UNIVERSIDADE ESTADUAL DE FEIRA DE SANTANA

DEPARTAMENTO DE TECNOLOGIA

TEC498 PROJETO DE CIRCUITOS DIGITAIS

PROBLEMA 2: GESTÃO DIGITAL DE VAGAS PARA ESTACIONAMENTO

Carlos Henrique de Oliveira Valadão e Luis Fernando do Rosario Cintra

Tutor: Thiago Cerqueira de Jesus

1 INTRODUÇÃO

No Brasil existem 111.446.870 veículos com cadastro ativo (IBGE, 2020), com tendência de crescimento. Essa frota de veículos reflete-se na mobilidade das cidades brasileiras, a exemplo da cidade de Recife detentora do título de cidade com o pior trânsito do Brasil (SOUZA, 2020), uma vez que trânsitos caóticos têm como característica uma alta indisponibilidade de vagas, parar ou estacionar um veículo vem se tornando um desafio. Nesse sentido, estacionamentos privados surgem como uma alternativa à mobilidade de cidades e megalópoles (SOUZA, 2020). De tal forma que o mercado de estacionamentos está cada dia mais concorrido, empresas buscam meios de tornar o seus serviços e produtos os melhores possíveis, um dos meios mais utilizados de fazê-lo é por meio da tecnologia, tais como: automação de cancela, pagamento por aproximação, implementação de sistemas de gestão dentre outros(ECONOMIASC, 2019).

Este documento apresenta o desenvolvimento de um sistema gerenciador de vagas para estacionamento de 8 vagas, que exibe de forma discernível a quantidade de vagas livres e ocupadas do estacionamento, bem como exibe uma relação em uma matriz de LEDs, 4 linhas e 2 colunas, de quais vagas estão livres ou não. A solução valeu-se de circuitos sequenciais e circuitos puramente combinacionais, bem como verilog em seu modelo comportamental. As 8 vagas do estacionamento são representadas por meio de um conjunto de 8 chaves de acionamento mecânico, do tipo HH, a apresentação da quantidade de vagas livres e ocupadas em é realizada em realizada 4 displays de 7 segmentos, todo o estacionamento é mapeado por meio de uma matriz de LEDS 4x2 de tal forma que cada vaga está associada a um LED da matriz de LEDS, estando este ligado caso sua determinada vaga esteja ocupada e desligado em caso contrário, todas as informações supracitadas são paralelamente exibidas. O design do sistema foi realizado por meio da IDE Quartus II, e a sua síntese na FPGA da família MAX II EPM240T100C5.

Este trabalho estrutura-se da seguinte forma: A seção 2 apresenta os principais conceitos na qual todo este relatório está fundamentado. A seção 3 elucida a implementação do sistema de gerenciamento do estacionamento de 8 vagas, enquanto na seção 4 os resultados obtidos por meio da solução são discutidos. Por fim, a seção 5 dispõe os testes realizados da solução sob condições específicas.

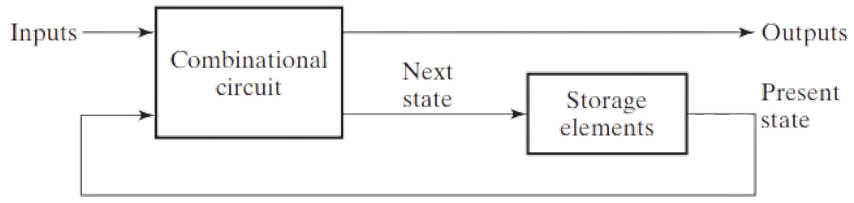
2 FUNDAMENTAÇÃO TEÓRICA

Nesta seção aborda-se circuitos sequenciais: contadores assíncronos e verilog comportamental: estrutura always, detector de borda, case, blocos estruturais, atribuições bloqueantes e não bloqueantes, e variável do tipo reg. Conceitos estes, imprescindíveis para o desenvolvimento da solução do sistema de gerenciamento de estacionamento.

2.1 CIRCUITOS SEQUENCIAIS

A utilização de circuitos sequenciais fez-se obrigatória ao decorrer do desdobramento da solução, pelo motivo do uso de recursos que permitissem a memorização de informações essenciais, bem como ao intrínseco paralelismo, isto é, as saídas da solução desenvolvida alternavam aproximadamente 50 vezes por segundo, cada. Um circuito sequencial é composto pela interconexão de um circuito combinacional com elementos de armazenamento capazes de armazenar bits, conforme a figura 1. O estado de um circuito sequencial é definido pelo estado do circuito de armazenamento, em um dado momento, uma vez que as entradas de um circuito sequencial, bem como o seu estado anterior determinam o seu novo valor binário, estado. A figura 1 descreve que as saídas de um circuito sequencial não são em função somente de suas entradas, mas também do estado anterior do do elemento de armazenamento. Dessa forma, o estado das saídas de um circuito sequencial é uma combinação específica de suas entradas, estados internos e saídas. (MANO; KIME, 2007).

Figura 1 – Diagrama de Blocos de um Circuito Sequencial

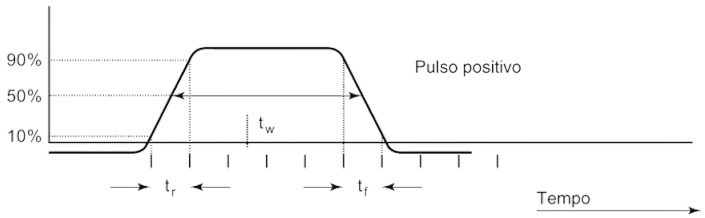


Fonte: Mano e Kime (2007, p. 198)

Existem dois tipos principais de circuitos sequenciais, e suas classificações dependem única e exclusivamente do momento ao qual suas entradas são fornecidas e seus estados internos mudam. O comportamento de um circuito sequencial síncrono pode ser definido conhecendo os seus sinais em um discreto instante de tempo, enquanto o comportamento de um circuito sequencial síncrono depende de suas entradas em qualquer instante de tempo e o tempo em que as entradas mudam, (MANO; KIME, 2007), neste documento serão abordados circuitos sequenciais síncronos, pois a abordagem do problema utilizando circuitos sequenciais síncronos permite uma solução simples. Um circuito sequencial síncrono faz o uso de sinais que afetam os elementos de armazenamento, apenas em um discreto instante de tempo. A sincronização é alcançada por meio de um circuito chamado gerador de clock, ele produz uma sequência de pulsos de clock. O clock é uma onda quadrada pulsante em determinada frequência, na lógica positiva a borda de subida simboliza nível lógico ALTO, enquanto a de descida simboliza nível

lógico BAIXO. A onda quadrada é enviada a todo o circuito, medida em Hertz (Hz), tem como função sincronizar os componentes de um circuito que o utiliza. Como mostrado no pulso positivo da figura 2, todos os circuitos associados ao pulso positivo executam as suas funções planejadas enquanto ele estiver em nível alto, permanecendo inativo enquanto ele estiver em nível BAIXO. (TOCCI; WIDMER, 1997).

Figura 2 – Pulso Positivo de um Clock



Fonte: Tocci e Widmer (1997, p. 184)

Em dispositivos de memorização, os pulsos são distribuídos de forma que os elementos de armazenamento são afetados apenas em algum específico relacionamento a cada pulso, os pulsos de clock são aplicados em conjunto com outros sinais para especificar uma mudança nos elementos de armazenamento, e a saída deles podem mudar de valor apenas na presença de pulso de clock. De modo geral, circuitos assíncronos mais complexos são de difícil design, então em grande parte dos casos são preferíveis os circuitos síncronos, entretanto em determinados casos, circuitos assíncronos são imprescindíveis, um caso importante é o uso de latches assíncronos como blocos para construir elementos de armazenamentos chamados flip-flops, que guardam informações em circuitos síncronos. Circuitos sequenciais síncronos que usam pulsos de clock como entrada são chamados de circuitos sequenciais com clock (*clocked sequential circuits*) (MANO; KIME, 2007).

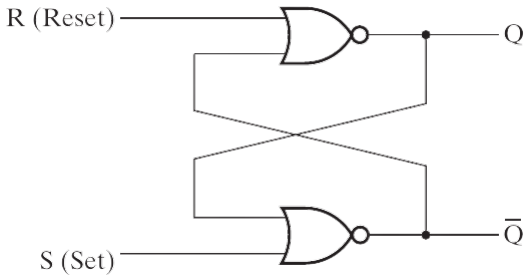
2.1.1 Latches e Flip-flops

Latch é um elemento de armazenamento que permite manter um estado binário, por tanto tempo quanto a duração de sua corrente elétrica, até a sinalização para mudança de estado, em sua entrada. Um latch difere-se de um flip-flop em seus números de entrada, e de quais entradas irão afetar os seus estados binários. Os elementos de armazenamento mais básicos são os latches, dos quais os flip-flops são usualmente construídos.

O latch *SR* é um circuito construído a partir de duas portas *NOR* realimentadas, figura 3. Possui duas entradas, chamadas S para setar e R para resetar, uma saída Q e $\bar{Q}$, negação de Q, e dois estados úteis. Quando a saída $Q = 0$ e $\bar{Q} = 1$. o latch é dito em estado setado (*set*) quando $Q = 0$ e $\bar{Q} = 1$ em seu estado resetado (*reset*) (MANO; KIME, 2007). Conforme mostrado na figura 4, duas combinações de entradas fazem o circuito assumir a saída $Q = 1$, $S = 1$, $R = 0$ e $S = 0$ e $R = 0$ deixa o circuito com saída $Q = 0$, anteriormente. É possível fazer $Q = 0$ aplicando 1 a entrada R, $R = 1$ $Q = 0$ e $R = Q = 0$, com Q anteriormente em 0 (MANO; KIME, 2007). Se 1 for aplicado em ambas as entradas do latch, $S = R = 1$, ambas as entradas vão para zero, $Q =$

$\bar{Q} = 0$. Este é um estado indefinido, pois viola o requisito de que as saídas são complementares, uma à outra (MANO; KIME, 2007).

