Vending Machine

Kaique da Cunha Lima (Nº 10368980) -> Aula Teórica
Matheus Carvalho Raimundo (Nº 10369014) -> Aula Teórica e Prática

Orientador: Prof. Mauricio Acconcia Dias

USP - São Carlos

Setembro/2017

Índice Analítico

1. Considerações iniciais

No cotidiano das pessoas, diversos dispositivos e máquinas complexos são desenvolvidos em hardware. O objetivo deste trabalho é desenvolver um destes: uma *Vending Machine*.

Uma Vending Machine é uma máquina de vendas rápida de diversos produtos. No geral, são colocadas em funcionamento em locais de grande movimentação de pessoas, com produtos alimentícios industrializados à venda. Será feito neste trabalho uma Vending Machine que aceita como entrada uma cédula de dois reais ou uma moeda de um real. Como saída, ela devolve um produto (de acordo com a soma da quantia depositada), ou devolve o dinheiro, caso a soma não resulte em nenhum produto. A máquina implantada neste trabalho não retorna troco. Além disso, há um tempo limite para inserção do dinheiro (*Timeout*), que é nossa "variável" controladora.

Figura 1. Exemplo de Vending Machine.

2. Contexto-exemplo da Vending Machine

Atenção: a história contada nesta seção é fictícia e tem como único intuito contextualizar um cenário exemplo em que a Vending Machine pode ser utilizada.

Em 9 de agosto de 2017, no Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo, alguns alunos do curso de Ciências de Computação perceberam que deslocavam grandes distâncias para conseguir um lanche quando estavam nos laboratórios fazendo os trabalhos de programação, pois a lanchonete do instituto nunca estava perto o suficiente para eles. Visando a

comodidade, em conjunto com a grande quantidade de sedentarismo que o curso proporciona, os alunos decidiram colocar uma máquina de vendas (Vending Machine) dentro dos próprios laboratórios.

O que os alunos não esperavam é que após obter autorização da comissão de graduação, eles mesmos seriam os responsáveis pela implementação dos circuitos que executariam dentro da máquina. De acordo com os alunos, seria construído um circuito. Visando menor quantidade de problemas, maior velocidade e maior precisão, optaram por implantar o circuito em hardware, de forma lógica e utilizando Máquinas de Estado Finito (FSM).

3. Implementação

A Vending Machine que será implantada neste trabalho, por definição, aceita duas quantias de dinheiro (isoladas; nunca simultaneamente mais de uma entrada de dinheiro):

- Moeda de 1 real (+R$ 1,00);
- Nota de 2 reais (+R$ 2,00).

Além disso, ela retorna dois produtos:

- Refrigerante Pepsi, por R$ 2,00;
- Salgado Coxinha, por R$ 5,00.

E há a variável de controle, Timeout; quando Timeout estiver ativo (1), a máquina irá calcular toda a quantia inserida e devolver uma resposta (seja um produto, ou retornar o dinheiro caso a quantia não seja o bastante); quando Timeout não estiver ativo (0), a máquina irá ficar ociosa, aguardando por entradas de dinheiro. Por definição, Timeout será uma entrada do circuito, que fica não-ativa (0) por 10 segundos, ativa por 1 período de clock, e retorna ao estado inicial (0): ou seja, “o Timeout é de 10 segundos”.

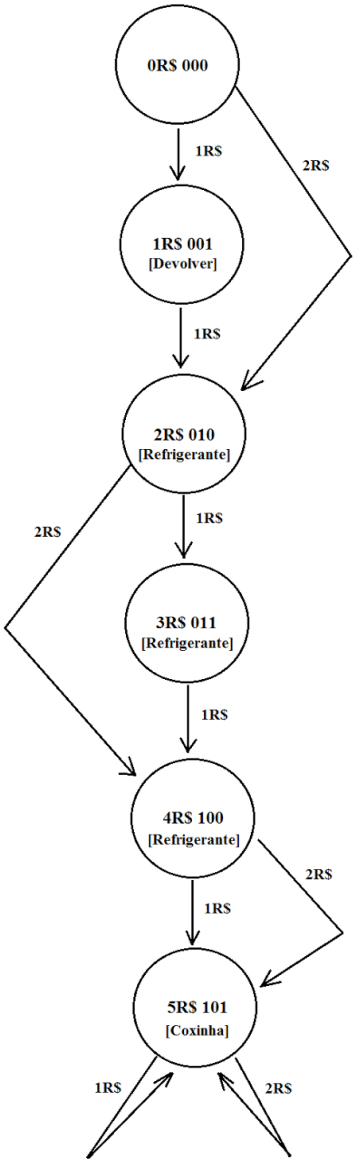
A Vending Machine não devolverá troco, mas devolverá todo o dinheiro acumulado caso tal quantia não seja o bastante para comprar um produto. Por exemplo, se o produto mais “barato” da máquina custa 5 reais, e o usuário inseriu apenas 4 reais quando Timeout ativou, a máquina irá devolver ao usuário a quantia de 4 reais que foi inserida, porque não foi o bastantes para adquirir nada; ao mesmo tempo, se o produto mais “caro” da máquina custa 10 reais, e o usuário inseriu 13 quando Timeout ativou, a máquina irá devolver ao usuário apenas o produto adquirido, sendo que o usuário perde 3 reais, porque não há troco.

