UNIVERSIDADE FEDERAL RURAL DO SEMIÁRIDO

***CAMPUS* CARAÚBAS**

MARIA BRUNA DE OLIVEIRA FONSECA

ESTUDO E DESENVOLVIMENTO DE MULTIPLICADORES PARA PROCESSAMENTO DE SINAIS UTILIZANDO FPGA

CARAÚBAS - RN

2015

MARIA BRUNA DE OLIVEIRA FONSECA

ESTUDO E DESENVOLVIMENTO DE MULTIPLICADORES PARA PROCESSAMENTO DE SINAIS UTILIZANDO FPGA

Trabalho de conclusão de curso apresentado ao Departamento de Bacharelado em Ciências e Tecnologia da Universidade Federal Rural do Semiárido, como parte dos requisitos para obtenção do Título de Bacharel.

Orientador:

Prof. Dr Francisco de Assis Brito Filho

CARAÚBAS - RN

2015

Dados Internacionais de Catalogação na Publicação (CIP)
Biblioteca Setorial Campus Caraúbas (BSCA)
Setor de Informação e Referência

F676e Fonseca, Maria Bruna de Oliveira.

Estudo e desenvolvimento de multiplicadores para processamento de sinais utilizando fpga / Maria Bruna de Oliveira Fonseca -- Caraúbas, 2015.
43f.: il.

Orientador: Prof. Francisco de Assis Brito Filho.

Monografia (Graduação em Ciência e Tecnologia) – Universidade Federal Rural do Semi-Árido.

1. Linguagem de programação. 2. FPGA. 3. Multiplicadores. I. Título.

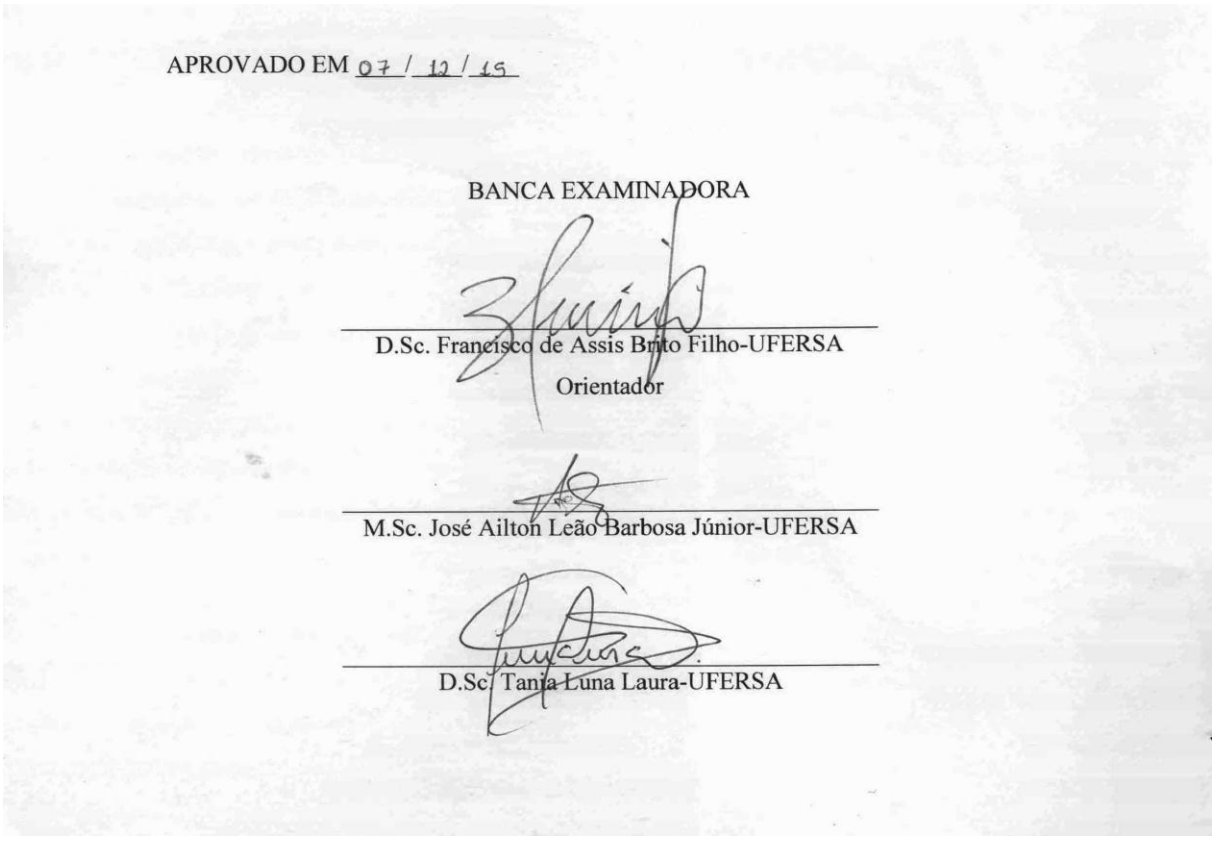
RN/UFERSA/BSCA CDD: 005.13

Bibliotecário: Dalvanira Brito Rodrigues
CRB-15/700

MARIA BRUNA DE OLIVEIRA FONSECA

ESTUDO E DESENVOLVIMENTO DE MULTIPLICADORES PARA PROCESSAMENTO DE SINAIS UTILIZANDO FPGA

Este trabalho de conclusão de curso foi julgado adequado para a obtenção do título de Bacharel e aprovado em sua forma final pelo Bacharelado em Ciências e Tecnologia da Universidade Federal Rural do Semi-Árido.



APROVADO EM 07 / 12 / 15

BANCA EXAMINADORA

D.Sc. Francisco de Assis Brito Filho-UFERSA
Orientador

M.Sc. José Ailton Leão Barbosa Júnior-UFERSA

D.Sc. Tania Luna Laura-UFERSA

AGRADECIMENTOS

A Deus por permitir chegar a concluir esse trabalho.

Aos meus pais pela paciência e incentivo.

Ao meu irmão por estar sempre me apoiando.

Aos meus amigos pelo apoio.

Ao meu orientador Francisco Brito.

RESUMO

Os multiplicadores binários são circuitos base de diversos sistemas desde filtros digitais (responsáveis por isolar determinadas faixas de sinais) até sistemas de processamento. Eles são responsáveis pela operação de multiplicação binaria. Os circuitos foram descritos em VHDL(*VHSIC Hardware Description Language*) que é uma linguagem de descrição de Hardware incialmente desenvolvida pra padronizar os projetos do departamento de defesa dos Estados Unidos, para realizar a implementação foi utilizado o dispositivo FPGA (*Field Programmable Gate Array)*, esse é um dispositivo utilizado para a prototipação de circuitos digitais, possuindo um hardware configurável e a capacidade de processamento em paralelo, que o torna uma boa opção para a realização dos testes .

Palavra-chave: FPGA, multiplicador, VHDL.

ABSTRACT

Binary multiplier circuits are based on different systems since digital filters (isolate responsible for certain signs of tracks) until processing systems. They are responsible for binary multiplication operation. The circuits were described in VHDL (VHSIC Hardware Description Language) which is a Hardware Description Language initially developed to standardize the designs of the US Department of Defense to carry out the implementation we used the FPGA (Field Programmable Gate Array) this is a device used for prototyping of digital circuits, having a configurable hardware and processing capacity in parallel, which makes it a good choice for the tests.

Key-words: FPGA, multiplier, VHDL.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Em nossa sociedade busca-se obter maior velocidade em nossos aparelhos eletrônicos (telefones, computador, tablets) e nos sistemas de comunicação (telefonia e internet). Este objetivo depende diretamente do poder de processamento das máquinas envolvidas ou da velocidade de processamento dos microprocessadores, que são o núcleo destas máquinas. Os FPGAs surgem como uma opção diante dos microprocessadores para aplicações específicas, como o processamento de sinais, pois tais circuitos apresentam a possibilidade de realizar processamento em paralelo, ao passo que os microprocessadores realizam sempre o processamento de forma sequencial.

Os FPGAs são circuitos que apresentam blocos lógicos programáveis e possuem como característica a capacidade de processamento de informação em paralelo. Esta característica permite o aumento na velocidade do processamento da informação, bem como a possibilidade da implementação de um número considerável de operações logicas simultaneamente. Para programar um FPGA é utilizada uma linguagem de descrição de hardware (HDL, do inglês Hardware Description Language) (TOCCI; WIDMER, 2000).

Os multiplicadores são blocos básicos no processamento de sinais digitais, responsáveis pela realização das multiplicações binarias. Esta operação apresenta-se de modo semelhante a que ocorre com o sistema de numeração decimal, porém com um número considerável de parcelas no cálculo das somas que devem ser realizadas.

Tendo essa necessidade em vista esse trabalho se justifica pela busca de realizar essa operação de forma mais eficiente obtendo uma maior velocidade. Portanto teremos como objetivos, realizar a comparação, estudo e desenvolvimento de multiplicadores binários utilizando FPGA e a linguagem de descrição de hardware VHDL, e por objetivos específicos, Estudar os algoritmos de implementação de multiplicadores binários; Estudar uma linguagem de descrição de hardware e a sua metodologia de utilização em projeto de sistemas digitais; Desenvolver multiplicadores binários, simular e realizar a sua implementação física em hardware de lógica configurável.

.

2 REVISÃO BIBLIOGRÁFICA

2.1. FPGA

2.1.1. HISTORIA

Diante de uma sociedade onde a transmissão rápida e eficiente tornou-se uma necessidade, fica difícil lembrar dos computadores que ocupavam salas refrigeradas e que necessitavam de várias trocas de válvulas. Buscou-se o desenvolvimento de novas tecnologias para melhorar a questão da velocidade dos dispositivos como em seu tamanho. Um marco de vital importância para a eletrônica é a descoberta do diodo de emissão termiônica em 1902 por J. A Fleming, em 1902. Surge primeiro computador, o ENIAC, produzido na universidade da Pensilvânia, em 1946. Em 1960 surgi a segunda geração, com a utilização de transistores, os equipamentos diminuíram de tamanho e custo dos computadores. A terceira geração já na década de 70 utilizava CIs. Nessa história de avanços surge os PLD para diminuir o número de CIs a serem utilizados, bem como funções que não haviam, sem a necessidade de desenvolvimento de grandes placas, podendo ser utilizado para reunir funções logicas, flip-flop e registradores em um único chip. Um dos mais robustos dispositivos PLD são os FPGAs. Esses dispositivos chegaram ao mercado em 1984, fabricados pela empresa Xilins. Enquanto os dispositivos PLD simples deveriam ser configurados em sua fabricação, os FPGAs podem ser configurados pelo usuário.

Figura 1- Chip do de um FPGA da Xilinx.

Fonte:https://upload.wikimedia.org/wikipedia/commons/4/4e/Xilinx_Spartan-3E_ (XC3S500E).jpg

2.1.2. ESTRUTURA DOS FPGAS

Os dispositivos FPGAs tornara-se ferramenta importante para a prototipação de circuitos, são dispositivos que podem serem configurados simulando circuitos de diversos níveis de abstração, desde portas lógicas simples como a *and* a circuitos complexos, trazendo como vantagem a redução de custos com equipamentos e tempo para introdução de novos aparelhos ao mercado.

Os FPGAs são compostos basicamente por blocos programáveis e uma matriz de interconexões e periféricos de entrada e saída, na Figura 2Vemos os principais tipos de FPGAs que são: a matriz simétrica, *sea-of-gates*, *row-based*, PDL hierárquico.

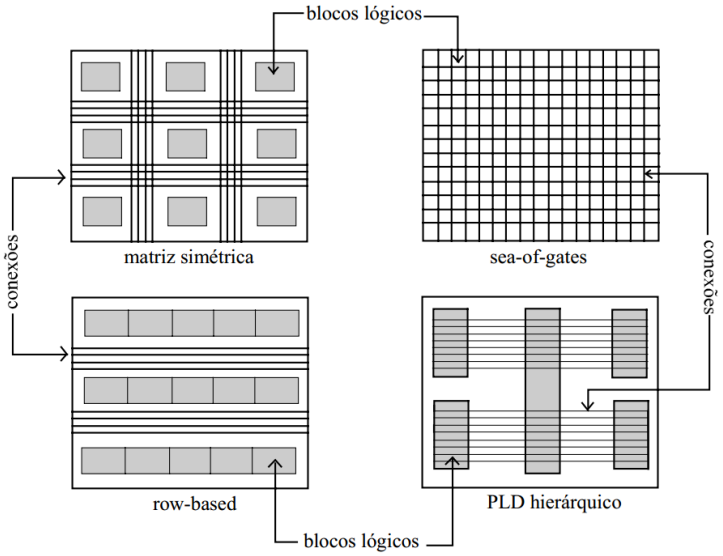
Nos dispositivos de matriz de simetria apresenta mais versátil com relação à transmissão de informação entre um bloco logico e outro.

