Universidade Federal da Bahia - UFBA

Departamento de Engenharia Elétrica e de Computação - DEEC

ENGG56 - PROJETO DE CIRCUITOS INTEGRADOS DIGITAIS

Felipe Ma

João Cerqueira

Relatório do Trabalho Semestral

Salvador - BA

2026

Sumário

1 Introdução

O trabalho foi desenvolvido no âmbito da disciplina ENGG56 – Projeto de Circuitos Integrados Digitais, da Escola Politécnica da Universidade Federal da Bahia (UFBA), e tem como objetivo consolidar os conceitos de projeto, modelagem e verificação de sistemas digitais utilizando a linguagem de descrição de hardware Verilog.

Ao longo do trabalho, são abordados problemas representativos de aplicações reais em sistemas digitais, envolvendo processamento serial de dados, controle de memória externa, máquinas de estados finitos (FSM), operações aritméticas sequenciais e lógica combinacional otimizada. Os projetos desenvolvidos foram pensados para síntese em FPGA, com plataforma alvo a placa DE2-115.

A metodologia adotada contempla todas as etapas clássicas do fluxo de projeto de circuitos digitais: especificação funcional, descrição em Verilog, simulação funcional por meio de testbenches no ModelSim, síntese lógica no Quartus e análise temporal com o TimeQuest, garantindo a conformidade do projeto com restrições de tempo.

Os quatro problemas propostos exploram diferentes aspectos do projeto digital. O primeiro trata da decodificação de sinais provenientes de um controle remoto infravermelho, com validação de quadros e sinalização de dados válidos. O segundo envolve o controle de uma memória SRAM externa e a coordenação de operações de leitura, soma e escrita por meio de uma FSM. O terceiro aborda o controle sequencial de cálculos associados à transformada inversa do cosseno discreto (IDCT), amplamente utilizada em sistemas de codificação de vídeo MPEG. Por fim, o quarto problema trata do controle lógico de sinais de tráfego, com base na detecção de veículos, exigindo uma solução combinacional otimizada.

Dessa forma, este trabalho busca integrar teoria e prática, reforçando competências essenciais no desenvolvimento de sistemas digitais complexos, desde a concepção até a validação funcional e temporal em FPGA.

2 Questão 1

Decodificação de Sinais de um Controle Remoto Infravermelho

Esta seção detalha o desenvolvimento e a validação do módulo `RemoteController`, projetado para decodificar sinais de infravermelho de um controle remoto seguindo um protocolo específico de comunicação serial.

2.1 Máquina de estados

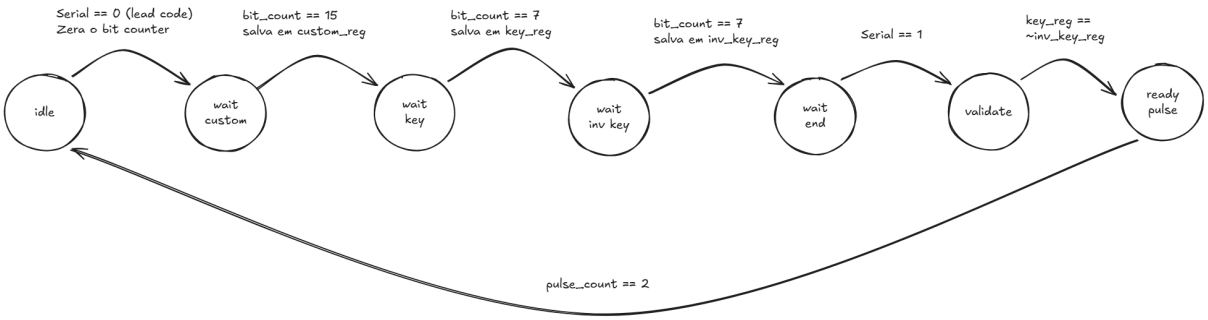

Figura 1: Diagrama de estados da questão 1

Estados

A figura 1 apresenta um diagrama de estados simplificado da máquina de estados implementada, que evidencia as condições para cada transição de estado. A máquina de estados funciona da seguinte maneira:

- **IDLE**: Estado de espera, aguardando o início do Lead Code (Serial = 0).
- **WAIT_CUSTOM**: Captura os 16 bits do código customizado.
- **WAIT_KEY**: Captura os 8 bits do código da tecla.
- **WAIT_INV_KEY**: Captura os 8 bits do código invertido.
- **WAIT_END**: Aguarda o bit de parada (Serial = 1).
- **VALIDATE**: Compara se o `key_reg` é o inverso exato de `inv_key_reg`.
- **READY_PULSE**: Gera um pulso de saída indicando que o dado é válido.

