IE-344B - Tópicos em Comunicações

Leitura Complementar

Aula 5: FPGA e Fluxo de Projeto

2º Semestre/2007

Fabbryccio A. C. M. Cardoso

Marcelo Augusto Costa Fernandes

Prof. Responsável: Dalton S. Arantes

DECOM-FEEC-UNICAMP

Introdução a FPGA

Antes do advento da lógica programável, circuitos lógicos eram construídos em placas de circuitos utilizando componentes padrões, ou pela integração de portas lógicas em circuitos integrados para aplicações específicas.

FPGA é um circuito integrado que contém um grande número (na ordem de milhares) de unidades lógicas idênticas. Neste aspecto estas unidades lógicas podem ser vistas como componentes padrões que podem ser configurados independentemente e interconectados a partir de uma matriz de trilhas condutoras e *switches* programáveis. Um arquivo binário é gerado para configuração da FPGA a partir de ferramentas de software seguindo um determinado fluxo de projeto. Esse arquivo binário contém as informações necessárias para especificar a função de cada unidade lógica e para seletivamente fechar os *swiches* da matriz de interconexão. Resumindo, o *array* de unidades lógicas e a matriz de interconexão, que podem ser programados pelo usuário, formam a estrutura básica da FPGA para especificação de circuitos integrados complexos.

Na família Virtex da Xilinx a menor unidade lógica configurável é denominada *slice* de lógica e é mostrada na Figura 1. O *slice* é bastante versátil e pode ser configurado para operar como LookUp Tables (LUT) com quatro entradas e uma saída, RAMs distribuídas de 16 bits e registradores de deslocamento de 16 bits. Na operação como LUT, os recursos adicionais como D-flip flops, multiplexadores, lógica de transporte (carry) dedicado, e portas lógicas, podem ser utilizados em conjunto com as LUTs para implementar funções booleanas, multiplicadores e somadores com palavras de comprimento bastante flexível. Na operação como SRL (Shift Register LUT), estes recursos adicionais podem ser utilizados para se implementar contadores, conversores serial-paralelo e paralelo-serial, entre outras funcionalidades. Os recursos adicionais mencionados podem ainda ser utilizados na interconexão de unidades lógicas para se implementar, por exemplo, multiplicadores, contadores, somadores e memórias com praticamente qualquer comprimento de palavra. O limite para este comprimento é determinado pela quantidade de unidades lógicas disponíveis na FPGA, que é proporcional a área do circuito integrado.

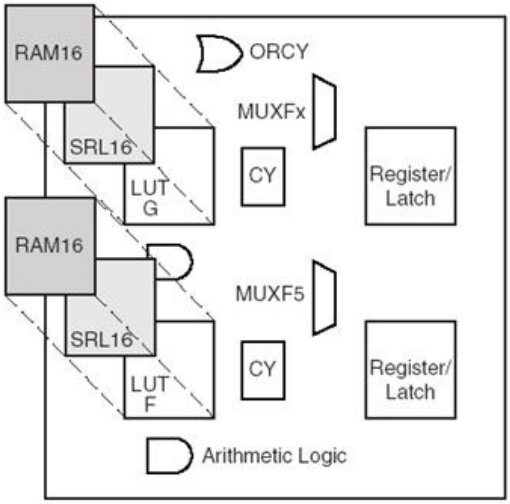

Figura 1. Slice – unidade lógica básica da família Virtex.

Além dos recursos padrões (*slices*) uma FPGA pode disponibilizar no arranjo bidimensional recursos bastante sofisticados, tais como multiplicadores dedicados, MACs programáveis (multiplicador e acumulador, também denominados de *slice XtremeDSP*), blocos de memória, DCM (Digital Clock Manager utilizados para multiplicar ou dividir a freqüência de um sinal de clock), microcontroladores (IBM PowerPC) e trans-receptores multi-giga bit (que servem para implementar interfaces seriais para transferência de bits a altíssima velocidade). A utilização de tais recursos embarcados possibilita otimizar o consumo de área (*slices*) e também desenvolver projetos mais eficientes.

