UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

RELATÓRIO

ACELERADOR ARCO-TANGENTE – LUT E CORDIC

Disciplina: Projeto e Prototipagem de Sistemas Digitais

Professor: Fernando Rangel

Alunos: Adauto Luís

Rafael Magalhães

Hélio Souza

Natal / 2006

Índice

1. Introdução

Este relatório tem como finalidade apresentar duas propostas para solução do cálculo do arco tangente de um par de coordenadas especificadas, apresentar os passos do projeto, sua implementação, seus resultados, e dificuldades encontradas.

O relatório está estruturado da seguinte forma: motivação, descrição dos métodos utilizados, o projeto usando cada método, a implementação de cada método, os resultados, e uma conclusão.

2. Motivação

Muitas são as aplicações onde se faz necessário o cálculo do arco-tangente, como modulação para transmissão de dados por exemplo. Por se tratar de uma função transcedental, seu cálculo não é trivial, necessitando de um algoritmo matemático para sua resolução.

Em sistemas digitais, vislumbramos duas possibilidades para sua resolução.

A primeira se trata da identificação de quadrantes, verificando as coordenadas passadas como entrada do sistema, e dependendo de seus sinais se escolhe o valor apropriado numa tabela contendo todos os valores possíveis calculados e corrigidos. Para esse método usou-se as Look Up Tables, ou comumente conhecidas de LUTs.

A segunda possibilidade se baseia na implementação de um algoritmo matemático com características que possibilitam sua aplicação em sistemas digitais. Esse algoritmo se chama CORDIC (Coordinate Rotation Digital Computer), e se trata de um algoritmo iterativo baseado em aproximações. Ele pode ser usado para resolução de diversas funções trigonométricas, que como o arco-tangente, tem sua resolução não trivial.

3. Cálculo do Arco Tangente

Por definição de projeto, duas serão as entradas de dados, três as entradas de controle, e uma saída de dados contendo o valor da operação. As entradas de dados serão representadas pelas letras X e Y, e as entradas de controle por clk, start, e reset, com as funções de relógio, início e reinicialização do sistema respectivamente. A saída se dará por Z, em:

$$Z = \arctan\left(\frac{\mathrm{Y}}{\mathrm{X}}\right)$$

Onde Z representa o ângulo entre as duas coordenadas dadas.

Como dito anteriormente, esse cálculo será feito de duas formas. Uma utilizando tabelas de pesquisa(LUTs) e outra através de um algoritmo iterativo(CORDIC).

A seguir cada método é descrito e implementado.

4. Look Up Tables (LUTs)

4.1. Descrição

A sistemática para resolução do problema através de LUTs será a da divisão do círculo trigonométrico em 8 diferentes setores.

Cada setor é identificado através de comparações entre os dados de entrada, fornecendo onde o ângulo se encontrará, para que seja feita a devida correção, e ter como saída o valor correto do ângulo.

A figura a seguir mostra a divisão do círculo trigonométrico em diferentes setores.

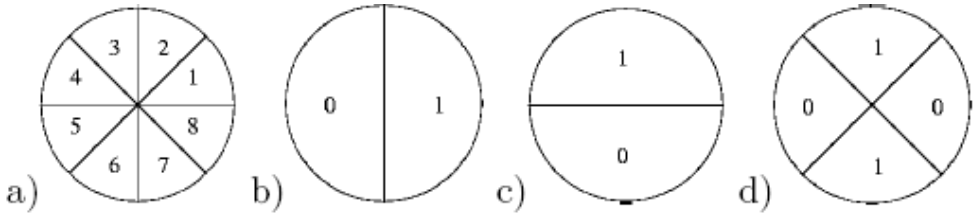


Figura 1: a) Divisão do círculo trigonométrico em oito setores. b) Divisão de setores para análise de A. c) Divisão de setores para análise de B. d) Divisão de setores para análise de C.

A identificação de setores se dará pelos valores de A, B e C, onde são verificados se X > 0, se Y > 0, e se X > Y respectivamente.

A tabela de correção para cada setor está mostrada logo a seguir.

C	B	A	Setor	Correção
0	0	0	5	$Y - \pi$
0	0	1	8	$-Y$
0	1	0	4	$\pi - Y$
0	1	1	1	Y
1	0	0	6	$-\frac{\pi}{2} - Y$
1	0	1	7	$-\frac{\pi}{2} + Y$
1	1	0	3	$\frac{\pi}{2} + Y$
1	1	1	2	$\frac{\pi}{2} - Y$

Tabela 1: Correção do ângulo para cada setor.

Na tabela 1 , Y representa o ângulo Z.

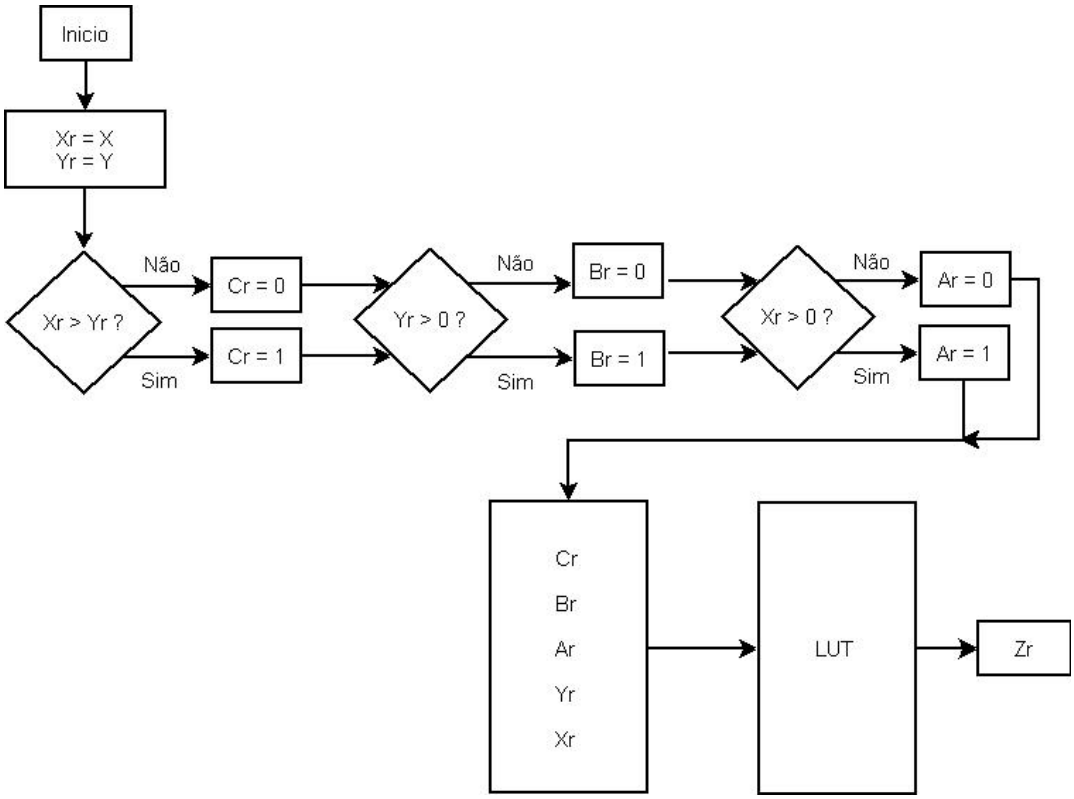
Todas as possibilidades de ângulos serão guardadas, já corrigidas numa tabela. A partir dessa tabela, se retira o valor desejado através de um código de endereçamento.

4.2. Implementação

Para a implementação, considerou-se como código de endereçamento os sinais: Cr, Br, Ar, Yr, Xr, nessa ordem no MSB para o LSB.

Na tabela estão guardados os valores dos ângulos referentes as entradas, já corrigidos para os testes de A, B e C, referentes a que setor pertence o ângulo.

A seguir um fluxograma descrevendo o comportamento do sistema.



Fluxograma 1: Esquemático do sistema usando LUT.

A metodologia PC-PO foi adotada para a implementação do sistema.

A PC é responsável pela transição de estados, checagem de status e envio de comandos.

A PO recebe os comandos da PC, processa as informações, e envia sinais de status quando requisitada.

