Teste Reação

Universidade de Aveiro

Rafael Marques, Tiago Pita

VERSAO 1

Teste Reação

Dept. de Eletrónica, Telecomunicações e Informática
Universidade de Aveiro

Rafael Marques, Tiago Pita
(119927) rafaelfmarques@ua.pt, (120152) tiagomsp18@ua.pt

5 de junho de 2024

Conteúdo

1 Introdução **1**

2 Manual de Utilização **2**
2.1 Funcionamento . . . 2
2.2 Esquema da Máquina . . . 2

3 Arquitetura e Implementação **4**
3.1 DebouncerUnit . . . 4
3.2 LightAndReactionTime . . . 5
3.3 DonAssignment . . . 5
3.4 RandomTimeGen . . . 6
3.5 Init . . . 7
3.6 StateMachine . . . 8
3.7 MEFcompetition . . . 9
3.8 Bin7SegDecoderEN . . . 10
3.9 BinToBCD . . . 10
3.10 ConFDisplay . . . 11
3.11 PlayerAWin . . . 11
3.12 PlayerBWin . . . 12
3.13 XORPenalty . . . 12
3.14 MUX4:1 7bits . . . 13
3.15 Pulse Gen . . . 13
3.16 tEStDisplay . . . 14
3.17 win lights . . . 14

4 Validações **16**

5 Conclusões e Contribuições **17**
5.1 Conclusões . . . 17
5.2 Contribuições dos autores . . . 17

6 Referências **18**

Lista de Tabelas

Lista de Figuras

2.1 Ilustração do Esquema da Máquina 2

3.1 Ilustração do Top-Level da Máquina 4
3.2 Ilustração do bloco DebounceUnit 4
3.3 Ilustração do bloco LightAndReactionTime 5
3.4 Ilustração do bloco DonAssignment 6
3.5 Ilustração do bloco RandomTimeGen 6
3.6 Ilustração do bloco Init 7
3.7 Ilustração do bloco StateMachine 8
3.8 Ilustração do bloco MEFcompetition 9
3.9 Ilustração do bloco Bin7SegDecoderEN 10
3.10 Ilustração do bloco BinToBCD 10
3.11 Ilustração do bloco ConFDisplay 11
3.12 Ilustração do bloco PlayerAWin 11
3.13 Ilustração do bloco PlayerBWin 12
3.14 Ilustração do bloco XORPenalty 12
3.15 Ilustração do bloco MUX4:1 7bits 13
3.16 Ilustração do bloco pulse gen 13
3.17 Ilustração do bloco tEStDisplay 14
3.18 Ilustração do bloco win lights 14
3.19 Esquema da Máquina de Estados 15

Capítulo 1

Introdução

Como projeto final da cadeira de Laboratório de Sistemas Digitais (LSD), foi-nos proposto o desenvolvimento de um teste de reação. Este projeto foi implementado utilizando VHSIC Hardware Description Language (VHDL) para simular o seu comportamento na placa FPGA Terasic DE2-115. O objetivo é medir o tempo de reação do utilizador a um estímulo luminoso.

O projeto contém:

1. Fase Configuração

   Secção de Display: Uma secção que exibe a palavra Config.

   Secção de Display: Uma secção onde é possível ver o incremento ou diminuação da pontuação alvo.

   Botões de Controle:

   - **Botão `Confirm`:** Utilizado para confirmar a pontuação.
   - **Botão de Aumentar:** Utilizado para aumentar a pontuação.
   - **Bottão de Dimuir:** Utilizado para diminuir a pontuação alvo.

2. Fase jogo

   Secção de Display: Uma secção que exibe o tempo de reação do utilizador ao estímulo luminoso.

   Secção de Display: Duas secções que exibem os pontos de cada jogador.

   Botões de Controle:

   - **Botão `Start`:** Utilizado para iniciar o teste de reação.
   - **Botão de Reset:** Utilizado para reiniciar o teste.
   - **Botões `Reação`:** Utilizado por cada jogador.

3. Fase Conclusão

   Secção display: Uma secção onde mostra o vencedor.

A implementação foi realizada em VHDL, e o comportamento foi simulado na placa FPGA Terasic DE2-115. O código foi desenvolvido para gerir a ativação das luzes, a contagem do tempo de reação, e a exibição do tempo medido no display. Além disso, a lógica inclui estados para aguardar a ativação do teste, iniciar a contagem do tempo de reação, e reiniciar o sistema quando necessário.

Capítulo 2

Manual de Utilização

2.1 Funcionamento

A máquina inicia pedindo ao utilizador que ajuste a pontuação alvo.

Quando o utilizador pressionar a tecla de confirmação, começa um jogo entre dois jogadores, para ver qual tem o menor tempo de reação perante um estímulo visual.O estímulo visual aparece de forma aleatório, devido a geração de um tempo aleatório, para que não exista fraude de resultados.

