

**INSTITUTO FEDERAL DE CIÊNCIA E TECNOLOGIA - SP**

**ENGENHARIA DE CONTROLE E AUTOMAÇÃO**



## **PROJETO DE SEMÁFORO INTELIGENTE COM BOTÃO DE PEDESTRE**

**GUSTAVO FERNANDO DE LIMA**

**PERÍODO: NOTURNO – 2º SEMESTRE DE 2025**

**BRAGANÇA PAULISTA**

**05/12/2025**

**BRAGANÇA PAULISTA – SP**

**Gustavo Fernando de Lima**



## **PROJETO DE SEMÁFORO INTELIGENTE COM BOTÃO DE PEDESTRE**

Trabalho apresentado à disciplina de Sistemas Digitais Reconfiguráveis, do curso de Engenharia de Controle e Automação, do Instituto Federal de Educação, Ciência e Tecnologia de São Paulo (IFSP) – Câmpus Bragança Paulista.

Docente: Alexandre Tomazati Oliveira

**Bragança Paulista, 2025**

## SUMÁRIO

1. INTRODUÇÃO.....	4
2. REVISÃO TEÓRICA .....	6
3.DESENVOLVIMENTO.....	25
4. CONCLUSÃO.....	45
5. REFERÊNCIAS.....	47

# 1. INTRODUÇÃO

O avanço das tecnologias digitais aplicadas ao controle e automação tem transformado a forma como sistemas urbanos são projetados, monitorados e otimizados. Entre esses sistemas, os semáforos constituem um dos elementos mais fundamentais da infraestrutura de mobilidade, sendo responsáveis por organizar o fluxo de veículos e pedestres, reduzir acidentes, melhorar a fluidez do tráfego e garantir condições mínimas de segurança em vias públicas. No entanto, apesar de sua aparente simplicidade operacional, os semáforos modernos incorporam diversos mecanismos inteligentes que permitem sua adaptação às condições reais de circulação, ao comportamento de usuários e às demandas crescentes de segurança viária.

Nesse contexto, este projeto tem como proposta o desenvolvimento de um semáforo inteligente com botão de pedestre, implementado inteiramente em hardware reconfigurável por meio da placa FPGA DE10-Lite, utilizando a linguagem de descrição de hardware VHDL. A tecnologia FPGA (Field Programmable Gate Array) permite a criação de circuitos paralelos, determinísticos e altamente confiáveis, características fundamentais para aplicações de controle que exigem precisão temporal e robustez estrutural. Para viabilizar a implementação, síntese e gravação do sistema no dispositivo físico, foi utilizado o ambiente Quartus Prime, ferramenta oficial da Intel para desenvolvimento, compilação, análise e programação de FPGAs.

O Quartus Prime desempenhou papel central no fluxo de desenvolvimento, sendo responsável por todas as etapas de transformação do código VHDL em um circuito configurável. Por meio dele, foi possível realizar a compilação do projeto, verificar erros de síntese, gerenciar o mapeamento físico dos sinais através do Pin Planner, além de utilizar o Programmer para gravar a configuração final na FPGA. Essa ferramenta integrou todas as fases necessárias, desde a escrita do código até sua execução no hardware real, representando um elo fundamental entre o ambiente teórico e a realização prática.

Um dos principais objetivos deste trabalho é demonstrar a aplicação prática do conceito de máquinas de estado finito (FSM – Finite State Machines), amplamente estudado na engenharia de sistemas digitais. As FSMs são essenciais no projeto de circuitos sequenciais, pois permitem modelar o comportamento de sistemas que dependem tanto de suas entradas quanto do histórico de estados anteriores. No caso de um semáforo, esse modelo é indispensável para garantir transições ordenadas entre as fases de verde, amarelo e vermelho, evitando condições perigosas, como o acionamento simultâneo de sinais conflitantes. Conforme apresentado no material da Semana 10 da disciplina de Sistemas Digitais Reconfiguráveis, as máquinas de estado representam uma das técnicas mais robustas e eficientes para controle de sistemas em hardware.

Além de aplicar o conceito de FSM, o projeto também abrange tópicos como temporização baseada em clock, divisão de frequência, lógica combinacional e lógica sequencial. A placa DE10-Lite opera com um clock principal de 50 MHz, frequência inadequada para tempos perceptíveis ao ser humano. Assim, foi necessário desenvolver um divisor de clock, responsável por gerar pulsos de 1 Hz utilizados para medir os intervalos de tempo de cada fase do semáforo. Essa temporização é crucial para simular o comportamento real de um semáforo urbano e garantir tempos adequados tanto para veículos quanto para pedestres.

Outro elemento fundamental do projeto é o botão de pedestre, um recurso presente em cruzamentos urbanos para garantir segurança durante a travessia. No sistema desenvolvido, a solicitação do pedestre é detectada por meio dos botões físicos KEY0 e KEY1 da placa DE10-Lite, ambos ativos em nível lógico baixo, conforme descrito no manual da placa. A FSM foi projetada de modo que, ao identificar um pedido de travessia, o sistema priorize o pedestre e antecipe a mudança para o estado vermelho dos veículos, desde que isso não comprometa o fluxo viário e a segurança operacional.

Além dos aspectos técnicos, o projeto contribui para o desenvolvimento de competências amplas na área de engenharia, como análise, modelagem e implementação de sistemas digitais, leitura de documentação técnica, resolução de problemas e validação de hardware. A integração entre o código VHDL, o uso do Quartus Prime e o comportamento físico observado na placa DE10-Lite proporciona ao estudante uma compreensão mais profunda da relação entre teoria e prática, tornando o aprendizado mais significativo e robusto.

Por fim, o desenvolvimento deste semáforo inteligente não apenas demonstra o domínio das ferramentas e técnicas utilizadas, mas também evidencia a relevância das soluções digitais reconfiguráveis para aplicações do mundo real. A utilização de FPGAs permite criar sistemas de baixo custo, alto desempenho e elevada confiabilidade — qualidades essenciais em infraestruturas urbanas modernas. Assim, o projeto se apresenta como uma oportunidade concreta de aplicar conhecimentos fundamentais da área, ao mesmo tempo em que explora tecnologias presentes em sistemas embarcados, automação urbana e controle digital.

## 2. REVISÃO TEÓRICA

### 2.1 Máquinas de Estado Finito (FSM)

As Máquinas de Estado Finito, ou FSM (Finite State Machines), constituem um dos pilares fundamentais da engenharia de sistemas digitais, sendo amplamente utilizadas no projeto de circuitos sequenciais, sistemas embarcados, protocolos de comunicação e mecanismos de controle. Uma FSM é um modelo matemático que descreve o comportamento de um sistema a partir de um conjunto finito de estados, entradas, transições e saídas, permitindo representar de forma organizada ações que dependem tanto das condições atuais quanto do histórico recente do sistema. Em sistemas digitais, as FSMs são essenciais para a coordenação de processos que exigem sequência lógica, temporização e previsibilidade.

#### 2.1.1 Definição formal

Formalmente, uma máquina de estado é composta por:

- **Estados (S):** representam as condições internas possíveis do sistema. Cada estado descreve uma configuração específica da saída ou do comportamento desejado.
- **Entradas (I):** sinais externos que influenciam o comportamento do sistema, podendo alterar o curso das transições.
- **Transições (T):** são as regras que determinam para qual estado o sistema irá na próxima etapa, com base no estado atual e nas entradas.
- **Saídas (O):** ações produzidas pelo sistema em função do estado atual ou das combinações entre estado e entrada.
- **Função de transição:** define a lógica que relaciona estado atual, entradas e próximo estado.
- **Função de saída:** define a relação entre estados (e entradas, no caso de máquinas Mealy) e os sinais de saída.

Esse modelo matemático permite a criação de sistemas digitais previsíveis, seguros e organizados, facilitando tanto a análise quanto a implementação em hardware.

#### 2.1.2 Máquinas de Moore e Mealy

Duas arquiteturas principais de FSM são utilizadas na prática: **Moore** e **Mealy**. Ambas são conceitualmente semelhantes, mas diferem na forma como geram suas saídas.

##### Máquina de Moore

Nas máquinas de Moore, as saídas dependem exclusivamente do estado atual. Portanto:

$$O = f(\text{estado atual})$$

**Vantagens:**

- Saídas mais estáveis e previsíveis, pois só mudam na troca de estado.
- Menos sensíveis a ruídos ou flutuações rápidas nas entradas.
- Ideal para hardware onde a segurança é importante.

#### **Desvantagens:**

- Pode exigir mais estados para representar o mesmo comportamento de uma máquina Mealy.
- Resposta às entradas pode ser um pouco mais lenta, já que depende da mudança de estado.

O semáforo do presente projeto utiliza um modelo Moore, garantindo que as saídas (sinais dos semáforos) mudem apenas nos instantes adequados, sem oscilações inesperadas.

#### **Máquina de Mealy**

Nas máquinas de Mealy, as saídas dependem do estado atual e das entradas:

$$O = f(\text{estado atual}, \text{entradas})$$

#### **Vantagens:**

- Resposta mais rápida a mudanças nas entradas.
- Pode exigir menos estados.

#### **Desvantagens:**

- Saídas podem apresentar mudanças assíncronas e instáveis.
- Exige maior cautela em sistemas que envolvem segurança.

Por essas razões, máquinas Mealy são evitadas quando pequenas variações de sinal podem resultar em comportamentos perigosos — como no controle de um semáforo.

### **2.1.3 Estrutura Sequencial de uma FSM**

Antes de aplicar ao semáforo, exemplos clássicos ajudam a ilustrar o funcionamento:

- **Portão automático:** estados como “fechado”, “abrindo”, “aberto”, “fechando”.
- **Controle de elevador:** “subindo”, “descendo”, “parado”, “porta abrindo”.
- **Máquinas industriais:** “espera”, “inicialização”, “execução”, “falha”.
- **Sistemas embarcados:** leitores de cartão, detectores de presença, controles remotos.

Em todos esses casos, o sistema responde a eventos externos (sensores, botões, sinais) e transita entre estados de forma ordenada.

