UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE INFORMÁTICA
ENGENHARIA DE COMPUTAÇÃO

Nathan Rodrigues Tavares de Lima - 20180017834

Weslley Dezidério da Silva - 20180017244

Docente: Hugo L. D. de S. Cavalcante

GDSCO0053 - Introdução à Microeletrônica

Multiplicador Digital de 4 bits

2 de novembro de 2023

1 Introdução

Neste projeto, embarcaremos na implementação de um multiplicador digital de 4 bits projetado para inteiros sem sinal. Adotando a abordagem TopDown, este processo de desenvolvimento é dividido em etapas essenciais. Inicialmente, utilizaremos um arquivo VHDL para descrever o comportamento do multiplicador, fornecendo uma representação clara e abstrata do algoritmo.

Em seguida, avançaremos para a geração do circuito, empregando um arquivo VBE para traduzir nossa descrição de alto nível em uma representação de circuito lógico. Esta etapa é crucial para a transformação da lógica em elementos físicos que podem ser posteriormente implementados em hardware.

A última fase envolverá a criação do circuito físico (AP) por meio de um arquivo VST, permitindo-nos uma representação completa do multiplicador em nível de hardware. O resultado será uma implementação concreta do algoritmo que pode ser carregada em dispositivos específicos, como FPGAs.

Por fim, realizaremos testes exaustivos para validar o desempenho e a precisão de nossa implementação. Utilizaremos uma biblioteca que nos permite gerar padrões de entrada e saída na linguagem C, chamada genpat. Em seguida, faremos uso do simulador Asimut para executar testes detalhados.

Este projeto visa demonstrar a trajetória completa, desde a concepção até a validação de um multiplicador digital de 4 bits, destacando o poder e a versatilidade das técnicas de design digital e da síntese de circuitos. Além disso, enfatiza a importância de testes rigorosos para garantir a funcionalidade e a confiabilidade do sistema.

2 Metodologia

2.1 Estrutura do Hardware

No desenvolvimento do Multiplicador de 4 bits utilizamos o método de *shift and add* que consiste em realizar deslocamentos sucessivos nos bits de entrada durante a execução da operação. Abaixo, segue a estrutura de hardware para essa operação.

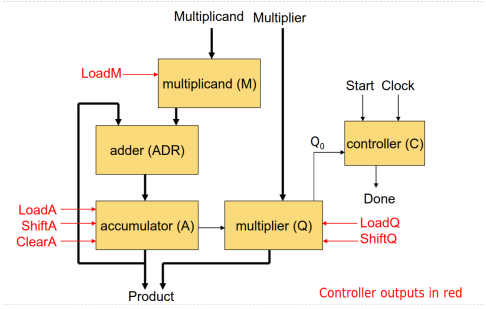

Figura 1: Multiplicador utilizando técnica shift and add.

2.2 Código Verilog HDL

Para obter este resultado, foi definido o seguinte código em VHDL que descreve nossa entidade, bem como sua arquitetura:

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

entity mult4 is
port (
    a :in std_logic_vector(3 downto 0);
    b :in std_logic_vector(3 downto 0);
    product :out std_logic_vector(7 downto 0);
    cout : out std_logic;
    vdd :in std_logic;
    vss :in std_logic
);
end mult4;

architecture mult4_behavioral of mult4 is
begin
    process(a, b)
    variable a_value, b_value, cout_tmp: std_logic_vector(7 downto 0);
    begin
        a_value := "00000000";
        b_value := "0000" & b;
        cout_temp := "0";

        for i in 0 to 3 loop
            if a_value(i) = "1" then
                a_value := a_value + b_value;
                if a_value(7) = "1" then
                    cout_tmp := "1";
                else
                    cout_tmp := "0";
                end if;
            end if;
            b_value := b_value(6 downto 0) & '0';
        end loop;
        product <= a_value;
        cout <= cout_tmp;
    end process;
end mult4_behavioral;
```

2.3 Código para o Gerador de Padrões