Caso um jogador tenha uma reação prematura , e reaja antes do estímulo visual , será penalizado.

2.2 Esquema da Máquina

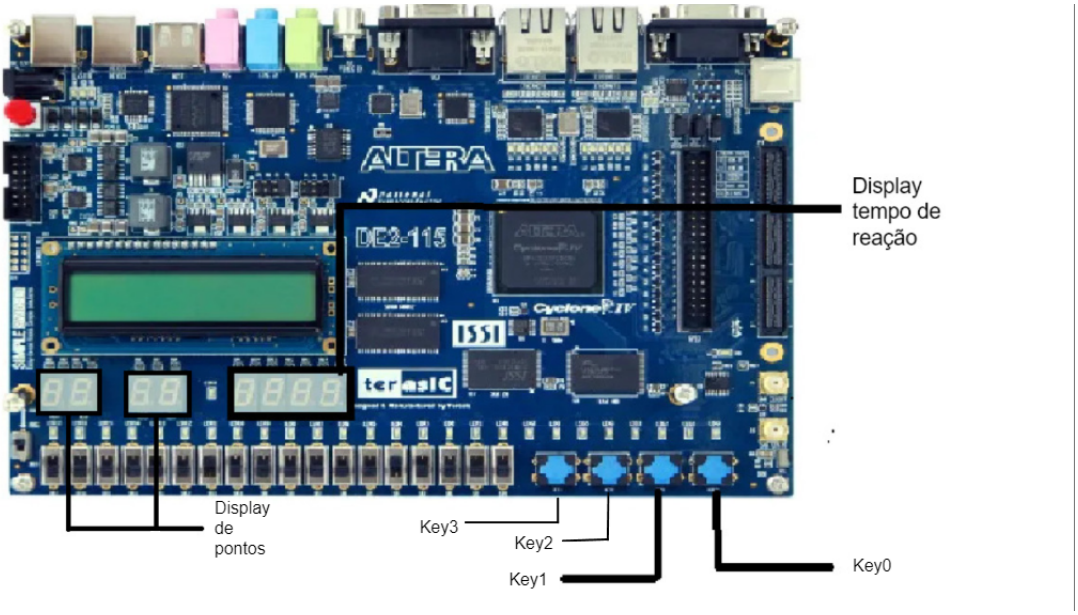

Figura 2.1: Ilustração do Esquema da Máquina

1. Key1: botão confirmar
2. **Inicio**
   ↪ Key0: botão confirmar
   ↪ Key2: Diminuir o máximo de pontos
   ↪ Key3: Aumentar o máximo de pontos
3. **Durante o jogo**
   ↪ Key0: Start
   ↪ Key2: Player A
   ↪ Key3: Player B

4. **Fim do jogo**
   ↪ Key0: Recomeçar o jogo

Capítulo 3

Arquitetura e Implementação

O Top-Level da máquina é composto por 3 componentes principais que depois se ramificam em subcomponentes mais pequenos.

A figura 3.1 representa uma ilustração gráfica do Top-Level da máquina implementado em VHDL.

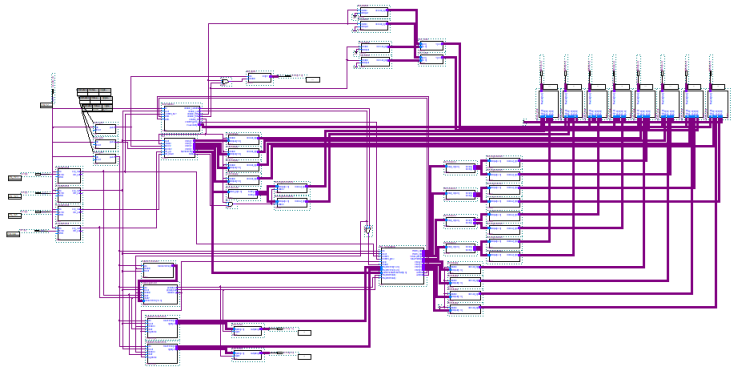

Figura 3.1: Ilustração do Top-Level da Máquina

3.1 DebouncerUnit

Esta componente chamada de DebounceUnit implementa uma unidade de debounce para dois sinais de entrada: KEYin e SWin. A unidade de debounce é usada para eliminar a oscilação (bounce) que pode ocorrer quando um botão é pressionado ou um interruptor é ativado.

A entidade tem várias portas de entrada e saída:

- Entradas: clk, KEYin, SWin
- Saídas: KEYout, SWout

O código implementa a seguinte lógica:

Em cada borda de subida do sinal de clock (clk), o estado anterior do sinal KEYin é invertido e armazenado e o estado atual do sinal SWin é armazenado. Os sinais são então atribuídos às saídas KEYout e SWout, respectivamente.

