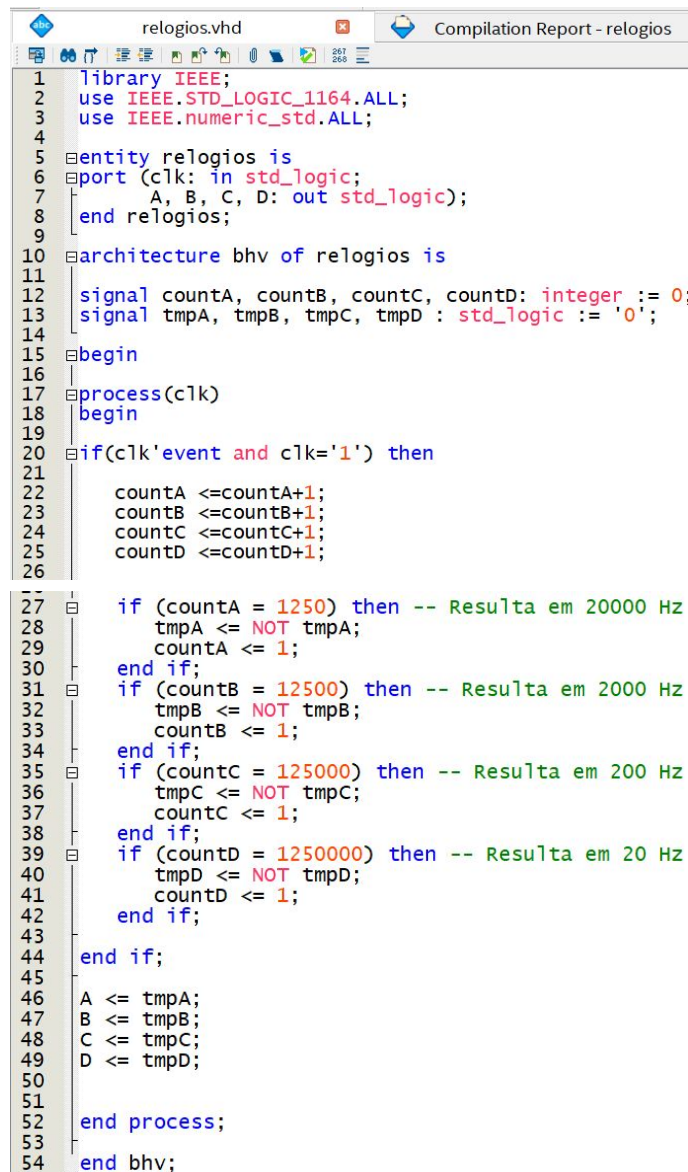
Para 200Hz, teremos  $50000000/200 = 250000$  contagens/clock.

Para 2000Hz, teremos  $50000000/2000 = 25000$  contagens/clock.

Para 20000Hz, teremos  $50000000/20000 = 2500$  contagens/clock.