Lógica de Saída

As saídas se comportam de acordo com o estado atual:

- **Ready**: Fica em nível alto apenas durante o estado `READY_PULSE`.
- **Tecla**: Durante o pulso de `Ready`, a saída exibe o valor de 8 bits armazenado; caso contrário, retorna a zero.

2.2 Simulação e Testbench

A validação do módulo `RemoteController` foi realizada através de um testbench em Verilog (`RemoteController_tb.v`). Para garantir a precisão temporal, a simulação utilizou uma escala de tempo de $1ns/1ps$ e um sinal de clock de $100MHz$, com período de $10ns$.

A principal ferramenta de teste foi a *task* `send_frame`, que automatiza o envio da sequência serial: *Lead Code* (0), seguido por 16 bits de *Custom Code*, 8 bits de *Key Code*, 8 bits de *Inv Key Code* e o *End Code* (1).

Foram realizados três cenários de teste principais:

- **Teste 1 (Tecla '1'):** Envio do código `0x01` com o campo *Inv Key Code* correto. O sistema validou a tecla e ativou o sinal `Ready` conforme esperado.
- **Teste 2 (Tecla 'Play/Pause'):** Envio do código `0x16`. O teste confirmou que o registrador interno `tecla_storage` armazena novos valores corretamente após sucessivas transmissões.
- **Teste 3 (Erro de Integridade):** Envio da tecla `0x01` com o *Inv Key Code* propositalmente incorreto (igual ao *Key Code*). O módulo rejeitou o frame, mantendo a saída `Ready` em nível baixo e a saída `Tecla` em `0x00`.

O monitoramento dos estados da FSM durante a simulação permitiu observar a transição precisa entre os estados de captura (`WAIT_CUSTOM`, `WAIT_KEY`, `WAIT_INV_KEY`) e o estado de validação final.

2.3 Síntese e Restrições de Tempo

O projeto foi configurado para o dispositivo FPGA Altera Cyclone IV, especificamente o modelo `EP4CE115F29C7`. As restrições de temporização foram definidas no arquivo `.sdc`.

As principais especificações técnicas de temporização são:

O período de $10ns$ implica numa frequência de $100MHz$, respeitando os limites impostos pelo time analyzer do Quartus. Os atrasos de entrada e saída foram aplicados a todos os inputs e outputs, seguindo o valor padrão de 10% do período de clock.

Parâmetro	Valor
Período do Clock (T_{clk})	10.000ns
Atraso de Entrada (*Input Delay*)	1.000ns
Atraso de Saída (*Output Delay*)	1.000ns

Tabela 1: Restrições de tempo aplicadas ao projeto.

3 Questão 2

3.1 Parte 1

3.1.1 Diagrama de blocos

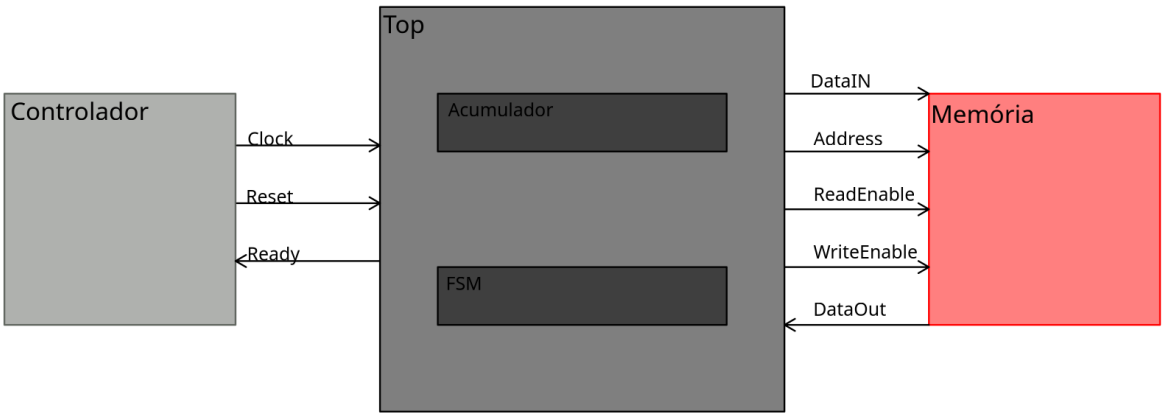

Figura 2: Diagrama de blocos da questão 2

A figura 2 apresenta a arquitetura de alto nível do sistema digital implementado. O módulo principal, denominado TOP, atua como a unidade de processamento central e realiza a interface entre dois periféricos externos: um Controlador (à esquerda) e uma Memória SRAM (à direita).

