UNIVERSIDADE DE BRASÍLIA

Adrianne Alves da Silva - 160047595
Ana Paula Gomes de Matos - 160023629

TURMA "F"
PRÁTICA DE ELETRÔNICA DIGITAL - 119466
Experimento 07: Circuitos contadores síncronos e assíncronos.

Brasília - DF

09 de junho de 2017

UNIVERSIDADE DE BRASÍLIA

PRÁTICA DE ELETRÔNICA DIGITAL - 119466

Relatório referente ao quarto experimento da disciplina Prática de Eletrônica Digital, ministrada pela professora Lourdes Mattos Brasil, realizado no dia 02 de junho de 2017.

Alunas:

Adrianne Alves da Silva

Ana Paula Gomes de Matos

Brasília - DF

SUMÁRIO

INTRODUÇÃO.....03
1.1 Contadores assíncronos.....03
1.1.1 Contador assíncrono crescente.....03
1.1.2 Contadores com módulo 2^n 04
1.1.3 Contadores assíncrono decrescente.....05
1.1.4 Contadores assíncrono ascendente-descendente.....05
1.1.5 Atrasos de propagação de contadores assíncronos.....06
1.2 Contadores síncronos.....06
1.2.1 Contadores síncronos ascendentes.....06
1.2.2 Contadores síncronos decrescentes.....07
1.2.3 Contadores com carga paralela.....07
1.3 Aplicação com contadores.....07
1.4 Objetivo.....08
1.5 Justificativa.....08
2 PARTE EXPERIMENTAL.....09
2.1 Preparação do ambiente de simulação.....09
2.2 Experimento.....12
2.2.1 Simulação Contador de módulo 10.....12
2.2.2 Contador Assíncrono de módulo 10 com ordem crescente e decrescente.....12
2.2.3 Contador síncrono de módulo 10.....17
2.2.4 Contador síncrono com sequência diferenciada.....20
3 DISCUSSÃO.....27
CONCLUSÕES.....28
REFERÊNCIAS BIBLIOGRÁFICAS.....29
DIAGRAMAS ESQUEMÁTICOS.....30

1 INTRODUÇÃO

As funções dos flip-flops são consideradas ilimitadas dentro de sistemas digitais. Eles podem ser utilizados e associados como contadores, registradores, entre outros circuitos.

Os contadores podem ser classificados em dois grupos:

- Assíncronos: apresentam sinal de clock dividido até o último flip-flop (FF).
- Síncronos: todos os FF possuem um sinal de clock em comum.

Também existe um grupo diferenciado denominado de "contadores em anel" que são obtidos diretamente dos registradores de deslocamento. [1]

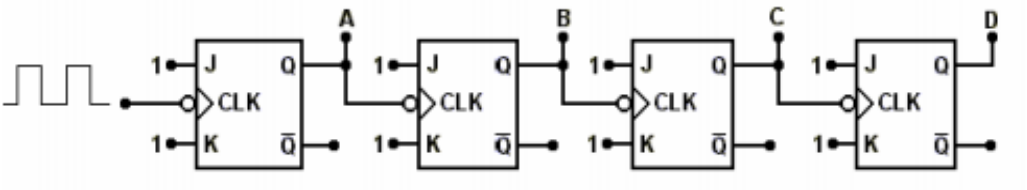
1.1 Contadores assíncronos

São aqueles conhecidos como seriais ou contadores por pulsação (*ripple counter*). Esse nome advém do fato dos FF's do controlador não serem disparados diretamente pelo sinal de clock, pois cada um é disparado pela saída do FF anterior, isso torna os contadores limitados em termo de velocidade, pois o tempo para a resposta chegar é dado pelo tempo de atraso de cada FF. Também podem ocorrer impulsos indesejados, caso os decodificadores sejam usados para indicar a ocorrência uma determinada saída.[1]

1.1.1 Contador assíncrono crescente

As entradas J e K ficam em nível lógico alto enquanto estão configurados como flip-flop tipo T, vale ressaltar que cada FF é disparado pela saída Q do FF anterior.

Figura 01: Contador assíncrono crescente



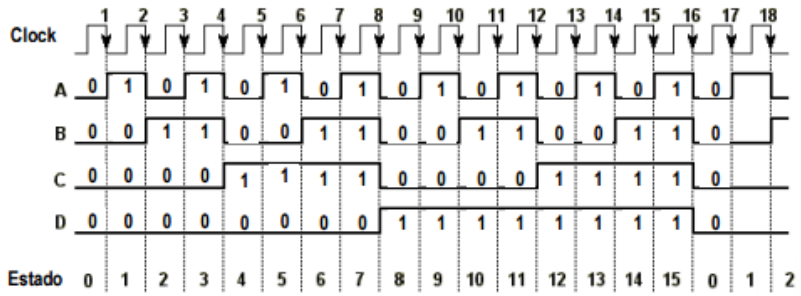


Tabela 01: Estados do contador assíncrono crescente

clock	D	C	B	A	Decimal	Estado
0	0	0	0	0	0	0
1	0	0	0	1	1	1
2	0	0	1	0	2	2
3	0	0	1	1	3	3
4	0	1	0	0	4	4
5	0	1	0	1	5	5
6	0	1	1	0	6	6
7	0	1	1	1	7	7
8	1	0	0	0	8	8
9	1	0	0	1	9	9
10	1	0	1	0	10	10
11	1	0	1	1	11	11
12	1	1	0	0	12	12
13	1	1	0	1	13	13
14	1	1	1	0	14	14
15	1	1	1	1	15	15
16	0	0	0	0	0	0
17	0	0	0	1	1	1
18	0	0	1	0	2	2

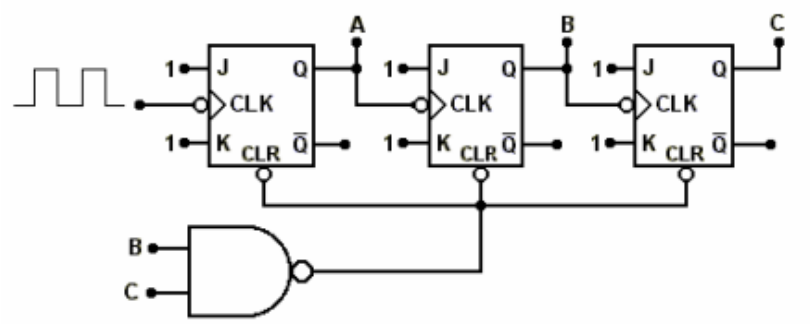
O contador tem como conteúdo de estado interno a contagem do número de transições negativas do clock, quando isso ocorre o conteúdo é incrementado uma unidade.

O número de estados distintos de um contador é denominado módulo. O módulo com N flip-flop pode ser no máximo 2^n saídas. Nos contadores assíncronos, a frequência do clock é dividida por 2 nos flip-flops.

