# SÍNTESE DE CIRCUITOS COM MEMÓRIA EM LÓGICA MULTINÍVEL

**Melitón Apaza Tito**

**CAMPO GRANDE**

**2008**

**UNIVERSIDADE FEDERAL DO MATO GROSSO DO SUL**

**PROGRAMA DE PÓS-GRADUAÇÃO**

**EM ENGENHARIA ELÉTRICA**

# SÍNTESE DE CIRCUITOS COM MEMÓRIA EM LÓGICA MULTINÍVEL

Dissertação submetida à

Universidade Federal de Mato Grosso do Sul

como parte dos requisitos para a

obtenção do grau de Mestre em Engenharia Elétrica.

**Melitón Apaza Tito**

Campo Grande, Dezembro 2008

# SÍNTESE DE CIRCUITOS COM MEMÓRIA EM LÓGICA MULTINÍVEL

Melitón Apaza Tito

"Esta Dissertação foi julgada adequada para obtenção do Título de Mestre em Engenharia Elétrica, Área de Concentração em *Inteligencia Artificial - Teoria e aplicações em Sistemas de Energia*, e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Campo Grande".

Evandro Mazina Martins - Dr.
Prof. DEL/UFMS
Orientador

Luciana Cambraia Leite - Dr.
Prof. DEL/UFMS
Coordenadora do Programa de Pós-Graduação
em Engenharia Elétrica

Banca Examinadora:

Evandro Mazina Martins - Dr.
Prof. DEL/UFMS
Presidente

Ricardo Ribeiro dos Santos - Dr.
Prof. CCET/UCDB

Milton Ernesto Romero Romero - Dr.
Prof. DEL/UFMS

Aos meus pais Teófilo e Victoria.

# AGRADECIMENTOS

Primeiramente agradecer ao meu orientador Prof. Evandro Mazina Martins PhD., por terme orientado e aconselhado no processo de desenvolvimento do meu trabalho de dissertação. Ao Prof. Milton E. Romero Romero PhD. por sua direta colaboração e sugestões que ajudaram na minha dissertação . Também lhe estou agradecido ao Prof. Ricardo Ribeiro dos Santos PhD por seus acertados comentários e sugestões e participar da banca examinadora.

Aos professores: Prof. João O. Pereira Pinto, Ph.D. Profa Luciana Cambraia Leite, Ph.D. Profa. Káthya Silvia Collazos Linares, Ph.D. Prof. Jorge L. Roel Ortiz, Ph.D. E ao pessoal administrativo do departamento DEL Marcira Crispim de Almeida.

Também lhe estou agradecido ao Prof. Marco A. Alavarez por seus conselhos.

E não poderia deixar de agradecer as pessoas que fizeram agradável a minha estadia no Brasil, Wellington Rocha Araújo, Herbert Luque Peralta, Edvaldo F. Freitas Lima, Rafael Nishimura, Marcio Lorenzoni Portella, Ângelo D. Molin Brun, André Luiz Pasquali, Roberto Henrique da Rocha Viana, Susana Guimarães de Paula Potrich, Kelly C. Gutterres de Souza, Edgard J. dos Santos Arinos, Marcelo Maldonado Correia, José E. Montalvan Barbaran, Juliana Xavier Silva.

A minha formação no mestrado não seria possível sem o apoio econômico da CAPES pelo qual estou-lhes agradecido.

Finalmente, eu estou-lhes agradecido aos meus pais, Teófilo e Victoria, que me ensinaram a entender o valor da educação na vida das pessoas, algo pelo qual sempre lhes estarei eternamente agradecido. A meu irmão Eduardo, minhas irmãs: Lucila, Lourdes, Paula, e a meus tios Fortunato e Brígida, minhas primas: Mary, Lucy e Flor pelo seu amor e apoio incondicional.

Resumo da Dissertação apresentada à UFMS como parte dos requisitos necessários para a obtenção do grau de Mestre em Engenharia Elétrica.

# SÍNTESE DE CIRCUITOS COM MEMÓRIA EM LÓGICA MULTINÍVEL

Melitón Apaza Tito

Dezembro/2008

Orientador: Prof. Evandro Mazina Martins, PhD.
Área de Concentração: Inteligencia Artificial - Teoria e Aplicações em Sistemas de Energia.
Palavras-chave: Síntese de Circuitos Seqüenciais MVL, Redução de Estados, Memorias MVL, Lógica de Multiplos Valores.
Número de Páginas: 69

Com o avanço na tecnologia VLSI (Very Large Scale Integration) dos circuitos integrados, tem-se gerado interesse nos circuitos que empregam mais de dois níveis lógicos de sinais discretos. Esses circuitos são chamados, circuitos de múltiplos valores lógicos (MVL) e oferecem um potencial no projeto dos circuitos VLSI, devido a seu potencial para armazenar e transmitir maior quantidade de informação por digito, ou seja, quanto maior a base menor é a quantidade de dígitos necessários para representar um valor.

Assim como nos circuitos digitais binários, os circuitos MVL são baseados numa Álgebra MVL e dividem-se em circuitos combinacionais e seqüenciais. Neste trabalho é proposta a metodologia de síntese para circuitos seqüenciais MVL (com memória) que envolve: 1) a descrição da Álgebra MVL; 2) a síntese de elementos de memória (latch RS, Flip-flop RS, Flip-flop D Flip-flop Máster Slave); 3) definição do Clock MVL; 4) os métodos de simplificação para circuitos seqüenciais; 5) metodologia para a síntese de circuitos seqüenciais MVL apresentado neste trabalho. As simulações mostram a robustez da metodologia proposta.

Abstract of Dissertation presented to UFMS as a partial fulfillment of the requirements for the degree of Master in Electrical Engineering.

# SYNTHESIS OF CIRCUITS WITH MEMORY IN MULTILEVEL LOGIC

Melitón Apaza Tito

December/2008

Advisor: Prof. Evandro Mazina Martins, PhD.
Area of Concentration: Artificial Intelligence - Theory and Applications in Power Systems.
Keywords: Synthesis of Sequential Circuits MVL, Minimization of States, MVL Memories, Multivalued Lógic.
Number of Pages: 69

With the advanced in the technology VLSI (very Large Scale Integration) of the integrated circuits, broad interest has been generated regarding circuits that utilize more than two logical levels for the discrete representation of signals. These circuits are named, logic circuits of multiple values (MVL) and offer a great potential for the design of VLSI, because their capability to store and transmit more information per digit, i.e. the higher the radix the lower the number of needed digits to represent a value.

As with the binary circuits, the MVL circuits are based on an MVL Algebra and are comprised of the combinational and the sequential circuits. This work proposes a method for the synthesis of MVL sequential circuits (with memory) that involve: 1) a description of the generated Algebra MVL; 2) the synthesis of memory elements (latch RS, RS flip-Flop-flip D Flip-flop flop Slave Master); 3) definition of the MVL Clock; 4) the simplification methods to MVL sequential circuits; and 5) methodology for the synthesis for MVL sequential circuits presented in this work. Simulations demonstrate soundness of the proposed methodology.

# Sumário

# Lista de Figuras

# Lista de Tabelas

CAPÍTULO

1

# Introdução

## 1.1 Contextualização

Com o avanço na tecnologia VLSI (*Very Large Scale Integration*) dos circuitos integrados, tem-se gerado interesse em circuitos eletrônicos que empregam mais de dois níveis de sinais discretos. Esses circuitos são chamados circuitos de múltiplos valores lógicos e oferecem um potencial no projeto dos circuitos VLSI [1].

A síntese de circuitos digitais de dois sinais discretos, conhecido como circuitos em lógica binária (*Switching Álgebra*) com base B=2 e domínio D = {0,1}, utiliza técnicas de minimização, como os mapas de Karnaugh, Quine-McCluskey e Petrick [2], permitindo reduzir a área do *chip* utilizado para as trilhas em aproximadamente $70\%$, $20\%$ para o isolamento e os restantes $10\%$ usados para os componentes [3].

A alternativa de projeto e síntese de circuitos digitais com múltiplos valores lógicos (MVL) é definida com base B>2 e domínio D = {0, 1, 2,..., (B-1)}. Esta lógica permite transmitir mais informação por linha de interconexão. Por exemplo, se desejamos transmitir o número 1024, em base binária (B=2) são necessários 11 bits $10000000000_2$, e os números excedentes a um milhão requerem mais de 20 bits, agora na lógica MVL com base quaternária (B=4) sua representação seria com 6 dígitos $100000_4$. Pode-se observar que a menor base tem um maior número de dígitos [3].

O presente trabalho apresenta uma metodologia de síntese para circuitos seqüenciais MVL. Sabe-se que os elementos de memória são constituídos por *latch e Flip-flops*, e que são parte fundamental nos circuitos digitais binários com base (B=2) [4], estes componentes tem três

funcionalidades básicas. Memorizar um valor (*set*), apagar o valor armazenado (*reset*) e manter o valor armazenado (*hold*). Estas três funcionalidades são implementadas para os circuitos de memória MVL assíncrono (*latch*) e síncrono (*flip-flop*) que implica definir o funcionamento do *clock* MVL, e o método de simplificação.

## 1.2 Revisão Bibliográfica

Sistemas digitais binários usam apenas dois símbolos, 0 e 1, para representar todas as informações. Uma pergunta a fazer seria se a representação binária é uma escolha adequada. O mundo real não é binário, a resposta é mais intuitiva sobre a razão para representar a informação em múltiplos níveis. Quando estão envolvidas operações aritméticas como soma ou subtração, o cálculo em sistema decimal iria responder melhor a nossa experiência [1].

A lógica de múltiplos valores é conhecida como a lógica Multiple-Valued, Multi-Valued ou Many-Valued que tem seus inícios na lógica de Lukasiewicz [5], Post e Álgebras de Kleene [6, 7, 8]. Um dos primeiros trabalhos em lógica de múltiplos valores lógicos foi desenvolvido por Lukasiewicz, a diferença da lógica clássica é que ela tem três valores lógicos [5], onde além dos valores verdadeiro e falso fica aberta a possibilidade de um terceiro valor. O seguinte trabalho na mesma linha de pesquisa, é de Emil L. Post, publicou uma Álgebra com completude funcional[1] para qualquer base [9]. Existem muitas representações da Álgebra de Post, que demonstra ser isomorfa [2], isto é um dos motivos pelo qual se projetaram circuitos digitais nesta Álgebra [10].

A mais convincente demonstração do sucesso da aplicação da lógica de múltiplos valores para o projeto do multiplicador de 200 MHz 54x54-b projetado em modo circuito MOS [11]. O projeto de somador completo baseado em lógica de três valores é apresentado no [12], outro trabalho desenvolvido sobre projeto de somador de 7 e 10 valores [13], utilizando uma representação numérica desbalanceada[3] sendo a sua implementação sobre uma tensão de 2.5V. O resultado das simulações é baseado sobre 0.8um CMOS considerando 1.0ns como o tempo de *delay*. Em [14] é descrito a implementação do somador utilizando módulo 7. Composto por 147 transistores. O resultado das simulações sobre a tecnologia de 0,8um CMOS e tensão 5V mostram um *delay* de 7,49ns.

As tentativas de construir circuitos integrados (CI) de múltiplos valores remontam até 1970, a partir dos primeiros trabalhos sobre circuitos integrados em três valores. Os circuitos lógicos de múltiplos valores foram implementados em tecnologias como semicondutor metal-óxido complementar (CMOS), tipo-n Metal-óxido complementar (MOS tipo-n), dispositivo de carga

[1] $http://pt.wikipedia.org/wiki/Completude\_funcional$

[2] O isomorfismo define-se por ser o morfismo (ou seta) $f : X \longrightarrow Y$ que admitem um morfismo inverso $h : Y \longrightarrow X$, uma função injetora e sobrejetora.

[3] Existem duas grandes convenções para a classificação dos valores num sistema de múltiplos valores sobre um conjunto de m valores. O mais comum é 0, 1, 2, ..., m-2, m-1, estendendo a notação binária numa só direção. Este é chamado de desbalanceado (positivos). O segundo caso -r, 1-r, ...,-1, 0, 1,..., r-1, r. Isto é chamado de balanceado.

acoplada (CCD). Das tecnologias aplicadas, a que mostrou grande potencialidade para comercialização, é o CMOS. Vários protótipos de *chips* de circuitos CMOS foram fabricados. Mostrando melhor desempenho em relação aos circuitos binários [14, 11, 15, 16]. Entre outras temos os dispositivos programáveis [17, 18] multiplexer [19], decodificador [20], e a memória em estrutura VLSI [21].

A aplicação em circuitos com memória em lógica de múltiplos valores inclui as memórias *Flash* de 64MB utilizando 4 níveis por célula [22], as memorias DRAM com capacidade de armazenamento de dados de ate 4Gb é obtido com células de multiplos níveis [23]. a utilização de quatro níveis de armazenamento reduz com eficácia o tamanho das células em aproximadamente $50\%$ [24].

Já no projeto de circuitos somadores MVL e multiplicadores, podemos mencionar o trabalhos baseados na Álgebra de Post, como o registrador cíclico utilizando transistores NMOS e PMOS para uma configuração de quatro níveis lógicos, permitindo o desenvolvimento de circuitos lógicos como, contadores, *toggle, switches, shift registers, Flip-flops* [25]. A mesma Álgebra foi utilizada para o projeto de circuito em lógica ternária onde foram apresentados os *flip-flops* e somadores, e posteriormente o teste e sua implementação [26]. E como os resultados das expressões lógicas MVL são muito extensas, é em esta situação que procuramos a assistência do computador implementando programas para analisar os projetos [27].

No trabalho [28] apresenta-se o projeto de um *Flip-flop* tipo SR e D baseados em portas ternárias, o projeto tem uma estrutura similar ao *Flip-flop* binário substituindo as portas NAND/NOR por portas ternárias, chegando à conclusão que o projeto requer menor número de FETS.

## 1.3 Objetivos

O presente trabalho tem como objetivo principal realizar a síntese do circuitos MVL com memória. A idéia fundamental é estender a metodologia já estabelecida na lógica binária para a lógica de múltiplos valores (MVL). A seguir são apresentados os objetivos específicos.

1. Apresentar a Álgebra MVL com seus respectivos operadores e propriedades que permitirão a síntese do circuito MVL com memória.

2. Desenvolver a síntese dos elementos de memória MVL como: o *latch RS*, *Flip-flop RS*, *Flip-flop D*, *Flip-flop Máster Slave*.

3. descrever o comportamento do *Clock* MVL para a síntese dos circuitos MVL com memória.

4. Desenvolver a metodologia para reduzir os estados equivalentes nos circuitos MVL completamente e incompletamente especificados.

5. Apresentar a metodología de síntese do circuito MVL com memória.

## 1.4 Organização do Trabalho

No Capítulo 1 é apresentado uma breve introdução e a revisão bibliográfica onde é mencionado trabalhos que foram desenvolvidos em relação a lógica MVL e a implementação de circuitos digitais baseados nesta lógica. O Capítulo 2 apresenta a descrição da Álgebra em MVL que será utilizada para sintetizar os circuitos seqüenciais com memória. No Capítulo 3 apresenta-se os elementos de memória e a descrição do *Clock* MVL. A simplificação dos circuitos seqüenciais completamente e incompletamente especificados são apresentados no Capítulo 4. No Capítulo 5 é apresentado a metodologia de síntese para circuitos MVL com memória.

CAPÍTULO

2

# Álgebra MVL

## 2.1 Introdução

A Álgebra que descreve o comportamento dos circuitos seqüenciais digitais binários é a Álgebra de chaveamento que foi proposta por Shannon [1] [29], a base desta Álgebra é a **Álgebra Boolena** [2]. Assim é proposta a Álgebra MVL no trabalho [30], como uma extensão da Álgebra de chaveamento e a Álgebra de Post [9], esta Álgebra como outra estrutura matemática, é caracterizada por apresentar questões fundamentais. É definida como um conjunto de elementos pertencentes a um domínio, um conjunto de operadores, e um número de axiomas, ou postulados [31].

- O **domínio** da Álgebra está definido por um conjunto de **elementos** sobre os quais define-se a Álgebra MVL, domínio D=$\{0, 1, 2, ..., (B-1)\}$.
- Um conjunto de **operações** que são efetuadas sobre os elementos MVL. Como os operadores binários, Máximo e AND Estendido $(+, \star^i)$, Mínimo e OR Estendido $(\cdot, +^i)$ e Sucessor, que é um operador unário. Onde $i \epsilon D$.
- Um conjunto de **postulados** e **axiomas** que definem a Álgebra MVL, aceitos como premissa sem demonstração.

[1] Claude Ewood Shannon, em sua tese de mestrado em 1936 apresenta a primeira aplicação da Álgebra Booleana para o projeto de circuitos de chaveamento, titulada *"Simbolic Analysis of Relay And Switching Circuit"*, que foi considerada como uma das melhores teses do século XX.

[2] O nome é em homenagem ao seu criador, o britânico George Bool Gergen. Em sua obra *An Investigation of the Laws of Thought* em 1854. Esse tratado foi fundamental em dar uma explicação sistemática da lógica.

Deve-se considerar que todos os exemplos apresentados neste trabalho são em lógica quaternária.

Para fins de notação utilizaremos as letras minúsculas para os literais e constantes, da forma $a_i$, onde *a* é um literal e *i* é uma constante, B indica a base da representação digital num domínio D. A Álgebra proposta está baseada em cinco operadores. O operador *Sucessor*, *Máximo*, *Produto Estendido*, *Mínimo* e *Soma Estendida*.

## 2.2 Operadores

A metodologia proposta é baseada em um conjunto universal de portas (*MVL*) que permite representar qualquer função MVL. Os operadores Máximo e Sucessor são amplamente utilizados na literatura e o operador AND Estendido, introduzido no trabalho [30], é uma extensão do operador AND da lógica binária. Dado o domínio ordenado de representação numérica D = {0, 1, 2, ..., (B-1)} para a síntese dos circuitos utilizando a Álgebra *MVL* em base *B*, se define um conjunto universal de portas lógicas *MVL* descritas a seguir.

### 2.2.1 Operador Sucessor

Dado $a_1$, $a_2$, $i \in D$, define-se $Suc(a_1) = a_2$, se e somente se $a_2$ é o seguinte de $a_1$. A notação utilizada nesta dissertação é: $Suc(a_1) = a_1^1$; $Suc(Suc(a_1)) = a_1^2$; $a_1 = a_1^0$, etc. Por exemplo: Seja a base *B*=4, e $a_1 = 2$ onde $a_1 \in D$ então $a_1^0 = a_1 = 2$, $a_1^1 = 3$, $a_1^2 = 0$ e $a_1^3 = 1$. Observe que o operador Sucessor é cíclico como se pode ver na Tabela 2.1.

| $a^i$ | $Suc(a_1) = a_1^0$ | $Suc(a_1) = a_1^1$ | $Suc(Suc(a_1)) = a_1^2$ | $Suc(Suc(Suc(a_1))) = a_1^3$ |
|---|---|---|---|---|
| 0 | 0 | 1 | 2 | 3 |
| 1 | 1 | 2 | 3 | 0 |
| 2 | 2 | 3 | 0 | 1 |
| 3 | 3 | 0 | 1 | 2 |

**Tabela 2.1:** Operador *Sucessor*.

### 2.2.2 Operador AND Estendido

Dado $a_1, a_2, i \in D$, define-se $a_1 \star^i a_2 = i$; se e somente se $a_1 = a_2 = i$, caso contrário, $a_1 \star^i a_2 = 0$, como se mostra na Tabela 2.2. A notação utilizada nesta dissertação é: AND Estendido $= \star^i$. Note que $\star^1$ corresponde ao operador *AND* da lógica binária (*B*=2).

| $(a_1 \star^1 a_2)$ | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 |

| $(a_1 \star^2 a_2)$ | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | **2** | 0 |
| 3 | 0 | 0 | 0 | 0 |

| $(a_1 \star^3 a_2)$ | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | **3** |

**Tabela 2.2:** Operador *AND* Estendido: ($\star^i$).

### 2.2.3 Operador OR Estendido

Dado $a_1, a_2, i \in D$, define-se $a_1 +^i a_2 = i$; se e somente se $a_1 = a_2 = i$, caso contrário, $a_1 +^i a_2 = (B-1)$, como mostra a Tabela 2.3. A notação utilizada nesta dissertação é: OR Estendido = $+^i$.

| $(a_1 +^0 a_2)$ | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | **0** | 3 | 3 | 3 |
| 1 | 3 | 3 | 3 | 3 |
| 2 | 3 | 3 | 3 | 3 |
| 3 | 3 | 3 | 3 | 3 |

| $(a_1 +^1 a_2)$ | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 3 | 3 | 3 | 3 |
| 1 | 3 | **1** | 3 | 3 |
| 2 | 3 | 3 | 3 | 3 |
| 3 | 3 | 3 | 3 | 3 |

| $(a_1 +^2 a_2)$ | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 3 | 3 | 3 | 3 |
| 1 | 3 | 3 | 3 | 3 |
| 2 | 3 | 3 | **2** | 3 |
| 3 | 3 | 3 | 3 | 3 |

**Tabela 2.3:** Operador *OR* Estendido: ($+^i$).

### 2.2.4 Operador Máximo

Dado $a_1, a_2 \in D$, se define $Max(a_1, a_2) = a_1$, se e somente se $a_1 \geq a_2$, caso contrário, $Max(a_1, a_2) = a_2$ como na Tabela 2.4. O operador Max denota-se com o símbolo $+$ nesta dissertação. Por exemplo: Seja a base *B*=4, $a_1 = 2$, $a_2 = 3$ então $Max(a_1, a_2) = 3$, ou $a_1 + a_2 = 3$.

| $a_1 + a_2$ | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 0 | 1 | 2 | 3 |
| 1 | 1 | 1 | 2 | 3 |
| 2 | 2 | 2 | 2 | 3 |
| 3 | 3 | 3 | 3 | 3 |

**Tabela 2.4:** Operador *Máximo*: $(+)$.

### 2.2.5 Operador Mínimo

Dado $a_1, a_2 \in D$, se define $Min(a_1, a_2) = a_1$ se e somente se $a_1 \leq a_2$, caso contrário, $Min(a_1, a_2) = a_2$. O operador Min se denota com o símbolo "$\cdot$" nesta dissertação, mostrado na Tabela 2.5. Por exemplo:

Seja a base *B*=4, $a_1 = 2$, $a_2 = 3$ então $Min(a_1, a_2) = 2$, ou $a_1 \cdot a_2 = 2$.

| $a_1 \cdot a_2$ | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 2 | 0 | 1 | 2 | 2 |
| 3 | 0 | 1 | 2 | 3 |

**Tabela 2.5:** Operador *Mín*: $(\cdot)$.

## 2.3 Postulados

Dado um domínio D, que tem ao menos dois elementos distintos e dois operadores binários *Máximo* e *AND Estendido* $(+, \star^i)$,o operador unario *Sucessor* (Suc) e os literais $a_1$ , $a_2$ e $a_3$ elementos $\in$ D e constantes *i, p* $\in$ D.

### 2.3.1 Fechamento

- $a_1 + a_2 \in$ D: É um elemento que pertence ao domínio D, a operação efetuada por $(+)$ recebe o nome de operador *Máximo*.
- $a_1 \star^i a_2$: É um elemento que pertence ao domínio D, a operação efetuada por $(\star^i)$ recebe o nome de operador *AND Estendido*.
- $Suc(a_1)$: É um elemento que pertence ao domínio D, a operação efetuada por $(Suc)$ recebe o nome de operador *Sucessor*.

### 2.3.2 Identidade

Seja $a_1$ um elemento no domínio D. Existe um elemento 0 em D chamado de elemento identidade com relação ao operador *Máximo*.

- $a_1 + 0 = a_1$

### 2.3.3 Elemento Nulo

Seja $a_1$ um elemento no domínio D, elemento superior (B-1). Existe um elemento nulo com relação aos operadores Máximo e AND Estendido.

- $a_1 + (B-1) = (B-1)$
- $a_1 \star^i 0 = 0$

### 2.3.4 Idempotência

A idempotência para o operador Máximo é:

- $a_1 + a_1 = a_1$

### 2.3.5 Comutatividade

A comutatividade com relação ao operador Máximo e AND Estendido é:

- $a_1 + a_2 = a_2 + a_1$
- $a_1 \star^i a_2 = a_2 \star^i a_1$

### 2.3.6 Associatividade

A Associatividade com relação ao operador Máximo e AND Estendido é:

- $a_1 + (a_2 + a_3) = (a_1 + a_2) + a_3$
- $a_1 \star^i (a_2 \star^i a_3) = (a_1 \star^i a_2) \star^i a_3$

### 2.3.7 Complemento

O complemento associado ao operador Máximo e AND Estendido é:

- $a_1^0 + a_1^1 + a_1^2 + ... + a_1^{B-1} = B - 1$
- $a_1^0 \star^i a_1^1 \star^i ... \star^i a_1^{B-1} = 0$

### 2.3.8 Redução

A Redução com relação ao operador AND Estendido é:

- $(a_1^p \star^i a_2^0) + (a_1^p \star^i a_2^1) + ... + (a_1^p \star^i a_2^{(B-1)}) = a_1^p \star^i i$

### 2.3.9 Unicidade

A lei da unicidade associado ao operador AND Estendido é:

- $i \star^i i = i$

### 2.3.10 Involução

A lei de involução com relação ao operador AND Estendido é:

- $a_i^{(B-1)} = a_i$

## 2.4 Funções

O conceito de função é bem conhecido na Álgebra ordinária. De maneira análoga, na Álgebra MVL temos a função MVL f($a_1, a_2, a_3, ..., a_n$), esta função é definida da seguinte maneira. Dadas as variáveis $a_1, a_2, ..., a_n$ em alguma base B, onde cada uma delas representa um valor do domínio D com *n* número de variáveis.

Para o caso em estudo temos a base B=4, domínio D = {0, 1, 2, 3}, se *n* é o número de variáveis onde cada variável tem quatro possíveis valores, existe $4^n$ formas de representar esses valores para *n* variáveis.