4.3. Particionamento PC-PO

Nesta seção está apresentada a máquina de estados do sistema, seu diagrama esquemático, a tabela de transferência de registradores, o RTL da PO, e as interconexões entre a PC e a PO.

O diagrama esquemático da máquina de estados é apresentada na Figura 2.

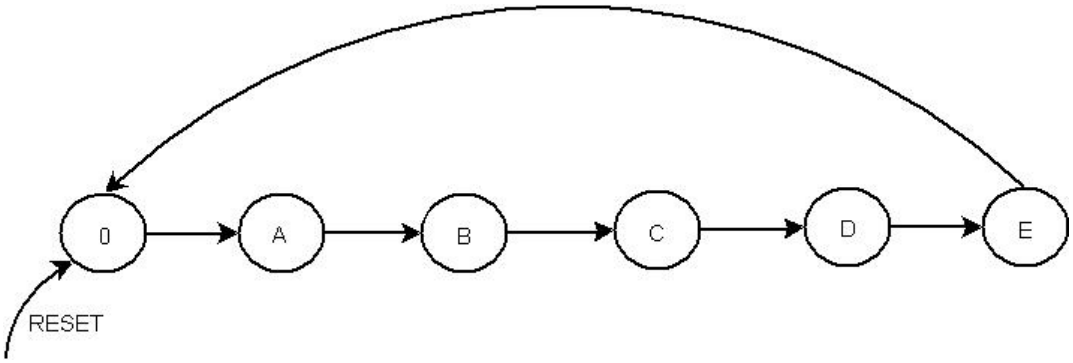


Figura 2: Máquina de Estados – LUT

Especificação:

- Estado 0: Resultado depois de um reset.
- Estado A: Carrega registradores Xr e Yr.
- Estado B: Cr = 1 se Xr > Yr, senão 0.
- Estado C: Br = 1 se Yr > 0, senão 0.
- Estado D: Ar = 1 se Xr >0, senão 0.
- Estado E: Carrega registrador Zr com resultado da saída da memória. Volta para estado 0.

A partir do projeto da máquina de estados é possível montar a tabela de transferência de registradores, que define as operações realizadas na PO, descrevendo por estado, os registradores envolvidos, a operação envolvida, e uma descrição da ação do estado.

Estado	Xr	Yr	Cr	Br	Ar	Zr	Operação
0							Reset.
A	IN(X)	IN(Y)					Carrega registradores.
B			Xr > Yr				Compara Xr, Yr.
C				Yr > 0			Compara Yr.
D					Xr > 0		Compara Xr.
E						Lê(Cr,Br,Ar,Xr,Yr)	Carrega em Zr resultado da memória.

Tabela 2: Tabela de Transição de Registradores

Uma vez com a tabela de transição de registradores, e a máquina de estados definida em mãos, é possível projetar o RTL e a partir dele, passar para o código de descrição de hardware.

Levando em consideração a escolha por um sistema rápido, foi decidido projetar um sistema utilizando multiplexadores, e carregamentos paralelos.

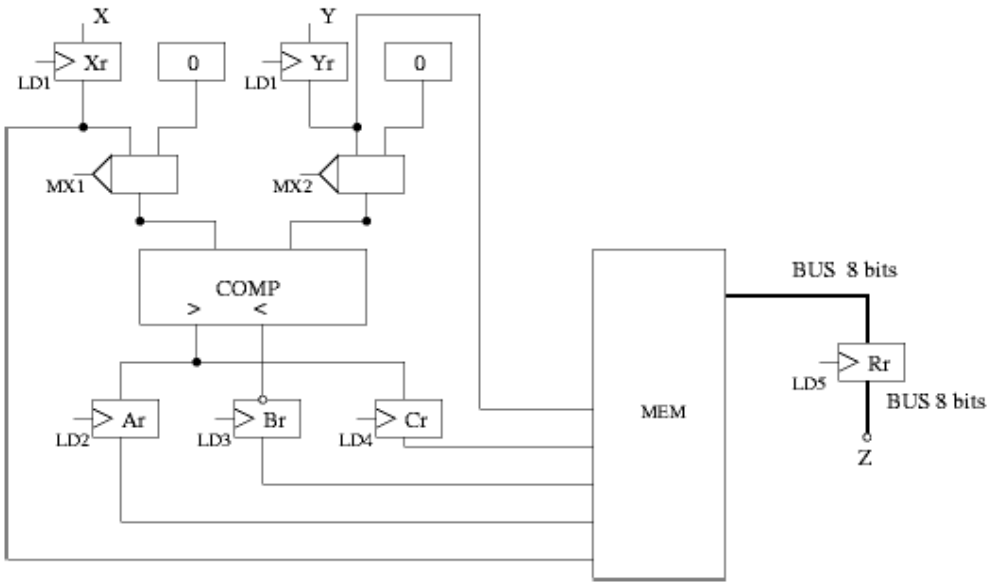


Figura 3: RTL da PO.

A figura 3 mostra o RTL da Parte Operativa do sistema.

Para que o sistema fique bem descrito, com todas as entradas, saídas, e conexões bem definidas, e dessa forma facilitar a implementação em código, é comum fazer a partição entre PC e PO, e mostrar, graficamente, suas entradas saídas, e interconexões.

Na figura 4 está mostrado tal partição.

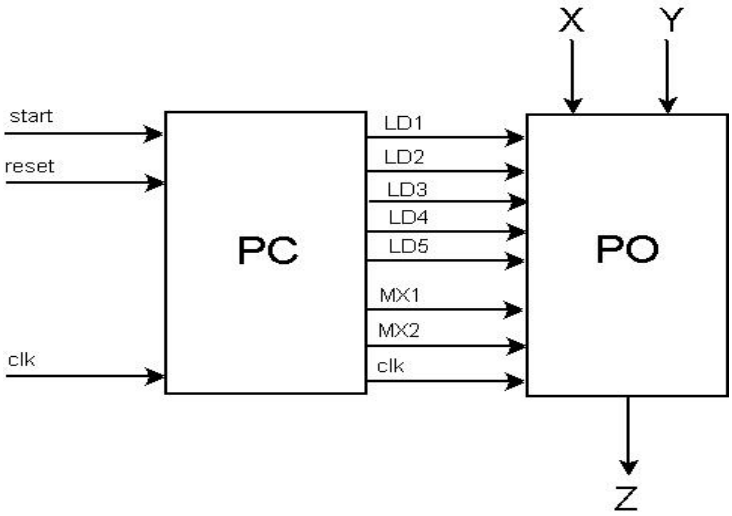


Figura 4: Interconexões - LUT

4.4. Resultados

O sistema apresentou um comportamento bem definido, passando de estado por estado, no momemto apropriado e sob os comandos corretos.

A simulação do sistema foi feita no software Quartus da Altera.

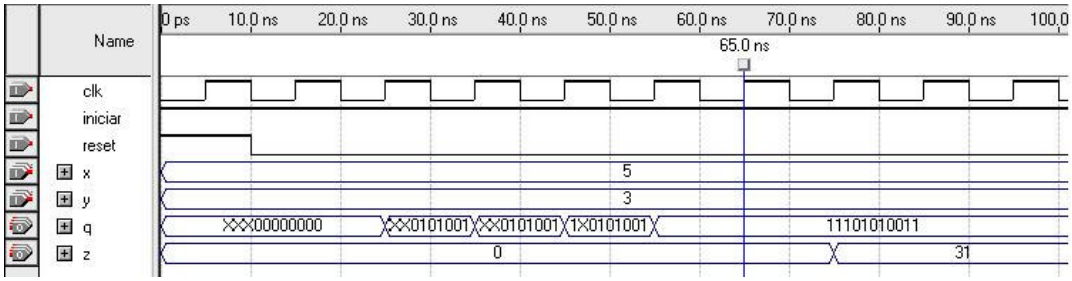


Figura 5: Resultado da simulação.

Vê-se pela figura 5 que os estados estão bem definidos. É válido salientar a necessidade de um estado “morto”, uma vez que o FPGA escolhido para simulação não dava suporte a uso de memória assíncrona. Colocou-se um estado inativo entre os estados D e E, apenas para que a resposta fosse a correta logo de início, sem necessidade de loop no sistema.

Pela figura 5 também é possível verificar a rápida resposta do sistema, em tornor de 75ns.

