



Ministério da Educação  
**Universidade Tecnológica Federal do Paraná**  
Câmpus Apucarana  
Bacharelado em Engenharia de Computação



---

Universidade Tecnológica Federal do Paraná

Engenharia de Computação

# **IMPLEMENTAÇÃO DE SINALEIRO UTILIZANDO MÁQUINA DE ESTADOS**

Maria Eduarda Pedroso

Professor orientador: Marcelo de Oliveira

Apucarana

Outubro/2023



## SUMÁRIO

<b>1. RESUMO.....</b>	<b>2</b>
<b>2. INTRODUÇÃO.....</b>	<b>3</b>
2.1. Objetivos.....	4
<b>3. METODOLOGIA.....</b>	<b>4</b>
3.1. Funcionalidades do projeto.....	5
3.1.1. Máquina de estados.....	5
3.1.2. Saída nos leds.....	6
3.1.3. Modo Standby.....	7
3.1.4. Modo teste.....	7
3.2. Código VHDL.....	7
<b>4. RESULTADOS E DISCUSSÃO.....</b>	<b>11</b>
<b>5. CONSIDERAÇÕES FINAIS.....</b>	<b>15</b>



## 1. RESUMO

Este projeto envolve a implementação de semáforos de cruzamento utilizando a linguagem VHDL e máquinas de estado finito (FSM). O objetivo principal é simular dois semáforos interagindo como em um cruzamento, onde o semáforo oposto permanece vermelho enquanto o outro está verde. O código VHDL incorpora modos de operação, como Normal, Standby e Teste, proporcionando flexibilidade ao sistema. O semáforo é controlado por uma máquina de estado, alternando entre os estados de vermelho, verde e amarelo. Além disso, foi implementada uma lógica para garantir que, durante o estado verde de um semáforo, o semáforo oposto permaneça vermelho, proporcionando a funcionalidade típica de um cruzamento. O código também inclui modos adicionais, como Standby, onde o semáforo entra em um estado de espera com a luz amarela piscando, e Teste, que acelera o ciclo dos semáforos para fins de depuração e teste.

Palavras-chave: Timer, VHDL, FPGA, LED, Display de 7 segmentos, Hardware específico.



## 2. INTRODUÇÃO

A implementação de semáforos desempenha um papel fundamental no controle do tráfego, contribuindo para a segurança e eficiência das interseções viárias. Este projeto visa desenvolver uma representação simulada de semáforos de cruzamento utilizando a linguagem VHDL e máquinas de estado finito (FSM). A simulação busca replicar o comportamento dinâmico de semáforos em um cruzamento, onde a coordenação eficiente é crucial para evitar conflitos entre fluxos de tráfego opostos.

A abordagem adotada inclui a modelagem de dois semáforos interdependentes, cada um controlando um braço do cruzamento. Durante a fase verde de um semáforo, o semáforo oposto permanece no estado vermelho, refletindo o comportamento realístico de um cruzamento. Essa interação é implementada por meio de máquinas de estado que coordenam as transições entre os estados de vermelho, verde e amarelo.

Além disso, foram incorporados modos de operação adicionais, como Standby, onde o semáforo entra em um estado de espera com a luz amarela piscando, e Teste, que acelera o ciclo dos semáforos para facilitar a depuração e teste.

O presente trabalho não apenas oferece uma solução funcional para a simulação de semáforos de cruzamento, mas também proporciona uma plataforma modular e expansível para futuras adaptações e otimizações. Através desta implementação, busca-se explorar as capacidades da VHDL na modelagem de sistemas complexos de controle de tráfego, com potencial aplicação em projetos de automação e segurança viária.