A *sea-of-gates* é composto por blocos lógicos de baixa complexidade, sua matriz de conexão está disposta sobre os blocos lógicos o que algumas vez pode impedir o aproveitamento de parte do dispositivo.

No *row-based* é muito similar ao *sea-of-gates* porem apresenta com principal distinção a matriz de conexões reservada.

PLD hierárquico nesses dispositivos temos a possibilidade de agrupar blocos lógicos entre si.

Figura 2-Esquema que ilustra as principais estruturas de uma FPGA.



Fonte: Livro Projeto, desempenho e aplicações de sistemas digitais em circuitos programaveís (FPGAs).

2.1.3. O CYCLONE II

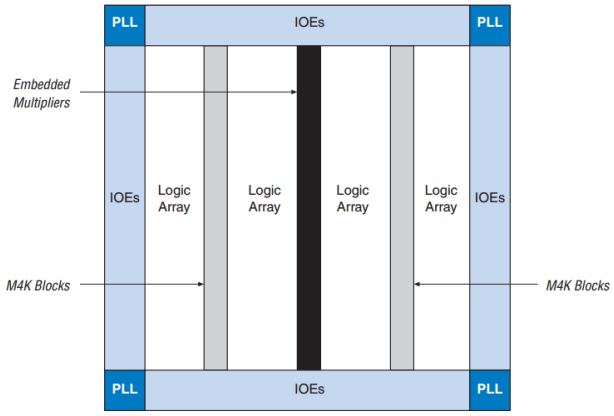
2.1.3.1. O FPGA EP2C35

O EP2C35 tem sua arquitetura semelhante à descrita na Figura 3, possuindo 475 pinos entre entradas e saídas (I/OS), 105 blocos de memória M4k (4 kbits plus 512 parity bits) 35 multiplicadores, 4 PLL (que são dispositivos que multiplicam ou dividem clock) e os arranjos lógicos (LAB) além da rede de interconexões.

Os arranjos lógicos são compostos por 16 elementos lógicos (LEs) com densidade de 33276 LEs. Registrador, sinais de controle e uma interconexão local.

A rede de interconexões que interliga os LABs é composta por linhas e colunas. As linhas são classificadas em R4 e R24. As linhas R4 fazem a conexão entre 4 LABs, ou entre 3 LABs e 1 multiplicador ou ainda entre 3 LABs e 1 memoria M4k. as R24 ligam 24 LABs. As colunas se dividem em C4 e C16 e realizam a mesma função que as linhas, porem com extensões maiores.

Figura 3- Diagrama de blocos do dispositivo Cyclone II EP2C20.



Fonte: Cyclone II Device Handbook, Volume 1.

2.2. LINGUAGEM DE DESCRIÇÃO DE HADWARE VHDL

A linguagem de descrição de hardware VHDL foi desenvolvida a mando do departamento de defesa dos Estados Unidos pelo a empresas Texas, IBM e Intermetrics para ser uma linguagem padrão para a projetos e documentação. Com base na versão7.2 a IEEE em 1987 é definida padrão IEEE 1076-1987. Posteriormente foi introduzido a linguagem os padrões, o IEEE 1964 e IEEE 1076.3.

“Algumas vantagens em utilizar VHDL, são: reduz tempo/custo de desenvolvimento, maior nível de abstração, projetos independentes da tecnologia, facilidade de atualização dos projetos, grande número de usuários (internacional). ” (ORDONEZ et al., 2003)

2.3. METODOLOGIA PC-PO

A metodologia PC-PO consiste em dividir o circuito em uma parte de controle e uma parte operacional.

A parte de controle é a responsável por coordenar as ações que devem ser efetuadas pela parte operacional, para que não haja confronto entre a as entradas. Isso se dá através de uma lógica de controle, geralmente implementada por um FSM (máquina de estado finito).

A parte operacional é a parte onde há a execução das operações propriamente dita, composto por registradores, conexões e operadores lógicos e matemáticos.

Esse método foi utilizado para o desenvolvimento do código dos multiplicadores.

2.2. MULTIPLICADORES BINÁRIOS

2.3.1. A ADIÇÃO BINÁRIA

A adição binaria é a operação que é realizada entre as parcelas da multiplicação. As regras para a adição binária são apresentadas na Tabela 1:

Tabela 1-Regras para a adição binaria.

0 + 0 = 0
0 + 1 = 1
1 + 1 = 10 (carry 1 para próxima posição)
1 + 1 + 1 = 11 (carry 1 para próxima posição)

Fonte: Autoria Própria.

2.3.2. A MULTIPLICAÇÃO BINARIA

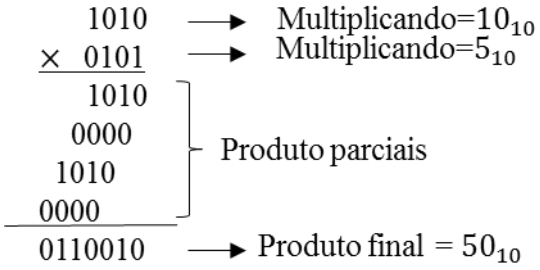
A multiplicação binaria se dá de forma análoga a multiplicação de números decimais. A regras básica para a multiplicação são dadas na Tabela 2:

Tabela 2- Regras para a multiplicação binaria.

$0 \times 0 = 0$
$0 \times 1 = 0$
$1 \times 0 = 0$
$1 \times 1 = 1$

Fonte: Autoria Própria.

Vejamos um exemplo:



3. CIRCUITOS MULTIPLICADORES BINARIOS

Os circuitos multiplicadores são circuitos sequenciais que dever realizar a operação de multiplicação nos dispositivos digitais. Eles são componentes essenciais para o funcionamento de microprocessadores, processadores gráficos, o mesmo tem impacto direto no desempenho das unidades logicas aritméticas (ALU). Os circuitos multiplicadores são elementos básicos dos filtros digitais, esses filtros são circuitos responsáveis por realizar a seleção de um trecho de um sinal seja para amplifica-lo ou para remove-lo.

O Ordonez et al. (2003) apresentam alguns parâmetros para analisar o desempenho de circuitos digitais:

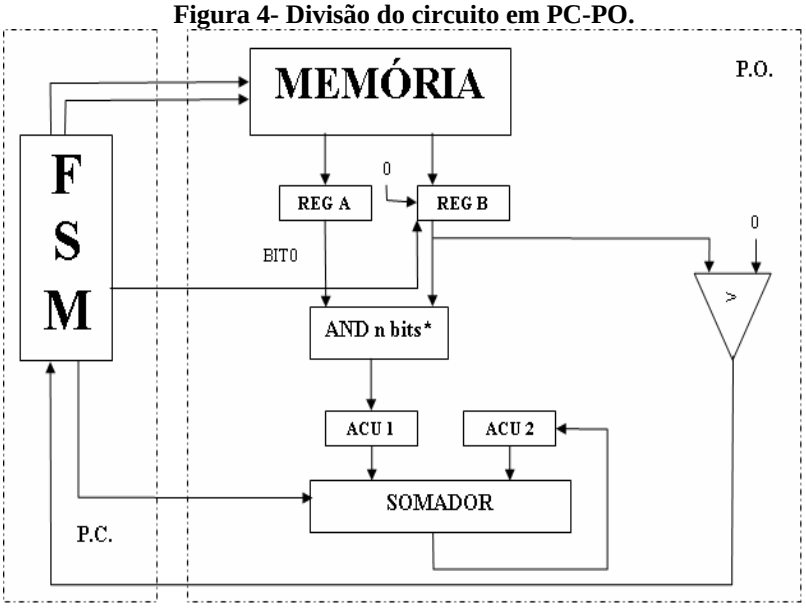
> Há vários parâmetros para se medir o desempenho de circuitos digitais em FPGAs, os principais parâmetros são: (i) a *ocupação espacial*, que determina quantos componentes são necessários para implementar o circuito. (ii) O *desempenho temporal*, que determina o tempo de atraso do sinal (informação) através do circuito.

A seguir são apresentados dois multiplicadores binários:

3.1. MULTIPLICADOR SOMA E DESLOCA

O algoritmo multiplicador soma e desloca realiza a multiplicação segundo as regras de multiplicação apresentadas anteriormente, em seguida realiza a soma e desloca a parcela para a direita. Na Figura 4, pode ser observado no diagrama abaixo como ocorre a divisão do circuito nas partes PC-PO. A parte de controlo é composta por uma máquina de estado finito, a parte operacional é composta por registradores, deslocador, somadores e acumuladores. Esse multiplicador realiza apenas a multiplicação de números positivos.

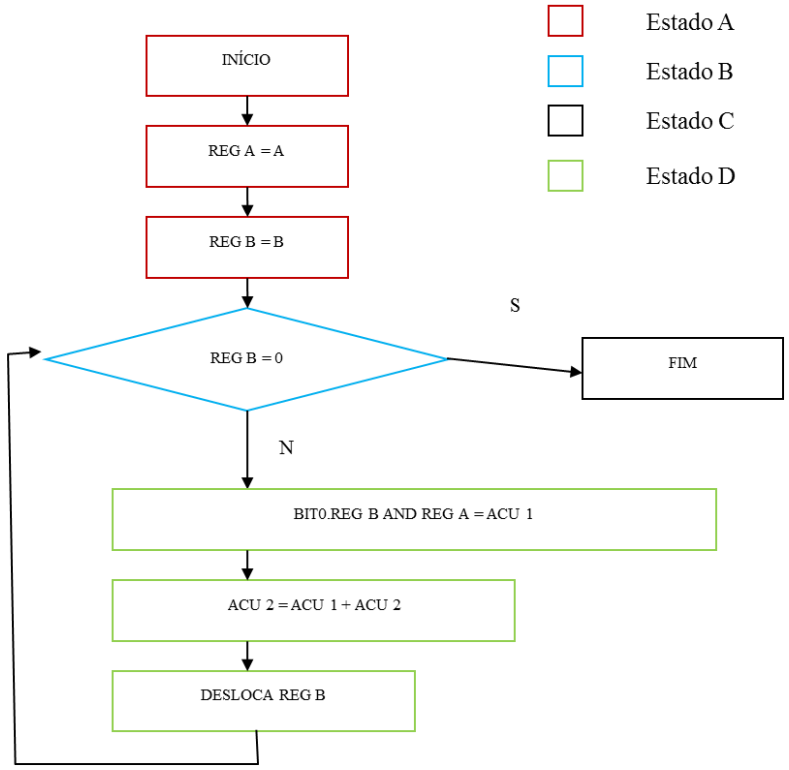
Figura 4- Divisão do circuito em PC-PO.



Fonte: Autoria Própria.

Na parte operacional o multiplicador deverá seguir a lógica descrita pelo fluxograma da Figura 5, podemos observar o que acontecerá a cada estado da máquina durante a multiplicação.

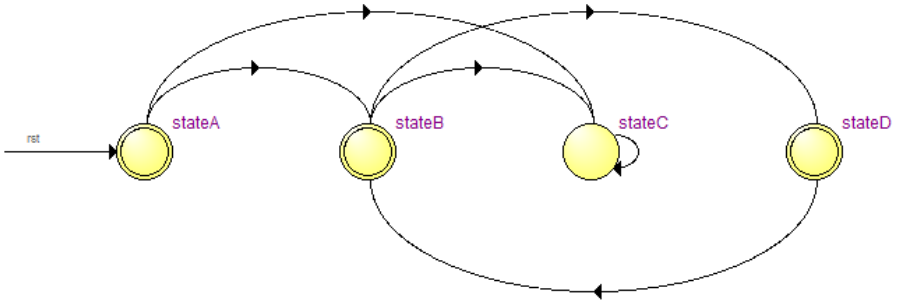
Figura 5-Fluxograma do algoritmo soma e desloca.



Fonte: Autoria Própria.

Na Figura 6, vemos o diagrama de estado da máquina que fará o controle do multiplicador as variáveis de estado (cin1, cin2) faram a lógica para a transição de um estado para outro, todas as mudanças de estados possível para o circuito estão no diagrama de estado.

