Descrição de um *Hardware* Multiplicador Parametrizável para Números Sinalizados em Complemento a Dois em Lógica Configurável

Matheus Barth Souza[1], Ewerton Artur Cappelatti [2]

Resumo

Com o aumento considerável das funções realizadas por sistemas eletrônicos digitais, muitos sistemas necessitam realizar operações matemáticas. Para executar uma multiplicação, o *hardware* deve seguir um algoritmo específico. Este trabalho apresenta o desenvolvimento de um hardware multiplicador para números sinalizados em complemento a dois utilizando FPGA. Para este *hardware,* o número de bits dos operandos pode ser informado através de um parâmetro em sua descrição VHDL. A teoria sobre hardware multiplicador, o projeto seguindo os critérios de um dado algoritmo, a validação deste módulo e os dados relativos à implementação são apresentados ao longo deste artigo.

Palavras-chave: *Hardware* multiplicador. Multiplicação de números sinalizados. Hardware parametrizável. FPGA.

Abstract

Concerning to the increase of the functions realized for digital electronic systems many systems need to do mathematical operations. To execute multiplications the hardware must follow a specific algorithm. This work presents the design of multiplier hardware for two's complement signalized numbers using FPGA. For this hardware the bit number of the operands is parametric in VHDL description. The theory about multiplier hardware, the criteria of algorithm used, the validation and information about implementation are describing along this article.

Keywords: Multiplier hardware. Signalized numbers multiplication. Parametric hardware. FPGA.

[1] Acadêmico do curso de Graduação em Engenharia Eletrônica da FEEVALE. Email: mbarthsouza@feevale.br.

[2] Professor do curso de Graduação em Engenharia Eletrônica da FEEVALE – Orientador do Projeto. Email: ewertonac@feevale.br.

Introdução

Com o crescente desenvolvimento dos sistemas eletrônicos digitais e com o aumento considerável das funções realizadas por estes, muitos sistemas necessitam realizar operações matemáticas. A necessidade de aceleração, flexibilidade e facilidade de desenvolvimento de sistemas digitais complexos fez com que surgissem as linguagens de descrição de *hardware* e a lógica programável. Tendo em vista a necessidade constante de um *hardware* multiplicador para o desenvolvimento de sistemas complexos, é proposto este projeto, que consiste na descrição em *Very High Speed Integrated Circuit Hardware Description Language* (VHDL) de um módulo que realiza operações de multiplicação com dois números sinalizados em complemento a dois e sua implementação no componente de maior flexibilidade da lógica programável denominado *Field Programmable Gate Array* (FPGA).

Este projeto visa o desenvolvimento de um módulo multiplicador para o FPGA XC4005E da Xilinx. Esse módulo realizará a multiplicação de dois números sinalizados em complemento a dois. Para uma maior flexibilidade do *hardware,* a descrição do módulo será projetada de modo a ser parametrizável, ou seja, o número de *bits* dos operandos pode ser informado através de um parâmetro ajustado na descrição do *hardware*.

Este trabalho possui cinco seções: a primeira apresenta a teoria sobre *hardware* multiplicador; a segunda expõe o projeto de um *hardware* multiplicador seguindo os critérios do algoritmo utilizado; a terceira seção mostra a validação do *hardware;* na quarta, são apresentados dados relativos à implementação e na seção final são feitas as considerações finais.

1. Teoria do *Hardware* Multiplicador

Para a multiplicação ser realizada através de um hardware, este precisa seguir ações estipuladas por algoritmos. O hardware multiplicador utilizado baseia-se em um algoritmo simples de multiplicação mostrado na Figura 1.

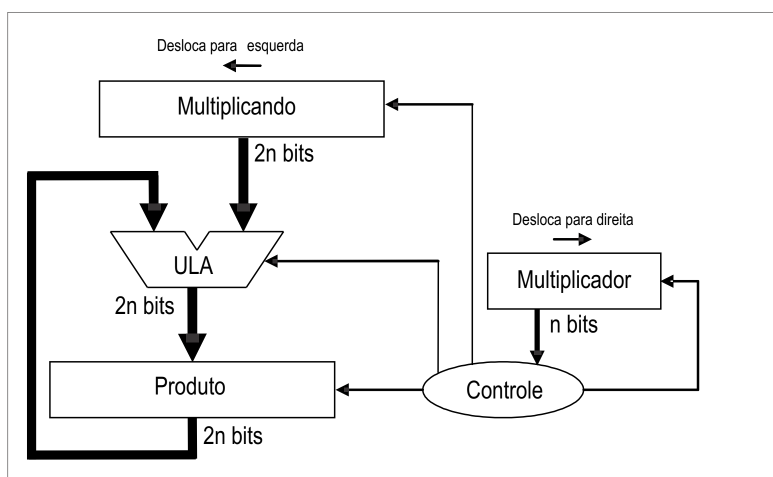

Figura 1: *Hardware* de Multiplicação.
Fonte: PATTERSON; HENNESSY, 2000.