5. CORDIC

5.1. Descrição

O CORDIC, ou Coordinate Rotation Digital Computer, trata-se de um algoritmo iterativo utilizado para resolução de funções trigonométricas, e função transcedentais[1].

Ele baseia-se na rotação de vetores e atualização por incremento das variáveis envolvidas. Por natureza possui três modos de operação, denotados pela letra m. Modo de operação m = 1, para funções circulares, modo m = 0 para funções lineares, e modo m = -1 para funções hiperbólicas.

O CORDIC pode ser descrito por:

$$\begin{bmatrix} X_{r+1} \\ Y_{r+1} \end{bmatrix} = \begin{bmatrix} 1 & m\delta_k 2^{-k} \\ m\delta_k 2^{-k} & 1 \end{bmatrix} \begin{bmatrix} X_k \\ Y_k \end{bmatrix}$$

$$Z_{k+1} = Z_k + \delta_k \theta_k,$$

donde, *m* é o modo de operação, δ é o sinal aplicado, θ é o incremento do ângulo.

Com uma escolha apropriada do modo de operação, o mesmo algoritmo pode ser usado para calcular outras funções trigonométricas.

No modo de operação, m = 1, há duas direções de rotação. Fazendo Y_r tender a 0, ou fazendo Z_r tender a 0. Para o cálculo do arco tangente, se faz a rotação com Y_r tendendo a 0, já que para este modo o valor de Z_k toma a seguinte forma,

$$Z_k = Z_0 \pm arctan(Y_0/X_0).$$

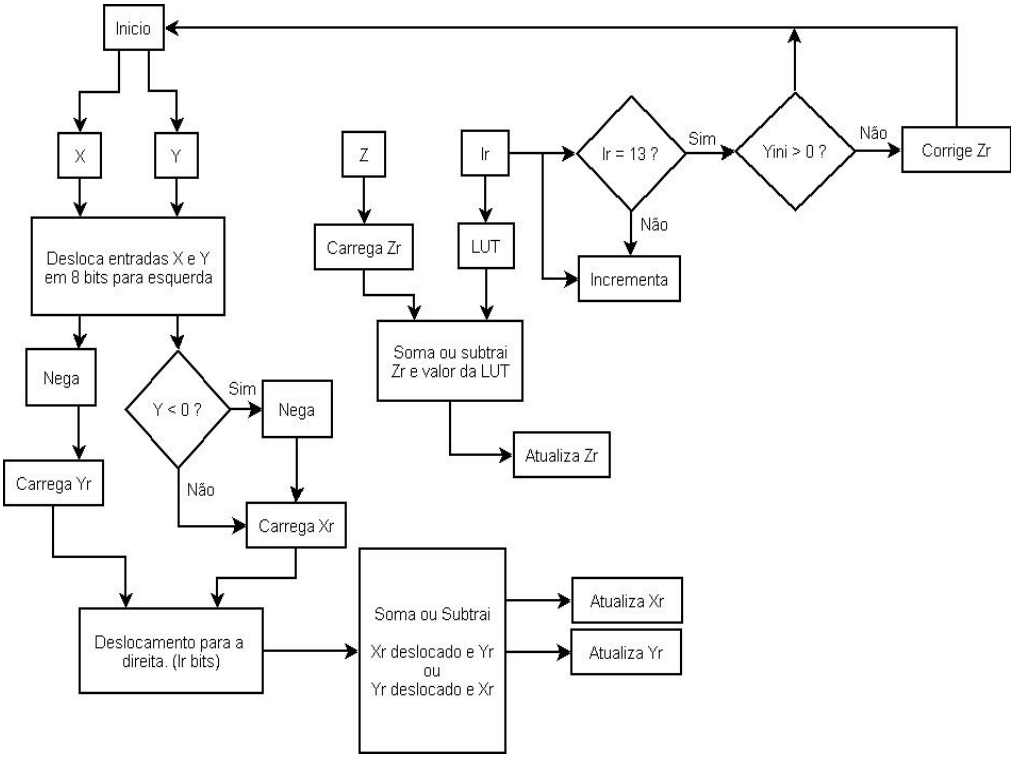
Os valores dos arco tangentes incremento são armazenados numa LUT, e se escolhido um valor inicial apropriado, o valor de Z_k converge para um resultado aproximado do arco tangente das coordenadas passadas como entrada.

5.2. Implementação

Para a implementação usando o CORDIC, inicialmente foi feito um fluxograma para representar o sistema, e a partir daí, usou-se uma linguagem de alto nível para validar o fluxograma.

A linguagem de alto nível utilizada foi a C, e seu código fonte está transcrito no apêndice ao final deste relatório.

O fluxograma feito para representar o sistema, está mostrado a seguir.



Fluxograma 2: Esquemático - CORDIC.

Com o fluxograma em mãos, e já validado pela linguagem de alto nível, passou-se a fase de criação da máquina de estados que representa o fluxo da informação dentro do sistema.

A máquina de estados está mostrada na figura 6, e logo em seguida, detalhada.

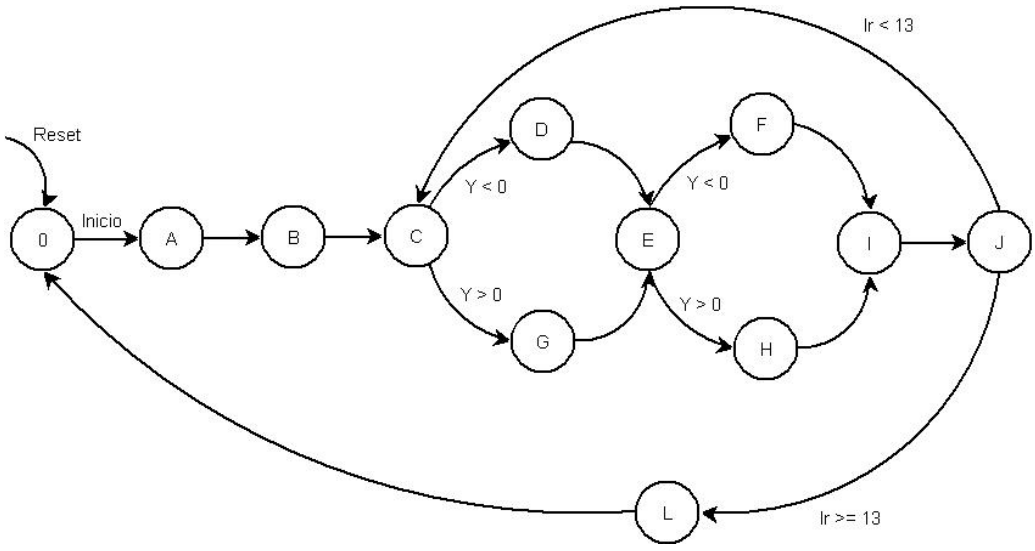


Figura 6: Máquina de Estados - CORDIC.

Especificação:

- Estado 0: Espera pelo sinal de início. Estado destino após término ou reset.

- Estado A: Aplica deslocamento inicial de 8 bits para a esquerda em X e Y. Carrega registrador Zr e Yr.
- Estado B: Carrega Xr e Yr. Se Y < 0, inverte o valor de Y. Guarda valor da comparação em Aux_r.
- Estado C: Aplica deslocamento para direita igual ao valor de Ir em Yr, verifica se Yr < 0.
- Estado D: Subtrai as entradas do somador / subtrator 1, e atualiza Xr_n.
- Estado E: Aplica deslocamento para direita igual ao valor de Ir em Xr.
- Estado F: Adiciona as entradas do somador / subtrator 1, atualiza Yr, seta soma no somador / subtrator 2.
- Estado G: Adiciona as entradas do somador / subtrator 1, e atualiza Xr_n.
- Estado H: Subtrai as entradas do somador / subtrator 1, atualiza Yr, seta subtração no somador / subtrator 2.
- Estado I: Atualiza Zr e Xr, seta soma no somador de iteração.
- Estado J: Atualiza Ir, verifica se menor que 13.
- Estado L: Se Aux_r = 0, inverte Zr. Volta para estado 0.

A partir da máquina de estados, é possível criar a tabela de transferência de registradores.