Figura 3 – Diagrama lógico de um latch NOR



Fonte: Mano e Kime (2007, p. 201)

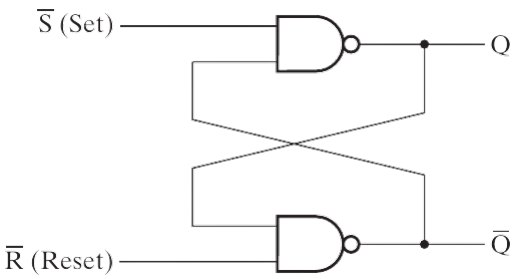
Figura 4 – Tabela Verdade Latch NOR

S	R	Q	$\overline{Q}$	
1	0	1	0	Set state
0	0	1	0	
0	1	0	1	Reset state
0	0	0	1	
1	1	0	0	Undefined

Fonte: Mano e Kime (2007, p. 201)

Existe outra forma de construir um latch, consiste no uso de duas portas NAND acopladas, conforme a figura 5. O latch construído com portas NAND, também chamado latch $\bar{S}R|$, as barras acima de R e S simbolizam que as entradas devem estar em seu estado complementar do latch SR para alterar o estado do circuito, conforme a figura 4 TabelaVerdadeLatchNAND (MANO; KIME, 2007).

Figura 5 – Diagrama lógico de um latch NAND



Fonte: Mano e Kime (2007, p. 202)

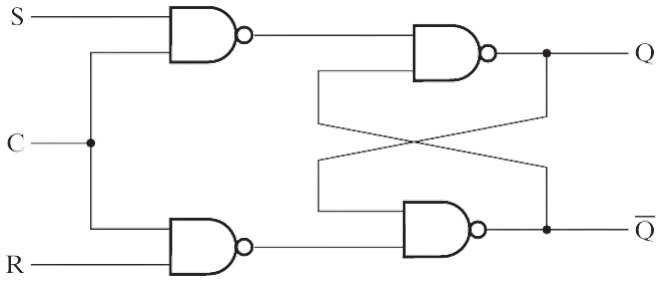
Figura 6 – Tabela Verdade Latch NAND

$\overline{S}$	$\overline{R}$	Q	$\overline{Q}$	
0	1	1	0	Set state
1	1	1	0	
1	0	0	1	Reset state
1	1	0	1	
0	0	1	1	Undefined

Fonte: Mano e Kime (2007, p. 202)

As operações básicas de um latch NOR ou NAND podem ser modificadas por meio de uma entrada adicional de controle, ela determina quando o estado do latch pode ser alterado. Um latch *SR* com uma entrada de controle, figura 7 consiste de um latch NAND básico. A entrada de controle consiste num sinal de ativação (*enable*) para as outras duas entradas. A saída das portas NAND permanecem em nível lógico ALTO enquanto a entrada de controle está em nível lógico BAIXO. Quando a entrada de controle vai para 1, as informações contidas em S e R podem, agora, afetar o estado do circuito. O estado é alcançado com S = 1, R = 0 e C = 1. Para mudar o estado reset, as entradas devem ser S = 0, R = 1 e C = 1. Em caso contrário, quando C está em 0, o circuito permanece em seu estado atual. Logo a entrada C = 0 desativa o circuito para que o estado da saída não mude, resguardando os valores de S e R. Contudo, quando C = 1 e S = R = 1, o estado do circuito não se altera. Todas as condições de operação do latch NAND com entrada de controle são descritas na tabela verdade do circuito, figura 8.

Figura 7 – Diagrama lógico de um latch NAND com Entrada de Controle



Fonte: Mano e Kime (2007, p. 203)

Figura 8 – Tabela Verdade Latch NAND com Entrada de Controle

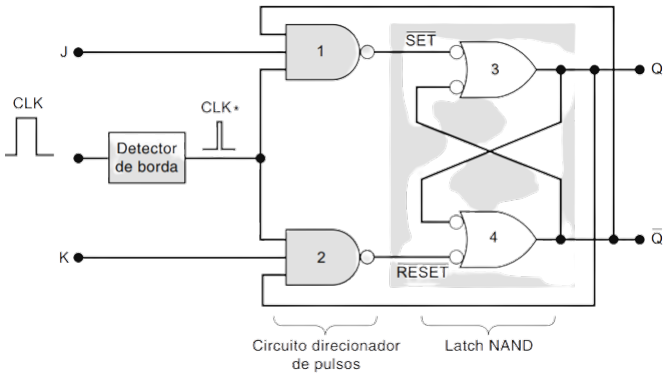
C	S	R	Next state of Q
0	X	X	No change
1	0	0	No change
1	0	1	Q = 0; Reset state
1	1	0	Q = 1; Set state
1	1	1	Undefined

Fonte: Mano e Kime (2007, p. 203)

É de suma importância a explanação do flip-flop JK, pois ele é a célula unitária de um circuito que proporcionou a redução do clock da placa utilizada, para uma frequência aceitável, dentro dos parâmetros do projeto do sistema de gerenciamento.

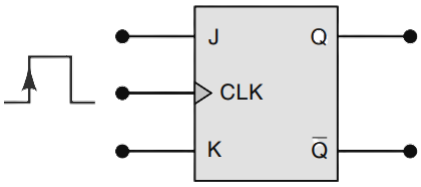
Flip-flop (*FF*) é um circuito biestável síncrono, conhecido como multivibrador biestável, e diferentemente dos latches, ele é acionado por borda, isto é, muda o estado de sua saída apenas no momento de transição da entrada de controle, comumente o clock, conservando a sua saída noutro específico momento. Em pulso positivo, as funções são executadas no tempo de subida (*rise time*), 0 para 1, e permanece inerte enquanto a entrada de controle estiver em 0, o pulso negativo as funções são executadas no tempo de descida (*fall time*) 1 para 0. O *FF JK* possui duas entradas, J e K, estas controlam os estados do flip-flop, e duas saídas, Q e $\bar{Q}$, Q é complementar de $\bar{Q}$ e vice-versa. Temos na figura 9, a representação do circuito do flip-flop acionado por borda de subida. A imagem 10, por sua vez, apresenta o diagrama de circuito de um flip-flop acionado por borda de subida de clock.. Conforme a figura 11, ao fazer J=0, K = 0, CLK = 1, a saída Q não se altera, por sua vez J = 1, K = 0 e CLK = 1, o estado do flip-flop vai para 1, enquanto J = 0, K = 1 o estado do flip-flop, quando J = K = 1, o estado da saída é comutado (TOCCI; WIDMER, 1997).

Figura 9 – Diagrama lógico de um Flip-flop JK



Fonte: Tocci e Widmer (1997, p. 192)

Figura 10 – Diagrama de Bloco de um Flip-flop JK



Fonte: Tocci e Widmer (1997, p. 191)

Figura 11 – Tabela Verdade de um Flip-flop JK

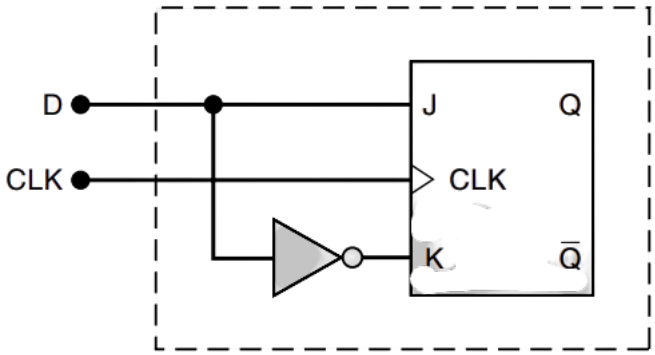
J	K	CLK	Q
0	0	↑	Q_0 (não muda)
1	0	↑	1
0	1	↑	0
1	1	↑	$\overline{Q}_0$ (comuta)

Fonte: Tocci e Widmer (1997, p. 191)

Apesar de o *FF JK* ser o foco deste trabalho, porque somente ele foi utilizado na implementação de alguns módulos utilizados no projeto, vale ressaltar, os outros tipos de *FF* que derivam justamente do *FF JK*. A partir do *FF JK* é possível obter-se dois outros tipos de *FF*, apenas alterando a forma como as suas entradas estão relacionadas, o tipo *D* e tipo *T*, ambos são um *FF JK* com suas entradas curto-circuitadas, contudo o *FF D* difere-se do *FF T* pois há um inversor na entrada *K*. figura 12 (a), apresenta um *FF* tipo *D* pulso positivo. Não existe o caso onde J = K, então os casos possíveis são, figura 13, J = 1, K = 0 e J = 0, K = 1. O D

significa Dado (*data*), este tipo de flip-flop é designado em sistemas de armazenagem de alta velocidade como registradores (TOCCI; WIDMER, 1997).