## 2.1. Objetivos

O presente trabalho tem como objetivo:

- **Modelagem de Semáforos de Cruzamento:** O principal objetivo deste projeto é desenvolver uma representação simulada de semáforos de cruzamento em VHDL. A modelagem busca refletir o comportamento dinâmico de semáforos em um cruzamento viário, considerando a interdependência entre diferentes braços do cruzamento.
- **Utilização de Máquinas de Estado Finito (FSM):** O projeto utiliza conceitos de máquinas de estado finito para controlar o ciclo de funcionamento dos semáforos. A implementação dessas máquinas de estado permite a coordenação eficiente entre os semáforos, garantindo transições suaves entre os estados de vermelho, verde e amarelo.
- **Simulação de Comportamento Realístico:** A simulação visa replicar o comportamento realístico de semáforos em um cruzamento, onde a mudança de estados é influenciada pela atividade do semáforo oposto. Durante a fase verde de um semáforo, o semáforo oposto permanece no estado vermelho para evitar conflitos entre fluxos de tráfego opostos.
- **Modos de Operação Adicionais:** Além dos modos de operação normais, como o ciclo padrão de semáforos, foram implementados modos adicionais. O modo Standby permite que o semáforo entre em um estado de espera com a luz amarela piscando, enquanto o modo Teste acelera o ciclo dos semáforos para facilitar a depuração e teste.

## 3. METODOLOGIA

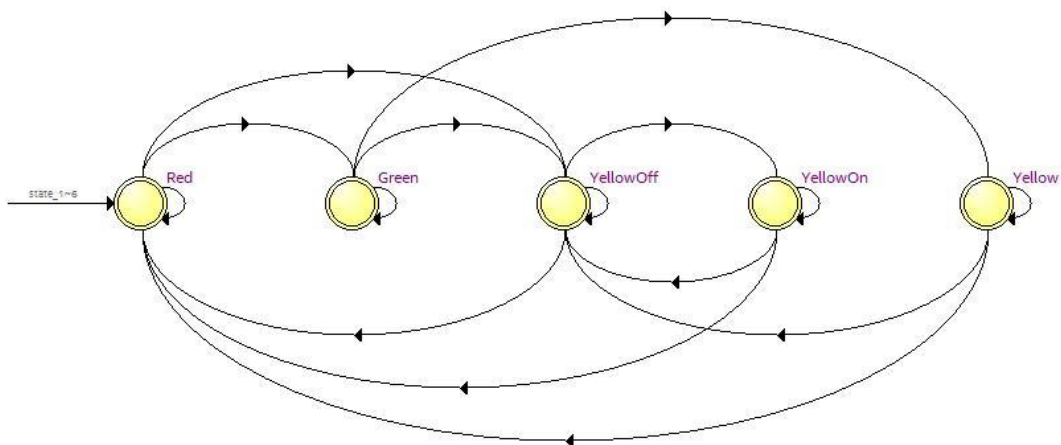
O presente trabalho foi desenvolvido utilizando os conceitos e técnicas estudados durante as aulas da disciplina. Bem como foi empregado o uso do software Quartus, de modo a facilitar e simular nossa lógica.

### 3.1. Funcionalidades do projeto

#### 3.1.1. Máquina de estados

A implementação do projeto utiliza uma máquina de estados para controlar o comportamento do sistema. Na figura abaixo podemos visualizar nosso diagrama.

Figura 1 - Máquina de estados.



Fonte: Autoria Própria

A máquina de estados é definida pelo tipo enumerado StateType, que possui os estados Red, Yellow, Green, YellowOff e YellowOn. Os estados dos semáforos 1 e 2 são controlados pelos sinais state\_1 e state\_2, respectivamente.

```
type StateType is (Red, Yellow, Green, YellowOff, YellowOn);
```

```
signal state_1, next_state_1 : StateType;  
signal state_2, next_state_2 : StateType;
```



A transição entre estados é realizada na borda de subida do sinal de clock (clk), e a frequência da máquina de estados é configurada pela constante StateMachineFrequencyHz. Durante o teste, a máquina de estados opera em uma frequência diferente, definida pelo botão de teste (teste).

```
constant StateMachineFrequencyHz : integer := 1;

process(clk, reset)
  variable counter : integer := 0;
  variable timeCounter : integer := 0;
  variable StateMachineSteps : integer := ClockFrequencyHz/2;

begin
  -- Código omitido para brevidade
end process;
```