O funcionamento do sistema é descrito a seguir:

- **Interface de Controle**: O módulo TOP opera de forma síncrona, recebendo os sinais de `Clock` e `Reset` provenientes do Controlador externo. Em contrapartida, o sistema sinaliza o término de suas operações através do sinal de saída `Ready`.
- **Interface de Memória**: O acesso aos dados é realizado através de um barramento dedicado conectado à memória SRAM. O módulo TOP gerencia o endereçamento (`Address`) e os sinais de controle de leitura (`ReadEnable`) e escrita (`WriteEnable`). O fluxo de dados é bidirecional em relação ao sistema: o sinal `DataOut` (saída da memória) alimenta o módulo TOP com os dados a serem somados, enquanto o sinal `DataIN` (entrada da memória) transporta o resultado acumulado gerado pelo TOP para ser gravado na SRAM.

- **Arquitetura Interna**: Internamente, o módulo TOP encapsula dois sub-blocos funcionais essenciais:

    1. **FSM (Máquina de Estados Finitos)**: Responsável pela lógica de controle sequencial, geração de endereços de memória e gerenciamento dos sinais de controle internos.
    2. **Acumulador**: Responsável pela execução das operações aritméticas (somas) sobre os dados recebidos.

3.2 Parte 2

- **Descrição do Módulo Acumulador:** O módulo Acumulador constitui o caminho de dados do sistema, sendo responsável por executar as operações aritméticas de soma. O circuito foi implementado em Verilog utilizando uma descrição comportamental que reflete o diagrama de blocos e o fluxo de sinais especificados no projeto.
- **Características Gerais:** O módulo foi desenvolvido de forma parametrizável (parâmetro `TAMANHO = 16`), permitindo que a largura do barramento de dados seja facilmente ajustada, embora opere nativamente com palavras de 16 bits, como indicado no projeto. A arquitetura interna é composta por:

    1. **REG B:** Armazena os dados vindo da memória.
    2. **Somador Combinacional:** Responsável pela soma.
    3. **REG A:** Armazena o resultado da soma.

- **Funcionamento e Sinais de Controle:** Este módulo segue a especificação funcional do diagrama fornecido, conforme solicitado no enunciado.

    - **Captura de Dados (Sinal `Load`):** O barramento de entrada `M` é conectado ao Registrador B. A escrita neste registrador ocorre de forma síncrona com a borda de subida do clock principal, sendo condicionada pela ativação do sinal `Load`. Quando `Load = 1` no ciclo de clock, o valor presente em `M` é armazenado em B.
    - **Operação de Soma:** Um somador implementa a operação $S = A + B$, onde A representa o valor atualmente acumulado e B o valor previamente capturado. Essa lógica é puramente combinacional, de modo que o resultado da soma reflete imediatamente as alterações nos registradores de entrada.
    - **Atualização do Acumulador (Sinal `Transfer`):** O resultado da soma é transferido para o Registrador A de forma síncrona com a borda de subida do clock do sistema, desde que o sinal `Transfer` esteja ativo. Nesse instante, o valor de saída do módulo (`Saidas`) é atualizado com o novo total acumulado.

– **Inicialização (Sinal `Clear`):** O módulo possui um sinal de inicialização assíncrona ativo em nível lógico baixo, controlado por `Clear`. Quando este sinal é levado a nível baixo (0), o Registrador A é imediatamente zerado, independentemente do clock, permitindo o início de um novo ciclo de somas.