### III. Projeto em VHDL para bloco “Relógios” (com Waveform)

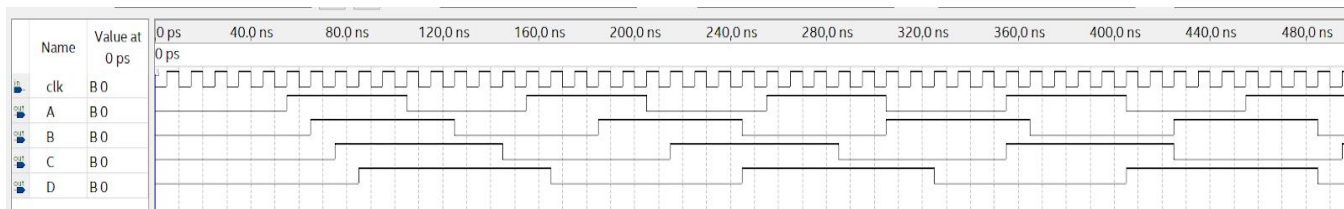
Utilizando a descrição dos comportamentos do item anterior, chegamos ao seguinte código em VHDL para o nosso circuito dos relógios:



```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.numeric_std.ALL;
4
5  entity relogios is
6  port (clk: in std_logic;
7        A, B, C, D: out std_logic);
8  end relogios;
9
10 architecture bhv of relogios is
11
12     signal countA, countB, countC, countD: integer := 0;
13     signal tmpA, tmpB, tmpC, tmpD: std_logic := '0';
14
15 begin
16
17     process(clk)
18     begin
19         if (clk'event and clk='1') then
20
21             countA <= countA+1;
22             countB <= countB+1;
23             countC <= countC+1;
24             countD <= countD+1;
25
26             if (countA = 1250) then -- Resulta em 20000 Hz
27                 tmpA <= NOT tmpA;
28                 countA <= 1;
29             end if;
30             if (countB = 12500) then -- Resulta em 2000 Hz
31                 tmpB <= NOT tmpB;
32                 countB <= 1;
33             end if;
34             if (countC = 125000) then -- Resulta em 200 Hz
35                 tmpC <= NOT tmpC;
36                 countC <= 1;
37             end if;
38             if (countD = 1250000) then -- Resulta em 20 Hz
39                 tmpD <= NOT tmpD;
40                 countD <= 1;
41             end if;
42         end if;
43
44         A <= tmpA;
45         B <= tmpB;
46         C <= tmpC;
47         D <= tmpD;
48
49     end process;
50
51 end bhv;
```

Em seguida, podemos verificar se o nosso comportamento foi descrito de forma correta a partir da criação de uma carta de tempos com alguns testes.

Para este, utilizamos os valores de contagem para meia onda como 5, 6, 7 e 8 (somente para exemplificar uma contagem de forma que o nosso Quartus pudesse rodar e que por inspeção visual fosse perceptível ver que o nosso programa de VHDL funciona corretamente).



Perceba que os valores para o nosso circuito real serão iguais aos escritos dentro do código (1250, 12500, 125000, 1250000 respectivamente).

Assim, podemos perceber que lógica foi pensada e transcrita para VHDL da maneira correta. Ou seja, não teremos erros conceituais dentro deste bloco e, caso venha a ocorrer erros, estes poderão estar atrelados a outros fatores, como pinagem ou da construção de outros blocos lógicos.

Vale ressaltar que não houve a necessidade de acrescentarmos sinais de depuração nem planejar solução de erros para o código em VHDL, pois conseguimos provar que sua implementação já foi bem sucedida e não precisamos corrigir nenhum comportamento inadequado do código.

#### IV. Comportamento do Codificador de Prioridade

Para o codificador de prioridade devemos habilitar 4 entradas que serão representações de 4 fatores externos, e cada uma das situações, aciona um clock específico. Assim, teremos 2 bits de saídas com as combinações 00, 01, 10, 11, sendo o “código” passado para o bloco seletor. Perceba que, como o próprio nome já diz, devemos definir prioridades no código e, portanto, utilizamos o X (=don’t care) para as outras entradas menos relevantes. Podemos desta forma definir o seguinte comportamento para este bloco:

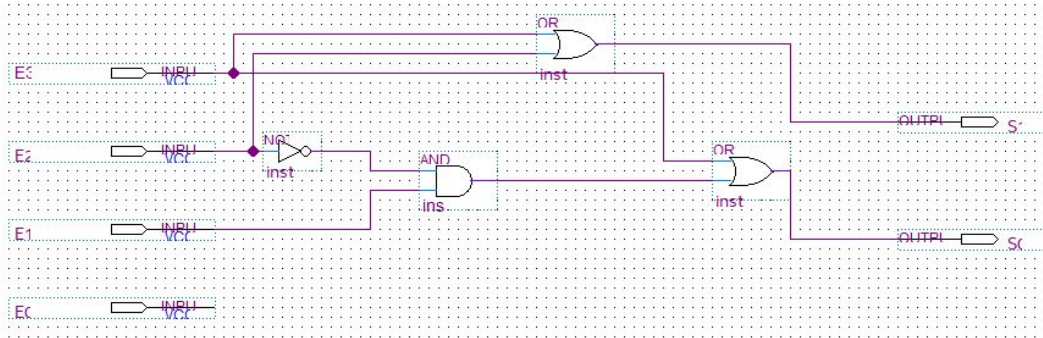
Entradas				Saídas	
E3	E2	E1	E0	S1	S0
0	0	0	0	X	X
0	0	0	1	0	0
0	0	1	X	0	1
0	1	X	X	1	0
1	X	X	X	1	1

