Projeto de ADPLL para sincronia de sinais usando FPGA

Wellington W. F. Sarmento, Paulo de Tarso C. Pequeno e Ricardo C. Ciarlini

Resumo

- Conceitos básicos
- Visão Geral
- Problema
- Projeto
- Implementação VHDL e Testes

Visão geral

- Um *Phase Locked Loop*(PLL) é um sistema de controle de circuito fechado (*closed-loop*) que mantém um sinal gerado com a mesma fase de um sinal de referência

- Tem usos em Telecomunicações, transmissões em linhas cabeadas ou não, controle de Jitter.

- Um PLL pode ser totalmente analógico, parcialmente digital (Digital PLL ou DPLL) ou totalmente digital (All-Digital PLL ou ADPLL)

- **Presente trabalho apresentará um ADPLL descrito em VHDL e implementado no FGPA Altera Cyclone V**

Problema

- Criar um PLL que possa recuperar a freqüência de rede de uma planta elétrica de alta potência a fim de manter o sincronismo entre o sinal que chega e o sinal distribuído na rede elétrica.

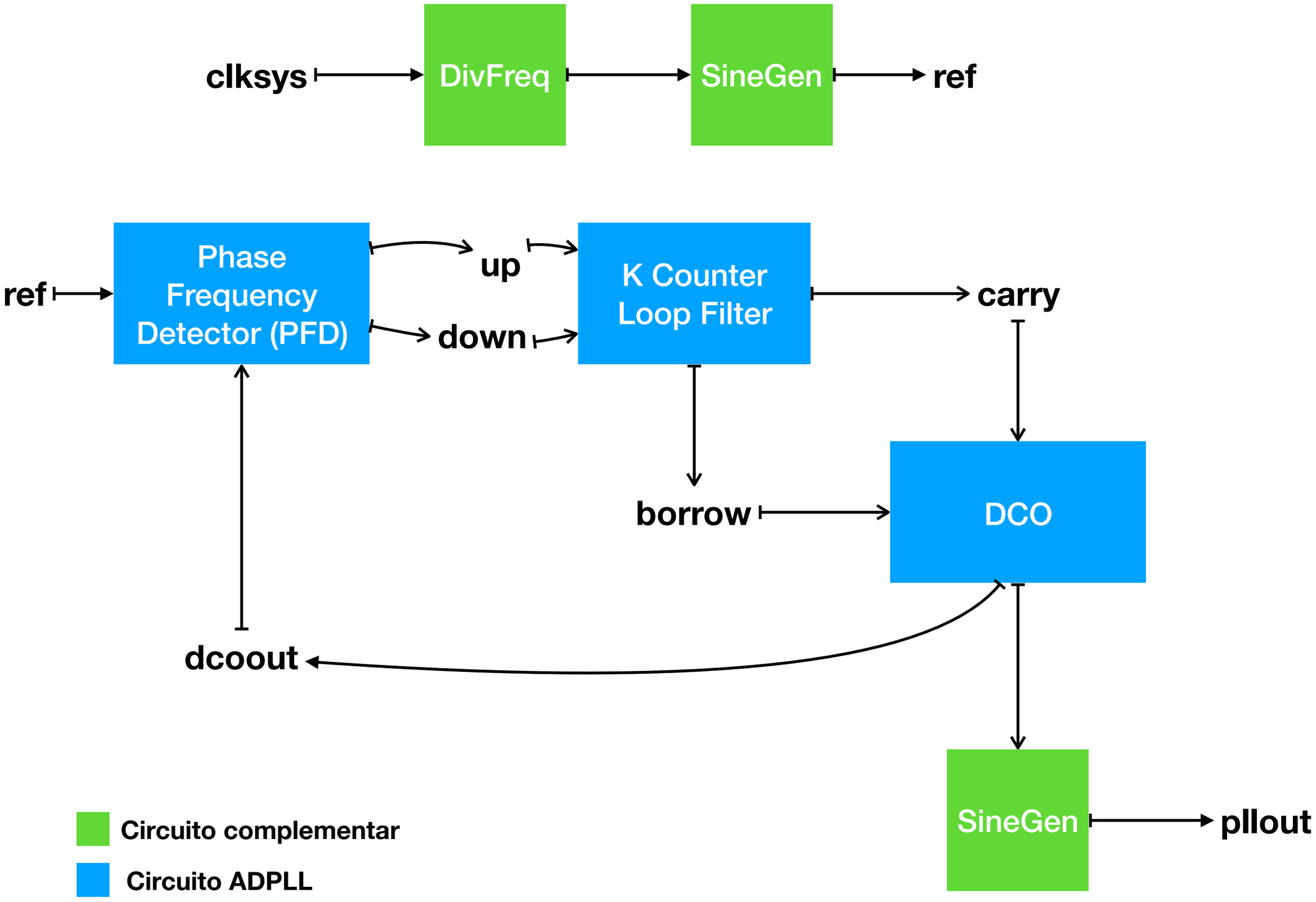
- O sinal de referência do PLL será de 60 Hz ou uma de suas harmônicas. O sinal de saída do PLL deverá ser de 60Hz em fase com o sinal de referência

Problema

- Como o kit de desenvolvimento não possui DACs, a onda senoidal de entrada deve ser *discretizada*, armazenada em tabela e usado em um contador a fim de gerar um sinal de saída com comportamento senoidal

Projeto

Diagrama de Blocos


clksys
DivFreq
SineGen
ref
ref
Phase
Frequency
Detector (PFD)
up
down
K Counter
Loop Filter
carry
borrow
DCO
dcoout
SineGen
pllout
Circuito complementar
Circuito ADPLL

Divisor de Frequência (DivFreq)

- Contador responsável por dividir a frequência fornecida ao sistema

- No caso, a frequência de entrada deste circuito será de 50 MHz

- A saída deste circuito deverá ser de 60 HZ ou 120Hz para a execução dos testes a serem realizados

Sine Generator(SineGen)

- Foram capturadas 135 amostras de seno armazenada em um vetor e utilizadas para imprimir um comportamento senoidal ao sinal que alimenta o sistema, bem como o sinal de saída do sistema
- A saída deste circuito é um barramento de 8 bits
- A entrada deste circuito é fornecida pelo DivFreq ou pelo DCO

Phase Frequency Detector (PFD)

- Como detector de fase do PLL foi escolhido o circuito Phase Frequency Detector baseado em Flip-flops, proposto por [1] e sugerido por [2] e [4] para implementação de um ADPLL

- Sua vantagem frente ao circuito baseado em porta XOR é a possibilidade de detectar mudanças de fase tanto na subida do sinal quanto na descida

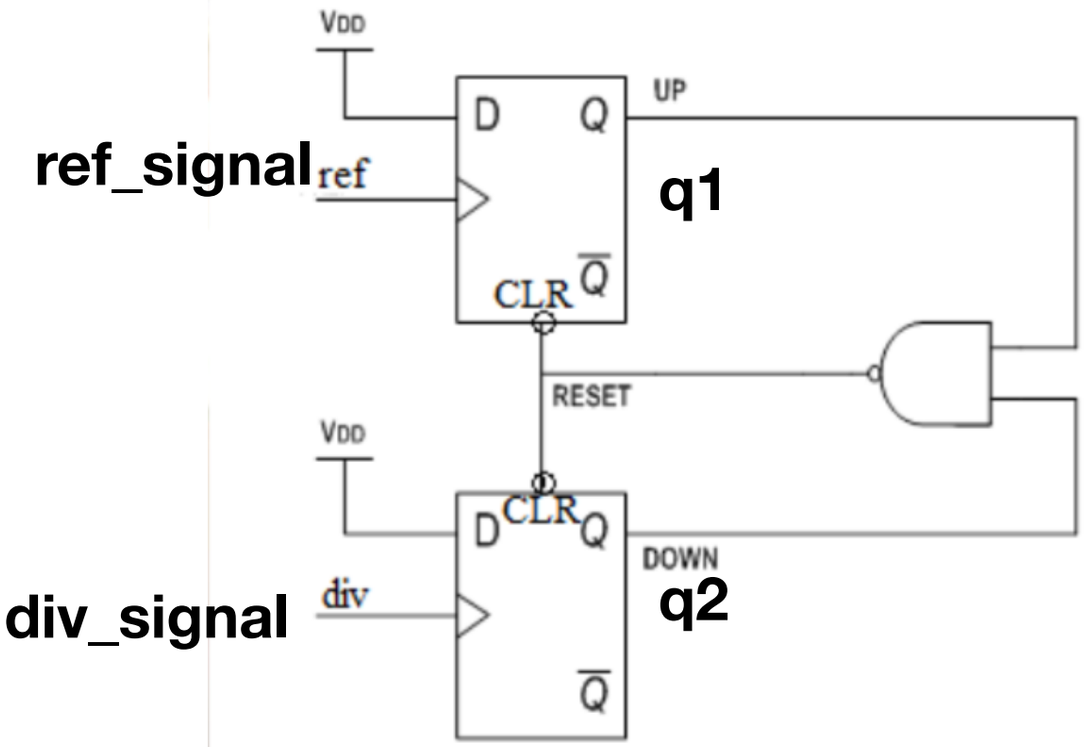
Comportamento do PFD

- Se a borda de subida de **ref** estiver adiantada em relação a borda de subida de **dcoout**, o sinal **up** será posto em "1" e **down** vai para "0".

