UNIVERSIDADE ESTADUAL PAULISTA "JULIO DE MESQUITA FILHO"

Instituto de Geociências e Ciências Exatas de Rio Claro
Departamento de Matemática Aplicada e Computação
Bacharelado em Ciências da Computação

Detector de onda em um sistema trifásico de energia

Disciplina: Microprocessadores I
Prof. Dr. Mario Roberto da Silva

Diogo Sgrillo 208001009
Vinícius Pfeifer 208001129

Agosto de 2015

Índice

Introdução

Em nosso cotidiano utilizamos sistemas elétricos de duas fases facilmente detectados com um voltímetro. Contudo, na indústria ou até mesmo na agricultura, dada a alta necessidade energética, utilizam-se sistemas polifásicos para o funcionamento de grandes máquinas.

O presente projeto tem a finalidade de receber uma onda elétrica de um sistema trifásico como entrada, identificar se a mesma é do tipo ‘R’, ‘S’ ou ‘T’ (de acordo com a classificação descrita futuramente), copiá-la e avisar ao usuário acendendo um led de acordo com a fase detectada.

Uma fonte trifásica, em sumo, gera três ondas senoidais de mesma amplitude, aqui identificadas como ‘R’, ‘S’ e ‘T’, sendo que uma possui 120º de defasagem da outra. Portanto, para identificar um sistema trifásico, basta que as três ondas possuam a mesma amplitude, mesmo período, e a onda ‘T’ seja 120º defasada da onda ‘S’, que por sua vez deve ser 120º defasada da onda ‘R’ (Figura 1).

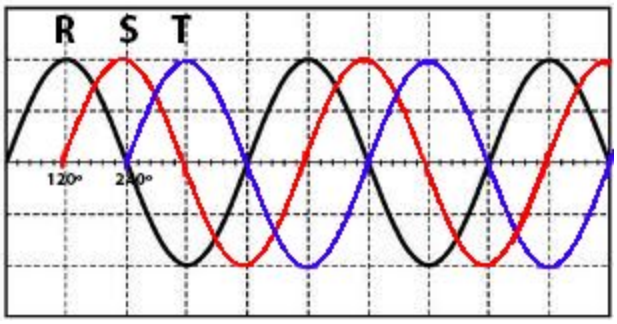


Figura 1: Ondas do sistema trifásico. R - preto; S - vermelho; T- azul.

Portanto, o objetivo do sistema é facilitar a identificação da onda para que o usuário não ligue os fios incorretamente, gerando danos ao maquinário e a rede.

Para montar a arquitetura do sistema foi utilizado o software disponibilizado pelo fabricante da placa Altera, o Quartus v9.0 SP2.

Projeto

Funcionamento

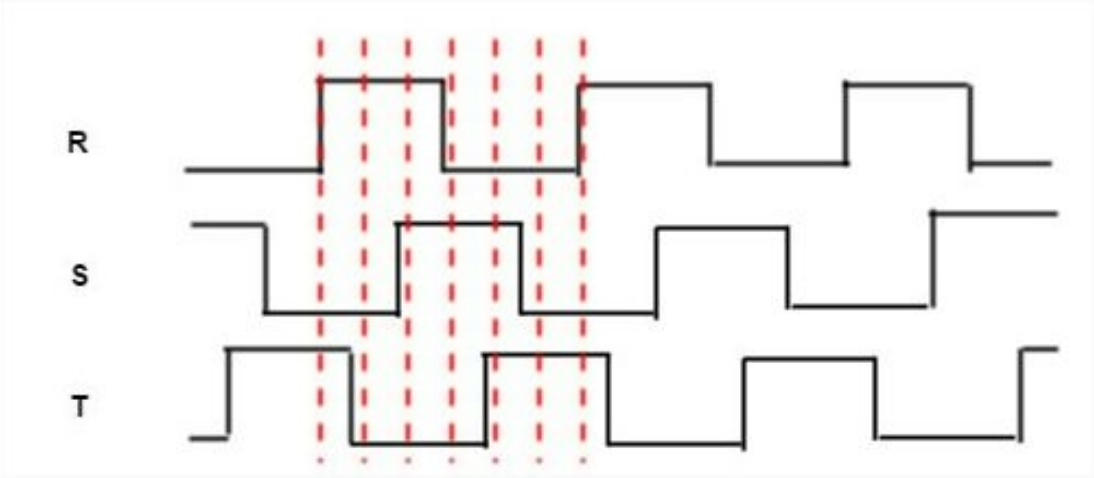
Para funcionar corretamente, o usuário deverá introduzir o fio de uma das fases na entrada da placa. A fase contida neste fio será indicada pelo usuário pressionando o botão correspondente a fase desejada, podendo ser ‘R’, ‘S’, ou ‘T’. A máquina fará o reconhecimento da onda, copiará a mesma, e neste momento o usuário poderá soltar o botão e remover o fio.