1.1.2 Contadores com módulo 2^n

Ao adicionar um circuito decodificador que reinicia a contagem antes de se chegar a valores máximos ou mínimos, obtém-se um contador com módulo menor que 2^n**.**

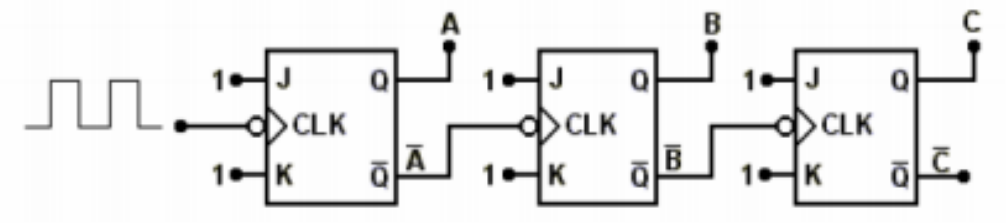
Figura 02: Contador assíncrono crescente



1.1.3 Contadores assíncrono decrescente

A configuração de um contador decrescente é construída com FF do tipo JK, a diferença dele para o crescente é só o fato de cada flip-flop ser disparado pela saída complementar de Q no lugar de ser só Q.

Figura 02: Contador assíncrono decrescente

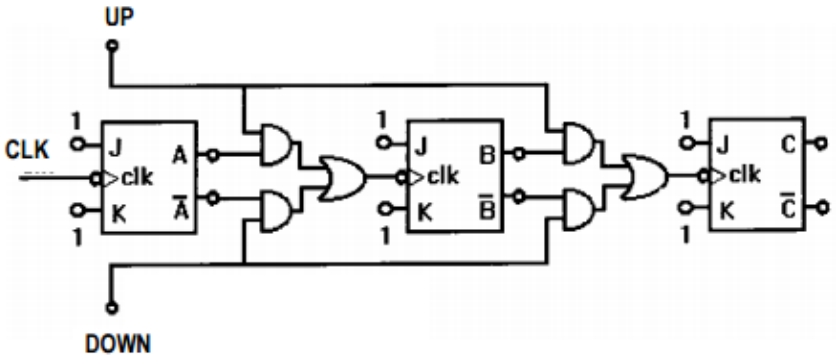


1.1.4 Contadores assíncrono ascendente-descendente

Esse é uma combinação dos contadores apresentados anteriormente. A configuração desse tipo de contador permite que ele tenha dois modos de operações distintos:

- **Modo de contagem ascendente:** UP = 1 e DOWN = 0, os flip-flops são disparados pelas saídas não complementares do FF anterior.
- **Modo de contagem descendente:** UP = 0 e DOWN = 1, os flip-flops são disparados pelas saídas complementares do FF anterior.

Figura 03: Contador assíncrono ascendente-descendente



1.1.5 Atrasos de propagação de contadores assíncronos

Como visto nos itens anteriores, um contador assíncrono é aquele em que o FF é disparado pela saída do anterior, essa característica traz como desvantagem o acúmulo de atrasos de propagação, pois ao passar por um flip-flop o sinal de clock sofre atraso de propagação e o efeito é somado até o último FF. Para que o contador funcione de modo confiável é necessário que o atraso total seja menor que o período do clock usado (T_{clock} >= N x t_{PD}), em termo de frequência máxima, tem-se:

$$f_{max} = \frac{1}{N \times t_{PD}}.$$

1.2 Contadores síncronos

Como foi visto anteriormente, os atrasos dos FF presentes nos contadores assíncronos limitam a sua frequência máxima, esse problema pode ser resolvido fazendo com que os FF mudem o estado de suas saídas no momento em que ocorre a transição de clock, e é essa a principal característica dos contadores denominados síncronos.

1.2.1 Contadores síncronos ascendentes

A análise da lógica do circuito contador deixa evidente que os FF do tipo JK só estão no estado TOOGLE (J = 1 e K = 1) quando as saídas dos flip-flops anteriores estão em nível lógico alto. Vale ressaltar que o atraso de propagação será dado pela soma do atraso dos FF e das portas AND:

$$\text{atraso total} = t_{PD}(FF) + t_{PD}(AND).$$

Portanto nota-se que o atraso não depende do número de bits, mas sim da tecnologia utilizada, em paralelo, como propagação do erro é menor, o contador síncrono trabalhar com frequências menores que o contador assíncrono.

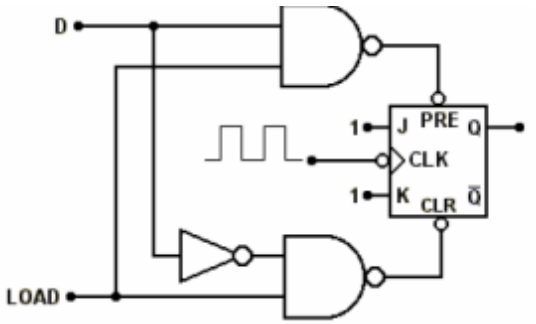
1.2.2 Contadores síncronos decrescentes

Da mesma forma que os contadores assíncronos, os contadores síncronos também podem contar de forma decrescente, para isso deve-se usar as saídas complementares de Q no lugar das Q propriamente ditas na lógica de habilitação da entrada Je K.

1.2.3 Contadores com carga paralela

Os contadores podem contar a partir de um valor previamente determinado pelo usuário, neste caso, o estado inicial de cada FF pode ser determinado através das entradas CLR ($Q \rightarrow 0$) e PRE ($Q \rightarrow 1$). Quando o LOAD é alto, D é armazenado na saída do FF por meio de entradas assíncronas.[2]

Figura 04: Contador assíncrono decrescente



Se isso for feito para cada flip-flop do contador, o usuário poderá predeterminar o valor que será incrementado, esse procedimento é conhecido como carga paralela, pois é utilizado simultaneamente em todos os FF do contador.

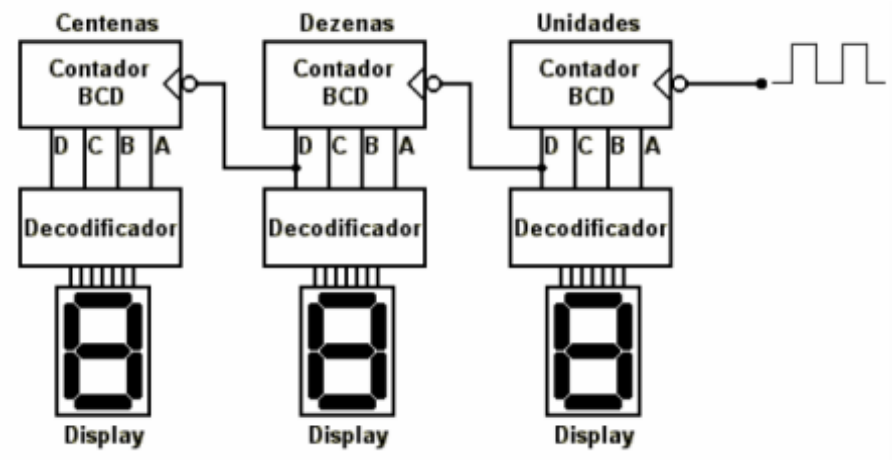
Obs.: O procedimento de construção do contador síncrono crescente-decrescente é semelhante ao do contador assíncrono. [2]

1.3 Aplicação com contadores

Sabe-se que é possível criar circuitos que podem multiplicar, somar, subtrair e dividir, comparar ou fazer qualquer outro tipo de operação aritmética, tudo em questão de nanosegundos, até são criados circuitos capazes de armazenar um valor, esses circuitos são chamados de flip-flop e são bastante utilizados em computadores e calculadoras.

Contadores BCD: são usados em circuitos onde os pulsos devem ser contados e mostrados em um display.