Figura 12 – Diagrama de Bloco de um Flip-flop D



Fonte: Tocci e Widmer (1997, p. 193)

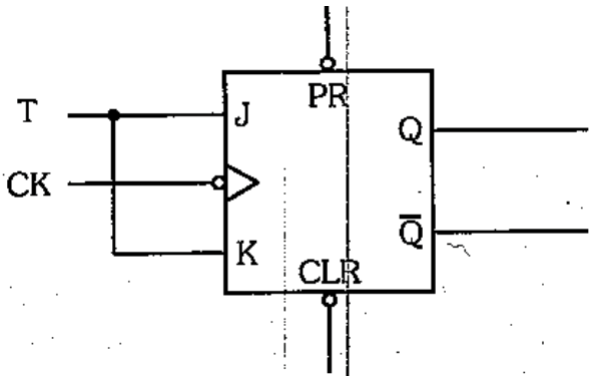
Figura 13 – Tabela Verdade de um Flip-flop D

D	CLK	Q
0	↑	0
1	↑	1

Fonte: Tocci e Widmer (1997, p. 193)

No *FF* tipo *T*, *J* sempre será igual a *K*, figura 14, dessa forma não existe um caso, onde *J* é diferente de *K*. O *T* vem de Troca (*Toggle*), pois o FF sempre comuta sua saída quando a sua entrada for 1, e guarda a informação caso contrário, podendo assumir dois estados, conforme a figura 15, *FF* tipo *T* pulso positivo, onde *Qa* é estado anterior do circuito, a tabela figura da tabela assume tacitamente que $CLK = 1$.

Figura 14 – Diagrama de Bloco de um Flip-flop T



Fonte: Capuano e Idoeta (2006, p. 246)

Figura 15 – Tabela Verdade de um Flip-flop T

T	Qf
0	Qa
1	$\overline{Q}$a

Fonte: Capuano e Idoeta (2006, p. 247)

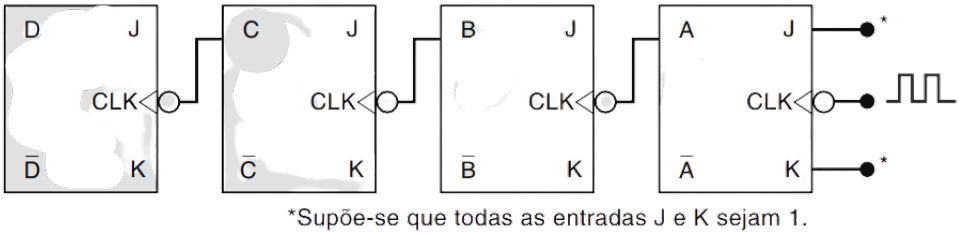
2.1.2 Contadores Assíncronos

A combinação de flip-flops em cascata permitiram a construção de um circuito que fez possível o aumento do período do *clock*. Circuitos contadores são utilizados em sistemas digitais com diversos propósitos. Eles talvez contem o número de ocorrências de certos eventos, gere intervalos de tempo para controlar diversas tarefas em um sistema, contem a quantidade de tempo gasto entre dois eventos, servir como divisor de frequência e assim por diante. (BROWN; VRANESIC, 2007).

Um contador binário de *n-bit* pode ser construído justamente com *n flip-flops* e mais nenhum outro componente, a figura 16 mostra um contador $n = 4$, com FF JK, assumindo que J = K = 1, dessa forma, todos os FF operarão em modo de troca (*toggle*). Então, cada bit do contador é invertido, se e somente se o seu imediato antecessor mudar de zero para 1. Isto corresponde a uma sequência de contagem binária normal quando um bit particular muda de 1 para 0, ele "*carrega*" o próximo bit mais significante. O contador é chamado de *ripple counter* (contador de ondulação), porque o *carry* da informação ondula dos bits *LSD's* até os *MSD' s*. É normal que divisores de frequência sejam construídos a partir de FF T, uma vez que apenas a

função de troca é utilizada neste tipo de funcionalidade, conforme a figura 16 o *MSB* é o Flip-flop de saída A e $\bar{A}$, e por sua vez o *LSD* é o Flip-flop cuja as saídas são D e $\bar{D}$ (WAKERLY,).

Figura 16 – Contador Assíncrono, ondulante, de 4 bits



Fonte: Tocci e Widmer (1997, p. 306)

2.2 VERILOG COMPORTAMENTAL

Serão apresentados aqui, elementos do verilog comportamental que viabilizaram a aplicação de circuitos que permitem o gerenciamento de clocks. Elementos como: o bloco *case*, *always*, *reg* serão abordados aqui, uma vez que o verilog em seu modo estrutural não dispõe destas funcionalidades, o que revela uma vantagem do verilog comportamental sobre o estrutural, pois com este é impossível trabalhar com clock (F., 2018).

2.2.1 Bloco Always

O elemento central do verilog comportamental é o bloco *always*. Um bloco *always* é sempre executado, e é opcionalmente seguido de um ou mais blocos "procedurais". Um tipo de bloco "procedural" é o bloco *begin-end*, figura 17, que inclui uma lista de outros blocos procedurais. Os blocos procedurais contidos em um bloco *always* executa sequencialmente assim como uma linguagem de programação de software.

Nas três primeiras formas do *always*, o @ seguido por parêntesis simboliza uma lista de nomes de sinais, chamada *lista de sensibilidade* (*sensitivity list*), ao qual pode-se especificar todos os sinais, que talvez afetem o resultado do bloco *always* (F., 2018). As quarta e quinta formas da lista de sensibilidade da imagem 17, são usadas em circuitos sequenciais. As siglas *posedge* e *negedge*, são utilizadas em circuitos sequenciais, isto é. para a detecção do momento de transição da onda de clock, de 0 para 1, no caso da quarta lista de sensibilidade, ou de 1 para 0, no caso da quinta lista de sensibilidade. (F., 2018), o bloco always foi utilizado na construção do *FF JK*, contudo também foi necessário um tipo especial de atribuição que será mostrado na seção seguinte.

Figura 17 – Modos de Declaração do Bloco Always

```
always @ (signal-name or signal-name or ... or signal-name)
  procedural-statement
always @ (signal-name, signal-name, ... , signal-name)
  procedural-statement
always @ ( * ) procedural-statement

always @ (posedge signal-name) procedural-statement
always @ (negedge signal-name) procedural-statement

always procedural-statement
```

Fonte: F. (2018, p. 205)

2.2.2 Atribuição Bloqueante e Não Bloqueante

Os dois primeiros blocos procedurais necessários são as *atribuições bloqueantes* e *não bloqueantes*, com a sintaxe mostrada na figura 18. O lado esquerdo de cada atribuição deve ser uma variável, porém o lado direito pode ser qualquer expressão que produza um valor compatível com a variável.

Uma atribuição bloqueante se comporta e parece com uma atribuição de qualquer outra linguagem de programação procedural, como C. Uma atribuição não bloqueante comporta-se como um pouco diferente, ela avalia o lado direito da atribuição imediatamente, mas ela não atribui o resultado à variável até um infinitesimal delay, um pouco antes, após todo o bloco always ter sido executado. Então o antigo valor da variável continua disponível pelo resto do bloco always, em grande parte de seu tempo de execução (F., 2018).

Figura 18 – Atribuição Bloqueante e Não Bloqueante em Verilog

```
variable-name = expression ;    // blocking assignment

variable-name <= expression ;   // nonblocking assignment
```

Fonte: F. (2018, p. 208)

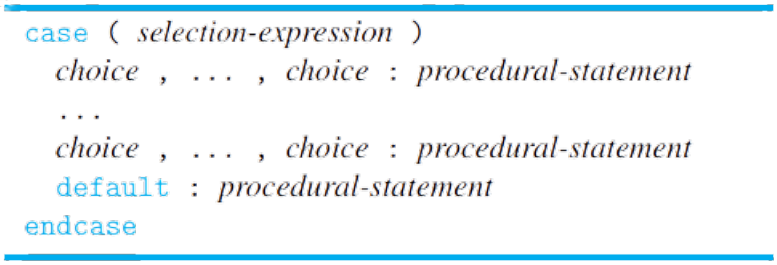
Deve-se usar a atribuição não bloqueante no bloco always sempre quando for necessária a criação de circuitos sequenciais por meio do bloco *always*, optando pela atribuição bloqueante ao criar circuitos combinacionais (F., 2018). A atribuição não bloqueante, por sua vez, é utilizada no bloco *case* para, agora, determinar o comportamento do *flip-flop*, perante as suas entradas e a sua entrada de controle.

2.2.3 Bloco Case

A declaração do bloco *case* começa com a palavra chave case seguida de parênteses, imagem 19, *expressão de seleção*. O passo seguinte é a declaração dos itens do *case*, cada um possuindo uma lista de ações e um bloco procedural. Existe ainda, a declaração *default* que pode ser opcionalmente incluída (F., 2018).

O funcionamento do case é simples, ele avalia as expressões de seleção, encontra o primeiro das escolhas que corresponde à expressão buscada e executa o respectivo bloco procedural. O case executa apenas um item dele, correspondente ao primeiro “
emphmatch” (F., 2018).

