PSI-3451 Projeto de CI Lógicos Integrados

Aula 9- Conceitos relacionados às memórias modeladas em VHDL

Nesta aula o aluno gerará no aplicativo Quartus da Altera, em forma de Megafunctions, 2 módulos VHDL de armazenamento de dados que serão utilizados no projeto ***Snake***:

- entidade *mem*: memória RAM estática baseada no componente *altsyncram* da biblioteca *altera_mf*.
- entidade *fifo*: memória FIFO baseada no componente *scfifo* da mesma biblioteca acima.

As principais características das entidades *mem* e *fifo* a serem utilizadas no projeto são apresentadas a seguir.

1. Memória RAM Estática

Dentre as muitas variantes de implementações de uma RAM estática, vamos apresentar as caracteríticas da forma proposta para o projeto Snake, uma RAM *dual-port* com leitura e escrita simultâneas. A configuração de porto único, adotada em simulações das aulas anteriores, é, na realidade, uma versão simplificada do *dual-port*, contendo um subconjunto do seu comportamento. Desta forma, vamos apresentar a forma mais genérica e avançada da RAM *dual-port*, o que permitirá também entender o caso de porta simples.

Como mostra a Figura 1, uma memória *dual-port* permite que duas escritas e leituras sejam feitas no mesmo ciclo de relógio, tudo controlado por um único clock. Para tal, existem duas interfaces (a e b) que tratam a memória como se fosse composta de duas partes separadas. Deve-se ficar claro, no entanto, que trata-se apenas de uma única memória com, com um único esquema de endereçamento, mas acessado concorrentemente por duas interfaces.

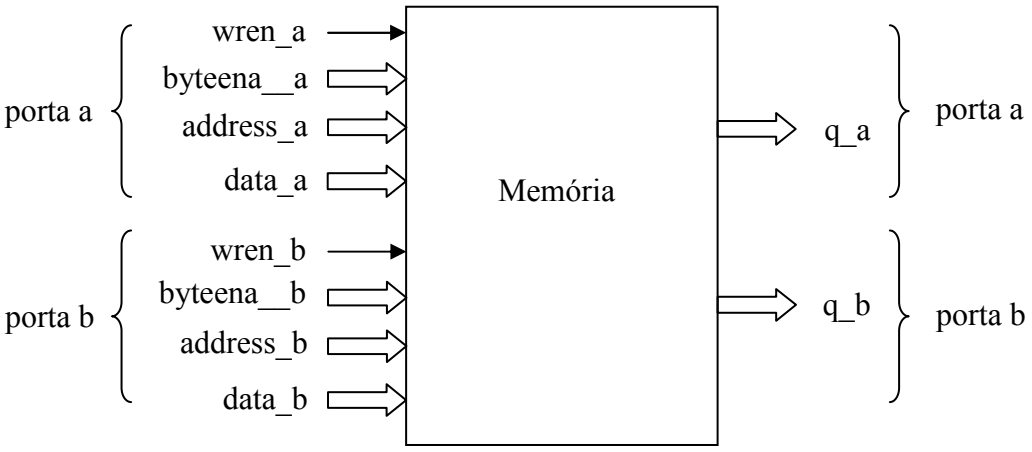


Figura 1. Entradas e saídas de uma memória *dual-port*

Pode haver duas solicitações simultâneas de escrita dos dados *data_a* e *data_b* para dois endereços *address_a* e *address_b*, respectivamente. As escritas são controladas pelos sinais de habilitação *wren_a* e *wren_b*, que são ativados quando iguais a '1'. A todo ciclo de relógio, independente dos valores de wren_a ou wren_b, os dados correspondentes aos endereços de memória são lidos nas saídas q_a e q_b correspondentes. Existe um sinal que permite a habilitação da escrita em bytes dos dados separadamente, caso exista mais de um byte. Trata-se do sinal *byteena_a* e *byteena_b* que são de tamanho *k* bits para palavras (dados) de *k* bytes; a habilitação dos bits específicos (em '1'), faz com que apenas o(s) byte(s) correspondentes sejam escritos. Observe-se que para dados de tamanho menor que 1 byte, é esperado que o *byteena* seja de 1 bit e sempre de

valor '1' (caso contrário, o resultado poderia ser contraditório ao *wren*). Quando a memória é gerada como uma Megafunction do Quartus/Altera, o sinal de byteena pode ser produzido automaticamente já no tamanho correspondente ao tamanho da palavra de dados.

Existem alguns aspectos particulares do comportamento das memórias *dual-port* que necessitam ser definidos. A primeira questão refere-se à escrita e leitura simultânea, ou seja, na necessidade de se definir qual é o valor lido em q_a quando ocorre, no mesmo ciclo, a escrita em data_a (o mesmo vale para o porto b). Nas memórias da Megafunctions do Quartus/Altera, a saída q_a assume o valor que está sendo escrito no mesmo ciclo de relógio.