Para simplificar uma função utilizam-se os postulados definidos anteriormente como, por exemplo: para obter a igualdade da equação 2.1 foi necessário o Postulado 2.3.8.

$$a_1 \star^i a_2 + a_1^1 \star^i a_2 + a_1^2 \star^i a_2 + a_1^3 \star^i a_2 = i \star^i a_2 \tag{2.1}$$

Deve-se considerar que a implementação mais adequada no circuito é apresentada na Equação 2.2, porque somente é usado a trilha que contenha o valor $a_2$.

$$a_1 \star^i a_2 + a_1^1 \star^i a_2 + a_1^2 \star^i a_2 + a_1^3 \star^i a_2 = a_2 \star^i a_2 \tag{2.2}$$

A propriedade de simplificação não é a lei distributiva como no caso da lógica binária, ainda quando a variável $a_1$ é apresentado em todas as formas do operador *Sucessor*: $a_1^0$, $a_1^1$, $a_1^2$, $a_1^3$. No caso binário a propriedade distributiva permite simplificar como é mostrado na Equação 2.3.

$$(a_1 \cdot a_2)OR(\overline{a_1} \cdot a_2) = a_2 \tag{2.3}$$

Onde $\overline{a_1}$ denota o complemento de $a_1$, OR é o operador OR, e "$\cdot$" o operador AND da lógica binária. O operador AND da lógica binária corresponde ao operador $\star^1$ da Álgebra MVL e $\overline{a_1}$ na lógica binária corresponde a $a_1^1$ na Álgebra MVL com $a_1 \in$ D = {0, 1}. Se aplicar a lei distributiva na Equação 2.1 não se obteria $i \star^i a_2$, se não $a_2$, o qual é falso e por tanto a lei distributiva não é aplicável na Álgebra MVL.

Na metodologia proposta aplicada neste trabalho para circuitos de memória se define a forma canônica da *Soma de Operações Produto Estendida* (SOPE), de maneira análoga à forma canônica (SOP) soma de operações produto da lógica binária. Quer dizer, a função sintetizada está representada em somas de mintermos, onde os mintermos apresentam todos os literais da função a ser sintetizada uma única vez, em alguma forma do operador Sucessor. Então, para sintetizar a função $F(a_1, a_2, ...a_n)$ se opera como apresentou-se na Equação 2.4, onde $n$ é o número de mintermos.

$$F(a_1, a_2, ..., a_n) = F_1(a_1, a_2, ..., a_n) + F_2(a_1, a_2, ..., a_n) + ... + F_{(B-1)}(a_1, a_2, ..., a_n) \tag{2.4}$$

Onde $F_1(a_1, a_2, ..., a_n)$ identifica a função $F(a_1, a_2, ..., a_n)$ com saída em 1; $F_2(a_1, a_2, ..., a_n)$ identifica a função $F(a_1, a_2, ..., a_n)$ com saída em 2; e sucessivamente, até a ultima função representada por $F_{(B-1)}(a_1, a_2, ..., a_n)$ que identifica a função $F(a_1, a_2, ..., a_n)$ com saída em *B*-1. A continuação se apresenta um exemplo.

Por exemplo, dada a Tabela 2.6, o procedimento para achar a função que sintetiza na forma SOPE é como segue.

| $a_1 \setminus a_2$ | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 2 | 2 | 2 | 2 |
| 1 | 0 | 0 | 0 | 2 |
| 2 | 1 | 1 | 1 | 3 |
| 3 | 0 | 1 | 1 | 0 |

**Tabela 2.6:** Exemplo para a síntese da função $F(a_1, a_2), B = 4$.

Então se identificam as funções $F_1(a_1, a_2)$, $F_2(a_1, a_2)$ e $F_3(a_1, a_2)$, que poderíamos representar em três tabelas separadas com suas funções correspondentes:

Para sintetizar $F_1(a_1, a_2)$ se requer o operador $\star^1$ para poder ter a saída igual a 1 utilizamos o operador Sucessor.

Para $a_1 = 3$, $a_1^2 = 1$ e $a_2 = 1$, $a_2^0 = 1$, então, $a_1^2 \star^1 a_2^0 = 1$,

para $a_1 = 3$, $a_1^2 = 1$ e $a_2 = 2$, $a_2^3 = 1$ então, $a_1^2 \star^1 a_2^3 = 1$,

para $a_1 = 2$, $a_1^3 = 1$ e $a_2 = 0$, $a_2^1 = 1$ então, $a_1^3 \star^1 a_2^1 = 1$,

para $a_1 = 2$, $a_1^3 = 1$ e $a_2 = 1$, $a_2^0 = 1$ então, $a_1^3 \star^1 a_2^0 = 1$,

para $a_1 = 2$, $a_1^3 = 1$ e $a_2 = 2$, $a_2^3 = 1$ então, $a_1^3 \star^1 a_2^3 = 1$,

Obtendo-se a função $F_1$ mostrado na Equação 2.5 para a Tabela 2.7.

| $a_1 \setminus a_2$ | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 2 | 1 | 1 | 1 | 0 |
| 3 | 0 | 1 | 1 | 0 |

**Tabela 2.7:** Exemplo para a síntese da função $F_1(a_1, a_2), B = 4$.

$$F_1(a_1, a_2) = a_1^2 \star^1 a_2^0 + a_1^2 \star^1 a_2^3 + a_1^3 \star^1 a_2^1 + a_1^3 \star^1 a_2^0 + a_1^3 \star^1 a_2^3 \tag{2.5}$$

Se queremos sintetizar $F_2(a_1, a_2)$ se requer o operador $\star^2$ para poder ter a saída igual a 2. Para $a_1 = 1$, $a_1^1 = 2$ e $a_2 = 3$, $a_2^3 = 2$, então $a_1^1 \star^2 a_2^3 = 2$, os demais casos segue o mesmo procedimento até completar todas as saídas para a função $F_2$ como é mostrado na Equação 2.6 da Tabela 2.8.

| $a_1 \setminus a_2$ | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 2 | 2 | 2 | 2 |
| 1 | 0 | 0 | 0 | 2 |
| 2 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 |

**Tabela 2.8:** Exemplo para a síntese da função $F_2(a_1, a_2), B = 4$.

Pode-se ver que o segundo termo da Equação 2.6 é a aplicação do Postulado 2.3.8, como um implicante primo formado pelos 4 valores de saída na função $F2(a_1, a_2)$ da Tabela 2.8 que são iguais a 2; isto quer dizer que, quando $a_1 = 0$ e $a_2 = 0$; $a_1 = 0$ e $a_2 = 1$; $a_1 = 0$ e $a_2 = 2$; e finalmente, $a_1 = 0$ e $a_2 = 3$. Deve-se ter presente que os implicantes têm $B^{(n)}$ elementos, com $n = 0, 1, 2, ...$ etc. Por exemplo, para o caso *B*=4, os agrupamentos que podem ser realizados são de 1, 4, 16, ..., $4^B$.

$$F_2(a_1, a_2) = (a_1^2 \star^2 a_2^2 + a_1^2 \star^2 a_2^1 + a_1^2 \star^2 a_2^0 + a_1^2 \star^2 a_2^3) + a_1^1 \star^2 a_2^3 \tag{2.6}$$

$$F_2(a_1, a_2) = a_1^1 \star^2 a_2^3 + a_1^2 \star^2 2 \tag{2.7}$$

A Equação 2.8 é equivalente à Equação 2.7 portanto.

$$F_2(a_1, a_2) = a_1^1 \star^2 a_2^3 + a_1^2 \star^2 a_1^2 \tag{2.8}$$

| $a_1 \setminus a_2$ | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 3 |
| 3 | 0 | 0 | 0 | 0 |

**Tabela 2.9:** Exemplo para a síntese da função $F_3(a_1, a_2), B = 4$.

Agora temos função que sintetiza $F_3(a_1, a_2) = 3$ da Tabela 2.9 obtendo como resultado parcial a Equação 2.9.

$$F_3(a_1, a_2) = a_1^1 \star^3 a_2^0 \tag{2.9}$$

A partir dos resultados parciais das Funções $F_1(a_1, a_2)$, $F_3(a_1, a_2)$ e $F_2(a_1, a_2)$ que tem como resultados as Equações 2.5, 2.8 e 2.9 respectivamente, obtém-se a função que sintetiza a Tabela 2.6, como é mostrado na Equação 2.10.

$$\begin{aligned} F(a_1, a_2) \;=\; & a_1^2 \star^1 a_2^0 + a_1^2 \star^1 a_2^3 + a_1^3 \star^1 a_2^1 + a_1^3 \star^1 a_2^0 + a_1^3 \star^1 a_2^3 \\ & + a_1^1 \star^2 a_2^3 + a_1^2 \star^2 a_1^2 + a_1^1 \star^3 a_2^0 \end{aligned} \tag{2.10}$$

## 2.5 Portas Lógicas

Num sistema digital as unidades básicas de construção são as portas lógicas. Estes dispositivos operam um ou mais sinais lógicos de entrada para produzir uma e somente uma saída a qual é dependente da função implementada no dispositivo. As portas lógicas são encontradas desde um nível de integração em larga escala (VLSI) até o nível de integração digital mais simples. Cada porta lógica MVL avalia uma entrada para obter uma função de saída, onde os valores lógicos serão: “0”, “1”, “2”, ..., “(B-1)” com base B. A seguir apresenta-se uma descrição para cada porta lógica utilizada nesta Álgebra MVL.

### 2.5.1 Porta Lógica Sucessor

Além da representação na Tabela da verdade 2.1 na Seção 2.2.1 visto anteriormente, este operador está associado a uma representação gráfica mostrado na Figura 2.1.

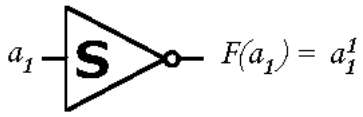



**Figura 2.1:** Porta lógica Sucessor.

### 2.5.2 Porta Lógica AND Estendido

A representação gráfica do operador AND Estendido mostrado na Seção 2.2.2 é apresentada na Figura 2.2 , onde para o caso em estudo se utilizará as portas AND Estendido 1, 2, e 3.

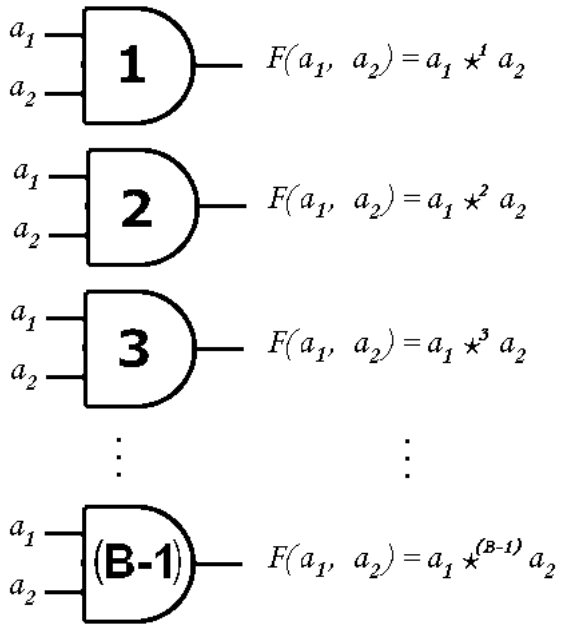



**Figura 2.2:** Portas lógicas AND Estendido.

### 2.5.3 Porta Lógica Máximo

A representação gráfica desta porta é mostrada na Figura 2.3, onde o comportamento lógico foi descrito na Tabela 2.4, esta porta pode ter duas ou mais entradas lógicas.

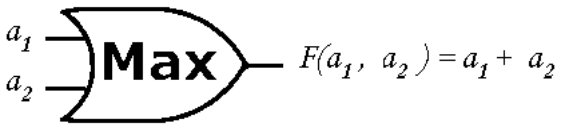



**Figura 2.3:** Porta lógica Máximo.

### 2.5.4 Porta Lógica OR Estendido

A porta OR Estendido é representada gráficamente como na Figura 2.4, para o caso em estudo somente utilizaremos as portas OR Estendido 0, 1 e 2, o comportamento lógico para esta porta foi descrita na Tabela 2.3.

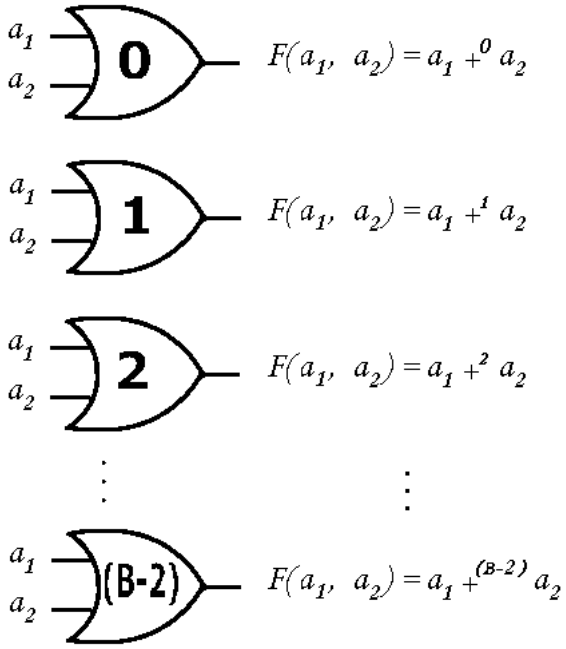



**Figura 2.4:** Portas lógicas OR Estendido.

### 2.5.5 Porta Lógica Mínimo

A porta lógica Mínimo é apresentada na Figura 2.5, o comportamento lógico desta porta foi apresentada na Tabela 2.5.

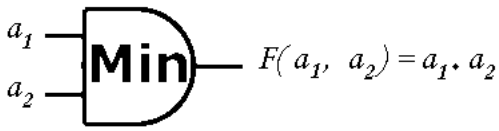



**Figura 2.5:** Porta lógica Mínimo.

CAPÍTULO

3

# Circuitos Seqüenciais MVL

## 3.1 Circuitos Seqüenciais MVL

Da mesma forma que os circuitos lógicos binários temos dois tipos de circuitos MVL: circuitos combinacionais e circuitos seqüenciais. Um circuito seqüencial é constituído por um circuito combinacional e elementos de memória MVL. Os elementos de memória MVL são capazes de armazenar informação em B níveis lógicos, onde B é a base (nesta dissertação a base a ser tomada para os exemplos é B = 4), os elementos de memória MVL apresentados nesta dissertação são o *latch RS, Flip-flop RS, Flip-flop D, Flip-flop Master-Slave*. Algumas das saídas do circuito combinacional são entradas para os elementos de memória, recebendo o nome de **variáveis do próximo estado** e a saída dos elementos de memória constituem parte das entradas para o circuito combinacional recebendo o nome de **variáveis de estado atual**. As conexões entre os circuitos combinacionais e os elementos de memórias configuram o que se chama laço de realimentação como se observa na Figura. 3.1.

O estado de um circuito seqüencial é determinado pela informação armazenada nos elementos de memória. Os valores do próximo estado e as saídas são determinados pela informação recebida nas entradas e a informação do estado atual armazenado nos elementos de memória. Desta forma pode ser dito que as saídas de um circuito seqüencial dependem não apenas das entradas, mas também do estado atual, armazenados nos elementos de memória.

### 3.1.1 Representação do Diagrama de Bloco

O modelo de circuito seqüencial pode ser criado a partir de um circuito combinacional, como se pode ver na Figura 3.1. A diferença com o modelo combinacional é que o modelo seqüencial possui memória. A Equação 3.1 representa o modelo da Figura 3.1a, onde a saída $z_i$ esta somente em função da entrada $x_i$. O modelo mostrado na Figura 3.1b representa o circuito seqüencial, onde o n-ésimo $(x_1, ..., x_n)$ é referente à entrada, o m-ésimo $(z_1, ..., z_m)$ é a saída, e o r-ésimo $(y_1, ..., y_r)$ e $(Y_1, ..., Y_r)$ representam variáveis do estado atual e variáveis do próximo estado respectivamente. A relação existente entre essas variáveis podem ser expressas em forma de equações matemáticas, como as Equações 3.2 e 3.3.

$$z_i = f_i(x_1, x_2, ..., x_n) \quad i = 1, ..., m \tag{3.1}$$

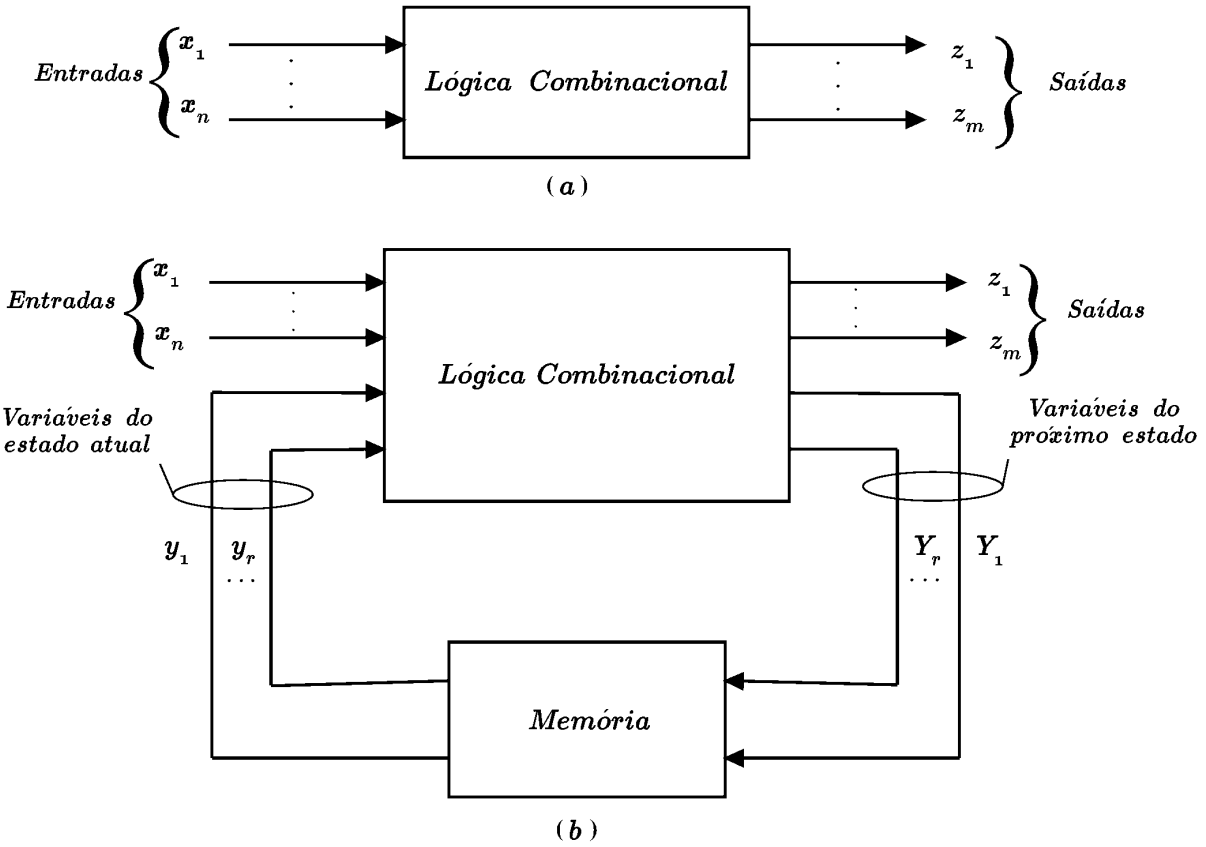



**Figura 3.1:** Diagrama de blocos de um circuito digital MVL. (a) Circuito lógico combinacional MVL. (b) Circuito lógico seqüencial MVL (máquina de *Mealy*).

$$z_i = g_i(x_1, ..., x_n, y_1, ..., y_r) \quad i = 1, ..., m \tag{3.2}$$

$$Y_i = h_i(x_1, ..., x_n, y_1, ..., y_r) \quad i = 1, ..., r \tag{3.3}$$

As funções $g_i$ e $h_i$ são funções MVL. As Equações 3.2 e 3.3 podem ser escritas em notação vetorial como:

$$z = g(x, y) \tag{3.4}$$

$$Y = h(x, y) \tag{3.5}$$

onde

$$z = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_m \end{bmatrix} \quad x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_r \end{bmatrix} \quad Y = \begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_r \end{bmatrix} \tag{3.6}$$

Deve-se considerar que $z_i$, $x_i$, $y_i$ e $Y_i$ são todas variáveis MVL, seus valores são 0, 1, 2 e 3 lógico, para o caso em estudo (lógica quaternária).

### 3.1.2 Tabela e Diagrama de Estado

As Equações 3.2, 3.3, 3.4 e 3.5 são as que definem o comportamento do circuito seqüencial mostrado na Figura 3.1b. No entanto, esse circuito pode ser representado alternativamente por um diagrama de estados como na Figura 3.2a, ou tabela de estados apresentado na Figura 3.2b, sendo estes uma representação gráfica das máquinas de estados finitos (MEF).

As saídas podem ser associadas as transições (máquinas de *Mealy*) ou aos estados (máquinas de *Moore*). O conceito básico da MEF para circuitos binários possui algumas restrições, pois a informação é limitada à lógica binária (0 ou 1). Nesta dissertação adota-se a MEF para a lógica MVL em base B=4 com saídas que podem tomar valores 0, 1, 2, ou 3.

Uma máquina de estados ou autômato finito *Mealy* permite modelar o comportamento de um sistema digital com memória limitada, esta máquina é composta por estados representados por círculos, transições e ações representado por setas, etiquetado com uma entrada $x$ e saída $z$. Um estado armazena informações do passado, isto é, ele reflete as mudanças de estado descritas por uma condição, que é necessária para que a transição ocorra. Uma ação é a descrição de uma atividade que deve ser realizada num determinado momento.

A tabela de transição da Figura 3.2b mostra a qual estado se moverá a máquina, dependendo do estado atual e a entrada. O vetor de entradas $x$ é listado na parte superior enquanto o vetor de estado presente $y$ é listado ao lado esquerdo em forma de coluna, e a intersecção entre o estado presente e a entrada fornece o estado seguinte $Y$ com sua respectiva saída $z$.

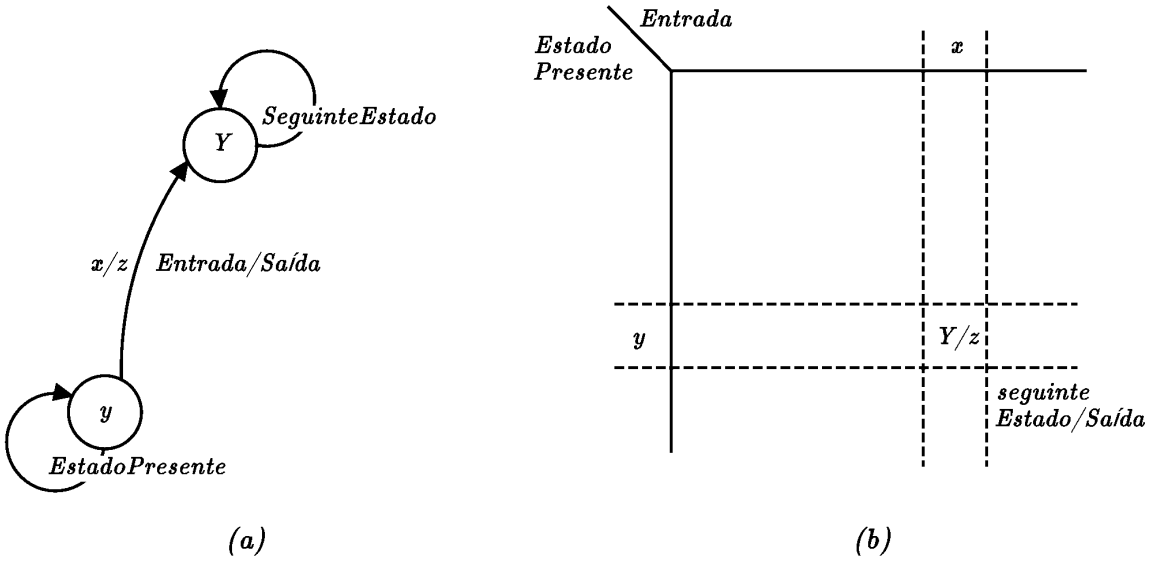



**Figura 3.2:** (a) Diagrama de estado. (b) Tabela de estado.

## 3.2 Dispositivos de Memória MVL

A unidade de memória é parte de um circuito seqüencial. Neste trabalho, trata-se o estudo da característica externa da memória (nível lógico MVL) e não os detalhes do funcionamento interno. Ou seja, a síntese de circuitos seqüenciais MVL está restringida ao uso desses elementos de memória no nível lógico MVL que serão utilizadas nos projetos de sistemas seqüenciais digitais mais complexos. Similar aos circuitos digitais seqüenciais binários, os circuitos digitais seqüenciais MVL dividem-se em dois grupos.

Circuitos seqüenciais síncronos, com um sinal de *clock* que determina quando os elementos de memória irão amostrar os valores em suas entradas, dependendo do tipo de memória, esta amostragem das entradas pode ser sincronizadas pela borda ascendente ou pela borda descendente do *clock*. Seja qual for o tipo de sincronização, o tempo que transcorre entre duas amostragens sucessivas, é equivalente a um tempo T ou período do *clock*. Isto implica que, qualquer mudança no estado de un circuito seqüencial síncrono irá ocorrer somente após a borda do sinal do *clock* na qual os elementos de memória são disparados. Este tipo de circuitos também são conhecidos como *Flip-flops*, construídos com base no *latch RS*.

Circuitos seqüenciais assíncronos dependem diretamente da mudança das entradas, alterando o estado do circuito em qualquer momento. Semelhante aos circuitos seqüenciais assíncronos binários, apresentam uma capacidade de armazenamento que está relacionado com o atraso de propagação dos circuitos que os compõem, este tipo de circuito apresenta dificuldades, desde que o funcionamento do circuito é dependente das características do circuito em relação ao *delay* (portas lógicas e fios). Nos circuitos seqüenciais assíncronos binários, a principal dificuldade é que os componentes apresentam atrasos que não são fixos, podendo ser diferentes mesmo para exemplares com mesma função. Esta é uma das razões pelas quais os circuitos

digitais assíncronos tem sido evitados, sempre que possível. Os circuitos seqüenciais MVL apresentam as mesmas dificuldades, que podem ser percebidas nas simulações. Estes circuitos também são conhecidos como *latch* MVL sendo esta a forma básica de memória. A partir dela podem ser construídos diferentes tipos de memória que serão apresentadas nesta dissertação. Os *latches* são sensíveis ao nível de entrada.

### 3.2.1 Clock MVL (Clk)

O *clock* é encarregado de sincronizar a troca de estado quando se apresenta um valor lógico nas entradas do *flip-flop*. Na forma de onda do *clock* pode-se identificar a borda de subida e a borda de descida. Na borda de subida tem os níveis altos 1, 2, 3, ..., (B-1) e na borda de descida é identificado como a mudança do nível (B-1) para o nível baixo 0. O período do *clock* é representado por *T*, a Figura 3.3 mostra o *clock* para a base B=4.