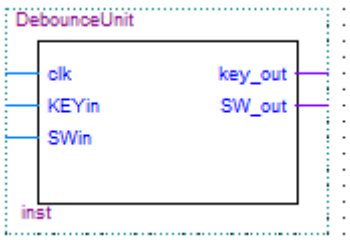

Figura 3.2: Ilustração do bloco DebounceUnit

3.2 LightAndReactionTime

Esta componente chamada LightAndReactionTime que implementa a lógica para medir o tempo de reação a um estímulo de luz. A entidade tem várias portas de entrada e saída, incluindo sinais de clock, reset, enable, início, luz ligada, tempo de reação e luz.

O código implementa a seguinte lógica:

1. Reset: Se o sinal de reset é '1', todos os sinais internos são resetados para seus valores iniciais.
2. Máquina de estados: Se o sinal de clock tem uma borda de subida e o sinal de enable é '1', a máquina de estados é ativada. A máquina de estados tem três estados:
   - Estado 0: Aguarda o sinal LightON. Se LightON é '1' e start é '0', a luz é ativada e o estado muda para 1.
   - Estado 1: Ativa o estímulo e verifica se start foi desligado. Se start é '0', o estado muda para 2.
   - Estado 2: Inicia a contagem do tempo de reação. Se start é '1', a luz é desativada, o tempo de reação é armazenado e o estado retorna para 0. Se start é '0', o tempo de reação é incrementado.

O sinal de luz ativa é atribuído à saída de luz.

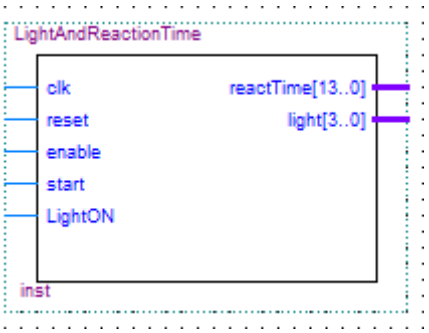

Figura 3.3: Ilustração do bloco LightAndReactionTime

3.3 DonAssignment

Esta componente chamada DonAssignment que implementa a lógica para um jogo entre dois jogadores. A entidade tem várias portas de entrada e saída, incluindo sinais de clock, reset, enable, início para cada jogador, tempo aleatório, penalidades para cada jogador e enable da luz.

O código implementa a seguinte lógica:

1. Reset: Se o sinal de reset é '1', todos os sinais internos são resetados para seus valores iniciais.
2. Máquina de estados: Se o sinal de clock tem uma borda de subida e o sinal de enable é '1', a máquina de estados é ativada. A máquina de estados tem cinco estados:
   - Estado idle: Aguarda o sinal de início de qualquer jogador. Se um jogador inicia, um valor aleatório é atribuído a Don, o timer é resetado, os jogadores são marcados como não tendo respondido e o estado muda para running.
   - Estado running: Incrementa o timer. Se o timer atinge Don, o estado muda para wait_response. Se um jogador inicia antes do timer atingir Don, o estado muda para penalty ou penalty2 dependendo do jogador.
   - Estado wait_response: Aguarda a resposta dos jogadores. Se um jogador responde, ele é marcado como tendo respondido. Se ambos os jogadores respondem, o estado retorna para running, o timer é resetado, um novo valor aleatório é atribuído a Don e os jogadores são marcados como não tendo respondido.
   - Estado penalty e penalty2: Aplica uma penalidade ao jogador que iniciou antes do timer atingir Don. A penalidade dura 1000 ciclos de clock. Após a penalidade, o estado retorna para running, o timer é resetado, um novo valor aleatório é atribuído a Don e os jogadores são marcados como não tendo respondido.
3. Saídas: O sinal de enable da luz é atribuído ao sinal enable_signal. As saídas de penalidade são '1' quando o estado correspondente é ativo e '0' caso contrário.

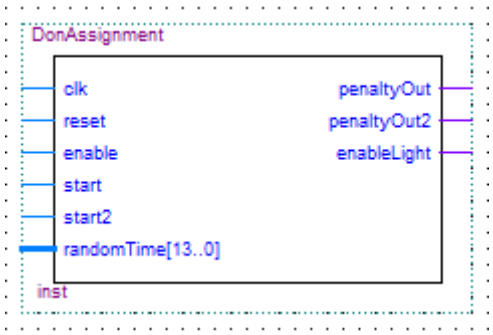

Figura 3.4: Ilustração do bloco DonAssignment

3.4 RandomTimeGen

Este bloco chamado RandomTimeGen gera um tempo aleatório entre um valor mínimo e máximo. A entidade tem várias portas de entrada e saída, incluindo sinais de clock, enable, reset e o tempo aleatório gerado.