4. Máquina de Estados

A Máquina de Estados é uma nova forma de pensar logicamente. Um nível acima da lógica de Flip-Flops, construir Máquina de Estados simplifica o circuito e torna mais fácil sua implementação, pois a única preocupação de quem está implementando o circuito é pensar nos próprios estados do circuito e suas saídas.

Neste trabalho, será implantada a Máquina de Estados de Moore, que possui as saídas representadas no próprio estado (diferentemente da Máquina de Estados de Mealy, que possui as saídas na transição entre os estados).

De acordo com a implementação descrita anteriormente, chega-se à conclusão que a Máquina de Estados de Moore que representa a Vending Machine deste trabalho é:



5. Tabela verdade e mapa Karnaugh

A tabela verdade que representa a Vending Machine deste trabalho é:

Present State	Inputs	Next State	Refrigerante	Coxinha	Devolver
Q2/Q1/Q0	2R$/1R$	D2/D1/D0			
000 (0R$)	00	000	0	0	0
	01	001	0	0	0
	10	010	0	0	0
	11	X	X	X	X
001 (1R$)	00	001	0	0	1
	01	010	0	0	1
	10	011	0	0	1
	11	X	X	X	X
010 (2R$)	00	010	1	0	0
	01	011	1	0	0
	10	100	1	0	0
	11	X	X	X	X
011 (3R$)	00	011	1	0	0
	01	100	1	0	0
	10	101	1	0	0
	11	X	X	X	X
100 (4R$)	00	100	1	0	0
	01	101	1	0	0
	10	101	1	0	0
	11	X	X	X	X
101 (5R$)	00	101	0	1	0
	01	101	0	1	0
	10	101	0	1	0
	11	X	X	X	X

Sendo 00 a entrada recebida quando nada é colocado na máquina, 01 quando um real é colocado e 10 quando dois reais são colocados. Para os itens que devem sair da máquina (refrigerante e coxinha), 0 indica que caso seja dado um Timeout naquele estado não sairá nada, e 1 indica que caso seja dado esse Timeout, o item sairá. No último estado (5R$) não importa o que seja colocado, pois a máquina permanecerá sempre neste estado até o Timeout.

Perceba que a variável Timeout não foi representada na tabela como uma entrada. Isso porque na hora de implementar, ela será colocada ao fim, na própria saída (como algo a ser tratado à parte na implementação)

Já os mapas de Karnaugh que representam a tabela verdade acima podem ser dados da seguinte forma:

PARA D2				
Q2 Q1 Q0/R2 R1	*00*	*01*	*11*	*10*
000	0	0	X	0
001	0	0	X	0
011	0	1	X	1
010	0	0	X	1
110	X	X	X	X
111	X	X	X	X
101	1	1	X	1
100	1	1	X	1

PARA D1				
Q2 Q1 Q0/R2 R1	*00*	*01*	*11*	*10*
000	0	0	X	1
001	0	1	X	1
011	1	0	X	0
010	1	1	X	0
110	X	X	X	X
111	X	X	X	X
101	0	0	X	0
100	0	0	X	0

PARA D0				
Q2 Q1 Q0/R2 R1	*00*	*01*	*11*	*10*
000	0	1	X	0
001	1	0	X	1
011	1	0	X	1
010	0	1	X	0
110	X	X	X	X
111	X	X	X	X
101	1	1	X	1
100	0	1	X	1