O registrador Multiplicando, a Unidade Lógica Aritmética (ULA) e o registrador Produto são de tamanho 2n *bits*, enquanto o registrador Multiplicador é de n *bits*, onde n é o número de bits dos operandos. O valor do Multiplicando é armazenado inicialmente na metade direita de seu registrador, a metade esquerda é zerada e, a cada iteração, seu conteúdo é deslocado um bit à esquerda. O Multiplicador armazena inicialmente seu valor e, a cada iteração, seu conteúdo é deslocado um *bit* para a direita. O registrador Produto inicia com seu conteúdo igual a zero. O módulo Controle decide quando os registradores devem ser deslocados e quando armazenar valores no registrador produto. Outra função do controle é converter o multiplicando e/ou o multiplicador para números positivos, se algum for negativo, e armazenar o sinal. O algoritmo possui n-1 iterações e troca-se o sinal do produto quando o sinal do multiplicando for diferente do sinal do multiplicador (PATTERSON; HENNESSY, 2000).

Após a inicialização dos registradores, é testado o *bit* menos significativo do registrador Multiplicador. Se o *bit* for “1”, adiciona-se o valor do Multiplicando com o Produto e o resultado é armazenado no Produto. Se o *bit* menos significativo for “0”, segue-se direto para a próxima etapa, que consiste no deslocamento do Multiplicador um *bit* para direita e do deslocamento do multiplicando um *bit* à esquerda. Estes passos, menos o da inicialização, são executados n-1 vezes resultando, ao final das iterações, o valor correto da multiplicação armazenado no registrador Produto (PARHAMI, 1999).

O fluxograma do algoritmo de multiplicação é mostrado na Figura 2.

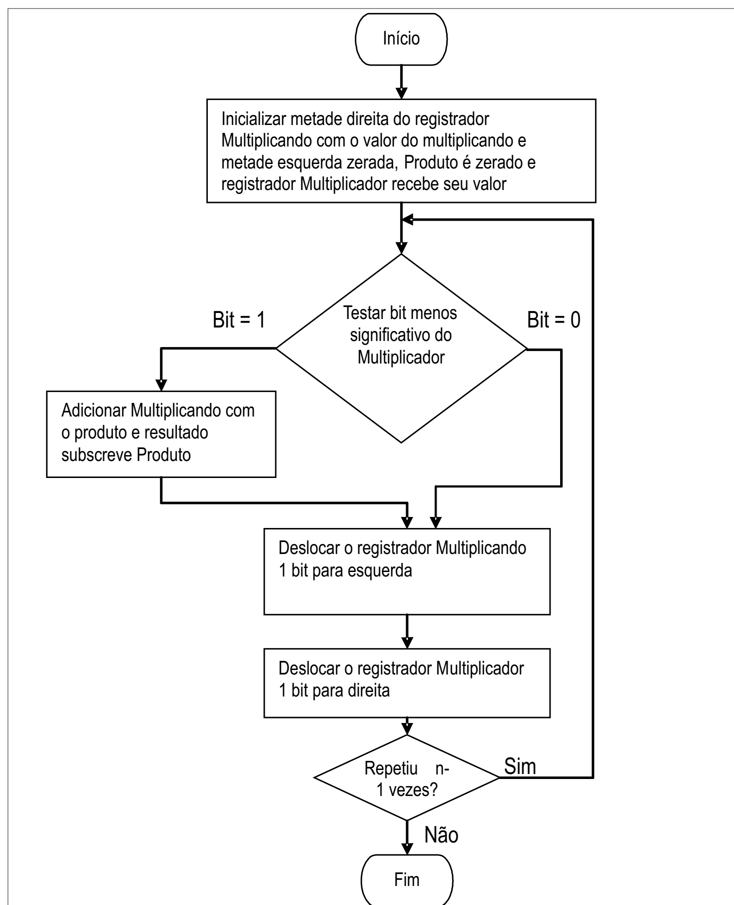

Figura 2: Fluxograma do Algoritmo do *Hardware* de Multiplicação Proposto.
Fonte: PATTERSON; HENNESSY, 2000.