Na segunda etapa o usuário deverá pegar um dos fios ainda não reconhecidos e colocar na entrada da placa. O sistema então fará a comparação da defasagem do sinal, com base no primeiro fio introduzido e acenderá o led correspondente para indicar ao usuário a fase atual.

Quando o sinal for copiado, e o nosso circuito começar a reproduzi-lo, é possível ocorrer um encurtamento durante a geração de cada período (explicaremos melhor mais a frente). Criamos então uma interface com o usuário para ele saber quando esse período estiver próximo do fim. O display de 7 segmentos da placa irá exibir um contador (que começará em 9), e irá decrescendo linearmente ate 0, quando o circuito deixará de ser confiável. O usuário então deverá novamente inserir o fio e identificar a fase.

Gerador Trifásico de testes

Com o intuito de realizar testes no circuito, implementamos um gerador trifásico que simula a geração de cada uma das fases ‘R’, ‘S’ e ‘T’. Num primeiro momento, transformamos a onda senoidal em uma onda quadrada, da seguinte maneira :



A partir da figura anterior geramos o seguinte grafo :

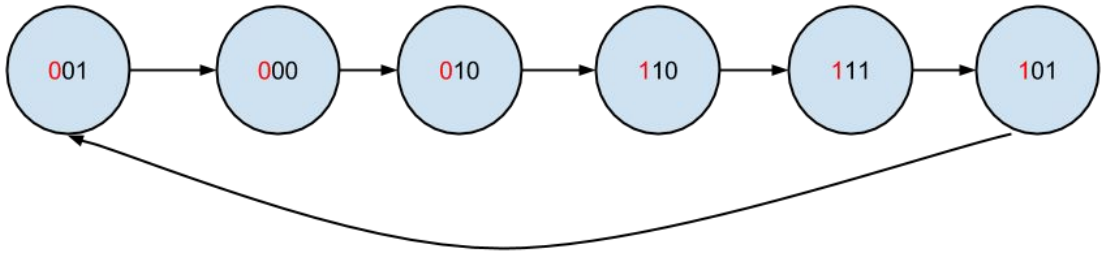


Tabela de saída

R	S	T
1	0	0
1	1	0
0	1	0
0	1	1
0	0	1
1	0	1

Tabela de próximos estados e excitação dos flip-flops

Utilizaremos R negado ao invés de R para obter o estado 000 que é o estado inicial da máquina, ou seja, na lógica saída teremos que negar o R.
!R signifca R negado.