```
#include <stdio.h>
#include "genpat.h"

#define step 400000LL
#define delay 400000LL

char intToStr(inteiro)
int inteiro; {
    char* str;
    str = (char) mbkalloc(32*sizeof(char));
    sprintf(str, "%d", inteiro);

    return (str);
}

int main() {
    int i;
    int j;
    int k;
    int time = 0;
    int cout = 0;

    DEF_GENPAT("mult4_genpat");
    DECLAR("a", ":2", "X", IN, "3 downto 0", "");
    DECLAR("b", ":2", "X", IN, "3 downto 0", "");
    DECLAR("s", ":2", "X", OUT, "7 downto 0", "");
    DECLAR("cout", ":2", "B", OUT, "", "");
    DECLAR("vdd", ":2", "B", IN, "", "");
    DECLAR("vss", ":2", "B", IN, "", "");

    LABEL("multiplicacao");
    AFFECT("0", "vdd", "0b1");
    AFFECT("0", "vss", "0b0");

    for (i = 0; i < 16; i++) {
        for (j = 0; j < 16; j++) {
            k = i*j;
            AFFECT(intToStr(time), "a", intToStr(i));
            AFFECT(intToStr(time), "b", intToStr(j));
            time += delay;
            AFFECT(intToStr(time), "s", intToStr(k&0xf));

            if (k < 9) {
                AFFECT(intToStr(time), "cout", "0");
```

```
                } else {
                    AFFECT(intToStr(time), "cout", "1");
                    time += step;
                }
            }
        }

        SAV_GENPAT();