O semáforo funciona exatamente assim: suas condições (verde, amarelo e vermelho) são estados bem definidos, e o “botão de pedestre” é um evento externo que influencia as transições.

### **Modelo sequencial: registradores e flip-flops**

Do ponto de vista físico, uma FSM é implementada por meio de elementos de memória, tipicamente flip-flops, que armazenam o estado atual. A cada pulso de clock:

1. O estado atual é carregado nos flip-flops (registrador de estado).
2. A lógica combinacional calcula o próximo estado.
3. As saídas são determinadas com base no estado (Moore) ou nas entradas (Mealy).

O funcionamento é dividido em:

- Lógica sequencial: atualiza o estado nos pulsos de clock.
- Lógica combinacional: define o próximo estado e as saídas.

Esse modelo garante sincronismo, previsibilidade e estabilidade — fatores essenciais para o funcionamento correto do sistema.

### **2.1.4 FSM em VHDL**

A implementação de uma FSM em VHDL segue uma estrutura padrão de três blocos:

#### **Declaração do tipo de estados**

```
type state_t is (S_GREEN, S_YELLOW, S_RED);  
signal state, next_state : state_t;
```

#### **Processo sequencial (registrador de estado)**

Acionado pelo **clock**, atualiza o estado atual:

```
process(clk)  
begin  
    if rising_edge(clk) then  
        state <= next_state;  
    end if;  
end process;
```

#### **Processo combinacional (transições)**

Define o próximo estado com base no estado atual e nas entradas:

```
process(state, ped_req, sec_count)  
begin
```

```

case state is
  when S_GREEN =>
    if ped_req = '1' then
      next_state <= S_RED;
    elsif sec_count >= T_GREEN_SEC-1 then
      next_state <= S_YELLOW;
    else
      next_state <= S_GREEN;
    end if;

```

### **Processo de saída**

Define quais LEDs acendem:

```

process(state)
begin
  case state is
    when S_GREEN =>
      hex0 <= SEG_GRN;
      hex1 <= SEG_RED;

```

Essa estrutura é padrão em projetos profissionais e acadêmicos.

## **2.1.5 Justificativa do Uso de FSM em Semáforos**

O funcionamento de um semáforo é, essencialmente, um ciclo lógico de estados. Por isso, a FSM é a forma natural e mais eficiente de implementá-lo. Entre as vantagens específicas:

### **Clareza e organização**

Cada cor (verde, amarelo, vermelho) corresponde a um estado bem definido.

### **Segurança**

FSM garante que não existam condições proibidas, como:

- carros vermelho e pedestre vermelho
- carros verde e pedestre verde

### **Temporização precisa**

A FSM pode integrar contadores e divisores de clock, controlando exatamente o tempo de cada fase.

### **Resposta controlada ao botão de pedestre**

O evento de pedestre pode influenciar as transições sem causar interrupções perigosas.

### **Simplicidade de implementação em VHDL**

O modelo Moore é direto, limpo e de fácil manutenção.

### **Determinismo**

Cada estado e transição é conhecido e previsível, essencial para sistemas urbanos.

## **2.2 Temporização e Divisor de Clock**

A temporização é um dos elementos mais essenciais em sistemas digitais sequenciais, especialmente quando o comportamento do circuito depende diretamente do tempo. No contexto deste projeto, o controle preciso do tempo é fundamental para definir as durações dos estados do semáforo, garantindo que as fases de verde, amarelo e vermelho ocorram com intervalos adequados para a travessia de pedestres e para o fluxo seguro de veículos. Para isso, a placa DE10-Lite utiliza um clock principal de alta frequência, e cabe ao projetista dividir e adaptar esse sinal para gerar temporizações adequadas ao uso humano. Nesta seção, são abordados os fundamentos do sinal de clock, os divisores de frequência e a aplicação prática da temporização no sistema de semáforo inteligente.

### **2.2.1 Sinal de clock**

O clock é um sinal oscilatório periódico responsável por marcar os instantes nos quais um circuito sequencial avalia suas entradas, atualiza seus estados e gera novas saídas. Em outras palavras, ele funciona como um “ritmo” que sincroniza todas as operações internas de um sistema digital. Cada subida ou descida do pulso de clock representa um instante de referência, permitindo que os componentes, como flip-flops e registradores, trabalhem de maneira coordenada e previsível.

A placa DE10-Lite possui um oscilador interno que fornece um clock padrão de 50 MHz, o que significa que ele oscila cinquenta milhões de vezes por segundo. Embora essa frequência seja adequada para aplicações de alta velocidade, ela é demasiadamente rápida para sistemas que exigem interações humanas, como o acionamento e a visualização de sinais de trânsito. Portanto, o clock deve ser transformado em um sinal de baixa frequência — tipicamente 1 Hz — para representar adequadamente intervalos de um segundo.

O uso do clock é fundamental para garantir:

- **determinismo**, pois cada mudança ocorre em um instante exato;
- **sincronismo**, evitando que partes do circuito operem de forma descoordenada;
- **estabilidade**, uma vez que alterações de estado não dependem de atrasos ou ruídos externos.

Sem um clock adequado, o comportamento do semáforo poderia se tornar imprevisível ou inseguro, prejudicando a integridade da lógica do sistema.

### 2.2.2 Divisores de frequência

Para converter o clock de 50 MHz em um pulso de 1 Hz, utiliza-se um divisor de frequência, normalmente implementado com contadores binários. Essa técnica consiste em incrementar um contador a cada pulso de clock e gerar uma saída sempre que um valor predeterminado for atingido. No caso deste projeto, a geração de 1 Hz requer contar até:

50 000 000 ciclos por segundo

Assim, o divisor de clock opera da seguinte forma:

1. Recebe o clock de 50 MHz.
2. Incrementa um contador (tick\_cnt) a cada borda de subida.
3. Quando o contador atinge 49.999.999, ele zera.
4. Nesse instante, ativa o sinal one\_sec\_tick, indicando a passagem de 1 segundo.

Esse mecanismo transforma um sinal extremamente rápido em um sinal lento e regular, que pode ser usado para medir tempos reais perceptíveis pelo usuário.

Os divisores de frequência são amplamente utilizados em sistemas embarcados e digitais, desempenhando funções como:

- geração de relógios secundários;
- controle de temporizadores;
- criação de ciclos de máquina;
- sincronização de protocolos de comunicação.

Sua implementação direta em VHDL é eficiente, permitindo que o FPGA execute o processo paralelamente sem afetar o desempenho geral do sistema.

### 2.2.3 Temporização aplicada ao semáforo

Após gerar o pulso de 1 Hz, esse sinal é utilizado como base de temporização para controlar os estados da FSM do semáforo. Cada vez que o pulso ocorre, incrementa-se o contador de segundos (sec\_count). O sistema compara esse valor com os tempos definidos para cada estado:

- **Tempo de verde (T\_GREEN\_SEC = 5 s):** período durante o qual os veículos têm prioridade no cruzamento.
- **Tempo de amarelo (T\_YELLOW\_SEC = 2 s):** indica a transição e alerta os motoristas para reduzir a velocidade e preparar a parada.
- **Tempo de vermelho/pedestre (T\_RED\_SEC = 3 s):** permite ao pedestre atravessar com segurança.

A combinação entre a FSM e a temporização garante que:

1. Cada estado dure exatamente o tempo especificado.

2. O botão de pedestre só modifique o fluxo no instante seguro.
3. Não ocorram mudanças abruptas que comprometam a segurança.
4. O ciclo reinicie de forma ordenada após o término de cada fase.

A temporização desempenha ainda o papel de registrar o histórico do tempo passado no estado atual, permitindo que a FSM tome decisões baseadas não apenas em eventos externos (como o botão), mas também no tempo já decorrido — comportamento essencial para sistemas de controle dependentes de ciclos temporais.

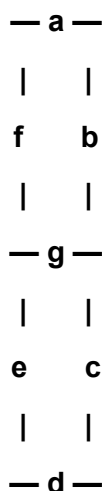
Assim, o divisor de clock e o mecanismo de temporização constituem o núcleo do funcionamento temporal do semáforo, garantindo precisão, segurança e realismo na simulação do comportamento de semáforos urbanos.

## 2.3 Displays de 7 Segmentos

Os displays de 7 segmentos são amplamente utilizados em sistemas digitais embarcados para a exibição de informações simples, como números, símbolos e indicadores luminosos. Na placa DE10-Lite, os displays HEX desempenham a função de representar visualmente os estados do semáforo desenvolvido neste projeto, permitindo que o usuário observe de maneira clara o comportamento da FSM. Para isso, é essencial compreender sua estrutura física, sua lógica de acionamento e as máscaras de bits utilizadas para controlar cada segmento.

### 2.3.1 Estrutura do display de 7 segmentos

Um display de 7 segmentos é composto por sete LEDs individuais, organizados de forma a permitir a formação de algarismos de 0 a 9, além de algumas letras ou símbolos. Esses LEDs são representados por segmentos nomeados tradicionalmente pelas letras a, b, c, d, e, f e g, formando o seguinte arranjo:



Além desses sete segmentos principais, alguns modelos incluem também um oitavo ponto luminoso chamado DP (Decimal Point), embora na DE10-Lite ele nem sempre seja utilizado.

Cada segmento do display funciona como um LED separado: quando energizado, ele acende; quando desligado, permanece apagado. A combinação de segmentos acesos permite formar caracteres. Por exemplo:

- Zero acende: a, b, c, d, e, f
- Um acende: b, c
- Três acende: a, b, c, d, g

Na DE10-Lite, o display é utilizado não para números, mas para representar o estado do semáforo em forma de pontos e barras iluminadas, simulando as cores vermelho, amarelo e verde do farol.

Assim, cada um dos displays HEX0 e HEX1 funciona como um conjunto de LEDs controlados digitalmente pela FPGA.

### 2.3.2 Lógica ativa baixa

Um aspecto crucial para o correto controle dos displays da DE10-Lite é a utilização de lógica ativa baixa, conforme descrito no manual da placa. Em lógica ativa baixa, o LED acende quando recebe nível lógico 0 e permanece apagado quando recebe 1. Isso é o inverso de muitos circuitos digitais, nos quais o nível alto (1) normalmente significa ativação.