- Se a borda de subida de **dcoout** estiver adiantada em relação a borda de subida de **ref**, o sinal **down** será posto em "1" e **up** vai para "0".

- Os sinais **up** e **down** serão iguais a 0 se as fases de **ref** e **div** forem iguais.

Phase Frequency Detector



Circuito do PFD

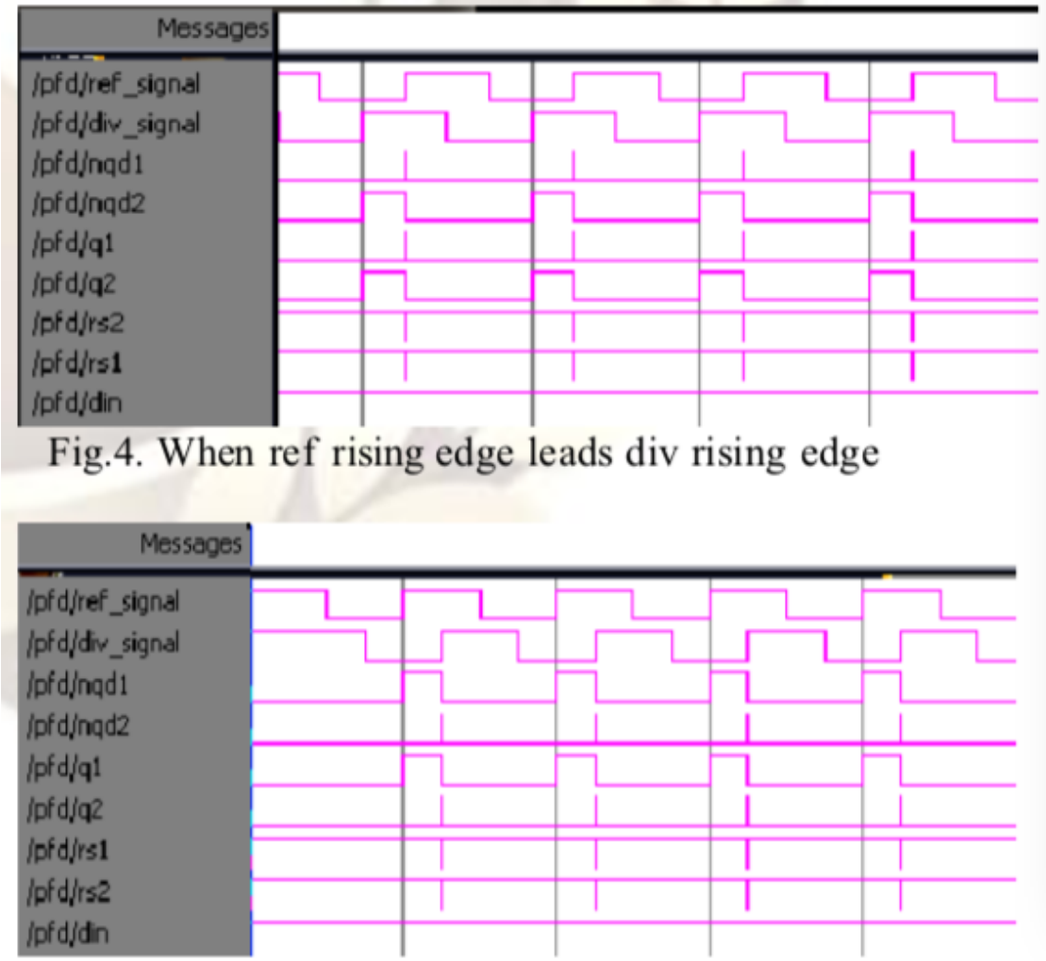


Fig.4. When ref rising edge leads div rising edge

Fig.5.When ref rising edge div rising edge

Teste efetuado por [4]

K-counter Loop Filter

- Circuito responsável pelo controle do Oscilador Controlado Digitalmente (DCO)
- Utilizado em PLLs para aplicação em Telecomunicações
- Sugerido por [2] e [4] para implementação de um ADPLL

K-counter Loop Filter

- Formado por dois contadores crescentes (UP e DOWN)

- O clock dos contadores é dado por M vezes Fc

- Os valores típicos de M são: 8,16,32…

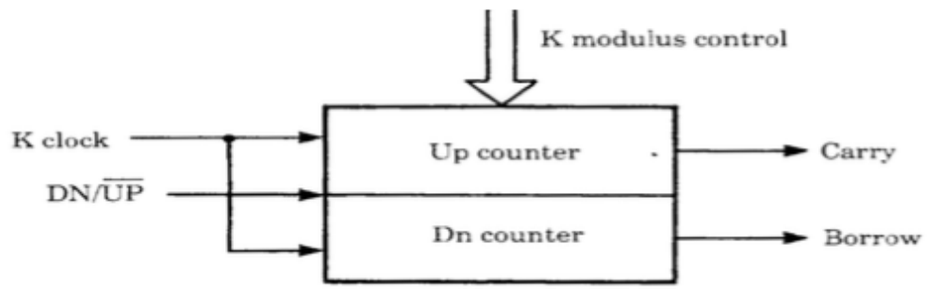


Diagrama de Blocos. Fonte: [2]

K-counter Loop Filter

- O contador tem um range de 0 à K-1
- O Down é habilitado quando DN/˜UP está em 1 e UP é habilitado quando DN/˜UP está em 0
- Quando a contagem excede K-1 ambos os contadores são resetados
- Quando Down conta valor maior ou igual a K/2, o “Borrow” vai para 1
- Quando Up conta valor maior ou igual a K/2, “Carry” vai para 1
- O sinal “Carry” é dado pelo MSB do contador UP e o “Borrow”, pelo MSB de Down

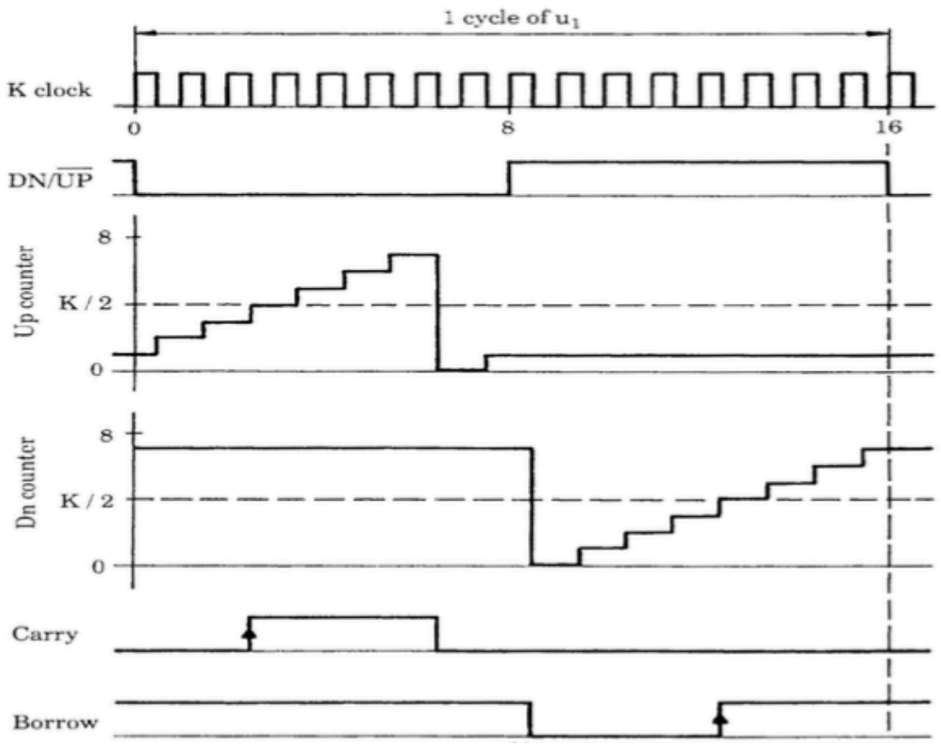


Diagrama de Tempo. Fonte: [7]