Figura 6-Diagrama de estado do multiplicador soma e desloca.



Source State	Destination State	Condition
stateA	stateB	(!cin1)
stateA	stateC	(cin1)
stateB	stateC	(cin2)
stateB	stateD	(!cin2)
stateC	stateC	
stateD	stateB	

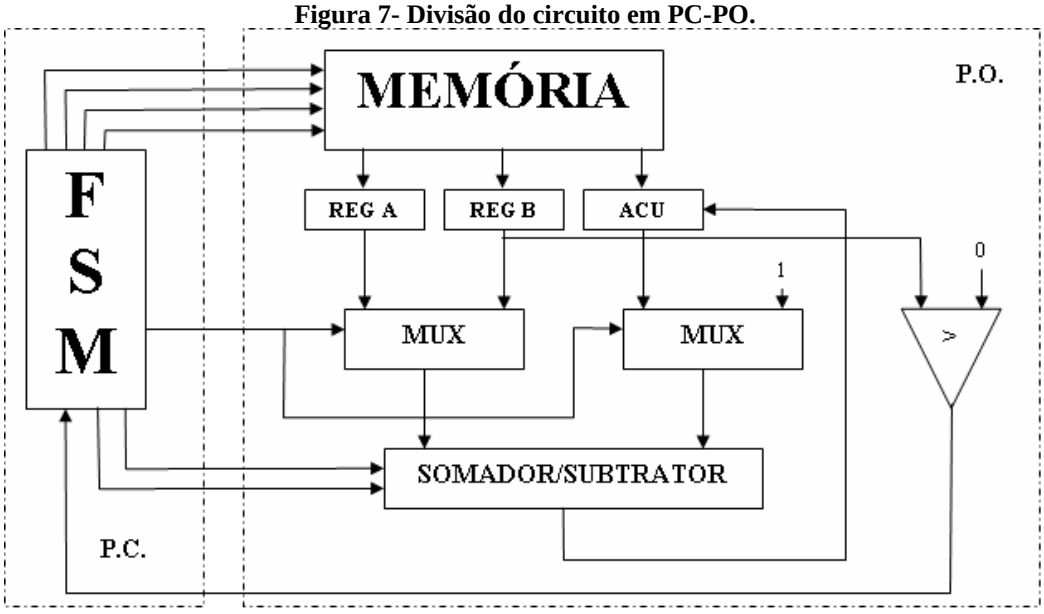
Fonte: Autoria Própria.

O circuito como um todo deverá se comportar da seguinte forma. Para iniciar o circuito multiplicador devemos levar a variável *start* ao nível alto, quando essa variável está em nível logico alto a máquina de estado entra no seu primeiro estado (*state* A), nesse estado a o registro dos valores a ser multiplicados. Uma vez registrados os valores devemos força o valor *start* para o nível logico baixo, então ocorre a verificação do bit a ser multiplicado se esse for 0 (reg B=0) então o valor da parcela é 0 e passamos para o próximo bit, se for 1 então teremos a multiplicação a soma e o deslocamento da parcela.

3.2. MULTIPLICADOR SOMA DE A,B VEZES

O algoritmo multiplicador soma de A,B Vezes realiza a multiplicação segunda as regras de multiplicação apresentadas anteriormente, o algoritmo implementa a operação de modo tradicional onde as parcelas são deslocadas para a esquerda. Podemos ver, na Figura 7 como ocorre a divisão do circuito nas partes PC-PO, onde temos parte operacional composta por registrados, multiplexadores, somadores e memória. A parte de controle é realizada por uma máquina de estado finito.

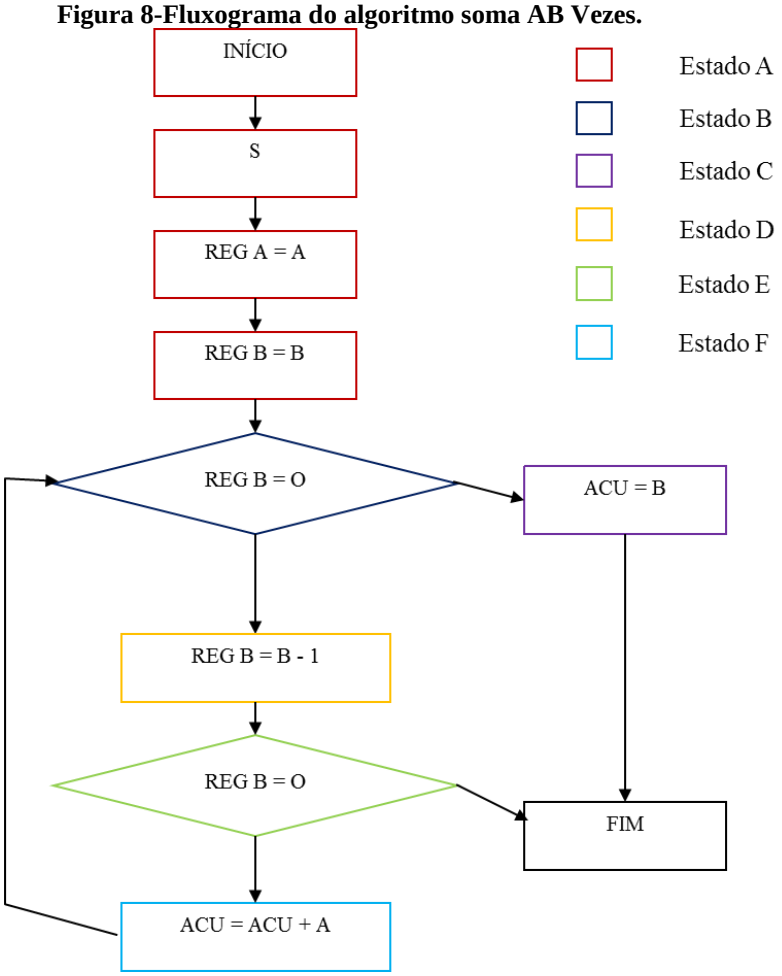
Figura 7- Divisão do circuito em PC-PO.



Fonte: Autoria Própria.

Na parte operacional o multiplicador deverá seguir a lógica descrita pelo fluxograma da Figura 8-Fluxograma do algoritmo soma AB Vezes.Figura 8, a legenda mostra em que estado da máquina é realizado cada operação.

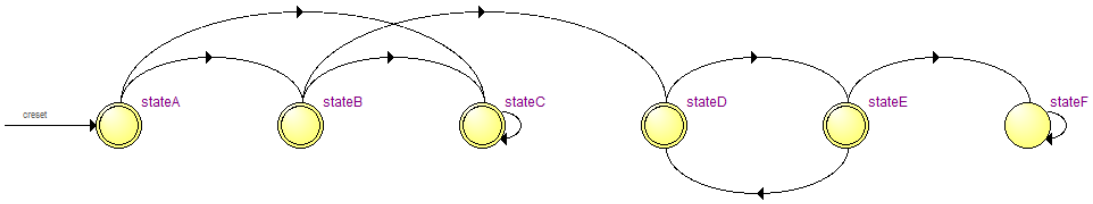
Figura 8-Fluxograma do algoritmo soma AB Vezes.



Fonte: Autoria Própria.

O diagrama de estado da Figura 9, vemos os estados do multiplicador as variáveis de estado (var1, var2, var3) faram as mudanças de estado.

Figura 9-Máquina de estado do algoritmo AB Vezes.



	Source State	Destination State	Condition
1	stateA	stateB	(!var1)
2	stateA	stateC	(var1)
3	stateB	stateC	(var2)
4	stateB	stateD	(!var2)
5	stateC	stateC	
6	stateD	stateE	
7	stateE	stateD	(!var3)
8	stateE	stateF	(var3)

Fonte: Autoria Própria.

O multiplicador como um todo deve funcionar da seguinte forma, inicialmente devemos mover a variável *start* para o nível lógico alto, quando start está em nível alto a máquina de estado inicia indo para o primeiro estado (*state* A), no estado A, nele haverá o registro dos números a serem multiplicados, após o registro dos números devemos levar a variável *start* para o nível logico baixo onde terá início a multiplicação o deslocamento e soma das parcelas.

4 ANÁLISE DOS RESULTADOS

Após o estudo dos multiplicadores e o seu desenvolvimento em VHDL, foi realizada as simulações.

4.1.SIMULAÇÕES NO MODELSIM

Utilizando a ferramenta de simulação modelsim da altera simulamos os circuitos descritos na linguagem VHDL. Os resultados de simulação, são apresentados a seguir.

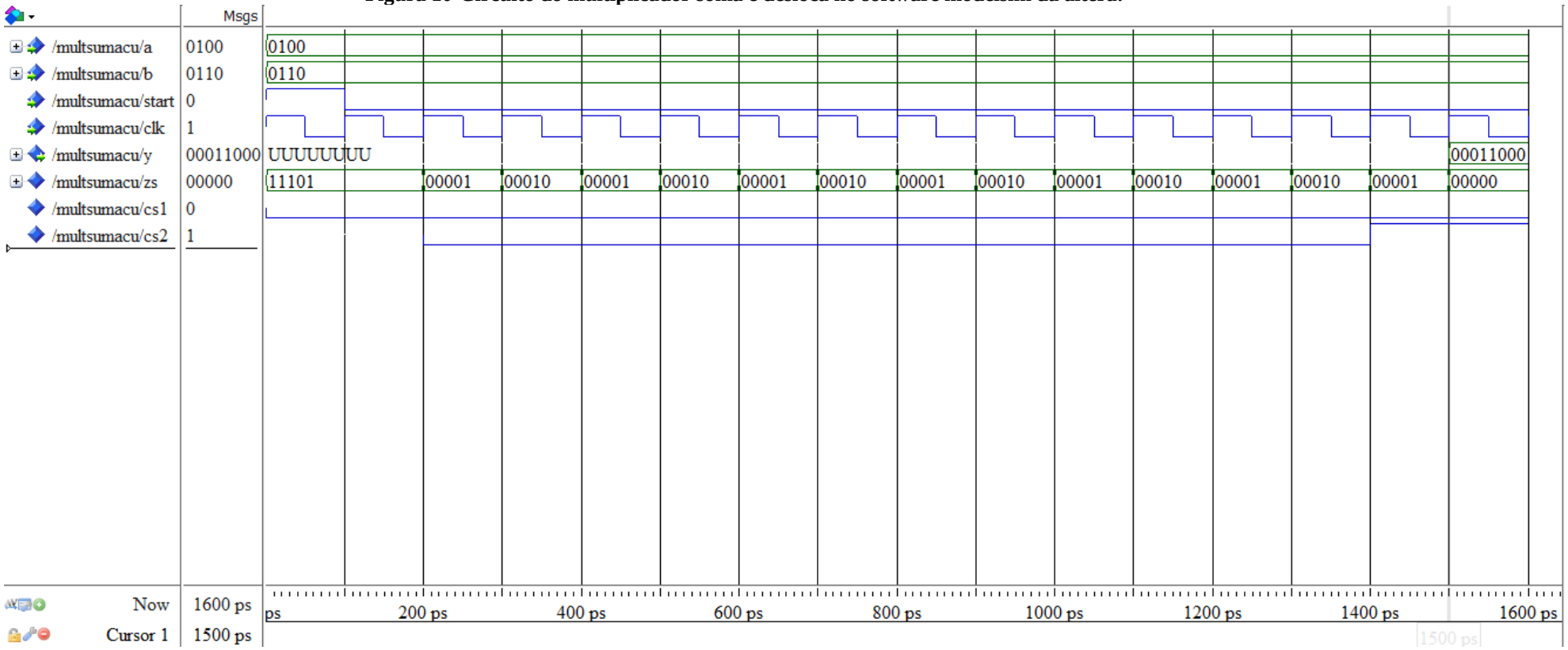
4.1.1. MULTIPLICADOR SOMA E DESLOCA

Na simulação, foram fornecidos os números binários 0100 (4 na base decimal) e 0110 (6 na base decimal), bem como um clock de 10^{-2} Hz.

A variável start inicialmente deve está no nível logico alto para levar o circuito a estado A, uma vez em A o start deverá ser alterado para o nível logico baixo para dá início a máquina de estado, pois uma vez que start igual a 1 o circuito ira para estado a independentemente de qual estado da máquina ele se encontre e lá permanecerá até que start volte a 0.