2. Projeto do Multiplicador

A primeira etapa do projeto consistiu no desenvolvimento do módulo proposto para o FPGA XC4005 da Xilinx. Como ferramenta de apoio ao projeto, foi empregado o ambiente de desenvolvimento ISE 4.2.03i da Xilinx.

O módulo multiplicador projetado pode ser visto na Figura 3. Para realizar a operação de multiplicação, deve-se colocar os operandos em seus respectivos barramentos de entradas. Ao dar um pulso em **Ini** de, no mínimo, um *clock*, o resultado estará pronto na saída produto em 2.(*nº bits*) + 2 *clocks*, a saída pronto fica com o valor de nível lógico “1” e só é resetada na borda da subida do pulso de **Ini.**

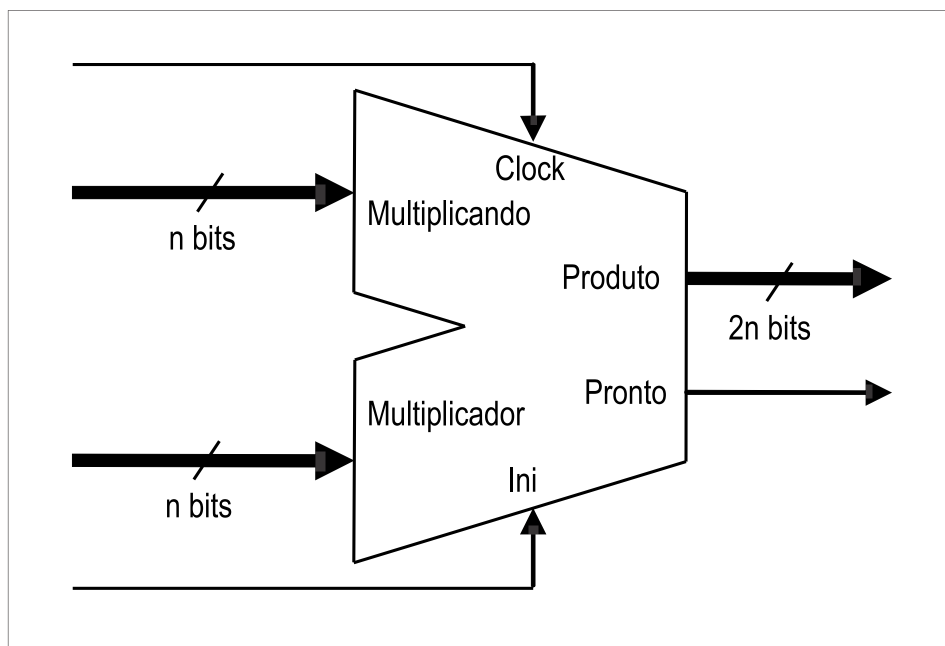

Figura 3: Módulo Multiplicador.

A descrição do *hardware* em VHDL pode ser visualizada, integralmente, a seguir.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity multiplicador is
generic(wn : integer); -- parâmetro que informa o número de bits dos operandos
port(
       clock          : in STD_LOGIC;
       ini            : in STD_LOGIC;
       multiplicador  : in STD_LOGIC_VECTOR (wn-1 downto 0);
       multiplicando  : in STD_LOGIC_VECTOR (wn-1 downto 0);
       saida          : out STD_LOGIC_VECTOR ((2*wn)-1 downto 0);
       pronto         : out STD_LOGIC
       );
end  multiplicador;


architecture  multiplicador of  multiplicador is

signal regA  : std_logic_vector((2*wn)-1 downto 0);  -- registrador de trabalho do
multiplicando
signal regB  : std_logic_vector(wn-1 downto 0);-- registrador de trabalho do multiplicador
signal regS   : std_logic_vector((2*wn)-1 downto 0); -- sinal que recebe o produto parcial
da multiplicação
constant zero : std_logic_vector(wn-1 downto 0) := (others => '0');  -- constante que
concatena zeros no início do registrador A
signal sinal1 : std_logic;            -- retém o sinal de polaridade do multiplicando
signal sinal2 : std_logic;            -- retém o sinal de polaridade do multiplicador
signal mdor  : STD_LOGIC_VECTOR (wn-1 downto 0);   -- multiplicador
signal mqdo  : STD_LOGIC_VECTOR (wn-1 downto 0);   -- multiplicando
signal cont    : std_logic_vector(1 downto 0);     -- controla as ações para realização da
multiplicação
signal cont2  : integer range 0 to wn;      -- conta o número de estapas executadas