        return 0;
    }
```

3 Resultados

3.1 Esquemático Gerado

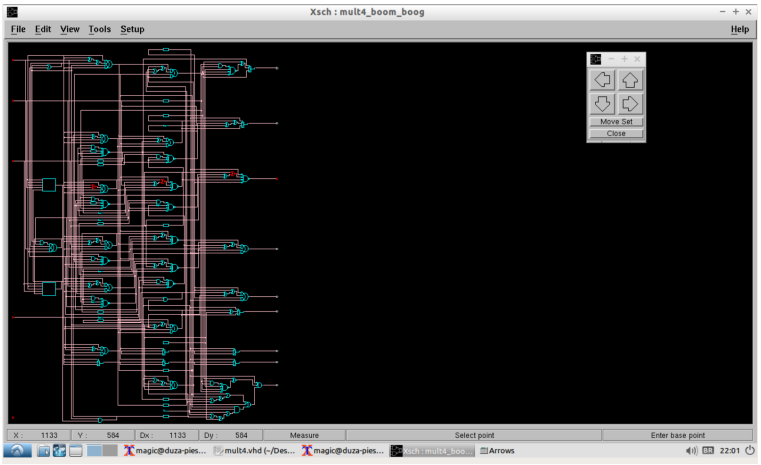

Figura 2: Esquemático do circuito.

3.2 Arquivo Físico Gerado

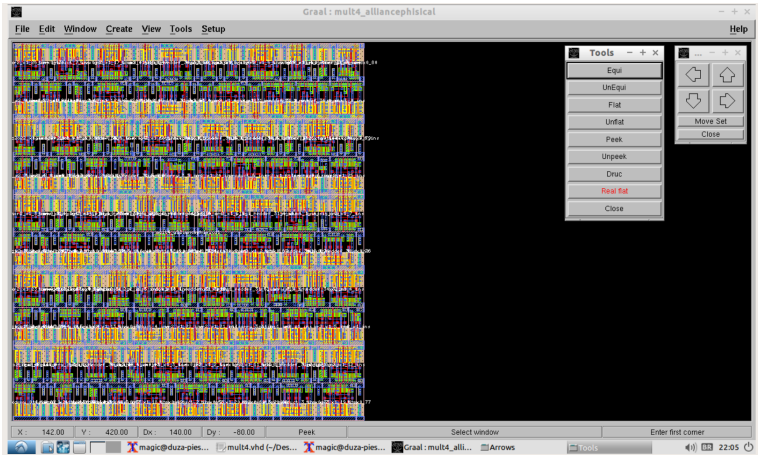

Figura 3: Layout físico gerado.

3.3 Arquivo Físico Gerado Roteado

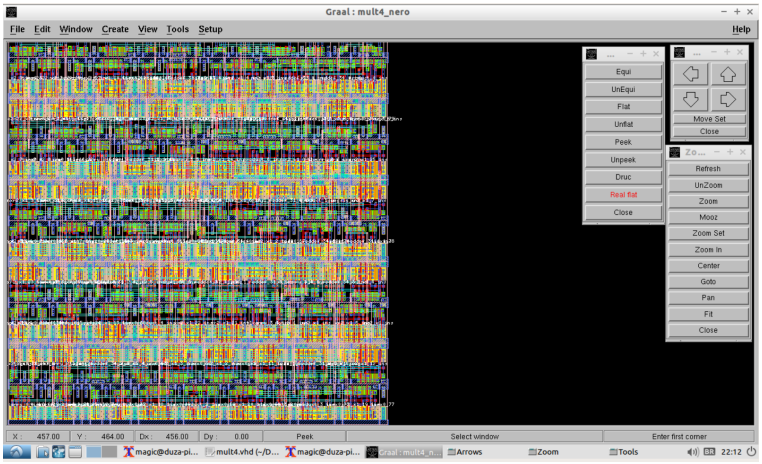

Figura 4: Layout físico gerado roteado.

Após rodar o Asimut, observamos que após a execucação de algumas instâncias, o programa é abortado devido a erros nos produtos que o genpat gerou com o que foi implementado em VHDL. No entanto, ao conversar com o professor, descobrimos que o problema poderia estar relacionado com o tempo de atraso de operação dos elementos envolvidos.

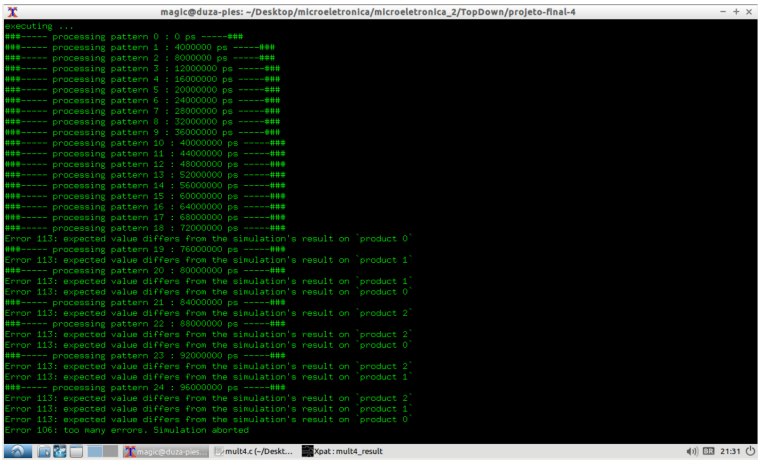

Figura 5: Saída do Asimut.

4 Conclusão

Dessa forma, podemos concluir que apesar da discrepância no Asimut, se analisarmos os arquivos gerados pelo genpat individualmente (arquivo de padrão de testes criado e o resultado da simulação), vemos que os valores esperados são satisfeitos.

É fato que, idealmente poderíamos utilizar outras ferramentas a fim de não só otimizar o processo de operação, bem como melhorar os resultados ao utilizar o Asimut. Uma dessas ferramentas seria o uso de paralelismo no código VHDL. No entanto, para o presente projeto, não tínhamos a habilidade técnica necessária da linguagem para realizar tal tarefa.