PCS3115 - Sistemas Digitais I - Trabalho 6

por Bruno de Carvalho Albertini

3/11/2020

Neste trabalho você desenvolverá uma máquina de estados finita em VHDL, partindo de uma ASM (*Algorithmic State Machine*).

Introdução

Os algoritmos permeiam a vida de toda a sociedade. Em todos os lugares você encontrará uma miríade de algoritmos, desde estatísticos até de aprendizado de máquina. A maneira mais eficiente de executá-los não é através de um programa, que será executado por um processador de uso geral, mas sim de um hardware. Quando transformamos um algoritmo em um hardware, ele executará da maneira mais eficiente possível, tanto em termos de energia como em velocidade. A desvantagem é que o hardware gerado só executará o algoritmo para o qual foi projetado, motivo pelo qual somente algoritmos muito usados ou com propósito bastante específico valem o esforço de transformá-los em hardware.

O seu celular provavelmente tem um algoritmo decodificador de áudio GSM em hardware, ou sua bateria duraria bem menos e haveria um atraso na voz dos interlocutores.

Um dos algoritmos mais antigos que conhecemos é o MDC (Máximo Divisor Comum), atribuído a Euclides, um matemático grego, que descreveu o algoritmo no livro "Os Elementos", aproximadamente em 300 A.C. Uma simplificação do algoritmo para o caso em que $a \neq 0$ e $b \neq 0$ pode ser vista abaixo:

```
def mcd(a,b):
    if a==b:
        return a
    if a>b:
        return mdc(a-b,b)
    else:
        return mdc(a,b-a)
```

Figura 1: Listagem: algoritmo de Euclides para MDC.

Quando temos um algoritmo para resolver o problema, a maneira mais fácil de implementar o hardware correspondente é usando uma ASM (*Algorithmic State Machine*). Contudo, uma máquina de estados sozinha não resolve o problema. Ainda precisamos de elementos de comparação ($A = B$ e $A > B$) e também de um subtrator.

Quando temos este tipo de projeto, uma abordagem muito comum é dividi-lo em uma unidade de controle (UC) e um fluxo de dados (FD). O FD contém todos os elementos necessários para operar sobre os dados, incluindo os elementos combinatórios como os comparadores e o subtrator mencionados. A UC é responsável por controlar o que acontece dentro do FD, portanto contém uma máquina de estados. Para este trabalho, usaremos a divisão mostrada na Figura 2.

A grande maioria das UCs são uma máquina de estados.

Na Figura 2 podemos ver dois blocos, um representando a UC e outro o FD. No FD, os dados iniciais são colocados nas entradas `A`

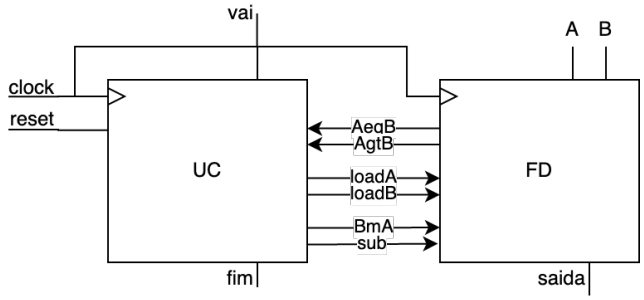

Figura 2: Diagrama de blocos da UC e do FD para este trabalho.

e `B`, e a `saida` é sempre o valor armazenado em `A`. O FD recebe um sinal de `clock`, mas não tem `reset`. Além disso, o FD produz dois sinais de controle para a UC, o `AeqB`, que é `1` quando `A=B` e `0` caso contrário, e o `AgtB`, que é `1` quando `A>B` e `0` caso contrário. O FD também recebe quatro sinais de controle da UC, que estão descritos abaixo:

`loadA`: Quando igual a `1`, na borda de subida do `clock`, guarda o valor da entrada `A` se `sub=0` ou o valor da saída do subtrator se `sub=1`.

`loadB`: Quando igual a `1`, na borda de subida do `clock`, guarda o valor da entrada `B` se `sub=0` ou o valor da saída do subtrator se `sub=1`.

O `load` tem comportamento igual para ambos os registradores. Veja o diagrama esquemático para entender como o par de multiplexadores faz a seleção do valor que é guardado.

`BmA`: Se este sinal for `1` o subtrator executa a operação `B-A`, caso contrário executa `A-B`.

`sub`: Escolhe se o valor passado para os elementos que armazenam `A` e `B` (registradores) vem das entradas (se `sub=0`) ou do subtrator (se `sub=1`).

O subtrator na verdade é combinatório e está sempre calculando a subtração `m-n`, mas no fluxo de dados há multiplexadores para escolher as entradas `m` e `n`, possibilitando a escolha proporcionada pelo sinal `BmA`.

Um diagrama esquemático interno do módulo do FD pode ser visto na Figura 3. As caixas que recebem o `clock` são chamadas de registradores e são compostos apenas por *flip-flops* que armazenam o valor na entrada na borda de subida do `clock`, se e somente se o valor do `load` for `1`, caso contrário mantém o que está armazenado. A saída dos registradores é sempre o conteúdo armazenado.