3.3 Parte 3

3.3.1 Implementação

Para atender aos requisitos impostos, implementou-se uma máquina de estados finitos (FSM) responsável por coordenar a leitura na memória externa e o acionamento do Acumulador (sinais `CLEAR`, `LOAD` e `TRANSFER`). A FSM opera de forma cíclica sobre os conjuntos de endereços 0–6, 8–14, 16–22 e 24–30, gravando os resultados em 7, 15, 23 e 31, respectivamente. Ao final dos quatro conjuntos, o sinal `Ready` é ativado por pelo menos um ciclo e o processo reinicia automaticamente. Pode-se verificá-la na figura 3

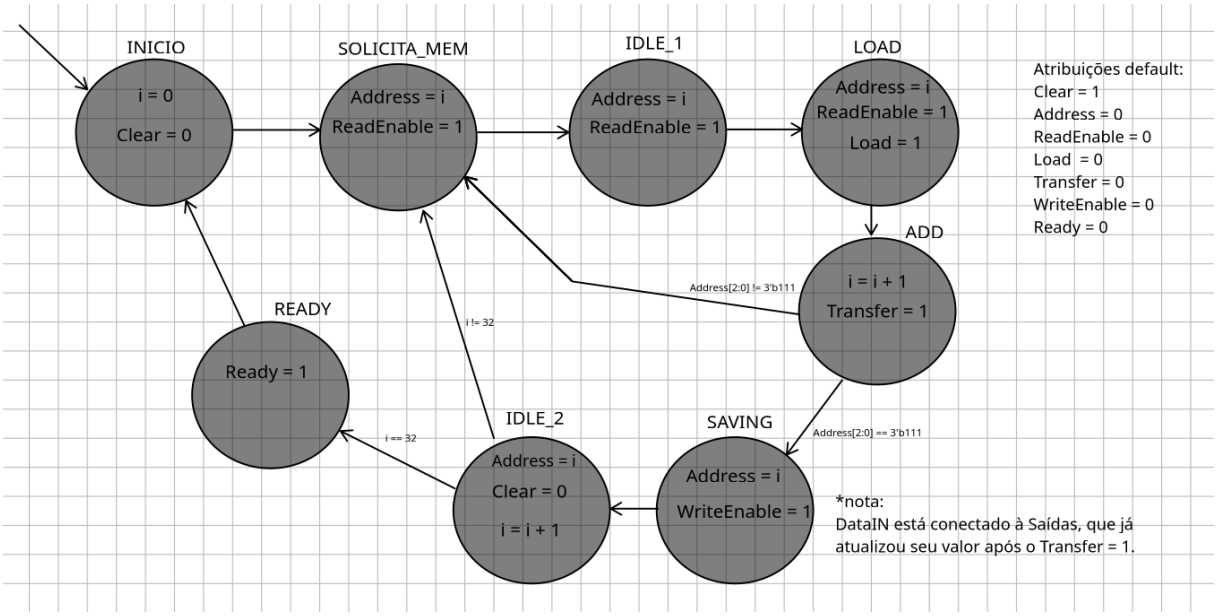

Figura 3: Máquina de Estados Finitos do projeto 2

A FSM foi projetada considerando a característica da SRAM descrita no enunciado: após habilitar `ReadEnable` ou `WriteEnable`, a operação efetiva ocorre com *delay* de 1 pulso de clock, e a saída `DataOut` vai para alta impedância quando `ReadEnable` é desativado. Por isso, foram incluídos estados de espera (“`IDLE_1 e IDLE_2`”) para respeitar o tempo de disponibilização do dado lido.

Fluxo de estados (visão geral)

- **INICIO:** inicializa o índice `i` (endereço atual) e aciona o `CLEAR` para zerar o acumulador.
- **SOLICITA_MEM:** coloca `Address = i` e ativa `ReadEnable` para solicitar leitura na SRAM

- **IDLE_1:** estado de espera para respeitar o *delay* da memória (garante que `DataOut` será lido no tempo correto) mantendo o endereço para evitar desestabilidade do sinal e `ReadEnable` pois, caso contrário, a saída iria para alta impedância imediatamente.
- **LOAD:** mantém o endereço e `ReadEnable` e aplica `LOAD = 1` por um ciclo para capturar o dado lido no acumulador.
- **ADD:** aplica `TRANSFER = 1` por um ciclo e incrementa `i` (avança para o próximo endereço).
- **Condição de fim de bloco:** quando `Address[2:0] == 3'b111` (endereços 7, 15, 23, 31), a FSM reconhece que o bloco terminou:
  - para **7/15/23**: segue para `SAVING` (grava resultado) e depois prepara o próximo bloco;
  - para **31**: finaliza o ciclo completo.
- **SAVING:** grava na SRAM ativando `WriteEnable = 1` com `Address = i` (endereço de escrita do resultado). Como `DataIN` está conectado às saídas do acumulador, o valor gravado corresponde à soma já atualizada após `TRANSFER`.
- **IDLE_2:** estado de ajuste e preparação para o próximo bloco (inclui avanço de `i` e controle do `CLEAR`).
- **READY:** ativa `Ready = 1` por pelo menos um ciclo para sinalizar término (como solicitado) em seguida retorna ao **INICIO**, reiniciando as operações.