Figura 05: Contador BCD de 000 a 999



No contador apresentado acima, a cada borda de descida do clock, o contador de unidades é incrementado e o valor do contador é mostrado no display, quando o valor máximo for atingido, a próxima transição de descida do clock fará o contador de unidades retornar a 0. Nesse momento, ocorrerá uma transição de 1 para 0 da saída D deste contador, o qual está ligada à entrada de clock do contador de dezenas.

-Contadores como divisores de frequência: outra aplicação dos contadores é na geração digital de clock de menor frequência (f_{clk}) a partir de uma onda quadrada de frequência superior (f_{osc}). [1]

Para identificar a contagem N que o contador deve realizar usando a característica de divisão de frequência, usa-se a seguinte fórmula:

$$N = f_{osc} / f_{clk}$$

1.4 Objetivo

Mostrar ao aluno o funcionamento e implementação dos circuitos contadores síncronos e assíncronos.

1.5 Justificativa

O uso de contadores para a manipulação de informações é extremamente importante para a maioria das atividades desenvolvidas. Nesse sentido, o uso desses circuitos em ferramentas tecnológicas computacionais, como computadores e calculadoras é comum e torna-se muitas vezes imprescindível. Dessa forma, a compreensão dos circuitos que realizam essa atividade é importante para a formação do estudante.

Os circuitos contadores utilizam a lógica sequencial dos flips flops, pois são compostos pelos mesmo, e visto que estes possuem uma gama de aplicações, por envolver memória, de certa forma, eles dominam a tecnologia atual. Dessa forma, o conhecimento e domínio desta tecnologia amplia as oportunidades de implementação, justificando as atividades realizadas neste experimento.

Tendo em vista o que foi exposto, o experimento realizado no dia 02 de junho de 2017, no laboratório de Eletrônica Digital, foi capaz de unir a relevância dos circuitos contadores às aplicações prática, informando e preparando o estudante para prováveis demandas de mercado. Dessa maneira, é de suma importância realizar o registro do processo por meio de documentos como este. Essa perspectiva demonstra a relevância que a visualização prática desses conceitos tem para os alunos.

2 PARTE EXPERIMENTAL

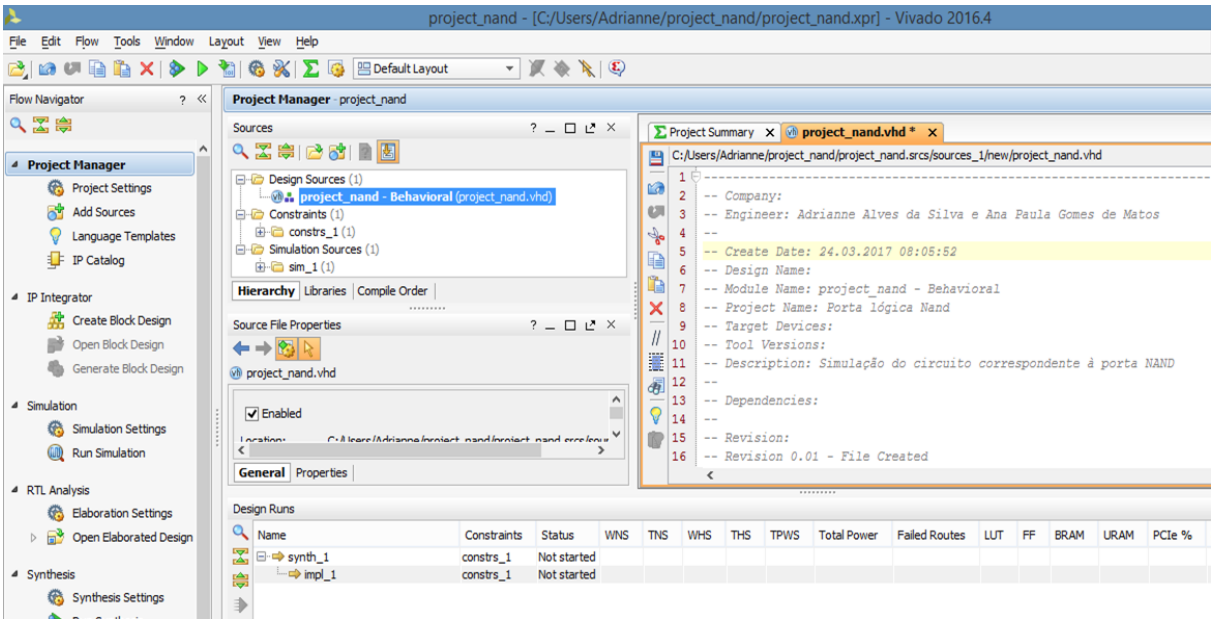
O software utilizado para codificar e simular os experimentos foi o VIVADO, versão 2013.4, rodando no sistema operacional Windows 8, em uma máquina pessoal. Parte do experimento foi realizado em sala, nos computadores da universidade.

2.1 Preparação do ambiente de simulação

O passo inicial para a realização do experimento foi criar um novo projeto no *software* Vivado. A configuração do mesmo foi realizada adicionando a placa Basys 3 ao projeto, por meio da aba "*add or create constraints*". Após esses passos foi necessário apenas criar o arquivo relativo ao *design* , adicionando o arquivo com o tipo vhdl, para que o código pudesse ser implementado nesta linguagem. O módulo foi então definido de acordo com o número de portas de entrada e saída do circuito referente à cada atividade solicitada, nomeadas as entradas, em sua maioria como A,B, va, vb , Cin,sel e Saída com S.

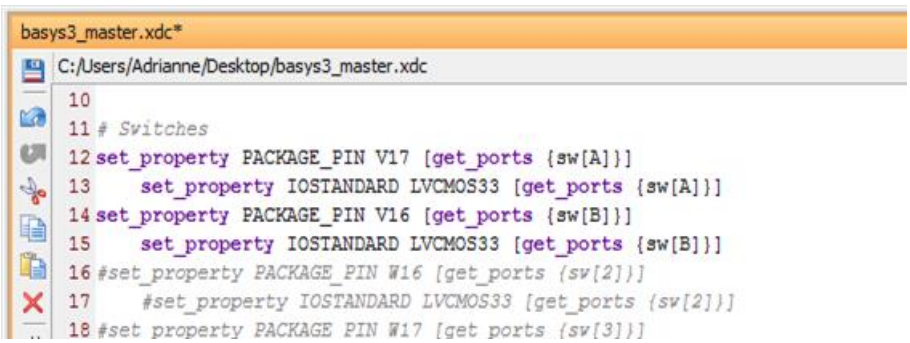
É necessário então ativar na aba de *project settings* , *bitstream* o arquivo bin_file que possibilita escrita binária sem uso do cabeçalho. Assim, obteve-se o ambiente demonstrado na figura a seguir.

Figura 06: Ambiente de trabalho



Sendo assim, foi possível abrir o arquivo vhdl definido e implementá-lo. Para cada atividade, forneceu-se ao arquivo o código e as expressões booleanas adequadas. Por sequência do passo-a-passo fornecido, foram linkados os pinos de entrada e saída da placa Basys com os dispositivos de entrada e saída implementadas. No arquivo basys3.xdc isso foi feito por meio da retirada do comentário das linhas referentes ao número de entradas e saídas presentes em cada atividade, além da adequada atribuição de cada uma das portas, segundo a nomenclatura adotada (J, K, S, R, Sel, Q,,S), essa atividade seria essencial para a verificação diretamente numa FPGA.