Estados	Xr	Yr	Zr	Ir	Xr_n	Aux_r	Operações
0							Espera início.
A			Z	1			Aplica BS em X e Y, carrega Zr e Ir.
B	X	Y				Y < 0 ?	Carrega Xr e Yr.
C							BS em Yr, Yr>0?
D					Xr-Ybs[1]		Atualiza Xr_n.
E							BS em Xr.
F		Yr+Xbs					Atualiza Yr.
G					Xr+Ybs		Atualiza Xr_n.
H		Yr-Xbs					Atualiza Yr.
I	Xr_n		Zr±LUT				Atualiza Xr e Zr.
J				Ir+1			Incrementa Ir, Ir<13?
L			-Zr				Atualiza Zr, se Aux_r = 0. Volta pro início.

Tabela 3: Transferência de Registradores.

Da tabela de transferência de registradores e da máquina de estados, é possível se obter o RTL da PO, bem como as interconexões entre PC e PO.

[1] Ybs é utilizado para denotar Y após o deslocamento. Mesmo para Xbs.

Na figura 6 está mostrado o RTL da PO.

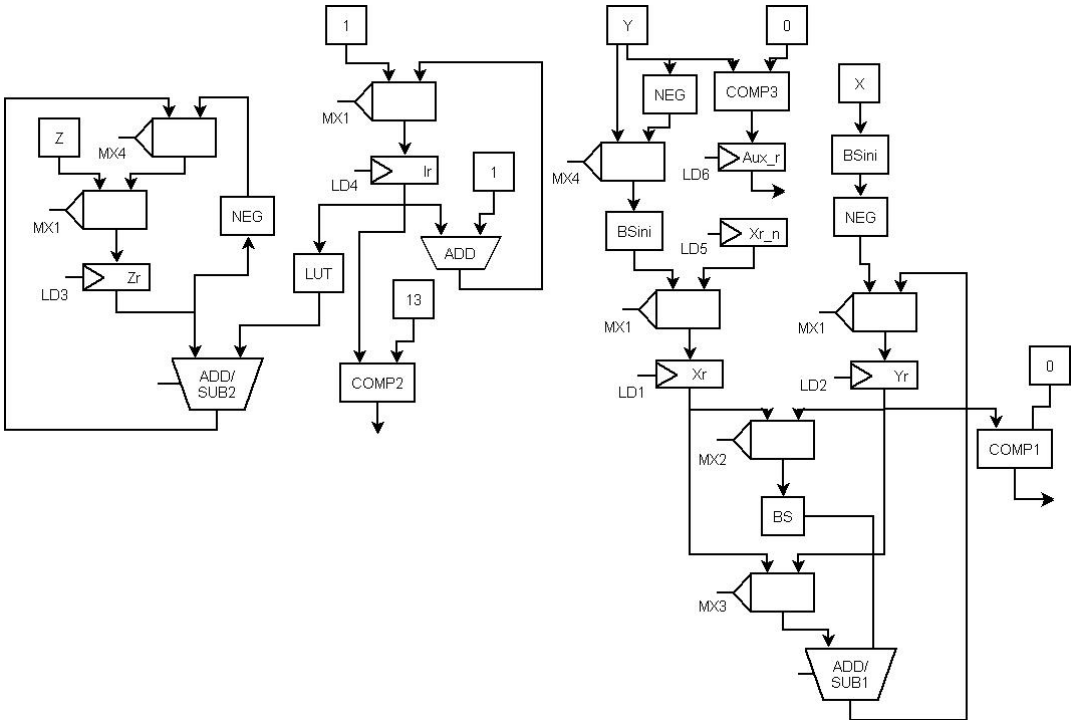


Figura 7: RTL da PO.

Com o RTL feito, passamos para a identificação das conexões, e posterior criação do particionamento entre PC e PO.

O RTL mostra uma malha de verificação dos sinais de Y de entrada, de Yr e de Ir. O Ir é um registrador usado como contador para indicar quando as iterações necessárias foram concluídas. O teste de Yr denota no valor de δ, ou seja, o sinal de atualização de Xr, Yr e Zr. O teste de Y no início é para indicar será necessária uma correção no valor de Zr ao final do processo.

Foi verificado no algoritmo em C, e suportado por algumas fontes que o CORDIC funciona de 0 a 180°. Necessitando de uma correção caso se queira uma completa excursão do círculo trigonométrico.

A correção de Zr é simplesmente uma substituição de sinal, uma vez que o sinal de Y será mudado no início caso ele seja menor que zero. Dessa forma se obtém ângulos de 0° a 180° e de 0° a -180°.

O sinal de saída Z_k tem 16 bits. Apesar de extrapolado em precisão, esse é o número de bits necessário para representar o menor incremento como um valor superior a zero.

Por tanto, a saída do sistema estará entre 180° e -180°.

Na figura 8 está mostrado tal particionamento.

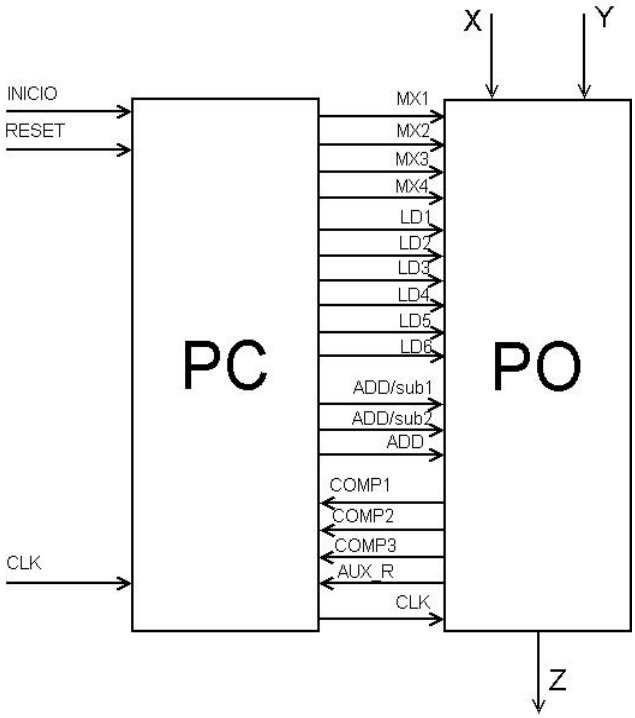


Figura 8: Particionamento PC-PO.

Na figura 8 são verificadas as conexões entre PC e PO, detalhando quais são os sinais de comando enviados, e quais os sinais de status recebidos pela PC, para determinar o comportamento do sistema.

5.3. Resultados

Alguns problemas foram encontrados na implementação do CORDIC em VHDL.

Notou-se que o sistema teve um comportamento correto, em relação à máquina de estados, entretanto a saída não foi a esperada.

Vários testes, de compatibilidade e funcionalidade dos blocos envolvidos, foram feitos, porém sem o êxito esperado.

Especula-se que o problema possa estar no fato da representação a complemento de 2 dos números negativos, e a escolha de número de bits das palavras para representar tais números. De qualquer forma, verificou-se os passos de comparações necessários para mudança adequada dos estados, e a própria mudança adequada. A atualização das variáveis está defeituosa.

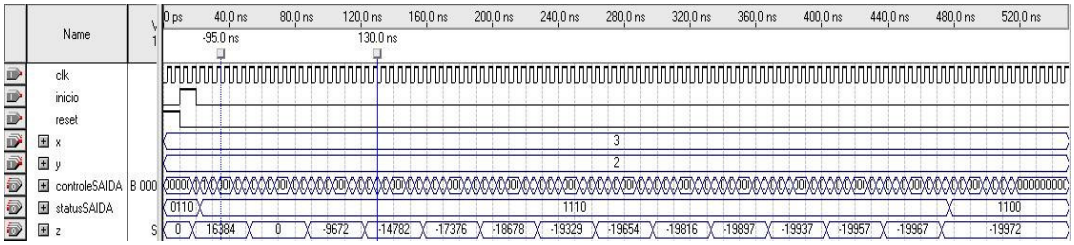


Figura 9: Simulação – CORDIC

Pela figura 9 é possível verificar as 13 iterações do sistema pelos 13 valores de Z, resposta do sistema. E a resposta após 520ns do começo da simulação.

Os sinais controleSAIDA e statusSAIDA representam os sinais entre PC e PO, apresentados na simulação para facilitar a depuração do sistema.

