Sundarapandian system2

Sistema dinamico

$$\begin{aligned}
\dot{z_1} &= a(z_2 - z_1) + z_2 z_3 + z_4 \\
\dot{z_2} &= z_1(b - z_3) + c z_4 \\
\dot{z_3} &= z_1^2 + z_1 z_2 - d z_3 \\
\dot{z_4} &= -z_2 + z_5 \\
\dot{z_5} &= -z_5
\end{aligned}$$

Forward Euler ecuación para el sistema

El método de forward Euler es el siguiente:

$$y_{n+1} = y_n + h f(x_n, y_n)$$

El sistema resultante es:

$$\begin{aligned}
z_{[}]_{n+1} &= w_n + h[a(x_n - w_n) + x_n y_n + d z_n] \\
x_{n+1} &= x_n + h[-w_n + b x_n - w_n y_n + d z_n] \\
y_{n+1} &= y_n + h[x_n^2 - c y_n] \\
z_{n+1} &= z_n + h[-x_n]
\end{aligned}$$

donde:

$$\begin{aligned}
a &= 40 \\
b &= 90 \\
c &= 16 \\
d &= 15
\end{aligned}$$

y las condiciones iniciales son:

$$\begin{aligned}
w_0 &= 0.1 \\
x_0 &= 0.1 \\
y_0 &= 0.1 \\
z_0 &= 0.1
\end{aligned}$$