Na simulação da Figura 10 a multiplicação propriamente dita se inicia no instante t = 200 ps quando o primeiro bit a ser multiplicado entra no estado B e se encerra t = 1600 ps quando a máquina entra no estado C e imprime o resultado.

Figura 10-Circuito do multiplicador soma e desloca no software modelsim da altera.



Fonte: Autoria Própria

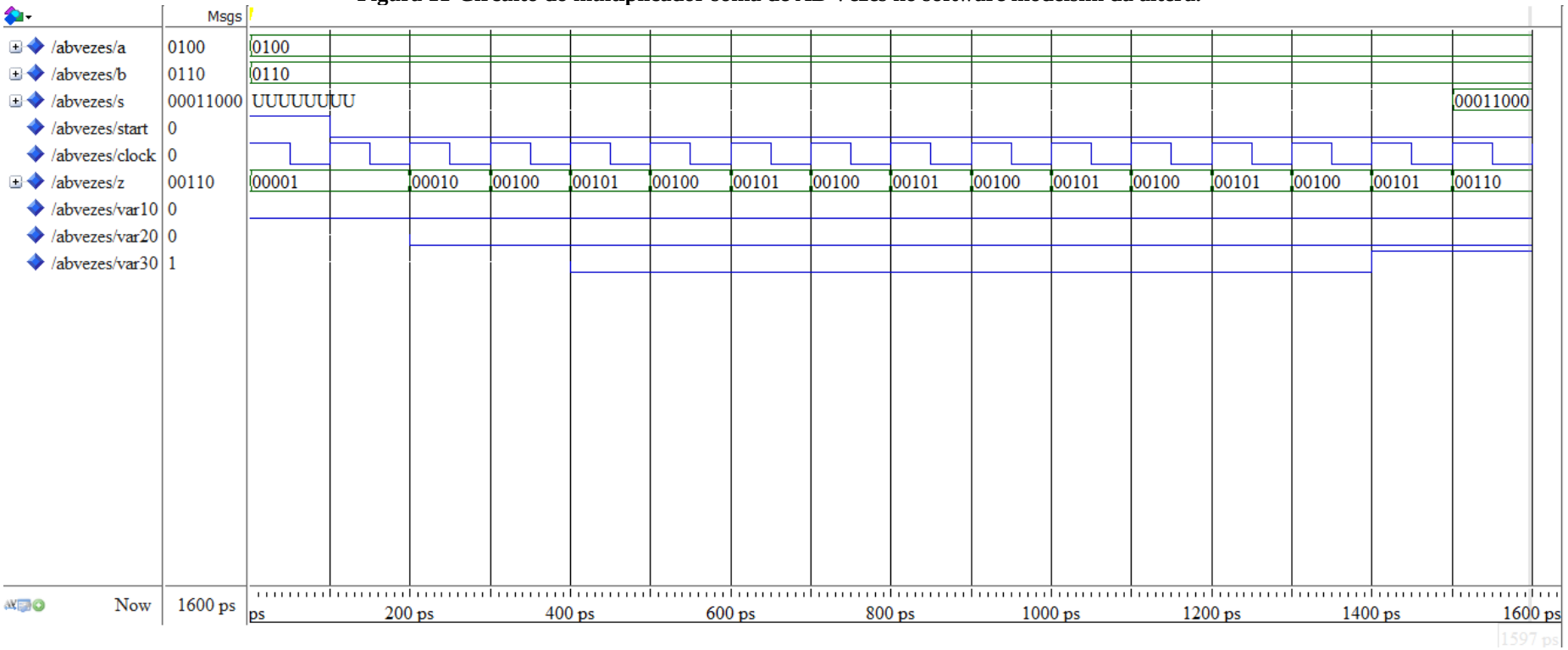
4.1.2. MULTIPLICADOR SOMA DE AB VEZES

Na simulação foram fornecidos os números binários 0100 (4 na base decimal) e 0110 (6 na base decimal), bem como um *clock* de 10^{-2} Hz.

A variável *start* inicialmente deve está no nível logico alto para levar o circuito a estado A, uma vez em A o *start* deverá ser alterado para o nível logico baixo para dá início a máquina de estado, pois uma vez que *start* igual a 1 o circuito ira para estado a independentemente de qual estado da máquina ele se encontre e lá permanecerá até que *start* volte a 0.

Na simulação da Figura 11, a multiplicação propriamente dita se inicia no instante t = 200 ps quando o primeiro bit a ser multiplicado entra no estado B e se encerra t = 1600 ps quando a máquina entra no estado F e imprime o resultado.

Figura 11-Circuito do multiplicador soma de AB Vezes no software modelsim da altera.



Fonte: Autoria Própria.

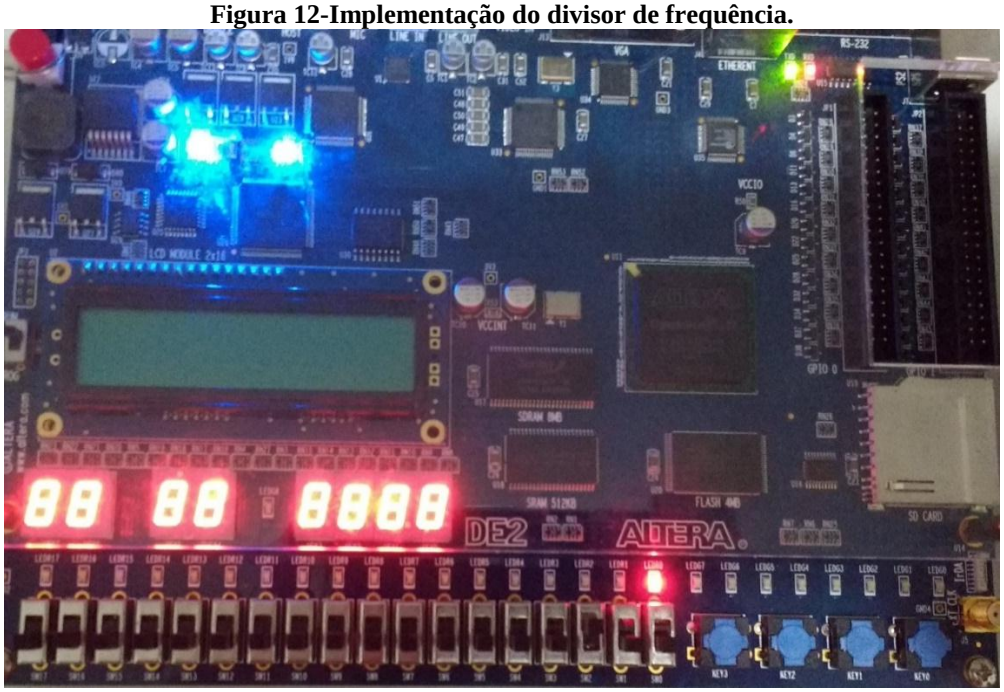
4.2. IMPLEMENTAÇÃO NA FPGA CICLONY II

A implementação da placa do FPGA se deu com a utilização do cabo USB. E da ferramenta quartus II da altera, utilizada para a descrições de hardware que será implementado, a seguir os resultados obtidos na implementação dos componentes.

4.2.1. DIVISOR DE FREQUÊNCIA

O circuito divisor de frequência foi utilizado para dividir a frequência de 50Mhz do *clock* disponível no FPGA para que um *clock* de 1 Hz para tornar possível a observação dos resultados no LEDs. Na Figura 12, podemos ver no LED vermelho os pulsos de *clock*, neste caso estamos vendo o nível lógico alto.

Figura 12-Implementação do divisor de frequência.



Fonte: Autoria Própria.

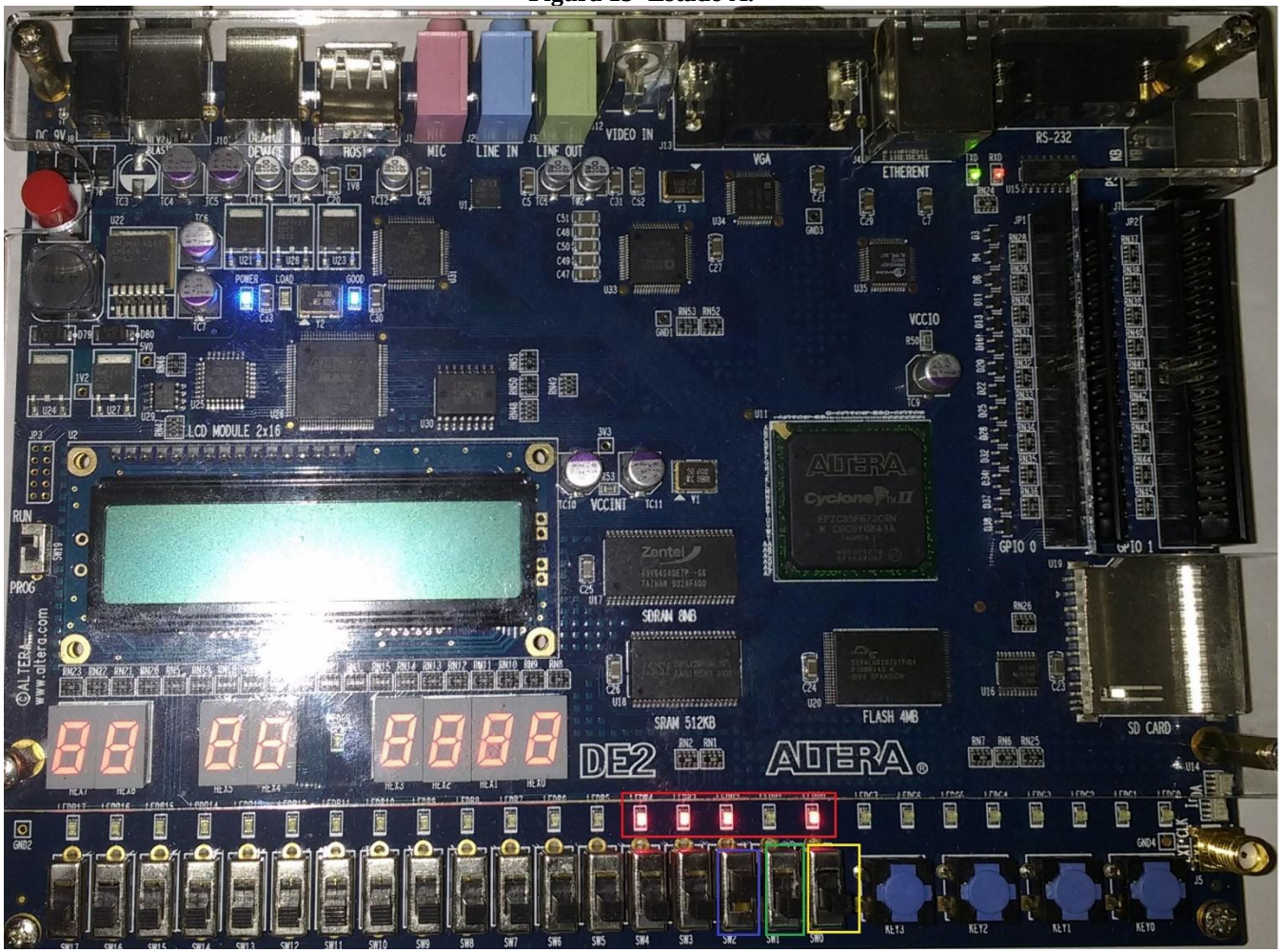
4.2.2. MULTIPLICADOR SOMA E DESLOCA

4.2.2.1. PARTE DE CONTROLE.

Na Figura 13, observamos a implementação da máquina de estado do multiplicado, onde a máquina está no estado inicial. Os LEDs imprimem os valores correspondente a cada estado, o botão *start* está destacado com a cor azul, e o das variáveis de estado cin1 e cin2 estão destacadas respectivamente verde e amarela.