Nota: Percebemos que os endereços a serem utilizados para gravar têm, em comum:

- 7 = 5'b00111;
- 15 = 5'b01111;
- 23 = 5'b10111;
- 31 = 5'b11111;

Ou seja, todos eles acontecem quando os 3 últimos bits possuem valor 1. Isso foi utilizado no estado `ADD` para decidir qual o estado seguinte.

3.3.2 Simulação e Testbench

Para validação funcional do módulo TOP, foi desenvolvido um testbench em Verilog no qual a UUT é instanciada e conectada a um modelo comportamental de SRAM 32×16 implementado como um vetor MEMORIA[0:31]. Esse modelo permite reproduzir, em simulação, o comportamento esperado da memória descrita no enunciado, incluindo delays nas leituras e saída em alta impedância.

A temporização de leitura foi implementada por meio de registradores intermediários. A cada borda de subida do clock, o testbench captura o endereço solicitado e o estado de ReadEnable (em addr_q e re_q). No ciclo seguinte, o dado correspondente é disponibilizado em dout_reg, e o barramento DataOut passa a ser dirigido apenas quando drive=1, resultando em:

- Dado válido em DataOut somente 1 ciclo após a ativação de ReadEnable, de acordo com o atraso especificado.
- DataOut em alta impedância quando ReadEnable = 0.

Inicialização dos dados e verificação automática (PASS/FAIL)

O testbench carrega a memória com valores conhecidos para permitir a verificação do somatório. Na primeira rodada, os endereços dos quatro blocos são inicializados com constantes (ex.: 1, 2, 10 e 5), de modo que os resultados esperados sejam:

- bloco 0–6 → grava em 7: 7
- bloco 8–14 → grava em 15: 14
- bloco 16–22 → grava em 23: 70
- bloco 24–30 → grava em 31: 35

Após a aplicação de `reset` e execução do circuito, o testbench aguarda `Ready=1`, que sinaliza o término do processamento dos quatro blocos. Em seguida, são executadas comparações com `if` nos endereços de resultado (7, 15, 23 e 31). Cada comparação emite mensagens [PASS] ou [FAIL], imprimindo o valor obtido e o valor esperado.

Segunda rodada: verificação de continuidade após Ready

Conforme exigido no enunciado, a FSM deve reiniciar automaticamente as operações após sinalizar `Ready`. Para verificar essa continuidade, o testbench realiza uma segunda rodada: a memória é configurada novamente (cada endereço recebe seu próprio índice, por exemplo MEMORIA[k]=k nos intervalos de interesse). Em seguida, o testbench espera Ready retornar a 0 e subir novamente (wait(Ready==0); wait(Ready==1);), garantindo que a checagem se refere ao próximo ciclo completo, e não ao mesmo pulso anterior.

Os valores esperados da segunda rodada são calculados a partir das somas dos blocos:

- 0–6 → soma = 21 (grava em 7)
- 8–14 → soma = 77 (grava em 15)
- 16–22 → soma = 133 (grava em 23)
- 24–30 → soma = 189 (grava em 31)