begin

controle: process(clock,ini)
begin
       if ini = '1' then
              cont <= (others => '0');
       elsif clock'event and clock='1' then
                     if cont2 = 0 then
                            if cont = "10" then
                                   cont <= "00";
                            else
                                   cont <= cont + '1';
                            end if;
                     else
                            if cont = "01" then
                                   cont <= "00";
                            else
                                   cont <= cont + '1';
                            end if;
                     end if;
       end if;
end process;

complemento: process(ini)
begin
       if ini = '1' then
              sinal1 <= multiplicando(multiplicando'left);
              sinal2 <= multiplicador(multiplicador'left);

              if multiplicando(multiplicando'left) = '1' then
                     mqdo <= not(multiplicando) + '1';
              else
                     mqdo <= multiplicando;
              end if;

              if multiplicador(multiplicador'left) = '1' then
                     mdor <= not(multiplicador)+'1';
              else
                     mdor <= multiplicador;
              end if;

       end if;

end process;
```

```
mult: process(cont,ini)
begin

	if ini = '1' then
		cont2 <= 0;
		pronto <= '0';
	elsif clock'event and clock='1' then

			if cont2=0 then

				case cont is
					when "00" => regA <= zero & mqdo;
					             regS <= (others => '0');
					             regB <= mdor;

					when "01" => if regB(0) = '1' then
					                    regS<=regA+regS;
					                    end if;

					when "10" => regA<=regA(regA'left-1 downto 0) & "0";
					              regB<="0" & regB(regB'left downto 1);
					              cont2 <= cont2 + 1;

					when others => cont2 <= cont2 + 1; -- Não é
executado, usado apenas para não gerar erro
				end case;

			elsif (cont2 < wn) and (cont2 > 0) then

				case cont is
					when "00" =>   if regB(0) = '1' then
					                     regS<=regA+regS;
					                   end if;

					when "01" => regA<=regA(regA'left-1 downto 0) & "0";
					             regB<="0" & regB(regB'left downto 1);

					             cont2 <= cont2 + 1;

					when others => cont2 <= cont2 + 1; -- Não é
executado, usado apenas para não gerar erro
				end case;
			end if;

			if cont2 = wn then
				if (sinal1 xor sinal2) = '1' then
					saida <= not(regS) + '1';
				else
					saida <= regS;
				end if;