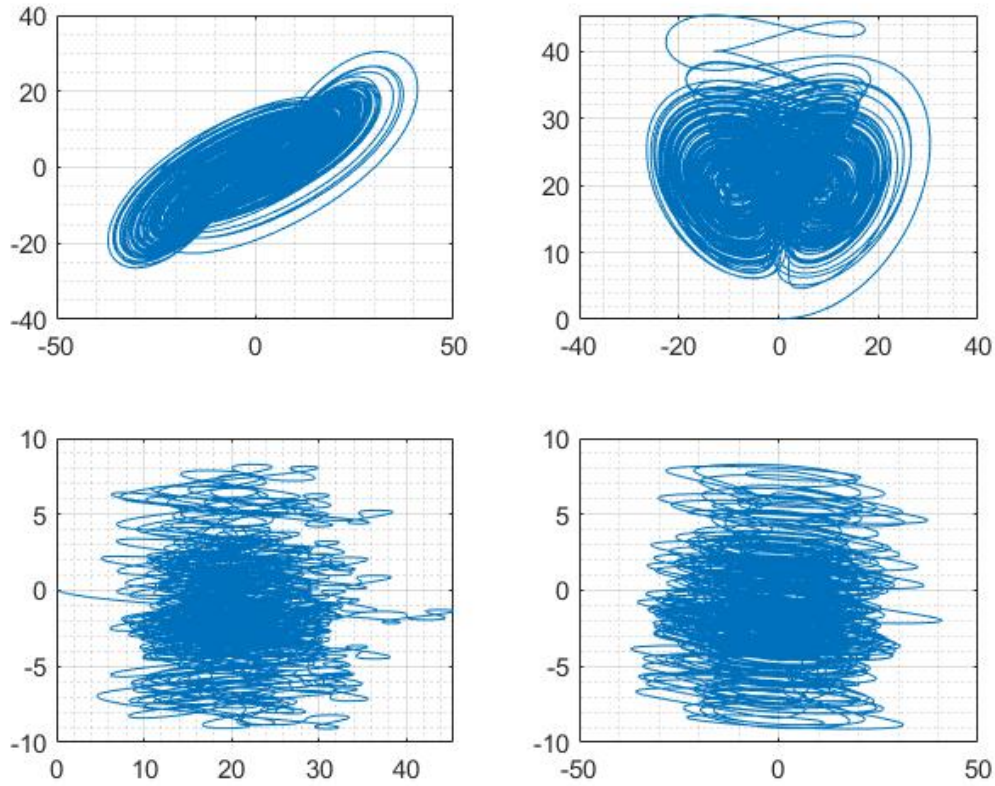
Simulación de MATLAB con forward Euler

1. Simulación básica de sistema

```
clear; close all; clc;
tic
h = 0.001;      % Tamanio de paso
t = 0:h:100;    % Vector de tiempo

% Parametros
```

```
a = 40;
b = 28;
c = 4;
d = 7;

y1 = zeros(size(t));   % Inicializacion de los vectores
y2 = zeros(size(t));
y3 = zeros(size(t));
y4 = zeros(size(t));

% Asignacion de condicion inicial
ini_cond = [0.1 0.1 0.1 0.1]';    % Condiciones iniciales
y1(1) = ini_cond(1);
y2(1) = ini_cond(2);
y3(1) = ini_cond(3);
y4(1) = ini_cond(3);

% Algoritmo forward euler
for i = 2:size(y1,2)
    y1(i) = y1(i-1) + y1_state(y1(i-1),y2(i-1),y3(i-1),y4(i-1),a,b,c,d)*h;
    y2(i) = y2(i-1) + y2_state(y1(i-1),y2(i-1),y3(i-1),y4(i-1),a,b,c,d)*h;
    y3(i) = y3(i-1) + y3_state(y1(i-1),y2(i-1),y3(i-1),y4(i-1),a,b,c,d)*h;
    y4(i) = y4(i-1) + y4_state(y1(i-1),y2(i-1),y3(i-1),y4(i-1),a,b,c,d)*h;
end

% f = figure; f.Position(1:2) = [800 800]; % [right bottom]
subplot(2,2,1); plot(y1,y2); grid on; grid minor;
subplot(2,2,2); plot(y2,y3); grid on; grid minor;
subplot(2,2,3); plot(y3,y4); grid on; grid minor;
subplot(2,2,4); plot(y1,y4); grid on; grid minor;

check_max = max( [max(y1) max(y2) max(y3) max(y4)] )
check_min = min( [min(y1) min(y2) min(y3) min(y4)] )
toc

% Descripcion de sistema dinamico
function R = y1_state(y1,y2,y3,y4,a,b,c,d)
    R = a*(y2-y1) + y2*y3 + d*y4;
end

function R = y2_state(y1,y2,y3,y4,a,b,c,d)
    R = -y1 + b*y2 - y1*y3 + d*y4;
end

function R = y3_state(y1,y2,y3,y4,a,b,c,d)
    R = y2*y2 - c*y3;
end

function R = y4_state(y1,y2,y3,y4,a,b,c,d)
    R = -y2;
end
```

De la simulación podemos notar que los límites absolutos son [-37.0094 , 45.4266], y un buen tamaño de paso es h = 0.001.



Desafortunadamente analizar unicamente las variables de estado para seleccionar el número de bits para la parte entera no es suficiente, es necesario realizar un análisis más profundo.