Conclusão

Apesar de muito utilizado, o algoritmo CORDIC se mostrou menos eficiente se comparado, em termos de resposta, ao método utilizando LUTs. Entretanto não há a necessidade de se ter um banco de memória previamente calculado e armazenado no dispositivo.

Analisando o tempo de resposta, foi verificado que a latência do sistema quando usado com CORDIC se deu em torno de 520ns, e quando usado com LUTs, o tempo de resposta teve uma diminuição considerável, em torno de 75ns.

Apesar do método utilizado para verificar esses tempos – Simulação no Altera Quartus –, é possível tirar as conclusões supra citadas com uma certa segurança.

De cada implementação, ao menos uma forte recomendação pode ser tirada. Caso a implementação seja através de LUTs, o projetista precisa vislumbrar bastante as possibilidades de endereçamento, sob pena de ter que preencher tabelas muito grandes, com endereços muitas vezes desencontrados. Caso a implementação seja através do CORDIC, o projetista precisa estar com todos os passos bem planejados, e bem documentados para não cair no problema de ter que rever o código à procura de erros.

O trabalho foi de grande valia para nos dar uma introdução à vivência com projetos de sistemas digitais, fundamentar um tanto da linguagem VHDL, e nos acostumarmos com o nível em que a tecnologia de sistemas digitais se encontra.

Bibliografia

[1] Andraka, Ray; A Survey of CORDIC Algorithms for FPGA Based Computers; Andraka Consulting Group, Inc.

[2] A.Th. Schwarzbacher,A. Brasching, Th.H. Wahl, P.A. Comiskey2 and J.B. Foley; Optimization and Implementation of the Arctan Function for the Power Domain; Dublin Institute of Technology.

[3] T. Vladimirova, H. Tiggeler; FPGA Implementation of Sine and Cosine Generators Using the CORDIC Algorithm; Surrey Space Centre.

[4] T. Trandafir; Fixed Point Two's Complement CORDIC Arithmetic on MSP430; Microtrend Systems, Inc.

Apêndice

Neste apêndice serão transcritos os códigos-fonte das implementações descritas neste trabalho.

- LUT:

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

ENTITY popc_atan_vhdl IS
  PORT
  (
     reset                                  : IN     STD_LOGIC;
     iniciar                                : IN     STD_LOGIC;
     clk                                    : IN     STD_LOGIC;
     x,y                                    : IN     STD_LOGIC_VECTOR(3 DOWNTO 0);
     z                                      : OUT    STD_LOGIC_VECTOR(8 DOWNTO 0);
     q                                      : OUT    STD_LOGIC_VECTOR(10 DOWNTO 0)
  );
END popc_atan_vhdl;

ARCHITECTURE a OF popc_atan_vhdl IS
        SIGNAL vetor : STD_LOGIC_VECTOR(6 DOWNTO 0);

COMPONENT pc_atan_vhdl
    PORT
    (
        clk                                                 : IN     STD_LOGIC;
        resetPC                                             : IN     STD_LOGIC;
        inicio                                              : IN     STD_LOGIC;
        output_vec                                          :                                OUT
        STD_LOGIC_VECTOR(6 DOWNTO 0)
    );
END COMPONENT;

COMPONENT po_atan_vhdl
    PORT
    (
        po_clk                                  : IN     STD_LOGIC;
        input_vec                               : IN     STD_LOGIC_VECTOR(6 DOWNTO 0);
        entraX,entraY                           : IN    STD_LOGIC_VECTOR(3 DOWNTO 0);
        saida                                   : OUT   STD_LOGIC_VECTOR(8 DOWNTO 0);
        lero                                    : OUT   STD_LOGIC_VECTOR(10 DOWNTO 0)
    );
END COMPONENT;

BEGIN

pc : pc_atan_vhdl
    PORT MAP
    (
        resetPC         =>      reset,
        output_vec      =>      vetor,
        inicio          =>      iniciar,
        clk             =>      clk
    );
```

```
po : po_atan_vhdl
    PORT MAP
    (
        input_vec       =>      vetor,
        entraX          =>      x,
        entraY          =>      y,
        saida           =>      z,
        po_clk          =>      clk,
        lero            =>      q
    );
END a;

LIBRARY ieee;
    USE ieee.std_logic_1164.all;
LIBRARY lpm;
USE lpm.lpm_components.all;
--------------------------------------------------------------------
ENTITY po_atan_vhdl IS
    PORT
    (
        input_vec                               : IN    STD_LOGIC_VECTOR(6 DOWNTO 0);
        entraX,entraY                           : IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
        po_clk                                  : IN    STD_LOGIC;
        saida                                   : OUT STD_LOGIC_VECTOR(8 DOWNTO 0);
        lero                                    : OUT STD_LOGIC_VECTOR(10 DOWNTO 0)
    );
END po_atan_vhdl;

ARCHITECTURE b OF po_atan_vhdl IS

    signal Xr, Yr                               :  STD_LOGIC_VECTOR(3 DOWNTO 0);
    signal Zr                                   :  STD_LOGIC_VECTOR(8 DOWNTO 0);
    SIGNAL Ar,Br,Cr                             :  STD_LOGIC;
    SIGNAL MX1,MX2,LD1,LD2,LD3,LD4,LD5          :  STD_LOGIC;

    signal      COMP_A_OUT,COMP_B_OUT,COMP_C_OUT    :   STD_LOGIC; -- entradas do
    subtrator / somador

    signal  enderecoMEM                         :       STD_LOGIC_VECTOR(10 DOWNTO 0);
    signal  mem_out                             :       STD_LOGIC_VECTOR(8 DOWNTO 0);
BEGIN

    MX1 <= input_vec(0);
    MX2 <= input_vec(1);
    LD1 <= input_vec(2);
    LD2 <= input_vec(3);
    LD3 <= input_vec(4);
    LD4 <= input_vec(5);
    LD5 <= input_vec(6);

do_XrYr:
PROCESS (LD1)
    BEGIN
    IF LD1'event and LD1='1' THEN
        Xr <= entraX;
        Yr <= entraY;
    END IF;
END PROCESS do_XrYr;

do_Zr:
```

```
PROCESS (LD5)
    BEGIN
    IF LD5'event and LD5='1' THEN
          Zr        <=        mem_out;
    END IF;
END PROCESS do_Zr;

COMP_A_OUT <= '1' WHEN  entraX  > "0000"      ELSE '0';

COMP_B_OUT <= '1' WHEN  entraY  > "0000"      ELSE '0';

COMP_C_OUT <= '1' WHEN  entraX  > entraY      ELSE '0';

Ar <= COMP_A_OUT WHEN LD2 = '1';

Br <= COMP_B_OUT WHEN LD3 = '1';

Cr <= COMP_C_OUT WHEN LD4 = '1';

enderecoMEM(0)          <= Yr(0);
enderecoMEM(1)          <= Yr(1);
enderecoMEM(2)          <= Yr(2);
enderecoMEM(3)          <= Yr(3);
enderecoMEM(4)          <= Xr(0);
enderecoMEM(5)          <= Xr(1);
enderecoMEM(6)          <= Xr(2);
enderecoMEM(7)          <= Xr(3);
enderecoMEM(8)          <= Ar;
enderecoMEM(9)          <= Br;
enderecoMEM(10) <= Cr;

memoria : lpm_rom
    GENERIC MAP
    (         LPM_WIDTH           => 9,
        LPM_WIDTHAD         => 11,
        LPM_ADDRESS_CONTROL   => "REGISTERED",
        LPM_OUTDATA         => "REGISTERED",
        LPM_FILE            => "mem.mif",
        LPM_TYPE            => "LPM_ROM",
        LPM_HINT            => "UNUSED"
    )
    PORT MAP
    (
          address           =>enderecoMEM,
          inclock           =>po_clk,
          outclock=>po_clk,
          q                           =>mem_out
    );

    lero  <= enderecoMEM;

    saida <= Zr;

END b;

LIBRARY ieee;
    USE ieee.std_logic_1164.all;
--------------------------------------------------------------------------
ENTITY pc_atan_vhdl IS
    PORT
```

```
    (
        clk                                     : IN    STD_LOGIC;
        resetPC                                 : IN    STD_LOGIC;
        inicio                                  : IN    STD_LOGIC;
        output_vec                              : OUT   STD_LOGIC_VECTOR(6 DOWNTO 0)
    );
END pc_atan_vhdl;