Um segundo problema a ser esclarecido é quando o mesmo endereço de memória está sendo considerado para as duas portas (o mesmo endereço de memória em address_a e adress_b). Para isto, existem mais dois casos a considerar:

- a escrita simultânea de dados distintos em data_a e data_b: normalmente, em uma memória física (implementada em silício), o valor escrito seria a do dado mais atrasado (último a chegar na célula de memória). Uma vez que, na implementação em Megafunctions do Quartus/Altera, não se pode garantir se a escrita pelo porto *a* é mais rápida ou mais lenta que no porto *b* (ou vice-versa), o valor escrito nesta condição é 'X'.

- a escrita em no porto *a* e a leitura no porto *b* (a situação com a troca dos portos seguiria o mesmo princípio): a saída com a leitura no outro porto *b* pode ser programada como 'X' ou com o valor de data_b anterior. No caso do Snake, a segunda opção é usada para evitar-se a indefinição.

2. Memória FIFO (First-In First-Out)

Uma FIFO é um tipo particular de implementação de memória onde a forma de escrita e leitura já está pré-definida e independe do endereçamento (na prática, há um endereçamento interno dentro do bloco FIFO, mas ele é transparente ao usuário). A FIFO representa um fila com entrada e saída separadas, onde todo dado que entra, caminha na fila em direção à saída. Desta forma, o primeiro a entrar é também o primeiro a sair, daí o seu nome em inglês, First-In First-Out. Em geral a FIFO é utilizada em uma situação de transferência de dados onde a entidade de envio está conectada à entrada da FIFO enquanto a entidade de recepção está conectada à sua saída.

A Figura 2 ilustra uma FIFO com as entidades de envio e de recepção. Uma FIFO tem a função de *buffer*, correspondendo a uma solução apropriada quando não há uma correlação entre os ritmos de envio e de recepção de dados. Para assimilar este descompasso, a memória é projetada para armazenar uma quantidade variável de dados no tempo. Quanto maior o descompasso, maior deve ser a capacidade da memória a fim de se evitar paradas na transferência de dados decorrentes de congestionamentos. Uma FIFO tem um tamanho finito, portanto deve-se ter sinais de controle para que situações especiais possam ser tratadas. Por exemplo, a entidade de envio deve ter a informação de buffer cheio para evitar que mais dados sejam encaminhados à FIFO sob pena de sobrescrever dados válidos; da mesma forma, a entidade de recepção deve ter a informação de buffer vazio para evitar que mais dados sejam recebidos da FIFO sob pena de obter dados inválidos.

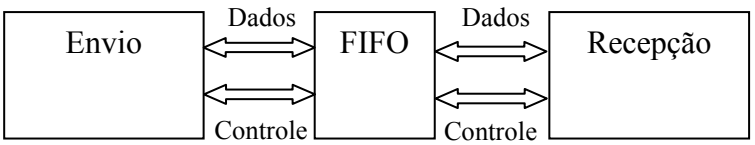


Figura 2. A FIFO como meio de transferência de dados

A Figura 3 ilustra os detalhes de um bloco FIFO onde fica evidenciada a presença da memória. No *Snake* utilizaremos uma implementação síncrona, significando que a escrita e a leitura da memória são coordenadas por um único sinal de relógio. O bloco tem a função de contabilizar a quantidade de dados armazenados e informar as condições limites de cheio e vazio. Para que a FIFO seja preenchida com os dados de forma correta e a quantidade de espaço disponível contabilizada, deve-se disponibilizar ponteiros (registradores) de escrita e de leitura. O primeiro indica o endereço vago para a próxima escrita e o segundo indica o endereço com conteúdo válido para a próxima leitura.

Durante os processos de leitura e/ou escrita, o conteúdo dos dois ponteiros podem tornar-se iguais, significando as duas seguintes condições: 1) caso de memória cheia, full='1', desde que o último evento tenha sido uma escrita; 2) caso de memória vazia, empty='1', desde que o último evento tenha sido uma leitura.