Esse comportamento ocorre porque o display é do tipo catodo comum, no qual todos os catodos dos LEDs estão conectados em comum, e cada anodo é controlado individualmente por um transistor. Assim:

- **0** = LED acende (ativo)
- **1** = LED apaga (inativo)

Essa inversão exige que as máscaras de controle considerem o bit “0” como segmento ligado e “1” como desligado.

Por exemplo:

- Para acender somente o segmento “a”, deve-se enviar 0 naquele bit específico e 1 nos demais.
- Para desligar totalmente o display, usa-se 11111111 (todos apagados).

A lógica ativa baixa é fundamental para entender o funcionamento das máscaras binárias e evitar erros como acender o segmento errado ou exibir padrões indesejados.

### 2.3.3 Máscaras usadas no projeto

O projeto utiliza três padrões de máscara, cada um representando uma cor do semáforo (vermelho, amarelo e verde). Em vez de exibir números, cada máscara acende um segmento específico do display para simular o farol correspondente.

As máscaras definidas em VHDL são:

```
constant SEG_OFF : std_logic_vector(7 downto 0) := (others => '1');
```

```
constant SEG_RED : std_logic_vector(7 downto 0) := "11111110"; -- segmento 0
constant SEG_YEL : std_logic_vector(7 downto 0) := "10111111"; -- segmento 6
constant SEG_GRN : std_logic_vector(7 downto 0) := "11110111"; -- segmento 3
```

### **SEG\_OFF**

- "11111111"
- Todos os bits em 1 → nenhum segmento acende.

### **SEG\_RED – Sinal vermelho**

- "11111110"
- Somente o bit menos significativo é 0 → acende o segmento **0**.
- Representa o LED vermelho do semáforo.

### **SEG\_YEL – Sinal amarelo**

- "10111111"
- Acende o segmento **6**.
- Representa o LED amarelo.

### **SEG\_GRN – Sinal verde**

- "11110111"
- Acende o segmento **3**.
- Representa o LED verde.

### **Por que apenas um segmento acende?**

Porque o objetivo do projeto não é exibir números, mas sim representar visualmente os estados do semáforo.  
Com isso:

- HEX0 funciona como o semáforo dos carros
- HEX1 funciona como o semáforo dos pedestres

A escolha desses segmentos é simples, minimalista e clara, simulando três LEDs separados.

### **Relação com a FSM**

Cada estado da máquina de estados seleciona a máscara apropriada:

- **S\_GREEN** → **SEG\_GRN**
- **S\_YELLOW** → **SEG\_YEL**
- **S\_RED** → **SEG\_RED** (carros) + **SEG\_GRN** (pedestre)

Essa associação garante que o comportamento visual reproduza exatamente a lógica da FSM.

## 2.4 Botões e Entradas Digitais

Os botões desempenham um papel fundamental em sistemas digitais interativos, pois permitem que o usuário envie comandos ou sinais ao hardware de forma simples e direta. No projeto do semáforo inteligente, os botões KEY0 e KEY1 da placa DE10-Lite são responsáveis por gerar o pedido de travessia do pedestre, alterando o comportamento da máquina de estados e priorizando o cruzamento humano quando necessário. Para compreender plenamente sua operação, é necessário discutir conceitos como níveis lógicos, sinais digitais, acionamento ativo baixo e os fenômenos físicos associados ao acionamento mecânico de botões, como o bounce. A seguir, cada uma dessas características é detalhada.

### 2.4.1 Níveis lógicos em entradas digitais

Em sistemas digitais, as informações são representadas por dois níveis elétricos: nível alto (1) e nível baixo (0). Esses níveis são interpretados pelo FPGA como valores binários, permitindo a criação de circuitos combinacionais e sequenciais. Entradas digitais, como as provenientes de botões, chaves ou sensores, atuam diretamente sobre a lógica interna do sistema, podendo:

- alterar estados de uma FSM;
- gerar eventos assíncronos ou síncronos;
- acionar interrupções;
- modificar o comportamento do sistema em tempo real.

No caso da DE10-Lite, cada botão KEY é conectado internamente a um circuito resistivo que define seu valor padrão quando não está sendo pressionado. Essa estrutura garante que o FPGA sempre leia níveis lógicos bem definidos, evitando leituras indefinidas ou flutuantes.

### 2.4.2 Botões ativo-baixo (active-low)

Um elemento particularmente relevante na DE10-Lite é o fato de seus botões operarem com lógica ativa baixa, também chamada de *active-low*. Isso significa que:

- Quando o botão está solto → o valor lido é 1 (nível alto).
- Quando o botão é pressionado → o valor lido é 0 (nível baixo).

Esse comportamento ocorre devido à arquitetura interna dos botões da placa, que utiliza resistores de pull-up conectados ao VCC. Assim, ao pressionar o botão, o circuito é fechado e o terminal é conectado diretamente ao terra (GND), resultando em nível lógico baixo.

Isso exige que o código VHDL interprete a lógica de forma invertida. Por exemplo:

```
ped_req <= (not key0) or (not key1);
```

Nesse caso:

- Se KEY0 = 1 → botão não pressionado → not KEY0 = 0

- Se  $KEY0 = 0 \rightarrow$  botão pressionado  $\rightarrow$  not  $KEY0 = 1$

Ou seja, o sistema considera  $ped\_req = 1$  quando qualquer botão é pressionado.

Essa característica deve ser tratada com cuidado, pois erros na interpretação da lógica ativa baixa podem levar a comportamentos incorretos, como o sistema entender que o botão está sempre pressionado ou nunca pressionado.

### 2.4.3 Fenômeno de bounce (oscilação) e debounce

Botões mecânicos físicos, como os presentes na DE10-Lite, não realizam uma troca instantânea e limpa entre os estados de aberto e fechado. Devido à natureza metálica e ao impacto físico das superfícies internas, ocorre o fenômeno chamado bounce (ou *rebounding*), caracterizado por uma série de oscilações elétricas rápidas antes da estabilização do sinal.

Durante o bounce:

- o botão não fornece imediatamente um único pulso limpo;
- em vez disso, gera vários pulsos  $0 \leftrightarrow 1$  sucessivos em poucos microssegundos;
- para sistemas digitais rápidos, como um FPGA, esses pulsos parecem múltiplos acionamentos.

Em um sistema mal projetado, isso poderia:

- gerar múltiplos pedidos de pedestre quando o usuário pressiona o botão uma única vez;
- provocar mudanças inesperadas na FSM;
- comprometer a segurança e consistência do semáforo.

Como a DE10-Lite possui um leve circuito de debounce interno, o efeito é minimizado, mas ainda pode ocorrer em determinadas situações. O uso de uma lógica síncrona baseada em clock — como neste projeto — ajuda a limitar os efeitos do bounce, pois o sistema só avalia o botão durante os pulsos de 1 segundo do divisor de clock, o que naturalmente filtra oscilações rápidas.

### Relação com o semáforo inteligente

O uso dos botões KEY0 e KEY1 como entrada para o pedido de pedestre permite que o sistema receba sinais externos e ajuste dinamicamente o comportamento da FSM. Além disso:

- a lógica ativa baixa garante que o hardware opere de forma previsível;
- o divisor de clock e o FSM filtram ruídos e bounce;
- qualquer acionamento é convertido em um único sinal  $ped\_req = 1$ ;
- o sistema só muda para vermelho quando estiver seguro fazê-lo.

Assim, a interação entre o usuário e a lógica de controle é simples, segura e eficiente, refletindo fielmente o funcionamento real de semáforos urbanos com acionamento manual.

## 2.5 FPGA e Arquitetura da DE10-Lite

Os FPGAs (Field Programmable Gate Arrays) representam uma das tecnologias mais versáteis e poderosas no campo dos sistemas digitais, permitindo que circuitos sejam configurados diretamente pelo projetista, com grande flexibilidade e desempenho. A placa DE10-Lite, utilizada neste projeto, integra o FPGA Intel MAX 10, que possui arquitetura moderna, ampla capacidade lógica e diversos recursos integrados, como memória interna, blocos DSP, PLLs e interfaces de entrada e saída. A compreensão dos fundamentos de FPGAs e dos recursos específicos da DE10-Lite é essencial para entender como o semáforo inteligente foi desenvolvido e executado no hardware físico.

### 2.5.1 O que é um FPGA

Um FPGA é um circuito integrado programável que permite ao usuário definir, por meio de linguagens de descrição de hardware (como VHDL ou Verilog), a forma como o chip funcionará internamente. Diferentemente dos microcontroladores, que executam instruções sequenciais em um processador central, os FPGAs oferecem paralelismo massivo, já que sua lógica é configurada diretamente em hardware.

Em termos gerais, um FPGA é composto por:

- **Células lógicas programáveis (LEs ou ALMs):** pequenos blocos que podem implementar funções lógicas combinacionais e flip-flops.
- **Matriz de interconexões:** redes internas que conectam livremente os blocos lógicos.
- **Blocos de memória:** utilizados como RAM, ROM ou buffers.
- **Unidades DSP:** aceleradores para cálculos matemáticos mais complexos (quando presentes).
- **Entradas e saídas configuráveis (GPIO):** permitem conexão com sensores, botões, LEDs e periféricos externos.
- **Recursos de temporização como PLLs:** para controlar e gerar múltiplos sinais de clock.

A principal vantagem dos FPGAs é que o comportamento final do sistema não depende de software interpretado, mas sim de circuitos efetivamente sintetizados no silício, garantindo:

- baixa latência,
- determinismo,
- paralelismo,
- confiabilidade,

- alto desempenho em aplicações de controle.

No caso deste projeto, esses benefícios tornam os FPGAs ideais para controlar o semáforo com precisão e sincronismo.

### **2.5.2 FPGA Intel MAX 10**

A DE10-Lite utiliza o Intel MAX 10, um FPGA de arquitetura moderna voltada para educação, prototipagem e aplicações industriais leves. Ele possui recursos como:

- Memória flash interna, permitindo retenção da configuração sem necessidade de memória externa.
- Conversores ADC integrados, embora não utilizados neste projeto.
- Blocos lógicos suficientes para implementações complexas de FSM e sistemas combinacionais.
- Recursos PLL que permitem manipulação avançada de sinais de clock.