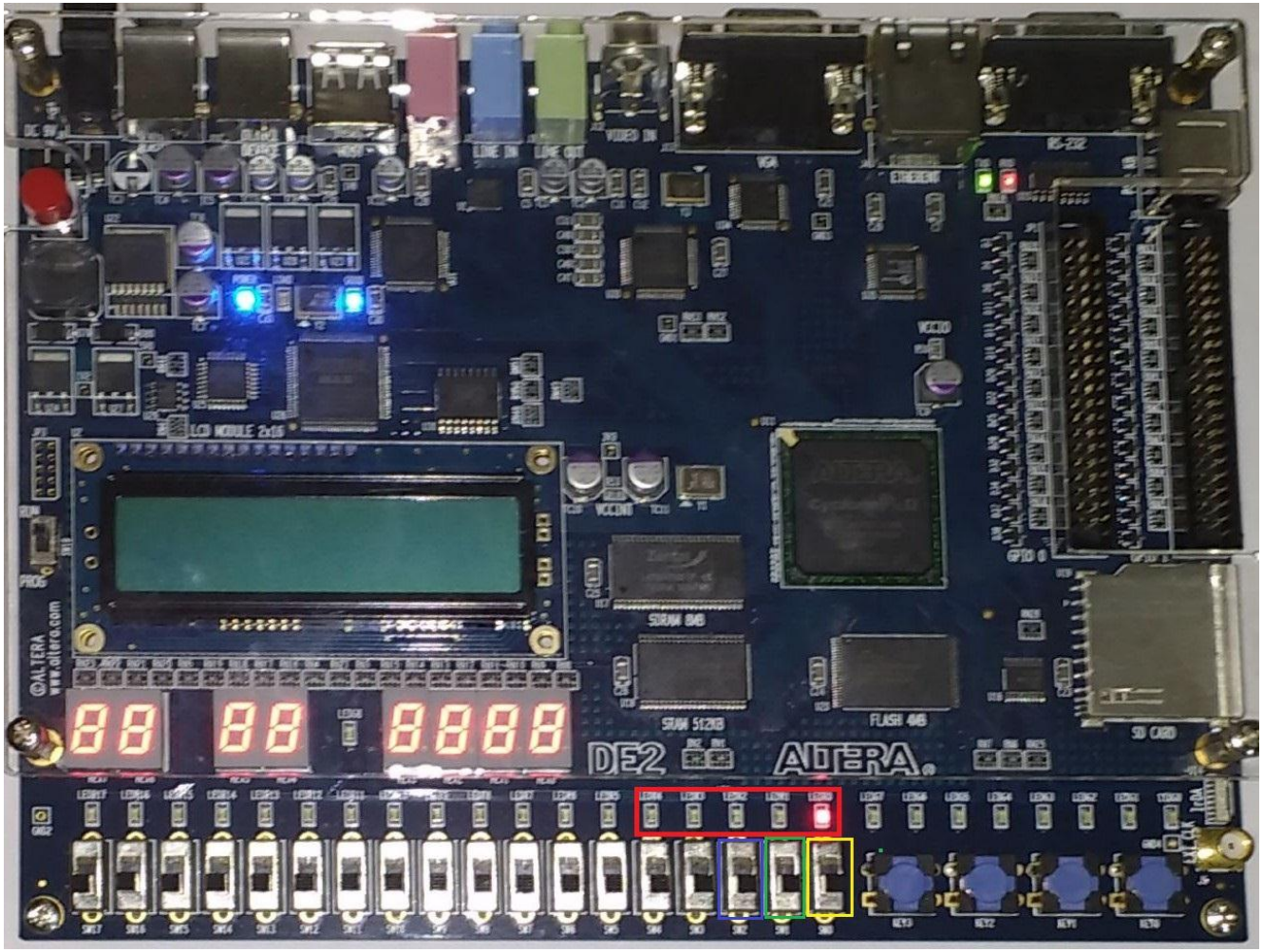
Nas demais figuras (Figura 14, Figura 15,Figura 16) temos os outros possíveis estados.

Figura 13- Estado A.



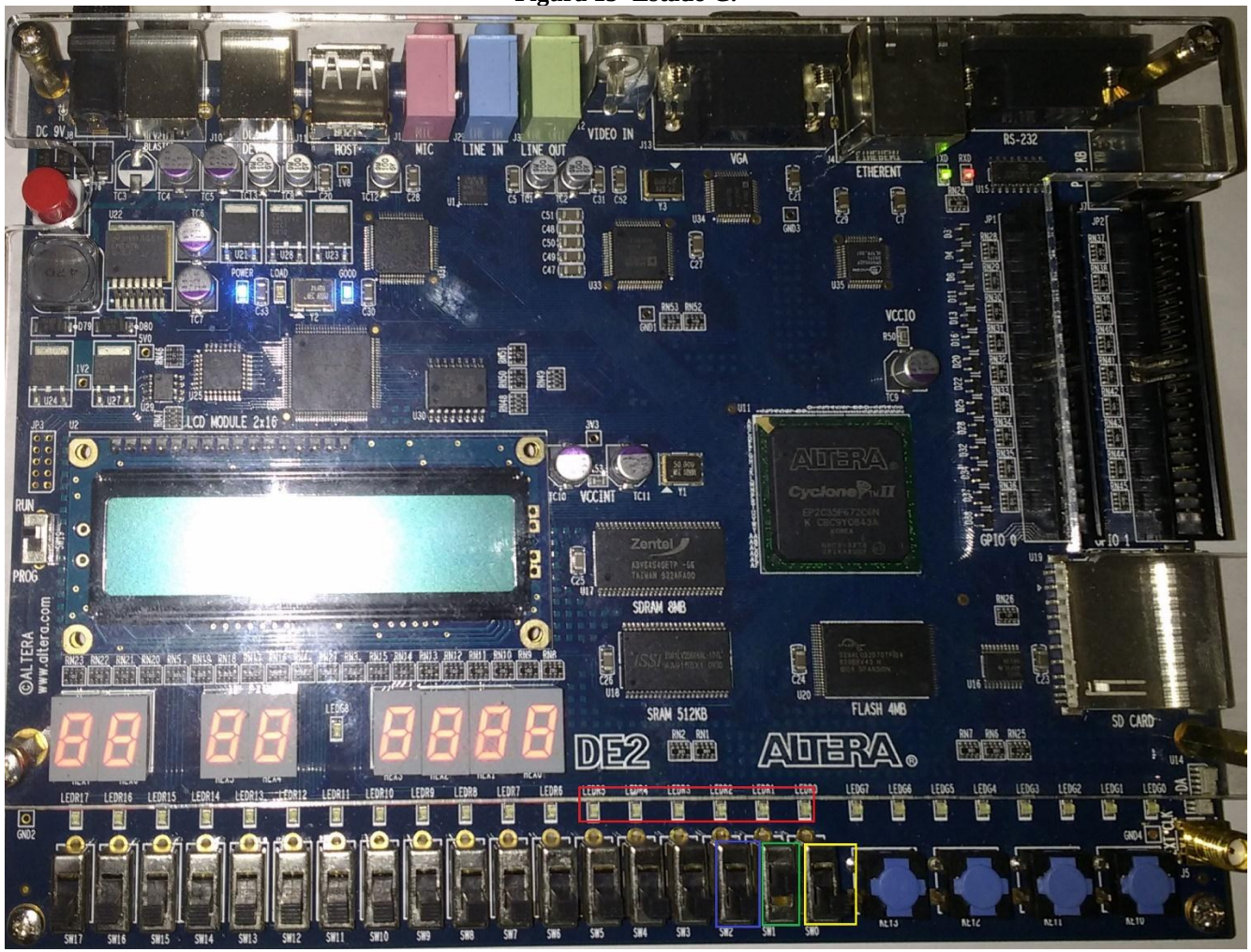
Fonte: Autoria Própria.

Figura 14- Estado B.



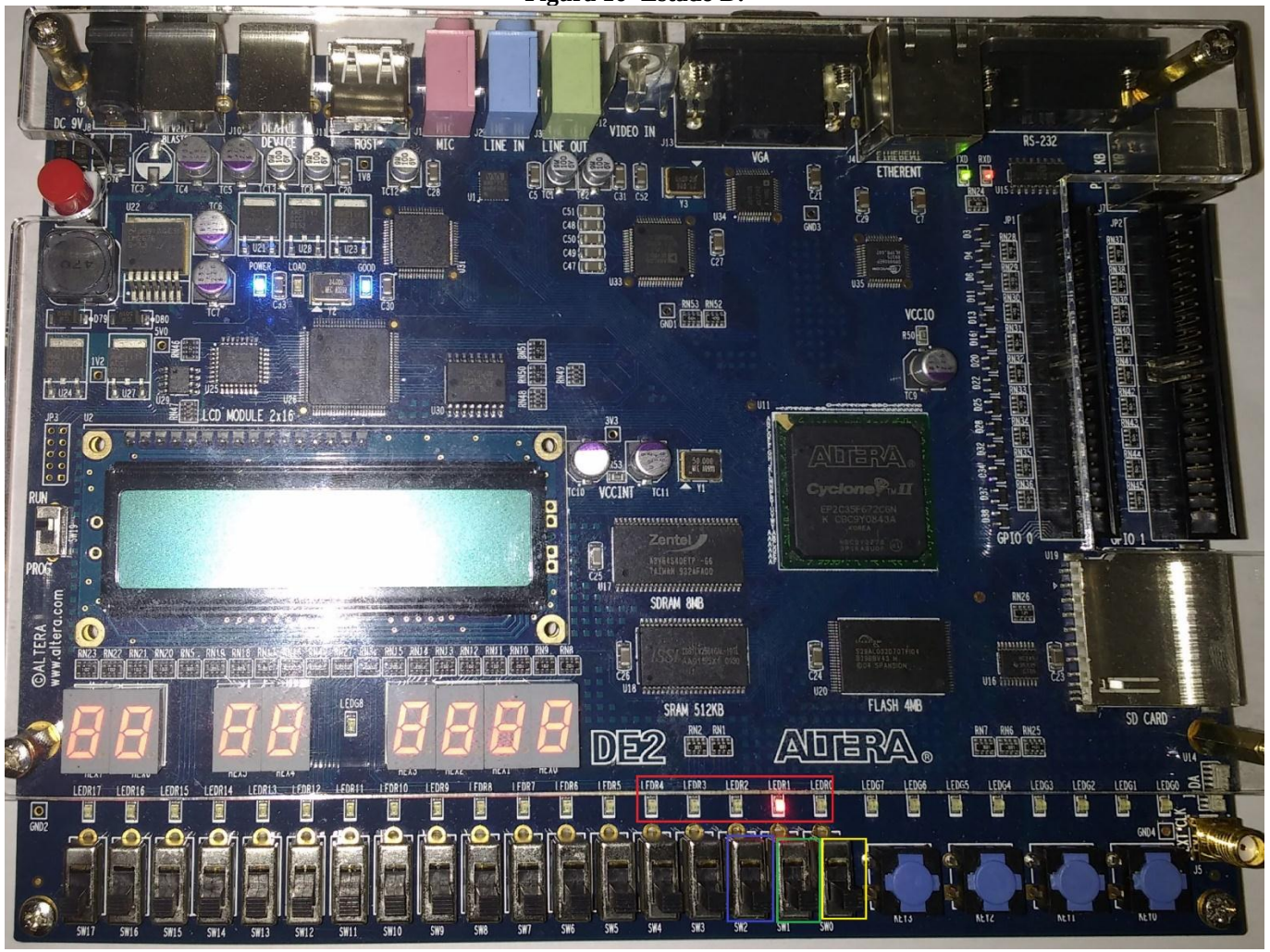
Fonte: Autoria Própria.

Figura 15- Estado C.



Fonte: Autoria Própria.

Figura 16- Estado D.