ARCHITECTURE c OF pc_atan_vhdl IS
    TYPE STATE_TYPE IS (st_0, st_A, st_B, st_C, st_D, st_E, st_F);
    SIGNAL state: STATE_TYPE;
BEGIN
    PROCESS (clk, resetPC,inicio)
    BEGIN
        IF resetPC = '1' THEN
                state <= st_0;
        ELSIF clk'EVENT AND clk = '1' THEN
                CASE state IS
                        WHEN st_0 =>
                                if inicio ='1' then
                                        state <= st_A;
                                end if;
                        WHEN st_A =>
                                output_vec(0)   <=      '0';
                                output_vec(1)   <=      '0';
                                output_vec(2)   <=      '1';
                                output_vec(3)   <=      '0';
                                output_vec(4)   <=      '0';
                                output_vec(5)   <=      '0';
                                output_vec(6)   <=      '0';

                                state <= st_B;

                        WHEN st_B =>
                                output_vec(0)   <=      '0';
                                output_vec(1)   <=      '0';
                                output_vec(2)   <=      '0';
                                output_vec(3)   <=      '0';
                                output_vec(4)   <=      '0';
                                output_vec(5)   <=      '1';
                                output_vec(6)   <=      '0';

                                state <= st_C;

                        WHEN st_C =>
                                output_vec(0)   <=      '1';
                                output_vec(1)   <=      '0';
                                output_vec(2)   <=      '0';
                                output_vec(3)   <=      '0';
                                output_vec(4)   <=      '1';
                                output_vec(5)   <=      '0';
                                output_vec(6)   <=      '0';

                                state <= st_D;

                        WHEN st_D =>
                                output_vec(0)   <=      '0';
                                output_vec(1)   <=      '1';
                                output_vec(2)   <=      '0';
                                output_vec(3)   <=      '1';
```

```
                                    output_vec(4)   <=      '0';
                                    output_vec(5)   <=      '0';
                                    output_vec(6)   <=      '0';

                                    state <= st_E;

                                WHEN st_E =>
                                    state <= st_F;

                                WHEN st_F =>
                                    output_vec(0)   <=      '0';
                                    output_vec(1)   <=      '0';
                                    output_vec(2)   <=      '0';
                                    output_vec(3)   <=      '0';
                                    output_vec(4)   <=      '0';
                                    output_vec(5)   <=      '0';
                                    output_vec(6)   <=      '1';

                                    state <= st_0;
                        END CASE;
                END IF;
        END PROCESS;
END c;
```

- CORDIC:
    - Código em C:

```
#include <stdio.h>
#include <math.h>
#define AG_CONST        0.6072529350
#define  FIXED(X)       ((long int)((X) * 65536.0))
#define  FLOAT(X)       ((X) / 65536.0)
#define  DEG2RAD(X)     0.017453 * (X)

typedef long int   fixed; /* 16.16 fixed-point */

static const fixed Angles[]={
        FIXED(45.0),    FIXED(26.565),  FIXED(14.0362),
        FIXED(7.12502),         FIXED(3.57633),         FIXED(1.78991),
         FIXED(0.895174),FIXED(0.447614),FIXED(0.223811),
        FIXED(0.111906),FIXED(0.055953),FIXED(0.027977),
         FIXED(0.013988), FIXED(0.006994) };
int main()
{       fixed X, Y, TargetAngle,aux;
        //unsigned Step;
        int Step;
        int x = 2;
        int y = 7;
    int v=0;
    printf("digite x:");
    scanf("%d",&x);
    while(x > 255) {
          printf("Extrapolou...");
          printf("digite x:");
          scanf("%d",&x);
    }

        printf("digite y:");
        scanf("%d",&y);
```

```
	while (y>255){
		printf("Extraplou...");
		printf("digite y:");
		scanf("%d",&y);
	}
	    X=FIXED(x);	/* argc1 2 AG_CONST * cos(0) */
	    Y=FIXED(y);	/* argc2 7 AG_CONST * sin(0) */
	    TargetAngle=FIXED(-90);
	aux = X ;
	X = Y;
	Y = - aux;

	if(y<0){v=1; X = -X;}
	for(Step=0; Step < 13; Step++)
	    {	fixed NewX;
		printf("  %d  X:  %7.5f  Y:  %7.5f,  %7.5f  %7.5f",  Step,FLOAT(X),  FLOAT(Y),
FLOAT(Angles[Step]), FLOAT(TargetAngle));
	 if  (Y < 0)
			{	NewX=X - (Y >> Step);
				Y=(X >> Step) + Y;
				X=NewX;
				printf(" Valor do Angulo %f "), FLOAT(Angles[Step]);
				TargetAngle = TargetAngle +  (Angles[Step]);
				printf(" soma: %f \n",FLOAT(TargetAngle));
			}
			else
			{	NewX=X + (Y >> Step);
				Y=-(X >> Step) + Y;
				X=NewX;
				printf(" Valor do Angulo %f "), FLOAT(Angles[Step]);
				 TargetAngle =TargetAngle -  Angles[Step];
				printf(" diferenca: %f\n", FLOAT(TargetAngle));
			}
	}
	if(v==1) {TargetAngle =-(TargetAngle + FIXED(360)); }
	printf("\nCORDIC atan(%d/%d) = %7.5f\n",y,x, -FLOAT(TargetAngle) );
	    return(0);
}
```

- VHDL:
  - Actg.vhd (interface entre pc e po):

```
	use ieee.std_logic_1164.all;
	use ieee.std_logic_arith.all;
ENTITY arctg IS
  PORT
  (
	x, y                                    : IN    std_logic_vector(4 DOWNTO 0);
	clk                                     : IN    STD_LOGIC;
	inicio                                  : IN    STD_LOGIC;
	reset                                   : IN    STD_LOGIC;
	z                                       : OUT   std_logic_vector(15 DOWNTO 0);
	statusSAIDA                             : OUT   std_logic_vector(3 downto 0);
	controleSAIDA                           : OUT   STD_LOGIC_VECTOR(12 DOWNTO 0)
  );
END arctg;

ARCHITECTURE a OF arctg IS
SIGNAL controlePOPC : STD_LOGIC_VECTOR(12 DOWNTO 0);
SIGNAL statusPOPC : STD_LOGIC_VECTOR(3 DOWNTO 0);
```

```
COMPONENT atan_CORDIC
  PORT
  (
        clk, start                              : IN    STD_LOGIC;
        reset                                   : IN    STD_LOGIC;
        controle                                : OUT   STD_LOGIC_VECTOR(12 DOWNTO 0);
        status                                  : IN    STD_LOGIC_VECTOR(3 DOWNTO 0)
  );
END COMPONENT;

COMPONENT po
  PORT
  (
        x, y                                    : IN    std_logic_vector (4 DOWNTO 0);
        controle                                : IN    STD_LOGIC_VECTOR(12 DOWNTO 0);
        clk                                     : IN    STD_LOGIC;
        status                                  : OUT   STD_LOGIC_VECTOR(3 DOWNTO 0);
        z                                       : OUT   std_logic_vector(15 DOWNTO 0)
  );
END COMPONENT;
begin
pc : atan_CORDIC
        PORT MAP(
                        clk => clk,
                        start => inicio,
                        reset => reset,
                        controle => controlePOPC,
                        status => statusPOPC
        );

op : po
        PORT MAP(
                        x => x,
                        y => y,
                        controle => controlePOPC,
                        clk => clk,
                        status => statusPOPC,
                        z =>z
        );

controleSAIDA   <= controlePOPC;
statusSAIDA     <= statusPOPC;
end a;
```

- Atan_CORDIC.vhd (PC):

```
library ieee;
        use ieee.std_logic_1164.all;
------------------------------------------------------
ENTITY atan_CORDIC IS
PORT
(
        clk, start                              : IN    STD_LOGIC;
        reset                                   : IN    STD_LOGIC;
        controle                                : OUT   STD_LOGIC_VECTOR(12 DOWNTO 0);
        status                                  : IN    STD_LOGIC_VECTOR(3 DOWNTO 0)
);
END atan_CORDIC;