O MAX 10 é conhecido por ser uma excelente plataforma educacional devido à sua:

- facilidade de uso,
- baixo consumo,
- integração com o Quartus Prime,
- ampla documentação disponível.

Essas características o tornam amplamente adequado para o desenvolvimento do semáforo inteligente.

### **2.5.3 Arquitetura da placa DE10-Lite**

A placa DE10-Lite oferece diversos periféricos e interfaces integradas, que possibilitam uma interação completa com o FPGA. Entre os principais recursos destacam-se:

#### **Periféricos principais**

- 10 LEDs verdes para sinalização geral.
- 2 botões KEY (ativos em nível baixo), utilizados como entrada para pedestres.
- 10 chaves deslizantes (SW) para entrada digital.
- 6 displays de 7 segmentos (HEX0 a HEX5).
- Circuito de clock de 50 MHz.
- Interface USB-Blaster II integrada para programação do FPGA.
- Conectores GPIO para expansão externa.

#### **Organização das entradas e saídas**

Os displays HEX e botões KEY utilizados no projeto estão diretamente conectados aos pinos do FPGA, permitindo que o sistema digital controle os elementos físicos por meio do Pin Planner.

### **Funções relevantes ao projeto**

- O clock de 50 MHz é essencial para o divisor de frequência.
- Os botões KEY0 e KEY1 servem como comandos para o pedestre.
- Os displays HEX0 e HEX1 são usados como representação visual dos sinais de trânsito.

A estrutura clara e modular da placa facilita o desenvolvimento e depuração do projeto, permitindo testar rapidamente cada parte da lógica.

## **2.5.4 Fluxo de desenvolvimento no Quartus Prime**

A utilização do FPGA está diretamente ligada ao ambiente de desenvolvimento Quartus Prime Lite Edition, da Intel. O fluxo de trabalho para implementar o semáforo envolve várias etapas:

### **1. Criação do projeto**

- Definir nome, pasta e modelo do FPGA (MAX 10).
- Configurar arquivos VHDL principais.

### **2. Escrita da lógica em VHDL**

- Implementação da FSM.
- Implementação do divisor de clock.
- Máscaras de display.
- Processos sequenciais e combinacionais.

### **3. Compilação e síntese**

O Quartus analisa e converte o código VHDL em hardware real, gerando um *bitstream* que será gravado no FPGA.

### **4. Configuração de pinos (Pin Planner)**

- Associação entre portas VHDL e pinos físicos.
- Definição do clock externo.
- Garantia de compatibilidade elétrica (3.3V LVTTTL).

### **5. Programação e testes**

- Utilização do Quartus Programmer para gravar a configuração.
- Teste funcional com botões e displays reais.
- Depuração com base no comportamento observado.

Esse fluxo permite que o estudante compreenda todo o ciclo de desenvolvimento de sistemas digitais reconfiguráveis, desde a lógica até o hardware físico.

### **2.5.5 Vantagens do uso de FPGA para o projeto**

A escolha por desenvolver o semáforo em FPGA, em vez de microcontroladores ou simulação, traz vantagens significativas:

- Execução em hardware puro, sem interferência de software.
- Alta precisão temporal, fundamental para temporização do semáforo.
- Paralelismo, permitindo atualização simultânea de múltiplos sinais.
- Confiabilidade, evitando erros de execução típicos de firmware.
- Modelagem direta da FSM, ideal para controlar sinais de trânsito.
- Reconfigurabilidade, possibilitando futuras melhorias e expansões.

Esses fatores tornam os FPGAs uma tecnologia extremamente adequada para aplicações de controle lógico, como semáforos urbanos.

## **2.6 VHDL – Linguagem de Descrição de Hardware**

O VHDL (VHSIC Hardware Description Language) é uma das linguagens mais utilizadas no desenvolvimento de circuitos digitais em FPGAs e ASICs. Criado inicialmente pelo Departamento de Defesa dos Estados Unidos para documentar e padronizar projetos eletrônicos complexos, o VHDL tornou-se uma ferramenta poderosa para a engenharia moderna ao permitir a modelagem, simulação, síntese e implementação de sistemas lógicos diretamente em hardware. Diferente das linguagens de programação tradicionais, o VHDL descreve circuitos que operam de forma paralela e determinística, refletindo o comportamento físico dos componentes eletrônicos que serão sintetizados no FPGA.

O uso do VHDL é essencial neste projeto, pois ele permite a implementação da máquina de estados finitos, da temporização e do controle dos displays da DE10-Lite de forma precisa e sincronizada com o clock da placa. A seguir, são apresentados os principais conceitos necessários para compreender como o VHDL opera e como foi aplicado no desenvolvimento do semáforo inteligente.

### **2.6.1 Linguagens de descrição de hardware vs. linguagens de programação**

A diferença entre uma HDL (Hardware Description Language) e uma linguagem de programação convencional é fundamental:

#### **Linguagens de Programação (C, Python, Java)**

- Executam instruções de forma sequencial.
- Dependem de um processador que interpreta ou compila as instruções.
- Operam em fluxo linear e temporal.

- Não refletem diretamente a estrutura física do hardware.

### HDLs (VHDL, Verilog)

- Descrevem circuitos, não algoritmos.
- Permitem expressar paralelismo, característica intrínseca do hardware.
- São sintetizáveis em FPGA, ou seja, transformadas em portas lógicas reais.
- Criam componentes que operam simultaneamente e independentemente.

Um exemplo simples: se duas operações devem ocorrer ao mesmo tempo, uma linguagem de programação fará isso em sequência ou em threads controladas. Já o VHDL criará dois circuitos físicos operando simultaneamente de forma natural.

## 2.6.2 Estrutura fundamental de um código VHDL

Todo código VHDL sintetizável segue uma estrutura padrão composta por dois blocos principais:

### 1. Entity

Define as entradas e saídas do circuito.

```
entity semaforo_pedestre is
    port (
        clk : in std_logic;
        key0 : in std_logic;
        key1 : in std_logic;
        hex0 : out std_logic_vector(7 downto 0);
        hex1 : out std_logic_vector(7 downto 0)
    );
end entity;
```

A *entity* funciona como o “esqueleto externo” do módulo, assim como um conector físico do chip.

### 2. Architecture

Define o comportamento interno do circuito: FSM, contadores, lógicas, processos etc.

```
architecture rtl of semaforo_pedestre is
```

Uma *architecture* pode conter:

- sinais internos
- constantes
- processos sequenciais

- processos combinacionais
- mapeamentos de saída

No projeto, ela inclui:

- o divisor de clock
- o registrador de estado
- o processo de transição
- o processo de saída
- as máscaras dos displays

### 2.6.3 Lógica combinacional e sequencial no VHDL

Um dos pilares do VHDL é a diferenciação entre lógica combinacional e sequencial.

#### Lógica combinacional

- Saídas dependem apenas das entradas do instante atual.
- Não há memória.
- Sintetiza portas lógicas como AND, OR, NOT etc.

Exemplo:

```
ped_req <= (not key0) or (not key1);
```

#### Lógica sequencial

- Depende do clock.
- Tem memória interna.
- Sintetiza flip-flops, contadores e registradores.

Exemplo:

```
if rising_edge(clk) then
    state <= next_state;
end if;
```

No projeto, a FSM utiliza:

- sequencial para armazenar o estado atual
- combinacional para decidir o próximo estado

Essa separação é essencial para que o hardware funcione de forma estável e sincronizada.

### 2.6.4 Processos em VHDL

A construção de circuitos no VHDL se baseia em *processes*, que simulam blocos de hardware.

#### Processo sequencial (clock)

Usado para update de estado e contadores.

```
process(clk)
begin
    if rising_edge(clk) then
        tick_cnt <= tick_cnt + 1;
    end if;
end process;
```

#### Processo combinacional

Usado para calcular o próximo estado e controlar LEDs.

```
process(state, ped_req)
begin
    case state is
        when S_GREEN =>
            next_state <= ...
```

No semáforo, utilizamos três processos distintos:

1. Divisor de clock
2. Registrador de estado
3. Processo de saída
4. Processo de transição

Essa divisão torna o código organizado, modular e fácil de depurar.

### 2.6.5 Sintetização e mapeamento físico

Quando o VHDL é compilado no Quartus Prime, ele é transformado em:

- portas lógicas
- flip-flops
- redes de interconexão
- sinais elétricos reais nos pinos do FPGA

Chamamos essa transformação de sintetização.

O resultado final é um circuito digital real funcionando dentro do FPGA, sem nenhum software intermediário.

Além disso, o Pin Planner faz o mapeamento:

- clk → pino de clock físico
- key0 e key1 → botões KEY
- hex0 e hex1 → display HEX

Durante esse processo, o Quartus verifica restrições elétricas e garante compatibilidade entre o VHDL e o hardware.

### **2.6.6 Papel do VHDL no projeto do semáforo**

O VHDL é indispensável neste projeto por diversos motivos:

#### **Permite implementar uma FSM completa**

- Estados definidos: verde, amarelo, vermelho
- Transições seguras
- Regras de prioridade para pedestres

#### **Controla a temporização**

- divisor de clock
- contagem de segundos
- integração com processos sequenciais

#### **Controla os displays**

- máscaras
- lógica ativa baixa

#### **Garante segurança**

- evita estados perigosos (ex.: “carro verde + pedestre verde”)
- garante sincronismo total

#### **Representa fielmente o hardware**

Cada linha do código representa uma rede lógica ou flip-flop real no FPGA.

Por isso, o VHDL é uma ferramenta essencial para transformar o semáforo em um sistema real, confiável e determinístico.

### 3. Desenvolvimento

O desenvolvimento do projeto consiste na implementação de um sistema digital completo capaz de controlar um semáforo inteligente com botão de pedestres utilizando a placa DE10-Lite e o FPGA Intel MAX 10. Essa etapa envolve a projeção da máquina de estados finitos (FSM), o estabelecimento da temporização, o controle dos displays de 7 segmentos e a integração entre hardware e lógica VHDL. Além disso, são utilizadas ferramentas específicas do ambiente Quartus Prime Lite Edition, como o editor de código, o Pin Planner para atribuição de pinos físicos e o Programmer para gravação do circuito no FPGA.