!R	S	T	D2	D1	D0
0	0	0	0	1	0
0	1	0	1	1	0
1	1	0	1	1	1
1	1	1	1	0	1
1	0	1	0	0	1
0	0	1	0	0	0

Minimização

Q0\Q2Q1	00	01	11	10
0	0	0	1	x
1	0	x	1	1

D0 = Q2

Q0\Q2Q1	00	01	11	10
0	1	1	1	x
1	0	x	0	0

D1 = !Q0

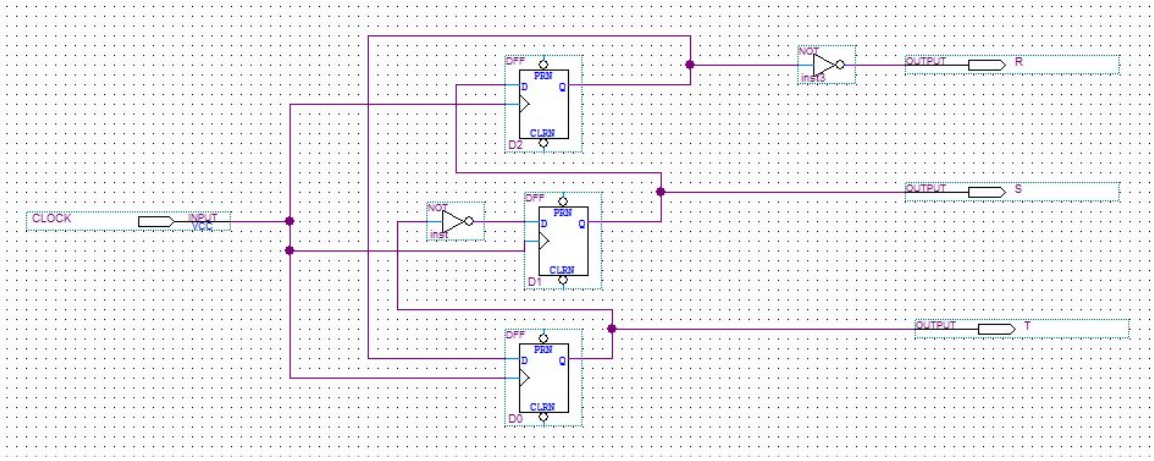
Q0\Q2Q1	00	01	11	10

0	0	1	1	x
1	0	x	1	0

D2 = Q1

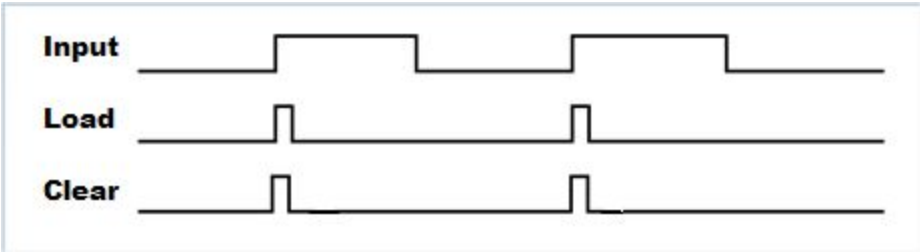
Implementação

Implementação do gerador no Quartus II :



Unidade de Controle

A Unidade de controle é uma máquina sequencial no nosso circuito responsável por gerar os sinais de Load e Clear, onde o Load é responsável por carregar o período da fase de entrada em um registrador e o Clear é responsável por limpar o contador que conta tal período.



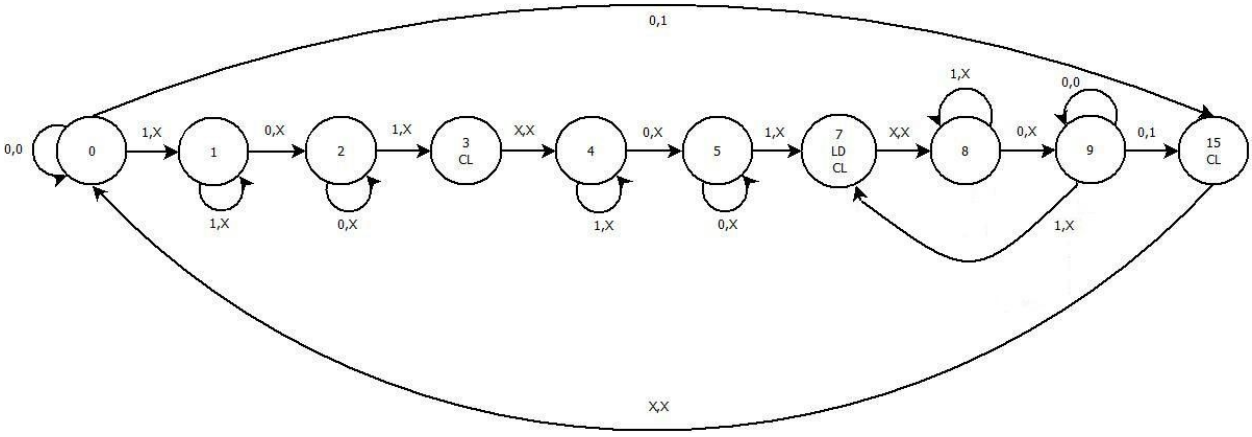
A cada borda de subida do sinal de entrada da referência de fase, a unidade de controle provê sincronamente com a borda de subida do clock da nossa máquina, o pulso de Load que faz com que a contagem em curso seja registrada no registrador (inst1) e sincronamente com a borda de descida do clock da nossa máquina, gera o pulso de Clear, que por sua vez é responsável por limpar o contador (inst), o que permite que a cada borda de subida do sinal de referência seja registrado o seu período anterior.

Este período registrado é usado em um oscilador que deve ser síncrono com a fase de referência, permitindo que a última seja desconectada, pois o oscilador continuará gerando um sinal em "sincronismo" com o sinal de referência escolhido. A identificação da nova fase que será conectada posteriormente, será obtida usando-se a saída deste oscilador em sincronismo com a fase de referência.

Com o objetivo de reutilizar os recursos disponíveis, o oscilador foi implementado utilizando-se o mesmo contador, porém a geração dos pulsos de Clear deve considerar um pulso denominado "CMP", obtido no comparador (inst2), comparando-se a contagem em curso com a última contagem registrada, indicando que mais um período foi transcorrido na fase de referência, ainda que esta não mais esteja conectada ao fio de entrada.

Com a desconexão da fase de referência da única entrada, o pulso "CMP" é usado na unidade de controle para prover o pulso de Clear, que permite a limpeza do contador sincronamente com o transcurso do período, ou seja sincronamente com a fase de referência. Nessa situação, o pulso de Load não será mais produzido, de modo a preservar o último registro válido do período.