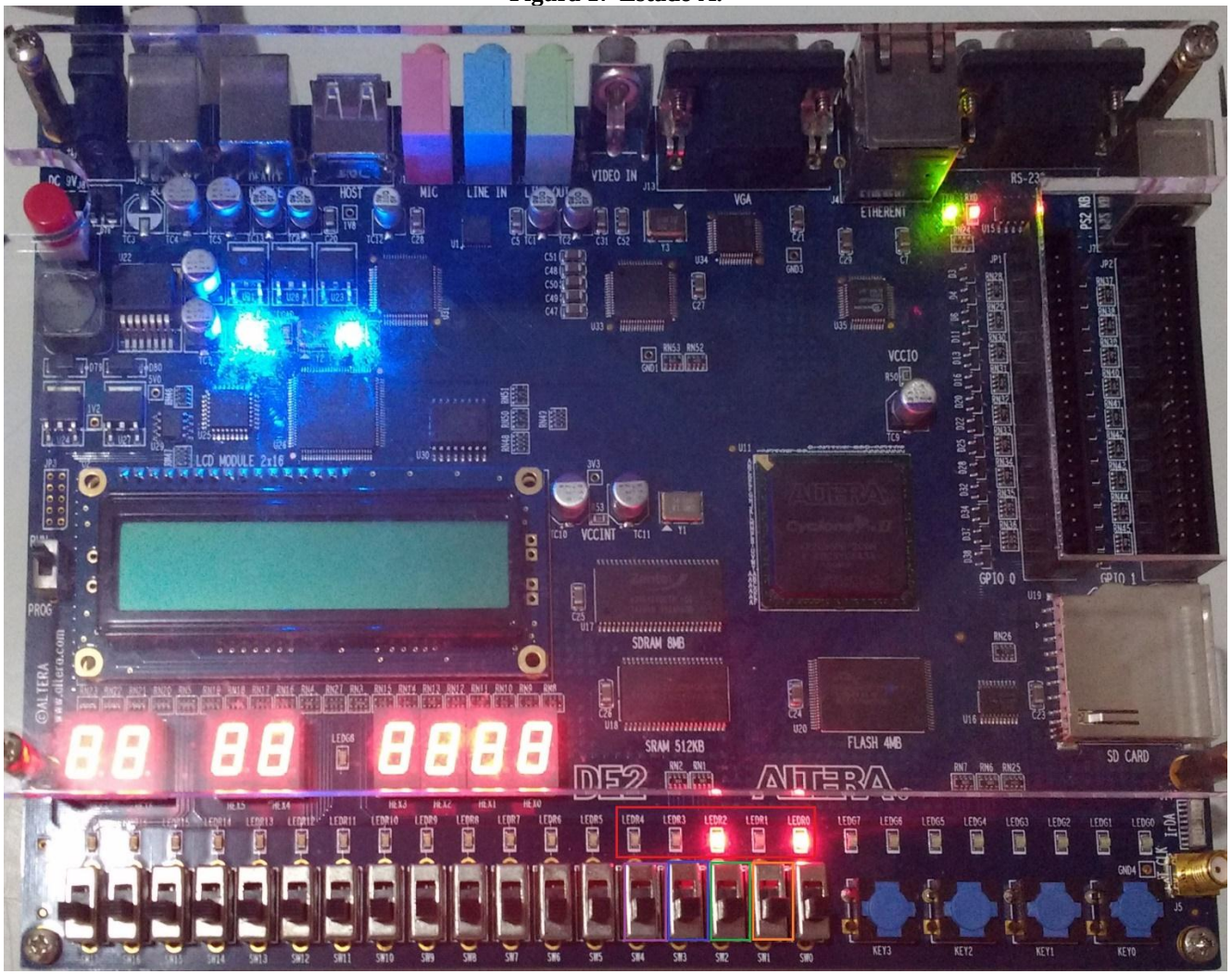
Fonte: Autoria Própria.

4.2.3. MULTIPLICADOR SOMA DE A,B VEZES

4.2.3.1. PARTE DE CONTROLE

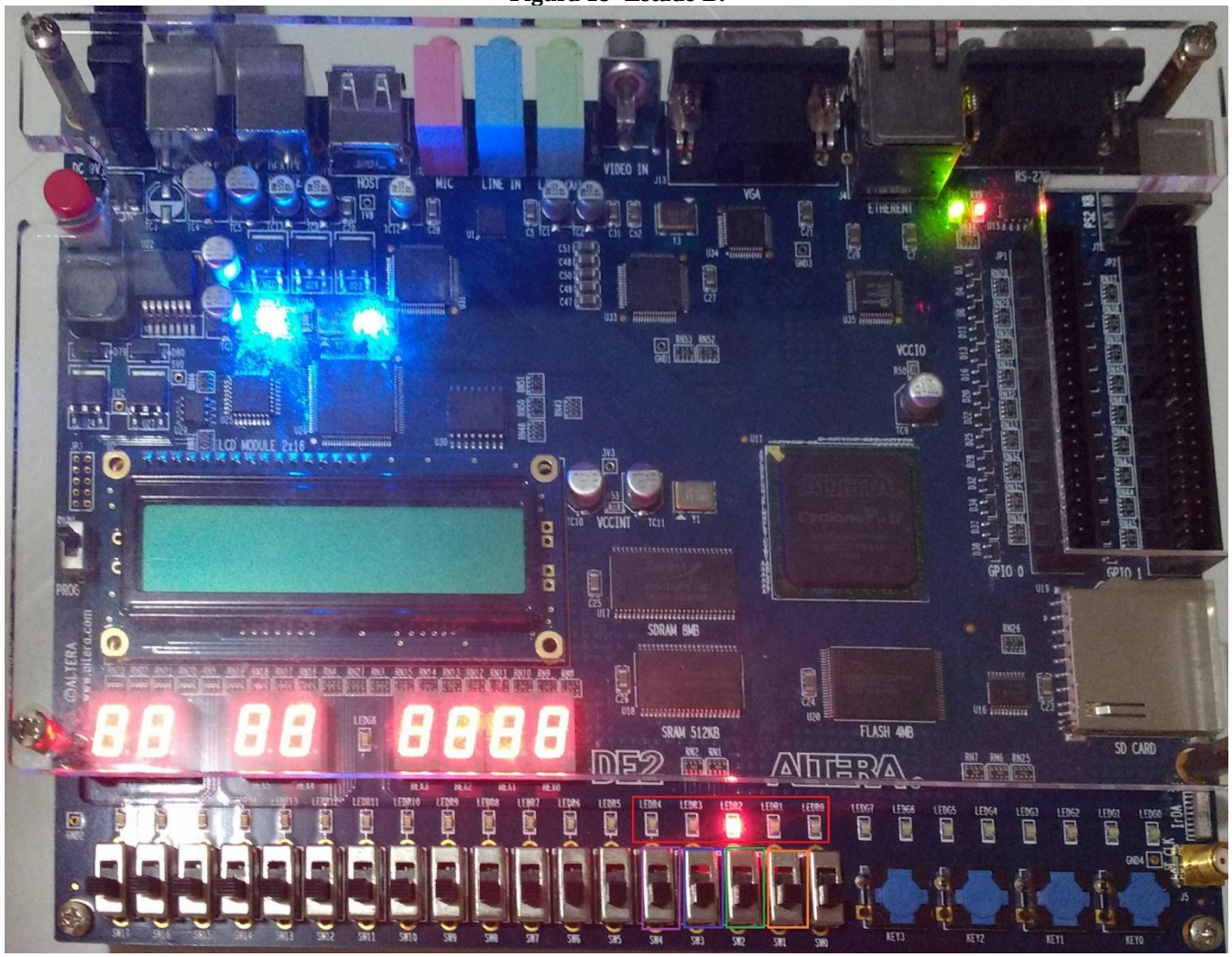
Como dito anteriormente o a máquina de estado é a parte responsável pelo controle dos estados. Na Figura 17 vemos a implementação da máquina de estado do circuito soma de A,B vezes, no seu primeiro estado, os LEDs destacados na cor vermelha imprime os resultados, o botão em *start* está na cor laranja. As variáveis (var10, var20, var30) estão disposta de esquerda para a direita com cores respectivamente verde, azul e roxo. Nas demais figuras a seguir é possível observar todos os estados da máquina (Figura 18, Figura 19, Figura 20, Figura 21, Figura 22), a legenda para os botões é a mesma para todas as figuras deste multiplicador.

Figura 17-Estado A.



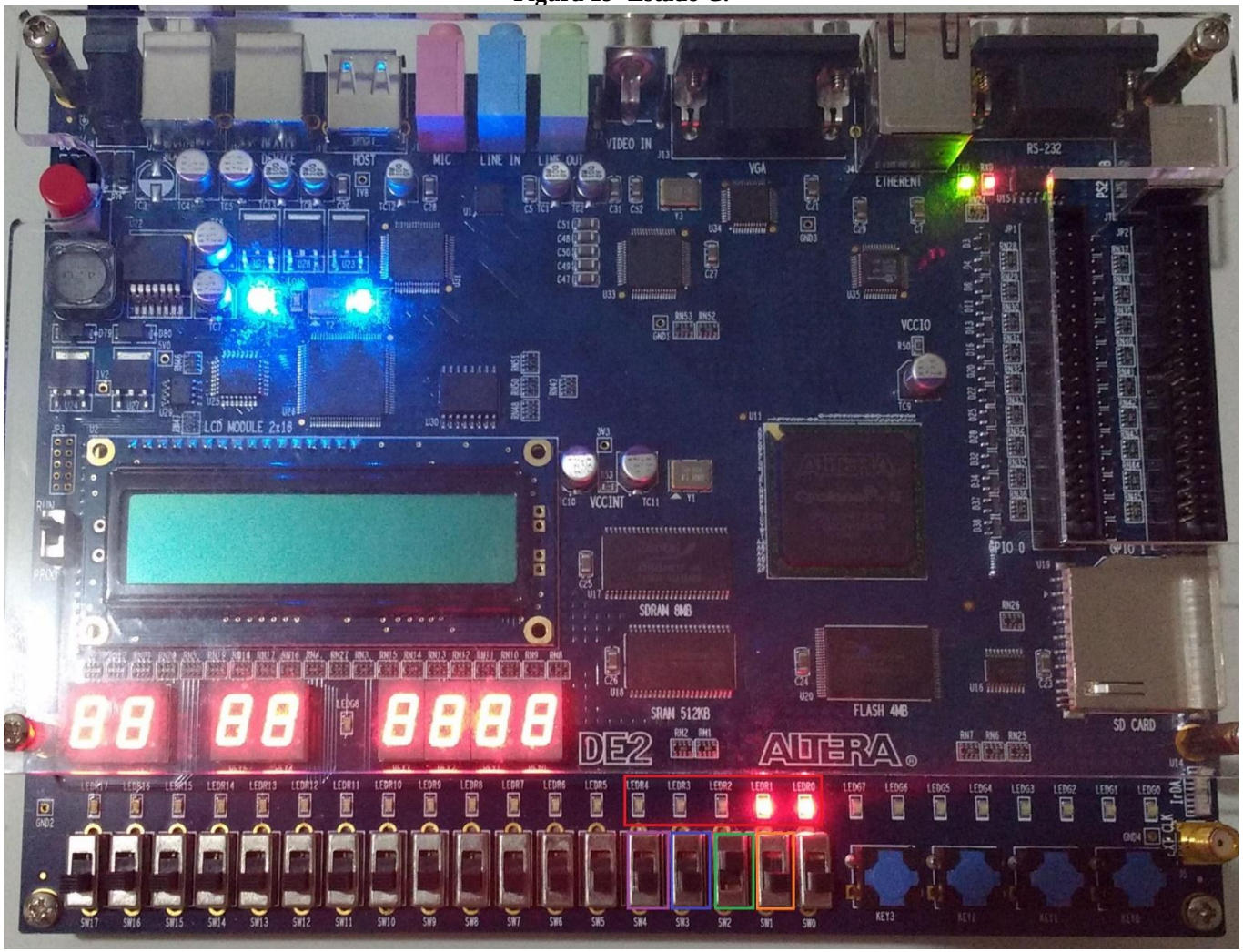
Fonte: Autoria Própria.

Figura 18- Estado B.