É importante definir qual é a semântica deste *clock* quando se utiliza para sincronizar circuitos *MVL*. Têm-se duas opções que dependem do tipo de aplicação. A primeira opção que os tipos de ativação dos circuitos de memória sejam definidos, unicamente na borda de decida do *clock* na transição de *(B-1)* para *0*, diminuindo a freqüência de sincronismo do circuito e seria similar a utilizar um *clock* binário. A segunda opção é permitir que os tempos de ativação dos circuitos de memória sejam definidos em todas as transições do *clock*: de 0 para 1, de 1 para 2, de 2 para 3, etc., neste trabalho utiliza-se a segunda opção como na Figura 3.3.

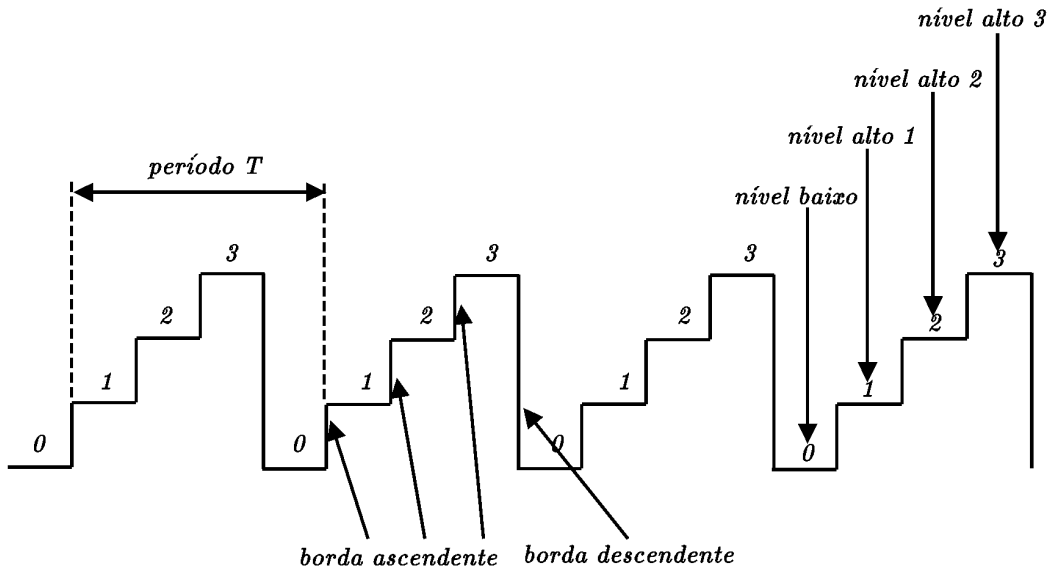



**Figura 3.3:** *Clock MVL* em base B = 4

A freqüência (F) é associada ao movimento ondulatório que indica o número de ciclos, voltas ou oscilações, por unidade de tempo. Neste caso nos referimos às oscilações do *clock* MVL com base B = 4, ele alterna entre 0, 1, 2, 3, e volta para zero. O tempo que leva é denominado de período (T) como é mostrado na Figura 3.3, este tempo decorrido para uma

oscilação é medido como freqüência, sendo o inverso do tempo como é mostrado na Equação 3.7, um período completo é também chamado de ciclo do *clock*.

$$F = \frac{1}{T} \tag{3.7}$$

### 3.2.2 Requisitos Temporais de Operacionalidade de *Flip-flops* MVL

De maneira similar aos circuitos seqüenciais binários síncronos que funcionam com condições temporais especificadas pelo fabricante, nesta secção definem-se as condições que devem ser tomadas para os circuitos de memória MVL. Para garantir a funcionalidade dos *flip-flops* MVL é necessário definir os requisitos temporais de operacionalidade, para isto define-se um tempo mínimo T para o clock, sendo este dependente do tempo de preparação (*setup time*), tempo de propagação (*propagation time*) e o tempo que demora em se propagar o sinal para os componentes (*skew*), esta relação é mostrada na Equação 3.8. A seguir são detalhados os tempos mínimos que definem o tempo T para o *clock* que é mostrado na Figura 3.4.

**Skew** é conhecido como desvio que sofre o *clock* no tempo. O *skew* é produzido quando o sinal do *clock* não chega a todos componentes ao mesmo tempo. Isto pode acontecer pelos seguintes motivos: o material dos condutores não é perfeito, e faz que o sinal do circuito seja rápido ou lento. Outro motivo é que distância desde o *clock* até cada componente não é a mesma, e o sinal tarda em percorrer os caminhos mais longos. Existem algumas soluções para a lógica binária, como o algoritmo apresentado no artigo [32]. Menciona-se isto para ter presente este fator, quando um circuito for projetado no caso MVL. Isto é um fator muito importante porque alguns componentes como os elementos de memória, necessitam ter os dados na entrada necessariamente no momento em que é ativado o *clock*, não antes nem depois.

**Tempo de Setup** ou tempo de pré-ativação, considera-se como tempo de *setup* ao tempo necessário para ter a entrada ficar num valor lógico antes de uma transição do *clock*. Isto significa que neste período o sinal na saída do circuito seqüencial deve estar estável.

**Tempo de Hold** também chamado de tempo de manutenção, se refere ao tempo necessário com uma entrada estável após uma transição do *clock* para que esta seja armazenada no circuito seqüencial.

**Tempo de Propagação** o tempo de propagação também conhecido como *delay*, é referido ao tempo que uma variação de valor lógico numa das suas entradas demora a fazer efeito na saída da porta lógica. Já no caso das memórias tem-se várias portas lógicas e o *delay* total é dado pela soma dos *delays* de cada uma das portas. Nesta dissertação, os *delays* utilizados em cada porta são valores estimados e isto pode ser observado na Figura 3.4.

Estes tempos mínimos ajudam a definir a freqüência máxima de operação do *clock*, o tempo entre duas bordas ascendentes ou descendentes deve ser superior à soma de tempo de *setup* (Tsetup) e tempo de propagação (Tpropagacao), isto significa que a freqüência máxima (Fmaxima) do sinal do *clock* é o inverso da soma como mostrado nas Equações 3.8 e 3.9.

$$T \geq Tsetup + Tpropagacao + Tskew \tag{3.8}$$

$$Fmaxima = 1/T \tag{3.9}$$

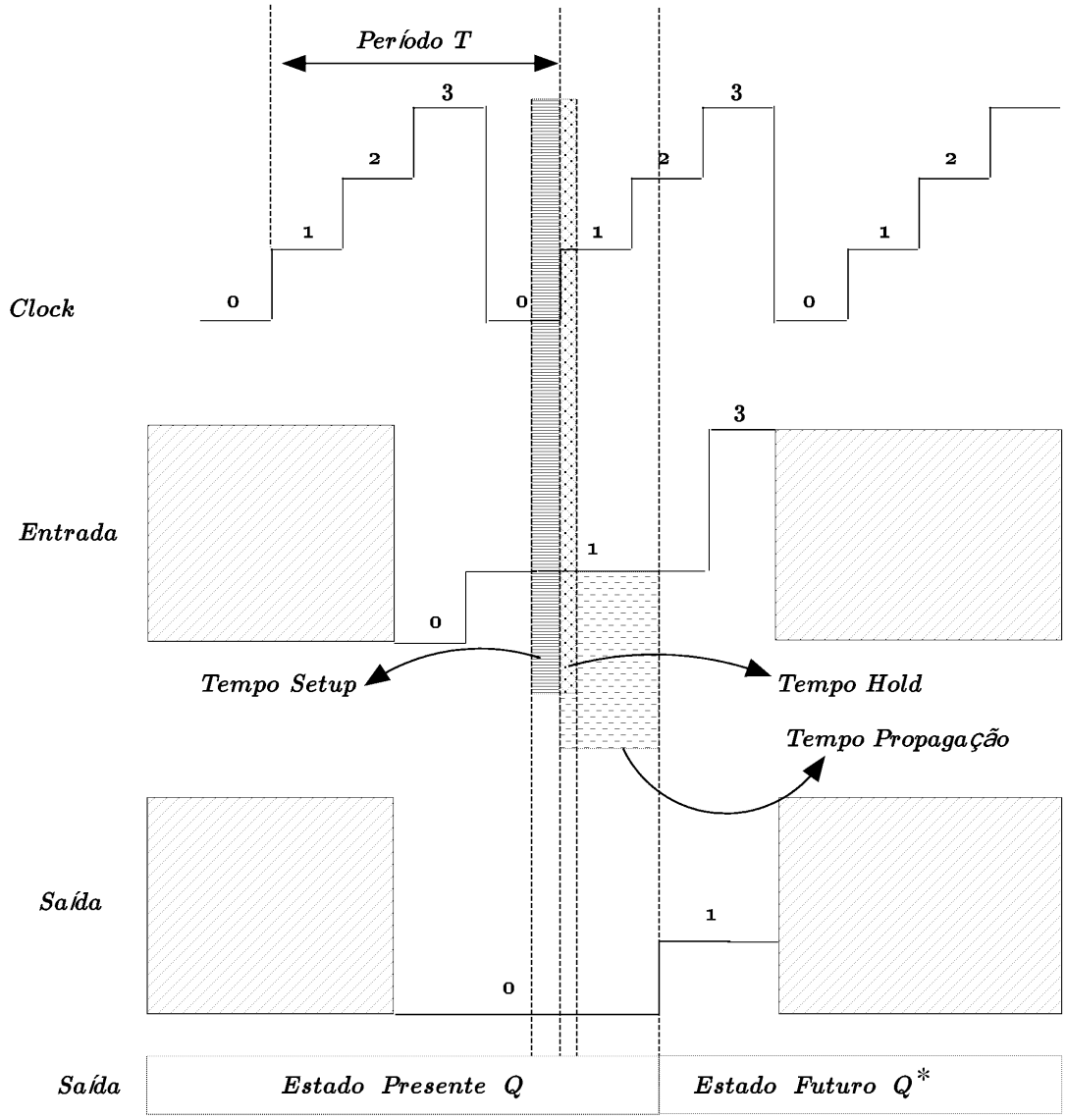



**Figura 3.4:** Requisitos temporais de operacionalidade

### 3.2.3 Síntese do *Latch RS*

O *Latch RS* é um circuito assíncrono de memória com duas entradas *Set* (*S*) e *Reset* (*R*), e duas saídas (*Q*) e (*QQ*) (permite realimentar o circuito com o valor lógico memorizado) que realiza três funcionalidades: memorizar (*Set*), apagar (*Reset*) e manter (*Hold*). Na lógica binária

*R*, *S*, *Q*, $\bar{Q} \in D = \{0, 1\}$. Note-se que o *latch* mantém (*Hold*) o valor requerido com S = 0 e *R* = 0; memoriza (*Set*) o valor requerido com $S \neq 0$ e *R* = 0; e apaga (*Reset*) o valor requerido com *S* = 0 e $R \neq 0$. *S* ativada significa armazenar; *R* ativada significa apagar; e o dois desativado significa manter o valor. Assume-se que o circuito ativado é determinado quando o sinal de entrada é igual a 1. Note que a combinação dos valores de entrada *R* e *S* onde nenhum é zero simultaneamente, é considerada proibida [33].

A idéia fundamental é estender o domínio binário para o domínio *MVL* mantendo a funcionalidade do elemento de memória. Portanto,*R*, *S*, *Q*, *QQ* $\in D = \{0, 1, 2, 3, ..., (B\text{-}1)\}$ caracterizando o domínio *MVL*. Como no caso binário, com *R* = 0, a entrada *S* = 1, *S* = 2, ..., ou *S* = (*B-1*) memoriza *Q* = 1, ou *Q* = 2, ..., ou *Q* = (*B-1*), respectivamente. Com *S* = 0, a entrada *R* = 1, *R* = 2, ..., ou *R* = (*B-1*) limpa *Q*, sendo *Q*=0 em todos os casos. As entradas *R* = 0 e *S* = 0 mantém o valor de *Q* inalterável. Finalmente, a combinação R $\neq 0$ e S $\neq 0$ é proibida, de maneira análoga ao circuito binário. Para a síntese do elemento de memória, como esta combinação é proibida, a saída se trata como *don't care* como mostrada na Tabela 3.2, e portanto, escolhe-se a saída da maneira mais adequada em relação à necessidade de simplificar o circuito lógico. Por esta razão, a saída do *latch* que melhor simplifica o circuito lógico é mostrada na Tabela 3.1, que permite utilizar convenientemente a propriedade de simplificação, escolhendo os implicantes quando *S = 1, S = 2, S = 3* e as outras variáveis de entrada com qualquer valor. O Tabela 3.1 mostra o estado futuro $Q^*$, onde Q identifica o estado de saída, *R* a entrada de *Reset*, *S* a entrada de (*Set*) e *Hold*, *Reset* e *Set*, na parte superior, identificam as funcionalidades já descritas. Por exemplo, quando as entradas *S* e *R* estão simultaneamente no nível 0, o circuito lógico mantém o estado anterior (*Hold*), se a entrada *R* muda para 1, ou 2, ou 3 e *S* = 0, então, o circuito lógico passa ao estado *Reset* (*Q* = 0).

| | *Hold* | *Reset* | | | *Set* | *Set* | | | *Set* | *Set* | | | *Set* | *Set* | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *Q* \ *S-R* | 00 | 01 | 02 | 03 | 10 | 11 | 12 | 13 | 20 | 21 | 22 | 23 | 30 | 31 | 32 | 33 |
| **0** | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 3 | 3 | 3 | 3 |
| **1** | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 3 | 3 | 3 | 3 |
| **2** | 2 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 3 | 3 | 3 | 3 |
| **3** | 3 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 3 | 3 | 3 | 3 |

**Tabela 3.1:** Estado futuro $Q^*$ para o *Latch RS*

| | *Hold* | *Reset* | | | *Set* | *Reset* | | | *Set* | *Reset* | | | *Set* | *Reset* | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *Q* \ *S-R* | 00 | 01 | 02 | 03 | 10 | 11 | 12 | 13 | 20 | 21 | 22 | 23 | 30 | 31 | 32 | 33 |
| **0** | 0 | 0 | 0 | 0 | 1 | - | - | - | 2 | - | - | - | 3 | - | - | - |
| **1** | 1 | 0 | 0 | 0 | 1 | - | - | - | 2 | - | - | - | 3 | - | - | - |
| **2** | 2 | 0 | 0 | 0 | 1 | - | - | - | 2 | - | - | - | 3 | - | - | - |
| **3** | 3 | 0 | 0 | 0 | 1 | - | - | - | 2 | - | - | - | 3 | - | - | - |

**Tabela 3.2:** Estado futuro $Q^*$ para o *Latch RS*. O símbolo '-' representa '*don't care*'

O *Latch RS* apresentado na Figura 3.5 (a) é projetado com portas lógicas MVL com base *B=4*, com seus respectivos *delays*. No projeto deste circuito são utilizadas as seguintes portas lógicas onde cada uma delas apresenta um *delay* respectivo, 6 portas *Suc* com 5ns, 2 portas *And1* com 5ns, 2 portas *And2* com 5ns, 2 portas *And3* com 5ns, 1 porta *Max* de 3 entradas com 5ns e 1 porta *Max* de 4 entradas com 20ns.

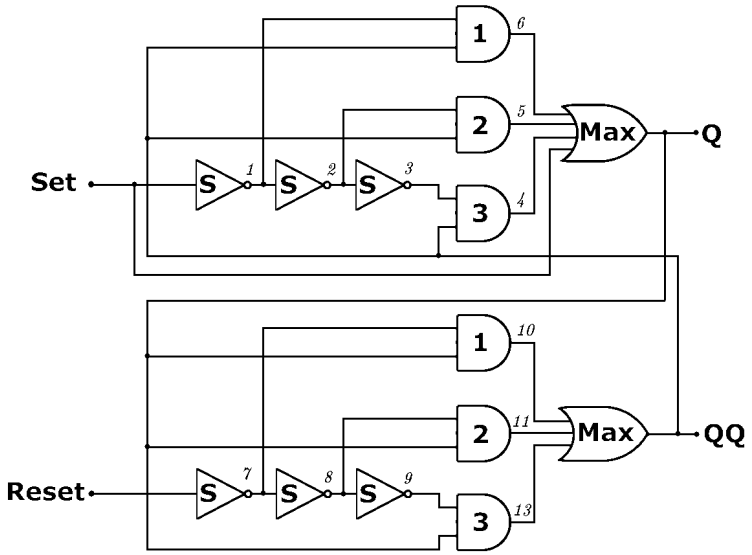



**Figura 3.5:** *Latch RS (a)*

Na Figura 3.6 (b) é apresentado o projeto do *Latch RS* com uma pequena modificação necessária, substituindo a conexão direta que existia entre a entrada *Set* e a porta *Max* de 4 entradas, por uma conexão a *Suc* a partir da terceira porta *Suc* da entrada *Set*. Isto evita que o valor apresentado na entrada *Set* chegue em diferentes instantes, o objetivo é evitar o *race condition* ou *race hazard* do circuito de memória. Este *latch* é utilizado como base para projetar os outros circuitos de memória. As simulações destes dois circuitos são apresentados na Figura 3.8(a) e 3.9(b), respectivamente.

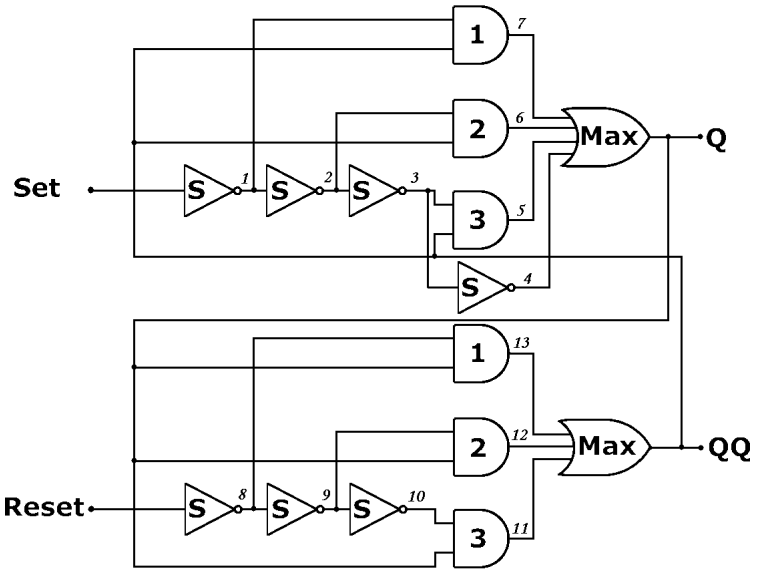



**Figura 3.6:** *Latch RS (b)*

A seguir são apresentadas as Equações 3.10 e 3.11 com características que descrevem o comportamento do circuito *latch SR* apresentado na Figura 3.6 (b) sendo S (set), R (reset) QQ (realimenta com o valor armazenado) e Q (saída do valor memorizado) ou seguinte estado.

$$Q^* = S^1 \star^1 QQ + S^2 \star^2 QQ + S^3 \star^3 QQ + S^4 \tag{3.10}$$

$$QQ = R^1 \star^1 Q + R^2 \star^2 Q + R^3 \star^3 Q \tag{3.11}$$

### Diagrama de Estados Para o *Latch SR*

O comportamento do circuito seqüencial *latch* RS pode ser representado por um diagrama de estados mostrado na Figura 3.7, os estados *reset*, *set1*, *set2*, *set3*, são representados por nós, as transições entre estados é representado por uma seta. A condição segundo a qual acontece uma transição é definida junto a aresta respectiva. Por exemplo, estando no estado *reset*, para ele ir para o estado *set1* é necessário que *R = 0* e *S = 1*.

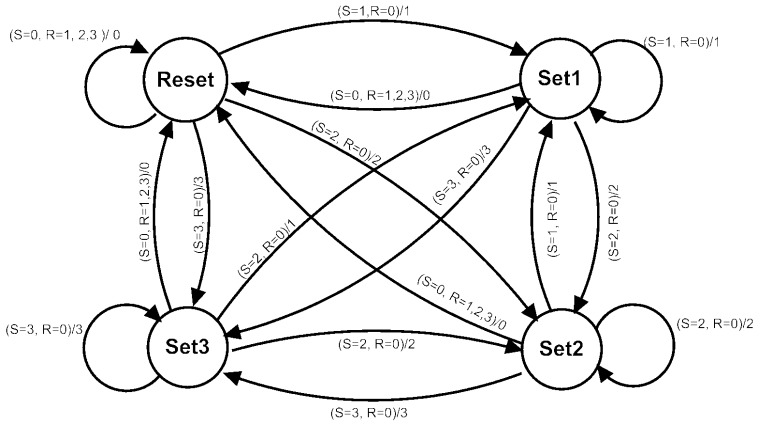



**Figura 3.7:** Diagrama de estados para o *Latch RS*

### Simulação em VHDL e Discussão dos Resultados Para o *Latch SR*

A simulação do elemento de memória assíncrono *latch* RS realizou-se considerando o tempo de *delay* de 5ns para as portas *Suc, And1, And2, And3, Max3* (com três entradas), e 20ns de *delay* para a porta *Max4* (com quatro entradas), este último apresenta maior *delay* que permite evitar o *race hazard*. A implementação deste circuito foi feita na linguagem de descrição VHDL, adaptando as entradas e saídas para o caso MVL.

A simulação da Figura 3.8 (a) corresponde ao circuito apresentado na Figura 3.5 (a), nesta simulação com linha do tempo em nanosegundos, pode-se observar no tempo 1220ns acontece uma troca de valor lógico na saída Q, estando com *Set = 0* e *Reset = 0*, o qual é identificado como *race hazard*, isto nos obriga a fazer uma modificação no circuito a qual é apresentado na Figura 3.6 (b), incrementando uma porta *Suc*. Com esta modificação obtemos uma simulação com saídas corretas que pode ser verificado na simulação da Figura 3.9 (b) e a Tabela 3.2.

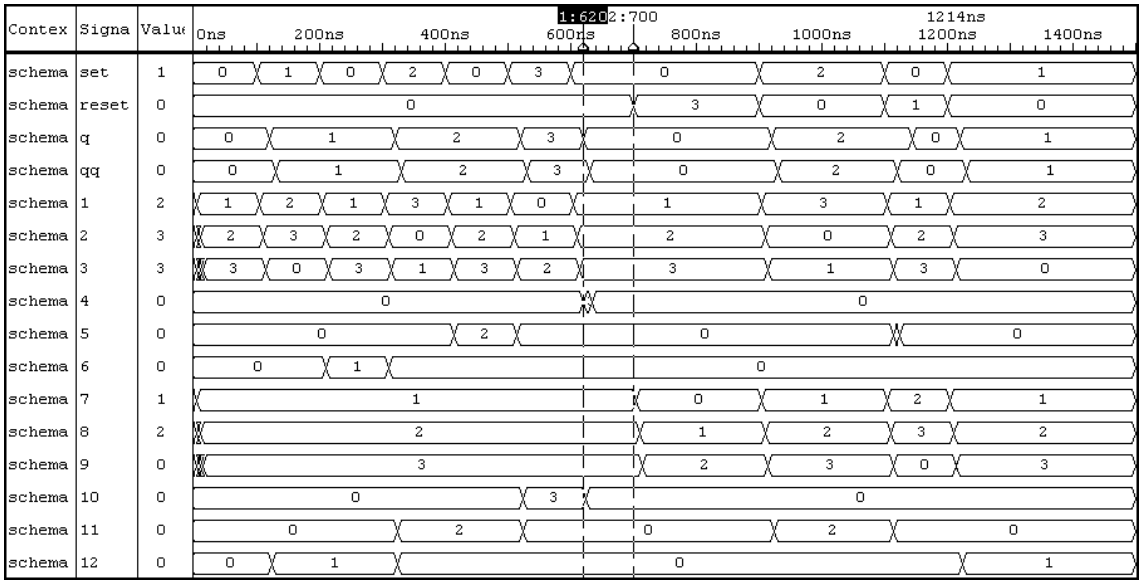



**Figura 3.8:** Simulação do *Latch RS (a)*

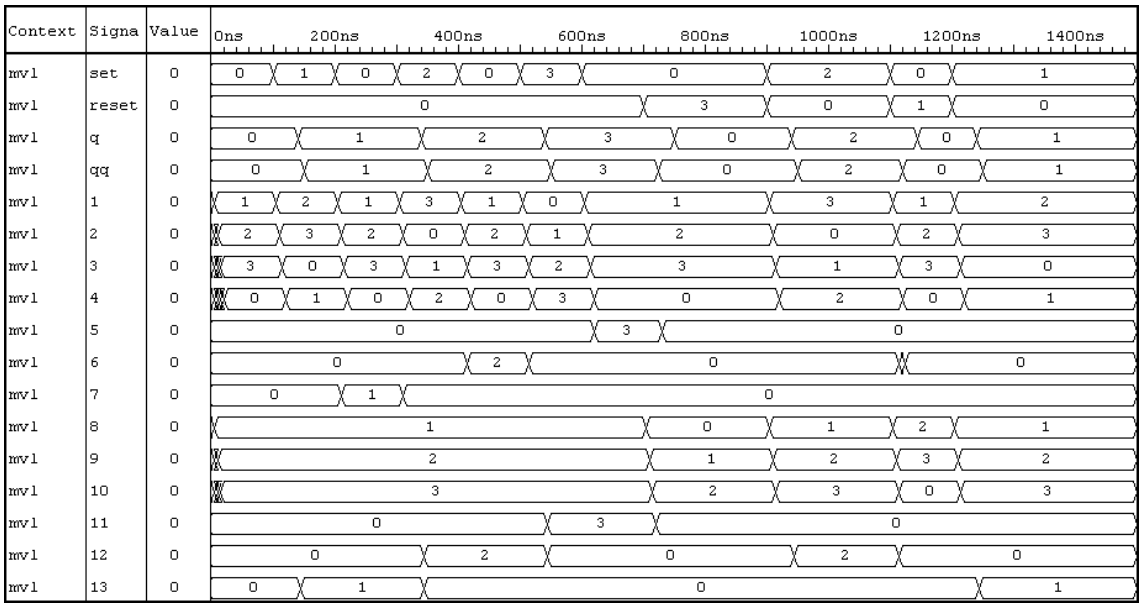



**Figura 3.9:** Simulação do *Latch RS (b)*

### 3.2.4 Síntese do *Flip-flop RS*

O *Flip-flop* tipo *RS* é um circuito síncrono de memória com três entradas, *set, reset e clock*, esta última é encarregada de cadenciar o circuito, memorizando as entradas em tempos determinados e duas saídas, *Q* encarregada de mostrar o valor lógico armazenado, QQ realimenta o circuito com o valor memorizado, isto pode ser observado na Tabela 3.4 mostrando o comportamento do circuito. Na lógica binária *Set, Reset, Clk, Q,* $\overline{Q} \in$ D= {0, 1 }. A Tabela 3.3 mostra o estado futuro $Q^*$ para as entradas *Set, Reset* (mostrado na parte superior) e *Q* que identifica o estado de saída presente (mostrado no lado esquerdo).