Grafo da unidade de controle :



Com a conexão do sinal de referência de fase na entrada externa da unidade de controle e partindo-se do estado inicial (0), a máquina sequencial desta unidade evolui para o estado

seguinte (1) quando houver a comutação desta entrada para o valor alto (primeira variável do par ordenado representado em cada aresta), porém, como isso pode se dar em função da conexão em um fio no instante em que sua fase já se encontre no valor alto, a máquina deve passar para o estado (2) quando a entrada comutar para o valor baixo e finalmente para o estado (3) quando a entrada retornar para o valor alto, este sim, necessariamente será síncrono com uma borda de subida na fase de referência. Neste estado (3), apenas o sinal Clear é provido, limpando o contador de tal forma que na transição para o estado (7), com a nova borda de subida da entrada, o pulso Load registre corretamente o período da fase de referência. Garantida esta primeira medição correta do período, a máquina evolui sucessiva e repetidamente pelos estados (8), (9) e novamente (7), sendo que neste último são providos os sinais Clear e Load, permitindo a realização de novas e corretas medições e registros a cada borda de subida do sinal de referência de fase com consequente passagem pelo estado (11), até que a entrada da fase de referência seja retirada do fio de entrada, impedindo a evolução do estado (9) para o estado (7) e evoluindo para o estado (15) sincronamente com o sinal “CMP” (segunda variável do par ordenado) representado nas arestas e, portanto, ainda que indiretamente, síncrono com a fase de referência. Neste estado (15) é produzido o sinal Clear por um período de clock, mantendo a máquina síncrona com as contagens do período registrado e, portanto síncrona com a fase de referência da qual foi obtido este período. Transcorrido este período de um clock, a unidade de controle evolui para o estado inicial e lá permanece até que novo sinal “CMP” seja produzido ao final da contagem, quando então a unidade de controle evolui para o estado (15) produzindo novo sinal “Clear”. Tal sequência se repete permitindo assim que a identificação de uma nova fase conectada seja possível.

Implementação

Abaixo segue as tabelas e códigos de como foi implementada a unidade de controle.

Tabela de próximos estados

Q3	Q2	Q1	Q0	PP	CMP	D3	D2	D1	D0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	X	0	0	0	1
0	0	0	0	0	1	1	1	1	1
0	0	0	1	1	X	0	0	0	1
0	0	0	1	0	X	0	0	1	0
0	0	1	0	0	X	0	0	1	0
0	0	1	0	1	X	0	0	1	1

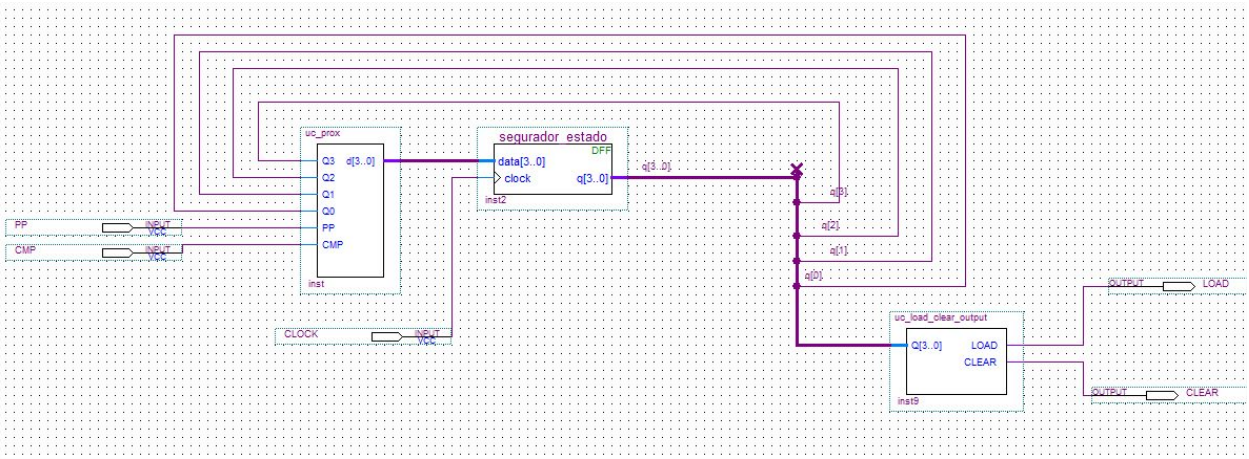
0	0	1	1	X	X	0	1	0	0
0	1	0	0	1	X	0	1	0	0
0	1	0	0	0	X	0	1	0	1
0	1	0	1	0	X	0	1	0	1
0	1	0	0	1	X	0	1	1	1
0	1	1	1	X	X	1	0	0	0
1	0	0	0	1	X	1	0	0	0
1	0	0	0	0	X	1	0	0	1
1	0	0	1	0	0	1	0	0	1
1	0	0	1	1	X	0	1	1	1
1	0	0	1	0	1	1	1	1	1
1	1	1	1	X	X	0	0	0	0

Como a tabela é muito extensa utilizamos o programa Logic Friday para minimizar a saída, obtendo as seguintes funções :