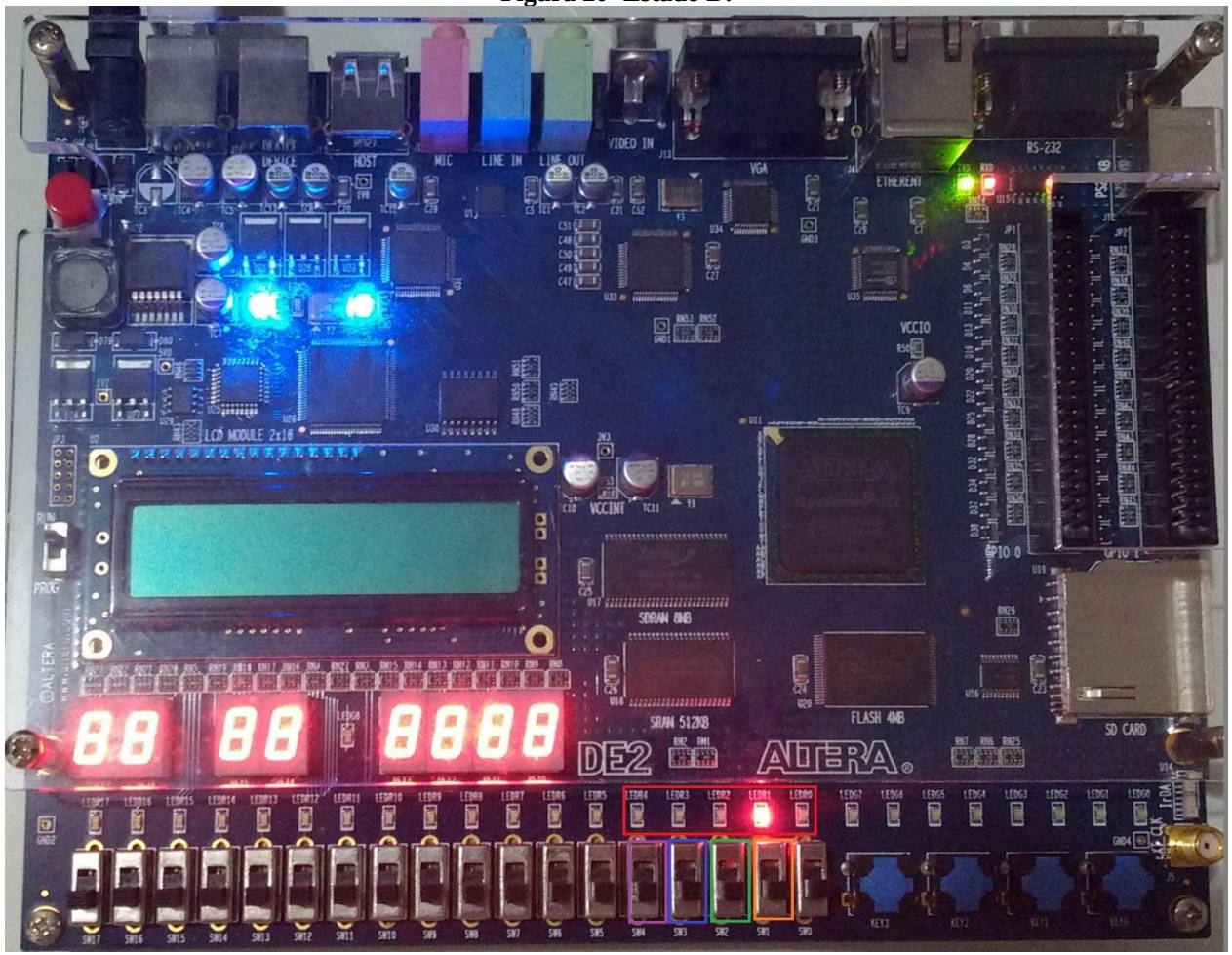
Fonte: Autoria Própria.

Figura 19- Estado C.



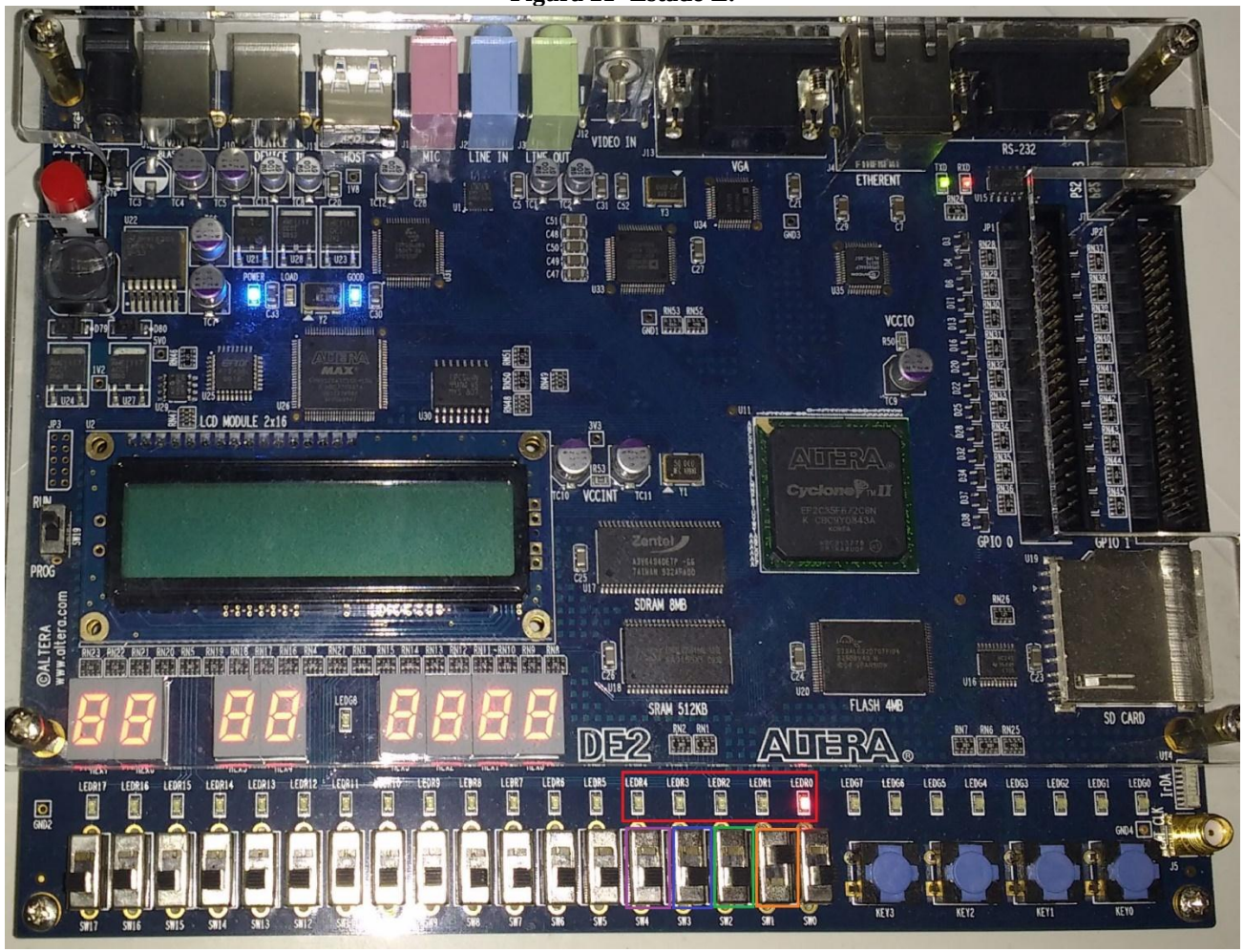
Fonte: Autoria Própria.

Figura 20- Estado D.



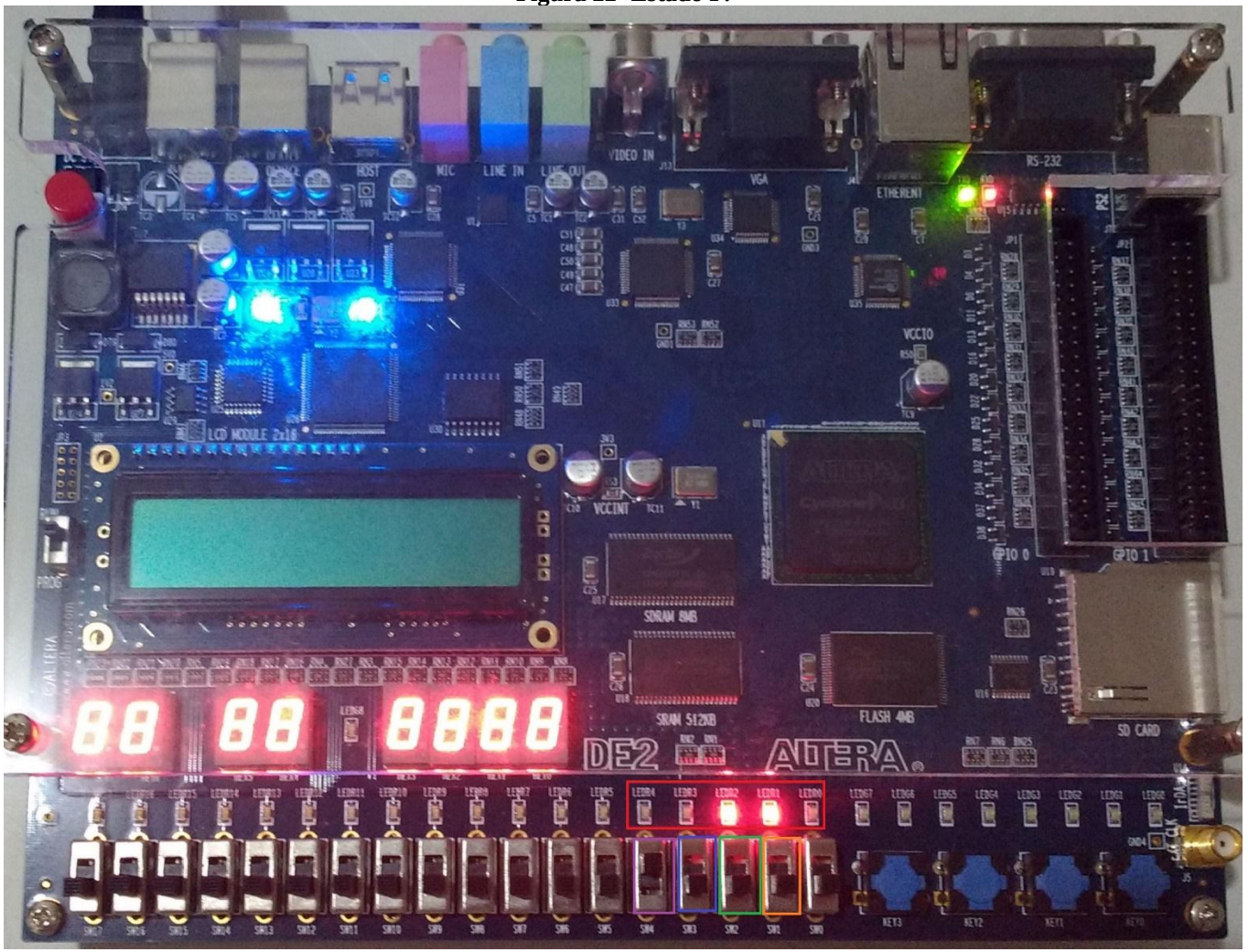
Fonte: Autoria Própria.

Figura 21- Estado E.



Fonte: Autoria Própria.

Figura 22- Estado F.



Fonte: Autoria Própria.

4.3. COMPARANDO OS CIRCUITOS

Comparando os resultados obtidos na simulação dos multiplicadores no software modelsim e da implementação na placa do FPGA Cyclone II obtemos os resultados da Tabela 3.

Tabela 3-Comparação entre os resultados das simulações no modelsim.

Multiplicador	Tempo para a decorrido para a impressão do resultado	Estados decorridos da máquina para a impressão do resultado	Registro logico dedicado	Total de elementos lógicos
Multiplicador soma desloca	1400ps	14	36	78
Multiplicador soma de A,B vezes	1400ps	14	38	137

Fonte: Autoria Própria.

Em conformidade como os parâmetros definidos por Ordonez et al. (2003) na seção de circuito multiplicador e com base nos resultados obtidos o circuito soma e desloca apresenta melhor desempenho por necessitar de menor quantidade de recursos do hardware. Porém devemos salientar que para a quantidade de bits dos números utilizados ambos tiveram o mesmo tempo de resposta.

5 CONCLUSÃO

Com o trabalho desenvolvido foi possível, realizar o desenvolvimento dos multiplicadores utilizando a metodologia PC-PO, bem como sua simulação, além de realizar as mesmas atividades para circuitos digitais relacionados a sua implementação e funcionamento como o somador e o divisor de frequência.

6 TRABALHOS FUTUROS

Para trabalhos futuros propõe-se:

- Modificar os multiplicadores para que utilizem bit de sinal e consequentemente números negativos;
- Testar multiplicadores com números binários com mais bits;

REFERÊNCIAS

DINIZ, Paulo Sergio Ramirez; SILVA, Eduardo Antônio Barros da; NETTO, Sergio Lima. **Processamento digital de sinais:** projeto e análise de sistemas. Porto Alegre: Bookman, 2004.

D'AMORE, Roberto. **VHDL:** descrição e síntese de circuitos digitais. 2. ed. Rio de Janeiro: Ltc e Gen, 2012.

FERNANDES, Victor Miranda. FPGA Altera Cyclone II. 2014. Disponível em: <http://vhdl.com.br/site/fpga/arquiteturas/117-fpga-altera-cyclone-ii>. Acesso em: 19 nov. 2015.

FLOYD, Thomas L.. **Sistemas digitais:** fundamentos e aplicações. 9. ed. Porto Alegre: Bookman, 2007.