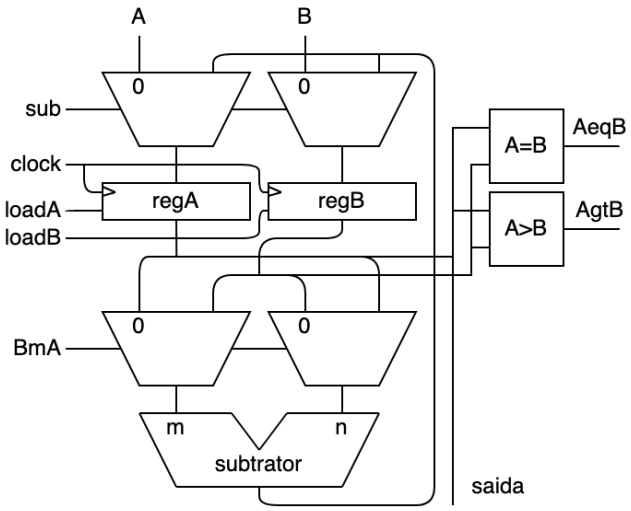

Figura 3: Diagrama esquemático do FD. Todos os elementos são combinatórios exceto os registradores, que nada mais são que um conjunto de *flip-flops* tipo D com `clock` comum e `enable` (este último é ligado no sinal de `load`).

Atividades

T6A1 Desenvolva a máquina de estados finita da UC. A entidade para a UC está abaixo. Você não precisa incluir o FD no seu projeto. Sua máquina deve ter um estado de espera inicial, que também é o estado após o `reset` (ativo em `1`). O sinal `vai`, quando em `1` libera a máquina para amostrar as entradas no FD e calcular o MDC, mas enquanto `vai=0` você deve permanecer no estado inicial. Este sinal é ativo somente na borda de transição do `clock`, quando você deve sair do estado inicial. O sinal `fim` deve ser `1` quando sua máquina terminar o cálculo, ou seja, quando o valor na `saida` for o resultado final. O sinal `fim` deve permanecer em `1` por um ciclo de `clock`, quando sua máquina deve retornar para o estado incial e aguardar uma nova sinalização do `vai`.

Trabalho 6, Atividade 1, 10 envios, maior nota, 10 pontos

```
entity gcd_uc is
  port (
    clock, reset: in bit;
    vai, AigualB, AmaiorB: in bit;
    carregaA, carregaB, BmA, sub, fim: out bit
  );
end entity;
```

Figura 4: Listagem: Entidade VHDL para a UC do T6.

Dicas:

- Não será fornecido um *testbench*. Você tem duas opções para fazer o seu próprio teste antes de enviar para o juiz:
  - Fazer um *testbench* para a máquina de estados considerando a sequencia correta de comandos que a UC deve emitir para que o FD execute corretamente o algoritmo fornecido.
  - Escrever o seu próprio FD (você pode fazer seu próprio registrador com *flip-flops* tipo D ou usar a descrição registrador no final deste arquivo) e testar com sua máquina de estados.

- A entrega é um arquivo VHDL só, contendo a descrição da sua UC conforme a entidade da Figura 4 e com ao menos uma arquitetura válida. Não envie o seu *testbench* ou FD que tenha escrito para testar.
- Este trabalho tem complexidade baixa se feito com ASM, porém média/alta se feito com máquinas de estado comuns.
- O juiz testa múltiplos cálculos de MDC, com `A` e `B` aleatórios, mas sempre diferente de zero e no máximo o valor máximo possível para o tipo `positive` na implementação do GHDL.
- Sua máquina deve honrar o `reset`, indo para o estado inicial assincronamente, e deve esperar o `vai` para começar o algoritmo. Também deve produzir o sinal de `fim` durante um ciclo de `clock` (da borda de subida até antes da próxima borda de subida), no momento onde a `saida` contém o resultado do MDC.
- Há uma tolerância em relação ao número de ciclos para executar o algoritmo, mas se sua máquina não for otimizada a sua nota pode ser prejudicada (e.g. se você construir uma máquina com muito mais estados que o necessário para este algoritmo).

O juiz considera 10% de tolerância (e.g. se para determinadas entradas a máquina ótima executa em 200 ciclos, sua máquina pode executar em até 220 ciclos).

Instruções para Entrega

Para este trabalho está permitida apenas a biblioteca `ieee.numeric_bit` do pacote `ieee`. Só há uma atividade e você pode usar o `process` mas não pode usar funções. A violação destas restrições acarreta nota zero automaticamente, sem direito a revisão.

Restrições, preste atenção!

Há um *link* específico no e-Disciplinas para a atividade deste trabalho. Acesse-o somente quando estiver confortável para enviar sua solução. 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

Pode demorar alguns segundos até o juiz processar seu arquivo.

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.

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.

```
entity reg is
  port (
    clock, load: in bit;
    entrada: in positive;
    saida: out positive
  );
end entity;
architecture ffd of reg is
  signal tmp: positive;
begin
  process(clock)
  begin
    if clock'event and clock='1' then
      if load='1' then
        tmp <= entrada;
      end if;
    end if;
  end process;
  saida <= tmp;
end architecture;
```

Figura 5: Listagem: Registrador genérico simples para números positivos. Use-o como componente caso opte por construir seu próprio FD para seus testes. Note que o tipo da informação armazenada é `positive` pois é o tipo mais adequado para este algoritmo do MDC, mas você pode trocá-lo para o tipo que desejar (e.g. `bit_vector`). A solução para o trabalho não leva em conta o tipo armazenado pois envolve somente a máquina de estados da UC. Este arquivo não deve ser enviado ao juiz.