D3 = Q3 Q0' PP' + Q2 Q1 Q0 PP' + Q2' Q1' Q0' PP' CMP
D2 = Q2 Q1' + Q2 Q0' + Q2' Q1 Q0 + Q3 Q0' PP + Q1 Q0 PP
D1 = Q1 Q0' + Q2 Q0 PP + Q3 Q0' PP + Q3' Q2' Q1' Q0 PP'
D0 = Q2 Q1 PP + Q2 Q1' PP' + Q3' Q2' Q1' PP + Q2 Q0' PP' + Q3' Q2' Q0' PP + Q1' Q0' PP' CMP

LOAD = Q3’ Q2 Q1 Q0
CLEAR = Q3’ Q1 Q0 + Q2 Q1 Q0

Como seria necessário criar muitos componentes para criar essa máquina, optamos por fazê-la em VHDL, da seguinte maneira :



Onde uc_prox, implementa a lógica de próximos estados com o seguinte código :

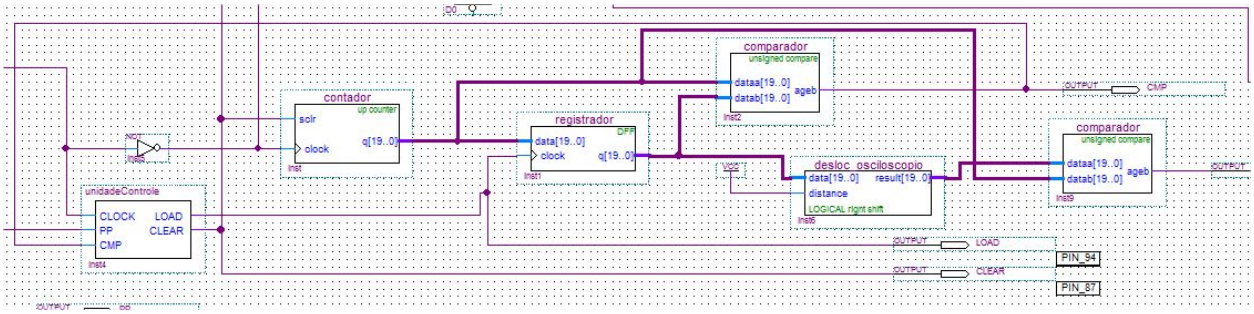
```
library ieee;
use ieee.std_logic_1164.all;


entity uc_prox is
        PORT (Q3,Q2,Q1,Q0,PP,CMP : IN std_logic;
                d: OUT std_logic_vector (3 downto 0));
end uc_prox;


architecture behavior of uc_prox is
begin

d(3) <= (Q3 AND (NOT Q0)) OR ((NOT Q3) AND Q2 AND Q1) OR  ((NOT Q2) AND (NOT Q1) AND (NOT Q0) AND (NOT PP) AND CMP) OR (Q3 AND (NOT Q1) AND (NO
d(2) <= (Q2 AND (NOT Q1))  OR ((NOT Q2) AND Q1 AND Q0 ) OR ( (NOT Q3) AND (NOT Q1) AND (NOT Q0) AND (NOT PP) and CMP) OR (Q3 and (NOT Q1) and Q
d(1) <= (Q1 AND (NOT Q0))  OR (Q2 and (NOT Q1) and Q0 and PP) OR ((NOT Q3) AND (NOT Q2) AND (NOT Q1) and Q0 and (NOT PP)) OR (Q3 and (NOT Q1) a
d(0) <= ((NOT Q1) and Q0 and PP ) OR (Q3 and (NOT Q1) AND (NOT PP) ) OR (Q2 and (NOT Q1) AND (NOT PP)) OR ((NOT Q3) AND (NOT Q2) AND (NOT Q0) a

end behavior;
```

e a uc_load_clear_output implementa a lógica de saída com o seguinte código :