Novamente, as comparações com if geram relatórios PASS/FAIL, confirmando não apenas a correção do somatório e do endereçamento, mas também que o controlador continua operando ciclicamente após `Ready`, como especificado.

```
VSIM 16> run -all
# ====== inicio da simulacao ======
# MEMORIA: Escreveu     7 no Endereco  7
# MEMORIA: Escreveu    14 no Endereco 15
# MEMORIA: Escreveu    70 no Endereco 23
# MEMORIA: Escreveu    35 no Endereco 31
# ====== verificacao dos primeiros resultados ======
# [PASS] Grupo 1 (End 7):     7 (Esperado 7)
# [PASS] Grupo 2 (End 15):    14 (Esperado 14)
# [PASS] Grupo 3 (End 23):    70 (Esperado 70)
# [PASS] Grupo 4 (End 31):    35 (Esperado 35)
# ====== fim dos primeiros resultados ======
# MEMORIA: Escreveu    21 no Endereco  7
# MEMORIA: Escreveu    77 no Endereco 15
# MEMORIA: Escreveu   133 no Endereco 23
# MEMORIA: Escreveu   189 no Endereco 31
# ====== verificacao dos segundos resultados(continuidade do ciclo, como demandado) ======
# [PASS] Grupo 1 (End 7):    21 (Esperado 21)
# [PASS] Grupo 2 (End 15):    77 (Esperado 77)
# [PASS] Grupo 3 (End 23):   133 (Esperado 133)
# [PASS] Grupo 4 (End 31):   189 (Esperado 189)
# ====== fim dos segundos resultados ======
# ** Note: $stop    : /home/joao-cerqueira/Documents/ENGG56/Project 2/tb_TOP.v(120)
#    Time: 5870 ns  Iteration: 1  Instance: /tb_TOP
# Break in Module tb_TOP at /home/joao-cerqueira/Documents/ENGG56/Project 2/tb_TOP.v line 120
```

Figura 4: Screenshot do testbench 2

3.4 Síntese e Restrições de Tempo

O projeto foi condicionado para síntese em um FPGA Altera Cyclone IV (EP4CE115F29C7) utilizando o arquivo de restrições SDC (`projeto_2.out.sdc`):

As principais especificações técnicas de temporização são:

Parâmetro	**Valor**
Período do Clock (T_{clk})	10.000ns
Atraso de Entrada (*Input Delay*)	1.000ns
Atraso de Saída (*Output Delay*)	1.000ns

Tabela 2: Restrições de tempo aplicadas ao projeto.

O período de $10ns$ implica numa frequência de $100MHz$, respeitando os limites impostos pelo time analyzer do Quartus. Os atrasos de entrada e saída foram aplicados a todos os inputs e outputs, seguindo o valor padrão de 10% do período de clock.

4 Questão 3

Controle de Fluxo para Cálculo de IDCT (MPEG)

Esta seção descreve o projeto e a validação do módulo `FSM_Control`, responsável por gerenciar a leitura de coeficientes $F(u,v)$ de uma SRAM e coordenar a unidade de acumulação (MAC) para a reconstrução de blocos de pixels 8×8 em sistemas MPEG. A operação baseia-se na expressão da IDCT:

$$f(x,y) = \frac{2}{N} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} C(u)C(v)F(u,v) \cos \frac{(2x+1)u\pi}{2N} \cos \frac{(2y+1)v\pi}{2N}$$

4.1 Máquina de Estados

A lógica de controle foi implementada através de uma Máquina de Estados Finitos (FSM) de seis estados, projetada para atender às especificações temporais da memória SRAM:

- **S_IDLE**: O sistema permanece em repouso até que o sinal `Start` seja ativado.
- **S_SEND_ADDR**: O endereço de leitura, composto pela concatenação dos índices $\{u,v\}$, é enviado à SRAM.
- **S_ACT_RE**: Ativação do sinal `Read_Enable` para iniciar o ciclo de leitura.
- **S_WAIT_DATA**: Estado de espera de um ciclo de clock para garantir que o dado `Data_Out` (12 bits) esteja estabilizado no barramento.
- **S_ACCUM**: Pulsa o sinal `Active_MAC`, permitindo que a unidade de hardware realize a multiplicação e acumulação dos coeficientes.
- **S_DONE**: Sinaliza o término do processamento de todos os pixels do bloco através do sinal `Ready`.

A transição entre os estados e a atualização dos contadores ocorrem no estado `S_ACCUM`. O sistema percorre quatro níveis de iteração (u,v,x,y) de 0 a 7, totalizando 4096 ciclos de acumulação antes de retornar ao estado inicial.