Nota – *Vale destacar que o System Generator consegue abstrair recursos mais complexos e mapeá-los eficientemente para recursos básicos (primitivas) da FPGA, tornando transparente o trabalho árduo de interconectar essas primitivas. Por exemplo, o bloco "Dual Port RAM" é uma memória bastante flexível no comprimento das palavras e na profundidade da memória (número de endereços) que extrapola a capacidade individual das BRAM e das RAM distribuídas (slices padrões) disponíveis no dispositivo. Dependendo dos parâmetros escolhidos, o System Generator irá gerar um código VHDL que irá conectar várias BRAMs ou RAM distribuídas para implementar o recurso desejado.*

Field Programmable por sua vez significa que as funções da FPGA são definidas por um programa do usuário em vez de serem definidas pelo fabricante do dispositivo. Em circuitos integrados típicos (ASIC – Application Specific Integrated Circuit) a implementação é realizada no tempo da manufatura. Nas FPGAs, dependendo do dispositivo, o programa pode ser "queimado" permanentemente, semi-permanentemente como parte do processo de montagem da placa, ou carregado a partir de uma memória *flash* cada vez que o dispositivo é ligado. No último caso, a tecnologia utilizada para implementação da FPGA é a de memória estática (SRAM). Por este motivo, toda vez que o dispositivo é desligado perde-se a programação. Esta flexibilidade de programação, associada a potentes ferramentas de desenvolvimento e modelagem, possibilita ao usuário acesso a projetos de circuitos integrados complexos sem os altos custos de engenharia associados aos ASICs.

Com relação à programação da FPGA, o conjunto de ferramentas de software associado a um fluxo de projeto provê ao desenvolvedor um nível de abstração que o permite focar no algoritmo que se deseja implementar, ao invés de se preocupar com os circuitos que serão implementados. Desta forma, a programação do dispositivo pode ser feita através de uma linguagem de programação (VHDL) ou mesmo através de modelagem de sistemas (System Generator).

Fluxo de Projeto

O fluxo tradicional de projeto de circuitos para FPGA pode ser dividido em quatro fases distintas: especificação, verificação, implementação e debug de sistema. Sobre este fluxo de desenvolvimento é adicionada uma nova etapa de modelagem de sistemas e geração de código HDL através da ferramenta *System Generator*.

Uma etapa importante do projeto consiste na especificação ou geração do Netlist, que é uma descrição compacta, ou mesmo textual, do circuito para as ferramentas de verificação e de implementação de circuitos. O Netlist é basicamente uma listagem de componentes do circuito e de como estes componentes estão interconectados, incluindo ainda os nomes dos pinos de IO do chip FPGA utilizados pelo circuito. Vale destacar que a descrição do circuito realizada pelo Netlist é dependente do fabricante e da família do dispositivo empregado, uma vez que os componentes utilizados na descrição são provenientes de bibliotecas específicas deste fabricante.

A geração do Netlist pode ser feita através de captura de esquemático ou de síntese de código HDL. Um esquemático pode ser visto como uma representação gráfica de um Netlist. Deste modo, a geração do Netlist a partir da captura de esquemático é imediata. A vantagem do esquemático é facilitar o desenvolvimento do projeto de circuitos em vez de se trabalhar diretamente na descrição textual do Netlist. Por outro lado, a desvantagem do projeto concebido por esquemáticos está na portabilidade. Uma vez concebido para uma família de dispositivos de um fabricante, a migração para um dispositivo de outra família, ou mesmo de outro fabricante, pode significar o reinício de todo o projeto a partir do zero.

Por causa desta dificuldade de migração, a especificação do projeto evoluiu para uma representação comportamental e funcional do circuito através de uma linguagem de programação HDL (*hardware description language*), como o VHDL e o Verilog. Neste caso, deve ser disponibilizada uma ferramenta de síntese que interprete o código HDL e gere um Netlist otimizado, em área ou velocidade, a partir de bibliotecas específicas de componentes de um determinado fabricante.