ARCHITECTURE fsm OF atan_CORDIC IS
        TYPE STATE_TYPE IS (st_0, st_A, st_B, st_C, st_D,st_9, st_E, st_F, st_G, st_H, st_I, st_J,
st_L);
```

```
        SIGNAL state                                    : STATE_TYPE;
        SIGNAL ld1, ld2, ld3, ld4, ld5, ld6             : std_logic;
        SIGNAL mx1, mx2, mx3, mx4                       : std_logic;
        SIGNAL add_SUB1, add_SUB2, add                  : std_logic;
        SIGNAL comp1, comp2, comp3, comp4               : std_logic;
        SIGNAL aux_r                                    : std_logic;
BEGIN
        comp1   <=      status(0);
        comp2   <=      status(1);
        comp3   <=      status(2);
        aux_r   <=      status(3);

        PROCESS (clk, reset)
        BEGIN
                IF reset = '1' THEN
                        state <= st_0;
                ELSIF clk'EVENT AND clk = '1' THEN
                        CASE state IS
                                WHEN st_0 =>
                                        ld1 <= '0';
                                        ld2 <= '0';
                                        ld3 <= '0';
                                        ld4 <= '0';
                                        ld5 <= '0';
                                        ld6 <= '0';
                                        mx1 <= '0';
                                        mx2 <= '0';
                                        mx3 <= '0';
                                        mx4     <= '0';
                                        add_SUB1 <= '0';
                                        add_SUB2 <= '0';
                                        add <= '0';
                                        IF start = '1' THEN
                                                state <= st_A;
                                        END IF;

                                WHEN st_A =>
                                        ld1 <= '0';
                                        ld2 <= '0';
                                        ld3 <= '1';
                                        ld4 <= '1';
                                        ld5 <= '0';
                                        ld6 <= '0';
                                        mx1 <= '1';
                                        mx2 <= '0';
                                        mx3 <= '0';
                                        mx4     <= '0';
                                        add_SUB1 <= '0';
                                        add_SUB2 <= '0';
                                        add <= '0';
                                        state <= st_B;

                                WHEN st_B =>
                                        if (comp3='1') then
                                                mx4 <= '1';
                                        else
                                                mx4 <= '0';
                                        end if;
                                        ld1 <= '1';
                                        ld2 <= '1';
```

```
		ld3 <= '0';
		ld4 <= '0';
		ld5 <= '0';
		ld6 <= '1';
		mx1 <= '1';
		mx2 <= '0';
		mx3 <= '0';
		add_SUB1 <= '0';
		add_SUB2 <= '0';
		add <= '0';
		state <= st_C;

	WHEN st_C =>
		IF comp1 = '1' THEN
			state <= st_G;
		ELSE
			state <= st_D;
		END IF;
		ld1 <= '0';
		ld2 <= '0';
		ld3 <= '0';
		ld4 <= '0';
		ld5 <= '0';
		ld6 <= '0';
		mx1 <= '0';
		mx2 <= '0';
		mx3 <= '1';
		mx4 <= '0';
		add_SUB1 <= '0';
		add_SUB2 <= '0';
		add <= '0';

	WHEN st_D =>
		ld1 <= '0';
		ld2 <= '0';
		ld3 <= '0';
		ld4 <= '0';
		ld5 <= '1';
		ld6 <= '0';
		mx1 <= '0';
		mx2 <= '0';
		mx3 <= '1';
		mx4 <= '0';
		add_SUB1 <= '0';
		add_SUB2 <= '0';
		add <= '0';
		state <= st_E;

	WHEN st_E =>
		IF comp1 = '1' THEN
			state <= st_H;
		ELSE
			state <= st_F;
		END IF;
		ld1 <= '0';
		ld2 <= '0';
		ld3 <= '0';
		ld4 <= '0';
		ld5 <= '0';
		ld6 <= '0';
```

```
            mx1 <= '0';
            mx2 <= '1';
            mx3 <= '0';
            mx4 <= '0';
            add_SUB1 <= '0';
            add_SUB2 <= '0';
            add <= '0';

        WHEN st_F =>
            ld1 <= '0';
            ld2 <= '1';
            ld3 <= '0';
            ld4 <= '0';
            ld5 <= '0';
            ld6 <= '0';
            mx1 <= '0';
            mx2 <= '0';
            mx3 <= '0';
            mx4 <= '0';
            add_SUB1 <= '1';
            add_SUB2 <= '1';
            add <= '0';
            state <= st_I;

        WHEN st_G =>
            ld1 <= '0';
            ld2 <= '0';
            ld3 <= '0';
            ld4 <= '0';
            ld5 <= '1';
            ld6 <= '0';
            mx1 <= '0';
            mx2 <= '0';
            mx3 <= '0';
            mx4 <= '0';
            add_SUB1 <= '1';
            add_SUB2 <= '0';
            add <= '0';
            state <= st_E;

        WHEN st_H =>
            ld1 <= '0';
            ld2 <= '1';
            ld3 <= '0';
            ld4 <= '0';
            ld5 <= '0';
            ld6 <= '0';
            mx1 <= '0';
            mx2 <= '0';
            mx3 <= '0';
            mx4     <= '0';
            add_SUB1 <= '0';
            add_SUB2 <= '0';
            add <= '0';
            state <= st_I;


        WHEN st_I =>
            ld1 <= '1';
            ld2 <= '0';
```

```
                ld3 <= '1';
                ld4 <= '0';
                ld5 <= '0';
                ld6 <= '0';
                mx1 <= '0';
                mx2 <= '0';
                mx3 <= '0';
                mx4 <= '1';
                add_SUB1 <= '0';
                add_SUB2 <= '0';
                add <= '1';
                state <= st_J;

            WHEN st_J =>
                IF comp2 = '1' THEN
                        state <= st_C;
                ELSE
                        state <= st_L;
                END IF;
                ld1 <= '0';
                ld2 <= '0';
                ld3 <= '0';
                ld4 <= '1';
                ld5 <= '0';
                ld6 <= '0';
                mx1 <= '0';
                mx2 <= '0';
                mx3 <= '0';
                mx4 <= '0';
                add_SUB1 <= '0';
                add_SUB2 <= '0';
                add <= '0';

            WHEN st_L =>
                if aux_r = '0' then
                        ld3 <= '1';
                else ld3 <= '0';
                end if;
                        ld1 <= '0';
                        ld2 <= '0';
                        ld4 <= '0';
                        ld5 <= '0';
                        ld6 <= '0';
                        mx1 <= '0';
                        mx2 <= '0';
                        mx3 <= '0';
                        mx4 <= '0';
                        add_SUB1 <= '0';
                        add_SUB2 <= '0';
                        add <= '0';
                        state <= st_0;
            END CASE;
        END IF;
END PROCESS;
controle(0) <= ld1;
controle(1) <= ld2;
controle(2) <= ld3;
controle(3) <= ld4;
controle(4) <= ld5;
controle(5) <= ld6;
```

```
	controle(6) <= mx1;
	controle(7) <= mx2;
	controle(8) <= mx3;
	controle(9) <= mx4;
	controle(10) <= add_SUB1;
	controle(11) <= add_SUB2;
	controle(12) <=   add;
END fsm;
```

- Po.vhd (PO)

```
------------------------------------------------------------
library ieee;
	use ieee.std_logic_1164.all;
	use ieee.std_logic_unsigned.all;
library lpm;
	use lpm.lpm_components.all;
------------------------------------------------------------

ENTITY po IS
PORT
(
	x, y                          : IN    STD_LOGIC_VECTOR (4 DOWNTO 0);
	controle                      : IN    STD_LOGIC_VECTOR(12 DOWNTO 0);
	clk                           : IN    STD_LOGIC;
	status                        : OUT   STD_LOGIC_VECTOR(3 DOWNTO 0);
	z                             : OUT   STD_LOGIC_VECTOR (15 DOWNTO 0)
);
END po;

ARCHITECTURE arch OF po IS

	CONSTANT zr_in : std_logic_vector := "0100000000000000";