O código implementa a seguinte lógica:

1. Reset: Se o sinal de reset é '1', o sinal temporário temp é resetado para o valor mínimo.
2. Geração de tempo aleatório: Se o sinal de clock tem uma borda de subida e o sinal de enable é '1', o valor de temp é incrementado. Se temp atinge o valor máximo, ele é resetado para o valor mínimo.
3. Saída: O valor de temp é convertido para um vetor de lógica padrão e atribuído à saída randomTime.

Os valores mínimo e máximo são definidos como constantes no início do código. Neste caso, o tempo aleatório gerado estará entre 1000 e 5000 ciclos de clock.

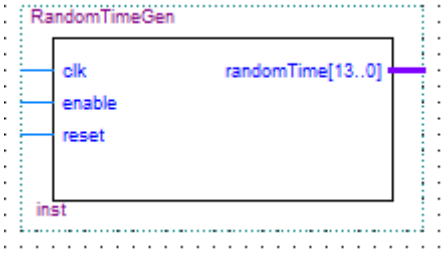

Figura 3.5: Ilustração do bloco RandomTimeGen

3.5 Init

Este bloco chamado init implementa a lógica para um contador que pode ser incrementado ou decrementado por meio de botões. A entidade tem várias portas de entrada e saída, incluindo sinais de clock, reset, enable, botões para incrementar e decrementar o contador, saídas para exibir o contador em displays de 7 segmentos e um sinal de piscar.

O código implementa a seguinte lógica:

1. Reset: Se o sinal de reset é '1', o contador é resetado para 10.
2. Incremento e decremento do contador: Se o sinal de clock de 10Hz tem uma borda de subida e o sinal de enable é '1', o contador pode ser incrementado ou decrementado. Se o botão para cima é pressionado e o contador é menor que 50, o contador é incrementado. Se o botão para baixo é pressionado e o contador é maior que 1, o contador é decrementado. Além disso, se um botão é mantido pressionado por mais de 1 segundo, o contador é incrementado ou decrementado novamente.
3. Piscar: Se o sinal de clock de 1Hz tem uma borda de subida, o sinal de piscar é invertido.
4. Saídas: O contador é convertido para um vetor de lógica padrão e atribuído à saída NumberOut. Os valores fixos "00", "01", "10"e "11"são atribuídos às saídas HEX3, HEX2, HEX1 e HEX0, respectivamente. O sinal de piscar é atribuído à saída blink.

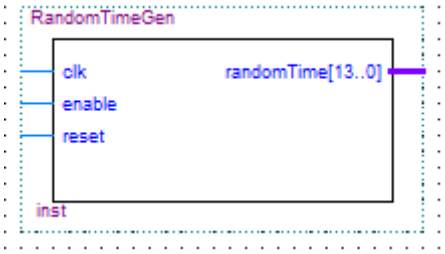

Figura 3.6: Ilustração do bloco Init

3.6 StateMachine

Este bloco chamado StateMachine implementa uma máquina de estados para um jogo. A entidade tem várias portas de entrada e saída, incluindo sinais de clock, reset, confirmação de inicialização, vitória do jogador A, vitória do jogador B, enable de diferentes estados do jogo e uma saída para selecionar um multiplexador.

O código implementa a seguinte lógica:

1. Reset: Se o sinal de reset é '1', o estado é resetado para init e o sinal confirm_prev é resetado para '0'.
2. Máquina de estados: Se o sinal de clock tem uma borda de subida, o estado anterior de Confirm_INIT é atualizado e a máquina de estados é ativada. A máquina de estados tem cinco estados:
   - Estado init: Aguarda a confirmação de inicialização. Se Confirm_INIT é '1' e confirm_prev é '0', o estado muda para gameINFO.
   - Estado gameINFO: Aguarda a confirmação de inicialização. Se Confirm_INIT é '1' e confirm_prev é '0', o estado muda para game.
   - Estado game: Verifica se algum jogador ganhou. Se winA é '1', o estado muda para win_state_A. Se winB é '1', o estado muda para win_state_B.
   - Estado win_state_A e win_state_B: Aguarda a confirmação de inicialização. Se Confirm_INIT é '1' e confirm_prev é '0', o estado retorna para init.
3. Saídas: As saídas de enable são '1' quando o estado correspondente é ativo e '0' caso contrário. O sinal confirm_prev é atribuído à saída confirm_prev_out. O valor de muxselect é determinado pelo estado atual.

