Disciplina: PCS 3335 – Laboratório Digital A	
Prof.: *Anarosa*	**Data:** *03/04/2020*
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 das máquinas de estados descritas em VHDL e como utilizá-las em circuitos digitais. A ideia é conseguirmos desenvolver sistemas digitais mais complexos, compostos por fluxo de dados e unidade de controle.

2. Objetivo

O intuito desta experiência é que nós consigamos desenvolver e entender a descrição de máquinas de estados em VHDL e a aplicação de máquinas de estados como unidade de controle de um circuito digital, tudo isso com circuitos mais simples.

3. Planejamento

Projeto de uma Máquina de Estados em VHDL

a. Projeto

i. Descrição funcional (e respectivo Pseudocódigo)

Acordando com o tema do experimento 6 (Máquinas de Estados em VHDL), foi dado para nós o código descrito em VHDL descrito abaixo:

```
-- maquina_estados.vhd
--

library IEEE;
use IEEE.std_logic_1164.all;

entity maquina_estados is
  port ( clock, reset :          in  std_logic;
         iniciar, condicao1, fim: in  std_logic;
         pronto:                 out std_logic;
         zera1, conta1:          out std_logic;
         conta2, carrega2:       out std_logic;
         db_estado:              out std_logic_vector(2 downto 0)
       );
end maquina_estados;

architecture comportamental of maquina_estados is
    type Tipo_estado is (inicial, zeraconts, testacondicao1,
                         contacont2, contacont1, testafim, final);
    signal Eatual, Eprox: Tipo_estado;

begin

  -- proximo estado (reset, borda do clock)
  process (reset, clock)
  begin
      if reset = '1' then
          Eatual <= inicial;
      elsif clock'event and clock = '1' then
          Eatual <= Eprox;
      end if;
  end process;

  -- proximo estado
  process (iniciar, condicao1, fim, Eatual)
  begin
    case Eatual is
      when inicial =>          if iniciar='1'
                               then Eprox <= zeraconts;
                               else Eprox <= inicial;
                               end if;

      when zeraconts =>        Eprox <= testacondicao1;

      when testacondicao1 =>   if condicao1='1'
                               then Eprox <= contacont2;
                               else Eprox <= contacont1;
                               end if;

      when contacont2 =>       Eprox <= contacont1;

      when contacont1 =>       Eprox <= testafim;

      when testafim =>         if fim='1'
                               then Eprox <= final;
                               else Eprox <= testacondicao1;
                               end if;

      when final =>            Eprox <= inicial;

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

  -- saidas
  with Eatual select
      zera1 <= '1' when zeraconts, '0' when others;

  with Eatual select
      carrega2 <= '1' when zeraconts, '0' when others;

  with Eatual select
      conta2 <= '1' when contacont2, '0' when others;

  with Eatual select
      conta1 <= '1' when contacont1, '0' when others;

  with Eatual select
      pronto <= '1' when final, '0' when others;

  with Eatual select
      db_estado <= "000" when inicial,
                   "001" when zeraconts,
                   "010" when testacondicao1,
                   "011" when contacont2,
                   "100" when contacont1,
                   "101" when testafim,
                   "110" when final,
                   "111" when others;

end comportamental;
```

Ao analisarmos tal código, podemos perceber que ele é um circuito sequencial (pela maneira de estruturação de seus ‘process’) e, olhando mais a fundo,

reconhecemos que se trata de uma máquina de estados de Moore (pois seu estado está relacionado com a saída).
Conseguimos ver também que há três ‘inputs’ utilizados como condicionais para nossa máquina de estado (‘iniciar’; ‘condicao1’ e ‘fim’) e que há 7 estados em nosso sistema.

Podemos, portanto, descrever o programa em VHDL acima por um pseudocódigo que demonstra bem sua funcionalidade:

Programa ‘maquina-estados.vhd’

Início:

A qualquer momento, se reset = 1, então o estado atual se torna ‘inicial’.
Senão, se clock estiver em subida, o próximo estado se tornará o estado atual.

//A respeito dos estados atuais e futuros:

Quando o estado atual for ‘inicial’:
‘db_estado’ é = ‘000’.
Se ‘iniciar’ = 1, então próximo estado é ‘zeraconts’
senão próximo estado é inicial.

Quando o estado atual for ‘zeraconts’:
‘db_estado’ é = ‘001’.
O próximo estado será ‘testacondicao1’.

Quando o estado atual for ‘testacondicao1’:
‘db_estado’ é = ‘010’.
Se ‘condicao1’ = 1, então próximo estado é ‘contacont2’.
senão próximo estado é ‘contacont1’.