---------------------- Sinais de Controle --------------------
	SIGNAL ld1, ld2, ld3, ld4, ld5,ld6        : std_logic;
	SIGNAL mx1, mx2, mx3, mx4                 : std_logic;
	SIGNAL add_SUB1, add_SUB2, add            : std_logic;
	SIGNAL comp1, comp2,comp3, aux_r          : std_logic;
-------------------------------------------------------------------
	SIGNAL xr_data, xrn_data, yr_data         : STD_LOGIC_VECTOR(12 DOWNTO 0);
	SIGNAL zr_data                            : STD_LOGIC_VECTOR(15 DOWNTO 0);
	SIGNAL ir_data                            : STD_LOGIC_VECTOR(3 DOWNTO 0);
	SIGNAL aux_data                           : std_logic;
-------------------------------------------------------------------
	SIGNAL as1_1in, as1_2in          : STD_LOGIC_VECTOR(12 DOWNTO 0);
	SIGNAL as1_out                   : STD_LOGIC_VECTOR(12 DOWNTO 0);
	SIGNAL as2_1in, as2_2in          : STD_LOGIC_VECTOR(15 DOWNTO 0);
	SIGNAL as2_out                   : STD_LOGIC_VECTOR(15 DOWNTO 0);
	SIGNAL add_out                   : STD_LOGIC_VECTOR(3 DOWNTO 0);
-------------------------------------------------------------------
	SIGNAL lut_out                   : STD_LOGIC_VECTOR(15 DOWNTO 0);
	SIGNAL inv_z_out                 : STD_LOGIC_VECTOR(15 DOWNTO 0);
	SIGNAL bs_in, bs_out             : STD_LOGIC_VECTOR(12 DOWNTO 0);
	SIGNAL bs_y_out, bs_x_out        : STD_LOGIC_VECTOR(12 DOWNTO 0);
	SIGNAL inv_y_out, inv_x_out      : STD_LOGIC_VECTOR(12 DOWNTO 0);
-------------------------------------------------------------------



BEGIN
```

```
z <= zr_data;
ld1 <= controle(0);
ld2 <= controle(1);
ld3 <= controle(2);
ld4 <= controle(3);
ld5 <= controle(4);
ld6 <= controle(5);
mx1 <= controle(6);
mx2 <= controle(7);
mx3 <= controle(8);
mx4 <= controle(9);
add_SUB1 <= controle(10);
add_SUB2 <= controle(11);
add <= controle(12);

status(0) <= comp1;
status(1) <= comp2;
status(2) <= comp3;
status(3) <= aux_r;

        --registrador Xr
xr:     process(ld1,ld5,mx1)
        begin
                if (ld1'event and ld1='1') then
                        if(mx1 = '0') then
                                xr_data <= xrn_data;
                        elsif mx4='0' then
                                xr_data <= inv_y_out;
                        else
                                xr_data <= bs_y_out;
                        end if;
                end if;
        end process xr;


        --registrador Yr
yr:     process(ld2,mx1)
        begin
                if (ld2'event and ld2='1') then
                        if(mx1 = '1')       then -- entrada depende da saída do mux1
                                yr_data  <=         inv_x_out;
                        else
                                yr_data  <=         as1_out;
                        end if;
                end if;
        end process yr;



        --comparador 1
comparando: lpm_compare
        GENERIC MAP
                (               LPM_WIDTH         => 13,
                    LPM_REPRESENTATION  => "SIGNED",
                    LPM_TYPE            => "LPM_COMPARE",
                    LPM_HINT            => "UNUSED"
                 )
        PORT MAP
                (                       dataa => yr_data,
```

```
                                datab => "0000000000000",
                                agb => comp1
                );


        --comparador 3
comparando3: lpm_compare
        GENERIC MAP
                (LPM_WIDTH          => 5,
                LPM_REPRESENTATION  => "SIGNED",
                LPM_TYPE            => "LPM_COMPARE",
                LPM_HINT            => "UNUSED"
                )
                  PORT MAP
                                (dataa => y,
                                datab => "00000",
                                agb => comp3
                );

doAux_R:
                PROCESS (ld6)
                BEGIN
                        IF ld6'event and ld6='1' THEN
                                aux_r <= comp3;
                        END IF;
                END PROCESS doAux_R;

        --inversor de y
        inv_y_out       <=      (not bs_y_out) + 1;

        --inversor de x
        inv_x_out       <=      (not bs_x_out) + 1;

        --bitshift em x
        bs_x_out        <= x & "00000000";

        --bitshift em y
        bs_y_out        <= y & "00000000";

---------------------- Atualizacao do X-Y ----------------------------
        --entrada do bitshift
        bs_in   <=      xr_data when mx2='1' else yr_data;

        --Bitshift em Xr ou Yr
shiftingY: lpm_clshift
  GENERIC MAP
    (LPM_WIDTH     => 13,
    LPM_WIDTHDIST  => 4,
    LPM_SHIFTTYPE  => "ARITHMETIC",
    LPM_TYPE       => "LPM_CLSHIFT",
    LPM_HINT       => "UNUSED"
   )
        PORT MAP (
                        data => bs_in,
                        distance => ir_data,
                        direction => '1',
                        result => bs_out
                        );

doXrn:
```

```
        PROCESS (ld5)
        BEGIN
                IF ld5'event and ld5='1' THEN
                        xrn_data <= as1_out;
                END IF;
        END PROCESS doXrn;

----------------------SOMADOR 1-----------------------------------

        --somador subtrator 1
        as1_1in <=      xr_data WHEN MX3 ='1' ELSE  yr_data;
        as1_2in <=      bs_out;

somando1: lpm_add_sub
        GENERIC MAP
   (LPM_WIDTH           => 13,
    LPM_DIRECTION       => "UNUSED",
    LPM_REPRESENTATION  =>  "SIGNED",
    LPM_TYPE            => "LPM_ADD_SUB",
    LPM_HINT            => "UNUSED"
        )

  PORT MAP
        (       dataa => as1_1in,
                datab => as1_2in,
                add_sub => add_SUB1,
      result  => as1_out
        );

---------- Atualizacao do Z ----------

        --registrador Zr
zr:     process(ld3,mx1,mx4)
        begin
                if (ld3'event and ld3='1') then
                        if (mx1 = '1')then -- entrada depende da saída do mux5
                                zr_data <= zr_in;
                        elsif (mx4='1')then -- entrada depende da saída do mux8
                                        zr_data <= as2_out;
                                else
                                        zr_data <= inv_z_out;       --correção se y<0
                        end if;
                end if;
        end process zr;

        --registrador Ir
ir:     process(ld4,mx1)
        begin
                if (ld4'event and ld4='1') then
                        if(mx1='1') then
                                ir_data   <=      "0000";
                        ELSE
                                ir_data <= add_out; -- entrada depende da saída do mux7
                        end if;
                end if;
        end process ir;
----------------------SOMADOR 2-----------------------------------
        --somador subtrator 2
        as2_1in <=      zr_data;
        as2_2in <=      lut_out;
```

```
somando2: lpm_add_sub
        GENERIC MAP
   ( LPM_WIDTH                  =>16,
     LPM_DIRECTION              => "UNUSED",
     LPM_REPRESENTATION         =>"SIGNED",
     LPM_TYPE                   =>"LPM_ADD_SUB",
     LPM_HINT                   => "UNUSED"
        )
        PORT MAP
        (               dataa => as2_1in,
                        datab => as2_2in,
                        add_sub => add_SUB2,
                        result  => as2_out
        );

        --somador de iteração
        add_out <=      (ir_data + 1) WHEN add='1' ELSE add_out;

        --comparador 2
comparando2: lpm_compare
        GENERIC MAP
                (   LPM_WIDTH               =>4,
                    LPM_REPRESENTATION      =>"UNSIGNED",
                    LPM_TYPE                =>"LPM_COMPARE",
                    LPM_HINT                =>"UNUSED"
                )
        PORT MAP
                (               dataa => ir_data,
                                datab => "1100",
                                aleb => comp2
                );

        --LUT
mem: lpm_rom
        GENERIC MAP
        (       LPM_WIDTH               =>16,
                LPM_WIDTHAD             => 4,
                LPM_ADDRESS_CONTROL     => "UNREGISTERED",
                LPM_OUTDATA             =>"UNREGISTERED",
                LPM_FILE                =>"mem.mif",
                LPM_TYPE                =>"LPM_ROM",
                LPM_HINT                => "UNUSED"
        )
        PORT MAP (
                        address  => ir_data,
                        q        => lut_out
                );

        --inversor de Zr
        inv_z_out       <=      (not zr_data) + 1;

END arch;
```