Ficamos com as seguintes equações para as saídas:

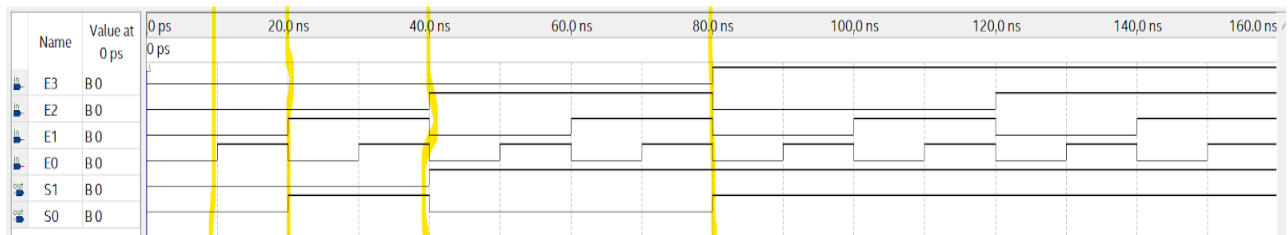
$$S1 = E3 + E2$$

$$S0 = E3 + E2' \cdot E1$$

A partir destas equações conseguimos chegar a seguinte construção de lógica combinatória:



Em seguida, podemos verificar se o nosso comportamento foi descrito de forma correta a partir da criação de uma carta de tempos com alguns testes. Configurei um end time de 160 ns e separei o circuito nos 5 tipos de comportamentos desejados, seguindo a tabela desde o '0000' até o '1111' (para as entradas).



Assim, podemos perceber que nossa montagem lógica foi pensada da maneira correta.

## V. Comportamento do Seletor

O seletor tem o comportamento similar ao de um MUX 4:1 com dois bits seletores que vem do “código” e as 4 entradas a serem seleccionadas para a saída como os clocks. Logicamente, faz sentido que o clock de maior prioridade tenha a maior frequência, pois seria o circuito mais “ágil” ou de alarme “mais estridente”.

Considerando as entradas dos clocks de A a D, sendo A a entrada mais prioritária e D a entrada menos prioritária. Temos a seguinte tabela de comportamento:

Seletor	Saída
---------	-------

S1	S0	Y
0	0	D
0	1	C
1	0	B
1	1	A

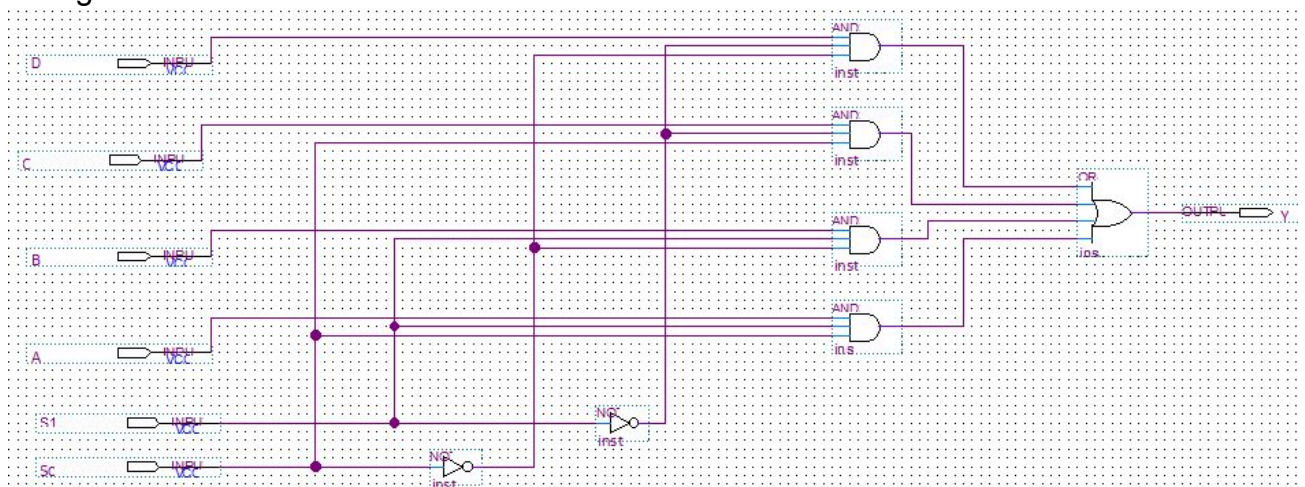
A = 20000Hz será o clock selecionado.