Figura 19 – Declaração do Bloco Case em Verilog



Fonte: F. (2018, p. 213)

2.2.4 Variável Reg

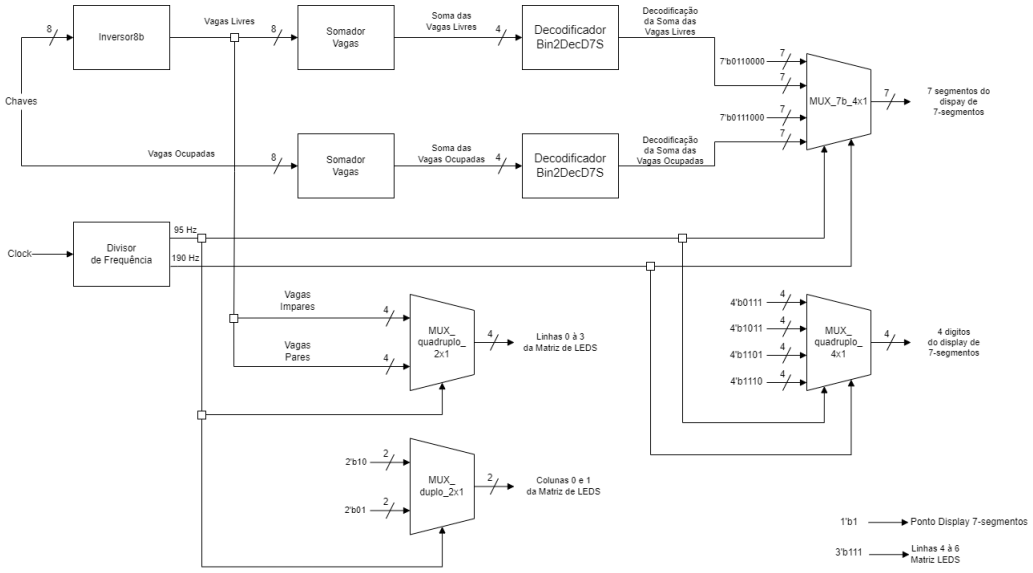
Uma variável do tipo *reg* em verilog não possui relação com flip-flops e registradores encontrados em circuitos sequenciais. A palavra *reg*, para o verilog são registros de armazenamento, então variáveis *reg* podem ser usadas para modelar a saída de circuitos sequenciais e combinacionais. Uma variável *reg* pode conter um bit ou um vetor de bits (F., 2018). “É muito triste que os designers do Verilog não usaram uma palavra chave mais adequada, algo como var ou bitvar” (F., 2018, p. 186). Contudo, um *reg* em verilog é sintetizado como um registrador ou não, dependendo da forma como ele é utilizado no código, contudo em circuitos sequenciais eles são sintetizados como registradores, que são constituídos de flip-flops.

3 CARACTERIZAÇÃO DA SOLUÇÃO

Com o intuito de atender aos requisitos apresentados, foi desenvolvido um circuito digital como protótipo de sistema de gestão para estacionamentos. Para implementação do circuito, foi utilizado o ambiente de desenvolvimento integrado da Intel, Quartus II, juntamente com a linguagem de descrição de hardware, Verilog, em seu modelo estrutural e comportamental. Já para a síntese da implementação, foi utilizado o Kit de Desenvolvimento LEDS-CPLD, composto pelo CPLD (*complex programmable logic device*, ou dispositivo lógico complexo programável) pertencente à família MAX II, modelo EPM240T100C5N, e a placa de componentes LEDS-CPLD.

A solução desenvolvida consiste sistema digital composto por circuitos sequenciais e combinacionais, dentre eles pode-se citar somadores, decodificadores, multiplexadores e flip-flops, como divisores de frequência, devidamente interligados como mostrado na Figura 20. As chaves, que correspondem às vagas, estão conectadas diretamente em um *SomadorVagas*, que calcula a quantidade de vagas ocupadas, e um *Inversor8b*, que inverte o estado das chaves. O clock está conectado ao *DivisorFrequencia*, que divide a frequência do clock da placa (50 Mhz) até chegar a aproximadamente 95 Hz, na primeira saída, e 190 Hz, na segunda saída. A saída do *Inversor8b* está conectada a outro *SomadorVagas*, que dessa vez calcula a quantidade de vagas livres, e também ao *MUX_quadruplo_2x1*, que decide quais linhas da matriz de LEDs estão acesas, com base na primeira saída do *DivisorFrequencia*. A saída de cada *SomadorVagas* é ligada a um *DecodificadorBin2DecD7S* diferente, no qual o valor obtido é traduzido para o formato do display de 7-segmentos. A saída de cada decodificador está conectada a um *MUX_7b_4x1*, que seleciona qual número vai ser exibido no display de acordo com as duas saídas do *DivisorFrequencia*. O *MUX_quadruplo_4x1* recebe como entrada as duas saídas do *DivisorFrequencia*, e as usa para selecionar qual dígito do display está ligado. Já o *MUX_duplo_2x1* recebe como entrada a primeira saída do *DivisorFrequencia*, e a usa para selecionar qual coluna da matriz de LEDs está ligada. Uma melhor descrição das partes do circuito será feita nas seções seguintes.

Figura 20 – Diagrama de Blocos da Solução



Fonte: Os autores

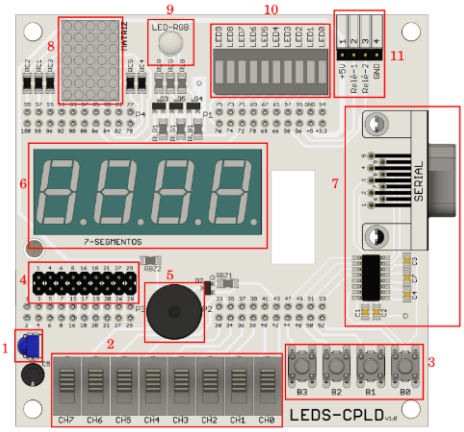
3.1 INTERFACES DE ENTRADA E SAÍDA

Referente às interfaces de entrada do circuito, foi utilizado o conjunto de oito chaves presentes na Placa LEDS-CPLD, indicadas pelo número 2 na Figura X. Optou-se por utilizá-las pois com as oito chaves é possível representar cada uma das oito vagas do estacionamento. Desse modo, quando uma das chaves está na posição para cima, ela gera nível lógico alto nas entradas

do CPLD, sinalizando que a vaga está ocupada, o contrário também é válido. A atribuição das chaves às vagas foi feita da esquerda para direita, deste modo a primeira chave da esquerda corresponde a vaga 1 do estacionamento e assim sucessivamente.

Já no que diz respeito às interfaces de saída, foram utilizadas o Display de 7-segmentos e a Matriz de LEDs, também presentes na Placa LEDS-CPLD, indicadas pelos números 6 e 8 respectivamente, na Figura X. O Display de 7-segmentos foi utilizado para exibição distinta das quantidades de vagas livres e ocupadas. Para tal, o display foi dividido em dois conjuntos, os dois dígitos mais à esquerda, que foram utilizados para mostrar a quantidade de vagas livres, e os dois dígitos mais à direita, que foram utilizados para mostrar a quantidade de vagas ocupadas. Em ambos conjuntos, o primeiro dígito foi utilizado para indicar o que está sendo mostrado. Sendo no primeiro, um "E" de *empty* (vazio) para as vagas livres, e no segundo um "F" de *full* (cheio) para as vagas ocupadas. Já a matriz de LEDs foi utilizada para representação do mapa de vagas do estacionamento. Para tal, foram utilizadas as duas primeiras colunas e as quatro primeiras linhas da matriz. A atribuição das LEDs da matriz as vagas foram feitas da esquerda para direita, de cima para baixo, deste modo a LED localizada na primeira coluna e primeira linha corresponde a vaga 1 do estacionamento e assim sucessivamente.

Figura 21 – Placa LEDS-CPLD



Fonte: LEDS (2020, p. 4)

3.2 SOMA DAS VAGAS LIVRES E OCUPADAS

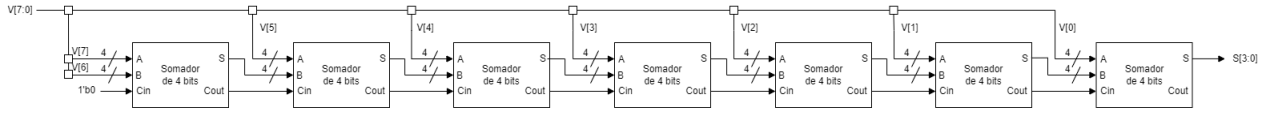
Para que possam ser exibidas as quantidades de vagas livres e ocupadas do estacionamento no display de 7-segmentos é necessário uma série de etapas, sendo a primeira, realizar a soma das vagas livres e das ocupadas. Como visto na seção anterior, optou-se por utilizar as chaves para representar quando uma vaga está ocupada ou desocupada, logo para obter a soma das vagas livres e das ocupadas, basta descobrir quantas vagas estão em nível lógico alto e quantas estão em nível lógico baixo. Para tal foi feito o módulo *SomadorVagas*, mostrado na Figura 22.

O módulo *SomadorVagas* possui como entrada um conjunto de oito bits, cujo qual foi dado o nome de *V*. O módulo é composto por sete somadores de quatro bits em cascata. Nos

quais cada um, com exceção do primeiro, recebe um dos bits do conjunto V como entrada *A*, a saída *S* do somador anterior como entrada *B* e a saída *Cout* do anterior como entrada *Cin*. O primeiro somador, diferentemente dos outros, recebe como entradas *A* e *B*, os dois bits mais significativos de *V* respectivamente e como *Cin* o valor 0. Como saída final do módulo, tem-se a soma dos bits que compõem o conjunto *V*, utilizando o sistema de numeração binário.

O módulo *SomadorVagas* é utilizado duas vezes, sendo a primeira para somar as vagas ocupadas e a segunda para somar as vagas livres. Para somar as vagas ocupadas, passa-se como entrada *V*, o conjunto de bits que armazenam o estado das oito chaves. Já para as vagas livres passa-se como entrada *V*, o mesmo conjunto com os valores lógicos invertidos. Para essa inversão, faz-se o uso do módulo *Inversor8b*. O módulo *Inversor8b* recebe como entrada um conjunto de oito bits e tem como saída a entrada invertida.