Quando o estado atual for ‘contacont2’:
‘db_estado’ é = ‘011’.
O próximo estado é ‘contacont1’.

Quando o estado atual for ‘contacont1’:
‘db_estado’ é = ‘100’.
O próximo estado é ‘testafim’.

Quando o estado atual for ‘testafim’:
‘db_estado’ é = ‘101’.
Se ‘fim’ = 1, então próximo estado é ‘final’
senão próximo estado é ‘testacondicao1’’.

Quando o estado atual for ‘final’:

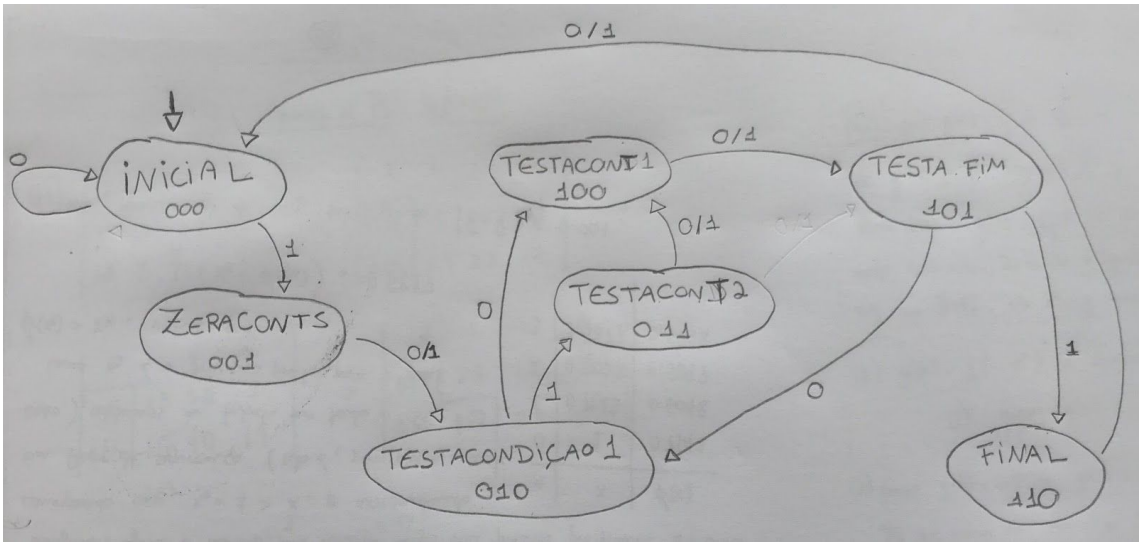
‘db_estado’ é = ‘110’.
O próximo estado é ‘inicial’.

Quando o estado atual for outro diferente dos estados citados acima:
‘db_estado’ é = ‘111’.
O próximo estado é ‘inicial’.

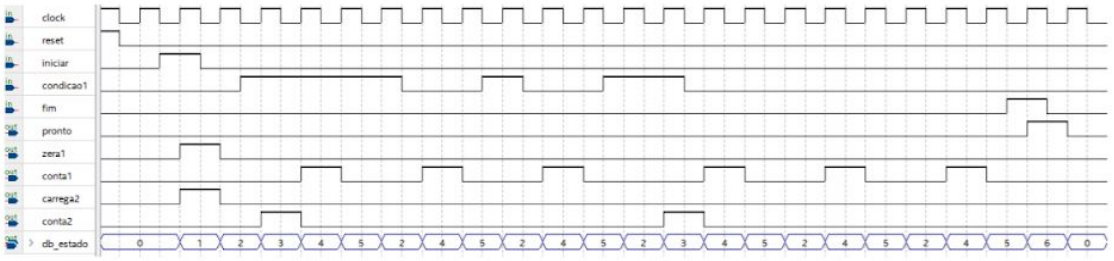
Fim

ii. Diagrama de Transição de Estados

A partir dos estudos citados acima com o código fornecido, conseguimos ver que há uma similaridade da maneira em que o código foi escrito com o exemplo de Moore com dois process no documento do professor Bruno Albertini [1] e, dessa forma, conseguimos desenvolver um diagrama de transição de estados para tal código.