B = 2000Hz será o clock selecionado.

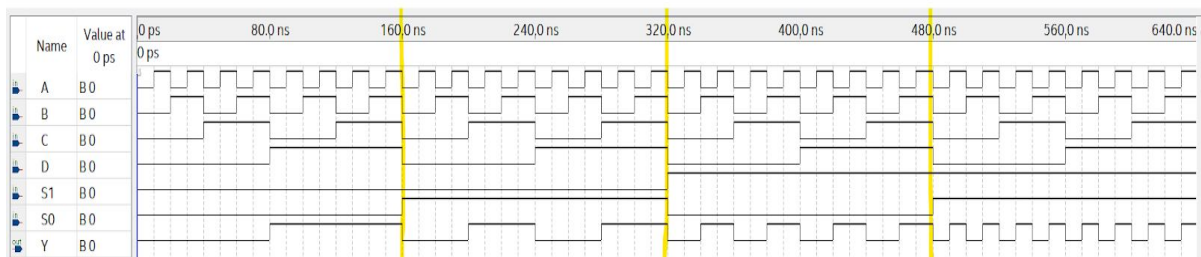
C = 200Hz será o clock selecionado.

D = 20Hz será o clock selecionado.

A partir desta definição, podemos definir o nosso bloco lógico seletor da seguinte forma:



Em seguida, podemos verificar se o nosso comportamento foi descrito de forma correta a partir da criação de uma carta de tempos com alguns testes. Para este teste, setei 4 clocks distintos nas entradas de A a D sendo a entrada A o clock mais rápido. Coloquei a carta de tempos em 640 ns e a dividi em 4 partes de 160 ns. Como podemos perceber, em cada parte do nosso teste, foi-se colocado um valor para S1S0 diferente e podemos ver que a saída corresponde aos clocks selecionados em cada período.



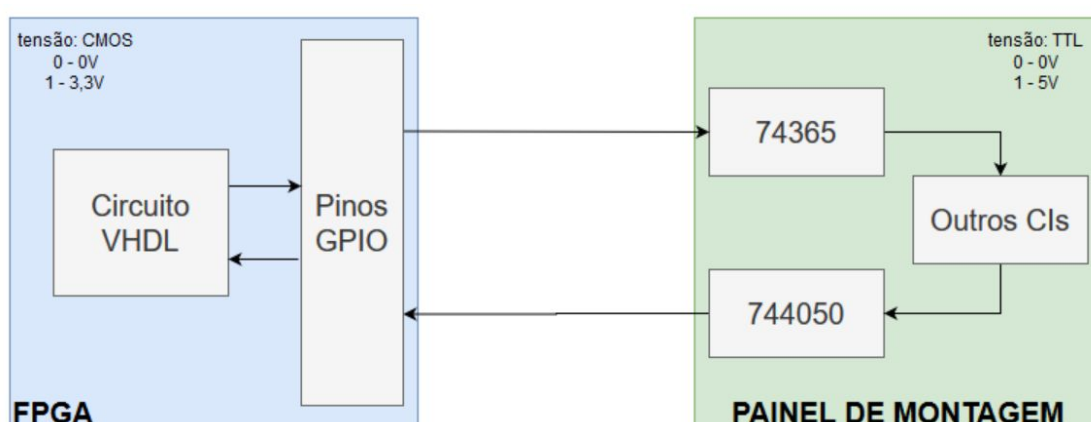


Portanto, podemos inferir que nossa montagem lógica foi pensada corretamente.