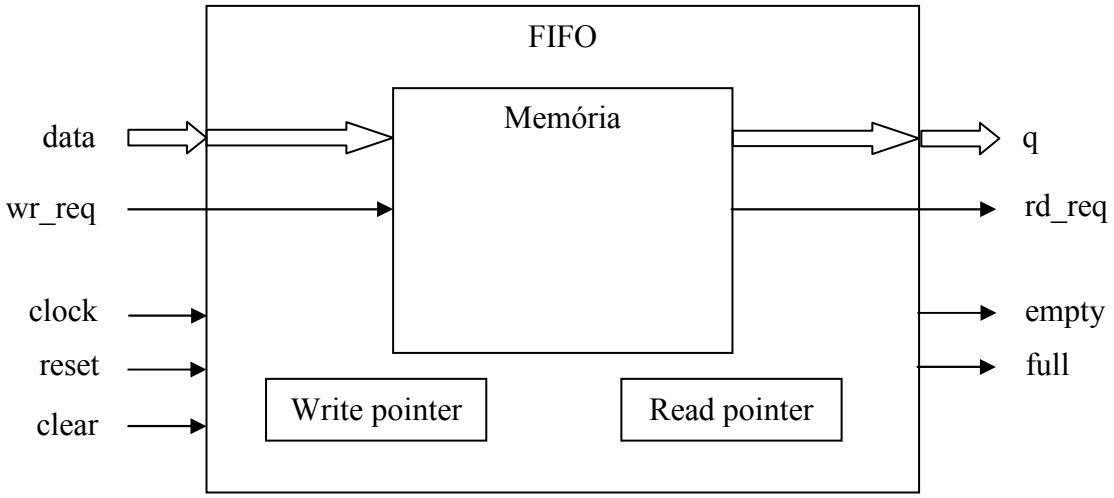


Figura 3. Visão da FIFO em detalhes

3. Ambiente de projeto Quartus II (versão 9.1)

O software Quartus II® da ALTERA® (atualmente, Intel) é um ambiente de trabalho para todas as etapas de projeto de um circuito integrado baseado em PLDs e FPGAs. Há duas atividades principais envolvidas: a síntese do circuito e a simulação das descrições iniciais e demais resultantes das etapas de síntese.

A tarefa de síntese automática é uma das mais relevantes devido à alta complexidade dos circuitos desenvolvidos atualmente e das inúmeras possíveis soluções existentes para a construção do circuito final, objetivando-se sempre ganhos de eficiência em área ocupada ou velocidade do circuito. A atividade de simulação permite avaliar esta exploração de soluções neste problema de otimização, permitindo testar e medir os resultados obtidos nos diversos cenários.

3.1.Compilação

Na compilação, a ferramenta transforma a descrição inicial (VHDL) e, na sequência, as demais resultantes das etapas intermediárias do fluxo de projeto, em códigos que permitem a simulação do circuito e, finalmente, a gravação do dispositivo. Há basicamente duas formas de se efetuar a

compilação: funcional e completa. A compilação funcional verifica a sintaxe e a coerência do projeto e gera um arquivo que poderá ser simulado, mas não mostrará os atrasos de propagação nem eventuais problemas de *setup time* e *hold time*. É equivalente à realização de uma síntese lógica independente da tecnologia. Este tipo de compilação serve para, em projetos maiores onde a compilação completa pode levar muito tempo, verificar rapidamente se o circuito se comporta como o esperado. A compilação completa, além de executar as tarefas da compilação funcional, executa o *fitting* do circuito, ou seja, adapta o projeto ao dispositivo físico sobre o qual ele será implementado (posicionamento e roteamento); ainda gera os arquivos necessários para fazer a análise de timing e a gravação do circuito. Com a compilação completa realizada pode-se fazer uma simulação muito mais realista, considerando todos os atrasos do dispositivo físico.

O Quartus permite que se incorpore funções ou IP cores de bibliotecas. As fornecidas pela própria Altera são conhecidas como Megafunctions e as suas descrições para síntese e simulação podem ser geradas a partir do Menu de ferramentas (Tools) do Quartus II, com a opção de MegaWizard Plug-in Manager. As seguintes classes de IPs estão disponíveis no catálago:

- Basic functions
- DSP functions
- Interface protocols
- Low power functions
- Memory interfaces and controllers
- Processors and peripherals

Para cada classe, vários dispositivos são possíveis; por exemplo, para memórias, podemos ter FIFO, memória de porta-única, dupla, etc. De acordo com as configurações definidas pelo usuário, o arquivo VHDL correspondente é gerado com os seus portos nos tamanhos selecionados. Na verdade o arquivo VHDL instanciará o componente de biblioteca correspondente ao IP, este contendo arquivos de descrição e de comportamento para a simulação e síntese. Tal componente instanciado terá portos que seguirão o formato configurado; além disto, os seus generics poderão ser modificados (dentro dos valores pré-establecidos) pelo usuário para realizar outras variações do IP, se assim desejar.

3.2.Simulação

O Quartus II® permite que a simulação seja feita por vários simuladores, inclusive o Modelsim. Para isto, há um conjunto de aplicações concentrados no NativeLink, pacote que realiza as transformações e transferências automáticas dos arquivos de simulação presentes no Quartus para o Modelsim. O mesmo esquema de testbench utilizado nas aulas anteriores diretamente no Modelsim poderá ser utilizado via Quartus II, facilitando a carga de arquivos que contenham IPs da Altera.

3.3.Gravação

O Quartus II® conta com um dispositivo externo (não instalado) onde acoplamos o soquete apropriado e o dispositivo a ser gravado. O software, através de um arquivo gerado pelo compilador, grava neste dispositivo as características de funcionamento designadas na etapa de projeto. A técnica de gravação e armazenamento de dados depende do dispositivo, cada família emprega uma técnica diferente (EPROM, EEPROM, SRAM, etc.). Outra forma de gravação é na própria placa onde o dispositivo é montado- para isso, o dispositivo deve ter o ISP (in-system programming) que é o caso da família CYCLONE II.