PCS3115 - Sistemas Digitais I - Trabalho 4

por Bruno de Carvalho Albertini

08/10/2020

Este trabalho consiste em desenvolver um corretor de memórias usando o código de Hamming, em VHDL.

Introdução

Com o avanço da microeletrônica, as memórias se tornaram mais densas, possibilitando o aumento da quantidade de memória RAM disponível para o processador. Contudo, tal avanço implica que as memórias estão susceptíveis a erros, provenientes de vários tipos de interferência. A mais comum é a interferência por raios cósmicos, cujos neutrôns são quase impossíveis de barrar. Estas interferências podem mudar um bit na memória de zero para um ou vice-versa.

Mais bits de memória por área.

Em aplicações comuns, é aceitável a ocorrência de alguns erros de vez em quando, mas algumas aplicações necessitam de uma taxa de erros próxima de zero. Imagine por exemplo o seu saldo do banco mudado de 1 bit: se fosse um bit representando o bit menos significativo, você provavelmente não perceberia, mas um bit no mais significativo transformaria seu saldo em negativo! Em aplicações médicas, militares e críticas, erros em um bit podem ser catastróficos, mesmo se ocorrerem de vez em quando.

Para mitigar este problema, há uma classe de memória que suporta a correção de erros. Os dados são sempre armazenados codificados de forma que o código indique a presença de erros. Um tipo muito comum é a memória SECDED (*Single Error Correction, Double Error Detection*), que utiliza o código de Hamming.

Estas memórias são chamadas de Memórias ECC (*Error Correction Code*).

O código de Hamming é um código de distância 3, o que significa que ele pode ser usado para corrigir um erro **OU** detectar até dois erros. Para ser usado como SECDED, o código precisa ser corretor de um erro **E** detector de dois erros. Consegue-se isso adicionando um bit de paridade extra, sempre na posição mais significativa. O Hamming(7,4), por exemplo, se torna um Hamming(8,4), sendo o bit extra a paridade dos outros bits restantes. Sem o bit extra, quando acontecerem dois erros, o código de Hamming corrigirá a palavra para uma palavra inválida, pois só é possível corrigir um erro. No entanto, a paridade extra do SECDED é usada para detectar esta situação, pois verifica a palavra como um todo. A URL abaixo aponta para um *whitepaper* da Xilinx explicando uma implementação muito comum do SECDED, o Hamming(22,16).

Recomenda-se ler o PDF da Xilinx na íntegra antes de continuar.

`https://www.xilinx.com/support/documentation/application_notes/xapp383.pdf`

Atividades

T4A1 Desenvolva um codificador de Hamming(22,16) exatamente como no *whitepaper*, seguindo a entidade abaixo:

Trabalho 4, Atividade 1, 5 envios, maior nota, 3 pontos

```
entity secded_enc16 is
  port (
    u_data: in bit_vector(15 downto 0);
    mem_data: out bit_vector(21 downto 0)
  );
end entity;
```

Figura 1: Listagem: Entidade VHDL para o codificador 16:21.

O seu codificador é utilizado quando o processador escreve na memória. O dado que o processador quer escrever é passado para o seu módulo através da entrada `u_data` e o dado efetivamente escrito na memória é a saída do seu módulo `mem_data`. Note que este código possui um *overhead* pois, para guardar 16 bits úteis, são necessários guardar efetivamente 21 bits.

As memórias ECC são mais caras por isso: para guardar 16Gb de dados, são necessários 21Gb neste exemplo.

T4A2 Agora que você tem o codificador, precisa projetar o caminho contrário. A entidade para decodificar é a seguinte:

Trabalho 4, Atividade 2, 5 envios, maior nota, 4 pontos

```
entity secded_dec16 is
  port (
    mem_data: in bit_vector(21 downto 0);
    u_data: out bit_vector(15 downto 0);
    syndrome: out natural;
    two_errors: out bit;
    one_error: out bit
  );
end entity;
```

Figura 2: Listagem: Entidade VHDL para o decodificador 21:16. O tipo `natural` é um inteiro limitado ao $\mathbb{N}$. Veja mais aqui.