### 3.1.2. Saída nos leds

As saídas para os LEDs dos semáforos são controladas pelos processos process(state\_1) e process(state\_2). Cada processo utiliza a instrução case para determinar o valor da saída com base no estado atual do semáforo.

```
process(state_1)
begin
  case state_1 is
    when Red =>
      traffic_light_1 <= "001";
    when Green =>
      traffic_light_1 <= "100";
    when Yellow =>
      traffic_light_1 <= "010";
    when YellowOff =>
      traffic_light_1 <= "000";
```



```
        when YellowOn =>
            traffic_light_1 <= "010";
        end case;
    end process;

    process(state_2)
    begin
        case state_2 is
            when Red =>
                traffic_light_2 <= "001";
            when Green =>
                traffic_light_2 <= "100";
            when Yellow =>
                traffic_light_2 <= "010";
            when YellowOff =>
                traffic_light_2 <= "000";
            when YellowOn =>
                traffic_light_2 <= "010";
        end case;
    end process;
```

### 3.1.3. Modo Standby

Durante o modo Standby, a máquina de estados é pausada. Isso é controlado pelo sinal pause. Quando pause é igual a '0', a máquina de estados permanece no estado atual, evitando transições.

```
if pause = '0' then
    -- Código da máquina de estados para o modo normal
else
    -- Código da máquina de estados para o modo Standby
end if;
```





#### 3.1.4. Modo teste

Durante o modo teste, a máquina de estados opera em uma frequência diferente, simulando condições específicas para o teste do sistema. Isso é controlado pelo sinal teste.

```
if teste = '0' then
  -- Configuração da máquina de estados para o modo de teste
else
  -- Configuração da máquina de estados para o modo normal
end if;
```