K-counter Loop Filter

- Caso o sinal DN/~UP seja desmembrado em dois (down e up), os sinais Carry e Borrow não serão produzidos consecutivamente

Comportamento de Carry com ref adiantado em relação a adpllout. Fonte: [4]

- O artigo [4] exemplifica este caso

Digital Controlled Oscillator (DCO)

- O DCO é um oscilador que modifica sua frequência de saída dependendo dos sinais enviados pelo Loop Filter

- O DCO usado é basicamente composto por um IDCounter (Increment and Decrement Counter) e um Divisor de Frequência por N

- Carry é colocado na entrada de DECR e Borrow em INCR (em [3] os sinais foram trocados, conforme pode ser visto em [2], [4] e [7])

- Se não houver sinais Carries e Borrows, IDCounter divide sua saída OUT por 2 na borda positiva de IDClock

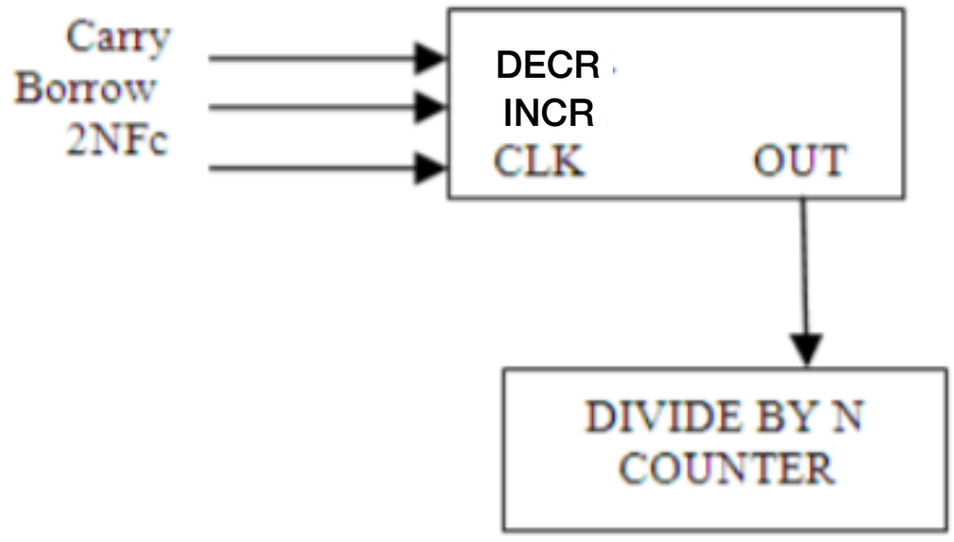


Diagrama de Blocos do DCO. Fonte: Figura modificada de [3]

Digital Controlled Oscillator (DCO)

- A saída OUT de IDCounter é usada como Clock para o Divisor de Frequência por N

- Esse divisor de frequência é usado para permitir controle da frequência que alimenta o PFD

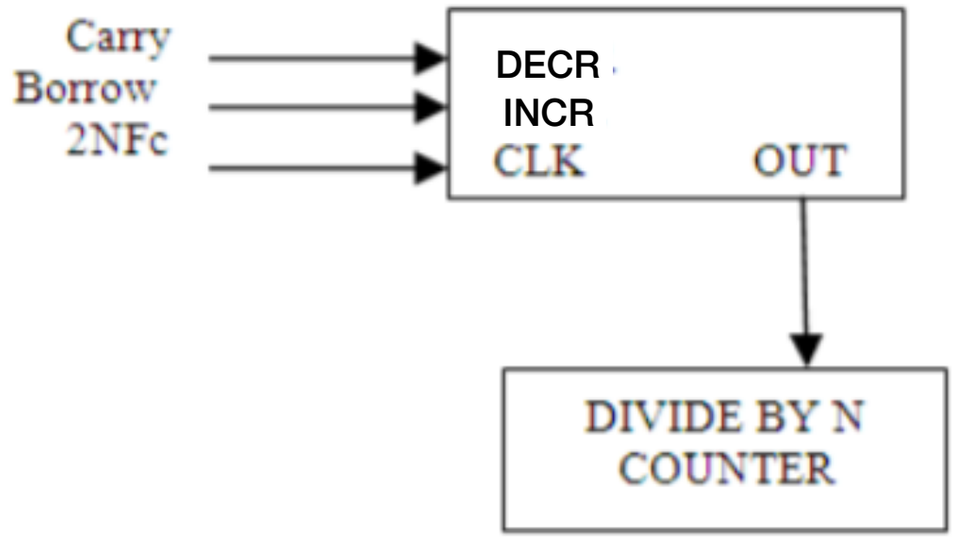


Diagrama de Blocos do DCO. Fonte: [4]

Digital Controlled Oscillator (DCO)

- Se Carry estiver presente na entrada de IDCounter, um 1/2 ciclo é adicionado a OUT

- Se Borrow estiver presente na entrada de IDCounter, um 1/2 ciclo é subtraído de OUT

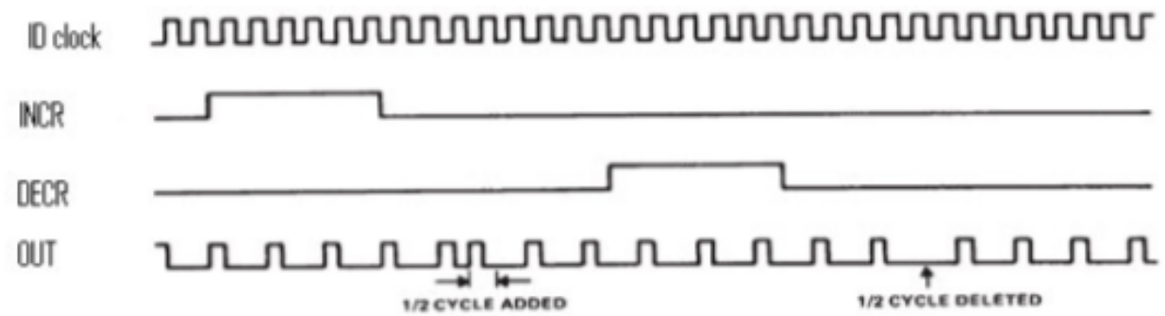


Diagrama de Tempo de INCR e DECR. Fonte: [3]

Digital Controlled Oscillator (DCO)

- A saída de IDCounter é IDout

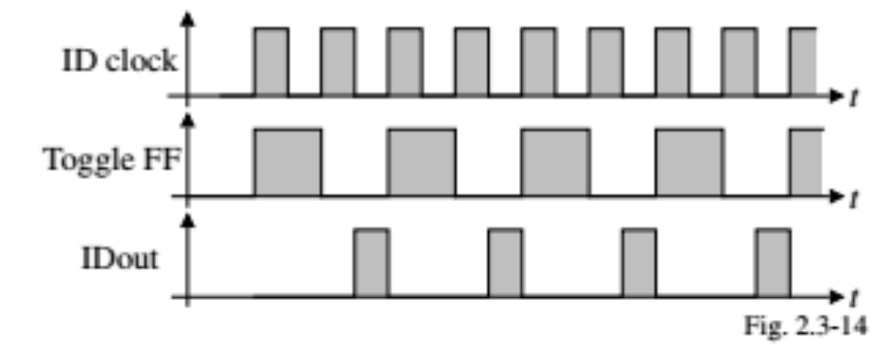
- A função lógica de IDCounter, usando um TFF pode ser vista na figura à direita

- O T-Flipflop muda seu valor de saída em cada borda positiva de IDClock se nenhum sinal Carry ou Borrow estiver presente

- Na figura pode ser visto o sinal Carry aplicado quando o T-Flipflop estiver em “0”

ID out = (NOT (ID clock) AND (NOT (toggle-FF))

Função Lógica de IDCounter. Fonte: [4]



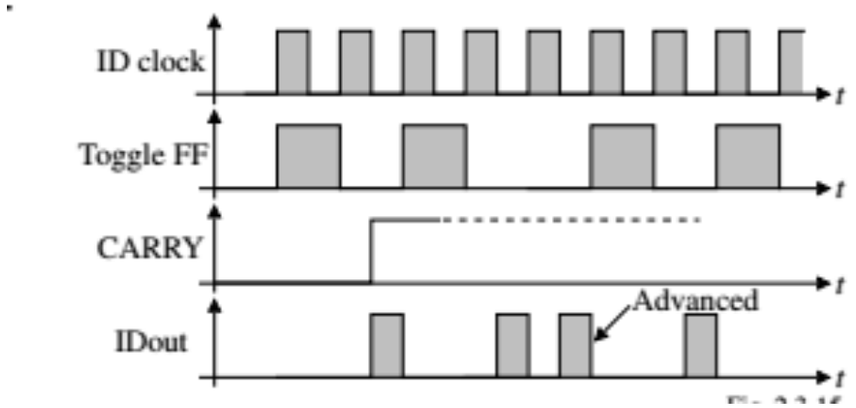


Diagrama de Tempo de Carry e Borrow. Fonte: [6]