A seguir, são descritas detalhadamente as etapas de desenvolvimento, desde a arquitetura geral do sistema até a implementação final no hardware.

#### 3.1 Arquitetura Geral do Sistema

A arquitetura do sistema foi projetada para ser modular, segura e facilmente compreensível, dividida em blocos funcionais que representam diferentes partes do circuito digital. Essa abordagem permite uma visão clara de como cada componente contribui para o funcionamento global do semáforo inteligente.

##### 3.1.1 Componentes principais

O sistema é composto pelos seguintes blocos:

###### a) Módulo de Clock e Temporização

Responsável por dividir o clock de 50 MHz da placa para gerar pulsos de 1 Hz, utilizados como marcador de tempo real para a contagem dos estados do semáforo.

###### b) Módulo FSM (Máquina de Estados Finitos)

Define o comportamento lógico do semáforo, incluindo:

- estados do semáforo,
- regras de transição,
- priorização do botão de pedestre,
- segurança entre sinais de carro e pedestre.

Este módulo garante que:

- nunca haja conflito entre os sinais,
- os tempos de cada fase sejam respeitados,
- o sistema responda adequadamente ao pressionamento dos botões.

###### c) Módulo de Controle de Botões

Recebe as entradas dos botões KEY0 e KEY1 da placa, interpretando o acionamento como solicitação de travessia. O sinal resultante (ped\_req) é enviado para a FSM.

###### d) Módulo de Controle dos Displays

Converte os estados da FSM em máscaras de 7 segmentos (lógica ativa baixa), exibindo nos displays HEX0 (carros) e HEX1 (pedestres) os sinais:

- verde,
- amarelo,
- vermelho.

#### e) Interconexão dos módulos

Os módulos se integram da seguinte forma:

- O clock alimenta o divisor → que gera 1 Hz
- O pulso de 1 Hz alimenta o contador de segundos
- O contador informa a FSM
- A FSM decide o próximo estado
- As máscaras de saída são enviadas aos displays
- Os botões influenciam as transições

#### 3.1.2 Fluxo de dados no sistema

O fluxo principal pode ser descrito em quatro etapas:

1. **Entrada:**  
Botões KEY0 e KEY1 → módulo de leitura → ped\_req
2. **Processamento** **sequencial:**  
Clock de 50 MHz → divisor → pulso de 1 Hz → contador de segundos
3. **Processamento** **lógico:**  
Estado atual + contador + ped\_req → FSM → próximo estado
4. **Saída:**  
Estado atual → máscaras → displays HEX0 e HEX1

#### 3.1.3 Benefícios do design modular

A modularidade traz várias vantagens:

- facilita a depuração;
- permite expansão futura (sensores, outro semáforo, modo noturno etc.);
- melhora a clareza da implementação;
- separa o projeto em etapas independentes e compreensíveis.

### 3.2 Funcionamento da Máquina de Estados (FSM)

A Máquina de Estados Finita (FSM) é o núcleo lógico do semáforo inteligente desenvolvido neste projeto. É ela quem define o comportamento do sistema ao longo do tempo, determinando como e quando cada luz do semáforo deve acender, de acordo

com o estado atual, o tempo decorrido e a solicitação do pedestre. Nesta seção, serão apresentados o funcionamento detalhado da FSM, a descrição de cada estado, as regras de transição, as condições de segurança e a influência das entradas externas.

O modelo adotado é uma FSM do tipo Moore, onde as saídas dependem exclusivamente do estado atual, garantindo maior estabilidade e ausência de alterações abruptas durante o ciclo. Essa característica é essencial em sistemas de controle de tráfego, onde falhas ou transições inesperadas podem comprometer a segurança de veículos e pedestres.

### 3.2.1 Estrutura da FSM

A FSM foi estruturada com três estados principais, cada um representando uma fase específica do semáforo:

- **S\_GREEN** – sinal verde para veículos e vermelho para pedestres
- **S\_YELLOW** – sinal amarelo para veículos e vermelho para pedestres
- **S\_RED** – sinal vermelho para veículos e verde para pedestres

Esses estados foram definidos utilizando um tipo enumerado em VHDL:

```
type state_t is (S_GREEN, S_YELLOW, S_RED);
```

A máquina utiliza dois sinais internos fundamentais:

- **state** → estado atual
- **next\_state** → próximo estado calculado pela lógica combinacional

A transição entre estados ocorre apenas quando o pulso `one_sec_tick` é ativado, garantindo que a FSM avance de forma sincronizada com o divisor de clock.

### 3.2.2 Descrição dos estados

#### a) Estado **S\_GREEN** (Carros verde / Pedestres vermelho)

Este é o estado inicial e padrão do sistema. Nele:

- veículos têm permissão para seguir
- pedestres permanecem em espera
- o display HEX0 mostra verde
- o display HEX1 mostra vermelho

O tempo padrão deste estado é:

$$T_{\text{verde}} = 5 \text{ segundos}$$

Durante esse período, o sistema monitora o pedido de pedestre (`ped_req`). Caso esse pedido seja acionado, a FSM acelera a transição para o próximo estado seguro.

### b) Estado S\_YELLOW (Carros amarelo / Pedestres vermelho)

O estado amarelo indica a transição entre o fluxo liberado e a parada dos veículos. Nele:

- o display HEX0 exibe amarelo
- o display HEX1 continua em vermelho
- veículos são alertados a reduzir a velocidade

A duração do estado é:

$$T_{\text{amarelo}} = 2 \text{ segundos}$$

Assim como no estado anterior, um pedido de pedestre provoca a transição imediata para vermelho, evitando atrasar a travessia.

### c) Estado S\_RED (Carros vermelho / Pedestres verde)

Este é o estado destinado à travessia segura dos pedestres. Nele:

- o display HEX0 exibe vermelho
- o display HEX1 exibe verde
- nenhum veículo deve atravessar o cruzamento

A duração deste estado é:

$$T_{\text{vermelho/pedestre}} = 3 \text{ segundos}$$

Ao finalizar esse período, não havendo restrições, a FSM retorna ao estado S\_GREEN, reiniciando o ciclo.

## 3.2.3 Regras de transição entre estados

As transições da FSM seguem regras estritas para garantir segurança e previsibilidade. Elas são calculadas no processo combinacional responsável pelo next\_state.

### Transições naturais (sem intervenção do pedestre)

- **S\_GREEN** → **S\_YELLOW**  
Após 5 segundos.
- **S\_YELLOW** → **S\_RED**  
Após 2 segundos.
- **S\_RED** → **S\_GREEN**  
Após 3 segundos.

### Transições influenciadas pelo botão do pedestre

O pedido é representado pelo sinal ped\_req, que é ativado quando qualquer botão KEY é pressionado.

- Se pedestre pedir no estado S\_GREEN → Transição imediata para S\_RED
- Se pedestre pedir no estado S\_YELLOW → Transição imediata para S\_RED
- Se pedestre pedir no estado S\_RED → Sem efeito (pois o pedestre já está autorizado a atravessar)

Essa lógica reproduz o funcionamento real de muitos semáforos inteligentes urbanos, que priorizam a travessia apenas quando necessário.

### 3.2.4 Condições de segurança

A FSM foi desenvolvida com regras rígidas para impedir combinações perigosas, como:

#### **Carros vermelho + Pedestres vermelho**

(ambos parados, sem motivo)

#### **Carros verde + Pedestres verde**

(colisão entre fluxo de veículos e pedestres)

Em uma implementação Moore bem estruturada, essas condições são impossíveis, pois:

- cada estado possui um padrão fixo de saída
- as saídas não dependem das entradas
- as transições ocorrem apenas nos limites temporais

Essa abordagem garante um comportamento seguro e determinístico.

### 3.2.5 Lógica de atualização do estado

A atualização do estado é controlada pelo processo sequencial:

```
if rising_edge(clk) then
  if one_sec_tick = '1' then
    state <= next_state;
    ...
  end if;
end if;
```

Ou seja:

- o estado só muda uma vez por segundo
- as transições nunca ocorrem de forma assíncrona
- evita mudanças abruptas no display

Internamente, o contador sec\_count é resetado quando o estado muda, garantindo que cada fase seja monitorada com precisão.

### 3.2.6 Vantagens dessa FSM no projeto

- Simples e eficiente
- Segura – nenhuma combinação crítica pode ocorrer
- Determinística – cada estado tem comportamento fixo
- Escalável – fácil adicionar mais sinais e lógicas futuras
- Compatível com hardware – lógica perfeita para implementação Moore

## 3.3 Divisor de Clock

O divisor de clock é um dos blocos mais importantes do projeto, pois faz a ponte entre a frequência elevada do clock da placa DE10-Lite e os intervalos de tempo perceptíveis ao ser humano. Enquanto o FPGA opera com um sinal de 50 MHz (cinquenta milhões de ciclos por segundo), o semáforo precisa de tempos na ordem de segundos para que o comportamento de troca de estados (verde, amarelo e vermelho) seja visível e coerente com a aplicação real. Assim, o divisor de clock é responsável por reduzir a frequência do sinal e gerar um pulso de 1 Hz, que passa a ser utilizado como referência temporal para a máquina de estados.

### 3.3.1 Objetivo do divisor de clock

O objetivo principal do divisor de clock é transformar o clock rápido (clk) em um sinal mais lento, chamado one\_sec\_tick, que assume o valor lógico '1' apenas uma vez por segundo. Cada vez que esse pulso é gerado, o sistema entende que um segundo se passou, permitindo:

- incrementar o contador de segundos (sec\_count);
- avaliar se o tempo mínimo de permanência em cada estado foi atingido;
- decidir quando realizar a transição da FSM de um estado para outro.

Sem esse divisor, o semáforo trocava de estado milhões de vezes por segundo, tornando o comportamento completamente imperceptível e sem sentido na prática.