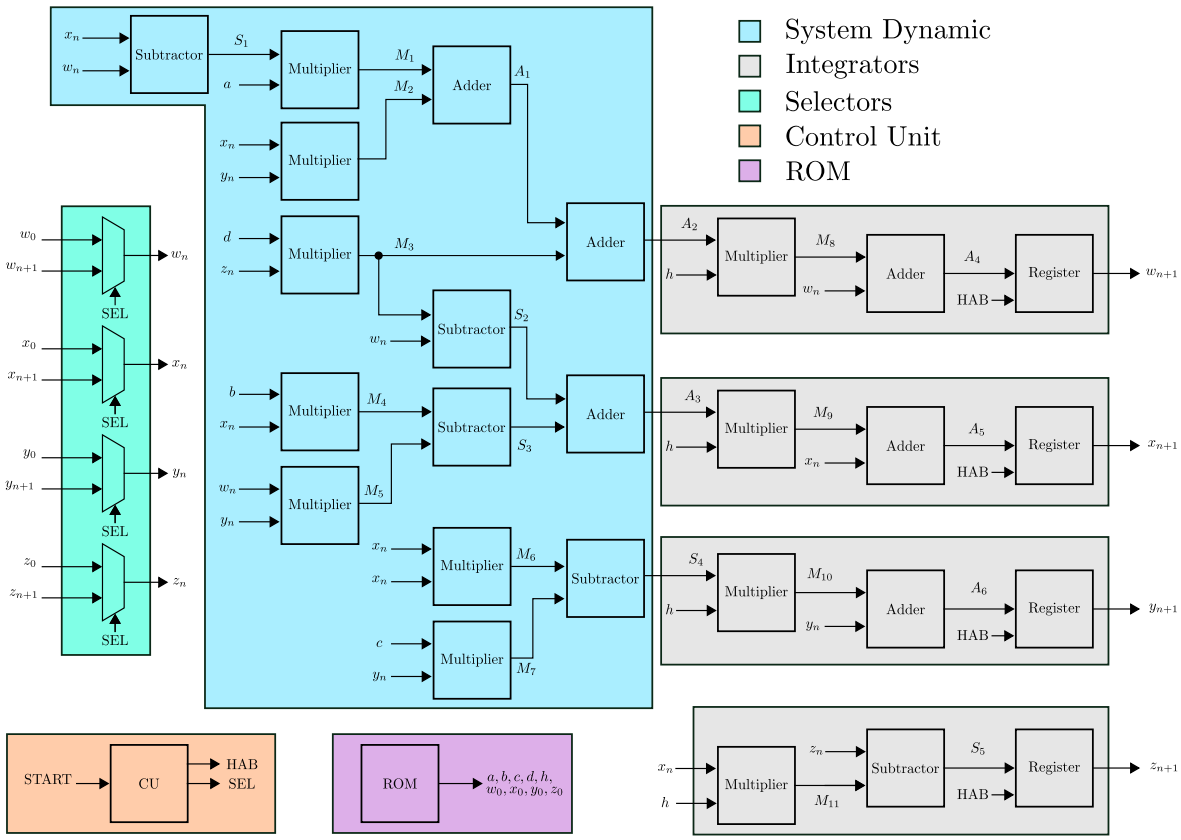
2. Simulación para determinar punto fijo

```
clear; close all; clc;
tic
h = 0.001;      % Tamanio de paso

% Parametros
a = 40; b = 28; c = 4; d = 7;

% Vectores
y1 = []; y2 = []; y3 = []; y4 = []; data = [];

% Asignacion de condicion inicial
y1(1) = 0.1; y2(1) = 0.1; y3(1) = 0.1; y4(1) = 0.1;

% Algoritmo forward euler
for i = 1:100001
    op11 = y2(i)-y1(i);
    op12 = a*op11;
    op13 = y2(i)*y3(i);
    op14 = op12 + op13;
    op15 = d*y4(i);
    op16 = op14 + op15;
    op17 = op16*h;
    y1(i+1) = y1(i) + op17;

    op21 = d*y4(i);
    op22 = op21 - y1(i);
```

```
    op23 = b*y2(i);
    op24 = y1(i)*y3(i);
    op25 = op23 - op24;
    op26 = op25 + op22;
    op27 = op26*h;
    y2(i+1) = y2(i) + op27;

    op31 = y2(i)*y2(i);
    op32 = c*y3(i);
    op33 = op31 - op32;
    op34 = op33*h;
    y3(i+1) = y3(i) + op34;

    op41 = y2(i)*h;
    y4(i+1) = y4(i) - op41;

    data(i,:) =
[y1(i+1),y2(i+1),y3(i+1),y4(i+1),op11,op12,op13,op14,op15,op16,op17,op21,op22,op
23,op24,op25,op26,op27, op31,op32,op33,op34, op41];
end

f = figure; f.Position(1:2) = [800 800]; % [right bottom]
subplot(2,2,1); plot(y1,y2); grid on; grid minor;
subplot(2,2,2); plot(y2,y3); grid on; grid minor;
subplot(2,2,3); plot(y3,y4); grid on; grid minor;
subplot(2,2,4); plot(y1,y4); grid on; grid minor;
check_max = max(data,[],'all')
check_min = min(data,[],'all')
toc
```

Calculamos el máximo y el mínimo en cada una de las operaciones, [-1546.4540,1583.5721], con esta información ya podemos seleccionar el número de bits de la parte entera, $2^{11} = 2048$, despues realizaremos una simulación en C para comprobar la arquitectura

Diagrama a bloques del sistema



Fixed point analysis

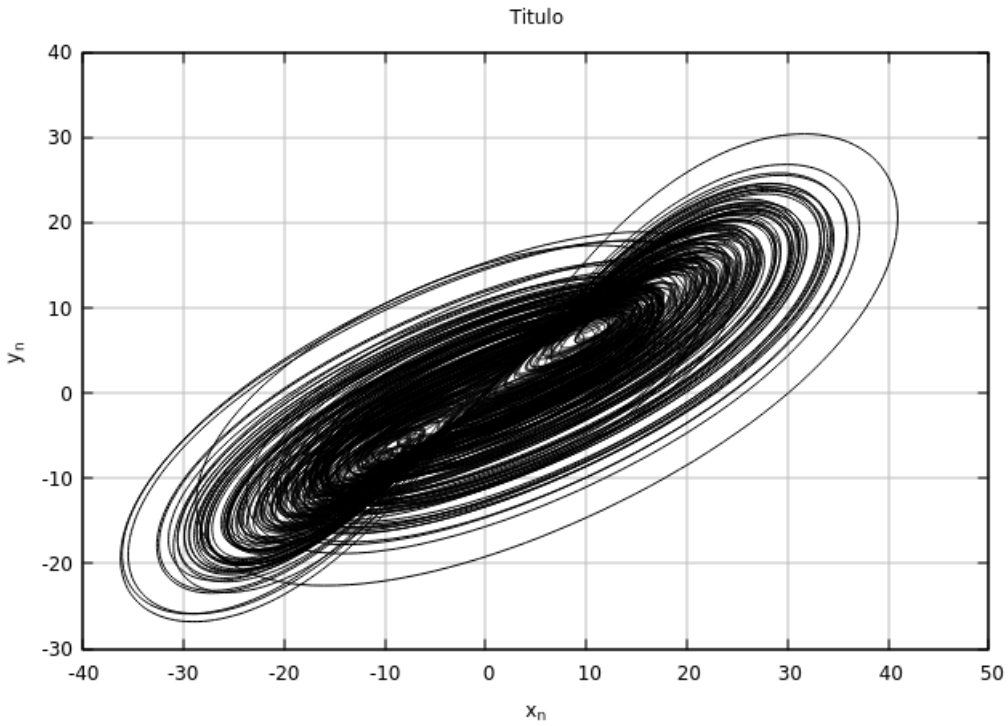
Variable	Number of bits	Format	Move point	Range $[-2^a, 2^a - 2^{-b}]$
X	64 bits	$X(11, 52)$	52	$[-2048, 2048]$