Para estender o domínio binário para o domínio MVL mantendo as funcionalidades do elemento de memória, definimos que *Set, Reset, Clk, Q, QQ* ∈ D = {0, 1, 2, 3, ..., (B-1) } caracterizado no domínio MVL mostrado na Figura 3.4. Como no caso binário, as entradas *Set = 1, ou Set = 2, ...,ou Set = (B-1)* e *Clk = 1 ou Clk = 2, ..., ou Clk = (B-1)* e *Reset = 0*, memoriza a entrada *Set* em Q = 1, Q = 2,... , Q = (B-1) respectivamente. Pode-se observar que o tempo no qual a saída *Q* assume o valor da entrada *Set* está definido pelo *Clock* de sincronismo. O clock MVL foi apresentado na Seção 3.2.1 como caso particular para a base B = 4 onde o domínio D = { 0, 1, 2, 3 }. O funcionamento do *Clock* neste circuito é dado da seguinte maneira: Quando o *clock* é ativado num nível lógico *Clk = 1, ou Clk = 2, ..., Clk = B-1* e *Set = 1, ou Set = 2, ..., ou Set = B-1* respectivamente e *Reset = 0* neste caso a entrada *Set* será memorizado na saída *Q*. O *clock* é definido com tempo de ativação nos níveis *1, 2, ..., (B-1)*.

| Q \ *Set-Reset* | 00 | 01 | 10 | 11 |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | - |
| 1 | 1 | 0 | 1 | - |

**Tabela 3.3:** Estado futuro para o *Flip-flop RS* binário

| | *Hold* | *Reset* | | | *Set* | *Reset* | | | *Set* | *Reset* | | | *Set* | *Reset* | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *Q \ S-R* | 00 | 01 | 02 | 03 | 10 | 11 | 12 | 13 | 20 | 21 | 22 | 23 | 30 | 31 | 32 | 33 |
| **0** | 0 | 0 | 0 | 0 | 1 | - | - | - | 2 | - | - | - | 3 | - | - | - |
| **1** | 1 | 0 | 0 | 0 | 1 | - | - | - | 2 | - | - | - | 3 | - | - | - |
| **2** | 2 | 0 | 0 | 0 | 1 | - | - | - | 2 | - | - | - | 3 | - | - | - |
| **3** | 3 | 0 | 0 | 0 | 1 | - | - | - | 2 | - | - | - | 3 | - | - | - |

**Tabela 3.4:** Estado futuro $Q^*$ para o *Flip-flop RS*. O símbolo '-' representa '*don't care*'

A Figura 3.10 apresenta o circuito de memória *Flip-flop* RS, projetado com portas lógicas MVL em base *B = 4*, e considerando os *delays* para cada porta lógica, sendo: 6 portas *Suc* com 5ns, 2 portas *And1* com 5ns, 2 portas *And2* com 5ns, 2 portas *And3* com 5ns, 1 porta *Max* de 3 entradas com 5ns e 1 porta *Max* de 4 entradas com 20ns. O projeto deste circuito para outros níveis lógicos é simples por apresentar uma forma intuitiva de construção, a simulação é apresentada Figura 3.12.

A equação característica para o *Flip-flop RS* para o próximo estado é apresentado na Equação 3.12 onde *set* é representado por *S*, *reset* por *R* e as saídas *Q* representando o valor armazenado e *QQ* que realimenta o circuito com o valor armazenado.

$$Q^* = S^1 \star^1 QQ + S^2 \star^2 QQ + S^3 \star^3 QQ + S^4 \tag{3.12}$$

$$QQ = R^1 \star^1 Q + R^2 \star^2 Q + R^3 \star^3 Q \tag{3.13}$$

### Diagrama de Estados Para o *Flip-flop RS*

O comportamento do circuito síncrono seqüencial *Flip-flop RS* pode ser representado por um diagrama de estados mostrado na Figura 3.11, os estados *reset, set1, set2, set3*, são re-

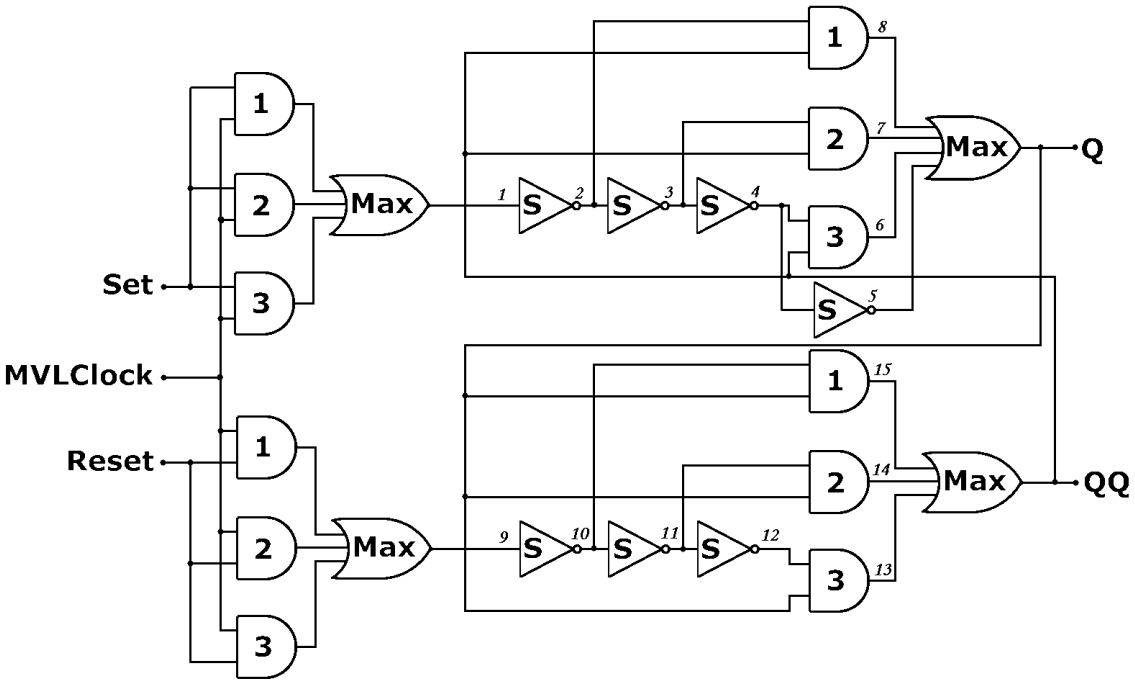



**Figura 3.10:** *Flip-flop RS*

presentados por nós, as transições entre estados são representadas por uma seta. A condição segundo a qual acontece uma transição é definida junto à seta respectiva. Por exemplo, estando no estado *reset*, para que o circuito vá para o estado set1 é necessário que R=0 e S=1.

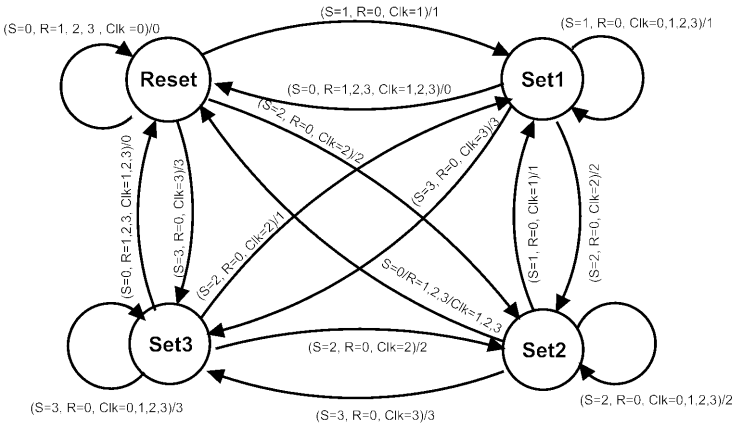



**Figura 3.11:** Diagrama de estados para o *Flip-flop RS*

### Simulação em VHDL e Discussão dos Resultados Para o *Flip-flop SR*

A simulação mostrada na Figura 3.12 apresenta o correto funcionamento do circuito de memória, que foi implementado considerando o tempo de atraso de 20ns para a porta *Max* de 4 entradas e 5ns para o restante das portas lógicas. Cada porta é descrita na linguagem VHDL, adaptando-se as entradas e saídas para o caso MVL. Esta simulação pode ser conferida na Tabela 3.4.

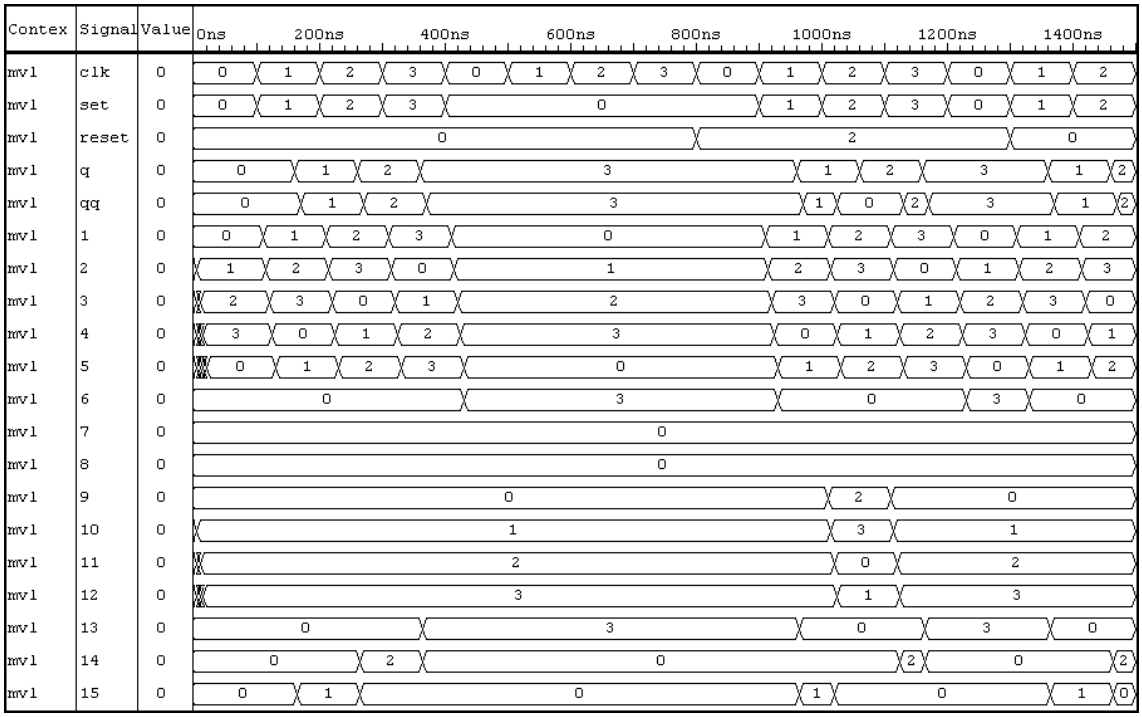



**Figura 3.12:** Simulação do *Flip-flop* RS

## 3.2.5 Síntese do *Flip-flop D*

O *flip-flop* tipo *D* é um circuito síncrono de memória com uma entrada (*D*), um sinal do *clock* e as saídas *Q* e *QQ*, esta última é encarregada de realimentar o circuito com o valor memorizado. O *flip-flop* tipo *D* memoriza a entrada *D* na saída Q em tempos definidos pelo *clock* (borda ascendente), significa, depois de um tempo de retardo (*delay*) de onde toma seu nome (tipo *D*), a Tabela 3.6 mostra o estado futuro $Q^*$ deste circuito. Na lógica binária o Flip-flop tipo D é criado pela necessidade de evitar a ocorrência do estado proibido. O *flip-flop D* é construído a partir do *Latch RS*, con variáveis *D, Q* e *Clk* $\in D = \{0, 1\}$. A Tabela 3.5 mostra o estado futuro $Q^*$, onde Q identifica o estado do circuito, com uma entrada *D* mostrada na parte superior da Tabela 3.5.

A idéia fundamental é estender o domínio binário para o domínio *MVL* mantendo a funcionalidade do elemento de memória. Portanto, *D*, *Q*, *QQ* e *Clk* $\in D = \{0, 1, 2, 3, ..., B-1\}$ é caracterizado no domínio *MVL*. Como no caso binário, a entrada *D* = 0, ou *D* = 1, ..., ou *D* = *B*-1 memoriza *Q* = 0, ou *Q* = 1, ou *Q* = 2, ..., ou *Q* = *B*-1, respectivamente. O tempo no qual a saída (*Q*) toma o valor da entrada (*D*) está definido pelo sinal de *Clock* de sincronismo. O *clock MVL* funciona da seguinte maneira: neste caso será considerado ativa na borda de subida, quando o *Clk* se ativa, o *Flip-flop* memoriza o valor da entrada *D* na saída Q. Portanto, é importante definir qual é a semântica deste *clock* quando se utiliza para sincronizar circuitos *MVL* (definido na secção 3.2.1), permitindo que os tempos de ativação seja somente na borda ascendente do *Clk*: de 0 para 1, de 1 para 2, de 2 para 3 no caso da base *B = 4*. A seguir são apresentados duas alternativas do *Flip-flop D*.

A Tabela 3.6 apresenta o estado futuro do *Flip-flop* tipo D mostrado na Figura 3.13, este circuito de memória armazena um dígito através da entrada *D* quando o *Clk* estiver habilitado (*Clk = 1, ou Clk = 2, ou Clk = 3, ..., ou Clk = (B-1)*), significa que qualquer valor apresentado na entrada *D* será memorizado quando o *Clk* estiver ativo, e não necessariamente tem que coincidir com a entrada *D*, este comportamento pode ser conferido na simulação deste circuito apresentada na Figura 3.17.

Neste caso, o circuito de memória memoriza a entrada *D* sempre e quando o *Clk* estiver no mesmo nível lógico, por exemplo, para memorizar uma entrada *D = 1* o *clock* necessariamente tem que estar em *Clk = 1*, se o *Clk* estiver em outro nível lógico o circuito simplesmente não memoriza, isto pode ser conferido na Tabela 3.7 que apresenta o estado futuro do *Flip-flop D* mostrado na Figura 3.14, e a sua simulação é apresentada na Figura 3.18

| Q \ *D* | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 0 | 1 |

**Tabela 3.5:** Estado futuro para o *Flip-flop D* binário

| | *H* | *Set* | | | *H* | *Set* | | | *H* | *Set* | | | *H* | *Set* | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *Q* \ *D-CLK* | 00 | 01 | 02 | 03 | 10 | 11 | 12 | 13 | 20 | 21 | 22 | 23 | 30 | 31 | 32 | 33 |
| **0** | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 2 | 2 | 2 | 0 | 3 | 3 | 3 |
| **1** | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 1 | 3 | 3 | 3 |
| **2** | 2 | 0 | 0 | 0 | 2 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 3 | 3 | 3 |
| **3** | 3 | 0 | 0 | 0 | 3 | 1 | 1 | 1 | 3 | 2 | 2 | 2 | 3 | 3 | 3 | 3 |

**Tabela 3.6:** Estado futuro do *Flip-flop* tipo *D* (a)

| | *H* | *Set* | | | *H* | *Set* | | | *H* | *Set* | | | *H* | *Set* | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *Q* \ *D-CLK* | 00 | 01 | 02 | 03 | 10 | 11 | 12 | 13 | 20 | 21 | 22 | 23 | 30 | 31 | 32 | 33 |
| **0** | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 3 |
| **1** | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 2 | 0 | 1 | 0 | 0 | 3 |
| **2** | 2 | 0 | 0 | 0 | 2 | 1 | 0 | 0 | 2 | 0 | 2 | 0 | 2 | 0 | 0 | 3 |
| **3** | 3 | 0 | 0 | 0 | 3 | 1 | 0 | 0 | 3 | 0 | 2 | 0 | 3 | 0 | 0 | 3 |

**Tabela 3.7:** Estado futuro do *Flip-flop D (b)*

O circuito de memoria *Flip-flop D* apresentado na Figura 3.13 mostra uma alternativa de construção deste tipo de circuito, pode se observar que ela é construída tomando como circuito

base o *Latch RS*, a equação característica deste circuito é apresentado na Equação 3.14 como as saídas *Q* e *QQ* realimenta o circuito com o valor armazenado. Estas Equações descrevem o que seria o valor da entrada *Set* e *Reset* do *Latch RS*

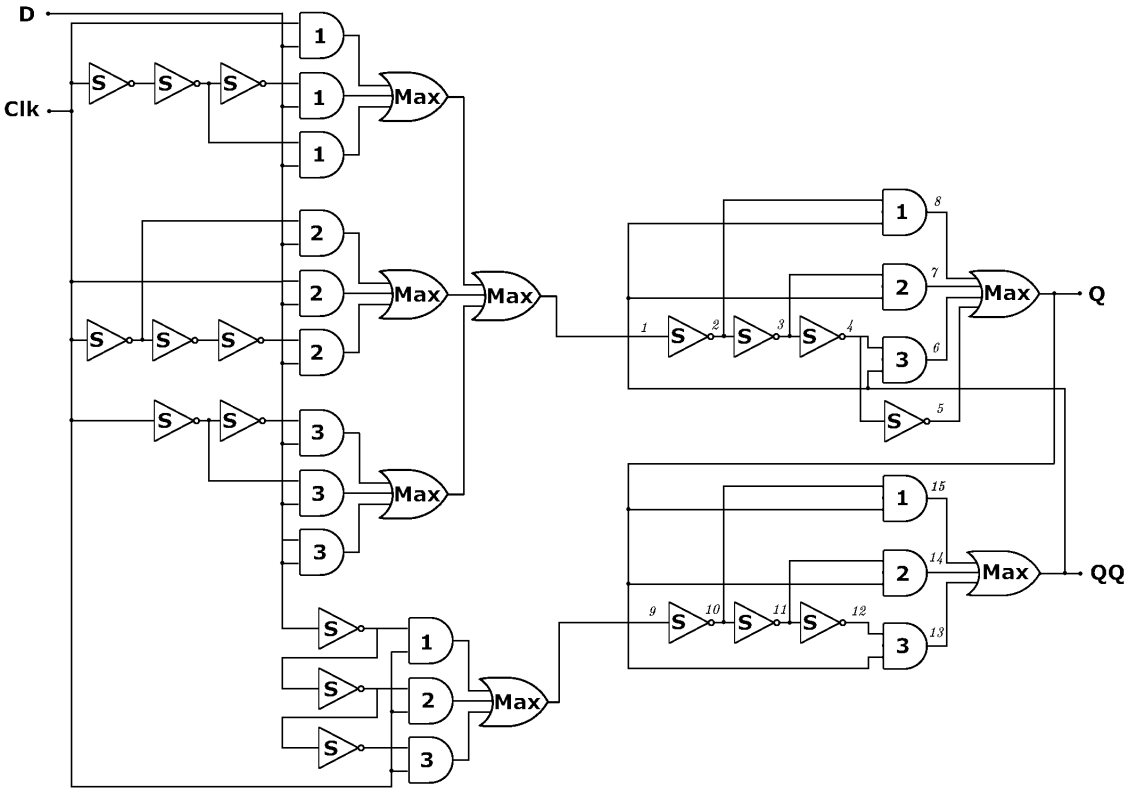



**Figura 3.13:** *Flip-flop D (a)*

$$\begin{aligned} Set \;\; = \;\; & (D \star^1 Clk) + (D \star^1 Clk^3) + (D \star^1 Clk^2) + (D \star^2 Clk) + (D \star^2 Clk^1) + \\ & (D \star^2 Clk^3) + (D \star^3 Clk^2) + (D \star^3 Clk^1) + (D \;\star^3 Clk). \end{aligned} \tag{3.14}$$

$$Reset = (D^1 \star^1 Clk) + (D^2 \star^2 Clk) + (D^3 \star^3 Clk) \tag{3.15}$$

Outra alternativa de construção do circuito *Flip-flop D* é apresentada na Figura 3.14, mostrando um circuito com menos quantidade de portas com relação ao circuito apresentado na Figura 3.13. A forma de operar deste novo circuito é um pouco diferente do primeiro, isto pode ser percebido na equação característica mostrada na Equação 3.16 com entradas *D*, *Clk* e saída *Q* e *QQ*.

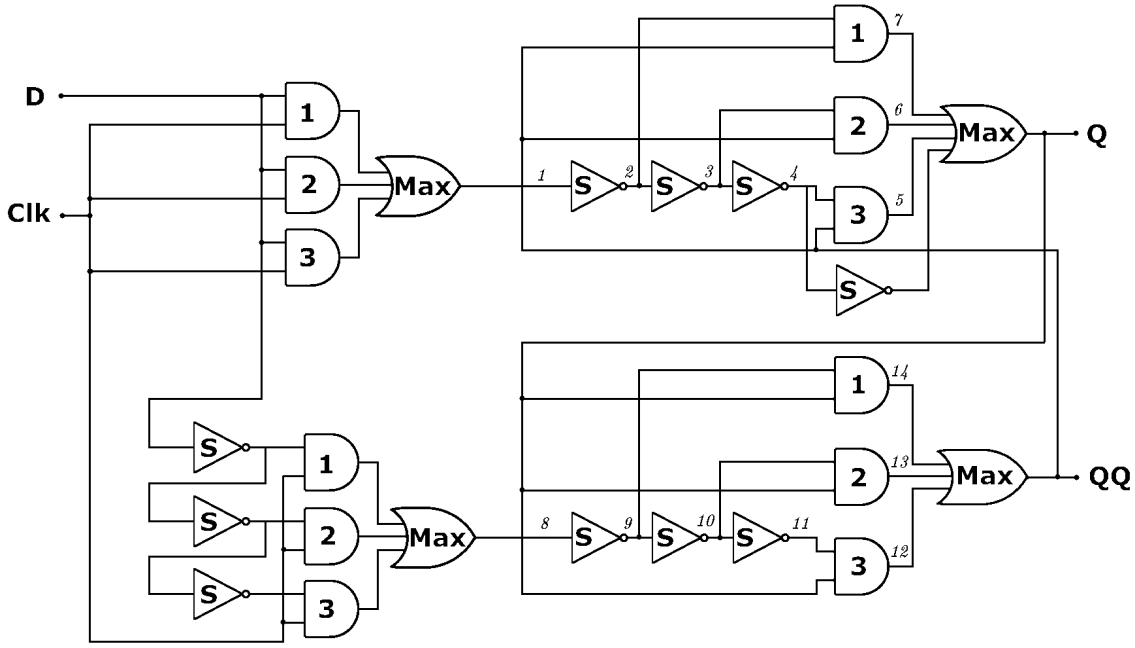



**Figura 3.14:** *Flip-flop D (b)*

$$Set \quad = \quad (D \star^1 Clk) + (D \star^2 Clk) + (D \star^3 Clk) \tag{3.16}$$

$$Reset = (D^1 \star^1 Clk) + (D^2 \star^2 Clk) + (D^3 \star^3 Clk) \tag{3.17}$$

### Diagrama de Estados Para o *Flip-flop D*

O diagrama de estados é uma maneira de representar o comportamento do circuito *Flip-flop D* mostrado na Figura 3.15, que apresenta quatro estados: o estado *reset, set1, set2 e set3*, representados por nós, e suas respectivas transições entre estados que são representados por uma seta. Neste caso utilizaremos a máquina de estado *Mealy*, cujo diagrama de estado descreve o comportamento do circuito apresentado na Figura 3.13

No diagrama mostrado na Figura 3.16 (b) é apresentado o comportamento do *Flip-flop D* mostrado na Figura 3.14 (b). Por exemplo se tomamos como estado inicial o estado *reset*, para poder passar ao estado set2 necessariamente as entradas D = 2 e Clk = 2, se o Clk estiver em outro estado que não seja 2, ele não mudará de estado.

### Simulação em VHDL e Discussão dos Resultados Para o *Flip-flop D*

A seguir são apresentadas as simulações do *Flip-flop tipo D*, a simulação mostrada na Figura 3.17 (a) corresponde ao circuito apresentado na Figura 3.13 (a). Pode-se observar a entrada D é memorizada na saída *Q* depois de um *delay*. Para que isto aconteça o *clock* tem que estar em

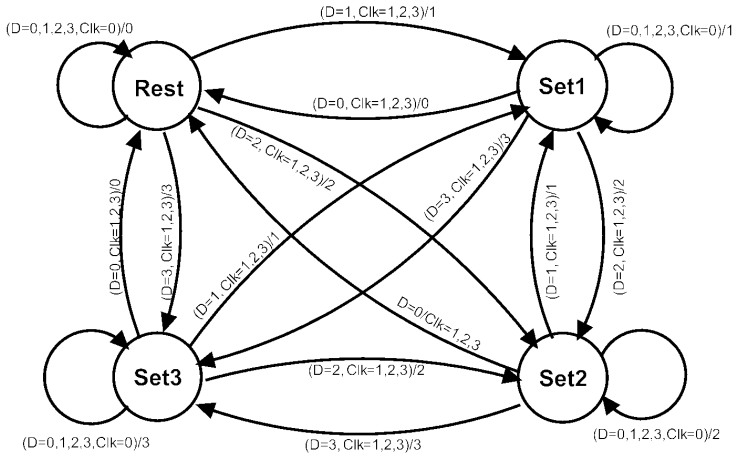



**Figura 3.15:** Diagrama de estados para o *Flip-flop D (a)*

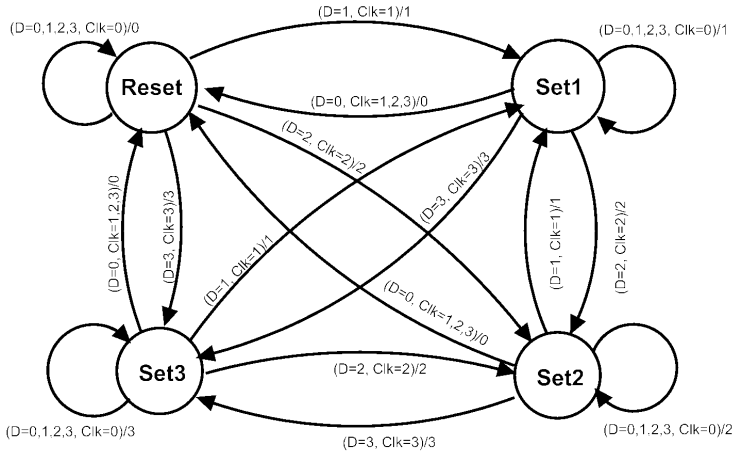



**Figura 3.16:** Diagrama de estados para o *Flip-flop D (b)*

estado ativo. Para este caso definimos o clock ativo quando ele está em 1, 2, 3, isto significa que ele memorizará qualquer valor da entrada *D* quando se encontra ativo. Esta definição do *clock* é similar ao *clock* binário, porque só temos 2 níveis: ativo e não ativo.