Dados de Projeto

- Parâmetros sugeridos em [4][5] e utilizados para o projeto do ADPLL
- IDClock = 2NFc (eq. 1)
- M = 2N (eq. 2) , usado para determinar KClock=MFc (Loop Filter)
- Onde,
    - **IDClock:** Sinal de Clock do DCO
    - **Fc:** Frequência Centro
    - **N:** Usado no Divisor de Frequência por N do DCO
    - **M:** Usado para determinar a frequência de operação do K-counter

Dados de Projeto

- M= 2K (eq. 3) (2K se for um detector de fase baseado em Flip-flop ou se o detector de fase for XOR o valor é 4K)

- Onde,

- **K:** Módulo dos contadores do Filtro de Loop (K-Counter)

- **N:** Usado no Divisor de Frequência por N do DCO

- **M:** Usado para determinar a frequência de operação do K-counter

Implementação VHDL e Testes

Dados de Projeto

- **N=8** e **Fc=60Hz**
- IDClock = 2NFc = 2*8*60Hz=> **IDClock=960Hz** (Usamos no Model Sim T=1.0416ms)
- M = 2N = 2*8=16, obtendo KClock=MFc=16*60Hz=> **KClock=960Hz** (Usamos no Model Sim T=1.0416ms)
- M= 2K (2K para PFD) K=M/2=16/2 => **K=8**

Repositório do Código e estrutura de arquivos

- Código VHDL publicado sob licença GPLv3 em https://github.com/wwagner33/adpll-vhdl

- O arquivo **adpll.vhd** traz a estrutura completa do circuito testado (ADPLL, divisor de frequência, gerador de seno)

- Os arquivos **pfd.vhd**, **k_counter.vhd** e **dco.vhd** compõem o ADPLL

Todos os componentes do circuito

```vhdl
component divfreq
   port(
      clk : in std_logic;
      ref : out std_logic
   );
end component;

-- Sine Generator

component sine_wave_gen
   port(
   clk            :  in  std_logic;
   dataout        :  out natural range 0 to 255
   );
end component;

-- Phase Detector

component pfd
   port(
      a     : in std_logic;
      b     : in std_logic;
      out_up    : out std_logic;
      out_down     : out std_logic
   );
end component;
```

Componentes DivFreq, SinGen e PFD em VHDL

```vhdl
component k_counter
port (
   in_up:       in  std_logic;
   in_down:     in  std_logic;
   -- up-down:     in    std_logic;
   kclock:  in  std_logic; -- M multiple of Fo
   carry:   out  std_logic;
   borrow:  out  std_logic);
end component;

-- Controlled Oscillator

component dco
   port(
      carry, borrow, clock :  in    std_logic;
      dco_out              :  out   std_logic

   );
end component;
```

Componentes K_Counter (LF) e DCO em VHDL

Estrutura dos componentes em VHDL: DivFreq e SinGen

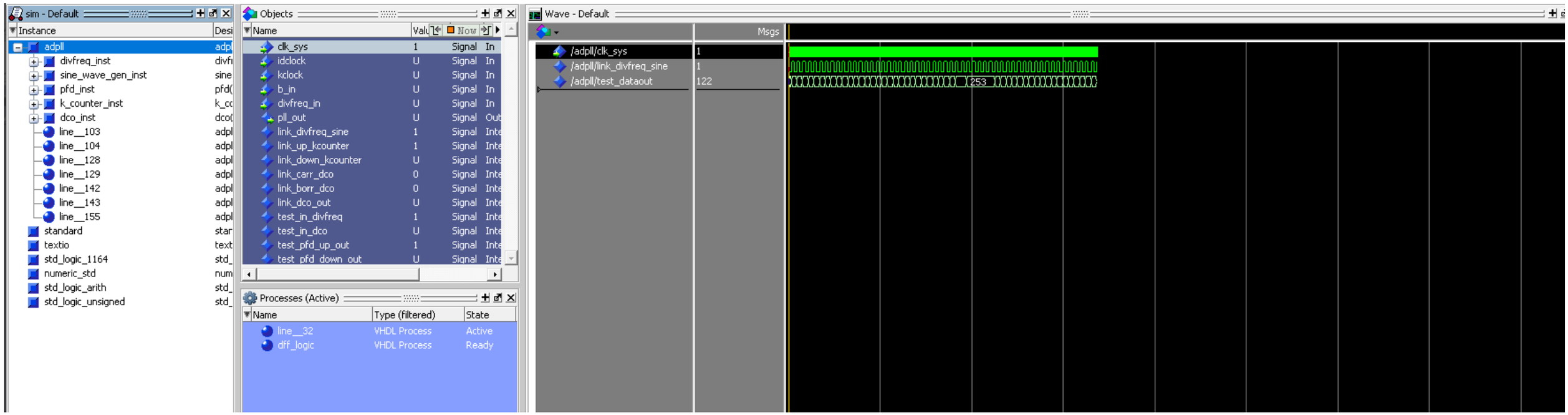
```
-- ************************************************************
-- Program: divfreq.vhd
-- Description: Frequency divider from clock of 50 MHz to 1Hz, 60/135, 1 KHz
-- Input: clk (extern clock)
-- Output: freq_1Hz
-- Author: Wellington, Paulo e Rodrigo
-- Date: 16/06/2021
-- State: No errors known
-- ************************************************************
-- Counter is 25000000 to 1 Hz (1 sample)
-- Counter is 416667 to +/- 60Hz (to 135 samples) = 3087
-- Counter is 208373 to +/- 120 Hz
-- counter is 25000 to 1 kHz
-- Counter is 1000 to 25KHz
-- Counter is 125 to 200KHz
-- Counter is 50  to 500KHz
-- Counter is 25  to 1MHz
-- Remove Counter to 25 MHz


library ieee;
use ieee.std_logic_1164.all;

entity divfreq is
   port(
      clk: in std_logic;
      ref: out std_logic);
end divfreq;

architecture divfreq_arch of divfreq is

signal count:  natural range 0 to 3087:=0;
signal ot:     std_logic:='0';

begin
   ref<=ot;

   divfreq_logic: process(clk)
   begin
      if (clk'event and clk='1') then
         count<=count+1;
         if (count=125) then
            count<=0;
            ot<=not ot;
         end if;
      end if;
   end process;


end divfreq_arch;
```

Código do Divfreq para saída de 60Hz com 135 amostras de seno

```
-- ************************************************************
-- Program: sine_wave_gen.vhd
-- Description: Sine Wave Generator
-- Input: clk - reference clock from DivFreq circuit output
-- Output: dataout -
--
-- Author: Wellington W. F. Sarmento, Paulo de T. C. Pequeno e Rodrigo Ciarlini
-- Date: 24/05/2021
-- State: tested
-- ************************************************************

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;  --try to use this library as much as possible.


entity sine_wave_gen is
port (clk            :in  std_logic;
      dataout        : out natural range 0 to 255);
end sine_wave_gen;

architecture sine_wave_gen_arch of sine_wave_gen is
signal counter          : natural range 0 to 255:=0;
type table_type is array (integer range<>) of natural;

--ROM for storing the sine values from Prática TAED 07.pdf.

signal sine             : table_type(0 to 134):=(127,132,138,144,150,156,162,167,173,178,184,189,194,199,204,↵
208,213,217,221,225, 228,232,235,238,241,243,245,247,249,250,252,252,253,253,253,253,253,252,251,250, 248,246,↵
244,242,239,236,233,230,226,223,219,214,210,206,201,196,191,186,180,175, 170,164,158,152,147,141,135,129,123,↵
117,111,105,100,94,88,83,77,72,66,61,56,51,47, 42,38,34,30,26,23,19,16,13,11,8,6,5,3,2,1,0,0,0,0,0,1,2,3,4,6,8↵
,10,13,15,18,22,25,29,33, 37,41,46,50,55,60,65,70,76,81,87,92,98,104,110,116,122);

begin

dataout<= sine(counter);
counter_logic: process(clk)
begin
--to check the rising edge of the clock signal

   if(counter = 135) then
      counter<=0;
   elsif (rising_edge(clk)) then
      counter<=counter+ 1;
   end if;
end process counter_logic;

end sine_wave_gen_arch;
```