				pronto <= '1';
			end if;
	end if;
end process;
end multiplicador;
```

3. Validação

Na etapa de validação, desenvolveu-se um módulo *testbench* para geração de testes, descrito em VHDL. O esquema deste módulo é mostrado na Figura 4. O módulo gerador de teste insere estímulos na entrada do módulo a ser testado e recebe a resposta do multiplicador (Torok et al., 2006).

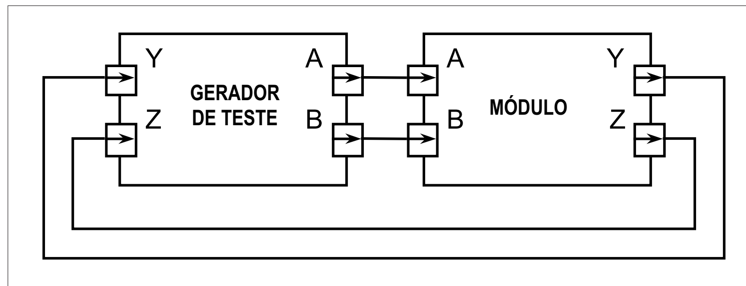

Figura 4: Esquema para Testes do Módulo Multiplicador Proposto.

As simulações foram feitas com o *software* ModelSim Starter 5.8c. A Figura 5 mostra e identifica os sinais gerados pelo módulo de teste e os sinais do multiplicador. Na simulação, o módulo multiplicador foi parametrizado para operandos de 4 *bits*. O sinal de *clock* gerado possui período de 10 ns.

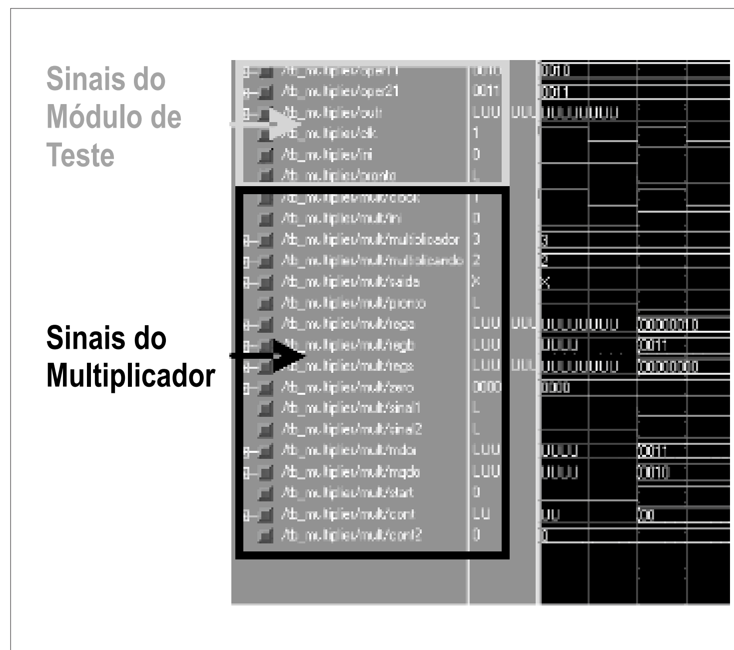

Figura 5: No detalhe, os Sinais Analisados no ModelSim.

A Figura 6 mostra que a inicialização do multiplicador é assíncrona. Para uma melhor visualização, utilizaram-se elipses brancas para indicar que, quando a entrada **ini** recebe nível lógico 1, o registrador **sinal1** recebe o sinal do multiplicando e **sinal2,** o do multiplicador. O registrador **mdor** guarda o valor do multiplicador, assim como **mqdo** armazena o valor do multiplicando e, como os números são positivos, ambos armazenam os valores sem realizar o complemento a dois. A saída **pronto**, os contadores **cont** e **cont2** são resetados nesta etapa. As elipses cinzas mostram que, na próxima borda de subida de *clock*, após o pulso em **ini**, o registrador **regA**, que faz o papel do registrador multiplicando do algoritmo, é inicializado com zeros na metade esquerda e com o valor do multiplicando na metade direita. O **regB**, que faz o papel do registrador multiplicador do algoritmo, recebe o valor do multiplicador e o **regs**, que faz o papel do registrador produto do algoritmo, é resetado. A operação inicial está de acordo com o definido no algoritmo.

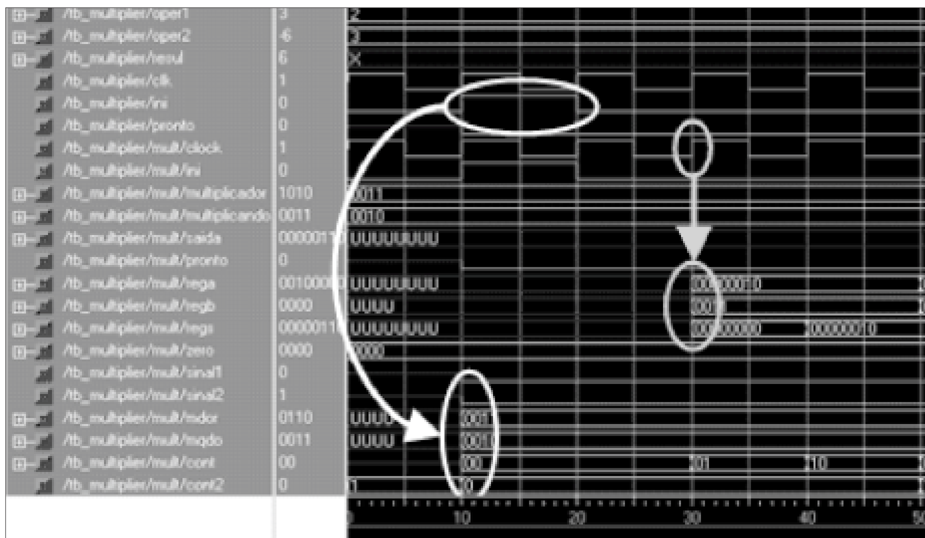

Figura 6: Inicialização do Multiplicador.