O HDL é bastante versátil. Possibilita três níveis de abstração em HDL, como mostrado na Figura 3. O nível mais alto de abstração é o comportamental (*behavioral*), que permite descrever o comportamento do circuito através de *loops* e processos. Neste nível de abstração também é possível compor equações através de multiplicações e somas. O próximo nível de abstração possibilita descrever o funcionamento do circuito em termos de lógica combinacional (por exemplo, if, then, else) e booleana. Este nível de abstração também engloba a representação do circuito no nível de registros de transferências (RTL – *Register Transfer level*), que consiste basicamente em uma representação por registradores interligados por lógica combinacional. O nível mais baixo de abstração de um HDL é o estrutural, que consiste em uma representação do circuito semelhante a um Netlist de portas lógicas ou de *switches*.

Uma vez especificado o Netlist, pode-se entrar na fase de implementação. Isto é necessário porque o Netlist descreve apenas os componentes e como os mesmos são interconectados. Em linhas gerais, na implementação, faz-se necessário *mapear* tais componentes para células lógicas, que podem ser configuradas como look-up tables (LUT), SRL ou mesmo RAM. Em seguida, é necessário definir o posicionamento dos componentes no dispositivo de tal forma que as interconexões (roteamentos) entre os mesmos atendam às restrições de tempo especificadas. Tipicamente, estas restrições podem ser geradas automaticamente a partir das informações dos *clocks* praticados no circuito. O processo de mapeamento ainda pode ser otimizado visando minimização da área ocupada da FPGA ou a maximização da velocidade de operação do circuito.

No fluxo de projeto da Xilinx existe ainda uma etapa anterior (Tradução) que traduz o Netlist de componentes lógicos para um Netlist de primitivas da Xilinx. O objetivo é facilitar a etapa de mapeamento.

Finalizadas as etapas de tradução, mapeamento, posicionamento e roteamento (*place and route*), obtém-se um arquivo binário que pode ser baixado diretamente no dispositivo, através de uma interface *JTAG*, para a sua configuração. As demais etapas do fluxo de projeto consistem basicamente em simulações (do HDL e do Netlist) e no debug do dispositivo por meio de um analisador lógico. Vale destacar que as simulações podem refletir o funcionamento do circuito de forma bastante realista através da realimentação de informações de *timing* pelo *Timing Analizer* geradas após a etapa *Place & Route*.