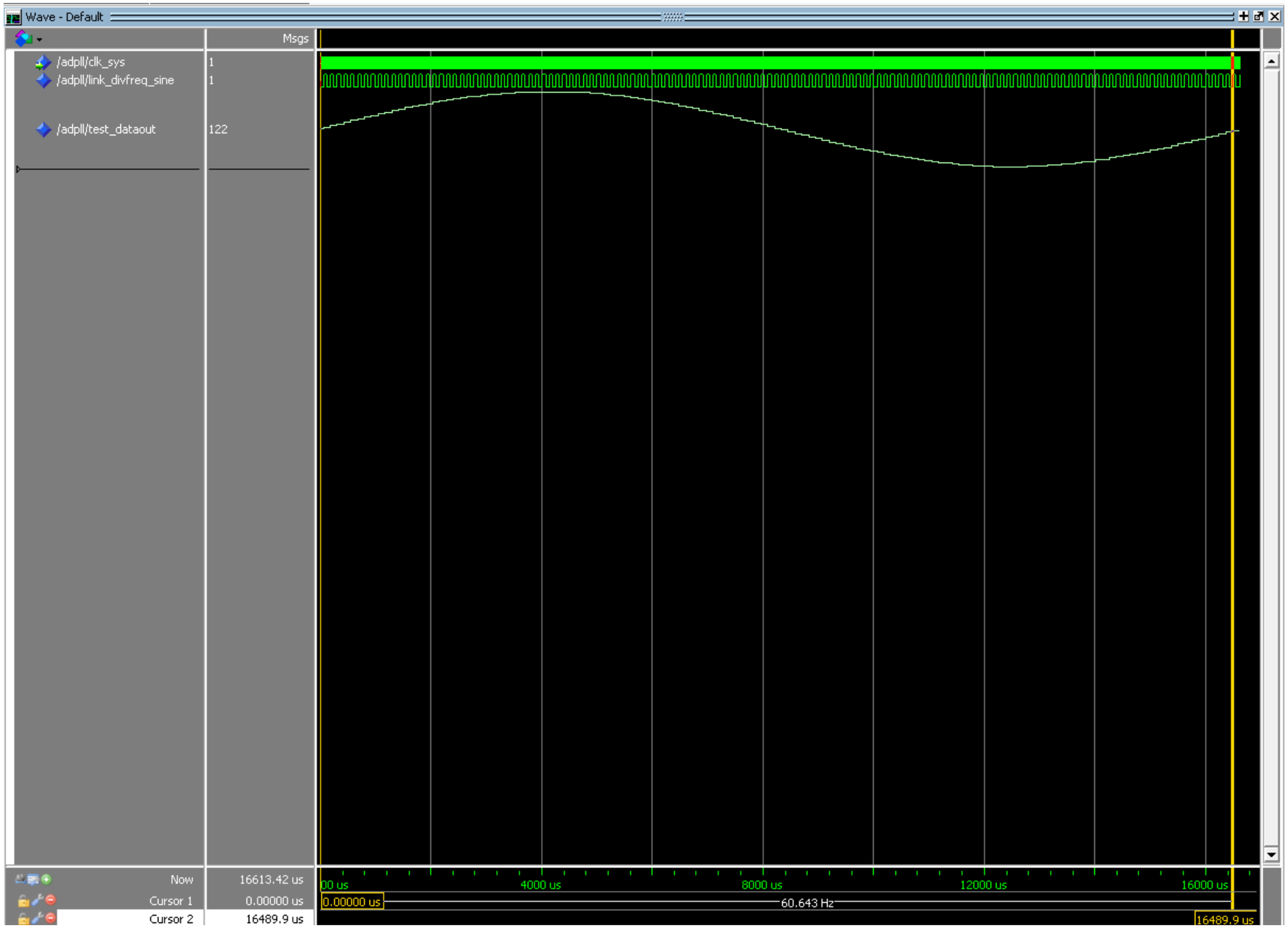
Código de criação e tabela com 135 amostras de seno e saída destes valores conforme sinal vindo de Divfreq

Simulação dos Circuitos



O circuito foi simulado com o sinal de entrada clk_sys=50MHz. O Signal link_divfreq_sine mostra o sinal de saída de Divfreq e entrada em SineGen. O Signal test_dataout mostra a saída de SineGen (sine_wave_gen_inst)

Simulação dos Circuitos



Signal test_dataout não formato Analógico no ModelSim

Estrutura dos componentes em VHDL: PFD

```
-- ************************************************************
-- Program: pfd.vhd
-- Description: Phase and frequency Detector proposed by Behzad Razavi
--    Paper: Design of Monolithic Phase-Locked Loops and Clock Recovery Circuits - a tutorial,1996 (IEEExplore)
-- Input: a, b (input signals)
-- Output: up,down. Behavior:
--    If freq(b) = freq(a), and Phase(a)=Phase(b), up=0,down=0
--    If freq(a) > freq(b), up=1, down=0
--    If freq(b) > freq(a), up=0,down=1
--    If freq(a)=freq(b), up=1 or down=1 with width pulse equal to Phase(b)-Phase(a)
-- Author: Wellington W. F. Sarmento, Paulo de T. C. Pequeno e Rodrigo Ciarlini
-- Date: 24/06/2021
-- State: No known errors
-- ************************************************************
library ieee;
use ieee.std_logic_1164.all;


entity pfd is
   port (
      a :          in  std_logic;
      b :          in  std_logic;
      out_up :     out  std_logic;
      out_down :   out  std_logic);
END pfd;

architecture pfd_arch of pfd is
-- create a component D_FF(d_ff.vhd)
   component d_ff
      port(clk     : in std_logic;
           d       : in std_logic;
           rst     : in std_logic;
           q       : out std_logic;
           nq      : out std_logic);
   end component;

   signal   srst,sup,sdown,sd       :  std_logic;
   constant set_signal     :  std_logic:='1'; -- The D input must be '1' because is necessery to PFD proposed ↵
by Rizavi
```

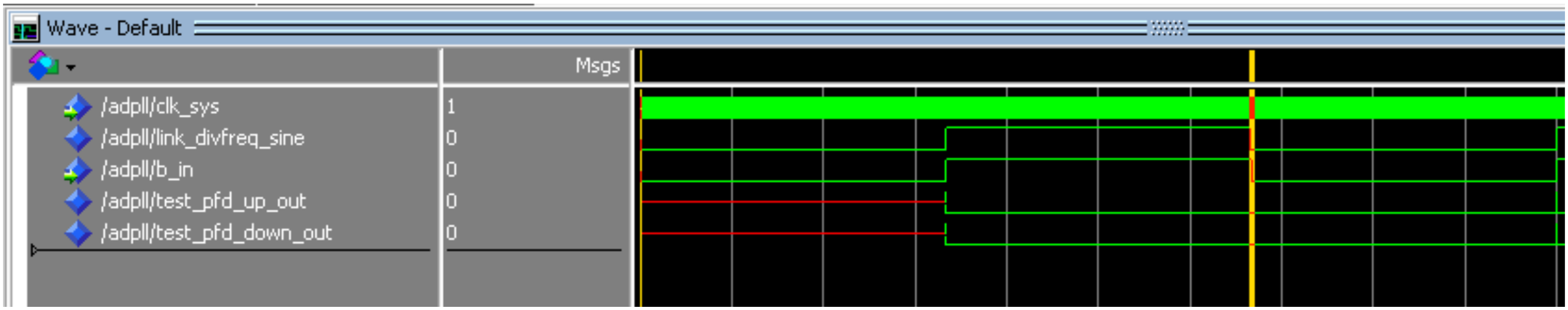
Trecho do código do PFD

Estrutura dos componentes em VHDL: PFD

```
begin
    out_up<=sup;
    out_down<=sdown;
    sd<=set_signal;

-- use component d_ff and a AND port to create the Phase and Frequency Detector

    -- map ports of component to signals
    d_ff_1_map_ports : d_ff
    port map(clk => A,
             d => sd,
             rst => srst,
             q => sup);

    -- both output's D-Flip Flops put in AND port. up and down set to zero,
    -- but the delay between A and B signal generate a width pulse equal to phases differences.
    srst <= sup nand sdown;

    -- map ports of component to signals
    d_ff_2_map_ports : d_ff
    port map(clk => B,
             d => sd,
             rst => srst,
             q => sdown);


END pfd_arch;
```

