PCS3115 - Sistemas Digitais I
Projeto 2 - Jokempô++

Glauber De Bona

Prazo: 16/06/2021

O objetivo deste trabalho é a prática da descrição de circuitos combinatórios em VHDL para correção de erros e a familiarização com flip-flops e seu uso em projetos.

Introdução

O sistema para jogar Jokempô em um disputa melhor de três foi um sucesso, e os dois amigos rivais estão montando uma startup, chamada de JKP^3, para comercializá-lo. Para isso, contrataram você para ajudar a melhorar o sistema, tornando-o robusto a pequenos erros e à falta de sincronia entre os dois jogadores.

Veja o enunciado do Projeto 1.

Um problema notado foi que os gestos que chegavam ao sistema nem sempre correspondiam aos escolhidos por jogadores, devido a erros de 1 bit, que precisarão ser corrigidos. Usando código de Hamming, alguns bits extras serão inseridos na saída do teclado que captura os três gestos, e sua tarefa será corrigir os erros na entrada do sistema.

Outra problema a ser corrigido é a necessidade de os dois jogadores terem que inserir os gestos em um mesmo momento para ver o resultado, que só vale para aquele momento. Para contornar isso, você nota que os gestos inseridos poderiam ser armazenados de alguma forma, daí a necessidade de componentes sequenciais. Então, os fundadores da JKP^3 encomendaram um lote de flip-flops, descritos pelo código VHDL abaixo:

```
entity flipflopd is
  port(
    D, reset, clock, EN: in  bit;
    Q:                   out bit
  );
end flipflopd;

architecture behavior of flipflopd is
begin
  process (reset, clock)
  begin
    if reset='0' then
      Q <= '0';
    elsif clock'EVENT and clock='1' and EN='1' then
      Q <= D;
    end if;
  end process ;
end behavior;
```

Sua tarefa será usar estes componentes, juntamente com circuitos combinatórios, para fazer com que os adversários possam inserir seus gestos em momentos diferentes e para permitir que o resultado

na saída perdure até que novos gestos sejam inseridos por ambos os jogadores. **Atenção**: A menos dos flip-flops fornecidos, todo o projeto deve ser combinatório!

Atividades

Trabalho 2, Atividade 1

T2A1 (3 pontos) Os 3 gestos de cada jogador são representados por 6 bits, então precisamos de 4 bits adicionais de paridade (assumimos a convenção de **paridade PAR**) para construir um código de Hamming com distância 3, que permite corrigir erros de 1 bit. Implemente um componente em VHDL (com a entidade abaixo) que receba estes 10 bits em `entrada` e forneça na saída `dados` os 6 bits dos gestos correspondentes, corrigindo eventuais erros de 1 bit. Assuma que os 4 bits menos significativos são bits de paridade (p_8, p_4, p_2 e p_1). Adicionalmente, há a saída `erro`, que deve ser igual a 1 somente quando não for possível corrigir o erro mudando apenas 1 bit.:

Por exemplo, se os todos bits de paridade estão errados, corrigiríamos o bit da posição 15, mas a entrada possui apenas 10 bits.

```
entity hamming is
  port(
    entrada: in  bit_vector(9 downto 0);    --! 3 gestos mais 4 bits de paridade
    dados : out  bit_vector(5 downto 0);    --! 3 gestos, corrigindo erros de 1 bit
    erro:   out  bit                        --! erro nao corrigido
  );
end hamming;
```

Trabalho 2, Atividade 2

T2A2 (3 pontos) Projete, seguindo a entidade abaixo, um versão melhorada do `jokempotriplo` (veja Projeto 1) que guarde os gestos inseridos pelos jogadores e o resultado da disputa utilizando instâncias do Flip-Flop descrito acima. A entrada `clock` é o sinal de relógio, a ser ligado nos flip-flops. As entradas `loadA` e `loadB`, botões ativos em alto, determinam quando os gestos do jogador A e do jogador B devem ser armazenados. A saída `z`, o resultado da disputa, deve ser atualizada, na borda de subida do `clock`, apenas quando a entrada `atualiza` (outro botão) for igual a 1, neste caso refletindo o resultado da disputa entre os gestos armazenados. A entrada `reset` deve ser ligada ao `reset` assíncrono dos flip-flops que guardam os gestos e o resultado. **Dica:** Utilize o componente `jokempotriplo` do Projeto 1.