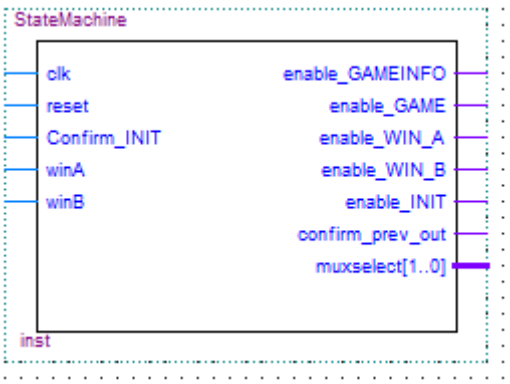

Figura 3.7: Ilustração do bloco StateMachine

3.7 MEFcompetition

Este bloco chamado MEF_competition que implementa a lógica para uma competição entre dois jogadores. A entidade tem várias portas de entrada e saída, incluindo sinais de clock, reset, confirmação, ciclo, enable, tempos dos jogadores, penalidades dos jogadores, número confirmado, pontos dos jogadores, ciclos totais, máximo de pontos, saídas para exibir em displays de 7 segmentos e sinais para indicar o vencedor.

O código implementa a seguinte lógica:

1. Reset: Se o sinal de reset ou o reset interno é '1', todos os sinais internos são resetados para seus valores iniciais.

2. Máquina de estados: Se o sinal de clock tem uma borda de subida e o sinal de enable é '1', a máquina de estados é ativada. A máquina de estados implementa a lógica da competição, incluindo a detecção de borda de subida para o sinal de ciclo, a aplicação de penalidades, a atualização dos tempos dos jogadores, a avaliação dos tempos dos jogadores, a determinação do vencedor e o reset interno quando um jogador ganha ou ambos os jogadores são desqualificados.

3. Saídas: Os pontos dos jogadores, os ciclos totais e o máximo de pontos são convertidos para vetores de lógica padrão e atribuídos às saídas correspondentes. Os valores fixos "00", "01", "10"e "11"são atribuídos às saídas HEX3, HEX2, HEX1 e HEX0, respectivamente. Os sinais internos winnerA_int e winnerB_int são atribuídos às saídas winnerA e winnerB.

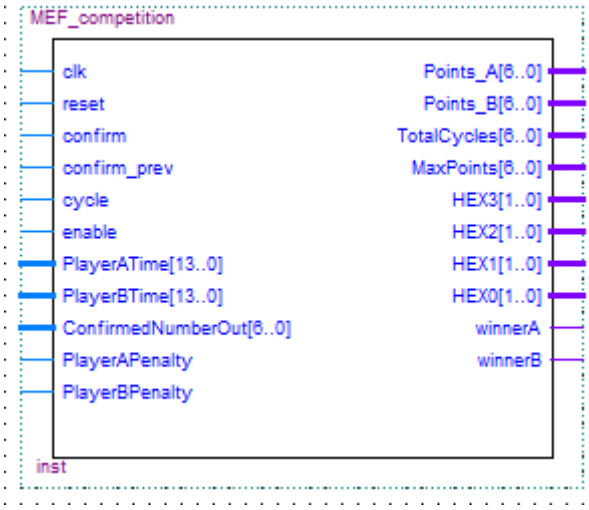

Figura 3.8: Ilustração do bloco MEFcompetition

3.8 Bin7SegDecoderEN

Este código implementa um decodificador binário para um display de 7 segmentos com uma entrada de enable. A entidade tem três portas: uma entrada binária de 4 bits (binInput), uma entrada de enable e uma saída de 7 bits (decOut_n).

O código implementa a seguinte lógica:

1. enable: Se o sinal de enable é '1', o decodificador está habilitado e a entrada binária é decodificada. Se enable é '0', todos os segmentos do display são desligados (todos os bits de decOut_n são '1').

2. Decodificação: A entrada binária é decodificada em um valor para o display de 7 segmentos. Cada valor binário de "0001"a "1111"corresponde a um número ou letra (1 a F) no display de 7 segmentos. O valor "0000"ou qualquer outro valor não listado é decodificado como 0.

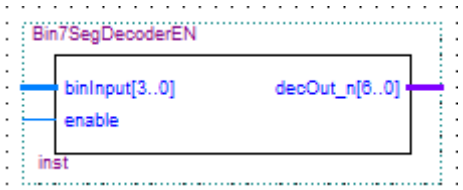

Figura 3.9: Ilustração do bloco Bin7SegDecoderEN

3.9 BinToBCD

Este código converte um número binário de 14 bits em um número BCD (Binary-Coded Decimal) de 16 bits. A entidade tem duas portas: uma entrada binária (binary_in) e uma saída BCD (bcd).

O código implementa a seguinte lógica:

1. Conversão para inteiro: A entrada binária é convertida para um número inteiro (bin_in_int).

2. Extração dos dígitos: Os dígitos das unidades, dezenas, centenas e milhares são extraídos do número inteiro. Isso é feito dividindo o número inteiro por 10, 100, 1000 e pegando o resto da divisão por 10.

3. Conversão para BCD: Os dígitos extraídos são convertidos para BCD. Cada dígito é convertido para um número binário de 4 bits e os quatro números binários são concatenados para formar o número BCD de 16 bits.