### 3.2. Código VHDL

Abaixo temos o código inteiro que foi implementado para a atividade.

```
library IEEE; -- Declaração da biblioteca IEEE
use IEEE.STD_LOGIC_1164.ALL; -- Utilização das definições padrão de lógica 1164
use IEEE.STD_LOGIC_ARITH.ALL; -- Utilização das definições padrão de aritmética
use IEEE.STD_LOGIC_UNSIGNED.ALL; -- Utilização das definições padrão de unsigned

entity aula230830 is
  generic(
    ClockFrequencyHz : integer := 50_000_000 -- Frequência do clock padrão é 50 MHz
  );
  Port (
    clk : in STD_LOGIC; -- Entrada do sinal de clock
    reset : in STD_LOGIC; -- Entrada do sinal de reset
    pause : in STD_LOGIC; -- Entrada do botão de pausa
    teste : in STD_LOGIC; -- Entrada do botão de teste
    traffic_light_1 : out STD_LOGIC_VECTOR (2 downto 0); -- Saída para o semáforo 1
    traffic_light_2 : out STD_LOGIC_VECTOR (2 downto 0) -- Saída para o semáforo 2
  );
end aula230830;

architecture Behavioral of aula230830 is
  type StateType is (Red, Yellow, Green, YellowOff, YellowOn); -- Definição do tipo
  enumerado para os estados
  signal state_1, next_state_1 : StateType; -- Declaração de sinais para os estados do
  semáforo 1
  signal state_2, next_state_2 : StateType; -- Declaração de sinais para os estados do
```



```
semáforo 2
    constant StateMachineFrequencyHz : integer := 1; -- Frequência de atualização da
    máquina de estados em Hz

begin

    process(clk, reset)
        variable counter : integer := 0; -- Variável para contar os passos da máquina de estados
        variable timeCounter : integer := 0; -- Variável para contagem de tempo
        variable StateMachineSteps : integer := ClockFrequencyHz/2; -- Passos da máquina de
        estados

        begin
            if teste = '0' then
                StateMachineSteps := ClockFrequencyHz;
                timeCounter := 0;
            else
                timeCounter := 3;
                StateMachineSteps := ClockFrequencyHz/2;
            end if;

            if reset = '0' then
                state_1 <= Red;
                state_2 <= Green;
                counter := 0;
            elsif rising_edge(clk) then
                if counter < StateMachineSteps then
                    counter := counter + 1;
                else
                    state_1 <= next_state_1;
                    state_2 <= next_state_2;
                    counter := 0;
                end if;
            end if;
        end process;

    process(state_1, state_2)
        variable passou1 : integer := 1; -- Variável para controle de passagem do semáforo 1
        variable passou2 : integer := 0; -- Variável para controle de passagem do semáforo 2
        begin
            if pause = '0' then
                case state_1 is
                    when Red =>
                        if passou1 = 1 then
                            next_state_1 <= Green;
                        else
                            next_state_1 <= Red;
                        end if;
                    end case;
                end if;
            end process;
```



```
end if;

when Green =>
    next_state_1 <= Yellow;
    passou2 := 0;

when Yellow =>
    next_state_1 <= Red;
    passou2 := 1;

when others =>
    next_state_1 <= Red;
end case;

case state_2 is
when Red =>
    if passou2 = 1 then
        next_state_2 <= Green;
    else
        next_state_2 <= Red;
    end if;

when Green =>
    next_state_2 <= Yellow;
    passou1 := 0;

when Yellow =>
    next_state_2 <= Red;
    passou1 := 1;

when others =>
    next_state_2 <= Green;
end case;
else
case state_1 is
when YellowOff =>
    next_state_1 <= YellowOn;
when others =>
    next_state_1 <= YellowOff;
end case;

case state_2 is
when YellowOff =>
    next_state_2 <= YellowOn;
when others =>
    next_state_2 <= YellowOff;
end case;
```



```
end if;
end process;

process(state_1)
begin
  case state_1 is
    when Red =>
      traffic_light_1 <= "001";
    when Green =>
      traffic_light_1 <= "100";
    when Yellow =>
      traffic_light_1 <= "010";
    when YellowOff =>
      traffic_light_1 <= "000";
    when YellowOn =>
      traffic_light_1 <= "010";
  end case;
end process;

process(state_2)
begin
  case state_2 is
    when Red =>
      traffic_light_2 <= "001";
    when Green =>
      traffic_light_2 <= "100";
    when Yellow =>
      traffic_light_2 <= "010";
    when YellowOff =>
      traffic_light_2 <= "000";
    when YellowOn =>
      traffic_light_2 <= "010";
  end case;
end process;

end Behavioral;
```