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3
4
5   entity uc_load_clear_output is
6           PORT (Q : IN std_logic_vector (3 downto 0);
7                   LOAD,CLEAR: OUT std_logic);
8   end uc_load_clear_output;
9
10
11  architecture behavior of uc_load_clear_output is
12  begin
13    LOAD <= (not Q(3)) and Q(2) and Q(1) and Q(0);
14    CLEAR <= ((not Q(3)) and Q(1) and Q(0)) or (Q(2) and Q(1) and Q(0));
15  end behavior;
```

Com o objetivo de se obter um sinal que possa ser comparado em um osciloscópio com a fase de referência, o circuito é expandido com outro comparador de magnitude, porém a

comparação da contagem em curso se faz contra a metade da contagem referente ao período registrado, obtida com um registrador de deslocamento (inst6) que desloca a contagem registrada um bit a direita. Na saída deste comparador obtemos então um sinal de ciclo ativo de 50%, síncrono com a fase de referência. Obtendo a seguinte implementação parcial :



Em resumo, depois que o sinal de referência é retirado da entrada, o contador continua contando repetidamente de zero até alcançar a contagem registrada correspondente ao período da fase de referência e em fase com ela, permitindo que nova fase conectada à entrada seja identificada.

Comparador de Fase

A identificação da nova fase conectada é feita com auxílio de uma máquina sequencial denominada “comparadorFase”, que passa por 6 estados igualmente distribuídos durante um período da fase de referência.
Conseguimos identificar a fase de acordo com o estado do grafo no instante da borda de subida da fase que queremos comparar. Se a borda de subida ocorre no primeiro estado, ou no sexto, isto significa que a fase lida é a mesma que a fase de referência. Caso seja o segundo ou terceiro estado, significa que a fase lida e uma a frente da fase de referência (caso referencia seja R, então S, caso seja S então T, e assim por diante). Seguindo a mesma lógica, caso seja o quarto ou quinto estado, significa que a fase lida é uma atrás da fase de referência (ou duas adiantadas).
Desta forma, podemos perceber que escolhemos 2 bits dentre os 3 que identificam os estados para identificar a nova fase conectada.
Resumindo:

00x => fase igual

01x => fase +1
10x => fase +2 ou fase -1

Essa maquina deve ter a cadência de 6x a fase de referencia. Portanto, podemos usar um contador de modulo 6 para dividir o clock por 6 e gerar o clock que é usado para a contagem entre as bordas de subida da fase de referência. Em seguida podemos ver o grafo de tal máquina:

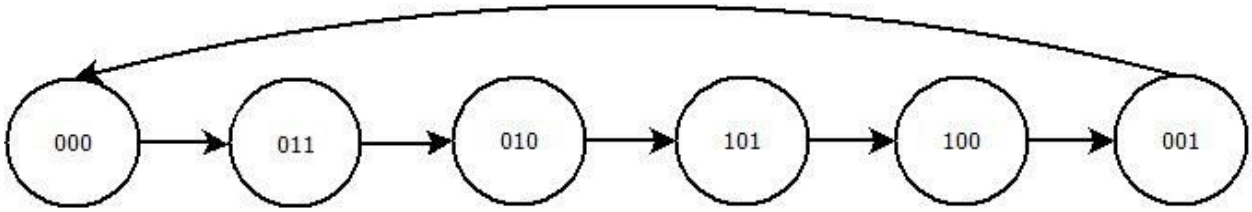


Tabela de excitação dos flip flops

Q2	Q1	Q0	D2	D1	D0
0	0	0	0	1	1
0	0	1	0	0	0
0	1	0	1	0	1
0	1	1	0	1	0
1	0	0	0	0	1
1	0	1	1	0	0
1	1	0	X	X	X
1	1	1	X	X	X

Minimização

Q2\Q1Q0	00	01	11	10
0	0	0	0	1
1	0	1	X	X

D2 = Q1.Q0’ + Q2.Q0

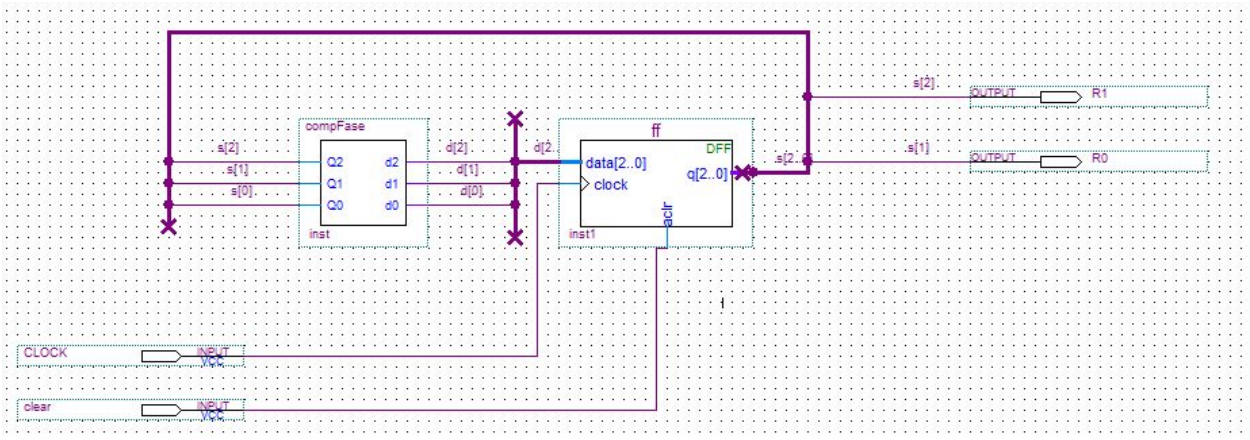
Q2\Q1Q0	00	01	11	10
0	1	0	1	0
1	0	0	X	X

D1 = Q1.Q0’ + Q2’.Q1’.Q0’

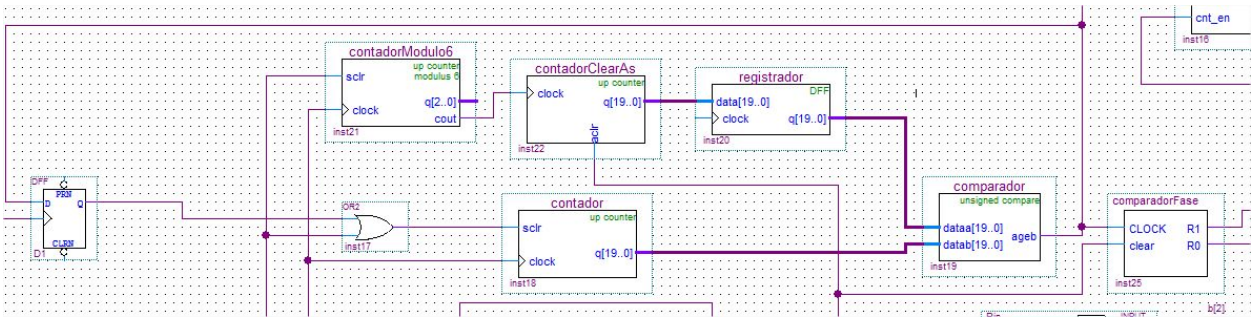
Q2\Q1Q0	00	01	11	10
0	1	0	0	1
1	1	0	X	X

D0 =Q0’

Implementação



Implementação com o contador de módulo 6