iii. Simulação base



iv. Simulações novo projeto no Intel Quartus Prime

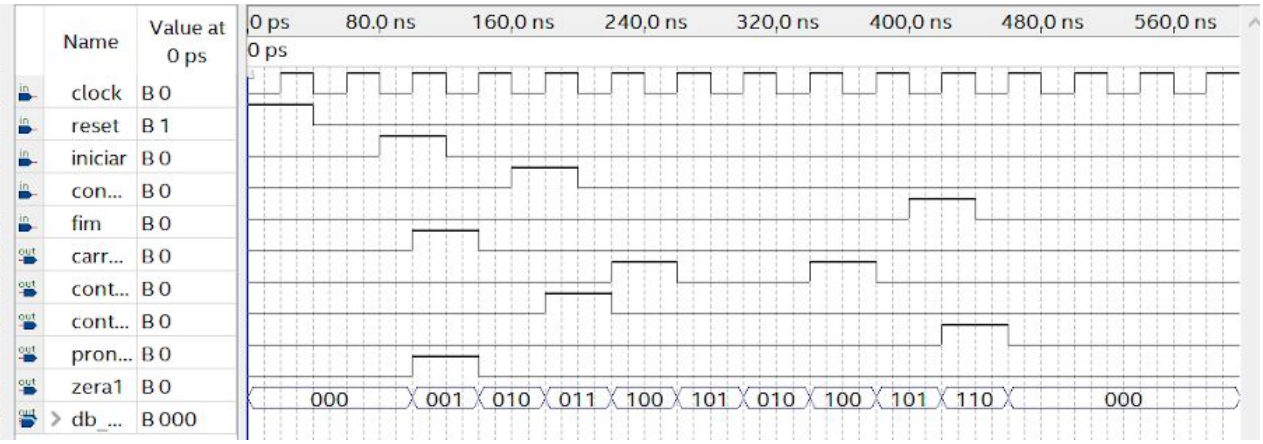
Após criarmos um projeto dentro do Quartus e reescrever o código mostrado na imagem acima, iremos fazer um plano de testes para simular os sinais de onda da saída e compreender se tudo ocorre como o esperado.

Primeiramente, ajustamos o ‘clock’ (com um ciclo de 40ns), setamos um end time para 0.6 micro segundos (tempo suficiente para o nosso teste) e colocamos o ‘clear’ 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 ‘0’ e ‘1’ e verificando se nosso programa se mantém no estado ‘inicial’ e vai para o estado ‘zeraconts’, respectivamente.

Logo, passaremos para o estado ‘testacondicao1’ e testaremos sua passagem para o estado ‘testacont1’. Iremos ao estado ‘testafim’ e colocaremos o input ‘1’ para ele retornar ao estado ‘testacondicao1’.

Após isso, iremos por outro caminho ao ‘testafim’, passando pelo ‘testacont2’ seguido do estado ‘testacont1’. Para terminar a nossa simulação, saíremos do ‘testafim’ para o estado ‘final’ e tentaremos retornar ao estado ‘inicial’. É perceptível que deixamos um tempo além do necessário no estado inicial para entender se ele continuaria lá (o que era desejado) até a indicação do input ‘1’ no ‘iniciar’.

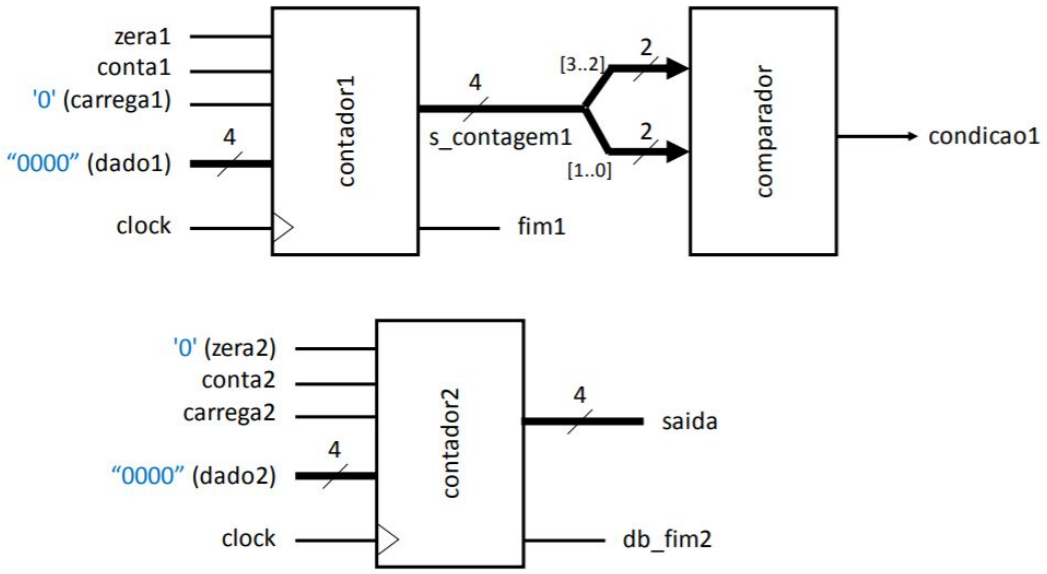


Projeto de um Fluxo de Dados em VHDL

I. Descrição funcional

Nesse projeto foi pedido para modificarmos o circuito digital do Experimento 05 para que conseguíssemos montar um componente de fluxo de dados. Basicamente, o novo circuito será uma combinatória dos projetos

desenvolvidos no experimento anterior, portanto, temos o seguinte diagrama de blocos:



Como podemos perceber na esquematização acima o fluxo de dados irá ativar a ‘condicao1’, quando os dois primeiros dígitos do vetor de contagem for igual ao dois últimos (em ‘0000’; ‘0101’; ‘1010’; ‘1111’), vale ressaltar que como fixamos a entrada do contador com o ‘carrega1’ zerado, o comparador atingirá a saída ‘1’ de forma periódica, pois não conseguiremos carregar valores (quebramos essa periodicidade somente se zerarmos).

Para o contador 2, percebemos que seu funcionamento é basicamente o mesmo do contador original. Contudo, esse não consegue ser zerado através do botão ‘zera2’, podemos fazer o mesmo efeito de zerar nesse contador através da ativação do ‘carrega2’, uma vez que o ‘dado2’ está fixado em ‘0000’.

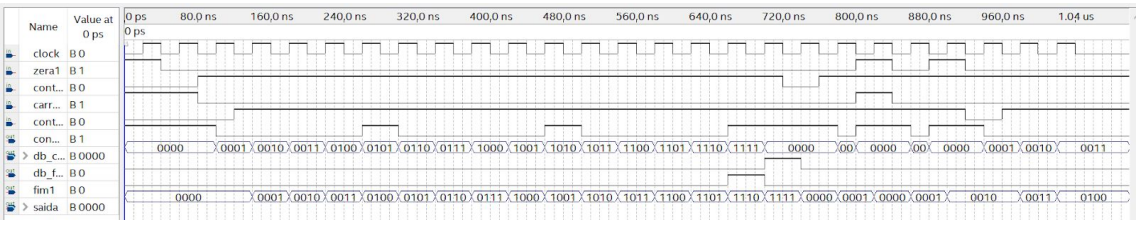
II. Plano de Testes e Simulação no Quartus

Após criarmos um novo projeto dentro do Quartus e ajustar o código dos sub componentes, iremos fazer um plano de testes para simular os sinais de onda da saída e compreender se tudo ocorre como o esperado.

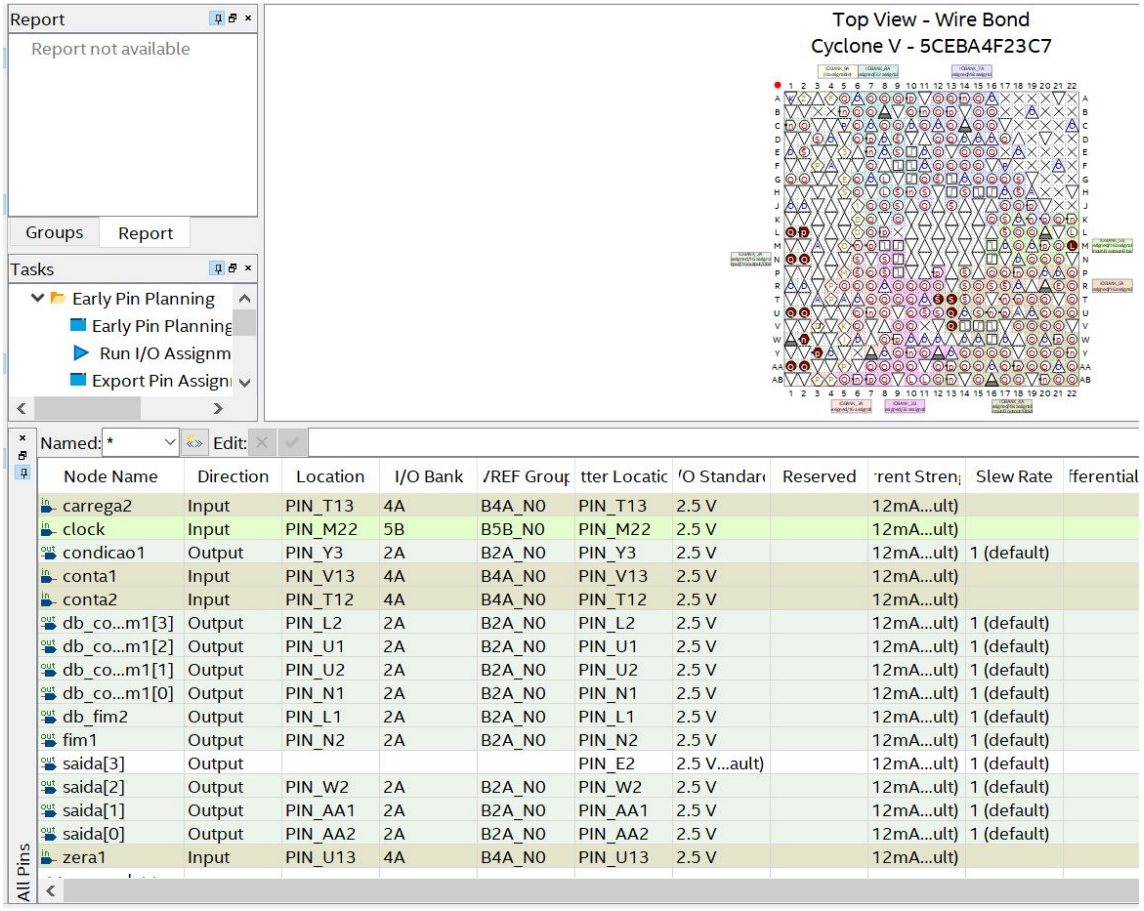
Primeiramente, ajustamos o ‘clock’ (com um ciclo de 40ns), setamos um end time para 1.1 micro segundos (tempo suficiente para o nosso teste) e colocamos o ‘zera1’ e ‘carrega2’ em ‘1’ para garantir que nosso circuito não tem resquícios que possa atrapalhar nosso teste (aplicamos a idéia similar ao reset comum).