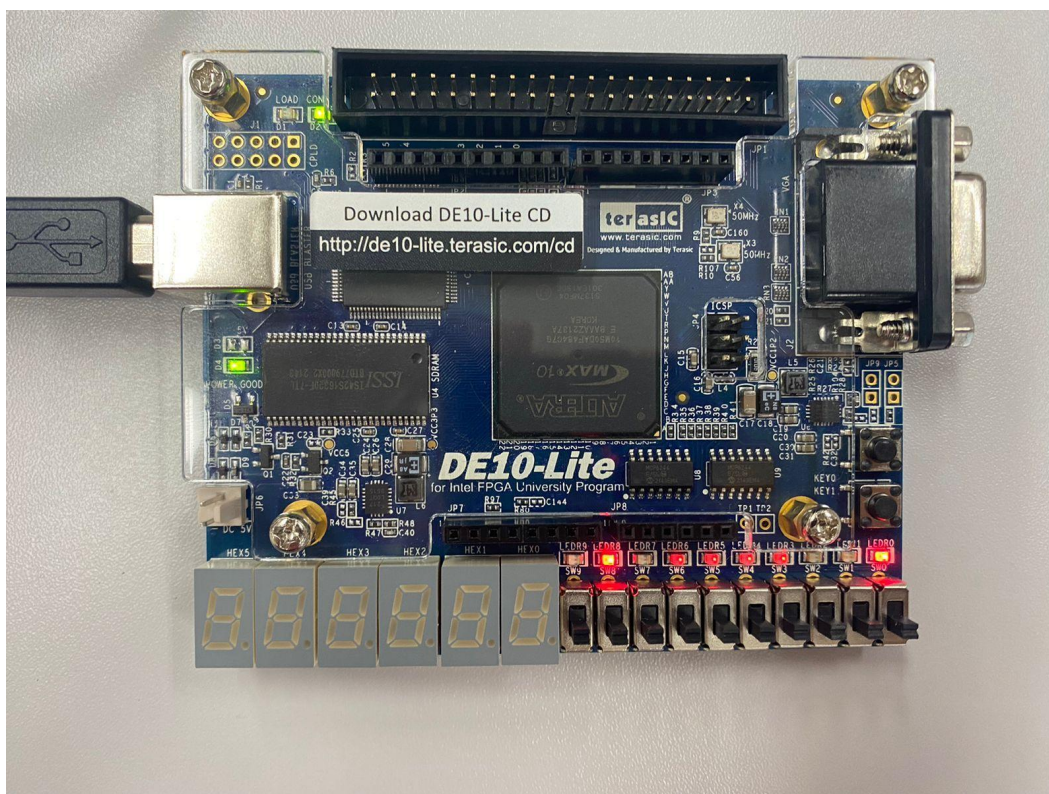
#### 4. RESULTADOS E DISCUSSÃO

Com a implementação funcionando utilizamos da placa FPGA quartus para testar nosso código, abaixo temos duas imagens mostrando as saídas simuladas.



Podemos concluir com as figuras 2, 3, 4 e 5 que nosso código funciona de maneira eficaz, sendo funcional a máquina de estados.

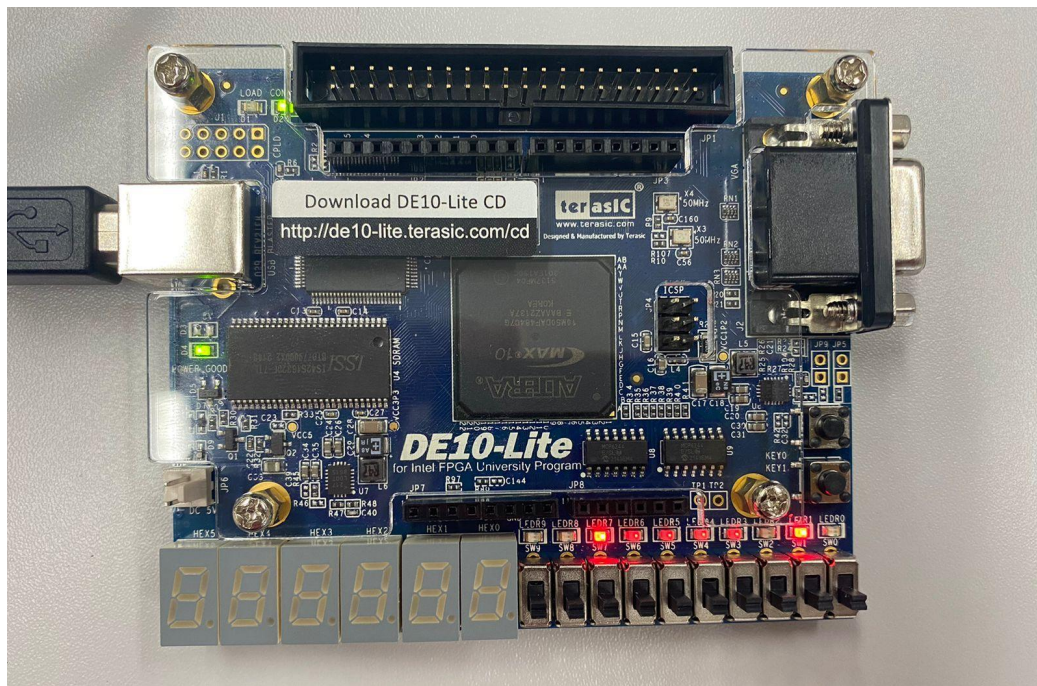
Figura 2 - Semáforos 1.