Como o contador recebe um Clock com 1/6 da frequência original do circuito, usamos um clear assíncrono que é acionado por um pulso que ascende na borda de descida anterior ao pulso Clear e com duração de metade dele, o que é provido pela operação “e” entre a entrada (conectada ao Clear) e a saída do flip-flop tipo D (D0), cujo chaveamento ocorre na borda de descida do clock geral.

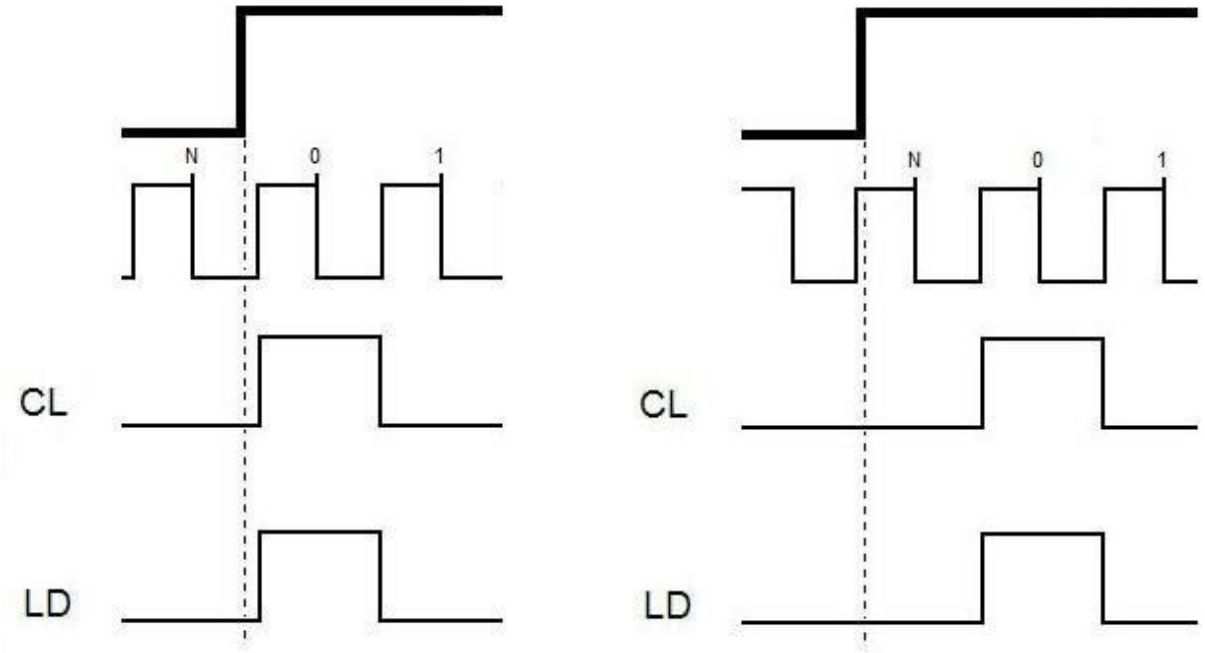
A saída de um comparador de magnitude (inst19) entre o registro de 1/6 do período da fase de referência e a saída do outro contador com contagem nas bordas de descida do clock geral

deve ser usado para zerar este contador (inst18) à cadência de 6x a frequência da fase de referência. Como a saída do comparador ocorre também na borda de descida do clock geral, ela deve então ser atrasada de meio período de clock pelo flip-flop tipo D (D1) chaveado pela borda de subida do Clock geral antes de ser tomado como entrada síncrona de clear (sclr) do contador acionado pela borda de descida, assim como a unidade de controle produz o sinal de Clear síncrono com a borda de subida do Clock geral a partir do “CMP” que é produzido em sincronia com a borda de descida do Clock geral.

Como a divisão inteira por 6 pode envolver resto, esta entrada sclr deve ter como entrada alternativa (inst22) o pulso Clear produzido a cada período em sincronismo com a fase de referência, evitando acúmulo de erro a cada ciclo em função do resto não contado. A máquina de 6 estados denominada “comparadorFase” evolui para o primeiro estado prematuramente em função do resto não contado, entretanto a evolução para o segundo estado não é afetada em função da correção pela zeragem do contador a cada pulso clear, produzido no estado (15) da máquina Unidade de Controle. Quando o pulso Clear é produzido no estado (3) e (7) da Unidade de Controle, em função da nova fase conectada à entrada, este pulso Clear deve também zerar a máquina “comparadorFase” e isso deve ser feito sincronamente com a borda de descida do clock geral, pois o chaveamento do registrador de saída de fase (lpm DFF1 inst24) ocorre na borda de subida do pulso Clear que é produzido pela Unidade de controle sincronizada com a borda de subida do clock geral. O mesmo sinal que zera o contador de 1/6 de período é usado então para zerar a máquina comparadorFase.

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std;
4
5
6  entity result is
7          PORT (ledIN : IN BIT_VECTOR (2 downto 0);
8                 r : IN BIT_VECTOR (1 downto 0);
9                 buttons : IN BIT_VECTOR (2 downto 0);
10                result : OUT BIT_VECTOR (2 downto 0));
11 end result;
12
13
14 architecture behavior of result is
15 begin
16   process (ledIN,r,buttons)
17   begin
18       if (buttons = "100") then
19         result <= buttons;
20       elsif (buttons = "010") then
21         result <= buttons;
22       elsif (buttons = "001") then
23         result <= buttons;
24       elsif (r = "00") then
25         result <= ledIN;
26       elsif (r = "01") then
27         result <= (ledIN ror 1);
28       elsif (r = "10") then
29         result <= (ledIN rol 1);
30       else
31         result <= ledIN;
32       end if;
33   end process;
34
35 end behavior;
```