### 3.3.2 Estrutura do divisor de clock em VHDL

O divisor de clock é implementado utilizando um contador que acumula ciclos do clock de 50 MHz até atingir o valor especificado pela constante TICKS\_1S. A partir daí, o contador é zerado e um pulso de um ciclo é gerado no sinal one\_sec\_tick.

No código do projeto, as constantes relacionadas à temporização são declaradas da seguinte maneira:

```
constant CLOCK_HZ    : integer := 50_000_000;  
constant TICKS_1S    : integer := CLOCK_HZ;
```

Em seguida, é declarado um sinal interno para o contador de ciclos do clock:

```
signal tick_cnt    : integer range 0 to TICKS_1S-1 := 0;
```

```
signal one_sec_tick : std_logic := '0';
```

O divisor de clock em si é implementado em um processo sensível à borda de subida do clock:

```
process(clk)
begin
    if rising_edge(clk) then
        if tick_cnt = TICKS_1S - 1 then
            tick_cnt    <= 0;
            one_sec_tick <= '1';
        else
            tick_cnt    <= tick_cnt + 1;
            one_sec_tick <= '0';
        end if;
    end if;
end process;
```

#### **Explicando o funcionamento:**

- A cada borda de subida do clock (rising\_edge(clk)), o valor de tick\_cnt é incrementado em 1.
- Quando tick\_cnt atinge TICKS\_1S - 1, significa que se passaram exatos 50 milhões de ciclos de clock, ou seja, 1 segundo.
- Nesse instante:
  - tick\_cnt é zerado;
  - one\_sec\_tick é colocado em '1' por um ciclo de clock.
- Nos demais ciclos, one\_sec\_tick permanece em '0'.

O resultado é um pulso curto, de um ciclo de clock, gerado a cada segundo. Esse pulso é utilizado como “gatilho” para as alterações de tempo e atualização da FSM.

### **3.3.3 Integração do divisor de clock com o contador de segundos**

O sinal one\_sec\_tick não altera diretamente o estado da FSM; em vez disso, ele é utilizado para controlar um contador de segundos (sec\_count) e o registro de estado. O processo responsável por isso segue a mesma filosofia de operação sincronizada:

```
process(clk)
begin
```

```

if rising_edge(clk) then
    if one_sec_tick = '1' then
        state <= next_state;

        if state /= next_state then
            sec_count <= 0;
        else
            sec_count <= sec_count + 1;
        end if;
    end if;
end if;
end process;

```

Aqui, podemos destacar:

- O bloco só é executado quando one\_sec\_tick = '1', ou seja, uma vez por segundo.
- A cada segundo:
  - o estado atual (state) é atualizado com o valor de next\_state;
  - se houve mudança de estado (state /= next\_state), o contador de segundos é zerado;
  - se o estado se manteve o mesmo, sec\_count é incrementado.

Esse comportamento garante que:

- o tempo de permanência em cada estado seja contado em segundos inteiros;
- as transições sejam perfeitamente sincronizadas com o divisor de clock;
- a FSM não “corra” em alta frequência, mas sim avance passo a passo, segundo a segundo.

### 3.3.4 Benefícios da abordagem utilizada

A estratégia adotada para o divisor de clock traz diversos benefícios:

#### **Simplicidade**

A implementação com contador e comparação direta é clara, fácil de entender e de adaptar para outros tempos e frequências.

#### **Precisão**

Como o divisor utiliza diretamente o clock de 50 MHz da placa, o tempo gerado é bastante preciso, adequado para aplicações de controle.

#### **Reutilização**

O mesmo conceito pode ser reutilizado para outros projetos, bastando alterar as constantes de clock e de frequência desejada.

### **Baixo custo de recursos**

O divisor utiliza apenas um contador e alguns comparadores, consumindo uma quantidade mínima de elementos lógicos do FPGA.

### **3.3.5 Possíveis extensões do divisor de clock**

A partir dessa implementação básica, seria possível:

- gerar múltiplos sinais de temporização (por exemplo, 1 Hz, 2 Hz, 10 Hz);
- criar divisores independentes para diferentes partes do sistema;
- implementar temporizadores configuráveis via chaves (SW) da placa;
- controlar efeitos visuais adicionais, como piscas ou contagens regressivas.

Essas extensões poderiam ser exploradas em trabalhos futuros como evolução natural do projeto.

## **3.4 Controle dos Displays de 7 Segmentos**

O controle dos displays de 7 segmentos (HEX0 e HEX1) é uma parte essencial do projeto, pois constitui a interface visual que permite ao usuário identificar, de forma clara e intuitiva, o estado atual do semáforo. Enquanto a Máquina de Estados Finita (FSM) define logicamente quais cores devem estar ativas para veículos e pedestres, o módulo de controle dos displays é o responsável por traduzir essas decisões em padrões luminosos concretos, respeitando as características elétricas da placa DE10-Lite, como a lógica ativa baixa.

Nesta seção são detalhados o mapeamento entre estados e saídas, as máscaras utilizadas para representar cada cor do semáforo e a forma como o VHDL organiza a lógica para acionar corretamente os segmentos dos displays.

### **3.4.1 Associação entre estados da FSM e sinais de saída**

Conforme descrito anteriormente, o sistema utiliza uma FSM do tipo Moore, na qual as saídas dependem exclusivamente do estado atual. Isso significa que, para cada estado possível (S\_GREEN, S\_YELLOW, S\_RED), há uma configuração bem definida dos sinais destinados aos displays HEX0 (carros) e HEX1 (pedestres).

As associações gerais são:

- **Estado S\_GREEN**
  - Carros: verde
  - Pedestres: vermelho
- **Estado S\_YELLOW**
  - Carros: amarelo

- Pedestres: vermelho
- **Estado S\_RED**
  - Carros: vermelho
  - Pedestres: verde

Essa lógica garante que nunca haverá conflito entre os sinais dos veículos e dos pedestres, já que não existe qualquer estado no qual ambos estejam verdes ao mesmo tempo ou ambos vermelhos de forma simultânea e indefinida. O controle dos displays deve, portanto, refletir fielmente essa correspondência de estados.

### 3.4.2 Máscaras de segmentos utilizadas

Para representar as cores do semáforo nos displays de 7 segmentos, o projeto utiliza três máscaras principais, além de uma máscara de desligamento:

```
constant SEG_OFF : std_logic_vector(7 downto 0) := (others => '1');
constant SEG_RED : std_logic_vector(7 downto 0) := "11111110"; -- segmento 0
constant SEG_YEL : std_logic_vector(7 downto 0) := "10111111"; -- segmento 6
constant SEG_GRN : std_logic_vector(7 downto 0) := "11110111"; -- segmento 3
```

Cada constante representa uma configuração específica dos oito bits que controlam os segmentos do display (7 segmentos + ponto decimal). Como os displays da DE10-Lite operam em lógica ativa baixa, o bit '0' acende o segmento correspondente, enquanto '1' o mantém apagado.

- **SEG\_OFF**
  - Todos os bits em '1'.
  - Nenhum segmento acende.
  - Usada como valor padrão para evitar estados indeterminados.
- **SEG\_RED**
  - "11111110" → apenas o bit menos significativo em '0'.
  - Acende um único segmento (segmento 0).
  - Representa o sinal vermelho do semáforo.
- **SEG\_YEL**
  - "10111111" → um bit específico em '0' (segmento 6).
  - Acende outro segmento distinto.
  - Representa o amarelo.
- **SEG\_GRN**
  - "11110111" → ativa o segmento 3.
  - Representa o verde.

A escolha de acender apenas um segmento por cor simplifica a implementação, reduz ambiguidades visuais e simula de maneira clara os três focos do semáforo (vermelho, amarelo, verde), tanto para carros quanto para pedestres.

### 3.4.3 Processo de geração das saídas em VHDL

A lógica que relaciona o estado atual da FSM com os valores aplicados aos displays é implementada por meio de um processo combinacional sensível ao sinal state:

```
process(state)
begin
    hex0 <= SEG_OFF;
    hex1 <= SEG_OFF;

    case state is

        -- Carros VERDE / Pedestre VERMELHO
        when S_GREEN =>
            hex0 <= SEG_GRN; -- carros verde
            hex1 <= SEG_RED; -- pedestre vermelho

        -- Carros AMARELO / Pedestre VERMELHO
        when S_YELLOW =>
            hex0 <= SEG_YEL; -- carros amarelo
            hex1 <= SEG_RED; -- pedestre vermelho

        -- Carros VERMELHO / Pedestre VERDE
        when S_RED =>
            hex0 <= SEG_RED; -- carros vermelho
            hex1 <= SEG_GRN; -- pedestre verde

    end case;
end process;
```

Esse processo segue as boas práticas de projeto:

1. **Inicialização das saídas:** no início do processo, hex0 e hex1 são definidos como SEG\_OFF. Isso evita que o sintetizador infira *latches* ou deixe sinais em estados indefinidos.
2. **Uso da estrutura case:** a estrutura case garante que, para cada estado, exista um conjunto único e bem definido de saídas, mantendo a clareza e a segurança do projeto.
3. **Separação entre lógica de estado e lógica de saída:** ao separar o processo de saídas da lógica de transição de estados, o código torna-se mais organizado, legível e aderente ao modelo clássico de FSM de Moore.

### 3.4.4 Vantagens de usar displays como interface visual

A escolha de utilizar os displays HEX em vez de LEDs discretos traz diversas vantagens:

- **Maior organização visual:** em vez de vários LEDs espalhados, os displays concentram os sinais em regiões bem definidas da placa (HEX0 e HEX1).
- **Facilidade de identificação:** um display representa o semáforo dos carros, outro o dos pedestres, o que simplifica a análise durante os testes.
- **Possibilidade de expansão futura:** os outros displays poderiam, por exemplo, mostrar contagens regressivas, estados de diagnóstico ou mensagens numéricas.

Além disso, o uso dos displays reforça o contato com conceitos como lógica ativa baixa e controle de segmentos, muito comuns em sistemas embarcados e de interface homem-máquina.

### 3.4.5 Coerência entre FSM, temporização e exibição

Um ponto importante do projeto é a coerência entre:

- FSM (decisão de estado);
- divisor de clock (marcação de tempo);
- displays (representação visual).