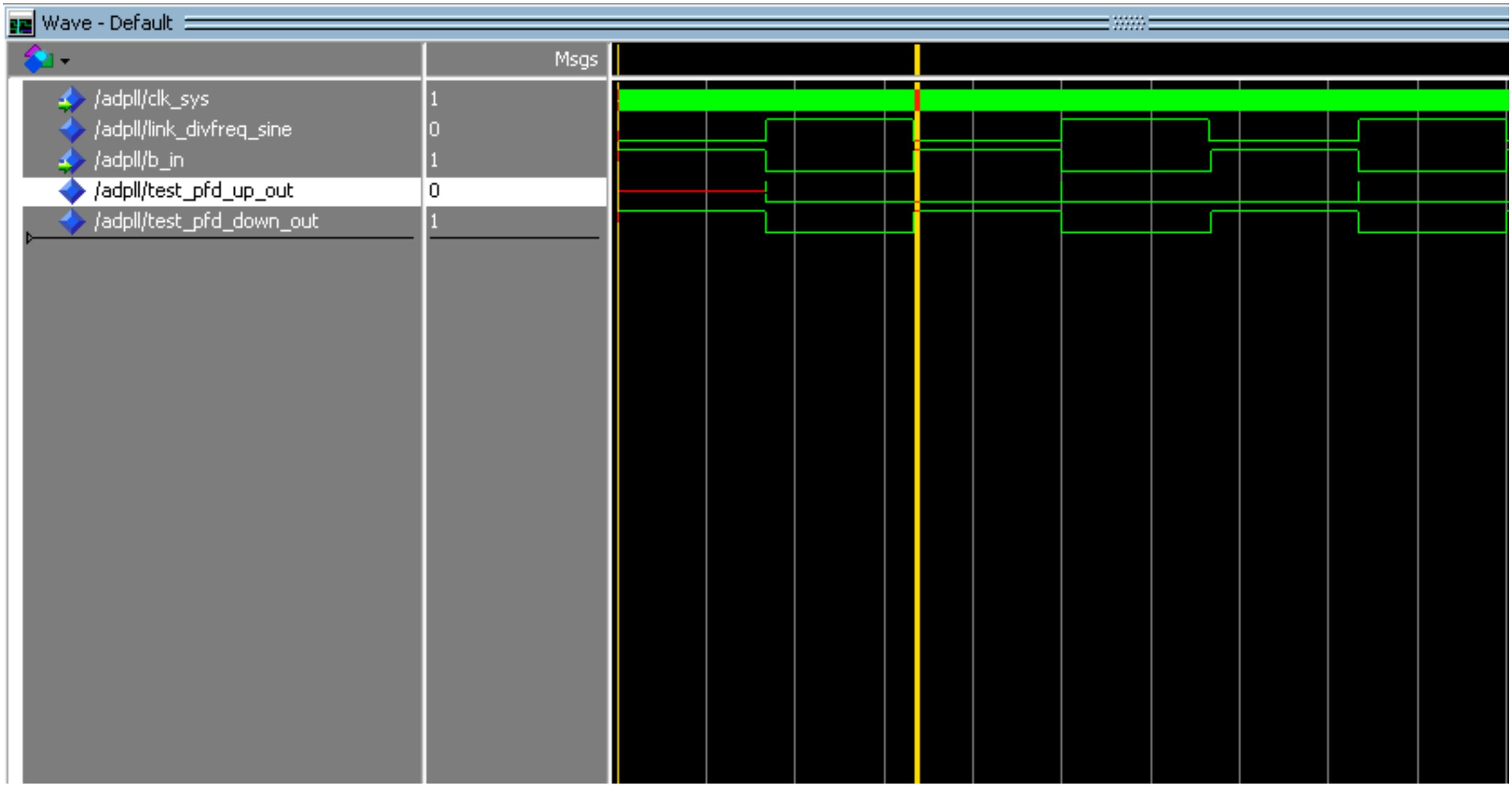
Código do circuito PFD formado por dois Flipflops D e uma porta NAND.

Simulação dos Circuitos



Teste do circuito de PFD com o DivFreq. Os sinais de entrada de PFD são link_divfreq_sine e b_in. O primeiro é a saída de DivFreq e está com 60 Hz, o segundo seria a saída do ADPLL e foi simulada com um sinal de 60 Hz. As duas entradas do PFD estão em fase.

Simulação dos Circuitos



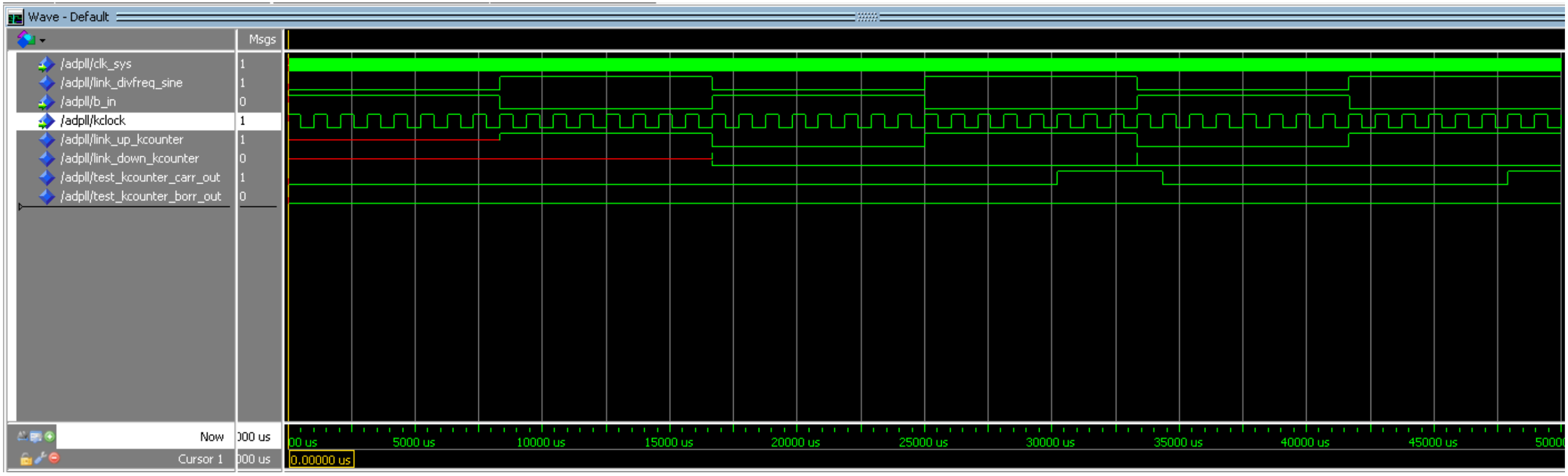
Teste do circuito de PFD com o DivFreq. Os sinais de entrada de PFD são link_divfreq_sine e b_in. O primeiro é a saída de DivFreq e está com 60 Hz, o segundo seria a saída do ADPLL e foi simulada com um sinal de 60 Hz. As duas entradas do PFD estão fora fase.

Estrutura dos componentes em VHDL: K_Counter

```vhdl
-- ******************************************************

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;


entity k_counter is
    port (
        in_up:      in  std_logic;
        in_down:    in  std_logic;
        kclock:  in  std_logic;
        carry:   out  std_logic;
        borrow:  out  std_logic);
end k_counter;

architecture k_counter_arch of k_counter is
    constant k: natural:=8;--16
    signal sup,sdown: std_logic;
    signal kcountup,kcountdown : natural:=0;
    signal otup,otdown: std_logic:='0';

begin
    carry<=otup;
    borrow<=otdown;
    sup<=in_up;
    sdown<=in_down;

    count_up: process(kclock)
        variable msb: std_logic_vector(3 downto 0);
    begin
          if (sdown='0' and sup='1') then
             kcountup<=kcountup+1;
             if (kcountup>(k-1)) then
                kcountup<=0;
             end if;
          end if;
          msb:=std_logic_vector(to_unsigned(kcountup, msb'length));
          otup<=msb(3);

    end process count_up;

    count_down: process(kclock)
        variable msb: std_logic_vector(3 downto 0);
    begin
          if (sdown='1' and sup='0') then
             kcountdown<=kcountdown+1;
             if (kcountdown>(k-1)) then
                kcountdown<=0;
             end if;
          end if;
          msb:=std_logic_vector(to_unsigned(kcountdown, msb'length));
          otdown<=msb(3);
    end process count_down;

end k_counter_arch;
```