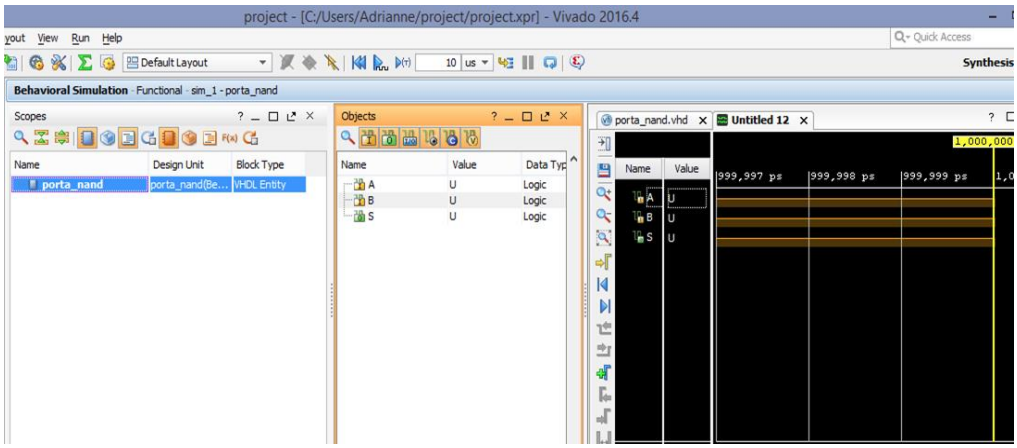
Figura 07: Linkagem dos pinos da placa Basys 3



Após todos esses passos foi necessário sintetizar o circuito por meio do *Run Synthesis* , gerando um esquemático facilmente visualizado no *RTL Analysis* . Nesse ponto, é que tornou-se possível realizar a simulação do experimento, bastando clicar em *run simulation*

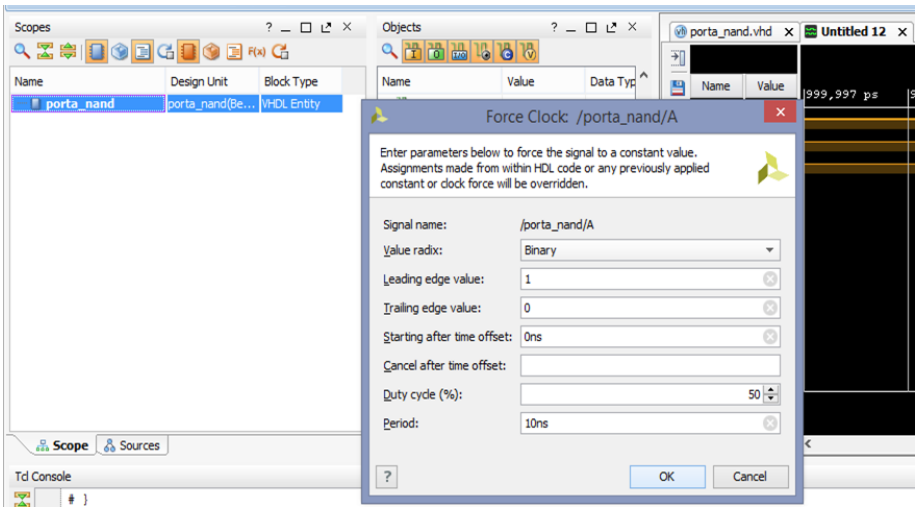
que gera um ambiente de simulação (**figura 08**).

Figura 08: Ambiente de Simulação do circuito



Nesse passo, é possível configurar os valores referentes à cada entrada, assim como o tempo de mudança de sinal (0 ou 1) para a geração de ondas . Isso é feito clicando sobre cada entrada com o botão direito, na opção de *Force Clock*, conforme a figura a seguir.

Figura 09: Configuração de entradas para simulação



A primeira configuração realizada é a modificação do argumento *value* para o tipo binário informando entradas diferenciadas para os argumentos *Leading edge value* e *Trailing edge value*, que seja 0 ou 1. Define-se um período para as ondas, diferentes para cada entrada. É legal se o valor de uma for o dobro da outra, por exemplo. E então, basta executar a simulação.

2.2 Experimento

O experimento consistiu em duas atividades diferentes envolvendo flip flops para o desenvolvimento de contadores. A abrangência alcançou de contadores síncronos à assíncronos, considerando a ordem crescente ou decrescente dos números. Os termos técnicos e ou teóricos dos mesmos foram explanados anteriormente para total compreensão do que foi observado.

2.2.1 Simulação Contador de módulo 10

Esse experimento consistiu na implementação e simulação de 2 contadores de módulo 10, sendo um deles assíncrono e o outro síncrono. Para cada contador implementou-se a função de escolha do tipo de contagem, sendo duas as opções. Entretanto, ambos os contadores foram implementados em um mesmo arquivo e utilizou-se uma seletora para mostrar o resultado escolhido.

2.2.2 Contador Assíncrono de módulo 10 com ordem crescente e decrescente

O contador assíncrono tem como principal característica o fato de apenas o primeiro flip flop receber um clock externo, e o clock dos demais flip flops variam de acordo à saída do anterior. Dessa forma, para executar esse comportamento, basta implementar esses flip flops, isso foi feito em apenas um arquivo. Foi necessária a implementação de um divisor de clock para que este variasse a cada dois segundos, o código obtido para o contador está expresso a seguir:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity contador is
   Port (  clear: in std_logic;
         clock: in std_logic;
         Scresc: out std_logic_vector (3 downto 0);
         Sdecresc: out std_logic_vector (3 downto 0));
end contador;

architecture behavioral of contador is

signal tempc: std_logic_vector (3 downto 0) := "0000";
signal tempd: std_logic_vector (3 downto 0) := "1001";

begin
```

```
process(CLEAR, clock)
begin
   if(CLEAR = '1') then
      tempc <= "0000";
      tempd <= "1001";
   elsif(clock'event and clock = '1') then
      tempc <= tempc + 1;
      tempd <= tempd - 1;
   end if;
end process;

Scresc <= tempc;
Sdecresc <= tempd;

end behavioral;
```

Da mesma maneira, o código obtido para o divisor de clock obedeceu os códigos já antes implementados, obtendo-se:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity divisorclock2 is
   Port (  clock02E: in std_logic;
         clock02S: out std_logic);
end divisorclock2;

architecture behavioral of divisorclock2 is

signal contador: std_logic_vector (28 downto 0) := (others => '0');

begin

process(clock02E)
begin
   if rising_edge(clock02E) then
      contador <= contador + 1;
   end if;
clock02S <= contador(28);
end process;

end behavioral;
```

O clear do contador foi necessário para garantir que a contagem variasse apenas de 0 a 9, para garantir que este fosse de módulo 10. Isso porque, para realizar esse tipo de contagem é necessário 4 flips flops o que daria uma abrangência de contagem até o número 15. O código de implementação do clear foi o seguinte:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity controlclear is
   Port (  B: in std_logic;
         D: in std_logic;
         Scount: out std_logic);
end controlclear;

architecture behavioral of controlclear is

begin

Scount <= B and D;

end behavioral;
```

Para garantir a seleção entre os dois modos, crescente e decrescente, foi preciso implementar uma seletora, obtendo-se o raciocínio expresso no código a abaixo.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity seletor is
   Port (  sel: in std_logic;
         SC: in std_logic_vector (3 downto 0);
         SD: in std_logic_vector (3 downto 0);
         S: out std_logic_vector (3 downto 0));
end seletor;

architecture behavioral of seletor is

begin

with sel select
   S <=    SC when '1',
```

```
        SD when others;

end behavioral;
```