REFRIGERANTE				
Q2 Q1 Q0/R2 R1	*00*	*01*	*11*	*10*
000	0	0	X	0
001	0	0	X	0
011	1	1	X	1
010	1	1	X	1
110	X	X	X	X
111	X	X	X	X
101	0	0	X	0
100	1	1	X	1

COXINHA				
Q2 Q1 Q0/R2 R1	*00*	*01*	*11*	*10*
000	0	0	X	0
001	0	0	X	0
011	0	0	X	0
010	0	0	X	0
110	X	X	X	X
111	X	X	X	X
101	1	1	X	1
100	0	0	X	0

6. Código VHDL detalhado

Construiu-se um código em linguagem VHDL completo para o circuito da Vending Machine deste trabalho de forma detalhada. Isso significa que a implementação de todas as partes do trabalho está descrita manualmente neste código, desde os Flip-Flops usados a até mesmo o retorno com o tratamento de Timeout.

Tal código está descrito abaixo:

```
-- VendingMachineDetailed.vhd
library IEEE;
use IEEE.std_logic_1164.all;


entity VendingMachineDetailed is -- Top-Level: Vending Machine
  port (
        InUmReal, InDoisReais, Timeout, Clock:  in bit;
        OutDinheiro, OutCoxinha, OutPepsi:       out bit
  );
end entity;

entity FlipFlopD is
  port (
        D, Set, Rst, Clk: in bit;
        Q:                out bit
  );
end entity;


architecture Main of VendingMachineDetailed is
component FlipFlopD is
  port (
        D, Set, Rst, Clk: in bit;
        Q:                out bit
  );
end component;
signal Q0:bit;
signal Q1:bit;
signal Q2:bit;
signal D0:bit;
```

```
signal D1:bit;
signal D2:bit;
signal R2,R1:bit;
begin
  R2<=InDoisReais;
  R1<=InUmReal;
  OutCoxinha <= (Q2 and not(Q1) and Q0) and Timeout;
  OutPepsi <= (Q1 or (Q2 and not(Q1) and not(Q0))) and Timeout;
  OutDinheiro <= (not(Q2) and not(Q1) and Q0) and Timeout;
   D0 <= (not(Q2) and not(Q1) and not(Q0) and not(R2) and R1) or (not(Q2) and Q1 and not(Q0) and not(R2) and R1) or
(not(Q2) and Q0 and not(R2) and not(R1)) or (not(Q2) and Q0 and R2 and not(R1)) or (Q2 and not(Q1) and R2 and
not(R1)) or (Q2 and not(Q1) and Q0 and not(R2)) or (Q2 and not(Q1) and not(R2) and R1);
  D1 <= (not(Q2) and Q1 and not(R2) and not(R1)) or (not(Q2) and Q1 and not(Q0) and not(R2)) or (not(Q2) and not(Q1)
and R2 and not(R1)) or (not(Q2) and not(Q1) and Q0 and not(R2) and R1);
   D2 <= (Q2 and not(Q1) and not(R2)) or (Q2 and not(Q1) and R2 and not(R1)) or (not(Q2) and Q1 and R2 and not(R1))
or (not(Q2) and Q1 and Q0 and not(R2) and R1);
  FFD0: FlipFlopD port map (
        D => D0 and not(Timeout),
        Set => '0',
        Rst => '0',
        Clk => Clock,
        Q => Q0
  );
  FFD1: FlipFlopD port map (
        D => D1 and not(Timeout),
        Set => '0',
        Rst => '0',
        Clk => Clock,
        Q => Q1
  );
  FFD2: FlipFlopD port map (
        D => D2 and not(Timeout),
        Set => '0',
        Rst => '0',
        Clk => Clock,
        Q => Q2
  );
end architecture;

architecture BuildFF of FlipFlopD is
begin
  process(Set,Rst,Clk)
  begin
        if Rst = '1' then
        Q<='0';
        elsif Set = '1' then
        Q<='1';
        elsif Clk = '1' and Clk'event then
        Q<=D;
        end if;
  end process;
end architecture;
```

Esse código foi testado manualmente em uma placa FPGA com uma frequência de Clock de 1 Hz (com conversor para frequências baixas, de modo a permitir que o ser humano lide com as entradas há tempo) e funciona. De mesmo modo, ele funciona para uma frequência de clock alta, de acordo com as simulações abaixo:

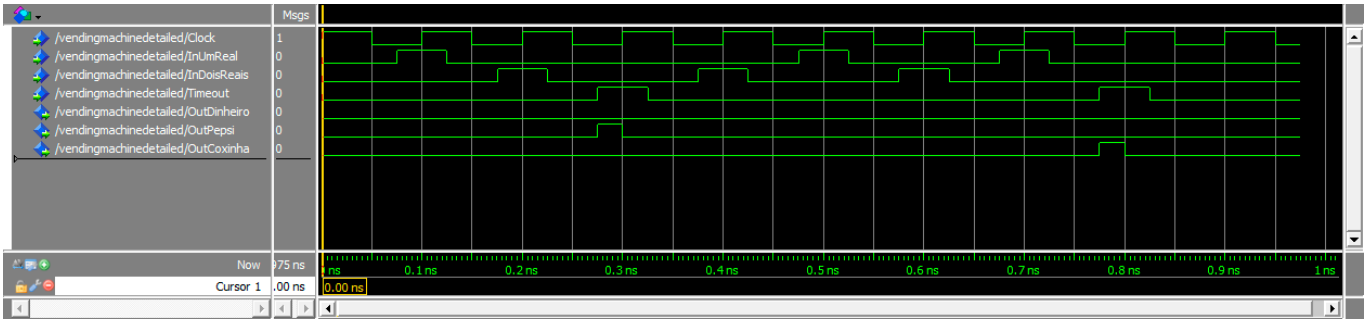


Figura 2. Resultados obtidos na simulação do código VHDL acima.

7. Código VHDL simplificado

O código acima pode ser muito simplificado, de forma a facilitar não apenas o entendimento, mas também a própria organização e documentação do projeto. Com isso, foi criado um código em linguagem VHDL mais simples, em que apenas a lógica da máquina de estados finitos é, propriamente dito, implantada. Óbvio que os resultados são os mesmos e que após a compilação do projeto em um software adequado, não haveria diferenças no funcionamento.

Este código melhorado está descrito abaixo:

```
-- VendingMachine.vhd
library ieee;
use ieee.std_logic_1164.all;

entity VendingMachine is -- Top-Level: Vending Machine
  port(
        Clock:                    in bit;
        InUmReal, InDoisReais, Timeout:   in bit;
        OutDinheiro, OutCoxinha, OutPepsi:  out bit
  );
end entity;

architecture Main of VendingMachine is
type fsm_state is (S0,S1,S2,S3,S4,S5);
signal State : fsm_state;
begin
  process (Clock)
  begin
        if Clock = '1' and Clock'event then
        if Timeout = '1' then -- Timeout no nosso projeto simula o Reset e é sincrono. porque antes de resetar ele
deve retornar a saída.
        case State is
        when S0 =>
                OutDinheiro <= '0';
                OutPepsi <= '0';
                OutCoxinha <= '0';
        when S1 =>
                OutDinheiro <= '1';
                OutPepsi <= '0';
                OutCoxinha <= '0';
        when S2 =>
                 OutDinheiro <= '0';
                OutPepsi <= '1';
                OutCoxinha <= '0';
        when S3 =>
                OutDinheiro <= '0';
                OutPepsi <= '1';
                OutCoxinha <= '0';
        when S4 =>
                OutDinheiro <= '0';
                OutPepsi <= '1';
                OutCoxinha <= '0';
        when S5 =>
                OutDinheiro <= '0';
                OutPepsi <= '0';
                OutCoxinha <= '1';
        end case;
        State <= s0; -- Resetar o estado
        else
        case State is
        when S0=>
                OutDinheiro <= '0'; -- Resetar os valores
                OutPepsi <= '0';
              OutCoxinha <= '0';
              if InUmReal = '1' then
                State <= S1;
              elsif InDoisReais = '1' then
                State <= S2;
```

```
            end if;
          when S1=>
            if InUmReal = '1' then
               State <= S2;
               elsif InDoisReais = '1' then
               State <= S3;
               end if;
        when S2=>
               if InUmReal = '1' then
               State <= S3;
               elsif InDoisReais = '1' then
               State <= S4;
               end if;
        when S3=>
               if InUmReal = '1' then
                State <= S4;
               elsif InDoisReais = '1' then
               State <= S5;
               end if;
        when S4=>
               if InUmReal = '1' or InDoisReais = '1' then
               State <= S5;
               end if;
           when S5=>
               -- Não fazer nada.
        end case;
        end if;
        end if;
  end process;
end architecture;
```

Esse código também foi testado em uma placa FPGA com as mesmas definições da seção anterior e funciona. Similarmente, produz os mesmos resultados na simulação:

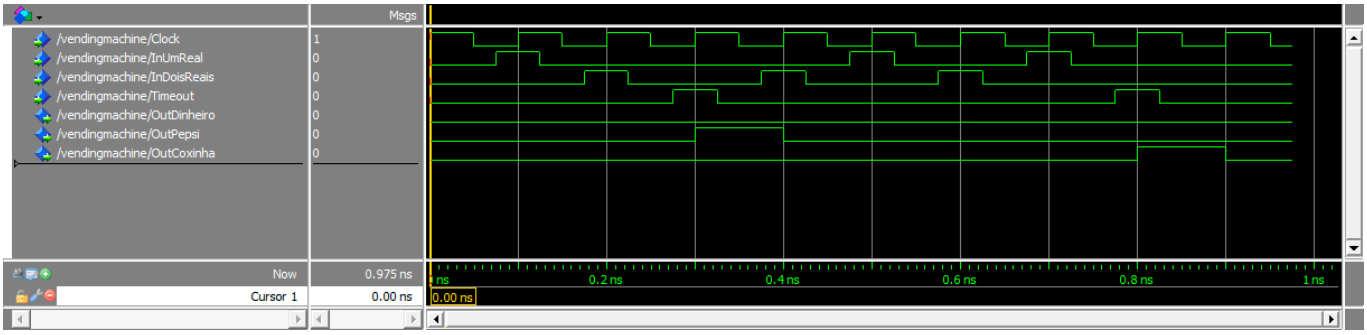


Figura 3. Resultados obtidos na simulação do código VHDL acima.

Perceba que as saídas são idênticas para um determinado conjunto de entradas. A única diferença está presente no tempo em que a saída permanece ativa. Enquanto no código manual esse tempo só fica até a próxima “subida” da borda do clock, no código simplificado esse tempo permanece durante todo um período de clock. São duas opções diferentes que podem ser usadas para lidar com a saída, e a escolha depende apenas de quem irá implantar a máquina.

8. Circuito final

Os códigos VHDL acima podem ser convertidos e simplificados de forma a gerarem um circuito final similar ao ilustrado abaixo:

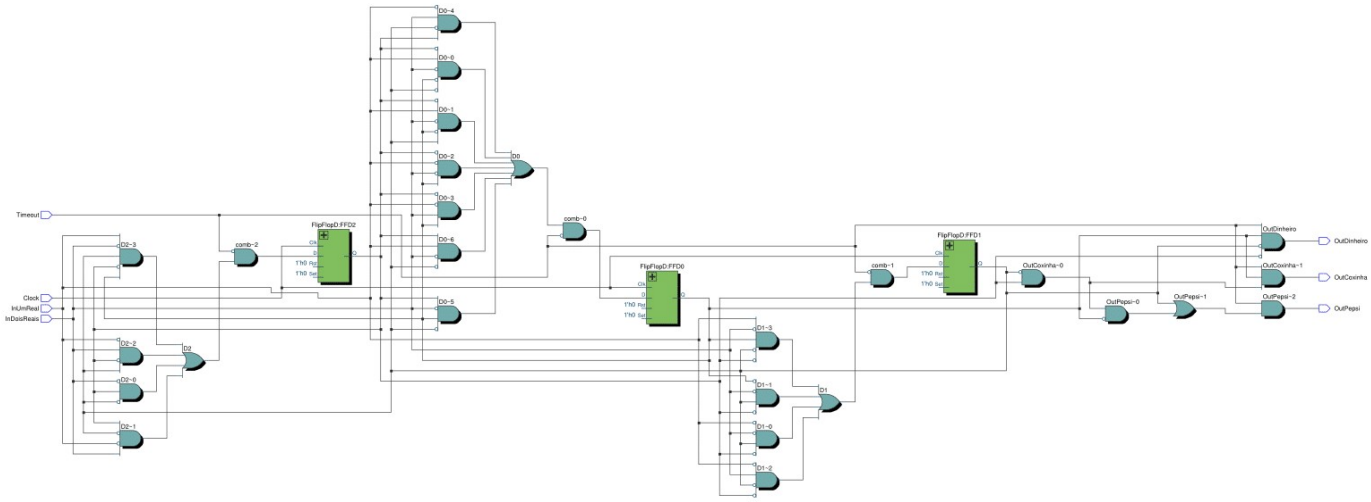


Figura 4. Projeto do circuito (este diagrama está disponível em maior definição no arquivo do projeto).