Assuma que os gestos já chegam corrigidos.

O armazenamento deve ocorrer na borda de subida do `clock` sempre que, e apenas quando, o sinal de carga correspondente estiver ativo.

```
entity jkp3 is
  port(
    reset, clock: in  bit;
    loadA, loadB: in  bit;                      --! armazenam os gestos
    atualiza:     in  bit;                      --! atualiza o resultado z
    a1, a2, a3:   in  bit_vector(1 downto 0);   --! gestos do jogador A para os 3 jogos
    b1, b2, b3:   in  bit_vector(1 downto 0);   --! gestos do jogador B para os 3 jogos
    z:            out bit_vector(1 downto 0)    --! resultado da disputa
  );
end jkp3;
```

Trabalho 2, Atividade 3

T2A3 (4 pontos) Após os primeiros testes com o `jkp3`, os amigos fundadores da JKP^3 se indagaram por que apertar `atualiza` era necessário após os dois jogadores já terem carregado seus gestos. Adicione a seguinte funcionalidade ao `jkp3`: toda vez que os dois jogadores já apertaram e largaram seus botões de carregamento (`loadA` e `loadB`), a saída `z` é atualizada automaticamente, sem a necessidade de apertar `atualiza`. A saída não deve ser atualizada automaticamente se algum sinal de carga ainda estiver ativo. **Dica**: Utilize flip-flops extras que indiquem quando um botão já foi apertado.

Podemos assumir que a frequência de `clock` é alta o suficiente para haver pelo menos uma borda de subida enquanto um botão é manualmente apertado.

```
entity jkp3auto is
  port(
    reset, clock: in  bit;
    loadA, loadB: in  bit;                          --! armazenam os gestos
    atualiza:     in  bit;                          --! atualiza resultado z
    a1, a2, a3:   in  bit_vector(1 downto 0);       --! gesto do jogador A para os 3 jogos
    b1, b2, b3:   in  bit_vector(1 downto 0);       --! gesto do jogador B para os 3 jogos
    z:            out bit_vector(1 downto 0)        --! resultado da disputa
  );
end jkp3auto;
```

Instruções para Entrega

Você deve acessar o link específico para cada tarefa (T2A1, T2A2 e T2A3) dentro do tópico "Projetos" no e-Disciplinas, já logado com seu usuário e senha, que levará à página apropriada do juiz eletrônico. Em cada atividade, você pode enviar apenas um único arquivo codificado em UTF-8. O nome do arquivo não importa, mas sim a descrição VHDL que está dentro. As entidades nas suas soluções devem ser idênticas àquelas neste enunciado ou o juiz não irá processar seu arquivo. **Importante: Seu arquivo NÃO deve incluir o código do flip-flop D fornecido, que já estará no juiz.**

Quando acessar o link no e-Disciplinas, o navegador abrirá uma janela para envio do arquivo. Selecione-o e envie para o juiz. Jamais recarregue a página de submissão pois seu navegador pode enviar o arquivo novamente, o que vai ser considerado pelo juiz como um novo envio e pode prejudicar sua nota final. Caso desista do envio, simplesmente feche a janela. Depois do envio, a página carregará automaticamente o resultado do juiz, quando você poderá fechar a janela. A nota dada pelo juiz é somente para a submissão que acabou de fazer.

O prazo para a submissão das soluções no Juiz é 16 de junho de 2021, quarta-feira, às 23:59. O Juiz aceitará até 5 submissões para cada atividade deste projeto. Sua submissão será corrigida imediatamente e sua nota será apresentada. A maior nota dentre as submissões será considerada. Neste trabalho, os problemas valem no máximo 10 pontos no juiz, porém a nota final deste trabalho será calculada com as ponderações indicadas em cada atividade neste enunciado, totalizando 10 para o trabalho todo. Como boa prática de engenharia, faça seus *test-benches* e utilize o GHDL ou o EDA Playground para validar suas soluções antes de postá-las no juiz.

Atenção: Para as três atividades do projeto, está proibido o uso de `process` ou das bibliotecas `std_logic_1164` e `textio`, ou de qualquer biblioteca não padronizada. Se seu arquivo mencionar essas bibliotecas, mesmo em um comentário, sua submissão nem será avaliada pelo juiz e ficará com nota 0 (zero).