O enunciado do experimento sugere que a passagem destes números deve ser mostrada em um display de 7 seguimentos, dessa forma, esse raciocínio foi implementado. Para tanto, necessitou-se de um divisor de clock que garantisse a passagem dos números, isso resultou no código implementado a seguir.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity divisorclock100 is
    Port (  clock100E: in std_Logic;
        clock100S: out std_logic_vector (1 downto 0));
end divisorclock100;

architecture behavioral of divisorclock100 is

signal count: std_logic_vector (19 downto 0) := (others => '0');

begin

process(clock100E)
begin
    if rising_edge(clock100E) then
        count <= count + 1;
    end if;
clock100S(1) <= count(19);
clock100S(0) <= count(18);
end process;

end behavioral;
```

O multiplexador utilizado , foi expresso da seguinte maneira :

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity mux is
    Port (  E1, E2, E3, E4: in std_logic;
        ESel: in std_logic_vector (1 downto 0);
```

```
        Smux: out std_logic);
end mux;

architecture behavioral of mux is

begin

with ESel select
   Smux <= E1 when "00",
        E2 when "01",
        E3 when "10",
        E4 when others;

end behavioral;
```

O decodificador por sua vez com o auxílio do multiplexador, apenas transformou a saída do contador em 7 segmentos, para que o comportamento desejado pudesse ser observado. É importante salientar que devido o fato de se estar trabalhando com números binários, foi preciso tratar apenas duas condições, quando o valor a ser apresentado deveria ser 0, e quando o mesmo deveria ser 1. Essas condições podem ser observadas na implementação feita.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity decod7seg is
   Port (  e: in std_logic;
        segmentos: out std_logic_vector (6 downto 0));
end decod7seg;

architecture behavioral of decod7seg is

begin

with e select
   segmentos <= "1000000" when '0',
        "1111001" when others;

end behavioral;
```

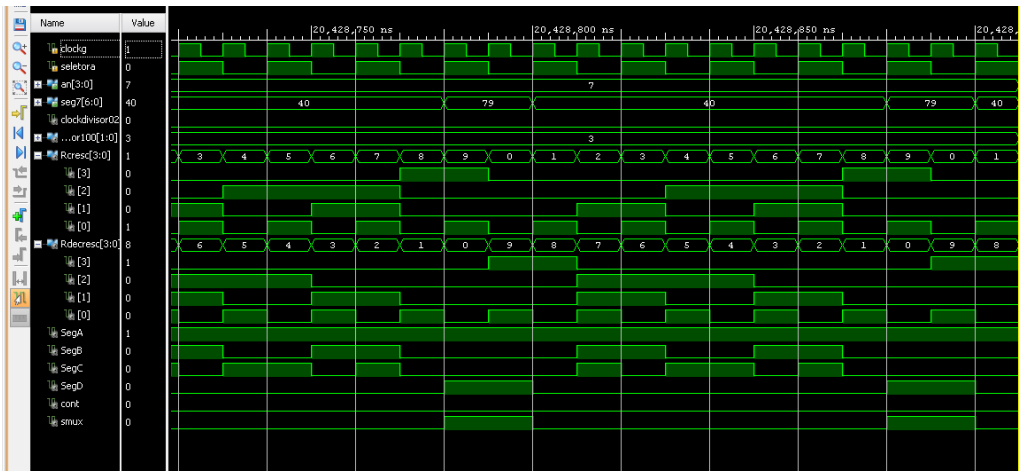
Por fim, dado que na placa basys3 trabalhada há 4 displays de 7 segmentos e como é necessário apenas 1 para demonstrar os números binários, foi preciso desenvolver uma

seletora de display, segundo a qual :

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity seletordisplay is
    Port (  Es: in std_logic_vector (1 downto 0);
            San: out std_logic_vector (3 downto 0));
end seletordisplay;

architecture behavioral of seletordisplay is

begin

with Es select
    San <=    "1110" when "00",
              "1101" when "01",
              "1011" when "10",
              "0111" when others;

end behavioral;
```

A simulação realizada forneceu a seguinte configuração de ondas quadradas:

Figura 10: Simulação de contador assíncrono de módulo 10 com seletoras crescentes/ decrescente



Os esquemáticos em bloco e normal constam ao fim deste documento.

2.2.3 Contador síncrono de módulo 10

Na mesma atividade foi solicitada a implementação de um contador síncrono de módulo 10 que desenvolvesse a mesma funcionalidade, ou seja, realizasse uma contagem

normal. Do mesmo modo que a atividade anterior, esperava-se que este fosse capaz de realizar a contagem de modo crescente e decrescente, selecionado por meio de um circuito seletor. A diferença deste para o implementado anteriormente decorre somente do fato de que um único clock deve entrar em todos os flip flops ao mesmo tempo, conforme pode ser observado abaixo.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity contador is
   Port (  clear: in std_logic;
         clock: in std_logic;
         Scresc: out std_logic_vector (3 downto 0);
         Sdecresc: out std_logic_vector (3 downto 0));
end contador;

architecture behavioral of contador is

signal tempc: std_logic_vector (3 downto 0) := "0000";
signal tempd: std_logic_vector (3 downto 0) := "1001";

begin

process(CLEAR, clock)
begin
   if(CLEAR = '1') then
      tempc <= "0000";
      tempd <= "1001";
   elsif(clock'event and clock = '1') then
      tempc <= tempc + 1;
      tempd <= tempd - 1;
     Scresc <= tempc;
     Sdecresc <= tempd;
  end if;

end process;
end behavioral;
```

Esse tipo de implementação é bem simplificada, mas funciona para o resultado esperado. Assim como o contador anterior, este necessita de um divisor de clock.

```
library ieee;
use ieee.std_logic_1164.all;
```

```
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity divisorclock2 is
   Port (  clock02E: in std_logic;
         clock02S: out std_logic);
end divisorclock2;

architecture behavioral of divisorclock2 is

signal contador: std_logic_vector (28 downto 0) := (others => '0');

begin

process(clock02E)
begin
   if rising_edge(clock02E) then
      contador <= contador + 1;
   end if;
clock02S <= contador(28);
end process;

end behavioral;
```

O clear, como anteriormente tem seu comportamento expresso pelo código abaixo .

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity controlclear is
   Port (  B: in std_logic;
         D: in std_logic;
         Scount: out std_logic);
end controlclear;

architecture behavioral of controlclear is

begin

Scount <= B and D;

end behavioral;
```