Fonte: Autoria Própria

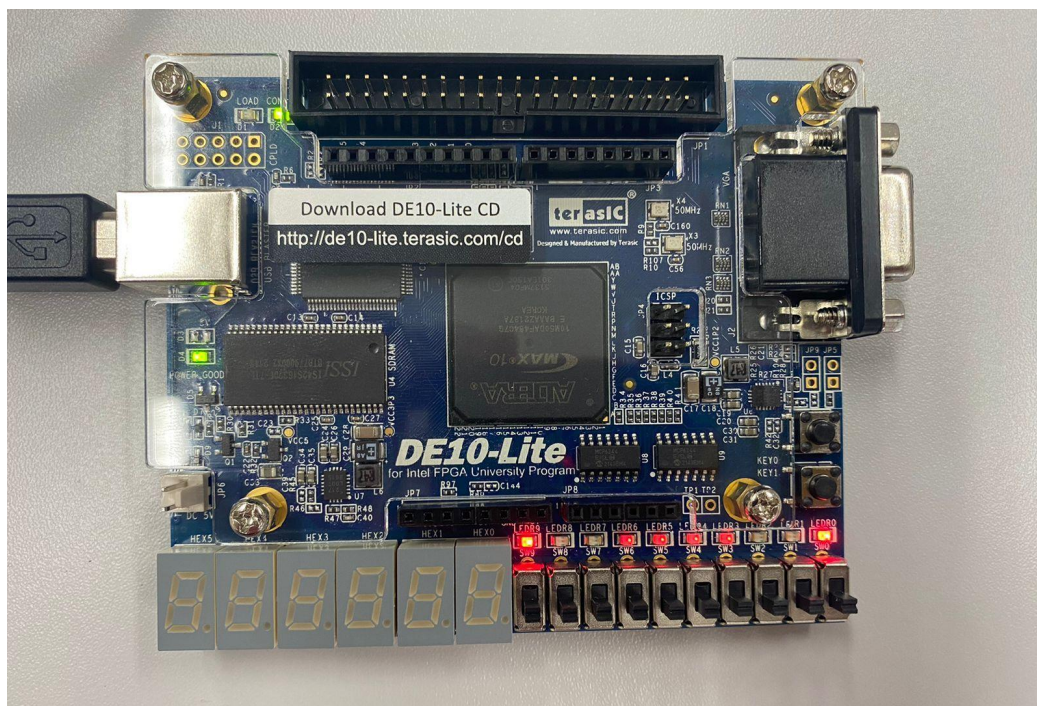


Figura 3 - Semáforos 2.



