Gerador de *testbenches* para pilhas de comunicação desenvolvidas em linguagens de descrição de hardware.

Josué P. J. de Freitas[1], Leonardo Guedes Martins[1], João Baptista dos Santos Martins[1]

[1]Grupo de Microeletrônica – Universidade Federal de Santa Maria (UFSM)

{josue.freitas,leoguema}@gmail.com,batista@inf.ufsm.br

Resumo. *Este artigo apresenta o desenvolvimento de um gerador de testbench para pilhas de comunicação desenvolvidas em linguagens de descrição de hardware (HDL, Hardware Description Language) como VHDL ou Verilog. O gerador cria a estrutura básica de um testbench para uma pilha de comunicação juntamente com os dados (nibble por nibble) de um quadro Ethernet a ser repassado para pilha de comunicação.*

1. Introdução

Os dispositivos de lógica programável (FPGA, *Field Programmable Gate-Array*) vem tornando-se nos últimos anos uma interessante alternativa para processamento de informações. Estes dispositivos são dotados de uma maior flexibilidade que os circuitos integrados (ASICs, *Application Specific Integrated Circuit*), pois podem ser reprogramados com diferentes aplicações. As aplicações com que um FPGA é programado normalmente são descritas em uma linguagem de descrição de hardware (HDL). Esta aplicação, ou descrição de hardware, é compilada gerando um *bitstream* com o qual o FPGA é programado.

Um circuito integrado também pode inicialmente ser descrito em um HDL, porém seu fluxo de concepção é diferente. Uma vez descrita e validada a descrição de hardware tem-se como objetivo gerar o *layout* do circuito que irá conter os transistores para implementação das portas lógicas que realizam a função da aplicação descrita em HDL. Com o *layout* do circuito concluído ele é enviado para fábrica (o processo de envio chama-se *tape out*) onde ele é fabricado e posteriormente enviado ao desenvolvedor para que possa ser elétricamente testado. Caso o circuito integrado apresente algum erro este só poderá ser corrigido com o envio de um *layout* corrigido para fábrica e a fabricação de um novo circuito, o que difere dos dispositivo FPGA onde para corrigir um erro basta alterar a descrição de hardware, gerar o *bitstream,* e reprogramar o dispositivo.

As principais vantagem de circuito integrado em relação aos dispositivos FPGA é o menor consumo de potência e seu custo reduzido quando justifica-se a fabricação de vários milhares até milhões de circuitos integrados. Enquanto os dispositivos FPGAs apresentam maior consumo de potência seu custo é mais reduzido quando um menor número de dispositivos (poucos milhares) são necessários, além da flexibilidade de serem reprogramáveis.

A característica em comum entre FPGAs e ASICs, e esta é a grande vantagem de desenvolver uma aplicação em hardware, é o processamento espacial paralelo destas tec [DEHON 1999]. Esta característica permite que diferentes funções sejam executadas ao mesmo tempo pois encontram-se implementadas em diferentes blocos do hardware. Assim é possível ter qualquer função matemática executada em paralelo com outra ou até mesmo duas ou mais somas, por exemplo, ao mesmo tempo desde que o hardware tenha vários somadores implementados. Também é possível visualizar a possibilidade de uma pilha de comunicação operando em modo *full-duplex* real onde diferentes blocos do hardware são responsáveis pelo envio e recebimento.

Neste contexto as pilhas de comunicação desenvolvidas em linguagens de descrição de hardware como VHDL ou Verilog tem como objetivo serem implementadas em dispositivos FPGA ou tornarem-se circuitos integrados, fazendo uso do paralelismo inerente destas tecnologias.

Soluções para redes fazendo uso de dispositivos programáveis têm crescido significativamente nos últimos anos. Em [RAO 2007] o autor realiza um levantamento de soluções Ethernet fazendo uso de FPGAs por parte da empresa fabricante de FPGAs Xilinx. As opções disponíveis no mercado, FPGAs Virtex 4 série FX e Virtex 5, atualmente incluem núcleos provendo funcionalidade Ethernet implementados diretamente dentro dos FPGAs (*hardcores*) [RAO 2007]. Além destas soluções, também é possível adquirir um FPGA sem estes núcleos fixos e programá-lo com soluções existentes da própria Xilinx [XILINX 2007] [XILINX 2006] ou soluções não proprietárias [GAO 2005] [MOHOR 2002].