MIDORIKAWA, E.T. Introdução às Linguagens de Descrição de Hardware. Apostila de PCS2304, 2007.

ORDONEZ, Edward David Moreno et al. **Projeto, desempenho e aplicações de sistemas digitais em circuitos programaveís (FPGAs).** Pompéia: Bless, 2003. Disponível em: <http://aberto.univem.edu.br/bitstream/handle/11077/820/LivroDacencio.pdf?sequence=1>. Acesso em: 16 jun. 2015.

SANTIAGO. Walnei. História da eletrônica. 2015. Disponível em: <**http://sabereletrico.blogspot.com.br/2010/07/historia-da-eletronica.html** >. Acessado em: 29/07/2015

SILVA, SANDRO; PANATO, ALEX; WAGNER, FLÁVIO; REIS, RICARDO; BAMPI, SERGIO. **Implementação em FPGA de um multiplicador de ponto flutuante em pipeline profundo**. In: X IBERCHIP Workshop. 2004.

TOCCI, RONALD J.; WIDMER, NEAL S. **Sistemas digitais: princípios e aplicações.** 7ª. edição. Rio de Janeiro, RJ. 2000.

UYEMURA, John P.. **Sistema digitais:** uma abordagem integrada. São Paulo: Pioneira Thomson, 2002.

APÊNDICE 1

CODIGO DO MULTIPLICADOR SOMA E DESLOCA EM LINGUAGEM DE DESCRIÇÃO DE HARDWARE VHDL.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY MultSumAcu IS--entidade principal
        PORT (a,b :  IN STD_LOGIC_VECTOR (3 DOWNTO 0);
                start,clk : IN STD_LOGIC;
                y : OUT STD_LOGIC_VECTOR (7 DOWNTO 0) );
END MultSumAcu;

ARCHITECTURE MSA_Arch OF MultSumAcu IS
        SIGNAL zs : STD_LOGIC_VECTOR (4 DOWNTO 0);-- Saída do circuito da
maquina de estado.
        SIGNAL cs1,cs2      :STD_LOGIC;-- as variáveis que fara o controle da máquina de
estado.
COMPONENT PC
        PORT (      cin1,cin2,rst,clkin     : IN    STD_LOGIC;
                    zout    : OUT STD_LOGIC_VECTOR (4 DOWNTO 0)   );
END COMPONENT;
COMPONENT PO
        PORT (      ain,bin : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
                    Zin :   IN STD_LOGIC_VECTOR (4 DOWNTO 0);
                    Yo : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
                    c1,c2 : OUT STD_LOGIC    );
END COMPONENT;
BEGIN
        I1      :       PC
        PORT MAP (      cin1    =>      cs1,
                        cin2    =>      cs2,
```

```
                    rst     =>      start,
                    clkin   =>      clk,
                    zout    =>      zs      );
        I2      :       PO
        PORT MAP    (       ain     =>      a,-- variável de entrada do multiplicador
                    bin     =>      b,-- variável  de entrada do multiplicador
                    yo      =>      y,-- variável de saida do multiplicador
                    c1      =>      cs1,-- variáveis de controlo
                    c2      =>      cs2,
                    zin     =>      zs      );-- saída da máquina de estado
END MSA_Arch;

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY PC IS-- entidade da máquina de estado
        PORT (      cin1,cin2,rst,clkin : IN        STD_LOGIC;
zout : OUT      STD_LOGIC_VECTOR (4 DOWNTO 0)    );
END PC;

ARCHITECTURE FSM OF PC IS
        TYPE STATE_TYPE IS (stateA, stateB, stateC, stateD);--estados da máquina
        SIGNAL state: STATE_TYPE;
BEGIN
        PROCESS (clkin, rst)
        BEGIN
                IF rst = '1' THEN
                        state <= stateA;
                ELSIF clkin'EVENT AND clkin = '1' THEN--borda de subida
                        CASE state IS
                                WHEN stateA =>
                                        IF cin1='0' THEN
                                                state <= stateB;
```

```
                        ELSE
                                state <= stateC;
                        END IF;
                WHEN stateB =>
                        IF cin2='0' THEN
                                state <= stateD;
                        ELSE
                                state <= stateC;
                        END IF;
                WHEN stateC =>
                                state <= stateC;
                WHEN stateD =>
                        state <= stateB;
            END CASE;
        END IF;
    END PROCESS;
    WITH state SELECT
        zout    <=      "11101"     WHEN        stateA,
                        "00001"     WHEN        stateB,
                        "00000"     WHEN        stateC,
                        "00010"     WHEN        stateD;
END FSM;
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
ENTITY PO IS
    PORT (      ain,bin : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
                Zin :   IN STD_LOGIC_VECTOR (4 DOWNTO 0);
                Yo : OUT    STD_LOGIC_VECTOR (7 DOWNTO 0);
                c1,c2   : OUT STD_LOGIC  );
END PO;

ARCHITECTURE datapath OF PO IS-- realiza a operação de multiplicação
```

```
        SIGNAL regB: STD_LOGIC_VECTOR (3 DOWNTO 0);
        SIGNAL ACC : STD_LOGIC_VECTOR (7 DOWNTO 0);
BEGIN
PROC:
PROCESS (zin,ain,bin)--o processo é sensível a mudança das entradas a saída da máquina de estado
        VARIABLE regA     :       STD_LOGIC_VECTOR (3 DOWNTO 0);
        VARIABLE regxB    :       STD_LOGIC_VECTOR (3 DOWNTO 0);
        VARIABLE xACC     :       STD_LOGIC_VECTOR (7 DOWNTO 0);
BEGIN
                IF zin = "11101" THEN -- estado A
                        xACC := "00000000";
                        regA := ain;
                        regxB := bin;
                        IF regA = "0000" THEN
                                c1 <= '1';
                        ELSE
                                c1 <= '0';
                        END IF;
                ELSIF zin = "00001" THEN-- estado B
                        IF regxB = "0000" THEN
                                c2 <= '1';
                        ELSE
                                c2 <= '0';
                        END IF;
                ELSIF zin = "00000" THEN-- estado C
                                yo <= xACC;
                ELSIF zin = "00010" THEN-- estado D
                                xACC := xACC + regA;
                                regXB := regxB - 1;
                END IF;
                ACC <= xACC;
                regB <= regxB;
```

```
END PROCESS PROC;
END datapath;
```

APÊNDICE 2

CODIGO DO MULTIPLICADOR SOMA DE A,B VEZES EM LINGUAGEM DE DESCRIÇÃO DE HARDWARE VHDL.

```
library IEEE ;
use IEEE.std_logic_1164.all ;
use IEEE.std_logic_unsigned.all ;
entity ABvezes is
port( a , b : in std_logic_vector (3 downto 0);
        s : out std_logic_vector (7 downto 0);
        start, clock: in std_logic);
end ABvezes;

architecture agindo of ABvezes is
signal z : std_logic_vector (4 downto 0);
signal var10, var20, var30 : std_logic ;

component PC
        port(clkin,creset ,var1,var2,var3 : in std_logic ;
                sout : out std_logic_vector (4 downto 0 ) );
end component;

component PO
        port(ain, bin : in std_logic_vector (3 downto 0) ;
        s0 : out std_logic_vector (7 downto 0);
        var_1,var_2,var_3 : out std_logic ;
        sin : in std_logic_vector (4 downto 0 ) );
end component;

begin
E1 : PC PORT MAP ( clkin   => clock,
                                        creset  => start,
```

```
                                        var1    => var10,
                                              var2    => var20,
                                              var3    => var30,
                                              sout    => z);

E2 : PO PORT MAP (  ain    => a,
                                               bin    => b,
                                               s0     => s,
                                               var_1  => var10,
                                               var_2  => var20,
                                               var_3  => var30,
                                               sin    => z);
end agindo;

library IEEE ;
use IEEE.std_logic_1164.all ;

entity PC is
port(clkin,creset ,var1,var2,var3 : in std_logic ;
sout : out std_logic_vector (4 downto 0 ) );

end PC;
architecture FSM of PC is
type state_type is (stateA, stateB , stateC, stateD , stateE , stateF );-- estados da máquina
signal state: state_type;
begin
process (creset,clkin)
begin
if creset = '1' then-- para iniciar a maquina essa condição deve inicialmente ser verdade
state <= stateA;

elsif clkin'event and  clkin = '1' then -- sensivel a borda de subida
```

```
case state is

when stateA =>
if   var1='0'  then
state <= stateB;
else
state <= stateC;
end if;

when stateB =>
if  var2 = '0'   then
state <= stateD;
else
state <= stateC;
end if;

when stateC =>
state <= stateC;

when stateD =>
state <= stateE;

when stateE =>
if   var3 = '0'  then
state <= stateD;
else
state <= stateF;
end if;

when stateF =>
state <= stateF;

end case;
```

```
end if;
end process;
with state select
sout <= "00001" when stateA,
                "00010" when stateB,
                "00011" when stateC,
                "00100" when stateD,
                "00101" when stateE,
                "00110" when stateF;
end FSM;

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

entity PO is
port(           ain, bin : in std_logic_vector (3 downto 0) ;
                                        s0 : out std_logic_vector (7 downto 0);
        var_1,var_2,var_3 : out std_logic ;
                                    sin :  std_logic_vector (4 downto 0 ) );
end PO;
architecture agindo of PO is

signal REGB  : std_logic_vector (3 downto 0) ;
signal busca : std_logic_vector (7 downto 0) ;
begin
process (ain, bin,sin)
variable guarA  :std_logic_vector (3 downto 0) ;
variable guarB  :std_logic_vector (3 downto 0) ;
variable acu  :std_logic_vector (7 downto 0) ;-- valores parciais

begin
```

```
if sin = "00001" then--estado A
        ACU  := "00000000";
        guarA := ain;
        guarB := bin;
        if guarA = "0000" then
        var_1 <= '1';
        s0 <= "00000000"      ;
        else
        var_1 <= '0';
        end if ;

elsif sin = "00010" then--estado B ;nesse estado ele verifica.
 if guarB = "0000" then
        var_2 <= '1';
        s0 <= "00000000";
        else
        var_2 <= '0';
        end if ;

elsif sin = "00011" then--estado C;a maquina para;
        s0 <= acu;

elsif sin = "00100" then --estado D
guarB:= guarB -1 ;
elsif sin = "00101" then--estado E
        if guarB = "0000" then
        var_3 <= '1';
        else
        var_3 <= '0';
        end if;
        ACU := ACU + ain;

elsif sin = "00110" then--estado F; a maquina para;
```

```
            s0 <= ACU;
end if;
busca<=ACU;
regB <= guarB;
end process;
end agindo;
```

APÊNDICE 3

SOMADOR COMPLETO DE 4 BITS

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

entity somador4bits is
port ( cin: in std_logic;
a,b: in std_logic_vector (3 downto 0);
cout: out std_logic;
s: out std_logic_vector (3 downto 0));
end somador4bits;

architecture arch_somador4bits of somador4bits is
begin
process(a, b, cin) -- executado na alteração do valor "a"

variable soma: std_logic_vector(3 downto 0);
variable c: std_logic;
begin
c := cin;

for i in 0 to 3 loop
soma(i) := a(i) xor b(i) xor c;
c := (a(i) and b(i)) or ((a(i) xor b(i)) and c);
end loop;

cout <= c;
s <= soma;
end process;
end arch_somador4bits;
```

APÊNDICE 4

DIVISOR DE FREQUÊNCIA

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

 ENTITY Div_Ferq IS
 GENERIC ( R : INTEGER := 49999999);

 PORT( clk_IN    : IN     STD_LOGIC  ;
      clk_rst   : IN     STD_LOGIC  ;
      clk_OUT   : BUFFER   STD_LOGIC );

END Div_Ferq;

ARCHITECTURE funcional OF Div_Ferq IS
BEGIN

PROCESS (clK_IN)
 VARIABLE count : INTEGER := 0;
 BEGIN
 if clk_rst = '1' then
  clk_out <= '0';
 else
 IF R /= 1 THEN
 IF (clk_IN'event AND clk_IN = '1') THEN
  count := count + 1 ;
  IF( count = R) THEN

     CLK_OUT <= NOT CLK_OUT;
     count := count + 1;
```

```
 END IF;
 If (count = R+49999999)then
   CLK_OUT <= NOT CLK_OUT;
   count := 1;
 end if;
END IF;
ELSE
CLK_OUT <= CLK_IN;
END IF;
end if;
END PROCESS;
END funcional;
```