Figura 22 – Diagrama do módulo *SomadorVagas*

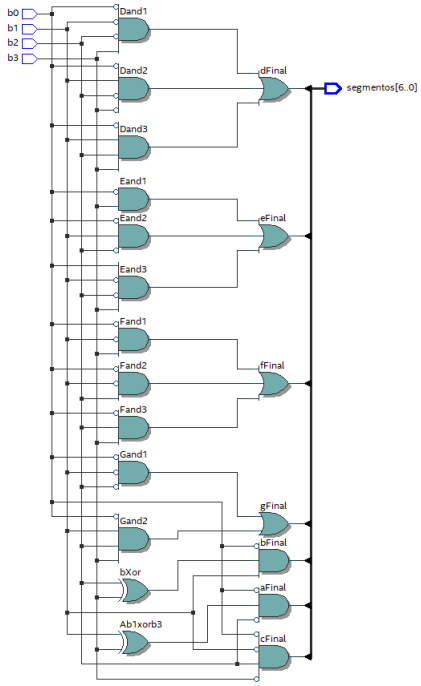


Fonte: Os autores

3.3 DECODIFICAÇÃO PARA O DISPLAY DE 7-SEGMENTOS

Como visto na seção anterior, através do uso do módulo *SomadorVagas* é possível obter o número de vagas livres e ocupadas, porém esses números se dão através de conjuntos de quatro bits, utilizando o sistema de numeração binário. Para que esses números possam ser exibidos através dos dígitos do display de 7-segmentos faz-se necessário realizar uma tradução do número, no formato em questão, para o formato 7-segmentos. Para tal foi desenvolvido o módulo *DecodificadorBin2DecD7S*, presente na Figura 23.

Figura 23 – RTL do módulo *DecodificadorBin2DecD7S*



Fonte: Os autores

O módulo *DecodificadorBin2DecD7S* possui quatro entradas, *b0, b1, b2, b3*, sendo *b0* o bit mais significativo e *b3* o menos significativo. Já como saída, o módulo possui um conjunto de sete bits, denominado *segmentos* que corresponde aos sete segmentos do display de 7-segmentos, sendo o bit mais significativo do conjunto o segmento A e o bit menos significativo o segmento G. Os bits deste conjunto são gerados a partir de portas lógicas associadas, seguindo as seguintes expressões lógicas:

$$A = \overline{b0}.b2(b1 \oplus b3)$$

$$B = \overline{b0}.b1(b2 \oplus b3)$$

$$C = \overline{b0}.\overline{b1}.b2.\overline{b3}$$

$$D = \overline{b0}.\overline{b1}.\overline{b2}.b3 + \overline{b0}.b1.\overline{b2}.\overline{b3} + \overline{b0}.b1.b2.b3$$

$$E = \overline{b0}.b3 + \overline{b0}.b1.\overline{b2} + b0.\overline{b1}.\overline{b2}.b3$$

$$F = \overline{b0}.\overline{b1}.b3 + \overline{b0}.\overline{b1}.b2 + \overline{b0}.\overline{b2}.b3$$

$$G = \overline{b0}.\overline{b1}.\overline{b2} + \overline{b0}.b1.b2.b3$$

Essas expressões lógicas foram obtidas através da tabela verdade do circuito, presente na Figura 24, utilizando álgebra booleana, através da soma de produtos e do Mapa de Karnough, os mapas de Karnaugh resultantes estão representados na Figura 25, porém não foi feito o mapa para todos os segmentos, os demais foram obtidos através da simplificação das expressões obtidas utilizando a soma de produtos. A tabela verdade feita mostra o estado necessário de cada um

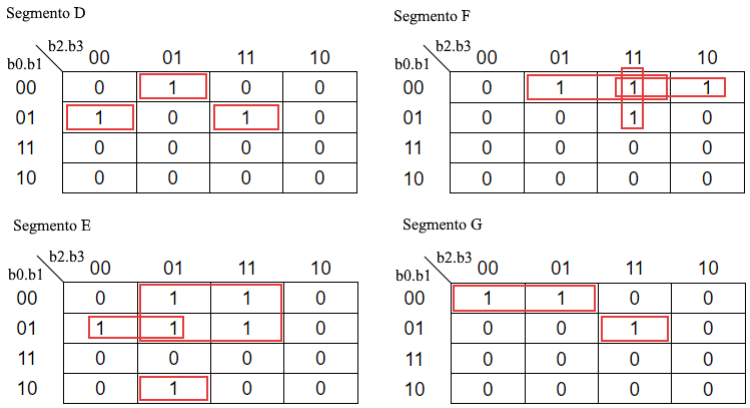
dos sete segmentos para mostrar o respectivo número em binário no display de 7-segmentos. É válido pontuar que é passado o nível lógico baixo para os LEDs que deseja-se ter ligado já que o display utilizado é anodo comum.

Figura 24 – Tabela verdade do módulo *DecodificadorBin2DecD7S*

Entradas				**Saídas**						
b0	b1	b2	b3	A	B	C	D	E	F	G
0	0	0	0	0	0	0	0	0	0	1
0	0	0	1	1	0	0	1	1	1	1
0	0	1	0	0	0	1	0	0	1	0
0	0	1	1	0	0	0	0	1	1	0
0	1	0	0	1	0	0	1	1	0	0
0	1	0	1	0	1	0	0	1	0	0
0	1	1	0	0	1	0	0	0	0	0
0	1	1	1	0	0	0	1	1	1	1
1	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	1	0	0
1	0	1	0	x	x	x	x	x	x	x
1	0	1	1	x	x	x	x	x	x	x
1	1	0	0	x	x	x	x	x	x	x
1	1	0	1	x	x	x	x	x	x	x
1	1	1	0	x	x	x	x	x	x	x
1	1	1	1	x	x	x	x	x	x	x

Fonte: Os autores

Figura 25 – Mapas de Karnaugh dos segmentos *D, E, F* e *G*



Fonte: Os autores

O módulo *DecodificadorBin2DecD7S* é utilizado duas vezes, sendo a primeira para decodificar o valor da soma das vagas livres e a segunda para decodificar o valor da soma das vagas ocupadas. Para decodificar o valor da soma das vagas livres, o decodificador recebe como entrada cada um dos bits da soma obtida pelo *SomadorVagas* que soma as vagas livres. Já para

decodificar o valor da soma das vagas ocupadas, o segundo decodificador recebe como entrada cada um dos bits da soma obtida pelo *SomadorVagas* que soma as vagas ocupadas.

3.4 MULTIPLEXAÇÃO DAS INTERFACES DE SAÍDA

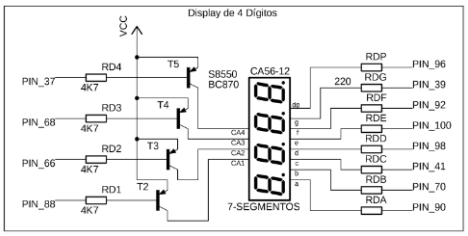
Como já falado na primeira seção utiliza-se, para exibição dos dados obtidos, o display de 7-segmentos e a matriz de LEDs. Porém o uso dessas interfaces não se dá de maneira simples, visto que cada uma funciona de uma forma.

3.4.1 Multiplexação do Display de 7-segmentos

O display de 7-segmentos utilizado é composto por quatro dígitos multiplexados, como mostrado na 26. Ou seja, todos os dígitos do display recebem as mesmas entradas referentes aos sete segmentos e ao ponto. O que dificulta a exibição de dados diferentes em cada um dos dígitos do display. As entradas em questão se encontram representadas na 26 pelas letras *a, b, c, d, e, f, g* e *dp*. Além dessas entradas, o display recebe mais quatro, *CA4, CA3, CA2* e *CA1*, que correspondem ao estado de cada um dos quatro dígitos do display, se está ligado ou desligado. Desse modo é possível fazer uso dessas quatro últimas entradas para manter um único dígito exibindo uma determinada informação, apenas desativando os demais.

O efeito da persistência da visão se dá por conta que os olhos humanos são incapazes de perceber a alteração de um sinal luminoso com frequência acima de 24 Hz. Porém com 24 Hz ainda é possível perceber a cintilação do sinal o que torna necessário a utilização de uma frequência maior, de no mínimo 48 Hz. (LIMA; VILLAçA, 2012). Seguindo esse conceito, foi utilizado o módulo *MUX_quadruplo_4x1* para alternar entre os quatro dígitos do display e o módulo *MUX_7b_4x1* para alternar entre o que será exibido no respectivo dígito.