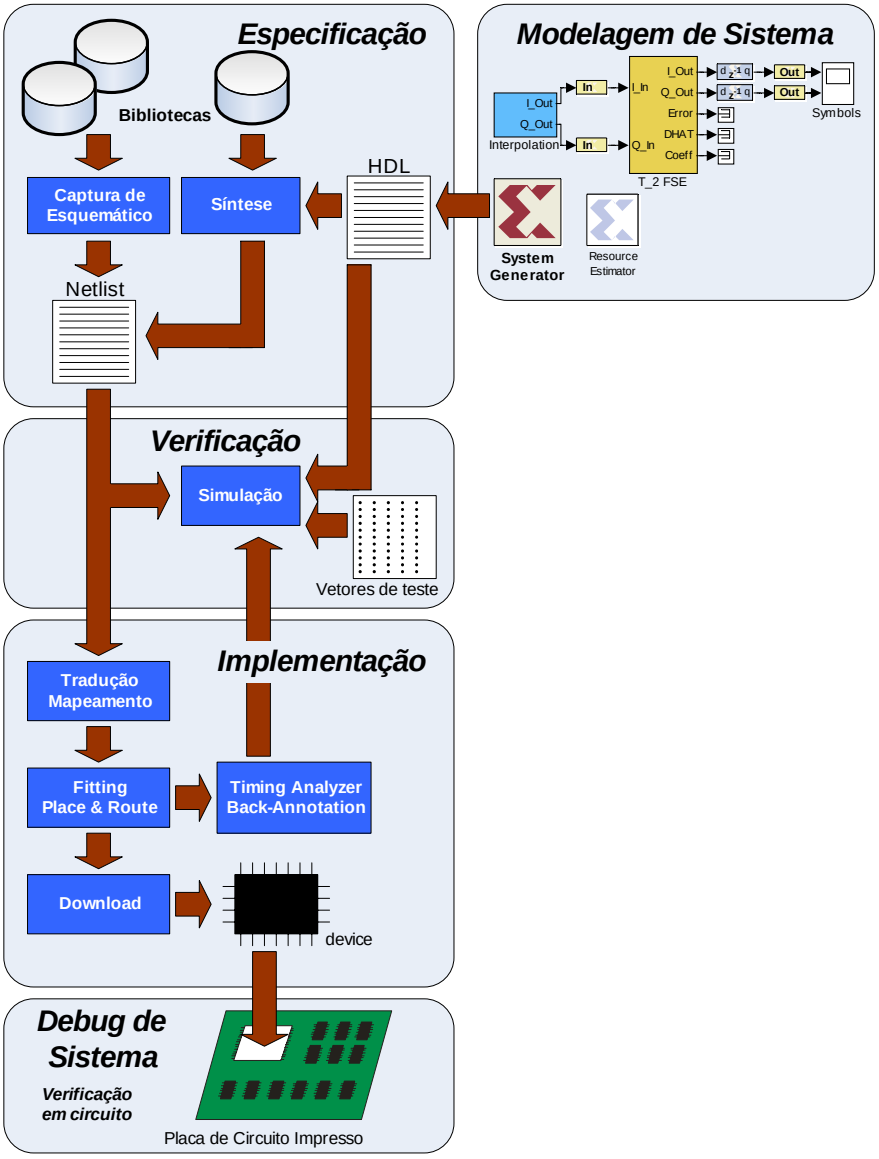

Figura 2. Fluxo de projeto de uma FPGA.

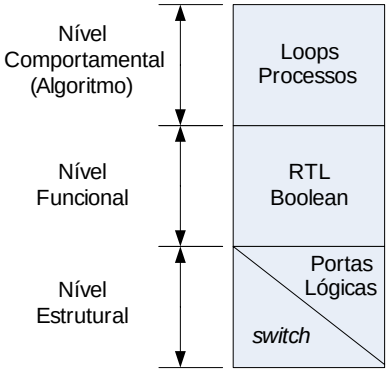

Figura 3. Níveis de abstração possíveis através de uma linguagem HDL.

A modelagem de sistemas através do System Generator representa o nível mais alto de abstração no fluxo de projeto apresentado na Figura 2. O System Generator estende as funcionalidades do Simulink para possibilitar o desenvolvimento de projetos de hardware através da geração automática de código VHDL. Através do System Generator também é possível acionar todas as ferramentas do fluxo de projeto apresentado na Figura 2. Além da abstração de funções aritméticas e de funcionalidades complexas, através do acesso a bibliotecas de IP Cores, vale destacar ainda que o System Generator possibilita acessar recursos básicos da FPGA, como as primitivas e os componentes embarcados. Por este motivo, quanto mais se conhece da arquitetura do chip utilizado maior será a possibilidade de se obter projetos mais otimizados e de melhor desempenho.

De acordo com a Xilinx, a idéia do System Generator não é substituir o desenvolvimento de projetos em HDL, mas sim focar o desenvolvimento de partes críticas relacionadas tipicamente ao processamento de sinais. Nas partes do projeto onde se envolvem interfaces externas e gerenciamento interno de clock, deve ser utilizado o HDL. Uma vantagem do System Generator é a possibilidade de importar módulos HDL em seus projetos, o que torna possível um alto nível de integração e reuso de códigos em HDL diretamente em System Generator. Outro aspecto do System Generator é a possibilidade de geração automática de *testbench* para simulação HDL (incluindo vetores de teste) e a geração de *netlists* para serem utilizados como componentes em projetos desenvolvidos em HDL. Finalmente, um recurso muito útil é a co-simulação em hardware através de interfaces tais como GigaBit Ethernet, PCI e JTAG (Cabo Paralelo IV e Plataforma USB). Este recurso de co-simulação também é denominado de "*FPGA Hardware in the Loop*" porque possibilita rodar o projeto em hardware sob o controle do Simulink, disponibilizando todo o potencial do Matlab/Simulink para análise e visualização de dados.