A Figura 7 mostra que, na segunda borda de subida, o *hardware* testa o *bit* menos significativo de **regb**. Como o bit é “1”, então o valor de **rega** é somado com **regs** e o resultado é posto em **regs**. Estas operações estão indicadas em branco. Conforme indicado em cinza, na próxima borda de subida do *clock,* o registrador **rega** é deslocado para a esquerda e **regb** para a direita. Note que o contador **cont** controla as ações executadas e que passa a ser incrementado sincronizadamente com o *clock* quando **ini** vai a “0”. O contador **cont2** controla o número de iterações feitas e é incrementado sincronizadamente após o deslocamento dos registradores **rega** e **regb**.

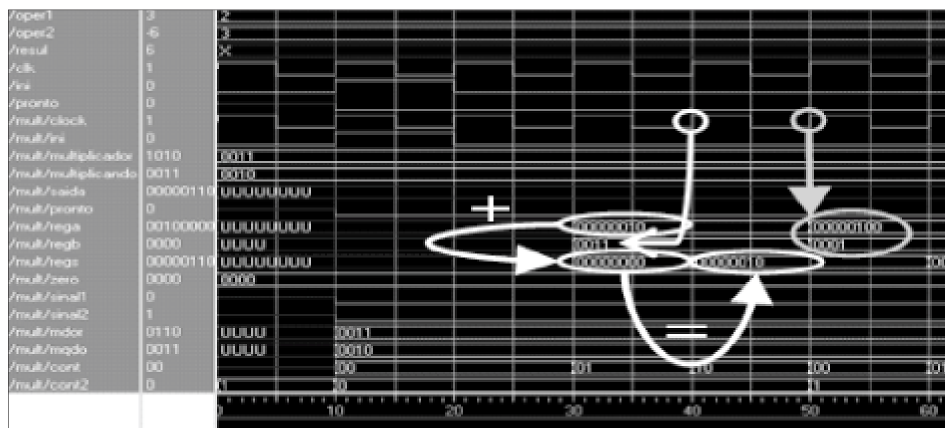

Figura 7: Operações de Teste, Soma, Armazenamento e Deslocamento nos Registradores rega, regb e regc.

A Figura 8 demonstra que uma transição de *clock* após o deslocamento dos registradores é novamente testado o *bit* menos significativo de **regb**. Como este *bit* é 1, soma-se novamente **rega** com **regs** e o resultado é armazenado em **regs**. Na próxima borda de subida do *clock,* os registradores são deslocados, ficando o registrador **regb** com “0000”. Como o bit menos significativo deste registrador é zero, na próxima transição de clock, **rega** não é adicionado com **regs** e, como **regs** possui todos seus

bits zerados, ele já está com o valor final da multiplicação. São necessárias mais duas iterações do *hardware* para que seja completado o processo e mais um ciclo de *clock* para atribuir o sinal correto ao resultado. Após estas etapas, a saída **pronto** é setada, indicando que a saída está com o resultado correto. Conforme indicado na Figura 8, a multiplicação de 3 vezes 2 resulta em 6.

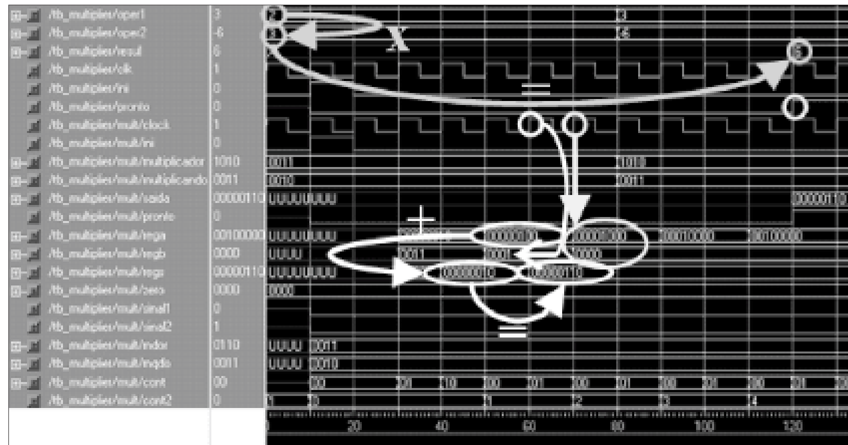

Figura 8: Operações do *Hardware* e Resultado Final.

A Figura 9 apresenta a multiplicação de 3 com -6. Ao aplicar "1" lógico à **ini** é guardado o sinal do multiplicador em **sinal2** e realiza-se o complemento a dois [(COOK, 1997), (HAYES, 1993)] em seu valor, pelo fato de que é negativo, e o valor positivo é armazenado em **mdor**. As operações realizadas seguem o algoritmo proposto e quando houver três interações, **regs** possuirá o valor da multiplicação. Como o resultado é positivo e os sinais dos valores a multiplicar são diferentes, o complemento a dois é realizado antes de sua apresentação à saída, necessitando um ciclo de clock a mais. Como mostrado, o hardware apresentou o resultado -18.

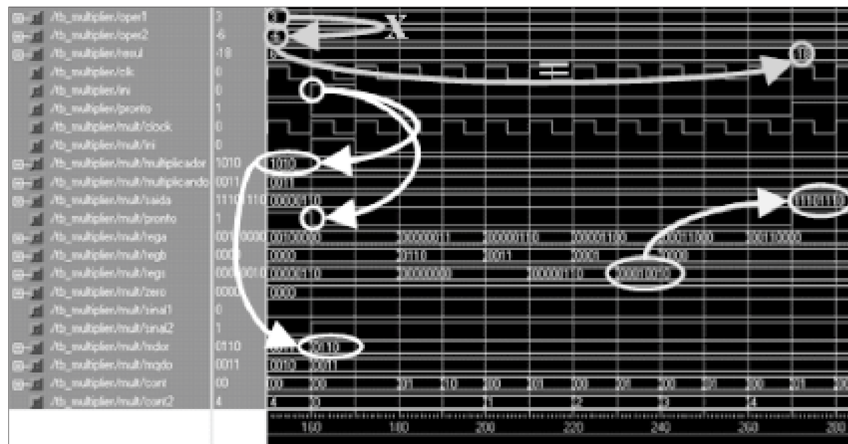

Figura 9: Operações do Hardware e Resultado Final.