Figura 26 – Esquema elétrico do display de 7 segmentos



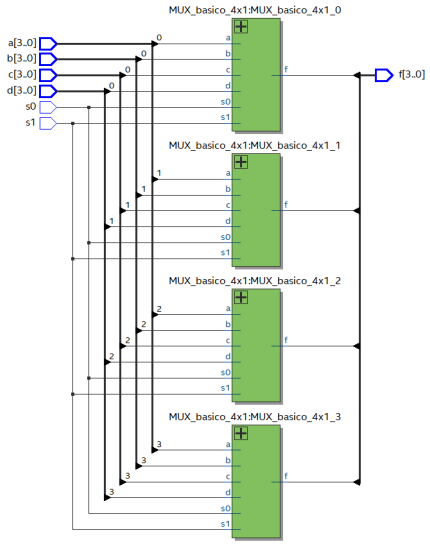
Fonte: LEDS (2020, p. 4)

O módulo *MUX_quadruplo_4x1* é um multiplexador de quatro bits com quatro entradas selecionáveis, *a, b, c* e *d*, ou seja cada uma das entradas é um conjunto de quatro bits, e duas entradas seletoras, *s0* e *s1*. Já como saída final o módulo tem o conjunto *f* de quatro bits, que corresponde exatamente a entrada selecionada a partir das entradas seletoras. Como mostrado na Figura 27, o módulo é composto por quatro multiplexadores básicos de quatro entradas. O primeiro multiplexador básico recebe como entradas o bit menos significativo de cada um dos

conjuntos de quatro bits do multiplexador de quatro bits, o segundo recebe como entradas o segundo bit menos significativo de cada um dos conjuntos do multiplexador de quatro bits e assim sucessivamente. Ambos multiplexadores básicos recebem as mesmas entradas seletoras do multiplexador de quatro bits.

O módulo *MUX_quadruplo_4x1* é utilizado para selecionar qual dígito do display estará ligado. Para isso utiliza-se como entradas *a, b, c e d* os seguintes conjuntos de quatro bits 0111, 1011, 1101 e 1110 respectivamente. Já como entradas *s0* e *s1* utiliza-se a segunda e a primeira saída do módulo *DivisorFrequencia* respectivamente.

Figura 27 – RTL do módulo *MUX_quadruplo_4x1*

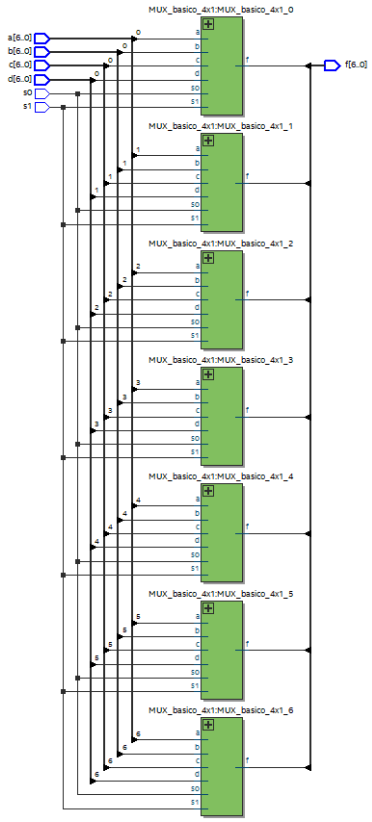


Fonte: Os autores

O módulo *MUX_7b_4x1* é um multiplexador de sete bits com quatro entradas selecionáveis, *a, b, c e d*, ou seja cada entrada é um conjunto de sete bits, e duas entradas seletoras, *a, b, c e d*. Já como saída final o módulo tem o conjunto de sete bits *f*, que corresponde exatamente a entrada selecionada a partir das entradas seletoras. Como mostrado na Figura 28, o módulo é composto por sete multiplexadores básicos de quatro entradas. O primeiro multiplexador básico recebe como entradas o bit menos significativo de cada um dos conjuntos de sete bits do multiplexador de sete bits, o segundo recebe como entradas o segundo bit menos significativo de cada um dos conjuntos do multiplexador de sete bits e assim sucessivamente. Ambos multiplexadores básicos recebem as mesmas entradas seletoras do multiplexador de sete bits.

O módulo *MUX_7b_4x1* é utilizado para selecionar qual dígito do display estará ligado. Para isso utiliza-se como entradas *a e c* os seguintes conjuntos de sete bits 0110000 e 0111000 respectivamente, que representam justamente as letras “E” e” F” no formato do display de 7-segmentos. Já a entrada *b* é a saída do decodificador que traduz o total de vagas livres. E a entrada *d* é a saída do decodificador que traduz o total de vagas ocupadas. Já como entradas *s0 e s1* utiliza-se a segunda e a primeira saída do módulo *DivisorFrequencia* respectivamente.

Figura 28 – RTL do módulo *MUX_7b_4x1*

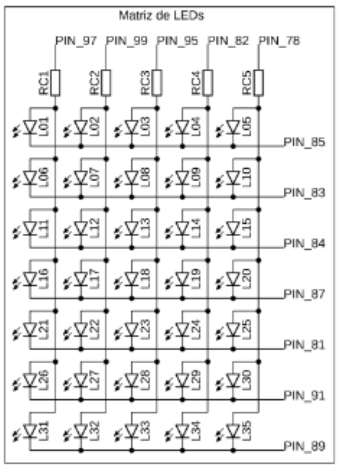


Fonte: Os autores

3.4.2 Multiplexação da Matriz de LEDs

A matriz de LEDs utilizada é composta por trinta e cinco LEDs e possui doze entradas, sendo cinco para as colunas e sete para as linhas, como mostra a Figura 29. Cada coluna tem os ânodos dos seus LEDs interligados e cada linha tem os cátodos interligados. Assim as colunas são ativadas por nível lógico alto e as linhas por nível lógico baixo. Apesar de esse acionamento por linha e coluna proporcionar uma economia de pinos, ele gera outro problema. Em alguns cenários não é possível acender simultaneamente certas LEDs sem apagar outras, o que dificulta a exibição do mapa de vagas do estacionamento. É possível contornar esse problema se for adotada uma alternância entre as colunas, assim quando uma coluna estiver ligada, as demais estarão desligadas. Essa alternancia funciona seguindo os mesmo conceitos sobre persistência da visão, abordados na subseção anterior. Para tal foi utilizado dois multiplexadores, o módulo *MUX_duplo_2x1*, que seleciona qual coluna estará ligada, e o módulo *MUX_quadruplo_2x1* que seleciona quais LEDs da coluna estarão ligados.

Figura 29 – Esquema elétrico da matriz de LEDs

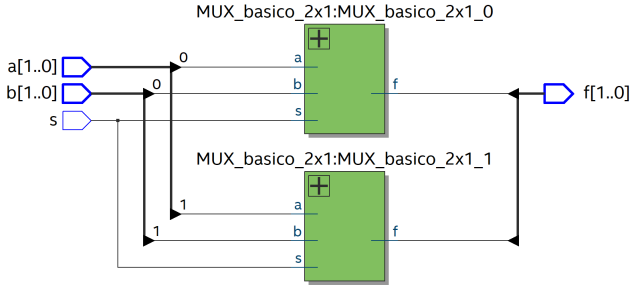


Fonte: LEDS (2020, p. 4)

O módulo *MUX_duplo_2x1* é um multiplexador de dois bits com duas entradas selecionáveis, *a* e *b*, ou seja cada uma das entradas é um conjunto de dois bits, e uma entrada seletora, *s*. Já como saída final, o módulo tem o conjunto *f* de dois bits, que corresponde exatamente a entrada selecionada a partir das entradas seletoras. Como mostrado na Figura 30, o módulo é composto por dois multiplexadores básicos de duas entradas. O primeiro multiplexador básico recebe como entradas o bit menos significativo de cada um dos conjuntos de bits do multiplexador de dois bits, já o segundo recebe como entradas o bit mais significativo de cada um dos conjuntos de bit do multiplexador de dois bits. Ambos multiplexadores básicos recebem a mesma entrada seletora do multiplexador de quatro bits.

O módulo *MUX_duplo_2x1* é utilizado para selecionar qual coluna estará ligada. Para isso utiliza-se como entradas *a* e *b* os seguintes conjuntos de dois bits 10 e 01 respectivamente. Já como entrada *s* utiliza-se a segunda saída do módulo *DivisorFrequencia*.

Figura 30 – RTL do módulo *MUX_duplo_2x1*

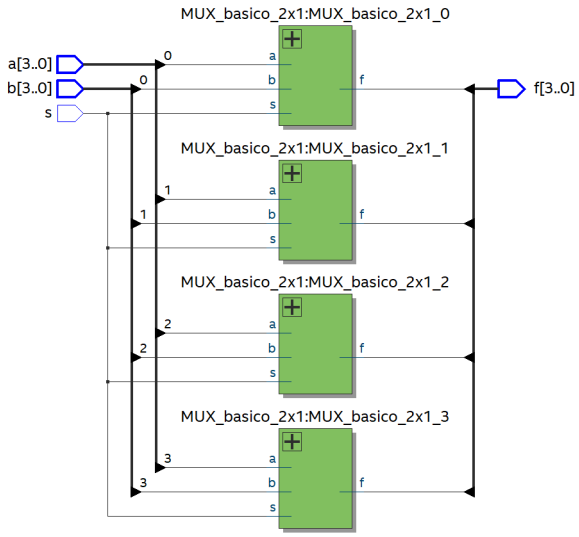


Fonte: Os autores

O módulo *MUX_quadruplo_2x1* é um multiplexador de quatro bits com duas entradas selecionáveis, *a* e *b*, ou seja cada uma das entradas é um conjunto de quatro bits, e uma entrada seletora, *s*. Já como saída final, o módulo tem o conjunto *f* de quatro bits, que corresponde exatamente a entrada selecionada a partir das entradas seletoras. Como mostrado na Figura 31, o módulo é composto por quatro multiplexadores básicos de duas entradas. O primeiro multiplexador básico recebe como entradas o bit menos significativo de cada um dos conjuntos de bits do multiplexador de quatro bits, já o segundo recebe como entradas o segundo bit menos significativo de cada um dos conjuntos de bit do multiplexador de quatro bits e assim sucessivamente. Ambos multiplexadores básicos recebem a mesma entrada seletora do multiplexador de quatro bits.