Representação em Ponto Fixo

O System Generator lida com sinais a partir de uma representação em ponto fixo, com ou sem sinal e com ponto binário. Na representação com sinal é utilizada a notação complemento de 2. Nesta notação, o número negativo é obtido invertendo-se os bits do número positivo e em seguida somando-se 1. Este procedimento é mostrado na Figura 4 para representações com quatro bits. Observe que a soma de um número positivo pelo seu complemento de 2 vai dar sempre zero se for descartado o "vai um" que excede o comprimento em bits da palavra. Por exemplo, +3 + (-3) = 0011 + 1101 = ~~1~~0000. Observe ainda a partir do exemplo da Figura 4 que os limites +8 e –8 têm a mesma representação, resultando que apenas um deles deve ser mantido na faixa de valores possíveis. Neste caso, o número 1000 é incluído como –8 porque o bit mais significativo é 1. Desta forma, note que o bit mais significativo pode ser utilizado para indicar o sinal do valor: 0 para positivo e 1 para negativo.

Na representação em ponto fixo do System Generator, utiliza-se também um fator de escala em potência de 2, chamado "ponto binário" (*Pb*), que possibilita a representação de números reais com precisão finita. Neste caso, vale a seguinte equação:

$$\text{valor real} = 2^{-Pb} \times \text{inteiro}\,.$$

O System Generator define dois tipos básicos Ufix_*Nb*_*Pb* e Fix_*Nb*_*Pb*, onde *Nb* é o número de bits e *Pb* é o ponto binário. Ufix representa um tipo sem sinal, enquanto Fix representa um tipo com sinal. Para o tipo Ufix_*Nb*_*Pb* a faixa de valores possíveis é

$$\left[0,\ 2^{-Pb} \times \left(2^{Nb} - 1\right)\right]$$

com passo 2^{-Pb}. Por exemplo, Ufix_4_0 representa os valores 0, 1, 2, …, 15 enquanto Ufix_4_2 representa os valores 0, 0.25, 0.5, …, 3.75. Para o tipo Fix_*Nb*_*Pb* a faixa de valores é

$$\left[-2^{-Pb} \times 2^{Nb-1},\ 2^{-Pb} \times \left(2^{Nb-1} - 1\right)\right]$$

com passo 2^{-Pb}. Por exemplo, Fix_4_0 representa os valores -8, -7, -6, …, 0, 1, 2, …, 7 enquanto Fix_4_2 representa os valores -2, -1.75, -1.5, …, 0, 0.25, 0.5, …, 1.75. Observe que o número de bits *Nb* aumenta a faixa de valores da representação numérica enquanto que o fator de escala *Pb* aumenta a precisão dessa mesma representação.

Uma vantagem da notação complemento de 2 está na possibilidade da utilização do mesmo circuito de adição para se implementar a subtração. Para mais informações sobre este tópico veja a página http://en.wikipedia.org/wiki/2%27s_complement.

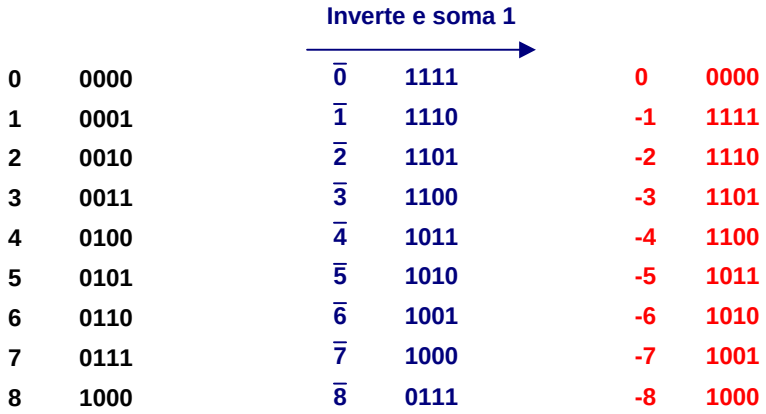

Figura 4. Obtenção do número negativo a partir do positivo na representação complemento de 2 com quatro bits.