A simulação da Figura 3.18 (b) pertence ao circuito da Figura 3.14 (b), aqui define-se que o *clock* estará ativo em 1, ativo em 2, ativo em 3, sempre que a entrada D coincida com o *clock* e não ativo em 0 ou chamado de *hold*.

### 3.2.6 Síntese do *Flip-flop Master Slave*

O *Flip-flop Master-Slave* binário é um circuito síncrono de memória com uma entrada *set*, um sinal de *clock* e duas partes como saídas *master* e *slave*. Um método para prever o *race condition* ou *race hazard* é a utilização de dois *Latches master-slave*, quando o sinal do *clock*

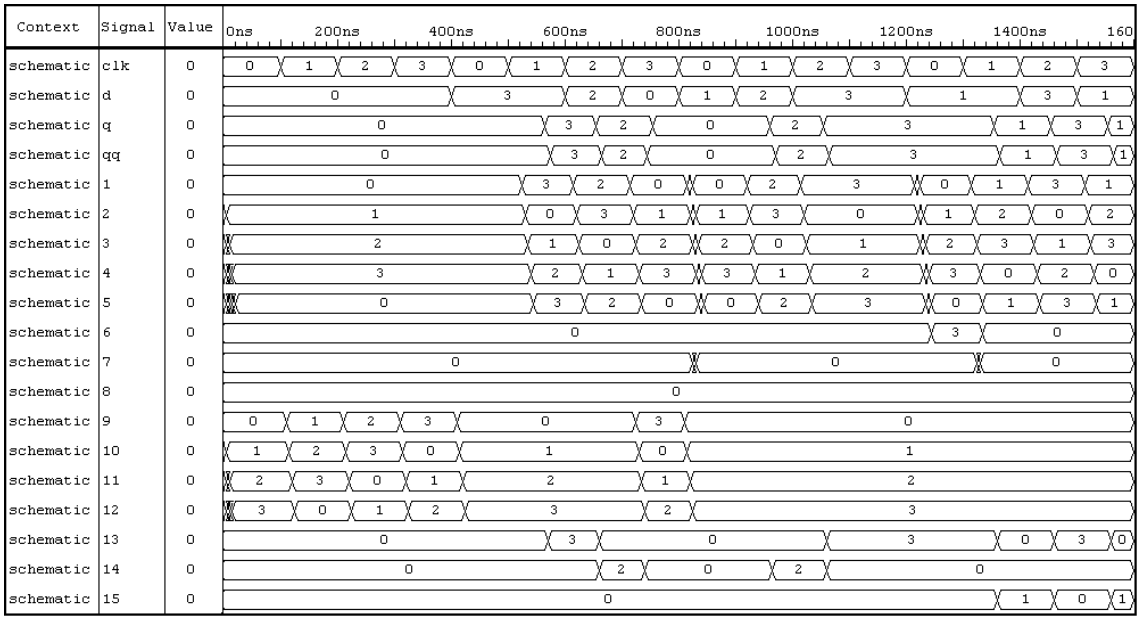



**Figura 3.17:** Simulação do *Flip-flop D (a)*

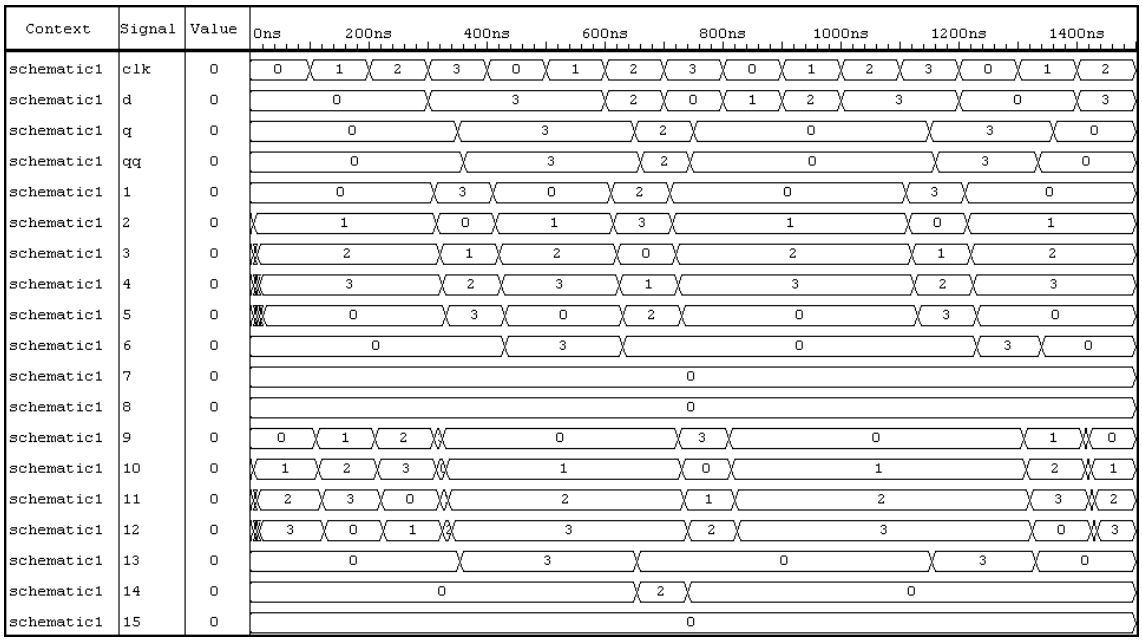



**Figura 3.18:** Simulação do *Flip-flop D (b)*

esta em nível baixo, o *latch master* fica em modo *gated* e o slave em modo *hold*. Neste caso o *master* copia o sinal da entrada *set* e o *slave* mantém seu valor ignorando o sinal de entrada. Quando o *clock* troca para o valor lógico 1, os dois *latches* trocam de função. O *latch slave* entra em modo *gated* recebendo o valor do *master* e enviando para a saída Q, enquanto o *latch master* entra em estado *hold* ignorando todas as trocas na entrada [31].

No caso MVL mantendo as funcionalidades como no binário, temos que *set*, *clock*, *QMaster*, *QSlave* $\in D = \{0, 1, 2, 3, ..., B-1\}$ sendo o domínio MVL . A entrada *set* é memorizada no latch *QMaster* sempre e quando se tenha o *clock* = 1, agora para passar o valor de *QMaster*

para *QSlave*, o *clock* tem que estar em *clock* = 0. O tempo no qual o *QMaster* toma o valor da entrada *Set* e o tempo no qual o *QMaster* passa o valor para *QSlave* está definido pelo *clock*.

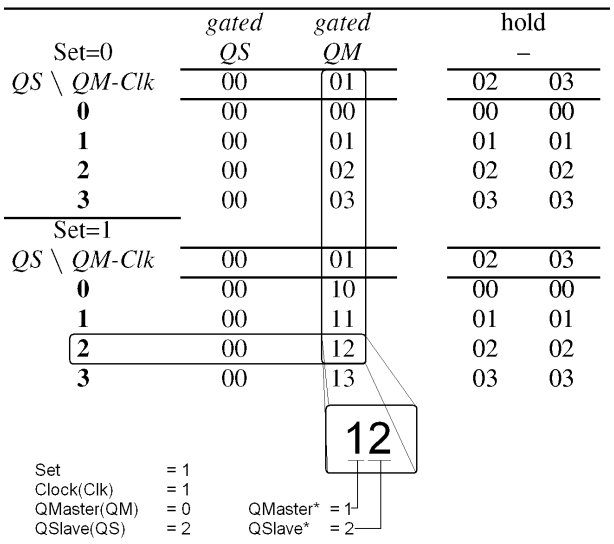



**Figura 3.19:** Interpretação da tabela 3.8 para o *Flip-flop Master Slave*

A Tabela 3.8 mostra o estado futuro de *QMaster* e *QSlave* em diferentes instantes que definem o *clock*. Como entradas temos *Set*; o *clock* para o sincronismo; as saídas são o *QMaster* que é descrito como *QM*; *QSlave* mostrado como *QS* e o valor futuro obtido é *QMaster* (QM*)* e *QSlave* (QS)** respectivamente. Para uma melhor compreensão da Tabela 3.8 utilizou-se a Figura 3.19, onde observa-se que para *Set* = 1, *Clk* = 1, *QM* = 0 e *QS* = 2. Neste caso quando o *Clk* = 1 o valor do *Set* é copiado para a saída do *QM* e o *QS* permanece inalterado.

| | *gated* | *gated* | hold | | *gated* | *gated* | hold | | *gated* | *gated* | hold | | *gated* | *gated* | hold | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Set=0 | *QS* | *QM* | – | | *QS* | *QM* | – | | *QS* | *QM* | – | | *QS* | *QM* | – | |
| *QS* \ *QM-Clk* | 00 | 01 | 02 | 03 | 10 | 11 | 12 | 13 | 20 | 21 | 22 | 23 | 30 | 31 | 32 | 33 |
| **0** | 00 | 00 | 00 | 00 | 11 | 00 | 10 | 10 | 22 | 00 | 20 | 20 | 33 | 00 | 30 | 30 |
| **1** | 00 | 01 | 01 | 01 | 11 | 01 | 11 | 11 | 22 | 01 | 21 | 21 | 33 | 01 | 31 | 31 |
| **2** | 00 | 02 | 02 | 02 | 11 | 02 | 12 | 12 | 22 | 02 | 22 | 22 | 33 | 02 | 32 | 32 |
| **3** | 00 | 03 | 03 | 03 | 11 | 03 | 13 | 13 | 22 | 03 | 23 | 23 | 33 | 03 | 33 | 33 |
| Set=1 | | | | | | | | | | | | | | | | |
| *QS* \ *QM-Clk* | 00 | 01 | 02 | 03 | 10 | 11 | 12 | 13 | 20 | 21 | 22 | 23 | 30 | 31 | 32 | 33 |
| **0** | 00 | 10 | 00 | 00 | 11 | 10 | 10 | 10 | 22 | 10 | 20 | 20 | 33 | 10 | 30 | 30 |
| **1** | 00 | 11 | 01 | 01 | 11 | 11 | 11 | 11 | 22 | 11 | 21 | 21 | 33 | 11 | 31 | 31 |
| **2** | 00 | 12 | 02 | 02 | 11 | 12 | 12 | 12 | 22 | 12 | 22 | 22 | 33 | 12 | 32 | 32 |
| **3** | 00 | 13 | 03 | 03 | 11 | 13 | 13 | 13 | 22 | 13 | 23 | 23 | 33 | 13 | 33 | 33 |
| Set=2 | | | | | | | | | | | | | | | | |
| *QS* \ *QM-Clk* | 00 | 01 | 02 | 03 | 10 | 11 | 12 | 13 | 20 | 21 | 22 | 23 | 30 | 31 | 32 | 33 |
| **0** | 00 | 20 | 00 | 00 | 11 | 20 | 10 | 10 | 22 | 20 | 20 | 20 | 33 | 20 | 30 | 30 |
| **1** | 00 | 21 | 01 | 01 | 11 | 21 | 11 | 11 | 22 | 21 | 21 | 21 | 33 | 21 | 31 | 31 |
| **2** | 00 | 22 | 02 | 02 | 11 | 22 | 12 | 12 | 22 | 22 | 22 | 22 | 33 | 22 | 32 | 32 |
| **3** | 00 | 23 | 03 | 03 | 11 | 23 | 13 | 13 | 22 | 23 | 23 | 23 | 33 | 23 | 33 | 33 |
| Set=3 | | | | | | | | | | | | | | | | |
| *QS* \ *QM-Clk* | 00 | 01 | 02 | 03 | 10 | 11 | 12 | 13 | 20 | 21 | 22 | 23 | 30 | 31 | 32 | 33 |
| **0** | 00 | 30 | 00 | 00 | 11 | 30 | 10 | 10 | 22 | 30 | 20 | 20 | 33 | 30 | 30 | 30 |
| **1** | 00 | 31 | 01 | 01 | 11 | 31 | 11 | 11 | 22 | 31 | 21 | 21 | 33 | 31 | 31 | 31 |
| **2** | 00 | 32 | 02 | 02 | 11 | 32 | 12 | 12 | 22 | 32 | 22 | 22 | 33 | 32 | 32 | 32 |
| **3** | 00 | 33 | 03 | 03 | 11 | 33 | 13 | 13 | 22 | 33 | 23 | 23 | 33 | 33 | 33 | 33 |

**Tabela 3.8:** Estado futuro para o *Flip-flop Master Slave*

$$\textit{SetMaster} = Set \star^3 Clk^2 + Set \star^2 Clk^1 + Set \star^1 Clk \tag{3.18}$$

$$\textit{ResetMaster} = Set^1 \star^1 Clk + Set^2 \star^2 Clk + Set^3 \star^3 Clk \tag{3.19}$$

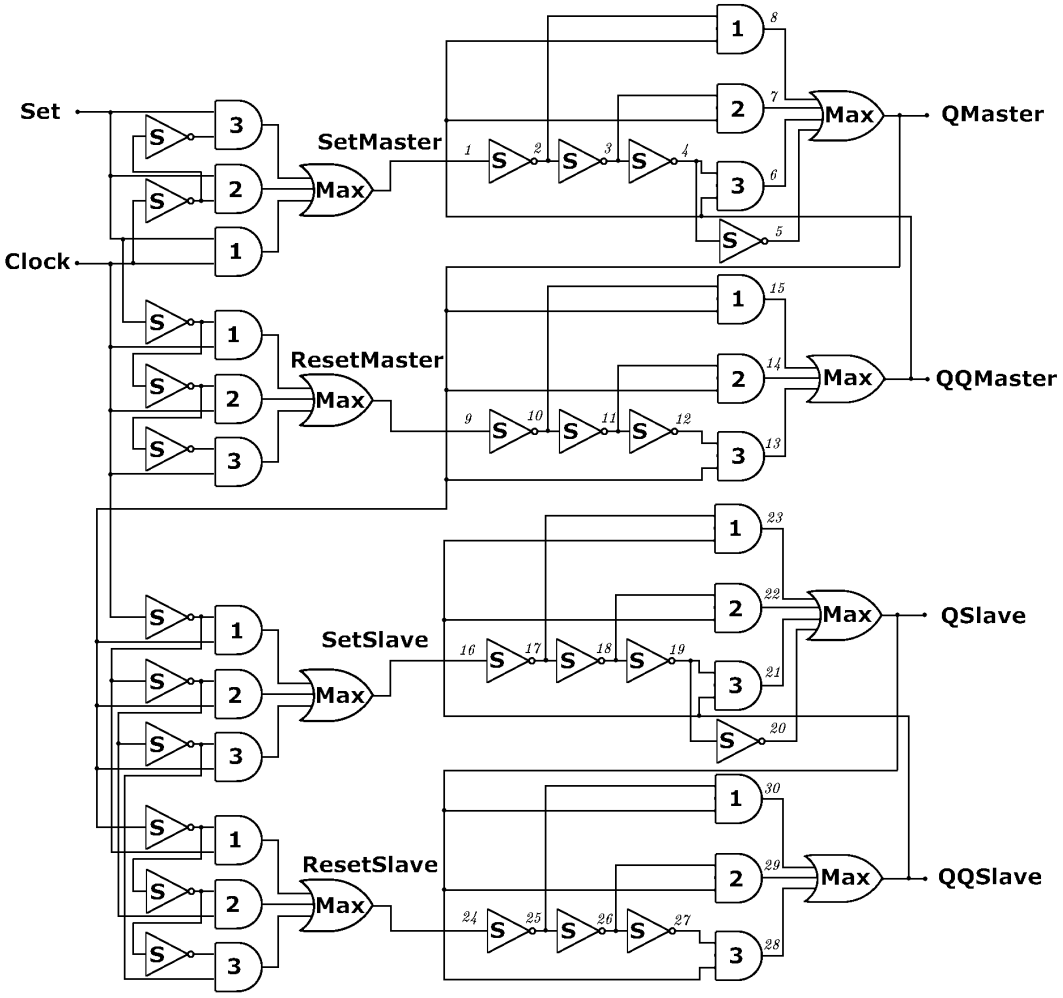



**Figura 3.20:** *Flip-flop Master Slave*

$$\textit{SetSlave} = Clk^1 \star^1 QMaster + Clk^2 \star^2 QMaster + Clk^3 \star^3 QMaster \tag{3.20}$$

$$\textit{ResetSlave} = QMaster^1 \star^1 Clk^1 + QMaster^2 \star^2 Clk^2 + QMaster^3 \star^3 Clk^3 \tag{3.21}$$

### Simulação em VHDL e Discussão dos Resultados Para o *Flip-flop Master Slave*

A simulação do *flip-flop* tipo *master-slave* se realizou considerando o tempo de *delay* em nanosegundos(ns), com a descrição das portas em VHDL, adaptando para o caso MVL. Para projetar o circuito foram utilizadas as portas sucessor (*Suc*) com *delay* = 10ns, *AND* estendido (1,2 e 3) com *delay* = 100ns e o *MAX* de 2 entradas com *delay* = 100ns, e o de 4 entradas com *delay* = 200ns. O resultado desta simulação pode-se observar na Figura 3.21. Onde observar-se

que o *QM* copia o valor da entrada *Set* somente quando o *Clk* = 1, e o *QS* copiará o valor do *QM* quando *Clk* = 0.

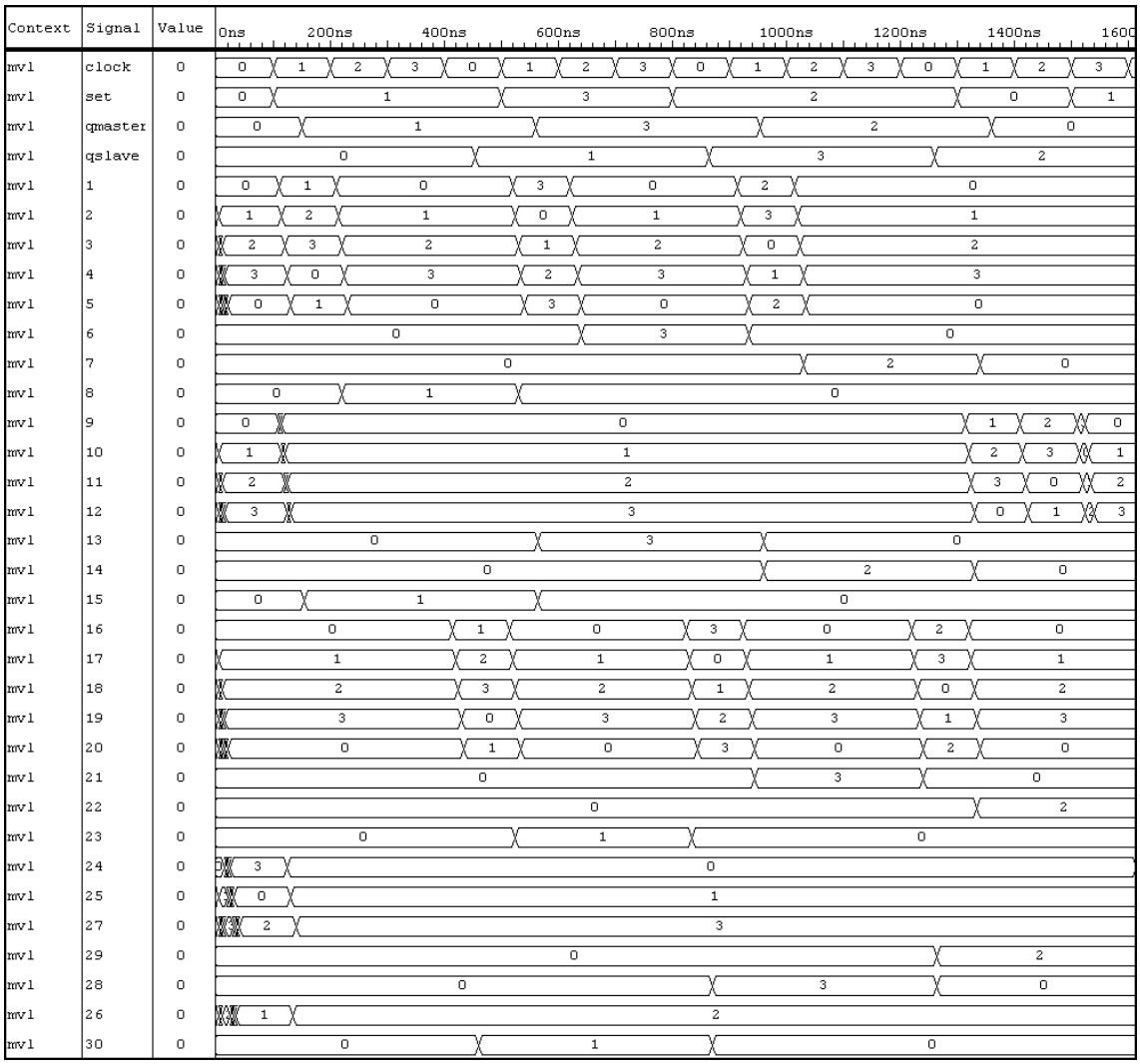



**Figura 3.21:** Simulação do *Flip-flop Master Slave*

CAPÍTULO

4

# Simplificação de Circuitos Seqüenciais MVL

Neste capítulo abordam-se os métodos de redução de estados para circuitos seqüenciais completamente especificados e incompletamente especificados, como uma extensão do binário para o caso MVL. A importância de reduzir estados redundantes em circuitos seqüenciais é permitir a redução de custo (reduzindo o número de elementos de memória), redução de complexidade (menor quantidade de estados), facilitar a análise de erros (a análise é feita supondo que não existem estados redundantes).

## 4.1 Estados Redundantes

Consideram-se estados redundantes se dois ou mais estados são equivalentes, ou seja, quando não é possível distinguir entre os estados e quando não é possível determinar em qual dos estados o circuito é inicializado. Se for dado o caso de estados equivalentes significa que existem estados redundantes e que estes podem ser retirados sem alterar o comportamento do circuito. Estes estados redundantes são retirados na fase inicial do projeto, ou seja, quando a descrição verbal é abstraída para uma tabela de estados que descreve o comportamento do circuito.

## 4.2 Estados Equivalentes

Dado os estados $E_1, E_2, ..., E_n$ que descrevem o comportamento de um circuito seqüencial pode-se afirmar que estes estados são equivalentes se e somente se para cada seqüência de

entrada existe uma seqüência de saída similar produzida pelo circuito. Uma forma alternativa de definir a equivalência de estados é por meio de um exemplo. Supondo que se tem dois estados $E_i, E_j$, uma seqüência de entrada $I_n$ e os seguintes estados $E_k$, $E_l$. Se sobre o circuito for aplicado uma seqüência de entrada $I_n$, pode-se afirmar que $E_i$ e $E_j$ são equivalentes se e somente se, para cada possível entrada $I$, a saída produzida pelo estado $E_i$ é igual à saída produzida pelo estado $E_j$, portanto conclui-se que $E_k$ e $E_l$ são equivalentes. Para afirmar a equivalência de estados é necessária que os estados garantam as duas condições mencionadas a seguir.

1. As saídas produzidas para os estados $E_i$ e $E_j$ são iguais e com os seguintes estados $E_k$ e $E_l$ respectivamente (condição nessesária).

2. Então os estados $E_k$ e $E_l$ são equivalentes (condição suficiente).

## 4.3 Relação de Equivalência

Dados dois elementos $e_i$ e $e_j$ que pertencem ao conjunto de estados $E$ e que são relacionados por uma propriedade $r$ chamada de relação $R$ sobre o conjunto $E$, de modo que esta relação pode ser representada como $e_i\ r\ e_j$ ou ($e_i$, $e_j$). Uma relação de equivalência sobre o conjunto $E$ é uma relação que é simétrica, reflexiva e transitiva, descritas a seguir. Os elementos de $E$ podem ser classificados mediante uma relação de equivalência em subconjuntos disjuntos chamados classes de equivalência, estas classes são as que ajudam a definir a tabela de estados reduzida.

1. R é reflexiva se e somente se $e_i\ r\ e_i$ para todo elemento $e_i$ em E.

2. R é simétrica se e somente se $e_i\ r\ e_j$ implica que $e_j\ r\ e_i$ em E.

3. R é transitiva se e somente se $e_i\ r\ e_j$ e $e_j\ r\ e_k$ implica que $e_i\ r\ e_k$ em E.

## 4.4 Relação de Compatibilidade

Considera-se como uma relação de compatibilidade sobre um conjunto de estados $E$, se e somente se, a relação é reflexiva e simétrica. A relação de compatibilidade define subconjuntos de $E$ conhecido como classes de compatibilidade.

## 4.5 Redução de Estados em Circuitos Completamente Especificados

O problema de redução de estados em circuitos completamente especificados na lógica binária pode ser resolvido em tempo polinomial [34], existem três técnicas, as quais são adaptadas para o caso de circuitos MVL que serão descritos a seguir.

## 4.6 Método por Inspeção

Este método é o mais simples dos apresentados para redução de estados completamente especificados, mas com algumas limitações já que existem estados redundantes que não podem ser detectados por simples inspeção. Este método permite reconhecer estados equivalentes por meio de uma inspeção visual sobre a tabela de estados. Para isto precisa-se reconhecer as múltiplas linhas apresentadas na tabela de estados, onde todos os estados que tenham uma função similar poderão ser removidas com o critério de equivalência, já que estes serão considerados como estados redundantes.

Por exemplo, para o primeiro caso apresentado na Figura 4.1 (b), pode-se observar que para uma seqüência de entrada no estado C e D é obtida uma seqüência de saída similar para o seguinte estado e a seguinte saída. Isto implica que o estado C e D são equivalentes, permitindo obter uma tabela reduzida que é mostrada na Figura 4.1 (c).