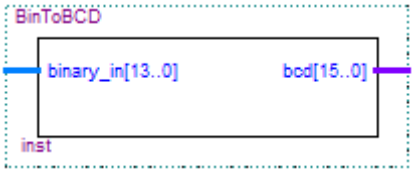

Figura 3.10: Ilustração do bloco BinToBCD

3.10 ConFDisplay

Este código implementa um decodificador binário para um display de 7 segmentos com uma entrada de enable. A entidade tem três portas: uma entrada binária de 2 bits (binInput), uma entrada de enable (enable) e uma saída de 7 bits (decOut_n).

O código implementa a seguinte lógica:

1. Enable: Se o sinal de enable (enable) é '1', o decodificador está habilitado e a entrada binária é decodificada. Se enable é '0', todos os segmentos do display são desligados (todos os bits de decOut_n são '1').

2. Decodificação: A entrada binária é decodificada em um valor para o display de 7 segmentos. Cada valor binário de "00"a "11"corresponde a uma letra (C, o, n, F) no display de 7 segmentos. Qualquer outro valor ou quando enable é '0' é decodificado como um display em branco.

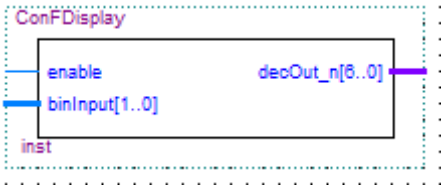

Figura 3.11: Ilustração do bloco ConFDisplay

3.11 PlayerAWin

Este código implementa um decodificador binário para um display de 7 segmentos com uma entrada de enable. A entidade tem três portas: uma entrada binária de 1 bit (binInput), uma entrada de enable (enable) e uma saída de 7 bits (decOut_n).

O código implementa a seguinte lógica:

1. Enable: Se o sinal de enable (enable) é '1', o decodificador está habilitado e a entrada binária é decodificada. Se enable é '0', todos os segmentos do display são desligados (todos os bits de decOut_n são '1').

2. Decodificação: A entrada binária é decodificada em um valor para o display de 7 segmentos. O valor binário '0' corresponde à letra P e o valor '1' corresponde à letra A no display de 7 segmentos. Qualquer outro valor ou quando enable é '0' é decodificado como um display em branco.

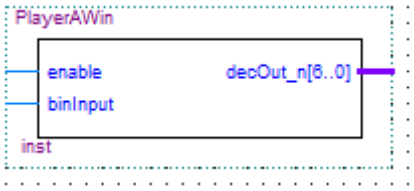

Figura 3.12: Ilustração do bloco PlayerAWin

3.12 PlayerBWin

Este código implementa um decodificador binário para um display de 7 segmentos com uma entrada de enable. A entidade tem três portas: uma entrada binária de 1 bit (binInput), uma entrada de enable (enable) e uma saída de 7 bits (decOut_n).

O código implementa a seguinte lógica:

1. Enable: Se o sinal de enable (enable) é '1', o decodificador está habilitado e a entrada binária é decodificada. Se enable é '0', todos os segmentos do display são desligados (todos os bits de decOut_n são '1').

2. Decodificação: A entrada binária é decodificada em um valor para o display de 7 segmentos. O valor binário '0' corresponde à letra P e o valor '1' corresponde à letra B no display de 7 segmentos. Qualquer outro valor ou quando enable é '0' é decodificado como um display em branco.

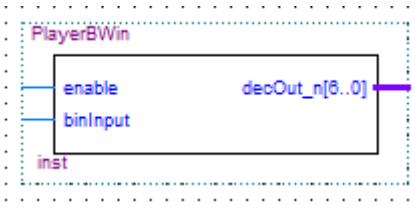

Figura 3.13: Ilustração do bloco PlayerBWin

3.13 XORPenalty

Este bloco chamado XORPenalty que implementa uma operação XOR (exclusive OR) bit a bit entre um vetor de 4 bits e um único bit replicado 4 vezes. A entidade tem três portas: uma entrada de 4 bits (input4), uma entrada de 1 bit (input1) e uma saída de 4 bits (output4).

O código implementa a seguinte lógica:

1. Replicação do bit: O bit de entrada input1 é replicado 4 vezes para formar um vetor de 4 bits (temp).

2. Operação XOR: A operação XOR é realizada bit a bit entre o vetor de entrada input4 e o vetor temp. O resultado é atribuído à saída output4.

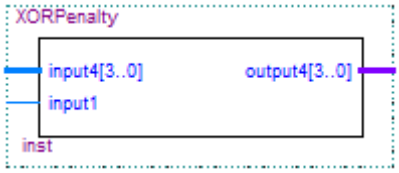

Figura 3.14: Ilustração do bloco XORPenalty

3.14 MUX4:1 7bits

Este bloco chamado MUX4:1 7bit implementa um multiplexador 4-para-1 para vetores de 7 bits. A entidade tem cinco portas: uma entrada de seleção de 2 bits (sel), quatro entradas de dados de 7 bits (i0, i1, i2, i3) e uma saída de 7 bits (MuxOut).