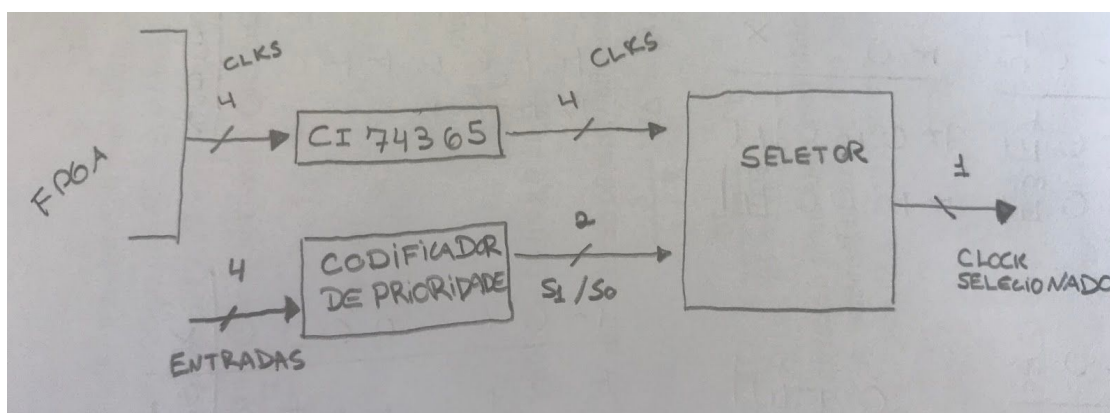
## VI. Elaboração do Circuito Combinatório Completo

Após definirmos como funcionará cada um dos nossos blocos, podemos definir e juntar os comportamentos a serem implementados na placa através de CIs e o outro bloco a ser implementado na FPGA.

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.



Concluimos que, teremos o seguinte bloco a ser montado através de CIs.



Teremos 1 NOT, 1 AND e 2 OR utilizados para o bloco do codificador de prioridade. Já para o bloco de seletor do circuito, é necessário 2 NOT, 3 OR e 8 AND. Logo, percebemos que serão utilizados:

- 1 CI 74365
- 1 CI 7404



- 2 CI 7432
- 2 CI 7408

Ao contarmos o número de CIs e quais CIs serão utilizados, é perceptível a viabilidade da montagem do que planejamos para a placa do Laboratório Digital.

## VII. Estratégia de Montagem, plano de testes e depuração para bloco de combinatória

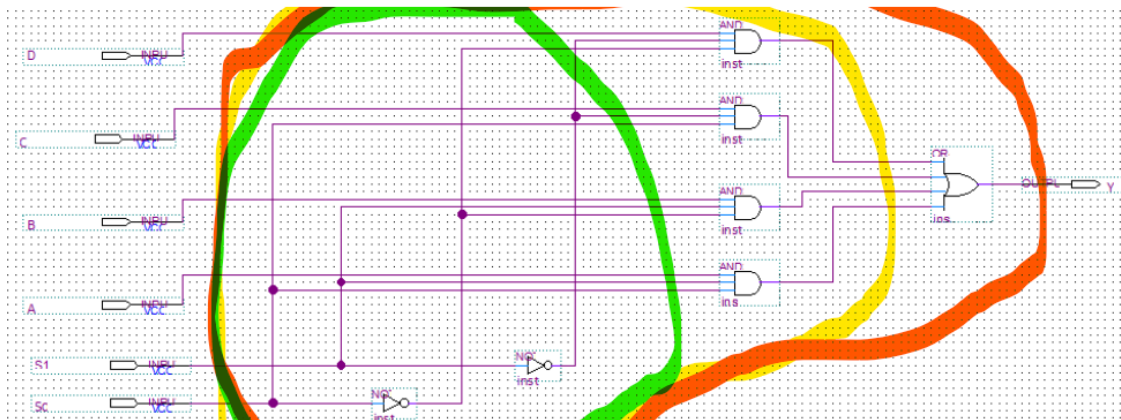
Para realizar a montagem do circuito esperado, descrito anteriormente neste documento, serão necessários alguns passos:

- 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; Inspeccionar tensões de saídas com o auxílio de um multímetro, aplicando diferentes entradas/saídas no circuito (que nesse caso, são A, B, C, D, E3, E2, E1 e E0 nas chaves C0 a C7 e a saída Y 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 três sub-partes (a primeira circulada em verde, a segunda em amarelo e a terceira na cor laranja) e, gradativamente, implementamos algumas portas lógicas e dispositivos, verificando se as saídas até aquele ponto estão conforme o esperado. Respectivamente, implementamos  $\alpha$ ,  $\beta$  e  $\gamma$ .

Perceba que a cada subparte montada, devemos testar os nossos sinais intermediários e comparar com aqueles teoricamente esperados (conforme foi explicitado na tabela verdade para cada uma das fases  $\alpha$ ,  $\beta$  e  $\gamma$ ). Gradativamente, vamos adicionando novos componentes a nossa montagem e, caso encontremos alguma inconsistência, devemos reiniciar a montagem daquela etapa do circuito.



$\alpha$ :

$S_1$	$\bar{S}_1$
0	1
1	0

$S_0$	$\bar{S}_0$
0	1
1	0

$\beta$ :

$D$	$\bar{S}_1$	$\bar{S}_0$	$K'$
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

$C$	$\bar{S}_1$	$\bar{S}_0$	$K''$
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

$B$	$S_1$	$\bar{S}_0$	$K'''$
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

$A$	$S_1$	$S_0$	$K''''$
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

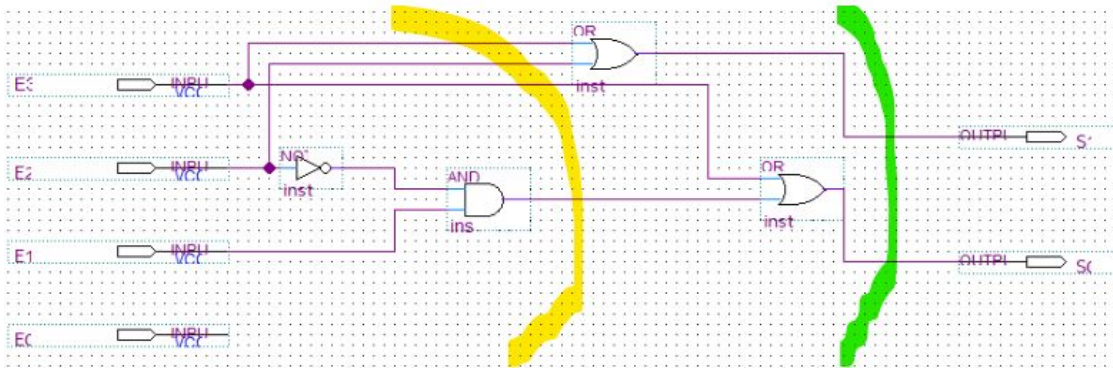
$\gamma$ :

$K'$	$K''$	$K'''$	$K''''$	$Y$
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Acima, temos os  $k'$  /  $k''$  /  $k'''$  /  $k''''$  simbolizando nossos sinais de depuração para este primeiro momento da montagem.

Podemos novamente, dividimos o nosso circuito em três subpartes para a montagem da segunda etapa do bloco combinatório. A primeira circulada em amarelo e a segunda em verde. E, gradativamente, implementamos algumas

portas lógicas e dispositivos, verificando se as saídas até aquele ponto estão conforme o esperado. Respectivamente, implementamos  $\alpha$  e  $\beta$ .



$$\alpha: \begin{array}{c|c} E_2 & \overline{E_2} \\ \hline 0 & 1 \\ 1 & 0 \end{array} \quad \begin{array}{c|c|c} \overline{E_2} & E_1 & x' \\ \hline 0 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{array}$$

$$\beta: \begin{array}{c|c|c} x' & E_2 & S_0 \\ \hline 0 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{array} \quad \begin{array}{c|c|c} \overline{E_2} & E_2 & S_1 \\ \hline 0 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{array}$$

Acima, temos os  $x'$  simbolizando nosso sinal de depuração para este segundo momento da montagem.