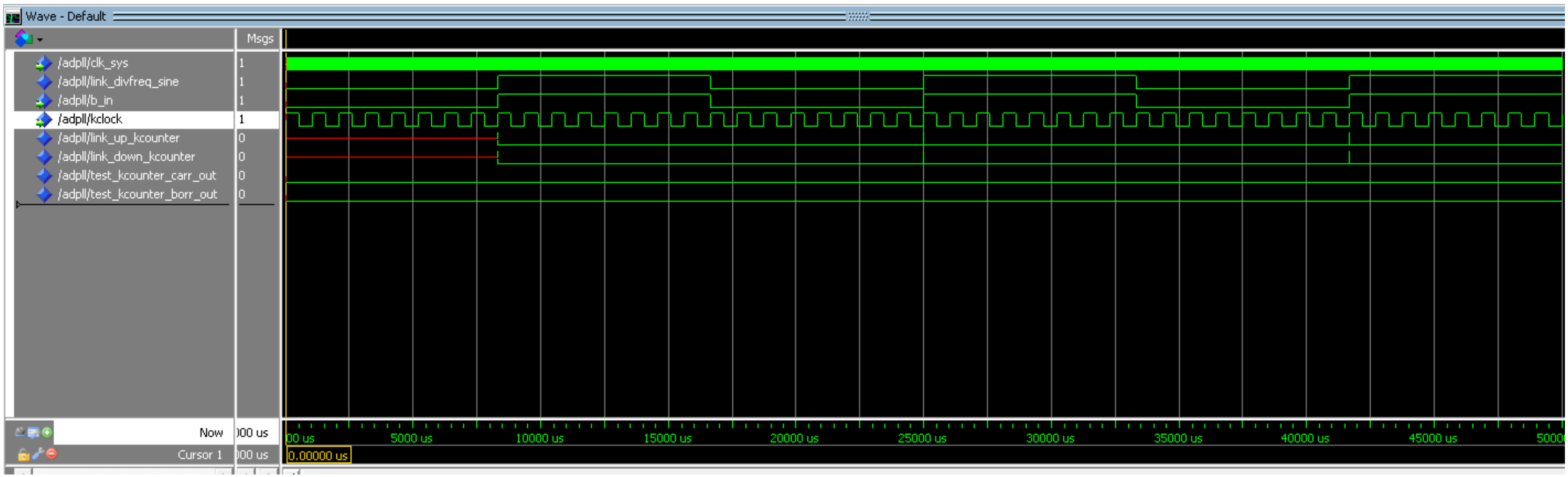
Código do K_Counter

Simulação dos Circuitos



Sinais de entrada do sistemas estão fora de fase, sendo sinal ref 180o defasado em relação ao b_in (que simula o sinal do DCO). Borrow foi ativado para compensar a defasagem.

Simulação dos Circuitos



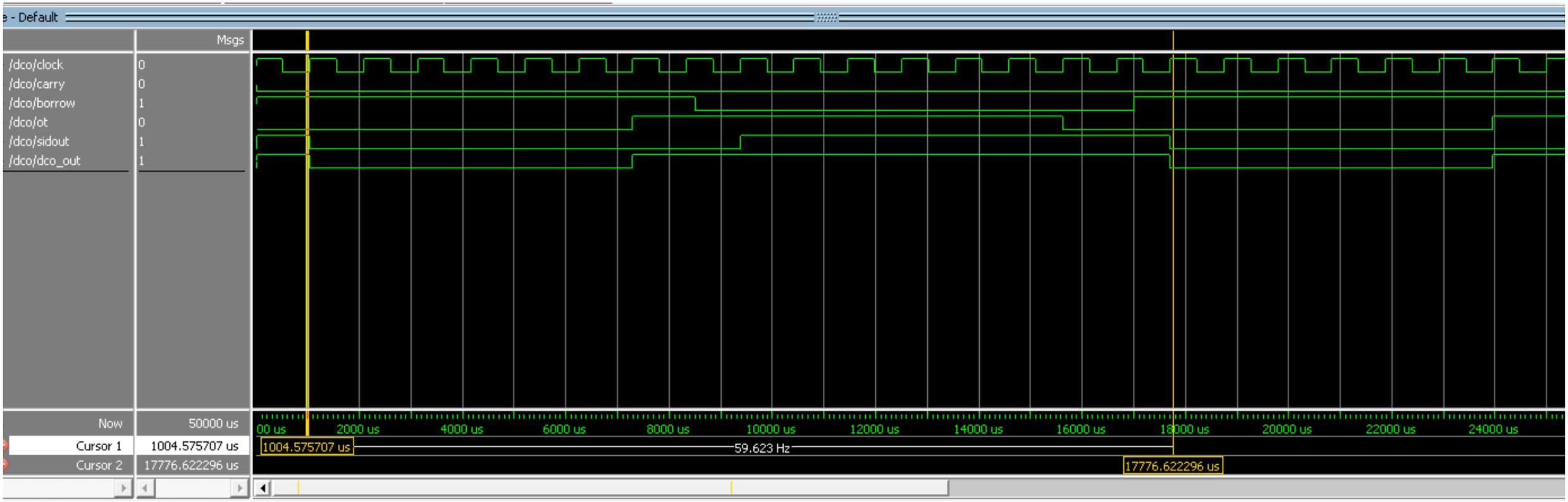
Sinais de entrada do sistemas estão em fase, ref está em fase com b_in (que simula o sinal do DCO). Borrow e Carry vão para zero.

Estrutura dos componentes em VHDL: DCO

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

entity dco is
                port (
                     carry, borrow, clock    :  in std_logic;
                     dco_out                 :  out std_logic
                );
end dco;


architecture dco_arch of dco is

   constant n: natural:=8;
   signal soutincr,soutdecr,sidoutincr, sidoutdecr,sidout,sfcout: std_logic:='0';
   signal sborrow,scarry: std_logic:='0';

   signal countincr,countfc : natural:=0;
   signal countdecr : natural:=8;

begin

sborrow<=borrow;
scarry<=carry;

incr: process(sborrow)
begin

       if (sborrow='1') then
          soutincr<= '1';
       else
          soutincr<= '0';
       end if;


end process incr;


decr: process(scarry)
begin

       if (scarry='1') then
          soutdecr<='1';
       else
          soutdecr<='0';
       end if;

end process decr;


counterincr: process(clock)