Além disso, empresas como Intel e GreenField Networks, comprada recentemente pela Cisco, também apresentam soluções nesta área disponibilizando uma grande variedade de soluções para rede implementadas em circuitos integrados . A Intel possui soluções que integram Gigabit Ethernet e PHY (*PHYsical layer*) em um único circuito integrado [INTEL 2005] além de várias soluções de *Network processors* [INTEL 2007]. A GreenField Networks/Cisco possui a família G8000 Packet Processor que provê portas de conexão Gigabit Ethernet e 10 Gigabit Ethernet, suporte a IPv4 e Ipv6 e suporte a quadros do tipo Jumbo até 16K entre outras características [GREENFIELD 2006].

Além das soluções comercialmente disponíveis, como as acima citadas, também

existem vários projetos de pesquisa em universidades que fazem uso de FPGAs para conexão de redes. Dentre estes projetos é possível citar desde detectores de intrusos [PRASSANA 2005], camadas de enlace (MAC, *Medium Access Control*) provendo funcionalidades Ethernet para 100 e 1000 Mb/s [HORNA 06] e também pilhas de comunicação completas implementadas em HDL [HAMERSKI 07]. Também encontram-se trabalhos que fazem uso dos núcleos MAC comercialmente disponíveis e as demais camadas da pilha de comunicação implementadas em software como em [AGUIAR 2007].

Dado o contexto acima exposto este trabalho visa permitir um maior dinamismo para a simulação e validação de pilhas de comunicação implementadas em hardware. No projeto de hardware a etapa de simulação possui papel vital. É nesta etapa que o projeto é inicialmente validado e avaliado para aferir que o projeto está de acordo com as especificações e requisitos de desempenho, desta maneira reduzindo a chance de erro nas próximas etapas do projeto.

Um recurso bastante utilizado na etapa de simulação é o *testbench* [HAMID 2001]. *Testbenches* são algoritmos que geram sinais para estimular as entradas de um determinado projeto. Eles podem conter em sua estrutura funções para reter os sinais de saída e avaliar estes sinais para concluir que o circuito está gerando a resposta correta.

A versão inicial deste trabalho, apresentada em [MARTINS 2007], gerava apenas os dados no formato que o PHY repassa à camada de enlace (MAC). A nova versão gera um esqueleto de um *testbench*, descrito na linguagem Verilog, com os dados gerados armazenados em um vetor e prontos para serem introduzidos na camada de enlace (MAC).

A seção 2 apresenta detalhes do desenvolvimento do gerador. Na seção 3 é apresentado o status atual em que se encontra o gerador juntamente com futuras implementações. A seção 4 apresenta as conclusões até o momento e, por fim, na seção 5 as referências bibliográficas.

2. Desenvolvimento

O gerador de *testbenches* para pilha de comunicação foi desenvolvido utilizando a linguagem Perl, que é uma linguagem largamente utilizada pela indústria de hardware na etapa de simulação e validação.

O gerador, um *script* Perl, recebe parâmetros da linha de comando do usuário em um ambiente contendo Perl e gera o *testbench* em Verilog com os dados de um quadro Ethernet no formato que PHY repassa a camada de enlace.

A Tabela 1 mostra os parâmetros recebidos pelo gerador e seus significados.

Tabela 1: Parâmetros do gerador de testbench

Parâmetro	**Significado**
--macd <endereço-mac>	Endereço MAC de Destino
--macs <endereço-mac>	Endereço MAC de Origem
--ips <endereço-ip>	Endereço IP de Destino
--ipd <endereço-ip>	Endereço IP de Origem
--sp <porta>	Porta de Origem
--dp <porta>	Porta de Destino
--data <dados>	Dados binários para a camada de aplicação

A versão atual do gerador gera dados para um quadro Ethernet do tipo IP contendo um datagrama do tipo IP e um pacote UDP. O dado para a camada de aplicação encontra-se dentro do pacote UDP. Os campos referentes ao tamanho do datagrama IP (campos *Internet Header Length* e *Total Length*) e do pacote UDP (campo *Length*) são preenchidos de maneira automática pelo gerador conforme o tamanho do dado especificado pelo parâmetro --data. O valor para o --data deve sempre conter um número par de *nibbles* (4 bits) pois os protocolos IP e UDP possuem seus campos referente ao tamanho total expresso em número de octetos (bytes). O tamanho do quadro gerado varia de 46 à 1518 bytes que são os tamanhos mínimo e máximo para *Fast Ethernet*. Para o cálculo do campo FCS (*Frame Check Sequence*) no final do quadro Ethernet foi utilizado o pacote Perl Digest::CRC que pode ser facilmente encontrado e instalado.