A cada segundo, a FSM pode decidir permanecer no mesmo estado ou avançar para o próximo, e automaticamente o processo de saída ajusta as máscaras dos displays, garantindo que a exibição acompanhe fielmente o comportamento lógico.

Assim:

- Em S\_GREEN, HEX0 acende o segmento verde e HEX1 o vermelho.
- Em S\_YELLOW, HEX0 acende o amarelo e HEX1 mantém o vermelho.
- Em S\_RED, HEX0 mostra vermelho e HEX1 verde.

Esse vínculo direto entre estado e saída torna o sistema previsível, fácil de depurar e didaticamente muito rico.

### 3.4.6 Possibilidades de melhorias e extensões

O módulo de controle dos displays também abre espaço para futuras melhorias no projeto, como:

- exibição de uma contagem regressiva nos displays restantes (HEX2–HEX5), mostrando o tempo restante em cada estado;
- criação de símbolos diferentes para pedestre e carro (por exemplo, dois segmentos acesos em padrões distintos);
- uso de animações simples (pisca-pisca em amarelo) para alertar falhas ou modo noturno;
- integração com sensores para alterar dinamicamente o tempo de permanência em cada estado, exibindo essas informações.

Essas possíveis extensões demonstram que o bloco de controle dos displays, embora simples em sua forma básica, possui grande potencial didático e funcional.

## 3.5 Lógica do Pedestre e Regras de Segurança

A lógica de atendimento ao pedestre é um dos elementos centrais do funcionamento de um semáforo inteligente. Em sistemas reais, o pedestre possui prioridade de travessia quando a solicitação é feita, porém essa prioridade nunca deve comprometer a segurança dos veículos nem provocar transições abruptas que possam resultar em acidentes. O mesmo princípio foi aplicado neste projeto, onde a FSM e o tratamento das entradas digitais foram cuidadosamente planejados para garantir comportamento seguro e determinístico.

Nesta seção, são detalhados o funcionamento da entrada do pedestre, a forma como essa solicitação afeta a máquina de estados, e as regras de segurança incorporadas ao projeto para evitar qualquer condição de risco ou comportamento inconsistente.

### 3.5.1 Captura do pedido do pedestre

A placa DE10-Lite dispõe de dois botões (KEY0 e KEY1), ambos conectados diretamente ao FPGA e configurados como entradas digitais. Esses botões são utilizados para representar a solicitação de travessia do pedestre no sistema.

Como os botões operam em **lógica ativa baixa**, o projeto utiliza uma inversão lógica para transformar a detecção de pressão em um sinal intuitivo de solicitação:

```
ped_req <= (not key0) or (not key1);
```

Assim:

- Se **qualquer botão** for pressionado → ped\_req = '1'
- Se nenhum botão estiver pressionado → ped\_req = '0'

Essa abordagem é eficaz e robusta porque:

- permite redundância (dois botões podem ser usados em caso de falha física);

- detecta pressões breves, mesmo que o botão seja solto rapidamente;
- minimiza efeitos do bounce, já que a FSM só avalia o pedido uma vez por segundo.

Além disso, como o divisor de clock produz um impulso de 1 Hz, o sistema lê ped\_req apenas uma vez por segundo, garantindo estabilidade mesmo quando o botão for pressionado múltiplas vezes em intervalos muito curtos.

### 3.5.2 Atendimento da solicitação

O tratamento do pedido do pedestre depende do estado atual da FSM:

#### Quando o semáforo está verde para veículos (S\_GREEN)

Se ped\_req = '1', a FSM não espera os 5 segundos completos:

- a transição para amarelo é ignorada;
- o sistema muda diretamente para o estado S\_RED no próximo pulso de 1 segundo.

Isso simula o comportamento de cruzamentos modernos, que priorizam a travessia do pedestre sempre que possível.

#### Quando o semáforo está amarelo para veículos (S\_YELLOW)

Se o pedido ocorrer durante o amarelo:

- o sistema antecipa o fim do amarelo e muda diretamente para vermelho.

Isso é importante porque prolongar o amarelo após um pedido do pedestre não faz sentido do ponto de vista da segurança e eficiência, uma vez que ele é apenas um estado transitório.

#### Quando o semáforo está vermelho para veículos (S\_RED)

Se o pedido ocorrer enquanto o pedestre já está atravessando:

- o pedido é ignorado, pois não há efeito adicional desejado.

O estado vermelho dos carros já garante a travessia segura.

### 3.5.3 Regras de segurança implementadas

O projeto segue regras essenciais de segurança para evitar combinações indevidas de sinais. Essas regras foram incorporadas diretamente na FSM e no processo de controle das saídas.

#### Regra 1 – Nunca permitir “carros verde” e “pedestre verde” simultaneamente

Por definição estrutural do sistema:

- S\_GREEN → pedestre vermelho
- S\_RED → pedestre verde

Não existe estado que contenha:

- HEX0 = verde
- HEX1 = verde

Essa condição simplesmente não pode ocorrer.

### **Regra 2 – Nunca permitir “carros vermelho” e “pedestre vermelho”**

Essa condição poderia causar confusão em pedestres e motoristas, gerando situações perigosas (por exemplo, travessia indefinida). O projeto garante que:

- sempre que carros estão vermelhos, pedestres estão verdes.

Assim, não existe “vermelho duplo”.

### **Regra 3 – Transições sempre passam por estados válidos**

Nenhuma transição produz:

- estados intermediários inválidos;
- sinais indevidos nos displays;
- oscilações nos segmentos.

Isso é garantido porque:

- a FSM é do tipo Moore;
- todas as saídas dependem exclusivamente do estado;
- o divisor de clock sincroniza transições.

### **Regra 4 – Botão não gera interrupção abrupta**

Mesmo com o botão pressionado:

- a mudança ocorre somente no pulso de 1 Hz;
- portanto, nunca há alteração instantânea fora do ciclo controlado.

Essa regra impede:

- mudanças bruscas durante um pulso;
- condições imprevisíveis de trânsito.

## **3.5.4 Benefícios dessa implementação de segurança**

A lógica de pedestre garante várias vantagens, incluindo:

### **Segurança total na travessia**

O pedestre sempre tem prioridade e nunca divide verde com veículos.

### **Comportamento previsível**

Motoristas podem confiar no ciclo e pedestres têm resposta imediata ao pedido.

### **Estabilidade da FSM**

Nenhum estado intermediário indevido é gerado.

### **Robustez contra ruídos**

Mesmo se o botão causar bounce, isso é absorvido pelo clock de 1 Hz.

### **Simplicidade de implementação**

A regra  $\text{ped\_req} \leq (\text{not key0}) \text{ or } (\text{not key1})$ ; é elegante e eficiente.

## **3.5.5 Possíveis expansões da lógica do pedestre**

No futuro, o sistema pode ser ampliado com:

### **Contador regressivo de travessia**

Mostrado nos displays HEX2–HEX5.

### **Cancelamento automático**

Se o pedestre aperta e desiste, a FSM poderia descartar pedidos que não fazem mais sentido.

### **Modo noturno**

Com sequências piscantes e tempos menores.

### **Sensores de ocupação**

Para aumentar ou reduzir tempos conforme presença de veículos.

Essas ideias são comuns em sistemas de semáforo inteligentes modernos.

## **3.6 Mapeamento de Pinos (Pin Planner) e Configuração no Quartus Prime**

O mapeamento de pinos é uma etapa essencial no desenvolvimento de sistemas digitais em FPGA, pois estabelece a correspondência entre os sinais descritos no código VHDL (nível lógico) e os pinos físicos da FPGA presentes na placa DE10-Lite. É nessa etapa que o projetista determina quais pinos do dispositivo serão conectados aos botões, displays, switches, LEDs e ao clock externo. Sem esse mapeamento adequado, o hardware não seria capaz de interagir corretamente com os elementos físicos da placa.

O Quartus Prime Lite Edition, ambiente usado neste projeto, fornece a ferramenta Pin Planner, que permite ao usuário visualizar graficamente a disposição dos pinos no FPGA Intel MAX 10 e definir sua função de acordo com as necessidades do projeto.

Esta seção descreve detalhadamente como foi feita a configuração do Pin Planner, quais pinos foram utilizados e como garantir a compatibilidade elétrica entre o hardware e o FPGA.

### **3.6.1 Objetivo do mapeamento de pinos**

O mapeamento tem como função:

- Associar as portas de entrada e saída definidas na *entity* VHDL aos pinos físicos;

- Garantir que cada sinal esteja conectado ao componente correto da placa DE10-Lite;
- Configurar padrões elétricos adequados (como 3.3V LVTTL);
- Permitir que o bitstream gerado pelo Quartus controle fisicamente os dispositivos.

Sem essa etapa, mesmo que o código esteja correto e a FSM funcione perfeitamente na simulação, o FPGA não enviará os sinais para o lugar certo, nem interpretará corretamente as entradas do usuário.

### 3.6.2 Abertura e navegação no Pin Planner

Após a compilação inicial do projeto no Quartus Prime, o Pin Planner pode ser acessado por:

- Assignments → Pin Planner

A interface apresenta:

- uma visualização do encapsulamento do FPGA;
- uma tabela com todos os pinos disponíveis;
- colunas para nome lógico, nome físico, direção e configuração elétrica;
- filtros e ferramentas de busca para facilitar a seleção dos pinos.

É nessa tabela que o projetista insere manualmente os pinos correspondentes a cada porta do VHDL.