As simulações realizadas e as análises das mesmas validaram a descrição do *hardware* multiplicador. A descrição do módulo gerador de teste para a geração de estímulos e observação da resposta do multiplicador, pode ser visualizada no código que segue.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;

entity tb_multiplier is
end tb_multiplier;

architecture tb of tb_multiplier is

component multiplicador is
generic(wn : integer);
port    (
        clock          : in STD_LOGIC;
        ini            : in STD_LOGIC;
        multiplicador  : in STD_LOGIC_VECTOR (wn-1 downto 0);
        multiplicando  : in STD_LOGIC_VECTOR (wn-1 downto 0);
        saida          : out STD_LOGIC_VECTOR ((2*wn)-1 downto 0);
        pronto         : out STD_LOGIC
        );
end  component;

constant clkprd : time := 10 ns;     -- tempo de ciclo do clk
constant cplx_w : integer := 4;      -- tamanho dos operandos em bits

signal  oper11 : std_logic_vector(cplx_w-1 downto 0) := "0010";   -- gera um valor para o
multiplicando
signal  oper21 : std_logic_vector(cplx_w-1 downto 0) := "0011";   -- gera um valor para o
multiplicador
signal  outr : std_logic_vector((cplx_w*2)-1 downto 0);    -- resposta
signal  clk : std_logic;                                   -- gera o clock
signal  ini, pronto : std_logic := '0';                    -- gera sinal de ini e pronto

begin

mult: multiplicador          -- módulo a ser testado
generic map (wn =>cplx_w)
port map(multiplicando => oper11,
        multiplicador => oper21,
        saida => outr,
        ini => ini,
        clock => clk,
        pronto => pronto);

clockgen: process            -- gera o clock
begin
        clk <= '1';
         wait for clkprd/2;
         clk <= '0';
         wait for clkprd/2;
end process;

vetor: process(clk)          -- gera sinais de teste
variable i: integer range 0 to 1000;
begin

        if (clk'event and clk = '1') then
                case i is
                        when 1 => ini <='1';
                        when 2 => ini <='0';
                        when 8 => oper11 <= "1101";
                                oper21 <= "1010" ;
                        when 10 => ini <='1';
                        when 11 => ini <='0';
                        when others => null;
                end case;
                i := i+1;
        end if;
end process;

end tb;
```