O módulo *MUX_quadruplo_2x1* é utilizado para selecionar quais LEDs da coluna estarão ligados. Para isso utiliza-se como entrada *a* o conjunto de quatro bits referente às vagas ímpares, invertido. Esse conjunto é obtido através dos bits ímpares do conjunto de saída do módulo *Inversor8b*. Já a entrada *b* consiste no conjunto de quatro bits referente às vagas pares, invertido. Esse conjunto é obtido de maneira semelhante ao anterior, porém dessa vez é utilizado os bits pares do conjunto de saída do módulo. Já como entrada *s* utiliza-se a segunda saída do módulo *DivisorFrequencia*.

Figura 31 – RTL do módulo *MUX_quadruplo_2x1*



Fonte: Os autores

3.5 FREQUÊNCIA DE ALTERNÂNCIA DAS INTERFACES DE SAÍDA

Como visto na seção anterior, para que as informações sejam exibidas corretamente nas interfaces de saída é necessário que os multiplexadores alterem suas saídas em uma determinada frequência. No caso dos multiplexadores responsáveis pela exibição de informações no display de 7-segmentos a frequência utilizada foi de aproximadamente 190 Hz. Esse valor foi escolhido por conta da necessidade de cada dígito do display ser ligado a uma frequência mínima de aproximadamente 48 Hz, logo como são quatro a frequência necessária seria próxima de 192 Hz, assim cada dígito

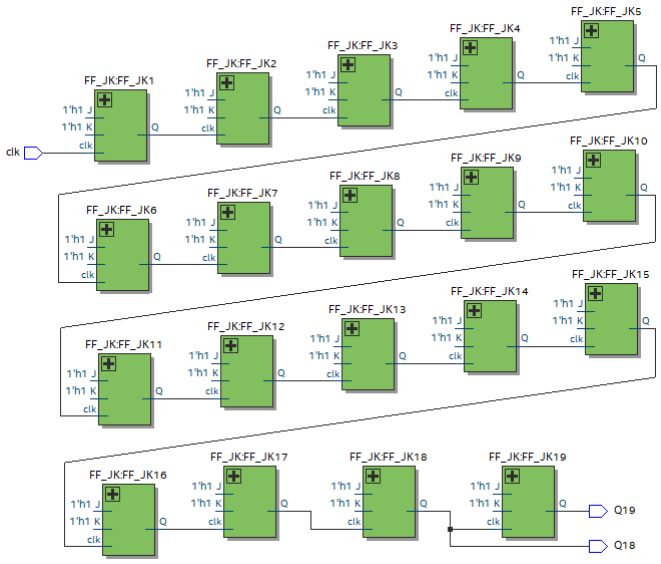
ficaria ligado durante aproximadamente 5 ms. Já em relação aos multiplexadores utilizados para exibição de informações na matriz de LEDs a frequência utilizada foi de aproximadamente 95 Hz, a obtenção desse valor se deu de maneira similar a anterior. Como a alternância é entre duas colunas, faz-se necessário uma frequência próxima de 96 Hz, já que cada coluna precisa estar ligada a uma frequência de aproximadamente 48 Hz, assim cada coluna ficaria ligada durante aproximadamente 10 ms.

Para gerar essas frequências foi feito o uso do oscilador, da placa CPLD contida no kit de desenvolvimento LEDS-CPLD. O oscilador produz uma onda quadrada de frequência igual a 50 MHz. Apesar de ser uma frequência bem acima do mínimo recomendado, ela não pode ser diretamente usada nos multiplexadores, pois, por ser demasiadamente grande, proporciona um grande atraso de propagação nos multiplexadores, que resulta na exibição incorreta das informações nas interfaces. Então faz-se necessário dividir essa frequência até alcançar algo próximo às frequências citadas, 192 Hz e 96 Hz. Para tal foi feito o uso do módulo *DivisorFrequencia* que possibilita obter as frequências recomendadas, a partir da entrada do oscilador, 50 MHz.

O módulo *DivisorFrequencia*, como visto na Figura 32, consiste em dezenove flip-flops JK, disparados por borda de descida, ligados em cascata. O módulo possui como entrada um clock externo, denominado *clk*. O primeiro flip-flop recebe como entrada de clock o clock externo, *clk*, e gera um clock com metade da frequência do da entrada, já o segundo, recebe como entrada de clock a saída do anterior, e gera um clock com metade da frequência dele, e assim sucessivamente. Isso acontece porque todos os flip-flops estão em modo de comutação, ou seja recebem como entradas J e K nível lógico alto, assim fazendo com que a cada borda de descida do clock de entrada o seu sinal de saída inverta, é válido pontuar que todos os flip-flops iniciam com suas saídas como sinal lógico baixo. Ao final, tem-se como saídas desse módulo o clock *Q19*, que corresponde ao clock externo, *clk*, dividido por 2^{19}, e o clock *Q18*, que corresponde ao clock externo, *clk*, dividido por 2^{18}.

O módulo *DivisorFrequencia* é utilizado recebendo com a entrada *clk* o clock do oscilador, com frequência de 50 MHz e tem como saídas *Q19*, um clock com frequência de aproximadamente 95 Hz, e *Q18*, um clock com frequência de aproximadamente 190 Hz.

Figura 32 – RTL do módulo *DivisorFrequencia*



Fonte: Os autores

Os flip-flops JK são implementados a partir do módulo *FF_JK*. O módulo possui três entradas, o clock, denominado *clk*, e o J e o K que controlam o estado lógico do flip-flop. E como saída Q que representa o estado lógico do flip-flop. Diferentemente do restante do circuito, que foi implementado utilizando verilog estrutural, esse módulo foi implementado utilizando verilog comportamental. Para tal foi utilizado o código presente na Figura 33, nele pode-se notar, na linha 3, que a saída Q é do tipo registrador, ou seja ela age como memória já que se trata de um flip-flop. Já nas linhas, 5, 6 e 7 é definido o valor inicial de Q, que significa o nível lógico armazenado inicialmente no flip-flop. Já entre as linhas 9 e 16 é verificado a cada borda de descida da entrada *clk* qual o estado das variáveis de controle J e K. De acordo com esses estados, o valor armazenado pelo flip-flop é alterado ou mantido.

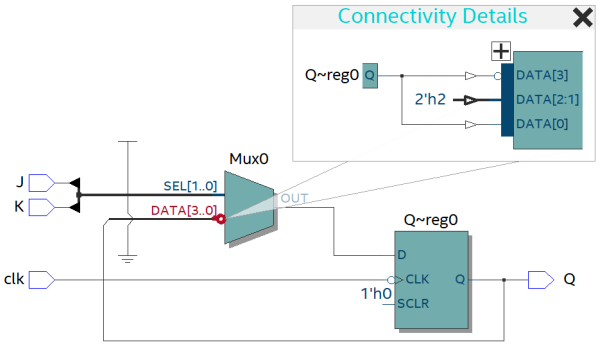
Figura 33 – Código em Verilog Comportamento do módulo *FF_JK*

```
1  module FF_JK (clk, J, K, Q);
2     input clk, J, K;
3     output reg Q;
4
5     initial begin
6     Q = 1'b0;
7     end
8
9     always @ (negedge clk) begin
10       case ({J,K})
11          2'b00: Q <= Q;
12          2'b01: Q <= 0;
13          2'b10: Q <= 1;
14          2'b11: Q <= ~Q;
15       endcase
16    end
17
18 endmodule
```

Fonte: Os autores

Apesar de se utilizar um flip-flop tipo JK como base para implementação, o Quartus II quando compila o código gera um circuito com base em um flip-flop tipo D associado a um multiplexador de 1b 4x1, como mostra a Figura 34. Nesse circuito o multiplexador possui as entradas a serem selecionadas Q, 0, 1 e Q negada e as entradas seletoras J e K, e tem sua saída ligada a entrada D do flip-flop tipo D. Como a entrada D controla o estado lógico do flip-flop tipo D, de acordo com as entradas J e K eu realizo o mesmo controle, através do multiplexador.

Figura 34 – RTL do módulo *FF_JK*



Fonte: Os autores

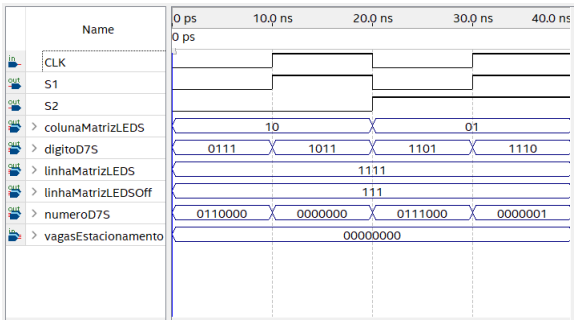
4 CASOS DE TESTE

A fim de verificar o funcionamento do circuito feito, foram realizados três testes que verificam o funcionamento do circuito em três cenários diferentes. Os testes foram realizados tanto através do Waveform do quartus II, quanto na placa do kit de desenvolvimento LEDS-CPLD. É válido pontuar que para os testes do Waveform o módulo *DivisorFrequencia* foi modificado para

dividir a frequência apenas uma vez, logo sua primeira saída, *S1* corresponde a frequência próprio clock do oscilador e a sua segunda, *S2*, corresponde a metade da primeira. Essa modificação foi feita por conta que não é possível visualizar a simulação utilizando frequências muito baixas, como as realmente utilizadas.