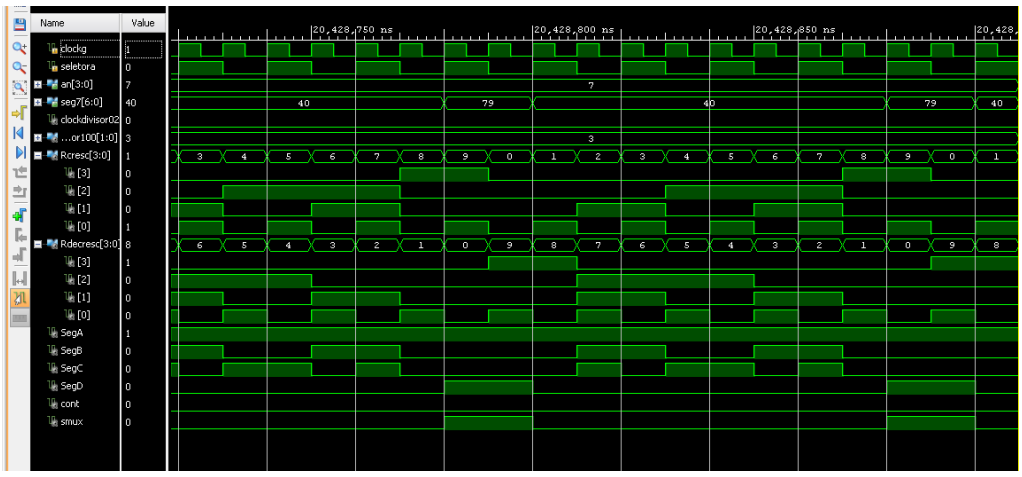
A seletora também possui lógica semelhante a demonstrada anteriormente, tal que:

```
library ieee;
```

```
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity seletor is
   Port (  sel: in std_logic;
           SC: in std_logic_vector (3 downto 0);
           SD: in std_logic_vector (3 downto 0);
           S: out std_logic_vector (3 downto 0));
end seletor;

architecture behavioral of seletor is

begin

with sel select
   S <=    SC when '1',
           SD when others;

end behavioral;
```

A ferramenta para mostrar em um display de 7 segmentos possui a mesma estrutura que a apresentada anteriormente por meio do mux, decodificador e seletor de display. As ondas oriundas da simulação deste circuito podem ser vista a seguir. Os esquemáticos encontram-se no fim deste documento.

Figura 11: Simulação de contador síncrono de módulo 10 com seletora crescente/ decrescente



2.2.4 Contador síncrono com sequência diferenciada

A terceira e última simulação diz respeito à um contador síncrono que reproduza a sequência 0,2,5,7. Sempre que for atingido um estado indesejável, o contador deve voltar para

o estado 000, permitindo também escolher qualquer um desses estágios como estágio inicial do circuito. Assim como os demais, sugeriu-se que essa contagem fosse mostrada em um display de 7 segmentos.

O inicializador dos flip-flops, segundo a equação obtida através da tabela verdade do circuito foi implementada segundo o código a seguir.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity inicializador is
   Port (  a: in std_logic;
           p: in std_logic;
           Sclear: out std_logic;
           Spreset: out std_logic);
end inicializador;

architecture behavioral of inicializador is

begin

process(p)

variable flag: std_logic;

begin
   flag := '1';
   if(p'event and p = '1') then
     flag := '0';
   end if;
   Sclear <= (not flag) nand (not a);
   Spreset <= (not flag) nand a;
end process;

end behavioral;
```

Foi necessário também realizar um controle das entradas, visto que esta era uma das propostas deste experimento. Para tanto, seguiu-se a lógica apresentada nos códigos sequenciais a seguir. Deve-se salientar que há um tratamento para cada flip flop utilizado, sendo estes 1,2 e 3 respectivamente.

```
library ieee;
use ieee.std_logic_1164.all;
```

```
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity entradas1 is
   Port (  B: in std_logic;
           C: in std_logic;
           J1: out std_logic;
           K1: out std_logic);
end entradas1;

architecture behavioral of entradas1 is

begin

J1 <= (not C) and B;
K1 <= '1';

end behavioral;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity entradas2 is
   Port (  A: in std_logic;
           C: in std_logic;
           J2: out std_logic;
           K2: out std_logic);
end entradas2;

architecture behavioral of entradas2 is

begin

J2 <= A xnor C;
K2 <= '1';

end behavioral;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity entradas3 is
   Port (  A: in std_logic;
           B: in std_logic;
```

```
        J3: out std_logic;
        K3: out std_logic);
end entradas3;

architecture behavioral of entradas3 is

begin

J3 <= B and (not A);
K3 <= (not A) or B;

end behavioral;
```

O flip-flop utilizado foi o flip-flop JK com as funcionalidades de preset e clear. Dessa forma, chegou-se à implementação que se segue :

```
library ieee;
use ieee.std_logic_1164.all;

entity flipflopJK is
   Port (  J: in std_logic;
        K: in std_logic;
        clear: in std_logic;
        preset: in std_logic;
        clock: in std_logic;
        S: out std_logic);
end flipflopJK;

architecture behavioral of flipflopJK is

begin

process(clear, preset, clock)

variable temp: std_logic := '0';

begin
   if(clear = '0') then
      S <= '0';
   elsif(preset = '0') then
      S <= '1';
   elsif(clock'event and clock = '0') then
      if(J = '0' and K = '0') then
         temp := temp;
      elsif(J = '1' and K = '1') then
```

```
            temp := not temp;
        elsif(J = '0' and K = '1') then
            temp := '0';
        else
            temp := '1';
        end if;
        S <= temp;
    end if;
end process;

end behavioral;
```

O decodificador utilizado para mostrar as informações em um display foi implementado como antes, sendo assim apresentando a seguinte lógica:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity decodificador is
    Port (  e: in std_logic_vector (2 downto 0);
            segmentos: out std_logic_vector (6 downto 0));
end decodificador;

architecture behavioral of decodificador is

begin

with e select
    segmentos <= "1000000" when "000",
            "1111001" when "001",
            "0100100" when "010",
            "0110000" when "011",
            "0011001" when "100",
            "0010010" when "101",
            "0000010" when "110",
            "1111000" when others;

end behavioral;
```

Por fim, todos os componentes adicionados foram reunidos por uma classe principal, segundo a qual:

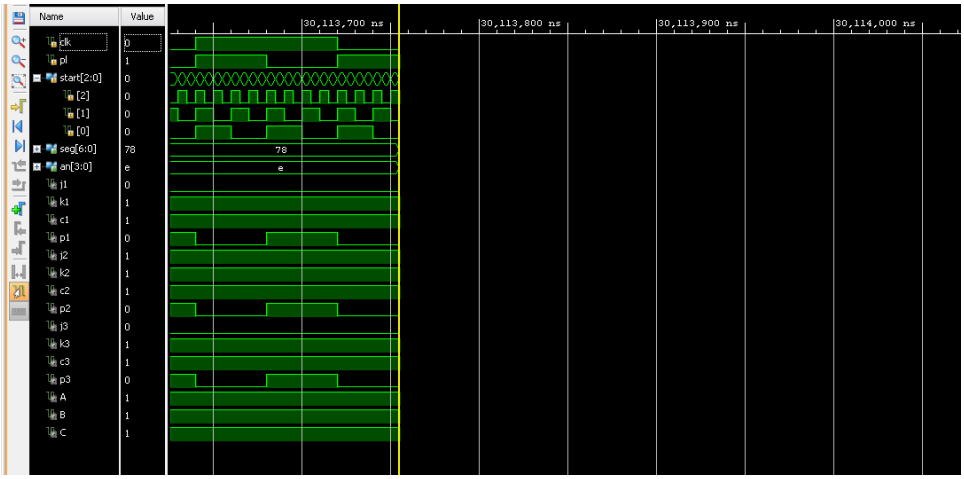
```
library ieee;
```

```
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity principal is
    Port (  clk: in std_logic;
            pl: in std_logic;
            start: in std_logic_vector (2 downto 0);
            seg: out std_logic_vector (6 downto 0);
            an: out std_logic_vector (3 downto 0));
end principal;

architecture behavioral of principal is

component inicializador
    Port (  a: in std_logic;
            p: in std_logic;
            Sclear: out std_logic;
            Spreset: out std_logic);
end component;

component entradasjkA is
    Port (  inB: in std_logic;
            inC: in std_logic;
            Ja: out std_logic;
            Ka: out std_logic);
end component;

component entradas1
    Port (  B: in std_logic;
            C: in std_logic;
            J1: out std_logic;
            K1: out std_logic);
end component;

component entradas2
    Port (  A: in std_logic;
            C: in std_logic;
            J2: out std_logic;
            K2: out std_logic);
end component;

component entradas3
    Port (  A: in std_logic;
            B: in std_logic;
            J3: out std_logic;
            K3: out std_logic);
end component;
```

```
component flipflopJK
   Port (  J: in std_logic;
           K: in std_logic;
           clear: in std_logic;
           preset: in std_logic;
           clock: in std_logic;
           S: out std_logic);
end component;

component decodificador
   Port (  e: in std_logic_vector (2 downto 0);
           segmentos: out std_logic_vector (6 downto 0));
end component;

signal j1, k1, c1, p1: std_logic;
signal j2, k2, c2, p2: std_logic;
signal j3, k3, c3, p3: std_logic;
signal A, B, C: std_logic := '0';

begin

Inicio1: inicializador port map (a => start(0), p => pl, Sclear => c1, Spreset => p1);
EntradaJK1: entradas1 port map (B => B, C => C, J1 => j1, K1 => k1);
FF1: flipflopJK port map (J => j1, K => k1, clear => c1, preset => p1, clock => clk, S =>
A);
Inicio2: inicializador port map (a => start(1), p => pl, Sclear => c2, Spreset => p2);
EntradaJK2: entradas2 port map (A => A, C => C, J2 => j2, K2 => k2);
FF2: flipflopJK port map (J => j2, K => k2, clear => c2, preset => p2, clock => clk, S =>
B);
inicio3: inicializador port map (a => start(2), p => pl, Sclear => c3, Spreset => p3);
EntradaJK3: entradas3 port map (A => A, B => B, J3 => j3, K3 => k3);
FF3: flipflopJK port map (J => j3, K => k3, clear => c3, preset => p3, clock => clk, S =>
C);
decod: decodificador port map (e(0) => A, e(1) => B, e(2) => C, segmentos => seg);

-- Utilizando apenas um display
an(0) <= '0';
an(1) <= '1';
an(2) <= '1';
an(3) <= '1';

end behavioral;
```

Dessa maneira, obteve-se as ondas a seguir:

Figura 12: Simulação de contador síncrono de ordem diferenciada



3 DISCUSSÃO

Os circuitos implementados para esse experimento envolvem conceitos relacionados a memória ou armazenamento de dados, mas mais especificamente, contadores. Pode-se considerar dessa maneira pois os contadores são formados por flip-flops, que por si só trabalham com armazenamento de dados. Entretanto, procura-se por meio desse experimento uma compreensão acerca dos contadores síncronos e assíncronos.

A primeira implementação relaciona o conceito de contadores e ordem de contagem, acerca de uma metodologia usual para realização dessa tarefa, ou seja, a contagem começa em 0 e vai sendo incrementada em 1. Dessa maneira, propôs-se a realização de um experimento em que o circuito implementado contasse em ordem crescente ou decrescente. Além disso, buscou-se possibilitar a existência de ambos em um mesmo circuito através do uso de uma seletora.

Assim como a maioria das pesquisas realizadas, há diversas maneiras de manipulação de informações, nesse sentido, os contadores são amplamente utilizados em calculadoras e computadores, por exemplo. Assim, na primeira parte deste experimento foi preciso realizar um total de 2 simulações, sendo uma relacionada a contadores assíncronos e a outra à contadores síncronos. Ambas possuindo lógicas semelhantes, entretanto, a maior diferença decorre do fato de os contadores assíncronos serem organizados de maneira que apenas o primeiro flip flop recebe o clock externo, e o clock dos demais consistem nas saídas uns dos outros.

O segundo experimento, assim como o primeiro, demonstrou uma ideia elaborada de memória, tratando-se de um contador síncrono. Para realizar tal implementação foram utilizados os mesmos recursos do primeiro, sendo assim, 4 flip flops do tipo JK, com as funcionalidades de preset e clear, divisores de frequência, seletores e decodificadores. Isso por que, esperava-se demonstrar no display a dinâmica dos números gerados.

Essa implementação no geral não apresentou grandes problemas, foi possível observar pelas respostas dos sinais na implementação que o circuito de fato obteve a resposta esperada . Para realizar a demonstração dinâmica dessa mudança de posições procurou-se demonstrar utilizando um display de 7 segmentos que dada a saída de cada flip-flop utilizava um decodificador para mostrar no display. Esse código foi bastante simplificado devido ao fato de não haver interesse em ligar mais que um display fornecido pela basys3.

No terceiro e último experimento apresentou-se a ideia de um contador, utilizando flip flops, que realizasse uma contagem diferente da usual, segundo uma ordem pré-estabelecida. Dessa forma, procurou-se contruir classes para o flip flop a ser utilizado, nesse caso o JK . As entradas de preset e clear foram estabelecidas conforme a relação com uma tabela verdade, obtendo-se equações válidas de controle para a inicialização das entradas, por exemplo, notou-se por meio das ondas obtidas que alcançou-se os resultados esperados.

Para cada um dos 3 experimentos foi realizada síntese, implementação, simulação e análise prévia dos resultados, de modo que estes foram satisfatórios em grande parte deles. Esse tipo de observação trouxe uma noção da capacidade de implementação com as ferramentas estudadas, assim como a respeito das possibilidades de uso das mesmas. Isso porque, a união de toda a lógica apresentada até o presente momento da disciplina retém um elevado poder computacional.

4 CONCLUSÕES

Diante do que foi exposto, nota-se que as atividades solicitadas no experimento foram executadas com resultados satisfatórios, ou seja, foi possível simular o circuito responsável pela implementação dos contadores, com diferentes modos de contagem, no software Vivado, logo, o objetivo do trabalho foi alcançado, contudo, sob muitas limitações devido às lógicas envolvidas na inicialização dos flip flops, além disso, o conteúdo relativo a contadores foi desafiador devido ao fato da prática não acompanhar as aulas teóricas. Além disso, é preciso citar o mau funcionamento de alguns computadores do laboratório, a demora na compilação dos códigos, por exemplo. Seria mais proveitoso turmas menores para maior aproveitamento da disciplina.