4. Implementação

A descrição do módulo multiplicador foi testada através de sua simulação funcional, conforme descrito anteriormente. Mas, para que o projeto possa ser implementado em um FPGA, deve-se realizar a sua síntese lógica e física (SKAHILL, 1996). O *software* utilizado para a síntese foi o Xilinx ISE 4.2.03i. A síntese do projeto foi realizada com sucesso e um resumo dos dados mais importantes utilizando operandos de 8 *bits* é apresentado a seguir.

A Tabela 1 mostra os dados extraídos do mapeamento da lógica feito em um FPGA XC4005E – 4PC84. Os *Control Logic Blocks* (CLB) são os blocos que irão se configurar, de acordo com a síntese lógica gerada através da descrição do *hardware*, para realizar a multiplicação.

Tabela 1: Dados do Mapeamento.

Number of CLBs:	98 out of 196 - 50%
CLB Flip Flops:	47
4 input LUTs:	137
3 input LUTs:	6
16X1 RAMs:	16
Number of bonded IOBs:	35 out of 61 - 57%
IOB Flops:	16
IOB Latches:	2
Number of clock IOB pads:	1 out of 8 - 12%
Number of primary CLKs:	1 out of 4 - 25%
Number of secondary CLKs:	1 out of 4 - 25%
Total equivalent gate count for design:	2567
Additional JTAG gate count for IOBs:	1680

A Figura 10 mostra as rotas de conexão utilizadas para interligar os CLB e os blocos de entrada e saída (*Input Output Blocks* – IOB). Os componentes configuráveis são identificados pelos quadrados, sendo que os que possuem conexões são utilizados nesta aplicação e as rotas são representadas pelas linhas.

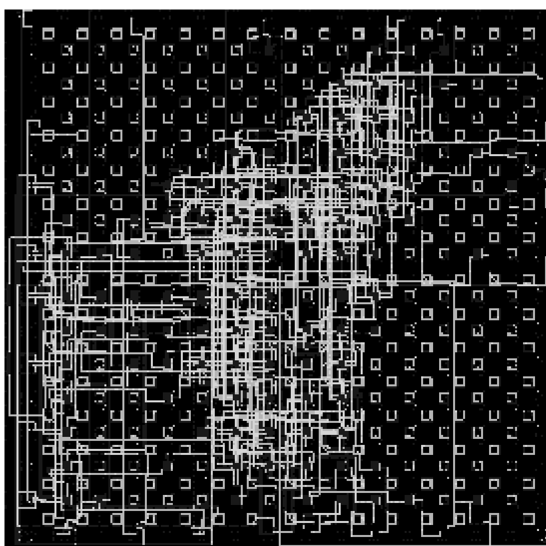

Figura 10: Rotas de Conexão entre Blocos Configuráveis.

As rotas e a lógica do circuito geram atrasos que restringem a velocidade de operação do *hardware*. Este *hardware* possui 1.360 caminhos, 210 redes e 491 conexões, gerando um atraso de rede máximo de 14,567 ns e freqüência máxima de 23,776 MHz. Ou seja, para executar a multiplicação de dois operandos de 8 *bits*, por exemplo, o *hardware* realiza a operação em 18 ciclos de *clock*, realizando, assim, 1.320.888 multiplicações por segundo.

Considerações Finais

Para o projeto de um *hardware* multiplicador ser implementado, deve-se escolher o algoritmo de multiplicação que mais se adapte aos requisitos do sistema eletrônico do qual fará parte. Para tanto, deve-se analisar o número de *bits* dos operandos que serão multiplicados, o tempo gasto pelo algoritmo para efetuar a operação com este número de *bits*, o custo de *hardware* necessário para a implementação e a facilidade de desenvolvimento e implementação do controle do multiplicador.

O modo como o algoritmo será implementado independe da tecnologia aplicada, mas, através deste projeto, pôde-se confirmar a flexibilidade e facilidade de desenvolvimento ao utilizar a linguagem de descrição de *hardware* VHDL e os FPGA. Os erros encontrados analisando a simulação são, na maioria das vezes, rapidamente corrigidos no código VHDL, novamente sintetizados e testados para confirmar a eliminação desses erros. Na etapa de síntese lógica e física, o modelo do FPGA da Xilinx proposto para implementação comportou todo o hardware necessário para operandos de 8 *bits*, tendo utilizado somente metade de seus blocos lógicos, obtendo uma excelente velocidade de operação.

Referências

COOK, Nigel P.. **Introductory Digital Electronics**. 1.ed. Upper Saddle River, New Jersey: Prentice Hall, 1997. 719p.

HAYES, John P. **Introduction to Digital Logic Design**. 1.ed. United States of America: Addison-Wesley, 1993. 815p.

PARHAMI, Behrooz. **Computer Arithmetic:** Algorithms and Hardware Design. 1.ed. New York: Oxford USA Trade, 1999. 512p.

PATTERSON, David A.; HENNESSY, John L. **Organização e projeto de Computadores:** A Interface Hardware/Software. 2.ed. Rio de Janeiro - RJ: LTC, 2000. 537p.

TOROK, Delfim L. et al. Functional Validation Strategy for the FFT and its Inverse Reconfigurable Hardware. **21th South Symposium on Microelectronics**, Porto Alegre, n. 2006, p. 47 - 50, mai. 2006.

SKAHILL, Kevin. **VHDL for Programmable logic**. 1.ed. Menlo Park, CA: Addison-Wesley, 1996. 594p.