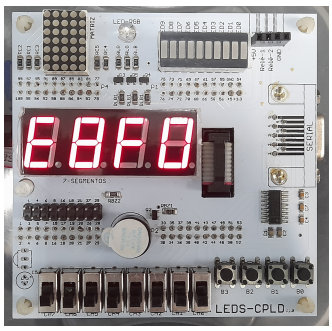
O primeiro teste realizado, presente na Figura 35, verifica se as saída estão corretas caso todas as chaves estejam para baixo, ou seja sinalizando nível lógico baixo. O estados das chaves é mostrado pelo conjunto de bits presentes em *vagasEstacionamento*. Em relação ao display de 7-segmentos, como observado, quando *S1* e *S2* estão em nível lógico baixo, somente o primeiro dígito está ligado, mostrado em *digitoD7S*, e o dado exibido no display é a letra "E" decodificada, mostrado em *numeroD7S*. Já quando *S1* está em nível lógico alto e *S2* baixo, o dígito ligado é o segundo e o dado exibido é o número oito. E quando *S1* está em nível lógico baixo e *S2* alto, o dígito ligado é o terceiro e o dado exibido é a letra "F". Ao final, quando *S1* e *S2* estão em nível lógico alto, o dígito ligado é o último e o dado exibido é o número zero. Já em relação a matriz de LEDs, quando *S2* está em nível lógico baixo, a coluna ligada, mostrada em *colunaMatrizLEDS*, é a primeira, e as linhas ligadas, mostradas em *linhaMatrizLEDS*, são nenhuma, já que não há nenhuma vaga ocupada. E quando *S2* está em nível lógico alto, a coluna ligada é a segunda e as linhas continuam desligadas. Esse teste em questão, se mostraria na placa, se usada as frequências corretas de modo semehante ao da Figura 36.

Figura 35 – Waveform do teste 1



Fonte: Os autores

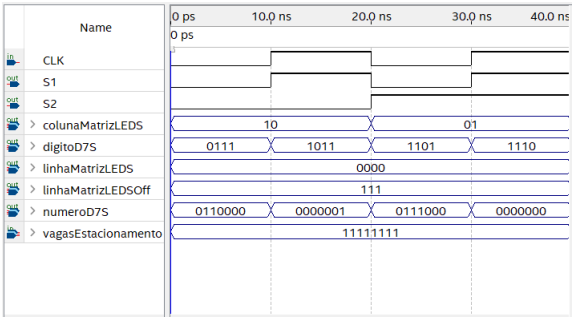
Figura 36 – Teste 1 na placa



Fonte: Os autores

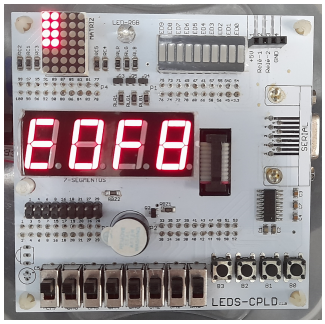
O segundo teste realizado, presente na Figura 37, verifica se todas as saídas estão corretas quando todas as chaves estão para o alto, sinalizando que todas as vagas estão ocupadas. Pode-se se observar em relação ao display que neste teste os valores de *numeroD7S* quando *S1* e *S2* estão em nível lógico alto e baixo respectivamente e quando estão em nível lógico baixo respectivamente se comparados com o teste anterior, estão trocados, o que faz sentido, já que todas as vagas foram ocupadas. Já em relação a matriz observa-se que todas as linhas de *linhaMatrizLEDs* estão ligadas o que corrobora com o fato de todas as vagas estarem ocupadas. Esse teste em questão, se mostraria na placa, se usada as frequências corretas de modo semelhante ao da Figura 38.

Figura 37 – Waveform do teste 2



Fonte: Os autores

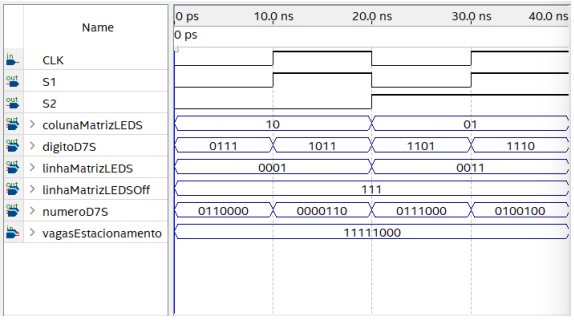
Figura 38 – Teste 2 na placa



Fonte: Os autores

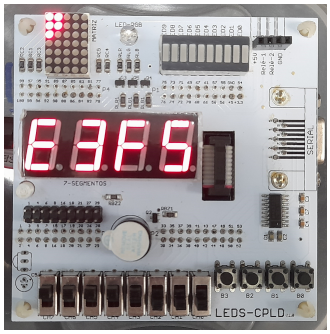
Já o terceiro, presente na Figura 39 verifica se a matriz de LEDs exibe corretamente o mapa de vagas quando uma vaga de uma mesma linha está ocupada e a outra não. Para tal foram postas em nível lógico alto as cinco primeiras chaves e as demais em nível lógico baixo. Como mostrado no waveform, quando a primeira coluna é ligada a segunda é desligada e as linhas ligadas são as das vagas correspondentes à própria coluna, o que se altera quando a segunda coluna é ligada e a primeira é desligada. Assim funcionando corretamente. Esse teste em questão, se mostraria na placa, se usada as frequências corretas de modo semelhante ao da Figura **??**.

Figura 39 – Waveform do teste 3



Fonte: Os autores

Figura 40 – Teste 3 na placa



Fonte: Os autores

5 CONCLUSÃO

Mediante ao que foi falado e aos testes feitos é perceptível que o produto solicitado, um sistema de gerenciamento de vagas, foi desenvolvido e se mostra funcional. Visto que os principais requisitos solicitados, uma interface para simulação das vagas, uma para exibição da quantidade de vagas livres e ocupadas, além de uma para exibição do mapa das vagas, foram alcançados.

Com isso, foi produzido um sistema digital composto por circuitos sequenciais e combinacionais. Assim foi feito o uso de flip-flops, somadores, multiplexadores e decodificadores para concretização do sistema produzido. O sistema em questão, como protótipo, faz uso dos componentes presentes no kit de desenvolvimento LEDS-CPLD como interfaces. Sendo as chaves para simular as vagas, o display de 7-segmentos para exibição da quantidade e a matriz de LEDs para o mapa.

A solução proposta se mostra eficaz já que cumpre com o que foi solicitado, porém é válido pontuar que ainda não se mostra em sua total eficiência. Isso em vista do uso repetido e desnecessário de alguns módulos, como o SomadorVagas e o Decodificador. Esses módulos poderiam ser utilizados apenas uma vez caso fosse feito o uso de um multiplexador para selecionar qual conjunto de vagas seria somado, as livres ou ocupadas. Assim proporcionando uma economia de circuitos lógicos utilizados e se caracterizando como uma possível melhoria.

REFERÊNCIAS

BROWN, S.; VRANESIC, Z. *Fundamentals of Digital Logic with Verilog Design*. 2. ed. USA: McGraw-Hill, Inc., 2007. ISBN 0077211642. Citado na página 9.

CAPUANO, F. G.; IDOETA, I. V. *Elementos de eletronica digital*. 38. ed. [S.l.]: São Paulo: Erica, 2006. Citado na página 9.

ECONOMIASC. *Estacionamentos se tornam a resposta para a mobilidade urbana*. [s.n.], 2019. Disponível em: <https://economiasc.com/2022/03/24/estacionamentos-se-tornam-a-resposta-para-a-mobilidade-urbana/>. Citado na página 1.

F., W. J. *Principles and Practices. Fifth edition with Verilog*. 5th. ed. [S.l.]: Pearson, 2018. ISBN 9780134460093. Citado 3 vezes nas páginas 10, 11 e 12.

IBGE. *Frota de veículos*. [s.n.], 2020. Disponível em: <https://cidades.ibge.gov.br/brasil/pesquisa/22/28120>. Citado na página 1.

LEDS. *Manual do Kit LEDS-CPLD*. [S.l.], 2020. Disponível em: <https://drive.google.com/file/d/168zWlJU0rbnq3q8QJXnrwRY8iO6Ds2xQ/view>. Acesso em: 02 nov. 2022. Citado 3 vezes nas páginas 14, 18 e 21.

LIMA, C. B. de; VILLAçA, M. V. M. *AVR e Arduino: técnicas ed projetos*. 2th. ed. [S.l.]: Edição dos Autores, 2012. ISBN 9788591140015. Citado na página 18.

MANO, M. M.; KIME, C. *Logic and Computer Design Fundamentals*. 4th. ed. USA: Prentice Hall Press, 2007. ISBN 013198926X. Citado 5 vezes nas páginas 2, 3, 4, 5 e 6.

SOUZA, K. *Quais são as cidades com pior trânsito do mundo?* [s.n.], 2020. Disponível em: <https://exame.com/tecnologia/quais-sao-as-cidades-com-o-pior-transito-do-mundo/>. Citado na página 1.

TOCCI, R. J.; WIDMER, N. *Digital Systems: Principles and Applications*. 12th. ed. [S.l.]: Pearson, 1997. ISBN 0137005105. Citado 5 vezes nas páginas 3, 6, 7, 8 e 10.

WAKERLY, J. *Digital Design - Principles and Practices: 3rd Edition*. [s.n.]. Disponível em: <https://books.google.com.br/books?id=qzinBQAAQBAJ>. Citado na página 10.