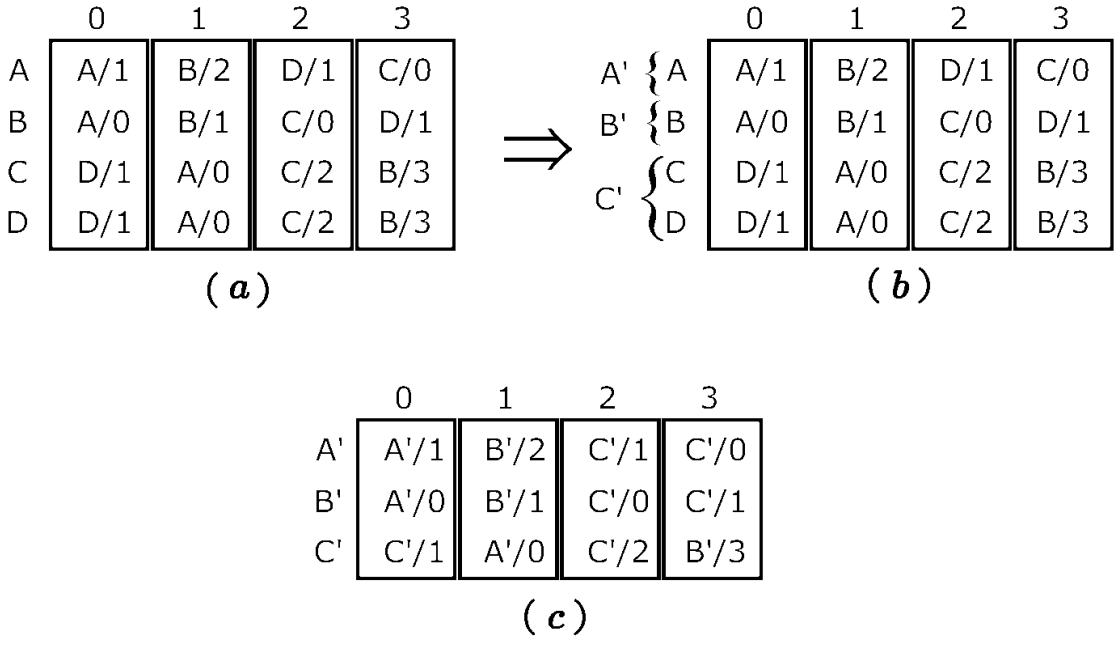



**Figura 4.1:** Redução de estados pelo método de inspeção (caso 1)

No segundo caso apresentado na Figura 4.2 (b) onde se pode observar uma tabela de estados que descreve o comportamento do circuito digital MVL, por exemplo, supondo-se que o circuito encontra-se no estado C, se for aplicada uma entrada igual a 0, será obtido como resultado, o mesmo estado C com sua respectiva saída igual a 1. Nota-se que neste caso ocorre um laço sobre o estado, o qual é considerado como estado equivalente. Outro caso é quando o circuito está no estado D: e se for aplicada uma entrada igual a 0 será obtido como estado seguinte o mesmo estado D com sua saída igual a 1. Neste caso também existe um laço no estado D para a entrada igual a 0. Para o resto das entradas, o resultado obtido é de equivalência já que não existe variação com relação ao estado seguinte e sua respectiva saída quando for aplicada uma entrada. Desde que o estado C e D sejam equivalentes pode-se obter uma tabela reduzida como é mostrada na Figura 4.2 (c).

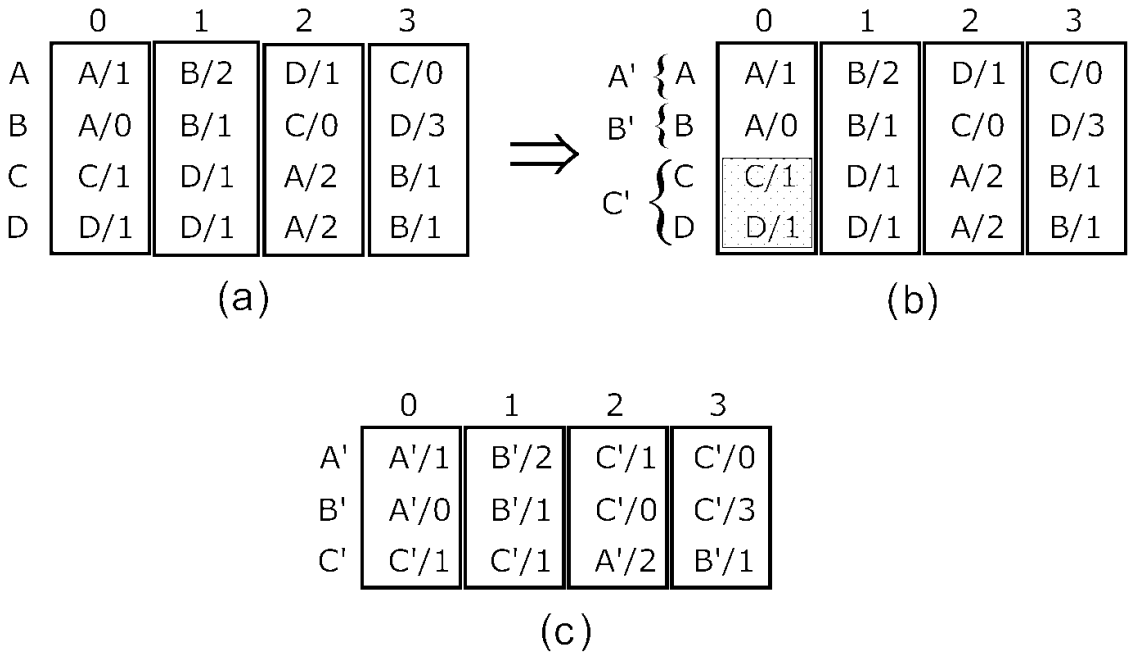



**Figura 4.2:** Redução de estados pelo método de inspeção (caso 2)

Como terceiro caso, é apresentado um circuito digital com o comportamento descrito pela tabela de estados mostrado na Figura 4.3 (b). Supondo que o circuito se encontra no estado C (terceira linha da tabela mostrado na Figura 4.3 (b)) é aplicada uma entrada igual a 0 e como resultado é obtido o estado D com saída igual a três. Agora, se o circuito se encontra no estado D (que é a quarta linha da tabela da Figura 4.3 (b)), quando for aplicada uma entrada igual a 0 é obtida a seguir o estado C com saída igual a 3. Neste caso os estados C e D se convertem num laço quando a entrada é igual a 0, e estes dois estados podem ser tomados como estados equivalentes se o circuito para o resto das entradas satisfaz a condição de equivalência como mostrada na tabela apresentado na Figura 4.3 (C).

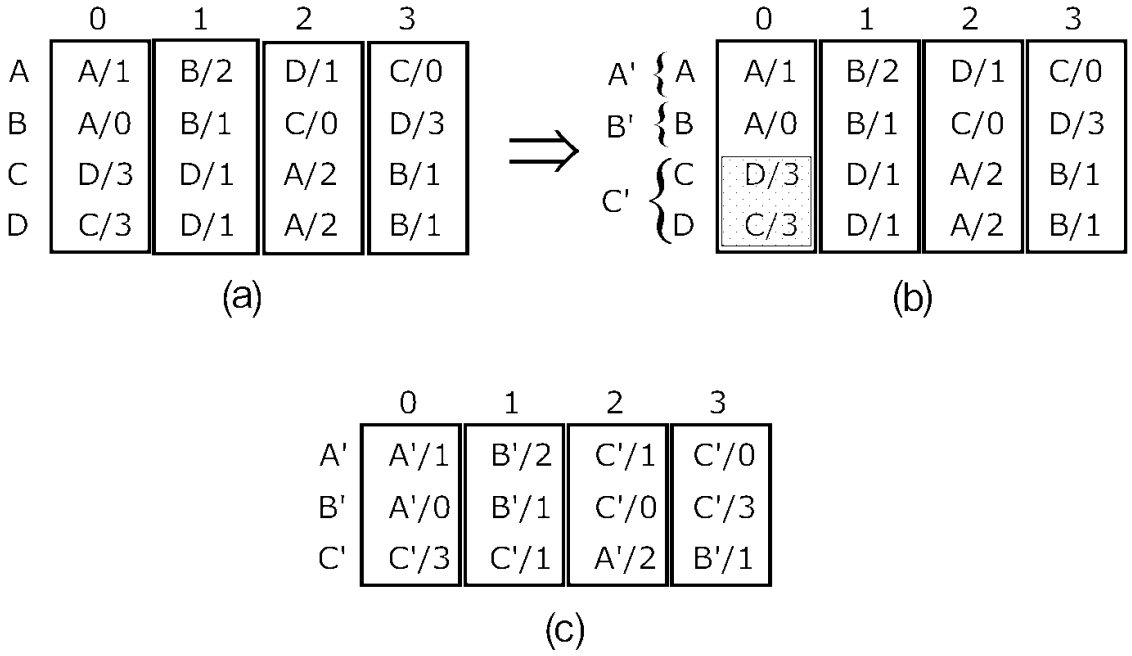



**Figura 4.3:** Redução de estados pelo método de inspeção (caso 3)

## 4.7 Método por Partição

O método de redução de estados pelo método partição envolve sucessivas partições $P_i$, onde cada partição é composta por um ou mais grupos contendo estados. Aqueles estados contidos num grupo são considerados como equivalentes ao resto de estados que pertencem a esse grupo.

Por exemplo, se temos um circuito seqüencial que tem os estados $E_1, E_2, E_3, E_4, E_5$, se a partição $P_i = (E_1), (E_2, E_3, E_4)(E_5)$, neste caso a partição $P_i$ contém três grupos onde $E_2, E_3, E_4$ são equivalentes, e no caso de $E_1$ e $E_5$ estes dois últimos estados são considerados como não equivalentes a nenhum estado daquela partição.

### 4.7.1 Algoritmo do Método por Partição

- Passo 1. A primeira partição $P_1$ é formada por dois ou mais agrupamentos dos estados, estes agrupamentos são feitos observando para cada entrada que vai gerar uma saída idêntica.

- Passo 2. As sucessivas partições $P_i$, $i = 2, 3, 4, ..., n$ onde os estados são agrupados em dois ou mais estados, é levado acabo o procedimento iterativo tendo presente a condição 2 de equivalência (relação de equivalencia) mencionada anteriormente.

- Passo 3. Quando a partição $P_{i+1}$ é igual à partição $P_i$, ou seja, quando a partição se repete, isto implica que não é possível conseguir mais partições, chegando a concluir que aquela partição é a redução mínima de estados por meio deste algoritmo.

A Figura 4.4 apresenta uma descrição do circuito digital mediante diagramas de estados (7 estados), os nós são conectados por meio de setas e cada seta leva consigo a informação do valor de entrada e a saída.

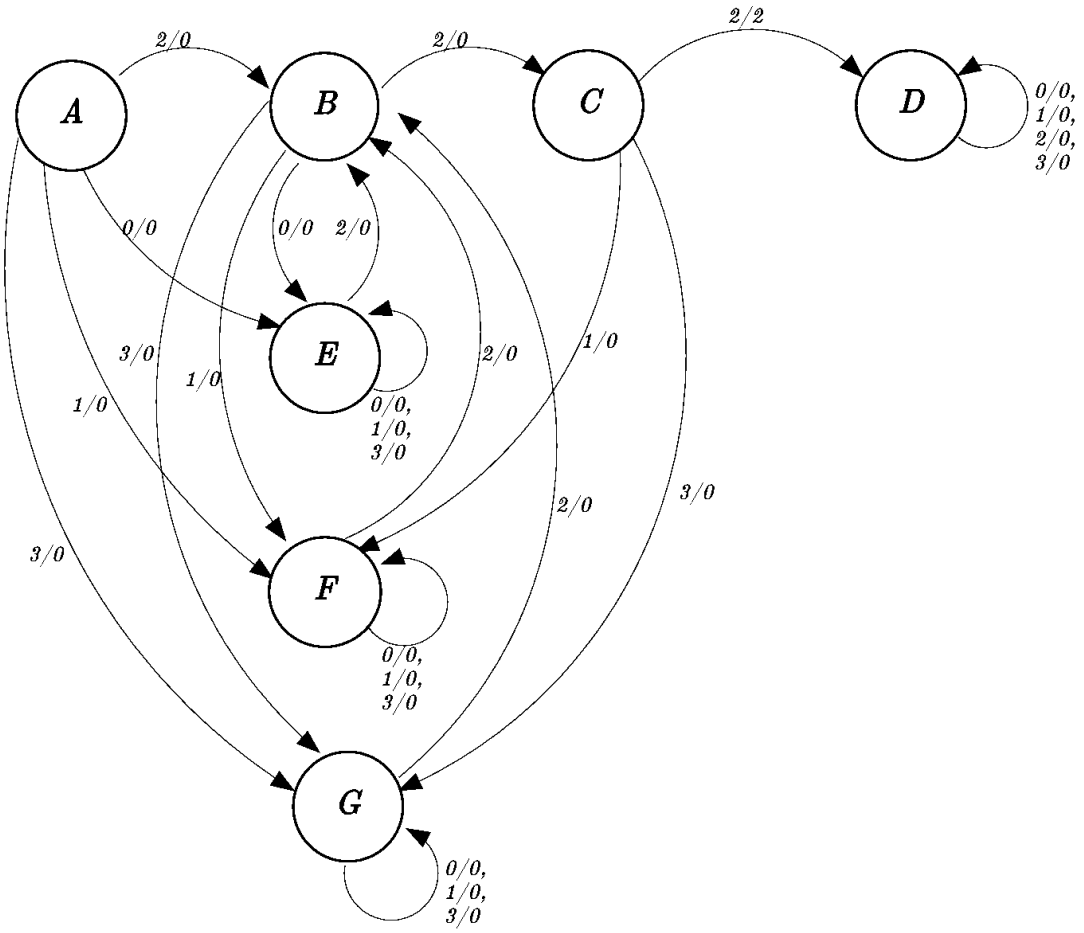



**Figura 4.4:** Exemplo: diagrama de estados

A tabela de estados mostrada na Figura 4.5, apresenta os estados na parte esquerda, as entradas na parte superior e a intersecção destas duas entradas resulta num estado seguinte com sua respectiva saída. Esta apresenta os estados A, B, C, D, E, F, G, e com entradas e saídas que tomam valores lógicos de 0, 1, 2, 3, o tipo de máquina de estado a ser utilizado nesta descrição é a Mealy.

| Estado Presente \ Entrada | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| A | E/0 | F/0 | B/0 | G/0 |
| B | E/0 | F/0 | C/0 | G/0 |
| C | E/0 | F/0 | D/2 | G/0 |
| D | D/0 | D/0 | D/0 | D/0 |
| E | E/0 | E/0 | B/0 | E/0 |
| F | F/0 | F/0 | B/0 | F/0 |
| G | G/0 | G/0 | B/0 | G/0 |

**Figura 4.5:** Exemplo: tabela de estado

Para um melhor entendimento do algoritmo, é utilizado como exemplo a tabela de estados apresentado na Figura 4.5. O objetivo é reduzir a quantidade de estados utilizando o método por partição cujo procedimento de redução de estados é mostrado na Figura 4.6. A partir da partição inicial $P_0$ onde todos os estados formam um bloco $(ABCDEFG)$ será obtida a primeira partição $P_1$ avaliando as saídas em cada estado com a finalidade de formar novos grupos. Neste caso o novo grupo é formado pelo estado C, pelo fato de que quando se apresenta uma entrada igual a 2 será obtido como resultado o estado seguinte D e com saída igual a 2, como se está avaliando somente as saídas, nota-se que somente neste caso a saída é diferente, enquanto o resto dos estados tem a saída igual a 0.

Uma vez obtida a partição $P_1$ onde os novos grupos são (ABDEFG) e (C), o seguinte passo é obter a partição $P_2$ a partir da partição $P_1$. Nesta etapa é avaliado o seguinte estado para todas as entradas. Por exemplo, se é avaliada a coluna do estado B da partição $P_1$, para as entradas x = 0, 1, 2, 3, nota-se somente quando a entrada é igual a x = 2 é obtido o estado C como seguinte estado. Revisando na listagem horizontal da partição $P_1$ no primeiro grupo não se encontra o estado C. Isto significa que a coluna do estado B será separada como um novo grupo, obtendo-se como novos grupos (ADEFG)(B)(C) listados na partição $P_2$.

Este procedimento é repetido de forma iterativa até que a partição atual seja idêntica à partição anterior como descrito no passo 3. Neste caso observa-se que a partição $P_4$ é idêntica a partição $P_3$ a qual indica que é o resultado de redução de estados por meio deste algoritmo, dando como resultado de redução os estados temos os novos estados $E_0$ = (AEFG), $E_1$ = (D), $E_2$ = (B), $E_3$ = (C) como é apresentado na Figura 4.7.

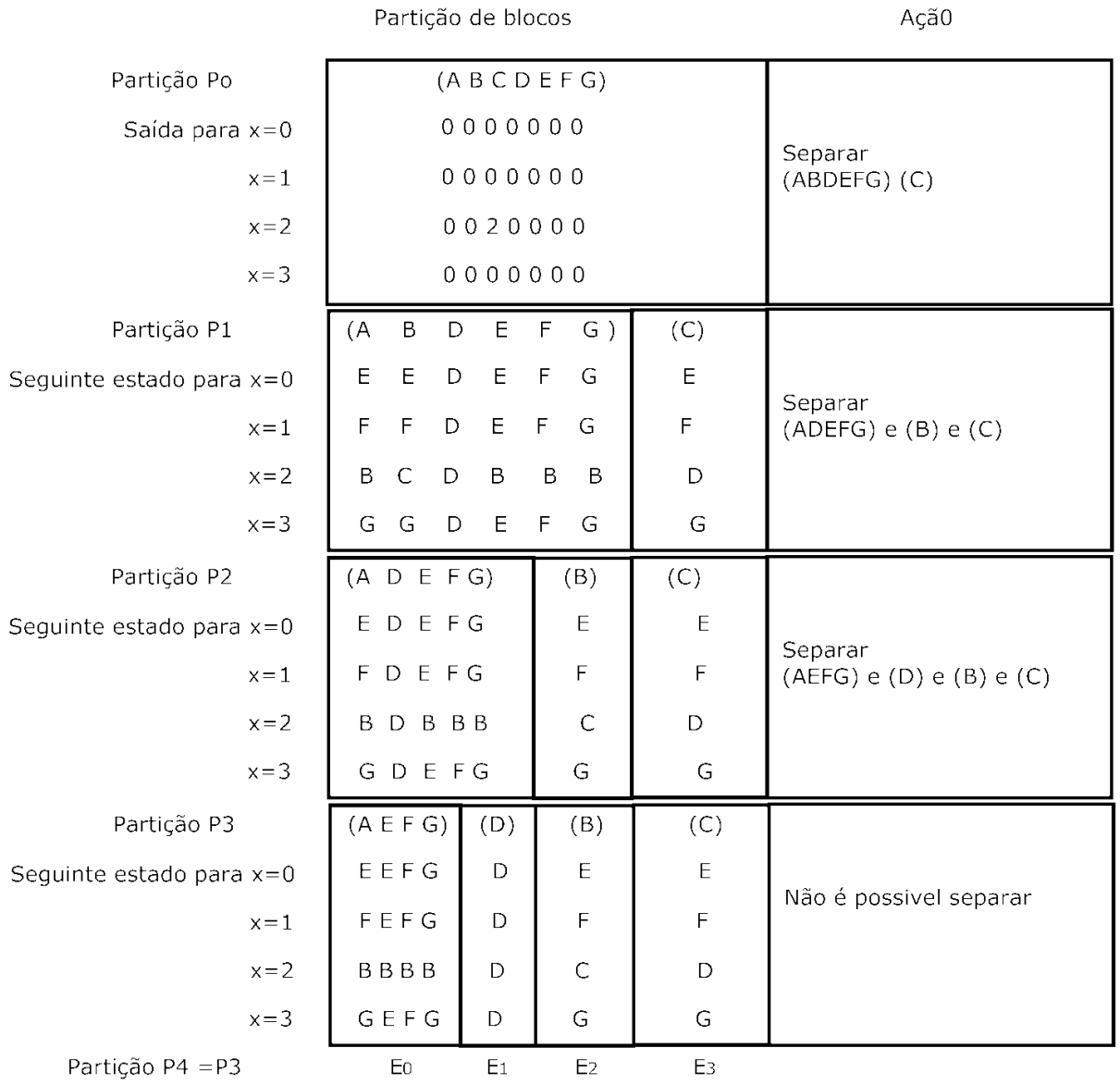



**Figura 4.6:** Exemplo: procedimento de redução de estados pelo método de partição

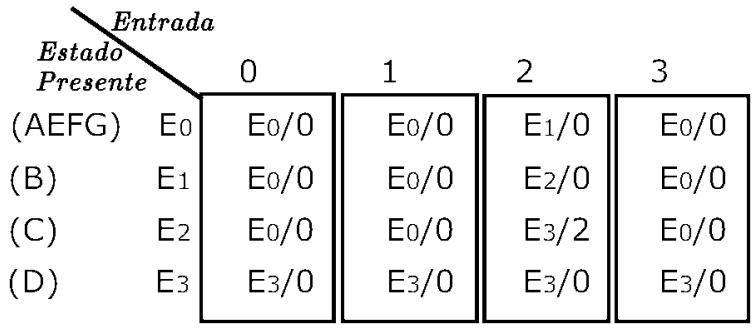



**Figura 4.7:** Exemplo: tabela de estados reduzida pelo método de partição

## 4.8 Método por Tabela de Implicação

Este método é outra ferramenta que pode ser usada para a redução de estados em circuitos seqüenciais completamente especificados com o critério de equivalência de estados. Uma característica deste método é a de poder ser usado para reduzir estados seqüenciais incompletamente especificados, em alguns casos o método pode tomar mais tempo do que pelo método de partição devido à quantidade de comparações que se realiza no procedimento de redução de estados.

### 4.8.1 Algoritmo do Método por Tabela de Implicação

Este algoritmo é apresentado por etapas como é mostrado a seguir:

- Passo 1. Como primeiro passo, oobjetivo é formar uma tabela utilizando a estrutura como mostrada na Figura 4.8, que é obtido pela listagem vertical de todos os estados na tabela, à exceção do último e horizontalmente todos os estados exceto o primeiro. A tabela resultante mostra todas as combinações possíveis dos estados e, conseqüentemente, cada célula na tabela corresponde à intersecção de uma linha e uma coluna representando dois estados que serão avaliados.

- Passo 2. Somente os estados que tenham resultados idênticos nas saídas podem ser equivalentes segundo a condição 1 de equivalência para estados (condição nessesária), os estados cuja seqüência de saídas não sejam iguais para cada seqüência de entrada devem ser marcadas com um ($X$) nas células correspondentes aos estados comparados, este procedimento é realizado até preencher todas as células como é mostrado na Figura 4.9 (c).

- Passo 3. As células vazias devem ser preenchidas usando o critério 2 de equivalência (condição suficiente), como mostra a Figura 4.9 (d). Dentro de cada célula são colocados os pares de estados cuja equivalência é implícita para os dois estados onde a interseção define a célula. Por exemplo, dada a célula definida pelos estados A e B. Esta célula é preenchida de acordo à comparação realizada entre a linha do estado A e o estado B. Para cada entrada como é mostrada na Figura 4.9 (a), nesta comparação pode-se observar que o resultado para os estados A e B com uma entrada igual a 2 o seguinte estado é B e C respectivamente. Enquanto que para o resto de entradas o seguintes estados são idênticos, o que nos leva à afirmação: A e B são equivalentes se B e C são equivalentes.

  Assim, o par B e C são listados na célula definida por A e B, como na Figura 4.9 (d). Este procedimento é realizado até verificar todas as células da tabela de implicação. Se o par implícito de qualquer célula contém apenas os estados que definem a célula ou se o próximo estado da intersecção define que as células são estados similares para uma

determinada entrada, então, marca-se a célula com um check (✓) indicando que os dois estados são equivalentes por inspeção e independente de algum par implícito, como nos casos das células (E, F) ou (E, G) ou (F, G), ou em caso contrário riscadas ($\chi$).

- Passo 4. Uma vez que a tabela tenha sido marcada e preenchida, e com a finalidade de determinar se alguma célula deve ser riscada, como no passo 2, verifica-se a equivalência implícita dos estados previamente preenchidos. Por exemplo, a célula definida por A e B é riscada porque ela continha o par BC que define uma célula que já foi riscada. Este procedimento é repetido até que todas as células sejam marcadas, como é mostrado na Figura 4.9 (e) .

- Passo 5. Finalmente a Figura 4.9 (f) apresenta a listagem de como uma coluna de estados define a linha horizontal da tabela de implicações. A seguir a tabela de implicações é examinada coluna por coluna da esquerda para a direita verificando se algumas das células não foram marcadas. Aqueles estados que não foram riscados são o par de estados equivalentes.

A geração da tabela de implicação é realizada a partir de uma tabela de intersecção como mostra o exemplo da Figura 4.8 para os estados A, B, C, D, E.