4.2 Simulação e Testbench

A validação funcional foi realizada via *testbench* (`FSM_Control_tb.v`) simulando um clock de 100MHz (período de 10ns). Os principais aspectos testados foram:

- **Monitoramento de Sinais**: Utilizou-se a diretiva `$monitor` para rastrear as coordenadas espaciais (x, y) e de frequência (u, v) em cada pulso de clock.
- **Contagem de Operações**: Um contador interno (`mac_count`) no testbench verificou se o sinal `Active_MAC` foi acionado exatamente 4096 vezes, garantindo a integridade do cálculo para o bloco completo.
- **Segurança (Watchdog)**: Implementou-se um *timeout* de 1ms através de um bloco `fork-join` para detectar travamentos na FSM caso o sinal `Ready` falhasse em subir.

4.3 Síntese e Restrições de Tempo

O projeto foi condicionado para síntese em um FPGA Altera Cyclone IV (EP4CE115F29C7) utilizando o arquivo de restrições SDC (`q3_mpeg.out.sdc`):

As principais especificações técnicas de temporização são:

Parâmetro	**Valor**
Período do Clock (T_{clk})	$10.000ns$
Atraso de Entrada (*Input Delay*)	$1.000ns$
Atraso de Saída (*Output Delay*)	$1.000ns$

Tabela 3: Restrições de tempo aplicadas ao projeto.

O período de $10ns$ implica numa frequência de $100MHz$, respeitando os limites impostos pelo time analyzer do Quartus. Os atrasos de entrada e saída foram aplicados a todos os inputs e outputs, seguindo o valor padrão de 10% do período de clock.

5 Questão 4

Para realização do projeto, primeiro observou-se os casos de ocupação das vias e suas saídas esperadas:

- O sinal leste-oeste (L-O) estará verde sempre que existirem veículos em ambas as vias de direção **C** e **D**: XX11 (4 configurações)
- O sinal leste-oeste (L-O) estará verde sempre que existirem veículos ou **C** ou em **D**, estando as outras duas vias, **A** e **B**, sem nenhum veículo detectado: 0001 e 0010 (2 configurações)

- O sinal norte-sul (N-S) estará verde sempre que existirem veículos em **A** e em **B**, estando **C** e **D** vazias: 1100 (1 configuração)
- O sinal norte-sul (N-S) estará verde quando ou **A** ou **B** estiverem ocupadas, enquanto **C** e **D** estiverem vazias: 1000 e 0100 (2 configurações)
- O sinal leste-oeste (L-O) estará verde quando nenhum veículo tiver sido detectado pelos sensores: 0000 (1 configuração)

Todas as configurações restantes são as que continham algum carro em ambas direções. Para resolver isso, levou-se em consideração o enunciado:

"A figura abaixo mostra a interseção de uma via preferencial com uma outra secundária. Vários sensores de detecção de veículos estão colocados ao longo das mãos de direção C e D **(via principal)** e A e B (via secundária)".

Então, para esses casos, ativou-se o `L_O`, já que esta é a via principal. A partir disso, construiu-se a seguinte tabela 4:

ABCD	N_S	L_O
0000	0	1
0001	0	1
0010	0	1
0011	0	1
0100	1	0
0101	0	1
0110	0	1
0111	0	1
1000	1	0
1001	0	1
1010	0	1
1011	0	1
1100	1	0
1101	0	1
1110	0	1
1111	0	1

Tabela 4: Tabela projeto 4

A partir dessa tabela, pôde-se construir a seguinte tabela de Karnaugh na figura 5 e obter a saída otimizada:

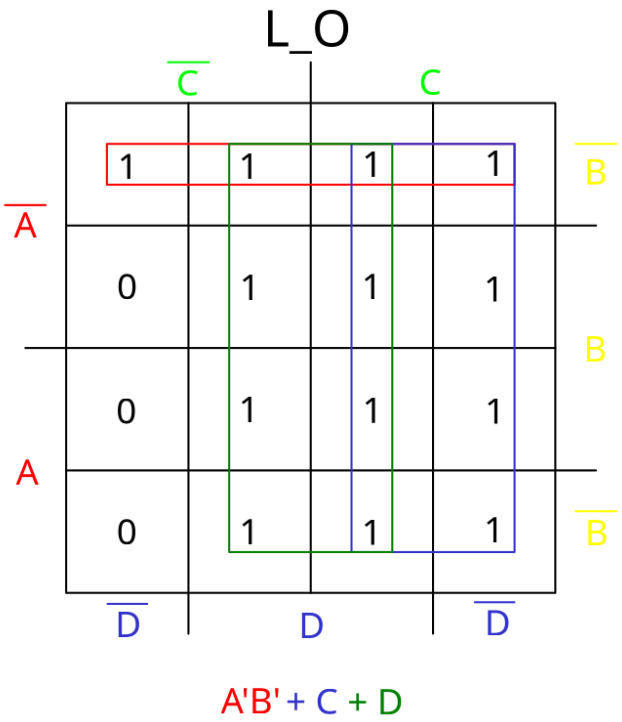

Figura 5: Mapa de Karnaugh

5.1 Simulação e Testbench

No Projeto 4, foi elaborado um testbench para validar o módulo combinacional responsável por controlar os sinais do semáforo a partir das entradas dos sensores A, B, C e D. O testbench varre sistematicamente todas as 16 combinações possíveis das entradas (de 0000 a 1111), atribuindo cada combinação aos sinais de entrada e aguardando um pequeno intervalo de tempo para estabilização da lógica combinacional. Em seguida, para cada caso, o testbench imprime no console uma tabela com três colunas (ABCD, N_S e L_O), registrando os valores efetivamente produzidos pelo circuito.

A tabela obtida por simulação coincidiu integralmente com a tabela de verdade definida no relatório, o que confirma que o circuito implementa corretamente as regras especificadas no enunciado. Em particular, os resultados mostraram que:

1. L_O permanece em nível alto sempre que há veículos na via principal (sensores C ou D ativos), garantindo prioridade para a via preferencial;

2. quando C e D estão vazios, o sinal N_S vai para nível alto sempre que houver veículos detectados em A ou B, permitindo a liberação do fluxo na via secundária;

3. quando nenhum sensor detecta veículo, a saída L_O permanece verde, conforme exigido.

Além disso, em todas as 16 combinações simuladas, observou-se que nunca ocorreu simultaneamente N_S = 1 e L_O = 1, nem N_S = 0 e L_O = 0, atendendo à restrição final do problema de não permitir ambos os sinais verdes ou ambos vermelhos ao mesmo tempo.

```
# run -all
# ====================================
#    TABELA OBTIDA POR SIMULACAO
#    (Projeto 4 - Controlador Semaforo)
# ====================================
#   ABCD  |  N_S  |  L_O
# --------+-------+------
#   0000  |   0   |   1
#   0001  |   0   |   1
#   0010  |   0   |   1
#   0011  |   0   |   1
#   0100  |   1   |   0
#   0101  |   0   |   1
#   0110  |   0   |   1
#   0111  |   0   |   1
#   1000  |   1   |   0
#   1001  |   0   |   1
#   1010  |   0   |   1
#   1011  |   0   |   1
#   1100  |   1   |   0
#   1101  |   0   |   1
#   1110  |   0   |   1
#   1111  |   0   |   1
# ====================================
# ** Note: $stop    : /home/joao-cerqueira/Documents/ENGG56/q4_via/tb_q4_via.v(34)
#    Time: 16 ns  Iteration: 0  Instance: /tb_q4_via
# Break in Module tb_q4_via at /home/joao-cerqueira/Documents/ENGG56/q4_via/tb_q4_via.v line 34
```

Figura 6: Tabela da questão 4 obtida através do testbench

Dessa forma, a simulação funcional por testbench comprova que o módulo implementado satisfaz integralmente o comportamento esperado para o controlador do cruzamento, cobrindo todas as possibilidades de entrada e respeitando as condições de prioridade e segurança descritas no trabalho.

Pode-se verificar o resultado obtido através do testbench através da figura 6.

5.2 Síntese e Restrições de Tempo

O projeto foi condicionado para síntese em um FPGA Altera Cyclone IV (EP4CE115F29C7) utilizando o arquivo de restrições SDC (`q4_via.out.sdc`):

Por se tratar de um circuito puramente combinacional, a análise temporal foi realizada por meio de uma restrição de atraso máximo entre entradas e saídas. Adicionalmente, os caminhos de hold foram declarados como false path, uma vez que não há janelas de captura associadas a bordas de clock nesse circuito.

Para isso, utilizou-se, para constraints, os seguintes comandos:

- set_max_delay 20.0 -from [get_ports A B C D] -to [get_ports N_S L_O]
- set_false_path -hold -from [get_ports A B C D]
- set_false_path -hold -to [get_ports N_S L_O]