Em seguida, vamos testar a funcionalidade do ‘contador1’ + ‘comparador’ e ver se eles funcionam corretamente um ciclo completo mostrando as saídas esperadas, faremos uma pausa na ativação do ‘conta1’ em alguns momentos e ver se o projeto responde ao que queríamos. Após um ciclo e meio, podemos testar se o zerar funciona corretamente.

Concomitantemente com o teste do ‘contador1’ + ‘comparador’, testaremos o ‘contador2’ em seu ciclo completo. Por fim, no meio do segundo ciclo, testaremos a pausa na ativação do ‘conta2’ e seu método de zerar (ativando o ‘carrega2’).



III. Pinagem placa FPGA



Node Name	Direction	Location	I/O Bank	/REF Group	tter Locatic	'O Standar	Reserved	rent Stren	Slew Rate	fferential
carrega2	Input	PIN_T13	4A	B4A_N0	PIN_T13	2.5 V		12mA...ult)		
clock	Input	PIN_M22	5B	B5B_N0	PIN_M22	2.5 V		12mA...ult)		
condicao1	Output	PIN_Y3	2A	B2A_N0	PIN_Y3	2.5 V		12mA...ult)	1 (default)	
conta1	Input	PIN_V13	4A	B4A_N0	PIN_V13	2.5 V		12mA...ult)		
conta2	Input	PIN_T12	4A	B4A_N0	PIN_T12	2.5 V		12mA...ult)		
db_co...m1[3]	Output	PIN_L2	2A	B2A_N0	PIN_L2	2.5 V		12mA...ult)	1 (default)	
db_co...m1[2]	Output	PIN_U1	2A	B2A_N0	PIN_U1	2.5 V		12mA...ult)	1 (default)	
db_co...m1[1]	Output	PIN_U2	2A	B2A_N0	PIN_U2	2.5 V		12mA...ult)	1 (default)	
db_co...m1[0]	Output	PIN_N1	2A	B2A_N0	PIN_N1	2.5 V		12mA...ult)	1 (default)	
db_fim2	Output	PIN_L1	2A	B2A_N0	PIN_L1	2.5 V		12mA...ult)	1 (default)	
fim1	Output	PIN_N2	2A	B2A_N0	PIN_N2	2.5 V		12mA...ult)	1 (default)	
saida[3]	Output				PIN_E2	2.5 V...ault)		12mA...ult)	1 (default)	
saida[2]	Output	PIN_W2	2A	B2A_N0	PIN_W2	2.5 V		12mA...ult)	1 (default)	
saida[1]	Output	PIN_AA1	2A	B2A_N0	PIN_AA1	2.5 V		12mA...ult)	1 (default)	
saida[0]	Output	PIN_AA2	2A	B2A_N0	PIN_AA2	2.5 V		12mA...ult)	1 (default)	
zera1	Input	PIN_U13	4A	B4A_N0	PIN_U13	2.5 V		12mA...ult)		