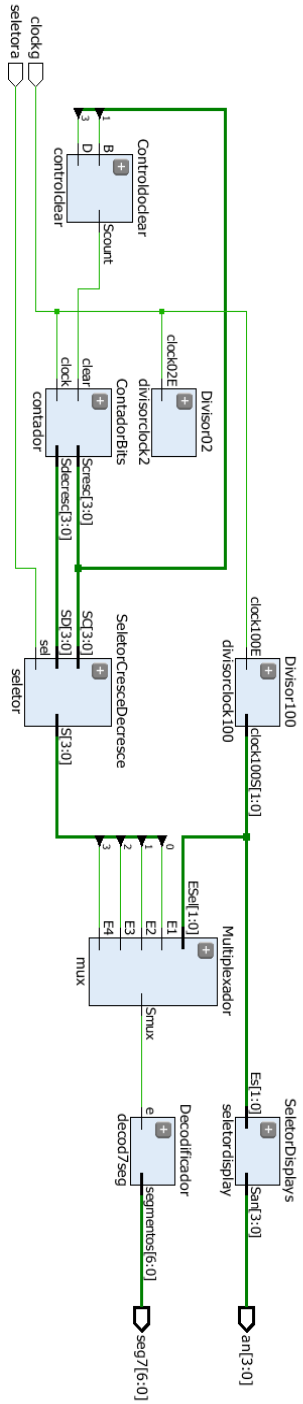
REFERÊNCIAS BIBLIOGRÁFICAS

[1] BAÚ, N. **Apostila de Eletrônica Digital – Contadores**, CEFET/SC, 1999.

[2] FLOYD, Thomas. Portas lógicas. In _____. **Sistemas digitais: fundamentos e aplicações**. Bookman Editora, 2009. Cap. 8, p. 442-480.

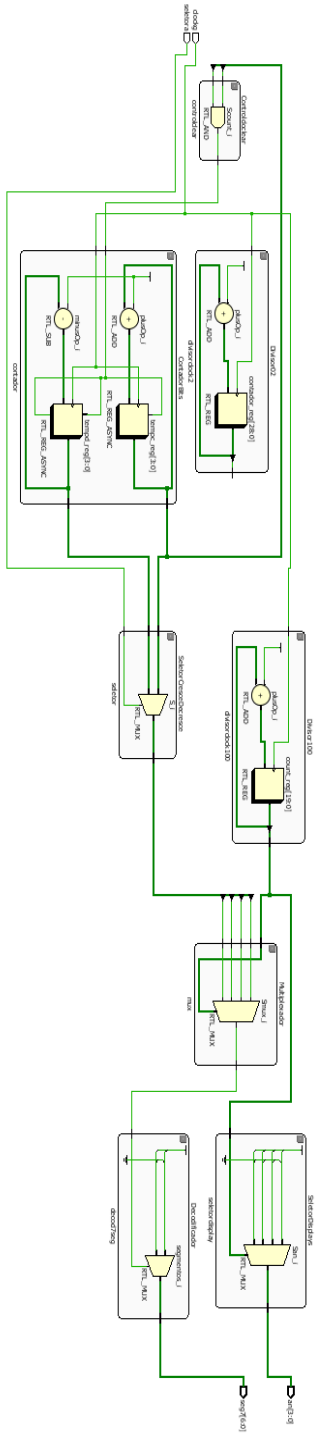
DIAGRAMAS ESQUEMÁTICOS

Figura 13: Diagrama em bloco de contador Assíncrono de módulo 10 com seletora



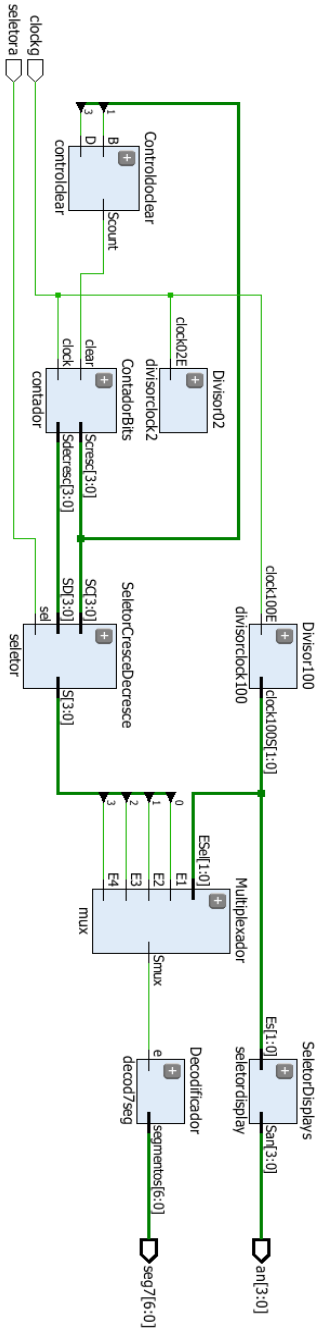
UnB - FGA	Prancha 1	Versão	Data	Elaborador por:	Adrianne Alves da Silva
LAB PED 1	contadores síncronos / assíncronos	1.0	02/06/2017	Revisado por:	Ana Paula Gomes de Matos

Figura 14: Diagrama aberto de contador Assíncrono de módulo 10 com seletora



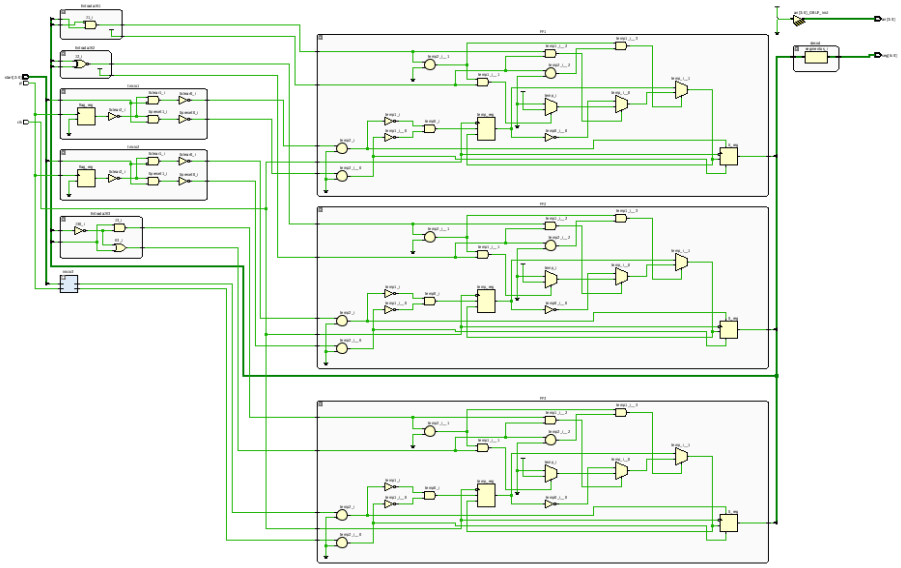
UnB - FGA	Prancha 1	Versão	Data	Elaborador por:	Adrianne Alves da Silva
LAB PED 1	contadores síncronos / assíncronos	1.0	02/06/2017	**Revisado por:**	Ana Paula Gomes de Matos

Figura 15: Diagrama em bloco de contador síncrono de módulo 10 com seletora



UnB - FGA	Prancha 1	Versão	Data	Elaborador por:	Adrianne Alves da Silva
LAB PED 1	contadores síncronos / assíncronos	1.0	12/06/2017	**Revisado por:**	Ana Paula Gomes de Matos

Figura 16: Diagrama de contador síncrono com contagem diferenciada



UnB - FGA	Prancha 1	Versão	Data	Elaborador por:	Adrianne Alves da Silva
LAB PED 1	contadores síncronos / assíncronos	1.0	02/06/2017	**Revisado por:**	Ana Paula Gomes de Matos