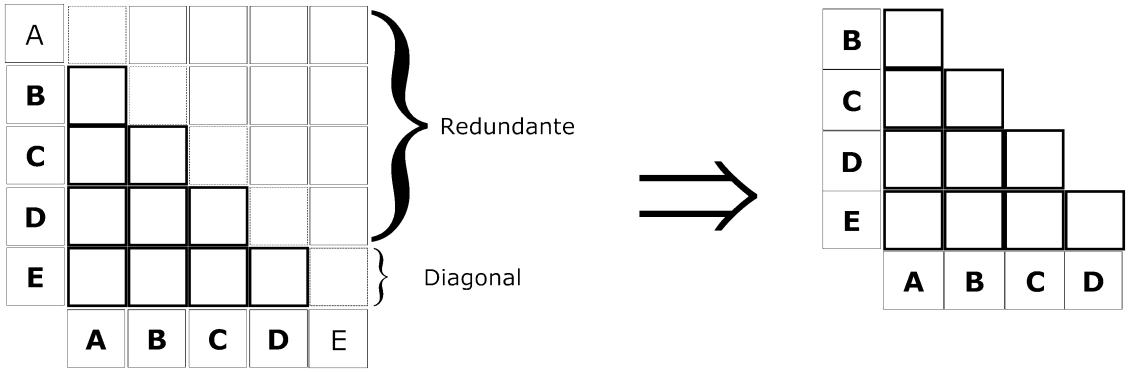



**Figura 4.8:** Tabela de implicação

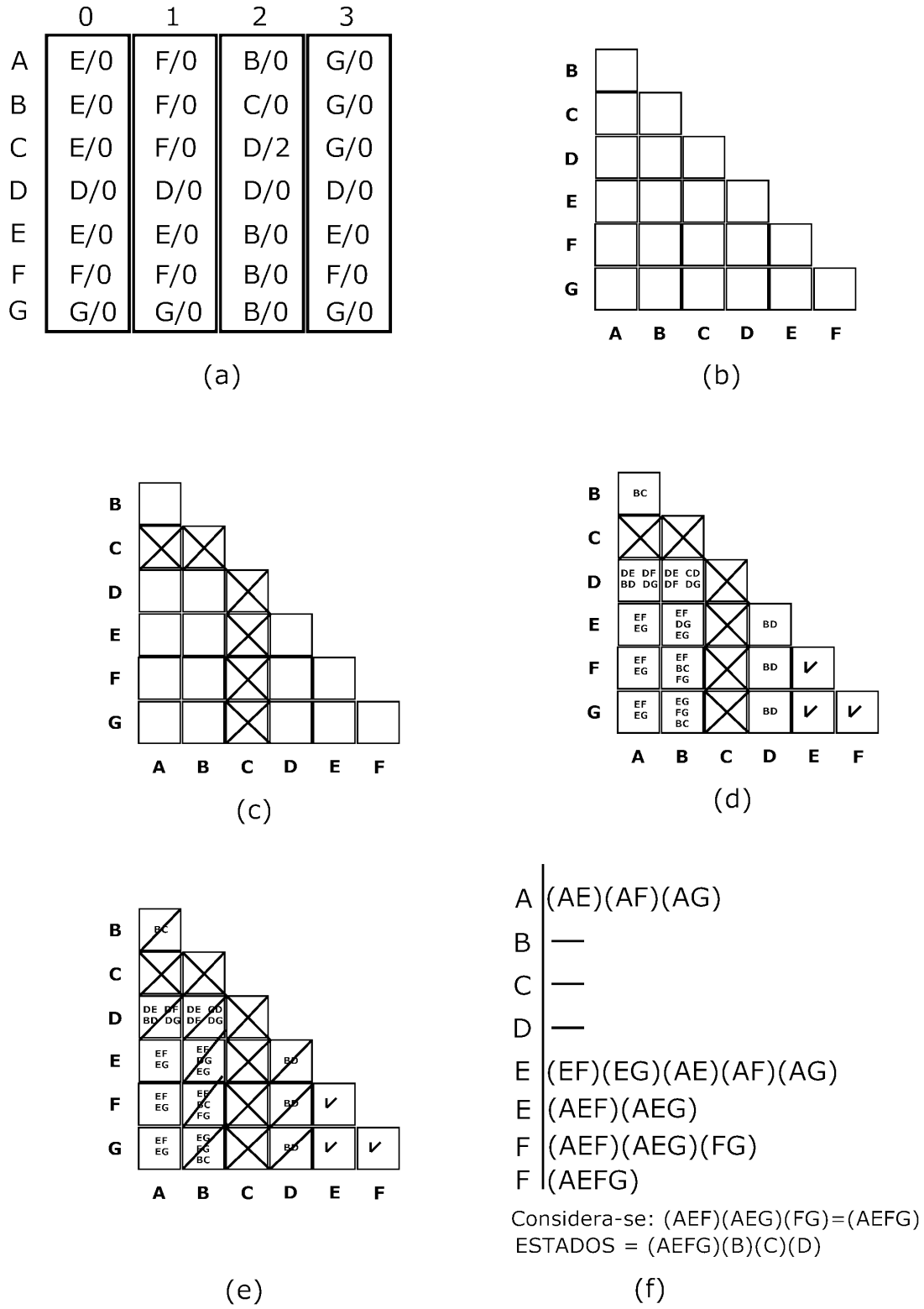



**Figura 4.9:** Procedimento de redução por tabela de implicação

## 4.9 Redução de Estados em Circuitos Incompletamente Especificados

A redução de estados em máquinas de estado finito (FSM) é um dos problemas de síntese de circuitos seqüenciais binários. Uma tabela de estados contém estados redundantes que podem ter sido introduzidos pelo projetista. A eliminação destes estados redundantes reduz a lógica que precisa para ser implementada ou sintetizada. O problema de redução de estados incompletamente especificados (em inglês ISFSM), é também conhecido como um problema NP-completo [35]. Para poder reduzir estados redundantes nos circuitos sequenciais MVL incompletamente especificados são adaptadas as metodologias utilizadas na lógica binária mostradas a seguir.

### 4.9.1 Estados Compatíveis

Define-se como estado compatível numa máquina de estados finitos, os estados que não têm diferença para nenhum dos valores de entrada, de modo que, para cada entrada, os estados devem ter a mesma saída. Dado dois estados $E_i$ e $E_j$ de um circuito incompletamente especificado, é dito que serão compatíveis se, e somente se, para cada seqüência de entrada aplicada a $E_i$ e $E_j$, será obtida uma seqüência de saída similar. Uma compatibilidade de estados pode ser usada para definir uma relação de compatibilidade. Portanto, um conjunto de estados compatíveis é chamado de compatibilidade de classe.

Para considerar dois estados como compatíveis são utilizados dois critérios.

1. Uma entrada I aplicada sobre os estados $E_i$ e $E_j$ produzirá saídas idénticas quando são especificados (condição nessesária).

2. Os seguintes estados de $E_i$ e $E_j$ serão similares quando ambos são especificados, para cada possível entrada (condição suficiente).

### 4.9.2 Compatibilidade de Classe

A compatibilidade de classes são agrupamentos de estados para gerar novos grupos que sejam compatíveis. Esses estados descrevem a máquina de estados reduzida. Por exemplo, dado os estados compatíveis ou classes (AE), (AD), (ED) e (AED), a máxima compatibilidade deste conjunto é (AED).

### 4.9.3 Incompatibilidade de Estados

Os estados $E_i$ e $E_j$ serão incompatíveis quando não cumprem as duas condições apresentadas na seção anterior. Da mesma maneira que nos estados compatíveis os estados incompatíveis podem formar uma classe de estados incompatíveis.

### 4.9.4 Diagramas de Merger

O diagrama de *Merger* é uma ferramenta gráfica utilizada na redução de estados redundantes numa máquina de estados incompletamente especificados, onde os estados da máquina de estados original são representados convenientemente como pontos igualmente espaçados ao redor do círculo. Esses estados são conectados por um segmento de linha quando é considerado como estados compatíveis, como mostrado na Figura 4.10 (a) e os incompatíveis apresentados na Figura 4.10 (b) [36]. As regras para obter conjuntos máximo no diagrama *Merger* são:

- Fazer o polígono fechado o maior possível para o conjunto maximal.
- Cada estado de um conjunto maximal deveria ser interconectado com outro dentro de um conjunto por um segmento de linha.
- Cada par relacionado deve aparecer dentro do conjunto maximal, tanto para compatíveis e incompatíveis.

### 4.9.5 Critérios de Minimização de Estados

Uma vez obtido o conjunto de compatibilidade de classes por meio do diagrama de *Merger*, o passo seguinte é procurar o conjunto de compatibilidade de classes que reúne as características como: completeza, consistência e minimalidade mencionadas a seguir[2].

1. Completeza. o conjunto de classes compatíveis encontrado deve conter todos os estados da máquina original quando é realizada a união deste.
2. Consistência. O conjunto escolhido de compatibilidade de classe deve ser fechado (closure), isto implica que o seguinte estado de cada classe compatível no conjunto escolhido deve estar contido em alguma classe compatível dentro do conjunto.
3. Minimalidade. A escolha da menor quantidade de compatibilidade de classe que reúne os critérios anteriormente mencionados.

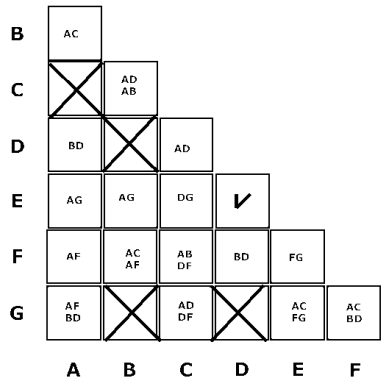
O processo de seleção do conjunto de classes compatíveis é encontrado por tentativa e erro, escolhendo a partir do intervalo entre o número máximo de estados e o número mínimo de estados e que ao mesmo tempo tenham as três características mencionadas anteriormente. Este conjunto de estados é considerado como redução que será utilizado para sintetizar o circuito MVL.

- O número máximo de estados no circuito mínimo é dado pela expressão *U*.

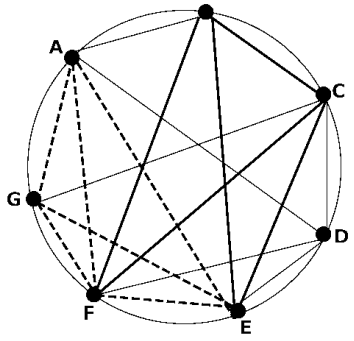
  U = mínimo$\{NCMC, NEOC\}$

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| A | A/1 | B/_ | _/_ | C/_ |
| B | A/_ | B/1 | C/_ | A/1 |
| C | D/3 | A/_ | _/_ | _/_ |
| D | _/_ | D/2 | C/1 | _/_ |
| E | G/_ | _/_ | C/_ | _/_ |
| F | F/_ | B/_ | C/_ | C/_ |
| G | F/_ | D/3 | A/2 | C/1 |

(a)

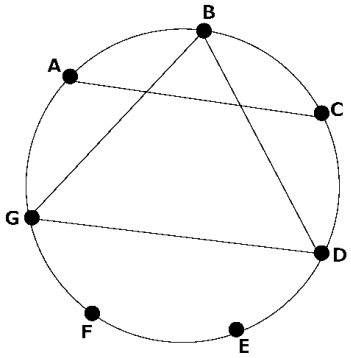



(b)




(AEFG)(BCEF)(CDF)(ADF)(ABE)(CEG)

(c)




(BDG)(AC)(E)(F)

(d)

**Figura 4.10:** (a) Tabela de estados incompletamente especificado. (b) Tabela de implicação. (c) Diagrama *Merger* para classes compatíveis (d) Diagrama *Merger* para classes incompatíveis

Onde

NCMC (Número de conjuntos de estados para máximos compatíveis).

NEOC (Número de estados no circuito original).

- O número mínimo de estados necessários é dado pela expressão *L*.

  L = máximo$\{NEMI_1, NEMI_2, ..., NEMI_i\}$

  Onde

  NEMI (Número de estados no i-ésimo grupo do conjunto máximal incompatíveis do circuito original).

  i (Número de conjuntos maximais incompatíveis).

- O número de estados K na máquina seqüencial é limitado ao intervalo dado pela inequação:

  $L \leq K \leq U$

Para obter um circuito mínimo é necessário avaliar se é possível montar uma tabela reduzida com L estados desde que esta cumpra com as condições de **completeza e consistência**. Este procedimento é realizado por tentativa e erro até achar um valor entre L e U que satisfaça as condições mencionadas.

### 4.9.6 Algoritmo de Redução de Estados

O algoritmo usado para reduzir os circuitos seqüenciais incompletamente especificados é mostrado a seguir.

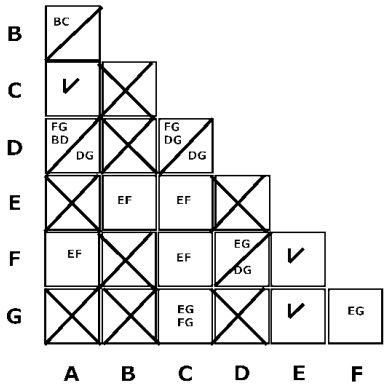
- Passo 1 encontrar o máximo compatíveis utilizando a tabela de implicação e o diagrama *Merger*.
- Passo 2 Encontrar os máximos incompatíveis utilizando a tabela de implicação e o diagrama *Merger*.
- Passo 3 Encontrar os limites sobre o número de estados que serão necessários, U (*upper*) e L (*lower*).
- Passo 4 Encontrar, por tentativa e erro, um conjunto de compatibilidade de classes que satisfaçam completeza, consistência, e minimalidade.
- Passo 5 Produzir a tabela de estados mínimal. Em geral isto ainda pode conter os seguintes estados e saídas inconpletamente especificadas.

Para ajudar no entendimento deste algoritmo será utilizado o exemplo apresentado na Figura 4.11 (a), que contém uma tabela de estados que descreve um circuito seqüencial incompletamente especificado e a Figura 4.11 (b) que mostra a tabela de implicação construída a partir da tabela de estados. A partir desta tabela de implicação encontram-se as classes compatíveis apresentados na Figura 4.11 (c) e as classes incompatíveis mostradas na Figura 4.11 (d). Esta forma de encontrar classes compatíveis e incompatíveis pode ser tediosa, e é por isso que recorre-se ao diagrama *Merger* como uma ferramenta gráfica que ajuda na busca de classes.

Como primeiro e segundo passo do algoritmo para minimização de estados de um circuito seqüencial incompletamente especificado devem-se encontrar as classes compatíveis e incompatíveis. Para isto, é necessário gerar os diagramas *Merger* para classes compatíveis e incompatíveis a partir da tabela de implicação apresentada na Figura 4.11 (b). Uma vez obtidos os diagramas *Merger*, por meio de uma análise visual é que são encontradas as classes compatíveis e incompatíveis apresentadas na Figura 4.12 (a) e (b), respectivamente.

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| A | E/0 | F/0 | B/_ | G/_ |
| B | E/0 | F/_ | C/0 | G/3 |
| C | E/_ | F/_ | B/_ | G/0 |
| D | _/_ | G/0 | D/0 | D/0 |
| E | E/_ | E/1 | _/_ | G/_ |
| F | _/_ | E/_ | _/0 | G/0 |
| G | G/0 | G/1 | B/0 | G/0 |

(a)




(b)

| | |
|---|---|
| F | (FG) |
| E | (EG)(EF)(FG) |
| E | (EFG) |
| C | (CG)(CF)(CE)(EFG) |
| B | (BE)(CG)(CF)(CE)(EFG) |
| A | (AF)(AC)(BE)(CG)(CF)(CE)(EFG) |
| A | (ACF)(CEG)(EFG)(BE) |

(c)

| | |
|---|---|
| F | - |
| E | - |
| D | (DG)(DF)(DE) |
| C | (CD)(DG)(DF)(DE) |
| B | (BG)(BF)(BD)(BC)((CD)(DG)(DF)(DE) |
| B | (BCD)(BDG)(BDF)(DE) |
| A | (AG)(AE)(AD)(AB)(BCD)(BDG)(BDF)(DE) |
| A | (ABCDG)(ADE)(BDF)(BDG) |

(d)

**Figura 4.11:** (a) Tabela de estados. (b) Tabela de implicação. (c) Máximos compatíveis (d) Máximos incompatíveis

Uma vez encontradas as classes compatíveis, o seguinte passo é construir a tabela de estados reduzida. Para isto, primeiro é construída a tabela de fechamento (*closure*) mostrada na Figura 4.13 (a) e, em seguida, a tabela de estados reduzida apresentada na Figura 4.13 (b).

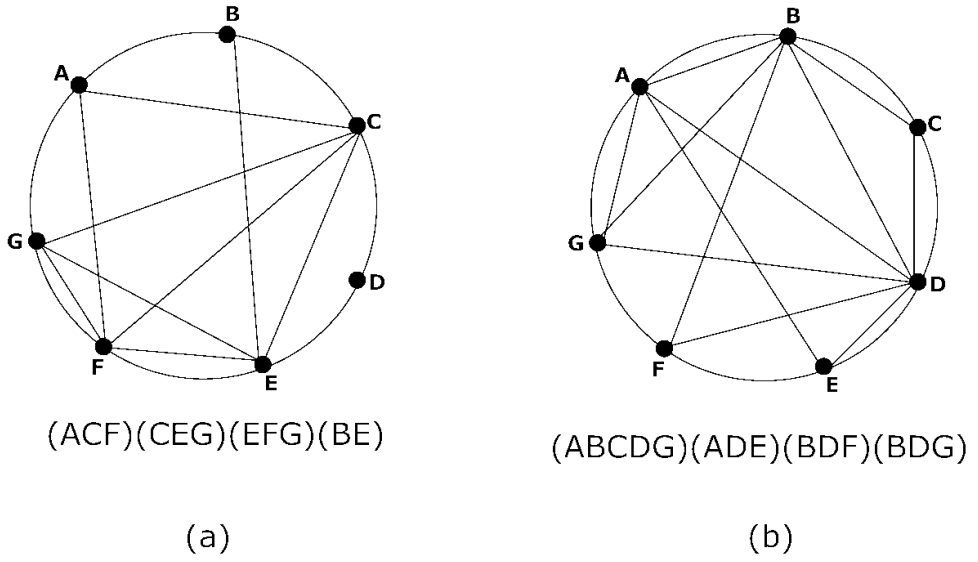



**Figura 4.12:** (a) Diagrama merger para máximos compatíveis. (b) Diagrama merger para máximos incompatíveis.

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| A' → (ACF) | E | EF | B | G |
| B' → (CEG) | EG | EFG | B | G |
| C' → (EFG) | EG | EG | B | G |
| D' → (BE) | E | EF | C | G |
| E' → (D) | _ | G | D | D |

(a)

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| A' | B',C',D/0 | C'/0 | D'/0 | B',C'/0 |
| B' | B',C'/0 | C'/1 | D'/0 | B',C'/0 |
| C' | B',C'/0 | B',C',D'/1 | D'/0 | B',C'/0 |
| D' | B',C',D'/0 | C'/1 | A',B'/0 | B',C'/3 |
| E' | _/_ | B',C'/0 | E'/0 | E'/0 |

(b)

**Figura 4.13:** (a) Tabela de fechamento (*closure*). (b) Tabela de estados reduzida.

O terceiro passo é encontrar os limites superior e inferior do número de estados que são necessários no circuito minimizado. Isto é obtido da seguinte maneira:

- O número máximo de estados no circuito mínimo é dado pela expressão *U*.

  U = mínimo{5,7}

  U = 5

- O número mínimo de estados necessários é dado pela expressão *L*. Os classes incompatíveis são (ABCDG)(ADE)(BDF)(BDG).

  L = máximo$\{5, 3, 3, 3\}$

  L = 5

- O número de estados K na máquina seqüencial é limitado pelo intervalo dado na inequação:

  $5 \leq K \leq 5$

  Isso significa que a mínima redução de estados pode ser de até 5 estados o que implica que a redução da tabela será da forma apresentada na Figura 4.14

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| A' | B',C',D/0 | C'/0 | D'/0 | B',C'/0 |
| B' | B',C'/0 | C'/1 | D'/0 | B',C'/0 |
| C' | B',C'/0 | B',C',D'/1 | D'/0 | B',C'/0 |
| D' | B',C',D'/0 | C'/1 | A',B'/0 | B',C'/3 |
| E' | _/_ | B',C'/0 | E'/0 | E'/0 |

**Figura 4.14:** (a) Tabela de estados reduzida.

CAPÍTULO

5

# Síntese do Circuito Seqüencial MVL

A síntese do circuito seqüencial MVL tem como ponto de partida a descrição verbal do projeto que logo é abstraído para uma tabela de estados ou diagrama de estados (máquina de estados). Em alguns casos, o projetista coloca mais estados do que os necessários no projeto. É neste ponto que a tabela necessitaria ser reduzida. No Capítulo 4 foram apresentados alguns métodos que permitem a redução de estados no projeto de um circuito seqüencial síncrono e assíncrono.

A seguir é apresentada a metodologia de síntese dos circuitos seqüenciais MVL (com memória). Esta metodologia pode ser dividida em etapas que fazem parte do projeto para a síntese de circuitos seqüenciais MVL como:

1. Descrição verbal do funcionamento;
2. Diagrama e tabela de estados;
3. Tabela minimal de estados;
4. Associação de estados e tabela de transição;
5. Equações das entradas dos *Flip-flops*;
6. Síntese e simulação do circuito.

## 5.1 Circuito MVL Detector de Início de Mensagem

O projeto de circuito seqüencial MVL tem como ponto de partida a descrição funcional do circuito, para depois ser abstraído para uma tabela ou diagrama de estados.

### 5.1.1 Descrição Verbal do Funcionamento

Projetar um detector de início de mensagem onde seja considerada somente uma linha de transmissão denotada por *Entrada* e um *Clock* que ajuda no sincronismo. O início da mensagem é considerada com saída igual a 2, quando na entrada é apresentado o valor 2 por 3 vezes consecutivas. Supondo que existe algum mecanismo que coloca o sistema detector de início da mensagem no estado inicial.

### 5.1.2 Diagrama e Tabela de Estados

Um diagrama de estados é uma abstração obtida a partir da descrição verbal do sistema detector de início da mensagem, apresentada na Figura 5.1. Outra maneira de representar esta abstração é por meio da tabela de estados, mostrada na Figura 5.2 que descreve o comportamento do circuito e que posteriormente será utilizada na redução de estados.

### 5.1.3 Tabela Minimal de Estados

Nesta etapa, o processo de redução da tabela de estados da Figura 5.2 é realizada pelo método da *Tabela de Implicação* ou o *Diagrama Merger* descrito no Capítulo 4. Estes métodos permitem realizar a redução de circuitos seqüenciais completamente especificados. As Figuras 5.3 (a) e (b) apresentam a redução pelo método da *Tabela de Implicação* e a Figura 5.4 (a) mostra a redução pelo *Diagrama Merger*. Obtendo-se como resultado as classes (AEFG)(B)(C) e (D), esta redução é apresentada na tabela de estados da Figura 5.4 (c) e o diagrama de estados reduzido é mostrada na Figura 5.4 (d).

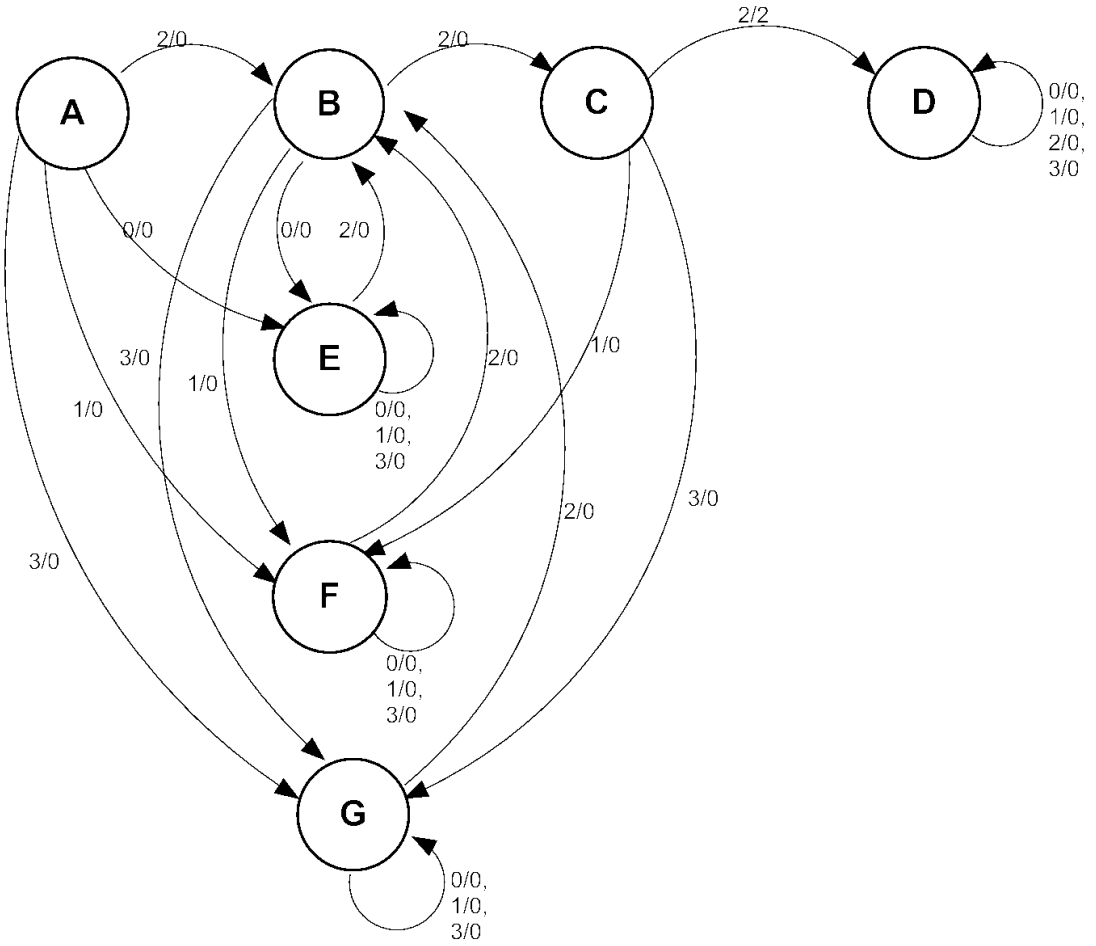



**Figura 5.1:** Diagrama de estados.

| Estado Presente \ Entrada | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| A | E/0 | F/0 | B/0 | G/0 |
| B | E/0 | F/0 | C/0 | G/0 |
| C | E/0 | F/0 | D/2 | G/0 |
| D | D/0 | D/0 | D/0 | D/0 |
| E | E/0 | E/0 | B/0 | E/0 |
| F | F/0 | F/0 | B/0 | F/0 |
| G | G/0 | G/0 | B/0 | G/0 |

**Figura 5.2:** Tabela de estados.

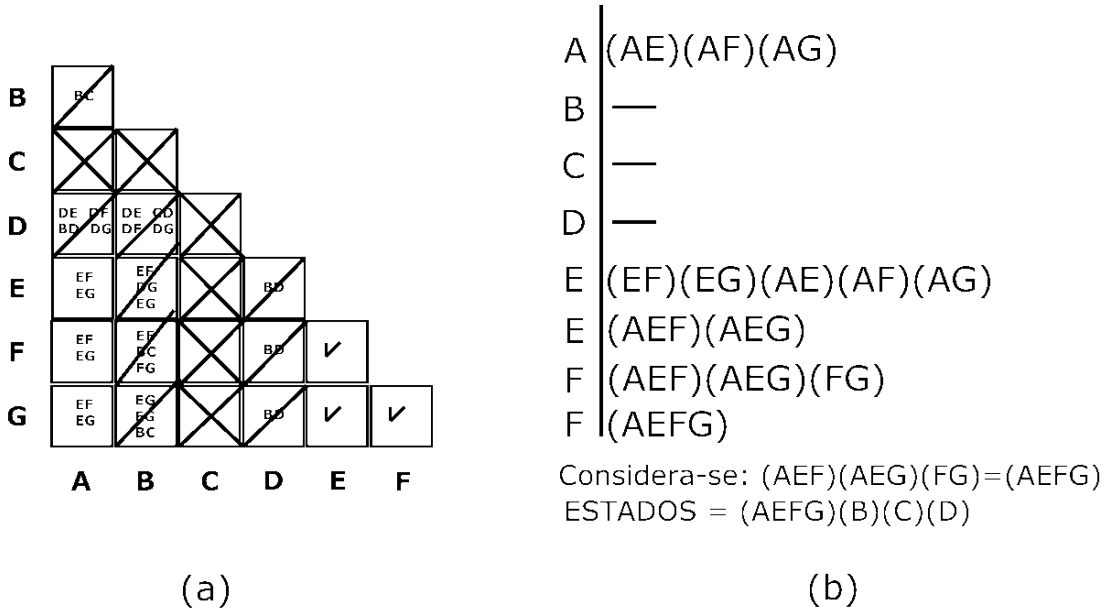
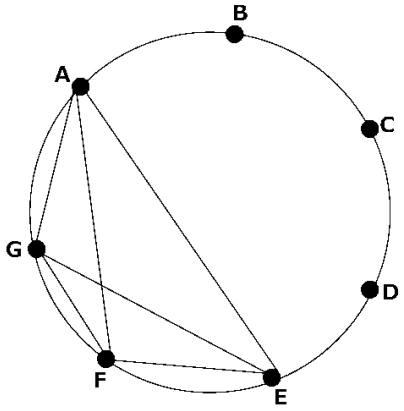



**Figura 5.3:** (a) Tabela de implicação. (b) Partição de equivalência.

### 5.1.4 Associação de Estados e Tabela de Transição

Nesta etapa do projeto é atribuído um código MVL para cada estado e a escolha do estado inicial. Pode ser observado na Figura 5.5 (a). A quantidade de dígitos necessários para representar um estado indica a quantidade de elementos de memória que serão necessários no projeto de circuito MVL. Logo, é necessario reduzir o número de dígitos que mudam entre cada transição de estados, ou seja, utilizando o critério do código Gray (0-1-2-3). A tabela de transição mostra qual será o próximo estado em função do estado atual e entradas atuais como são apresentadas na Figura 5.5 (b).