Projeto de um Sistema Digital

a. Projeto

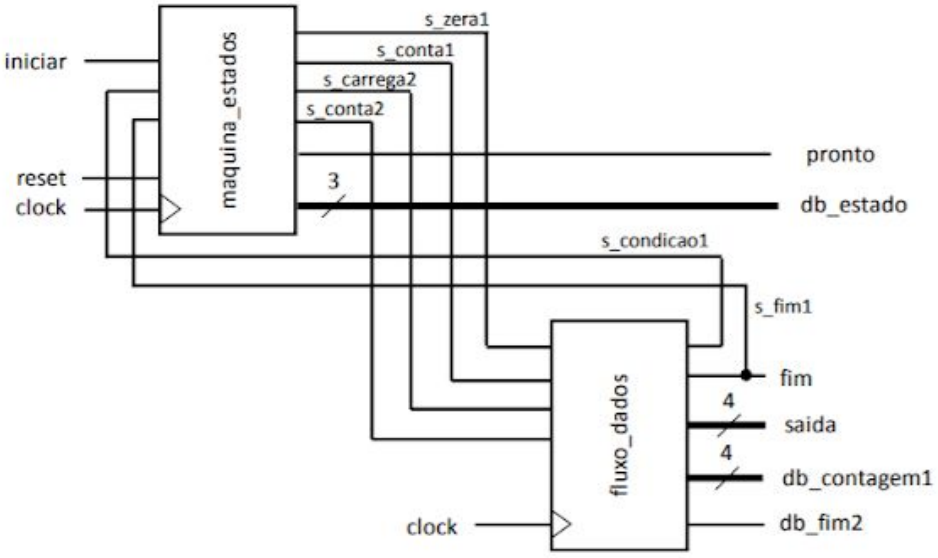
I. Descrição Funcional

O ‘circuito4’ é composto pela interligação dos dois outros componentes VHDL feitos neste relatório (a máquina de estados + o fluxo de dados) e sua função é exatamente exemplificar como a maioria dos sistemas são construídos: com 1 unidade de controle + um local de passagem de dados.

II. Diagrama de Blocos

Neste projeto foi pedido para combinarmos o circuito do fluxo de dados com o componente da máquina de estados para que conseguíssemos montar um projeto completo de UC + Dados. Basicamente, o novo circuito será uma

combinatória dos projetos desenvolvidos nesse experimento, portanto, temos o seguinte diagrama de blocos:

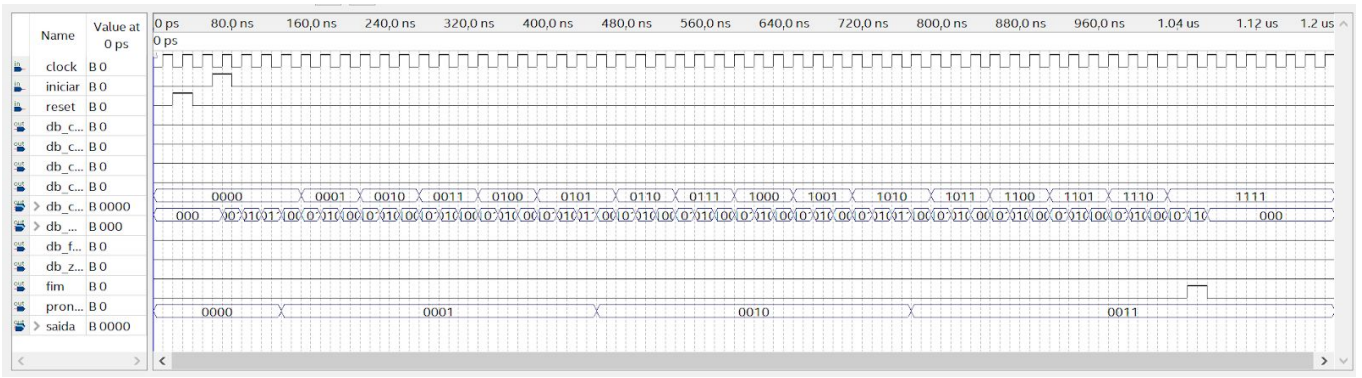


III. Plano de Testes e Simulação no Quartus

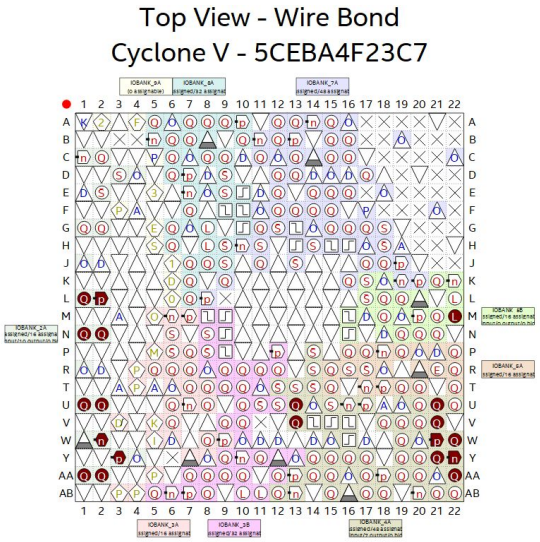
Após criarmos um último projeto dentro do Quartus e juntar o código do fluxo de dados e da máquina de estados, iremos fazer um plano de testes para simular os sinais de onda da saída e compreender se tudo ocorre como o esperado.

Primeiramente, ajustamos o ‘clock’ (com um ciclo de 20ns), setamos um end time para 1.2 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, aplicamos o input ‘iniciar’ e, como podemos ver, o sistema ocorre como o esperado nas etapas seguintes.

Podemos perceber que este projeto tem uma funcionalidade praticamente igual ao descrito nos outros experimentos. A ideia é que teremos a mesma tomada de decisão vista nas máquinas de estados e podemos acompanhar, através do output ‘db_estado’, em qual estado o programa está. O output ‘pronto’ é acionado quando se tem passado pelo estado ‘FINAL’ , assim, o programador pode saber quando o seu ciclo nas máquinas de estados se finaliza somente olhando uma saída. Vale ressaltar também que caso não se acione o botão ‘iniciar’ novamente após o ‘pronto’, o sistema seguirá no estado inicial e, portanto, estará em stand-by.



IV. Pinagem na placa FPGA

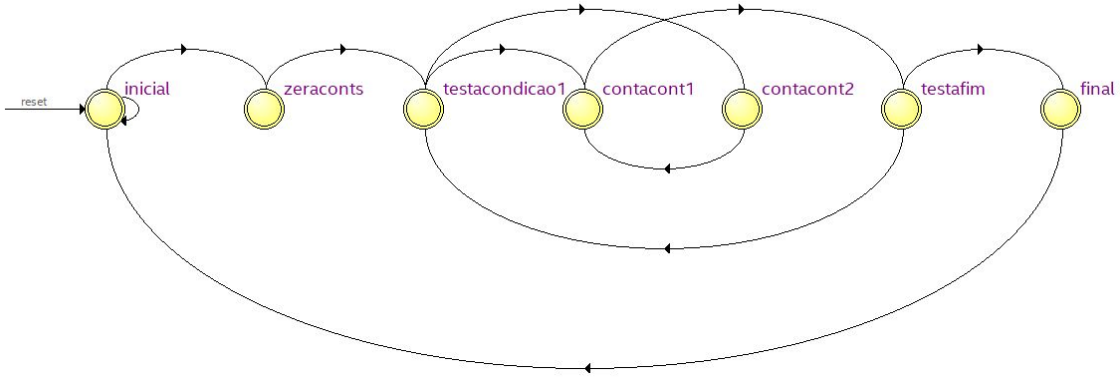


clock	Input	PIN_M22	5B	B5B_N0	2.5 V...ault)		12mA...ult)	
db_carrega2	Output	PIN_Y21	4A	B4A_N0	2.5 V...ault)		12mA...ult)	1 (default)
db_condicao1	Output	PIN_AA22	4A	B4A_N0	2.5 V...ault)		12mA...ult)	1 (default)
db_conta1	Output	PIN_W21	4A	B4A_N0	2.5 V...ault)		12mA...ult)	1 (default)
db_conta2	Output	PIN_Y22	4A	B4A_N0	2.5 V...ault)		12mA...ult)	1 (default)
db_co...m1[3]	Output	PIN_U2	2A	B2A_N0	2.5 V...ault)		12mA...ult)	1 (default)
db_co...m1[2]	Output	PIN_N1	2A	B2A_N0	2.5 V...ault)		12mA...ult)	1 (default)
db_co...m1[1]	Output	PIN_N2	2A	B2A_N0	2.5 V...ault)		12mA...ult)	1 (default)
db_co...m1[0]	Output	PIN_Y3	2A	B2A_N0	2.5 V...ault)		12mA...ult)	1 (default)
db_estado[2]	Output	PIN_L1	2A	B2A_N0	2.5 V...ault)		12mA...ult)	1 (default)
db_estado[1]	Output	PIN_L2	2A	B2A_N0	2.5 V...ault)		12mA...ult)	1 (default)
db_estado[0]	Output	PIN_U1	2A	B2A_N0	2.5 V...ault)		12mA...ult)	1 (default)
db_fim2	Output				2.5 V...ault)		12mA...ult)	1 (default)
db_zera1	Output	PIN_W22	4A	B4A_N0	2.5 V...ault)		12mA...ult)	1 (default)
fim	Output	PIN_V21	4A	B4A_N0	2.5 V...ault)		12mA...ult)	1 (default)
iniciar	Input	PIN_V13	4A	B4A_N0	2.5 V...ault)		12mA...ult)	
pronto	Output	PIN_U21	4A	B4A_N0	2.5 V...ault)		12mA...ult)	1 (default)
reset	Input	PIN_U13	4A	B4A_N0	2.5 V...ault)		12mA...ult)	
saida[3]	Output				2.5 V...ault)		12mA...ult)	1 (default)
saida[2]	Output	PIN_W2	2A	B2A_N0	2.5 V...ault)		12mA...ult)	1 (default)
saida[1]	Output	PIN_AA1	2A	B2A_N0	2.5 V...ault)		12mA...ult)	1 (default)
saida[0]	Output	PIN_AA2	2A	B2A_N0	2.5 V...ault)		12mA...ult)	1 (default)