O código implementa a seguinte lógica:

1. Seleção: Dependendo do valor da entrada de seleção sel, uma das entradas de dados é selecionada e atribuída à saída MuxOut. Se sel é "00", i0 é selecionado. Se sel é "01", i1 é selecionado. Se sel é "10", i2 é selecionado. Se sel é "11", i3 é selecionado.

2. Caso padrão: Se sel tem qualquer outro valor, MuxOut é atribuído um valor indefinido ('X') para todos os bits.

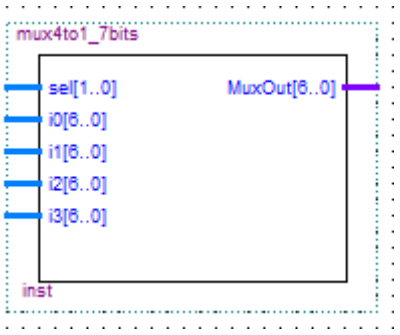

Figura 3.15: Ilustração do bloco MUX4:1 7bits

3.15 Pulse Gen

Este bloco chamado pulse_gen que gera um pulso a cada MAX ciclos de clock. A entidade tem três portas: uma entrada de clock (clk), uma entrada de reset (reset) e uma saída de pulso (pulse). O valor de MAX é um parâmetro genérico com um valor padrão de 50.000.

O código implementa a seguinte lógica:

1. Contagem: Em cada borda de subida do clock (clk), o contador s_cnt é incrementado, a menos que o sinal de reset esteja ativo. Se reset é '1', o contador é resetado para 0.

2. Geração de pulso: A saída pulse é normalmente '0'. Quando o contador atinge MAX-1, ele é resetado para 0 e um pulso é gerado (ou seja, pulse é definido como '1' por um ciclo de clock).

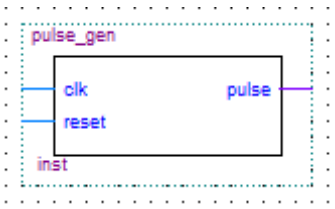

Figura 3.16: Ilustração do bloco pulse gen

3.16 tEStDisplay

Este bloco chamado tEStDisplay que implementa um decodificador binário para um display de 7 segmentos com uma entrada de enable. A entidade tem três portas: uma entrada binária de 2 bits (binInput), uma entrada de enable (enable) e uma saída de 7 bits (decOut_n).

O código implementa a seguinte lógica:

1. Enable: Se o sinal de enable (enable) é '1', o decodificador está habilitado e a entrada binária é decodificada. Se enable é '0', todos os segmentos do display são desligados (todos os bits de decOut_n são '1').

2. Decodificação: A entrada binária é decodificada em um valor para o display de 7 segmentos. Cada valor binário de "00"a "11"corresponde a uma letra (t, E, S, t) no display de 7 segmentos. Qualquer outro valor ou quando enable é '0' é decodificado como um display em branco.

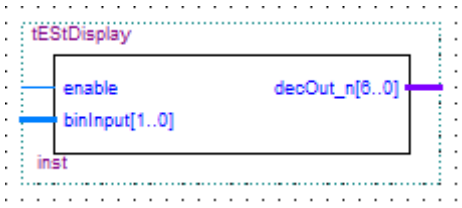

Figura 3.17: Ilustração do bloco tEStDisplay

3.17 win lights

Este bloco chamado win_lights que controla um conjunto de 18 LEDs. A entidade tem três portas: um sinal de clock (clk), uma entrada de habilitação (enable) e uma saída para os LEDs (led).

O código implementa a seguinte lógica:

1. Inicialização: O estado inicial dos LEDs (led_state) é definido como '0' para todos os 18 LEDs.

2. Processo: A cada borda de subida do sinal de clock (clk), se o sinal de habilitação (enable) é '1', o estado dos LEDs é invertido (todos os LEDs que estavam ligados são desligados e vice-versa).