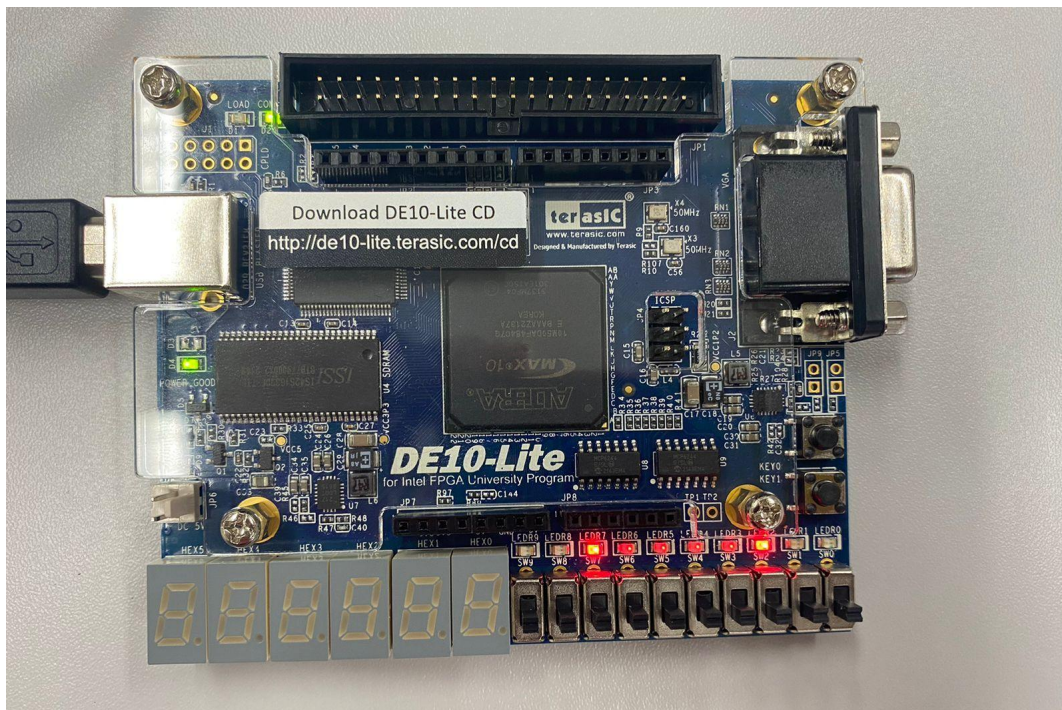
Fonte: Autoria Própria

Figura 4 - Semáforos 3.



Fonte: Autoria Própria

Figura 5 - Semáforos 4.

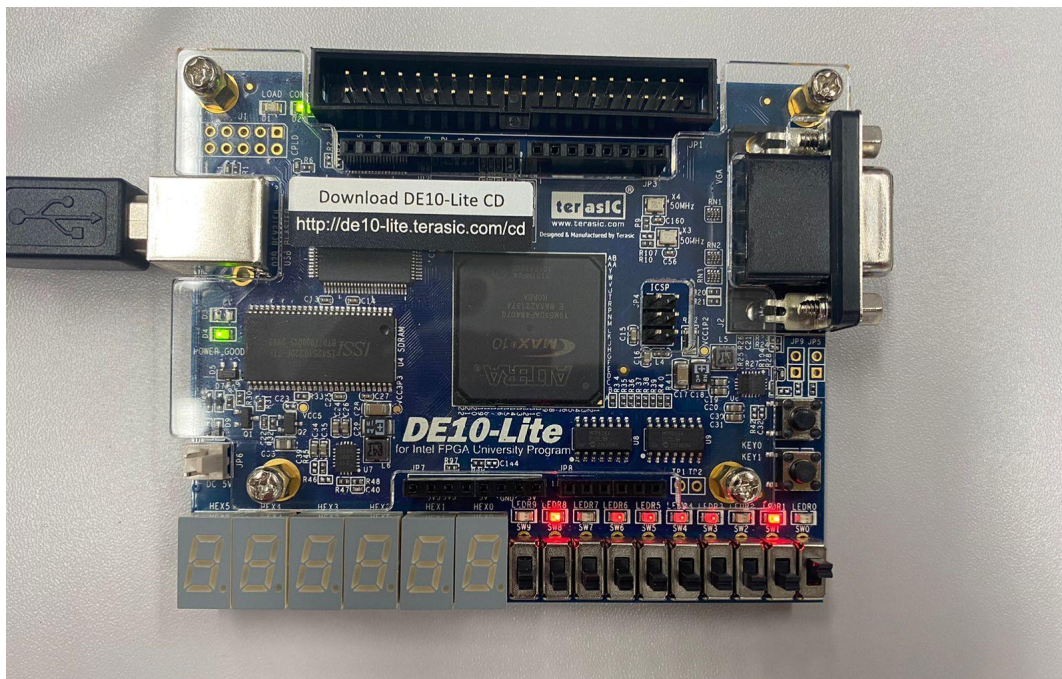


Fonte: Autoria Própria

Na figura 6 e 7 podemos observar o modo standby, mostrando que nosso código faz o amarelo piscar nesse modo, para fim de conhecimento o modo teste funcionou de maneira correta, aumentando a velocidade da máquina mas não é possível exemplificar visualmente.