Ao terminarmos a montagem beta 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.

É importante ressaltar que em um primeiro momento os testes realizados nessa etapa de montagem será feita com um clock manual. Ou seja, não iremos acoplar a FPGA para dar o clock aqui, mas sim, utilizar os botões para testar se os clocks estão funcionando corretamente de forma manual.

#### VIII. Tabela de Pinagem do Bloco Relógios + Designação de pinos dentro do Quartus

Sinal	Ligação na placa FPGA	Pino na FPGA
clk	CLOCK_50	PIN_M9

A	GPIO_0_D28	PIN_R15
B	GPIO_0_D31	PIN_T20
C	GPIO_0_D33	PIN_T18
D	GPIO_0_D35	PIN_T15

Report

Report not available

Groups

Report

Tasks

Early Pin Planning

Early Pin Planning

Run I/O Assignm

Export Pin Assign

Pin Finder...

Highlight Pins

Top View - Wire Bond

Cyclone V - 5CEBA4F23C7

Named: \*

Edit: x

PIN\_R15

Node Name	Direction	Location	I/O Bank	/REF Group	ttter Locatc	'O Standar	Reserved	rent Stren	Slew Rate	fferential P	nal
A	Output	PIN_R15	5A	B5A_N0	PIN_K22	2.5 V...ault		12mA...ult	1 (default)		
B	Output	PIN_T20	5A	B5A_N0	PIN_N20	2.5 V...ault		12mA...ult	1 (default)		
C	Output	PIN_T18	5A	B5A_N0	PIN_M22	2.5 V...ault		12mA...ult	1 (default)		
clk	Input	PIN_M9	3B	B3B_N0	PIN_M16	2.5 V...ault		12mA...ult			
D	Output	PIN_T15	5A	B5A_N0	PIN_K17	2.5 V...ault		12mA...ult	1 (default)		
<<new node>>											

## IX. Plano de testes para circuito completo

Para testarmos o nosso circuito por completo devemos primeiramente unir a parte da FPGA com a parte da montagem na placa de CIs. Para isso, vale ressaltar que já passamos por todos os outros testes o que garante que o nosso circuito está funcionando por completo em ambos os blocos (lógico combinatório e VHDL). Ou seja, devemos no atentar na conexão entre estes apenas.

Como já foi pinado a nossa FPGA, precisamos conectar os cabos manualmente entre os 4 pinos GPIO e as 4 pontas do CI 74365. A cada fio conectado devemos testar o nosso circuito e analisar se a saída que obtivemos é a esperada, o que garante que aquele fio foi colocado corretamente.

Caso em algum momento, o comportamento esperado não ocorra, devemos remover o último fio colocado, pois provavelmente conectamos-o de forma errada e devemos colocá-lo de novo. Ademais, se o processo for feito muitas

vezes e não obtermos sucesso, provavelmente devemos tentar esse plano de testes novamente em outra FPGA, pois talvez esta esteja quebrada ou não tenha conseguido carregar o nosso bloco de VHDL corretamente.

Se conseguirmos realizar todas essas etapas corretamente, significa que temos o nosso circuito do formato planejado.

Podemos assim, mudar o número de contagens do nosso código VHDL para números mais elevados a fim de conseguirmos enxergar o clock na saída. Ao se conectar o Y (saída) numa LED, por exemplo, podemos fazê-la piscar em diferentes velocidades.

Setamos, então, A para piscar 0.5 seg; B para piscar 1 seg; C para piscar 5 seg; D para piscar 10 seg. Conseguimos fazer uma inspeção visual sobre a saída no momento de teste do nosso circuito.

Em seguida, acionamos uma das “entradas” por vez no painel de montagens (lembre-se que estas estão ligadas aos botões on/off do painel), conseguimos relacionar ao período que pisca a nossa LED e finalizamos o nosso teste de circuito completo garantindo o funcionamento de nosso circuito inteiramente montado e conectado.

## 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](https://balbertini.github.io/vhdl_fsm-pt_BR.html)