Segundo a especificação do 802.3 [IEEE 2005] um *nibble* (4 bits) de dados deve ser entregue a camada de enlace a cada ciclo do *clock* de recebimento (rx_clk), que deve ser de 25MHz para Ethernet 100Mb/s que é o foco deste trabalho. Assim, o *testbench* criado pelo gerador fornece um *clock* nesta freqüência que é ligado ao rx_clk da camada de enlace (MAC) servindo de referência para o recebimento dos *nibbles*. Além disso, o *testbench* também deve ativar o sinal rx_data_valid para que a camada de enlace saiba que um quadro está sendo recebido.

Os dados do quadro gerado são armazenados em um vetor na sintaxe da linguagem Verilog contendo um *nibble* por posição, de modo que fazendo um uso de um contador torna-se simples o repasse destes dados para a camada de enlace. A Figura 1 ilustra o funcionamento do *testbench*.

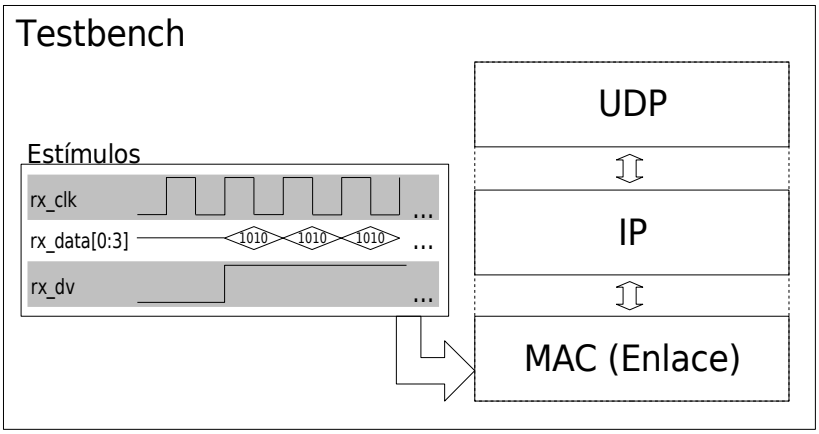

Figura 1: Funcionamento do testbench gerado

Como é possível observar na Figura 1, o *testbench* deve realizar a instanciação do projeto a ser testado (DUT, *Design Under Test*), neste caso a pilha de comunicação, e enviar os estímulos. Nota-se também que o *testbench*, no caso do nosso trabalho, realiza o mesmo papel de um PHY ao enviar os estímulos para a camada de enlace e este é justamente o comportamento esperado pelo *testbench*. A Figura 2 mostra parte do código Verilog gerado, neste código observa-se um processo para prover o *clock* adequado (linha 151) e outro processo (linha 155) que repassa a cada borda positiva de rx_clk um *nibble* para a camada de enlace através de rx_data.

```
148
149  Parameter Clock = 40; // 40ns=25MHz
150
151  always
152       #(Clock / 2) rx_clk = !rx_clk;
153
154
155  always @ (posedge rx_clk)
156       Count = Count+1;
157  initial
158   begin
159      rx_data_valid = 1;
160      rx_data = frame[Count];
161  end;
162
```

Design Summary | temp.v*

Figura 2: Código gerado para prover os estímulos à camada de enlace nos instantes corretos

3. Status atual e futuras implementações

No momento da escrita deste artigo o gerador encontra-se gerando um testbench na linguagem Verilog juntamente com os dados de um quadro Ethernet contendo um datagrama IP e um pacote UDP. Os campos referentes aos tamanhos bem como o campo de verificação de erro do quadro (FCS) são gerados corretos de maneira automática. A próxima versão do gerador permitirá gerar os campos de verificação de maneira incorreta, uma vez que o objetivo é validar uma pilha de comunicação a geração de maneira incorreta de dados também é importante.