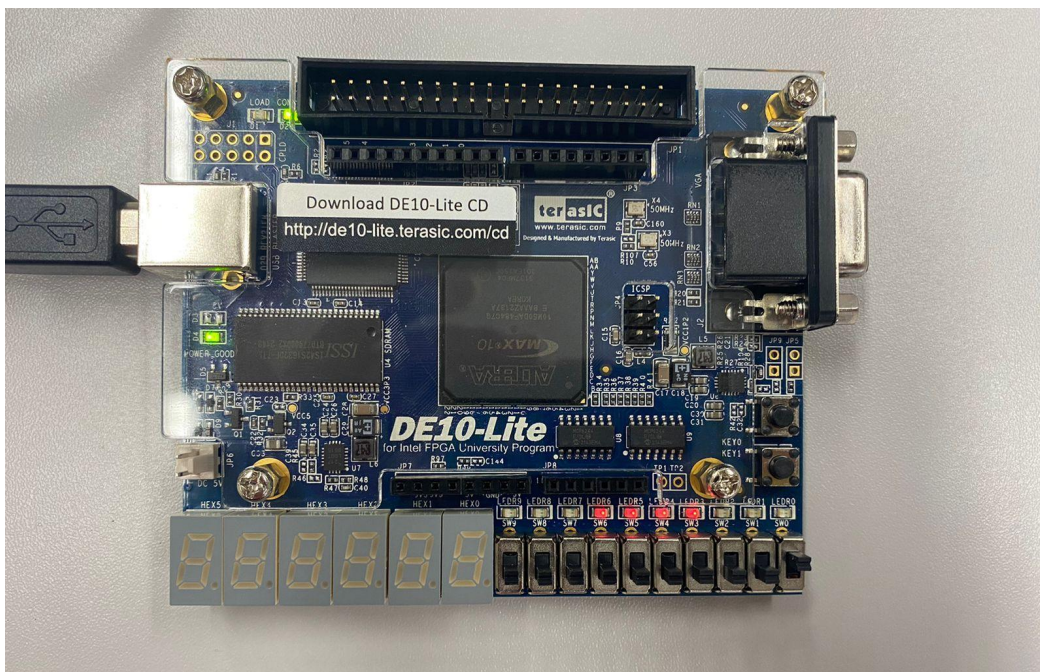


Figura 6 - Standby no estado aceso.



Fonte: Autoria Própria

Figura 7 - Standby no estado apagado.



Fonte: Autoria Própria





## 5. CONSIDERAÇÕES FINAIS

A implementação bem-sucedida dos semáforos de cruzamento em VHDL oferece insights valiosos sobre a modelagem eficiente de sistemas de controle de tráfego em ambientes simulados. A utilização de máquinas de estado finito demonstrou-se crucial para a representação lógica e estruturada do ciclo de funcionamento dos semáforos, permitindo a coordenação eficaz entre diferentes braços de um cruzamento viário simulado.

A consideração da interdependência entre semáforos durante a modelagem, onde o semáforo oposto permanece vermelho durante a fase verde, reflete com precisão as dinâmicas de um cruzamento real. Essa abordagem contribui para a segurança e eficiência do controle de tráfego, minimizando potenciais conflitos entre fluxos de tráfego opostos.

A inclusão de modos de operação adicionais, como Standby e Teste, proporcionou versatilidade ao sistema. O modo Standby permite uma transição suave para um estado de espera, enquanto o modo Teste acelera o ciclo dos semáforos, facilitando processos de depuração e teste em ambientes de desenvolvimento.

Em suma, a implementação não apenas cumpriu seus objetivos, mas também oferece uma base modular e expansível para futuros aprimoramentos e adaptações. A combinação de conceitos avançados da linguagem VHDL com uma abordagem prática aos desafios do controle de tráfego destaca a aplicabilidade e relevância dessa implementação em contextos mais amplos de automação viária.