As saidas R0 e R1 do comparadorFase sao as entradas utilizadas no result que ira gerar o output dos leds. Optamos por fazer esse componente utilizando VHDL e if's e funcoes de rotacao, uma vez que o ouput do comparadorFase ja nos diz que led deve ser acesso ou nao. Alem disso, e utilizada uma entrada com o botao que identifica a ponta de prova. Caso a ponta de prova esteja conectada, e o botao pressionado, essa deve ser imediatamente a saida dos leds. Caso contrario, se a entrada for 00, deve ser mantido o led que ja estava acesso. Caso seja 01, e feito um ror, e caso seja 10 um rol para exibir adequadamente os leds.



E possivel que aconteca um atraso de 0 a -2 pulsos de clocks. Podemos ver pela figura da esquerda que já transcorreu ½ do período de clock, quando a contagem é zerada, representando um erro de - ½ período de clock e pela figura da direita, que já transcorreu 1.½ períodos de clock quando a contagem é zerada, representando um erro de -1.½ períodos de clock. Por outro lado, pela figura da esquerda nota-se que no valor registrado é desconsiderado ½ período de clock, representando um erro de - ½ período de clock e pela figura da direita que é registrado um valor que considera ½ período de clock a mais do que o contido no período da fase de referência, representando um erro de + ½ período de clock.

Embora este erro de 0 a -2 possa parecer desprezivel, ele e acumulado enquanto o circuito estiver gerando o sinal. Esse timeout pode ser estimado considerando que ⅛ desta contagem maxima de clocks sera consumida numa razao de 2 contagens a cada periodo de fase de referencia que dura o inverso da frequencia desta fase. Chegaos entao na formula Timeout = clock/2*6*freq_rede*freq_red. Como o clock maximo da UP eh 25.175MHz e para a frequencia de nossa rede 60Hz, obtemos um timeout aproximadamente de 583s ou 9min e 43seg.

Criamos entao um dispositivo que conta linearmente quando esse periodo esta proximo de ocorrer, apagando 1 led cada vez que o periodo se aproxima, apagando todos os leds quando o sistema nao e mais confiavel.

Conclusao

Apesar de termos concluido o projeto proposto, acreditamos que poderiamos ter evoluido mais na apresentacao do timeout para o usuario. Acreditamos que a utilizacao do display de 7 segmentos com um contador que decresce linearmente com o tempo de uso do gerador seria uma interface melhor com o usuario. Contudo, nao tivemos tempo suficiente para implementa-la e testa-la.
Porem, acreditamos que conseguimos atender e implementar tudo que era requisito do sistema, e a implementacao deste projeto acarretou um grande aprendizado na pratica desta disciplina.