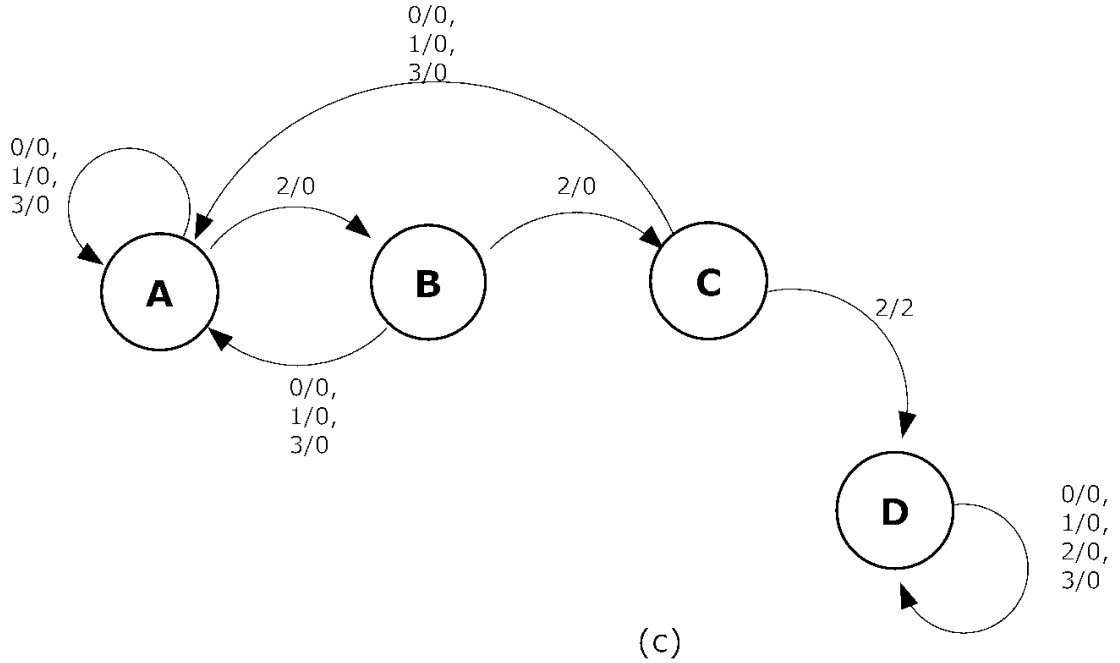



(AEFG)(B)(C)(D)

(a)

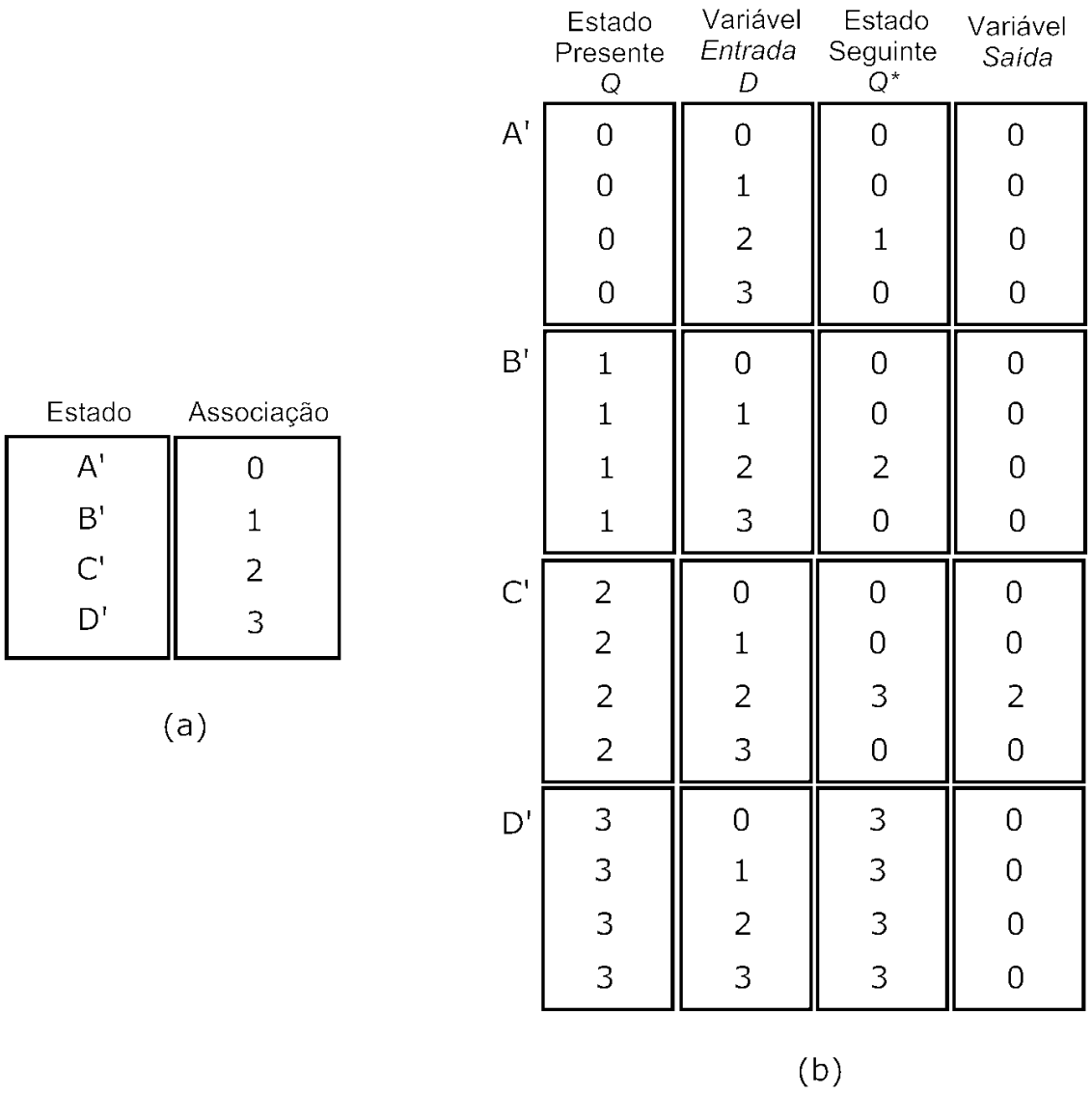
| Estado Presente \ Entrada | | 0 | 1 | 2 | 3 |
|---|---|---|---|---|---|
| (AEFG) | A' | A'/0 | A'/0 | B'/0 | A'/0 |
| (B) | B' | A'/0 | A'/0 | C'/0 | A'/0 |
| (C) | C' | A'/0 | A'/0 | D'/2 | A'/0 |
| (D) | D' | D'/0 | D'/0 | D'/0 | D'/0 |

(b)




(c)

**Figura 5.4:** (a) Diagrama *Merger* para circuitos completamente especificados. (b) Tabela de estado futuro reduzida. (c) Diagrama de estados reduzida.



| Estado | Associação |
|---|---|
| A' | 0 |
| B' | 1 |
| C' | 2 |
| D' | 3 |

(a)

| | Estado Presente *Q* | Variável *Entrada D* | Estado Seguinte *Q** | Variável *Saída* |
|---|---|---|---|---|
| A' | 0 | 0 | 0 | 0 |
| | 0 | 1 | 0 | 0 |
| | 0 | 2 | 1 | 0 |
| | 0 | 3 | 0 | 0 |
| B' | 1 | 0 | 0 | 0 |
| | 1 | 1 | 0 | 0 |
| | 1 | 2 | 2 | 0 |
| | 1 | 3 | 0 | 0 |
| C' | 2 | 0 | 0 | 0 |
| | 2 | 1 | 0 | 0 |
| | 2 | 2 | 3 | 2 |
| | 2 | 3 | 0 | 0 |
| D' | 3 | 0 | 3 | 0 |
| | 3 | 1 | 3 | 0 |
| | 3 | 2 | 3 | 0 |
| | 3 | 3 | 3 | 0 |

(b)

**Figura 5.5:** (a) Associação de estados. (b) Tabela de transição.

### 5.1.5 Equações das Entradas dos *Flip-flops*

O mapa de Karnaugh Estendido é uma ferramenta gráfica para minimização de funções lógicas MVL descrito em [30]. Esta técnica foi baseada na proposta dos *Mapas de Karnaugh* introduzido por Maurice Karnaugh em 1953 que tem a particularidade de mudar um só dígito entre as células facilitando a identificação dos termos que podem ser combinados com o objetivo de simplificar a função lógica. Os resultados obtidos a partir dos mapas de *Karnaugh* são apresentados nas Figuras 5.6 (a) e (b) que são representados nas Equações 5.1 e 5.2, respectivamente.

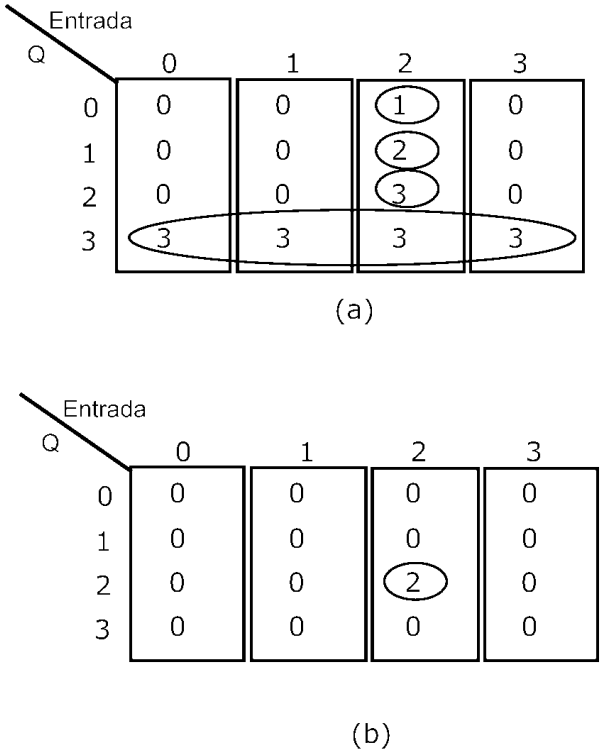



**Figura 5.6:** *Mapas de Kanaugh*. (a) Estado futuro do circuito. (b) Saída do circuito.

$$D = Q^1 \star^1 entrada^3 + Q^1 \star^2 entrada + Q^1 \star^3 entrada^1 + Q^0 \star^3 3 \tag{5.1}$$

$$saida = Q \star^2 entrada^0 \tag{5.2}$$

### 5.1.6 Síntese e Simulação do Circuito

A síntese do circuito apresentado na Figura 5.7 é realizada usando o *Flip-flop* MVL tipo D apresentadas no Capítulo 3. As equações que caracterizam o circuito são apresentadas nas Equações 5.1 e 5.2, onde a variável *D* do *Flip-flop D* está em função das variáveis *entrada* e *Q*. A variável *saida* está em função da variável *entrada* e o estado atual *Q* do circuito.

A simulação foi realizada considerando o tempo de *delay* em nanosegundos (ns), com a descrição das portas em VHDL, adaptado para o caso MVL. No projeto do circuito foram utilizadas 8 portas sucessor (*Suc*) com delay 5ns, 1 porta *AND* estendido (1) com *delay* 5ns, 2 portas *AND* estendido (2) com *delay* 5ns, 2 portas *AND* estendido (3) com *delay* 5ns, 1 porta *MAX* e um *Flip-flop* MVL tipo D. O resultado desta simulação pode ser observardo na Figura 5.8. Nota-se que a saída *Q* do *Flip-flop* D copia a entrada quando o *Clk* = 1,2 e 3.

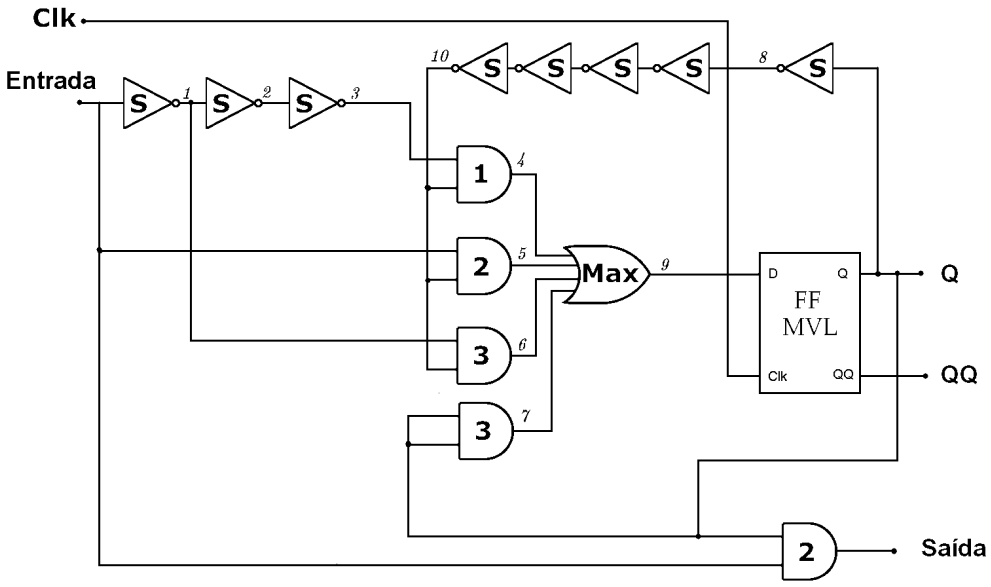



**Figura 5.7:** Diagrama lógico do circuito detector de inicio de menssagem

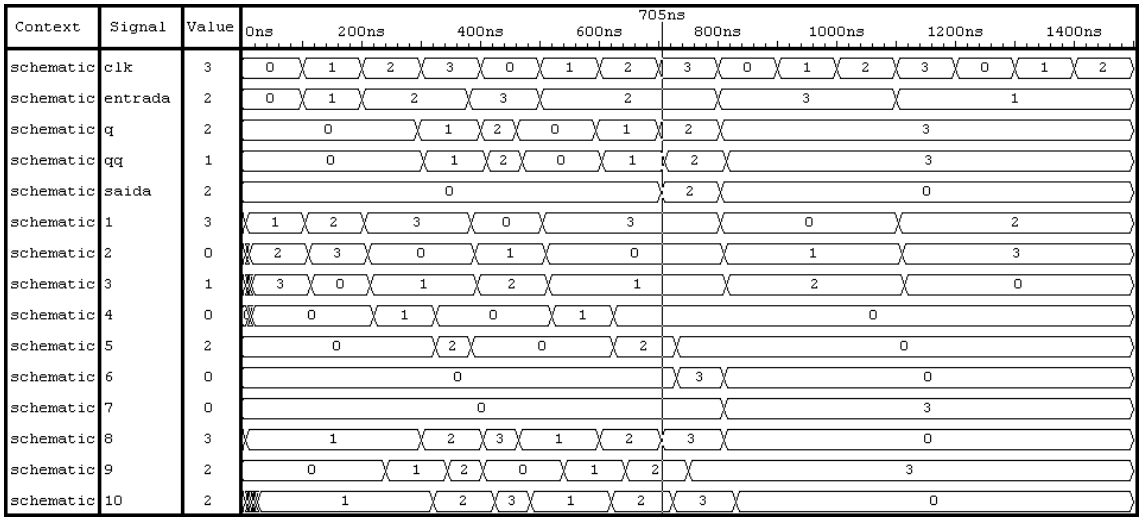



**Figura 5.8:** Simulação do circuito detector de inicio de menssagem

CAPÍTULO

6

# Considerações Finais

## 6.1 Conclusão

O objetivo deste trabalho foi desenvolver a metodologia de síntese para circuitos seqüenciais *MVL* com memória. Segundo alguns trabalhos publicados, a Álgebra de múltiplos valores lógicos (*MVL*) tem potencial para poder armazenar e transmitir maior quantidade de informação, e portanto, reduzir o espaço ocupado no *chip*.

No Capítulo 2 deste trabalho descrevem-se a Álgebra *MVL* com suas respectivas portas lógicas, que permitem sintetizar funções lógicas. No Capítulo 3 foram sintetisados os elementos de memória com uma funcionalidade similar aos existentes na lógica binária e define-se o funcionamento do clock *MVL*.

Em seguida, foram desenvolvidos os métodos para reduzir os estados completamente e incompletamente especificados a partir dos existentes para circuitos seqüenciais na lógica binária. A metodologia de síntese de circuitos seqüenciais baseados na Álgebra *MVL*, proposta neste trabalho, realiza a síntese de circuitos com memória.

Conclui-se que a metodologia desenvolvida para sintetizar circuitos seqüenciais completamente e incompletamente especificados baseado na Álgebra *MVL* é funcional. Os resultados das simulações mostram que os circuitos *MVL* resolvem em parte o problema apresentado pelo sistema binário em relação à quantidade de informação transmitida por cada linha de conexão.

## 6.2 Trabalhos Futuros

A seguir são mencionados alguns trabalhos que poderiam ser realizados no futuro.

- Neste trabalho foram apresentados dois tipos de definições sobre o comportamento do *Clock* para circuitos seqüenciais. Portanto, é necessário um estudo mais aprofundado para definir o tipo de funcionamento do *Clock*.
- A implementação física dos elementos de memória e as portas lógicas MVL ajudarão na implementação de circuitos MVL, que ao mesmo tempo permitirão realizar uma comparação quantitativa e qualitativa entre os circuitos binários e circuitos MVL.
- Assim como na lógica binária existe um elemento de memória genérico conhecido como *Flip-flop JK*. Seria muito útil o projeto do elemento de memória MVL com estas características (*Flip-flop JK* MVL). Já que isto permitiria realizar diferentes tipos de projetos de circuitos integrados MVL, devido a suas funcionalidades como elemento de memoria genérico.
- A implementação de um software com os algoritmos para redução de estados em circuitos seqüenciais completamente e incompletamente especificados ajudariam na otimização de este tipo de circuitos mais complexos.
- O presente trabalho apresenta a metodologia de síntese dos circuitos seqüenciais MVL em base quaternária. Esta metodologia pode ser estendida para outras bases MVL.

# Referências Bibliográficas


[1] E. DUBROVA. Multiple-valued logic in VLSI: Challenges and opportunities. *Multiple-Valued Logic in VLSI: Challenges and Opportunities. In Proceedings of NORCHIP'99, Oslo, Norway*, 1999.

[2] NELSON, V. P.; NAGLE, H. T.; CARROLL, B. D. et al. *Digital Logic Circuit Analiysis and Design*. Prentice Hall Englewood Cliffs, 1995.

[3] BUTLER, J. T. Multiple-Valued Logic. *Proc. IEEE, Vol.14, P. 11-14*, 1995.

[4] DANIELSSON, P. E. Boolean Memories. *Proc. IEEE Trans. Comp.*, 1966.

[5] LUKASIEWICZ, J. Elementy Logiki Matematycznej. *Translated as 'Elements of Mathematical logic', New York: Macmillan 1963*, 1929.

[6] EPSTEIN G. The Lattice Theory of Post Algebras. *Transactions of the American Mathematics Society , Vol.95, P. 300-317*, 1960.

[7] HORN A. G. E. P-algebras, an Abstraction from Post Algebras. *Algebra universalis, Vol.4, P. 195-206*, 1974.

[8] KOZEN, D. On Kleene Algebras and Closed Semirings. In B. Rovan, editor, *Mathematical Foundations of Computer Science 1990*, volume 452, pages 26–47, Banská Bystrica, 1990. Springer-Verlag.

[9] POST, E. L. Introduction to a General Theory of Elementary Propositions. *Proc. American Journal Mathematics,Vol. 43, P. 163-185*, 1921.

[10] ETIEMBRE D.; ISRAEL, M. Comparison of Binary and Multivalued ICs According to VLSI Criteria. *IEEE Journal or Magazine, Volume 21, Issue 4, April 1988 P. 28 - 42*, 1989.

[11] HANYU, T.; KAMEYAMA, M. A 200 MHz Pipelined Multiplier Using 1.5 V-supply Multiple-Valued MOS Current-Mode Circuits With Dual-Rail Source-Coupled Logic. *Solid-State Circuits, IEEE Journal of, Vol. 30 P. 1239-1245*, Nov 1995.

[12] GONZALEZ, A.F.; MAZUMDER, P. Multiple-Valued Signed Digit Adder Using Negative Differential Resistance Devices. *Computers, IEEE Transactions on, Vol. 47 P. 947-959*, Sep 1998.

[13] HANYU, T.; MOCHIZUKI, A.; KAMEYAMA, M. Design and Evaluation of a Multiple-Valued Arithmetic Integrated Circuit Based on Differential Logic. *Circuits, Devices and Systems, IEE Proceedings, Vol. 143, P. 331-336*, 1996.

[14] SHIMABUKURO, K.; ZUKERAN, C. Reconfigurable Current-mode Multiple-Valued Residue Arithmetic Circuits. *Multiple-Valued Logic, 1998. Proceedings. 1998 28th IEEE International Symposium on, P. 282-28*, May 1989.

[15] RADANOVIC, B.; SYRZYCKI, M. Current-Mode CMOS Adders Using Multiple-Valued Logic. *Electrical and Computer Engineering, 1996. Canadian Conference on, Vol. 1, P. 190-193*, May 1996.

[16] JING SHEN; TANNO, K.; ISHIZUKA, O. et al. Application of Neuron-MOS to Current-Mode Multi-valued Logic Circuits. *Multiple-Valued Logic, 1998. Proceedings. 1998 28th IEEE International Symposium on*, pages 128–133, May 1998.

[17] ABD-EL-BARR, M.; HASAN, M. N. New MVL-PLA Structures Based on Current-mode CMOS Technology. *Multiple-Valued Logic, Proceedings., 26th International Symposium on, P. 98-103*, May 1996.

[18] CHAN, H. L. E.; BHATTACHARYA, M.; MAZUMDER, P. Mask-Programmable Multiple-Valued Logic Gate Using Resonant Tunnelling Diodes. *Circuits, Devices and Systems, IEE Proceedings Vol. 143, P. 289-294*, Oct 1996.

[19] CHAN, H. L.; MOHAN, S.; MAZUMDER, P. et al. Compact Multiple-Valued Multiplexers Using Negative Differential Resistance Devices. *Solid-State Circuits, IEEE Journal of, Vol. 31, P 1151-1156*, Aug 1996.

[20] HAO TANG; LIN, H. C. Multi-Valued Decoder Based on Resonant Tunneling Diodes in Current Tapping Mode. *Multiple-Valued Logic, 1996. Proceedings., 26th International Symposium on, P. 230-234*, May 1996.

[21] HANYU, T.; TERANISHI, K.; KAMEYAMA, M. Multiple-Valued Logic-in-Memory VLSI Based on a Floating-Gate-MOS Pass-Transistor Network. *Solid-State Circuits Conference, 1998. Digest of Technical Papers. 1998 IEEE International, P. 194-195, 437*, Feb 1998.

[22] OHKAWA, M.; SUGAWARA, H.; SUDO, N.et al. A 98 mm2 Die Size 3.3-V 64-Mb Flash Memory With FN-NOR Type Four-level Cell. *Solid-State Circuits, IEEE Journal of, Vol. 31, P. 1584-1589*, Nov 1996.

[23] OKUDA, T.; MUROTANI, T. A Four-Level Storage 4-Gb DRAM. *Solid-State Circuits, IEEE Journal of, Vol. 32, P. 1743-1747*, Nov 1997.

[24] OKUDA, T. Advanced Circuit Technology to Realize Post Giga-Bit DRAM. *Multiple-Valued Logic, 1998. Proceedings. 1998 28th IEEE International Symposium on, P. 2-5*, May 1998.

[25] BERTONE, O. H. *Proposta de um Registrador Cíclico Para Lógica Multi-valóres e Aplicação em um Multiplicador Quaternario*. Dissertação (Mestrado em Engenharia Elétrica), Universidade Estadual de Campinas, 2005.

[26] YACOUB, M. N. R. D. *Proposta de Implementação de uma Lógica Ternaria em Tecnologia CMOS*. Tese (Doutorado em Engenharia Elétrica), Universidade Estadual de Campinas, 2000.

[27] NASCIMENTO, L. P. *Uma Ferramenta Automatizada para Análise e Projeto de Circuitos Digitais Multi-Valores*. Dissertação (Mestrado em Engenharia Elétrica), Universidade Estadual de Campinas, 2001.

[28] DHANDE, A. P.; INGOLE, V. T. Design of 3-Value R-S & D Flip-Flops Based on Simple Ternary Gates. *Proc. International Journal of Software Engineering and Knowledge Engineering, Vol. 15*, 2005.

[29] SHANNON, C. E. A simbolic Analysis of Relay and Switching Circuits. Dissertação (Master in Electrical Engineering), Massachusetts Institute of Technology, 1940.

[30] LUQUE, P. H.; APAZA, T. M.; TURQUETI, M. et al. Síntesis de Circuitos en Lógica Multi-nivel. *Proc. V COISIS y 7º Workshop SIIS*, 2006.

[31] BALABANIAN, N.; CARLSON, B. *Digital Logic Design Principles*. John Wiley Sons, 2001.

[32] SHIH-HSU, H.; YOW-TYNG, N. Synthesis of Nonzero Clock Circuits. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 2006.

[33] APAZA, T. M.; LUQUE, P. H.; TURQUETI, M. et al. Síntesis de Circuitos con Memoria en Lógica Multi-nivel. *Proc. V COISIS y 7º Workshop SIIS*, 2006.

[34] HOPCROFT, J. E. An nlogn Algorithm for Minimizing the States in a Finite Automaton. *The Theory of Machines and Computations*, 1971.

[35] PFLEEGER, C.P. State Reduction in Incompletely Specified Finite-State Machines. *Computers, IEEE Transactions*, C-22(12):1099–1102, Dec. 1973.

[36] KOHAVI, Zvi. Reduction of the Number of States in Incompletely Specified Sequential Machines. *Electronics Letters*, 1(7):209–210, September 1965.