4. Relatório

I. State Machine Viewer para o ‘maquina_estados’

Como podemos ver a nossa máquina de estados está de acordo com o que planejamos para o seu funcionamento.

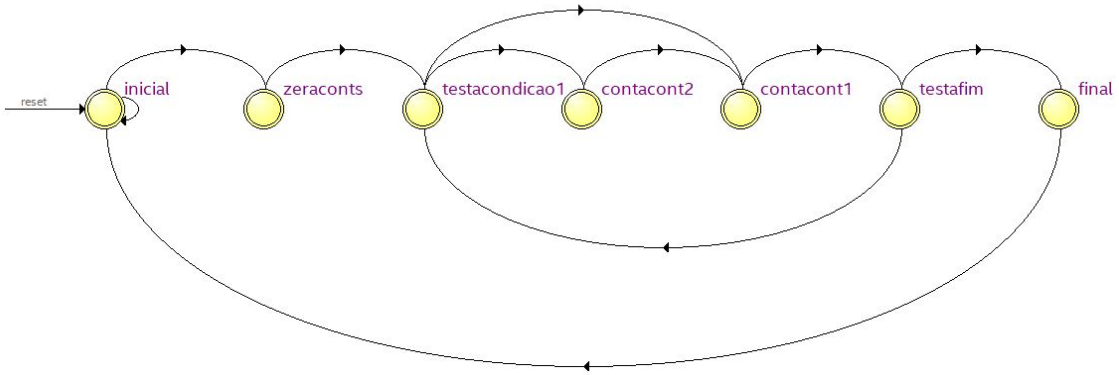


	Source State	Destination State	Condition
1	contacont1	testafim	
2	contacont2	contacont1	
3	final	inicial	
4	inicial	inicial	(!iniciar)
5	inicial	zeraconts	(iniciar)
6	testacondicao1	contacont1	(!condicao1)
7	testacondicao1	contacont2	(condicao1)
8	testafim	final	(fim)
9	testafim	testacondicao1	(!fim)
10	zeraconts	testacondicao1	

	Name	final	testafim	contacont1	contacont2	testacondicao1	zeraconts	inicial
1	inicial	0	0	0	0	0	0	0
2	zeraconts	0	0	0	0	0	1	1
3	testacondicao1	0	0	0	0	1	0	1
4	contacont2	0	0	0	1	0	0	1
5	contacont1	0	0	1	0	0	0	1
6	testafim	0	1	0	0	0	0	1
7	final	1	0	0	0	0	0	1

II. State Machine Viewer para o ‘circuito4’

Como podemos ver a nossa máquina de estados está de acordo com o que planejamos para o seu funcionamento.



	Source State	Destination State	Condition
1	contacont1	testafim	
2	contacont2	contacont1	
3	final	inicial	
4	inicial	zeraconts	(iniciar)
5	inicial	inicial	(!iniciar)
6	testacondicao1	contacont1	(!condicao1)
7	testacondicao1	contacont2	(condicao1)
8	testafim	testacondicao1	(!fim)
9	testafim	final	(fim)
10	zeraconts	testacondicao1	

	Name	final	testafim	contacont1	contacont2	testacondicao1	zeraconts	inicial
1	inicial	0	0	0	0	0	0	0
2	zeraconts	0	0	0	0	0	1	1
3	testacondicao1	0	0	0	0	1	0	1
4	contacont2	0	0	0	1	0	0	1
5	contacont1	0	0	1	0	0	0	1
6	testafim	0	1	0	0	0	0	1
7	final	1	0	0	0	0	0	1

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