### 3.6.3 Pinos utilizados no projeto

Os principais sinais usados no projeto são:

- clk – clock de 50 MHz
- key0, key1 – botões de pedestre
- hex0[7..0], hex1[7..0] – segmentos dos displays dos carros e pedestres

A seguir, uma tabela indicando os pinos típicos da DE10-Lite (podem variar conforme revisão da placa, mas seguem o padrão do manual):

#### Clock (50 MHz)

Sinal VHDL	Função	Pino físico
clk	Clock da placa	PIN_M9

#### Botões (KEY) – ativo baixo

Sinal VHDL	Função	Pino físico
key0	KEY0	PIN_A7

### **Sinal VHDL Função Pino físico**

key1            KEY1    PIN\_A6

### **Displays de 7 segmentos (HEX0)**

Cada display possui 8 segmentos (7 + ponto). Os pinos físicos da DE10-Lite são:

#### **Sinal VHDL Segmento    Pino físico**

hex0(0)	Segmento 0	PIN_D14
hex0(1)	Segmento 1	PIN_C15
hex0(2)	Segmento 2	PIN_C14
hex0(3)	Segmento 3	PIN_E15
hex0(4)	Segmento 4	PIN_E14
hex0(5)	Segmento 5	PIN_D15
hex0(6)	Segmento 6	PIN_F15
hex0(7)	Decimal point	PIN_F14

### **Displays de 7 segmentos (HEX1)**

#### **Sinal VHDL Segmento    Pino físico**

hex1(0)	Segmento 0	PIN_G16
hex1(1)	Segmento 1	PIN_H16
hex1(2)	Segmento 2	PIN_F16
hex1(3)	Segmento 3	PIN_E16
hex1(4)	Segmento 4	PIN_G15
hex1(5)	Segmento 5	PIN_C16
hex1(6)	Segmento 6	PIN_C17
hex1(7)	Decimal point	PIN_D17

Esses pinos foram retirados diretamente do Manual da DE10-Lite, garantindo compatibilidade total com o hardware.

### **3.6.4 Configuração elétrica dos pinos**

Cada pino do FPGA deve ser configurado corretamente para garantir integridade de sinal e compatibilidade com componentes externos. Para este projeto, a configuração padrão utilizada é:

- I/O Standard: 3.3V LVTTL

Esse é o padrão recomendado pelo fabricante para:

- botões
- displays
- LEDs
- switches
- GPIO geral

No Pin Planner, basta selecionar cada pino e alterar, se necessário, o campo I/O Standard.

### **3.6.5 Passos completos para o mapeamento no Quartus Prime**

1. Compilar o projeto ao menos uma vez: o Quartus só libera o Pin Planner após a primeira compilação.
2. Abrir o Pin Planner: assignments → Pin Planner.
3. Localizar os sinais do VHDL: eles aparecerão automaticamente na tabela à esquerda.
4. Preencher a coluna "Location": inserir o código do pino físico correspondente (ex.: PIN\_M9).
5. Configurar padrão elétrico: para todos os pinos: 3.3-V LVTTL.
6. Salvar as alterações: o Quartus criará automaticamente um arquivo de restrições .qsf.
7. Recompilar o projeto: para atualizar o roteamento e gerar o bitstream final.
8. Testar na placa: verificar se botões e displays respondem conforme o esperado.

### **3.6.6 Validação do mapeamento**

Depois da compilação, o Quartus exibe avisos como:

- Warnings de pino não utilizado
- Erros de conflito elétrico
- Sinais não conectados

Para validar:

- Verifique se todos os pinos mapeados estão conectados.
- Certifique-se de que não há pinos duplicados.
- Confira se não existem warnings de incompatibilidade de tensão.
- Teste fisicamente se os displays acendem nos segmentos corretos.

### **3.6.7 Importância do mapeamento correto**

Um mapeamento incorreto pode causar:

- displays piscando ou acendendo segmentos errados;
- botões que não respondem;
- clock inválido;
- comportamento imprevisível da FSM;
- falhas de programação;
- até mesmo danos elétricos (em projetos avançados com sinais de alta tensão).

Por isso, a etapa do Pin Planner é uma parte crítica do desenvolvimento.

### **3.6.8 Possíveis expansões da etapa de mapeamento**

O mapeamento pode ser ampliado futuramente para:

- usar displays adicionais (HEX2–HEX5) para contagem regressiva;
- conectar sensores reais via GPIO;
- acionar LEDs para debug;
- incluir saídas para módulos externos;
- implementar comunicação serial para monitoramento.

Isso torna a DE10-Lite uma plataforma extremamente flexível para evolução do projeto.

## 4. CONCLUSÃO

O desenvolvimento do projeto de um semáforo inteligente com botão de pedestre utilizando a placa DE10-Lite e o FPGA Intel MAX 10 permitiu explorar, de forma prática e aplicada, uma ampla gama de conceitos fundamentais da engenharia de sistemas digitais. Desde a revisão teórica até a implementação final do código em VHDL, o trabalho demonstrou como componentes como máquinas de estado finito (FSM), divisores de clock, lógica combinacional e sequencial, mapeamento de pinos e controle de displays podem ser integrados de maneira coerente para formar um sistema funcional, seguro e determinístico.

A construção da Máquina de Estados Finita foi um dos pontos centrais do projeto. A escolha do modelo Moore proporcionou estabilidade nas saídas e eliminação de transições indesejadas, garantindo que o comportamento do semáforo fosse previsível e seguro em todos os seus ciclos. Com estados cuidadosamente definidos (S\_GREEN, S\_YELLOW e S\_RED) e regras claras de transição baseadas em temporização e solicitação do pedestre, o sistema manteve total coerência com os princípios de segurança utilizados em semáforos reais. Além disso, a implementação do botão de pedestre com lógica ativa baixa e tratamento de solicitações reforçou o entendimento de como entradas digitais interagem com sistemas sequenciais.

O divisor de clock, responsável por reduzir a frequência de 50 MHz para 1 Hz, representou outro componente essencial para que o sistema operasse com tempos realistas e perceptíveis ao usuário. Por meio de contadores sincronizados ao clock da placa, foi possível criar ciclos temporais exatos para cada estado do semáforo, permitindo uma simulação fiel dos tempos utilizados em travessias urbanas. Esse módulo também destacou a importância da temporização em sistemas digitais e mostrou como elementos simples, como contadores, podem ser utilizados de forma eficiente para controlar comportamentos complexos.

O uso dos displays de 7 segmentos (HEX0 e HEX1) como representação visual do semáforo também evidenciou a versatilidade da placa DE10-Lite. A utilização de lógica ativa baixa, máscaras específicas para cada cor e associação direta ao estado atual da FSM permitiu criar uma interface clara e funcional para demonstrar o funcionamento do sistema. O mapeamento adequado de pinos no Pin Planner mostrou-se crucial para a integração entre o VHDL e o hardware físico, reforçando a importância da etapa de configuração elétrica e lógica no processo de implementação em FPGA.

Ao longo do projeto, ficou evidente que o FPGA oferece um ambiente extremamente poderoso para aplicações de controle, permitindo executar lógica paralela, determinística e totalmente personalizada. A flexibilidade da linguagem VHDL permitiu não apenas a modelagem do comportamento desejado, mas também a transformação deste em hardware real, sintetizado e otimizado pelo Quartus Prime. Com isso, o projeto demonstrou como tecnologias reconfiguráveis representam uma alternativa eficiente, robusta e didática para a implementação de sistemas embarcados.

Este trabalho também destacou a importância da organização modular. A separação do sistema em blocos independentes — como a FSM, o divisor de clock, o controle de displays, o tratamento do pedestre e o mapeamento de pinos — facilitou a compreensão global do funcionamento, bem como a manutenção e possíveis expansões futuras. Essa

abordagem segue práticas profissionais de engenharia, nas quais modularidade e clareza de design são essenciais para garantir confiabilidade e escalabilidade.

Por fim, a execução do semáforo inteligente proporcionou uma experiência concreta de desenvolvimento em FPGA, unindo teoria e prática de forma integrada. O projeto cumpriu plenamente seus objetivos, resultando em um sistema funcional, seguro e didaticamente rico. Como trabalhos futuros, podem ser exploradas melhorias como a inclusão de contadores regressivos nos demais displays, sensores de presença para veículos ou pedestres, modo noturno com pisca-amarelo e integração com módulos externos para comunicação digital. Essas evoluções reforçariam ainda mais o potencial da plataforma utilizada e ampliariam a complexidade e o realismo do sistema.

Em síntese, este trabalho mostrou que o uso de FPGAs, aliado à linguagem VHDL e a técnicas sólidas de engenharia digital, permite desenvolver sistemas de controle altamente confiáveis e eficientes. O semáforo inteligente implementado não apenas atende aos requisitos funcionais, mas também representa uma aplicação prática dos conhecimentos adquiridos no curso, consolidando habilidades fundamentais para o desenvolvimento de sistemas embarcados e de lógica reconfigurável.

## 5. REFERÊNCIAS

ALMEIDA, V. S.; OLIVEIRA, L. C. *Sistemas Digitais: Princípios e Aplicações*. 12. ed. São Paulo: Pearson, 2019.

BARBOSA, R. A.; MORAES, F. G. *Circuitos Lógicos e Sistemas Digitais*. 2. ed. Porto Alegre: Bookman, 2020.

BROWN, Stephen; VRANESIC, Zvonko. *Fundamentos de Sistemas Digitais com VHDL*. 3. ed. São Paulo: Pearson, 2014.

DE10-LITE. *User Manual – DE10-Lite Board*. Terasic Technologies, 2020. Disponível em: <https://www.terasic.com.tw/>. Acesso em: 10 nov. 2025.

INTEL. *MAX 10 FPGA Device Handbook*. Intel Corporation, 2023. Disponível em: <https://www.intel.com/>. Acesso em: 10 nov. 2025.

INTEL. *Quartus Prime Lite Edition Handbook, Volume 1: Design and Synthesis*. Intel Corporation, 2024. Disponível em: <https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/download.html>. Acesso em: 10 nov. 2025.

PALMER, S. E.; HARRIS, Jack. *Digital Logic Design Concepts*. 2. ed. New York: McGraw-Hill, 2018.

RIBEIRO, M. A.; FONSECA, T. P. *Introdução ao VHDL e Sistemas Digitais Reconfiguráveis*. 1. ed. Rio de Janeiro: LTC, 2021.

SEMANA 10 – SDRE. *Material Didático – Máquinas de Estado e FSM*. Instituto Federal de São Paulo (IFSP), 2025. Material interno da disciplina.

TERASIC. *DE10-Lite Board Schematic*. Terasic Technologies, 2020. Disponível em: <https://www.terasic.com.tw/>. Acesso em: 10 nov. 2025.

WAKERLY, John F. *Digital Design: Principles and Practices*. 5. ed. New Jersey: Pearson, 2018.