begin
   if (clock'event AND clock='1') then

       if (soutincr='1') then
```

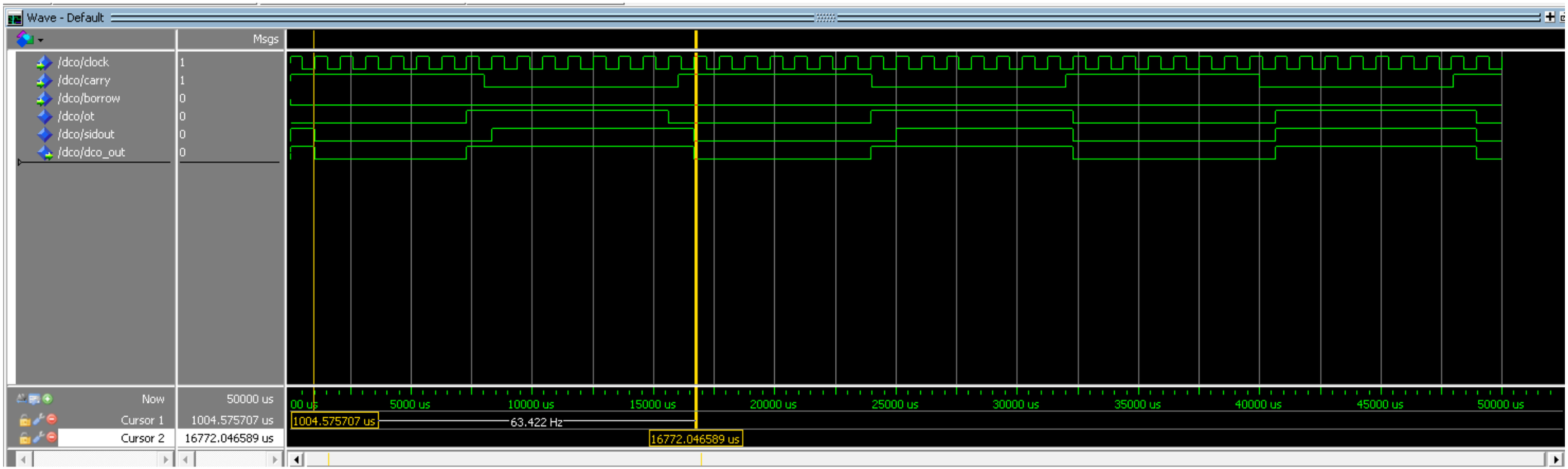
Trecho de código do DCO baseado nas propostas de [2],[4] e [6]

Simulação dos Circuitos



Sinais de entrada do sistemas estão fora de fase. A rede está atrasado em relação ao sinal gerado por DCO. Borrow foi ativado para compensar a defasagem.

Simulação dos Circuitos



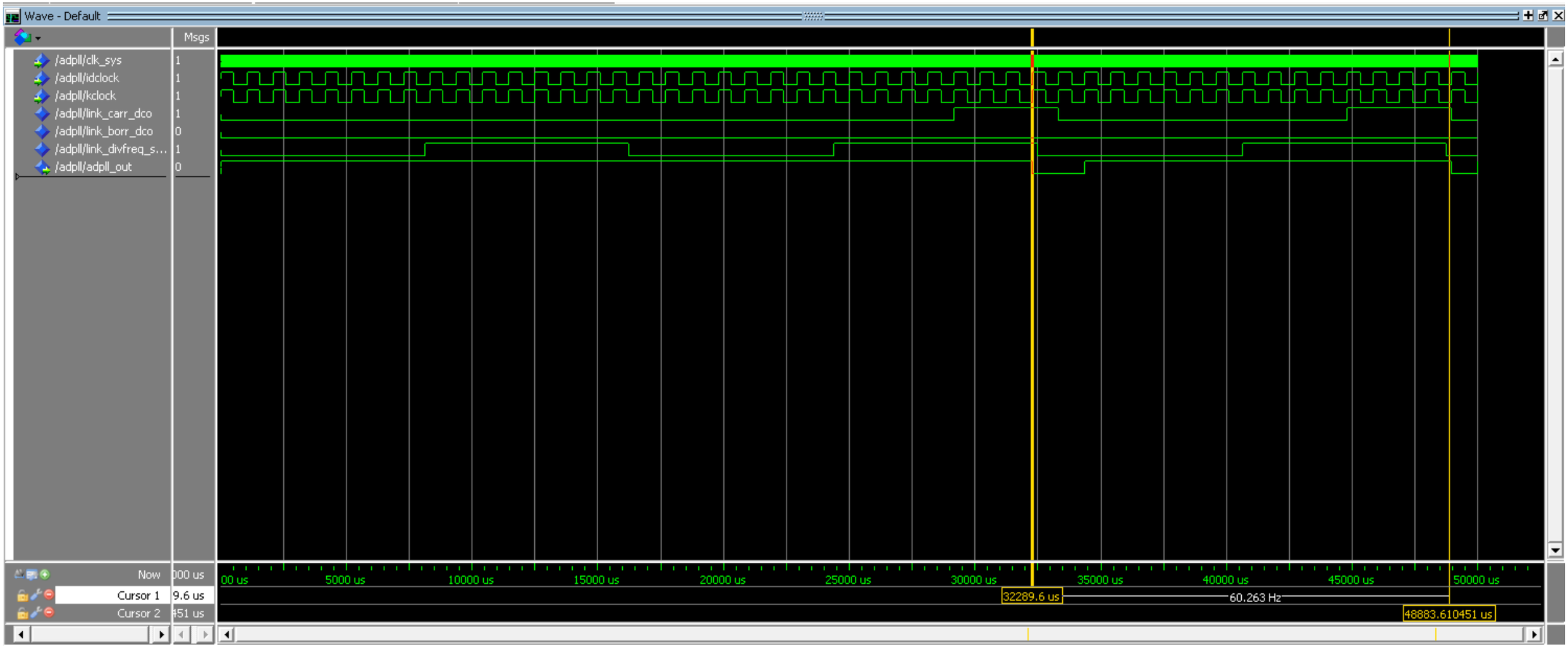
Sinais de entrada do sistemas estão fora de fase. A rede está adiantada em relação ao sinal de DCO. Carry foi ativado para compensar a defasagem.

Estrutura dos componentes em VHDL: Completo

```
   signal   link_divfreq_sine,link_up_kcounter,link_down_kcounter,link_carr_dco,link_borr_dco,link_dco_out :
std_logic;  --sdownup
   signal test_in_divfreq,test_pfd_up_out,test_pfd_down_out,test_kcounter_carr_out,test_kcounter_borr_out:
std_logic; -- sinais para proposito de teste
   signal test_dataout        :  natural range 0 to 255;  -- sinais para proposito de teste


begin

   adpll_out <= link_dco_out;

   -- Frequency divider to create smallest frequency from 50MHz frequency signal(ex.; 200KHz,60Hz,120Hz)

   divfreq_inst : divfreq
   port map(
         clk            =>    clk_sys,
         ref            =>    link_divfreq_sine
         );


   test_in_divfreq<= divfreq_in OR link_divfreq_sine; -- pin to test divfreq


   -- Sine Wave Generator

   sine_wave_gen_inst: sine_wave_gen
   port map(
      clk            => link_divfreq_sine,
      dataout        => test_dataout --link_swg_pfd,

   );


   -- Phase Detector

   pfd_inst : pfd
   port map(
         a              =>    test_in_divfreq, -- link_swg_pfd,
         b              =>    link_dco_out, -- Saida do DCO entra em PFD -- b_in,
         out_up         =>    link_up_kcounter,
         out_down       =>    link_down_kcounter
         );


-- test_pfd_up_out<=link_up_kcounter;
-- test_pfd_down_out<=link_down_kcounter;

   --- sdownup<= link_down_kcounter OR link_up_kcounter;

-- -- Loop Filter

   k_counter_inst : k_counter
   port map(
         in_up          =>    link_up_kcounter,
         in_down        =>    link_down_kcounter,
--       downup         =>    sdownup,
         kclock         =>    kclock,
         carry          =>    link_carr_dco,
```

Trecho do código do ADPLL implementado e os sinais de referência da rede.

Simulação dos Circuitos



Sinal da rede (link_divfreq_sine) está adiantado em relação ao sinal Fc usado no ADPLL (frequência a ser fornecida pelo ADPLL que sai de adpll_out). O sinal de Carry (link_carr_dco) é mantido até que os dois sinais sejam colocados em fase. O sinal de Borrow fica em zero.

Conclusões

- A simulação completa do sistema foi realizada usando ondas quadráticas com as frequências desejadas (60Hz e variações próximas)

- O comportamento geral do sistema foi aceitável, embora o Dirty Cicle do sinal de saída do ADPLL (**adpll_out**) não fosse o mesmo da rede (ver slide 42)

- O uso do gerador de seno (**SineGen**) não foi usado entrada de **PFD**, somente uma onda quadrática com a frequência igual a 60Hz que pode ser modificada para 61Hz para e para 59Hz a fim de testar o controle do sinal de saída do ADPLL

Conclusões

- As simulações com os geradores de Seno serão realizadas posteriormente para que se possa ter o comportamento completo do sistema, conforme foi requerido pelo professor da disciplina
- O código do DCO foi refeito e não está diretamente relacionado com a explicação presente nos slides 18,19, 20 e 21. As alterações foram necessárias após a melhor compreensão do funcionamento do sistema como um todo. As alterações serão discutidas em outra apresentação

Referências

(1) Behzad Razavi, "Design of Monolithic PhaseLocked Loops and Clock Recovery CircuitsA Tutorial," in Monolithic Phase-Locked Loops and Clock Recovery Circuits: Theory and Design , IEEE, 1996, pp.1-39, doi: 10.1109/9780470545331.ch1.

(2) E. Zianbetov, M. Javidan, F. Anceau and D. Galayko, E. Colinet, J. Juillard. Design and VHDL Modeling of All-Digital PLLs. 8th IEEE International NEWCAS Conference (NEWCAS'10), Montreal: Canada (2010).

(3) K. Lata, M. Kumar. ADPLL Design and Implementation on FPGA. International Conference on Intelligent Systems and Signal Processing (ISSP), IEEE, 2013.

(4) Gayathri M G. Design of All Digital Phase Locked Loop in VHDL. International Journal of Engineering Research and Applications (IJERA), Vol. 3, Issue 4, 2013, pp. 1074-1076.

(5) Henry Young, Alex Tong, Ahmed Allam. Projeto de um DPLL. Disciplina High Level Digital ASIC Design Using CAD (EE552), Departamento de Engenharia Elétrica e de Computadores, Universidade de Alberta, Canadá, 1999. Acessado em 10/05/2021.

(6) P. E. Allen. Lecture 080 - All Digital Phase Lock Loops (ADPLL). Material da disciplina *Frequency Sythesizers*, The School of Electrical and Computer Engineering of Georgia Institute of Technology,2003. Acessado em: 02/07/2021.

(7) M. Kumar, K. Lata. FPGA Implementation of ADPLL with Ripple Reduction Techniques. International Journal of VLSI design & Communication Systems (VLSICS) Vol.3, No.2, 2012 .