Este módulo é utilizado quando o processador lê um dado da memória. Os 21 bits provenientes da memória (possivelmente com um ou dois erros) são passados para o seu módulo pela entrada `mem_data` e o dado é passado para o processador pela saída `u_data`. Se não houve erros, o dado escrito é passado normalmente para o processador pois não há nada a ser corrigido. Caso você detecte um erro, você precisa corrigí-lo antes de passá-lo para o processador. Além disso, há um sinal para indicar que foi detectado um erro (`one_error`) e outro que foram detectados dois erros (`two_errors`), ambos ativos alto. Note que se houver dois erros, não é possível recuperar o conteúdo original e tanto faz o que você passar para o processador na saída `u_data`. Não é possível existir um e dois erros simultâneamente.

Por último, a síndrome (saída `syndrome`) não é uma saída presente em módulos ECC, mas foi colocada aqui para depuração. Obviamente a síndrome do código de Hamming tem uma forte ligação com a correção (ou não correção) da palavra.

O juiz não verificará a síndrome.

T4A3 (Desafio Chuck Norris) Ok, ok, já tenho um codificador e um decodificador SECDED. Mas... os decodificadores reais possuem muito mais bits que o que você fez. Na vida real, este decodificador usa um Hamming(72,64). Nesta tarefa, você deve descrever um decodificador como o da atividade T4A2, mas genérico para `data_size` bits, seguindo a entidade abaixo:

Trabalho 4, Atividade 3, 5 envios, maior nota, 3 pontos

```
entity secded_dec is
  generic(
    data_size: positive := 16
  );
  port (
    mem_data: in bit_vector(secded_message_size(data_size)-1 downto 0);
    u_data: out bit_vector(data_size-1 downto 0);
    uncorrectable_error: out bit
  );
end entity;
```

Figura 3: Listagem: Entidade VHDL para o decodificador X:Y genérico.

Este decodificador trabalha similiarmente ao da atividade anterior. De fato, se o `data_size` não for especificado ou for especificado para 16, ele se torna idêntico ao da atividade anterior. No entanto, ele pode ser usado para qualquer tamanho de dados SECDED, como 4 (Hamming(8,4)) ou 64 (Hamming(72,64)), o que o torna um projeto utilizável de verdade, em processadores comerciais. A saída `uncorrectable_error` é alta somente quando não for possível recuperar o erro na entrada.

Dica: Se você prestou atenção na entidade, ela utiliza uma função chamada (`secded_message_size`) para calcular o tamanho da mensagem codificada que vem da memória a partir do tamanho dos dados. Você precisa implementar esta função para resolver este problema. Na bibliografia da da disciplina há material sobre funções em VHDL e os professores fizeram um post também, disponível aqui.

Atenção: A solução do desafio é um projeto de verdade, utilizável na vida real em sistemas comerciais atuais. Não é um exemplo acadêmico ou didático e a dificuldade é alta. Aconselhamos a fazer somente se você está acompanhando a disciplina na íntegra e não está com problemas em outras disciplinas. Se você resolver este desafio fora do prazo estipulado, envie sua solução para o professor e sua nota será considerada (possivelmente aplicando penalidades de acordo com o atraso e com a quantidade de pessoas que entregaram no prazo).

Este projeto é vendável como IP (*Intellectual Property*) pois é o núcleo de uma memória ECC.

Instruções para Entrega

Para este trabalho estão permitidas bibliotecas `ieee.numeric_bit` e `ieee.math_real` do pacote `ieee`. Só são permitidas descrições combinatórias. A violação destas restrições acarreta nota zero automaticamente, sem direito a revisão.

Restrições, preste atenção!

Para cada atividade deste trabalho, há um *link* específico no e-Disciplinas. Acesse-o somente quando estiver confortável para enviar sua solução. Em cada atividade, você pode enviar apenas um único arquivo com sua descrição VHDL em UTF-8. O nome do arquivo não importa, mas sim a descrição que está dentro. As entidades devem ser como as especificadas ou o juiz te atribuirá nota zero.

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 antes do envio.

Depois do envio, a página carregará automaticamente o resultado do juiz, quando você poderá fechar a janela. Se não quiser esperar o resultado, feche a janela após o envio e verifique sua nota no e-Disciplinas posteriormente. A nota dada pelo juiz é somente para a submissão que acabou de fazer. Sua nota na atividade poderá ser vista no e-Disciplinas e pode diferir da nota dada pelo juiz dependendo da estratégia de atribuição de notas utilizada pelo professor que montou o problema.

Pode demorar alguns segundos até o juiz processar seu arquivo.

Atenção: não atualize a página de envio e não envie a partir de conexões instáveis (e.g. móveis) para evitar que seu arquivo chegue corrompido no juiz.