3. Saída: O estado atual dos LEDs é atribuído à saída led.

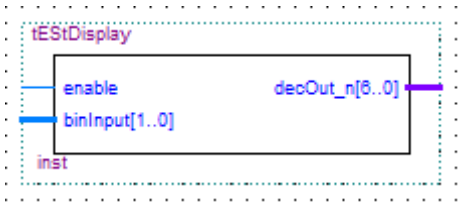

Figura 3.18: Ilustração do bloco win lights

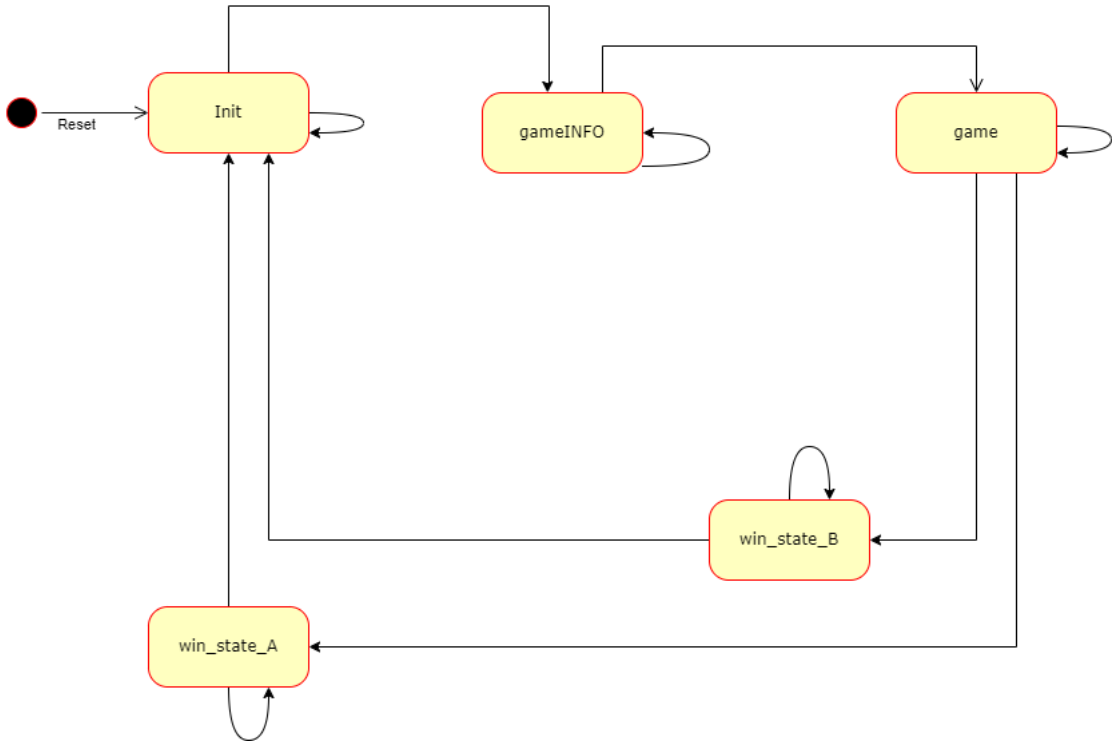

Figura 3.19: Esquema da Máquina de Estados

1. **Reset**
   ↪ A máquina volta sempre ao estado de Reset quando o botão 'Reset' é pressionado.

2. **(Init) → gameINFO**
   ↪ Após o utilizador ajustar a pontuação e clicar no botâo de confirmação, a máquina passa do estado init para gameINFO.

3. **gameINFO → game**
   ↪ Após o utilizador pressionar no botão Start a máquina passa para o estado game.

4. **game → win_state_B/win_state_A**
   ↪ Dependendo do utilizador que atinja primeiro a pontuação que foi definida no início, a máquina de estados transita para o estado win_state_B/win_state_A.

5. **Standby → Delay**
   ↪ Quando o botão de 'Start/Stop' é pressionado, muda de estado para 'Delay' começando o timer do atraso inicial com o valor escolhido.

6. **win_state_B/win_state_A→ init**
   ↪ Se o utilizador pretender recomeçar o jogo basta pressionar a tecla.

Capítulo 4

Validações

No decorrer do nosso projeto fomos confrontados com diversas adversidades no que toca a simulação e validação. Como tal, a principal maneira de verificação foi prática e feita com a placa, já que o trabalho funciona maioritariamente em segundos, um tempo dificil de se trabalhar tanto no simulador, como na testbench.

Capítulo 5

Conclusões e Contribuições

5.1 Conclusões

Após uma breve reflexão observámos que com este trabalho foram desenvolvidas novas capacidades em VHDL, lógica de estruturação (aplicada durante o planeamento das funções), otimização (de forma a simplificar o trabalho da melhor forma possivel) e ainda capacidades a nível de trabalho em grupo. Vimos também algumas das capacidades da placa e o potencial da disciplina.

Autoavaliamos o nosso trabalho com 16 valores.

5.2 Contribuições dos autores

Neste projeto, uma vez que houve um maior empenho pela parte do elemento Tiago Pita , este mesmo elemento tem uma percentagem de 70% enquanto o elemento Rafael Marques , uma percentagem de 30%.

Capítulo 6

Referências

Estruturas de código baseadas em respostas do OpenAI. 2024. ChatGPT. Acesso em 5 de junho de 2024. [https://www.openai.com/chatgpt](Este link disponibiliza imagens das respostas baseadas= `https://imgur.com/a/TuYMjsF`).

Acrónimos

LSD Laboratório de Sistemas Digitais

VHDL VHSIC Hardware Description Language