Uma limitação do gerador de *testbench* deve-se ao fato da instanciação da pilha de comunicação não estar sendo feita de maneira automática, sendo necessário editar manualmente o arquivo Verilog gerado. Para contornar este fato a instanciação de um MAC de exemplo deverá ser adicionada na próxima versão, uma vez que a variedade soluções encontrados inviabilizam a instanciação automática sem o prévio conhecimento da declaração do componente.

```
57  //                  [IP inicio]
58  //Datagrama IP Versao 4
59       assign frame[44]=4'b0100;
60  //Internet Header Length
61       assign frame[45]=4'b0101;
62  //Type of Service
63       assign frame[46]=4'b0000;
64       assign frame[47]=4'b0000;
65  //Total Length (a ser preenchido no final)
66  //Campo ID
67       assign frame[52]=4b'0000;
68       assign frame[53]=4b'0000;
69       assign frame[54]=4b'0000;
70       assign frame[55]=4b'0000;
71  //Campo Flags + 1 bit do Fragment offset
72       assign frame[56]=4b'0100;
73  //Fragment Offset
74       assign frame[57]=4b'0000;
75       assign frame[58]=4b'0000;
76       assign frame[59]=4b'0000;
77       assign frame[60]=4b'0000;
78  //TTL=1
79       assign frame[61]=4b'0000;
80       assign frame[62]=4b'0001;
81  //Procotol Type
82       assign frame[63]=4'b0001;
83       assign frame[64]=4'b0001;
84  //Header Checksum
85       assign frame[65]=4b'0000;
86       assign frame[66]=4b'0000;
87       assign frame[67]=4b'0000;
88       assign frame[68]=4b'0000;
89  //Source Address=192.168.1.12
90       assign frame[69]=4'b1100;
91       assign frame[70]=4'b0000;
92       assign frame[71]=4'b1010;
93       assign frame[72]=4'b1000;
94       assign frame[73]=4'b0000;
95       assign frame[74]=4'b0001;
96       assign frame[75]=4'b0000;
97       assign frame[76]=4'b1100;
98  //Destination Address=200.168.3.1
99       assign frame[77]=4'b1100;
100      assign frame[78]=4'b1000;
101      assign frame[79]=4'b1010;
102      assign frame[80]=4'b1000;
103      assign frame[81]=4'b0000;
104      assign frame[82]=4'b0011;
105      assign frame[83]=4'b0000;
106      assign frame[84]=4'b0001;
107 //                  [UDP inicio]
108 //Source Port=5023
109      assign frame[85]=4'b0001;
110      assign frame[86]=4'b0011;
111      assign frame[87]=4'b1001;
112      assign frame[88]=4'b1111;
113 //Destination Port=80
114      assign frame[89]=4'b0000;
115      assign frame[90]=4'b0000;
116      assign frame[91]=4'b0101;
117      assign frame[92]=4'b0000;
118 //Length (preenchido no final)
119 //UDP Checksum
120      assign frame[97]=4'b1111;
121      assign frame[98]=4'b1111;
122      assign frame[99]=4'b1111;
123      assign frame[100]=4'b1111;
124 //Dado UDP=10101111
125      assign frame[101]=4'b1010;
126      assign frame[102]=4'b1111;
127 //UDP Length
128      assign frame[93]=4'b0000;
```

Figura 3: Atribuições dos dados do quadro Ethernet ao vetor que o representa no testbench Verilog

A Figura 3 apresenta parte dos dados gerados para o quadro Ethernet. Optou-se por exibir na Figura 3 apenas os dados referentes a IP e UDP devido a simplicidade dos campos iniciais do quadro Ethernet que tratam-se de Preâmbulo, SFD (*Start of Frame Delimiter*) e endereço MAC de destino e origem.

4. Conclusões

Este artigo apresentou e justificou a implementação de um gerador de *testbenches* para validação de pilhas de comunicação implementadas fazendo uso de alguma linguagem de descrição de hardware, como VHDL e Verilog. A versão atual gera dados para um quadro Ethernet de 100 Mb/s contendo um datagrama IP e um pacote UDP com algum dado, juntamente com demais sinais auxiliares a serem repassados para camada de enlace (MAC). Os principais dados a serem preenchidos no quadro Ethernet são parametrizados permitindo que o usuário gere *testbenches* variados para validar o seu projeto. Versões futuras do gerador deverão incluir suporte a Gigabit Ethernet e quadros do tipo Jumbo de até 9000 bytes.

5. Referências bibliográficas

[DEHON 1999] Dehon, A.; Wawrzynek, J. - Reconfigurable Computing: What, Why, and Design Automation Requirements ?, in Proceedings of the 1999 Design Automation Conference, pp. 610-615, Junho de 1999.

[RAO 2007] Rao, Navneet. Xilinx. Inc. FPGAs and Ethernet: Providing Programmability to Pervasive Interconect Standard. FPGA Journal. Disponível em http://www.fpgajournal.com/articles_2007/20070710_xilinx.htm. Julho de 2007.

[XILINX 2007] Xilinx Inc. Gigabit Ethernet MAC (GEMAC) Datasheet. Disponível em http://www.xilinx.com/ipcenter/catalog/logicore/docs/gig_eth_mac.pdf. Agosto de 2007.

[XILINX 2006] Xilinx Inc. OPB Ethernet Lite MAC Datasheet. Disponível em http://www.xilinx.com/bvdocs/ipcenter/data_sheet/opb_ethernetlite.pdf. Março de 2006.

[MOHOR 2002] Mohor, Igor. Ethernet IP Core Design Document. Ethernet MAC 10/100Mbs em OpenCores.org. Disponível em http://www.opencores.org/tmp/cvsget_cache/ethernet/doc/eth_design_document.pdf. Outubro de 2002.

[GAO 2005] Gao, Jon. Tri-mode Ethernet MAC Specifications. 10_100_1000Mbps tri-mode ethernet MAC em OpenCores.org. Disponível em http://www.opencores.org/tmp/cvsget_cache/ethernet_tri_mode/doc/Tri-

mode_Ethernet_MAC_Specifications.pdf. Janeiro de 2006.

[INTEL 2005] Intel. 82544EI Gigabit Ethernet Controller Datasheet. Disponível em http://download.intel.com/design/network/datashts/82544ei.pdf. Acessado em 27 de Julho de 2007.

[INTEL 2007] Intel. Intel Network processors. Disponível em http://www.intel.com/design/network/products/npfamily/index.htm?iid=ncdcnav2+proc_netproc. Acessado em 27 de Julho de 2007.

[GREENFIELD 2006] Greenfield Networks. G8000 Packet Processor Family.Acessado http://web.archive.org/web/20060209115841/www.greenfieldnetworks.com/index.php?page=g8000 pois o site original redireciona para a página http://newsroom.cisco.com/dlls/2006/corp_120706.html.

[PRASSANA 05] Prassana, Viktor K. e Zachary K. Baker. High-throughput linked-pattern matching for intrusion detection systems. Proceedings of the 2005 symposium on Architecture for networking and communications systems. Outubro de 2005.

[HORNA 2006] Horna, Cris Thomás; Ramos, Fábio; Barcelos, Marcelo ; Reis, Ricardo. Implementação e validação de IP soft cores para interfaces Ethernet 10/100 E 1000 MBps sobre dispoitivos reconfiguráveis. XIII Taller Iberchip/IWS'2007. Março de 2007.

[HAMERSKI 2007] Hamerski, Jean Carlo; Reckziegel, Everton; Katensmidt, Fernanda. Analysis of TCP/IP Stack Implementations in Hardware. Anais do 22nd SIM/1st SAM 2007. páginas 139-142. Maio de 2007.

[AGUIAR 2007] Aguiar, A. C. P. ; Kreutz, M. ; Santos, R. R. ; Santos, T. G. S. . Design Flow of a dedicated Computer Cluster Customized for a Distributed Genetic Algorithm Applications. IEEE 18th International Conference on Application-specific Systems, Architectures and Processors. Montreal. Julho de 2007.

[HAMID 2001] Hamid, Mujtaba. Writing Efficient Testbenches. Xilinx Application Note:Testbenches. Disponível em http://direct.xilinx.com/bvdocs/appnotes/xapp199.pdf. Junho de 2001.

[MARTINS 2007] Martins, Leonardo Guedes L.; de Freitas, Josué P. J.; Martins, João B. dos Santos. Data Generator for HDL Network Stack Validation. Anais do 22nd SIM/1st SAM páginas 163-166. Maio de 2007.

[IEEE 2005] IEEE. Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications. Standard for Information technology. Junho de 2005.