Simulación en C

```c
/*
    Autor:       Ciro Fabian Bermudez Marquez
    Descripción: Simulador de diseños en VHDL de 64 bits en punto fijo
*/
/* Librerias*/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>

/* Variables globales */
int _a;         // parte entera
int _b;         // parte fraccionaria
long _power;    // factor de conversion

/*Funciones*/

// Inicializacion A(a,b) representacion en punto fijo
void inicializa( int a, int b ){
    _a = a;                 // _a: parte entera
    _b = b;                 // _b: parte fraccionaria
    _power = (long)1 << _b; // calculo de factor de conversion
```

```
}

// Convierte a punto fijo con truncamiento
long setNumber( double v ){
    return ( (long)(v*_power) );
}

// Convierte de vuelta a punto flotante
double getNumber( long r ){
    return ( (double)r/_power);
}

// Multiplicacion de punto fijo con truncamiento
long multTrunc( long x, long y ){
    __int128 r;
    __int128 a=0;
    __int128 b=0;
    a = x;
    b = y;
    r = a*b;
    r = r >> _b;
    return( r );
}

int main(int argc, char *argv[]){
    FILE *fpointer = fopen("salida.txt","w");       // Archivo de texto

    // Arq: 64 bits entera + frac + 1 = 64
    int entera = 11;
    int frac;
    frac = 64 - 1 - entera;

    // Variables y parametros de simulacion
    // Condiciones iniciales
    double y1_0 = 0.1;
    double y2_0 = 0.1;
    double y3_0 = 0.1;
    double y4_0 = 0.1;

    // Para metros
    double a = 40.0;
    double b = 28.0;
    double c = 4.0;
    double d = 7.0;
    double h = 0.001;

    // Variables para algoritmo en punto fijo
    long y1_n,y2_n,y3_n,y4_n;           // Actual
    long y1_ni,y2_ni,y3_ni,y4_ni;       // Siguiente
    long apf, bpf, cpf, dpf, hpf;

    // Inicializacion de arq
    inicializa( entera, frac);
    printf(" Representacion A(a,b) = A(%d,%d)\n a: entera\tb:
fraccionaria\n",entera,frac);
    printf(" Rango: [%30.20f,%30.20f] = \n", -pow(2.0,entera),pow(2.0,entera)-
pow(2.0,-frac));
```

```c
    // Conversion a punto fijo

    y1_n = setNumber( y1_0 );
    y2_n = setNumber( y2_0 );
    y3_n = setNumber( y3_0 );
    y4_n = setNumber( y4_0 );
    apf = setNumber( a );
    bpf = setNumber( b );
    cpf = setNumber( c );
    dpf = setNumber( d );
    hpf = setNumber( h );
    printf(" # y1_0:      %12.8f\n # y1_0 real: %12.8f\n", y1_0, getNumber( y1_n
) );
    printf(" # y2_0:      %12.8f\n # y2_0 real: %12.8f\n", y2_0, getNumber( y2_n
) );
    printf(" # y2_0:      %12.8f\n # y3_0 real: %12.8f\n", y3_0, getNumber( y3_n
) );
    printf(" # y4_0:      %12.8f\n # y4_0 real: %12.8f\n", y4_0, getNumber( y4_n
) );
    printf(" # a:      %12.8f\n # a real: %12.8f\n", a, getNumber( apf ) );
    printf(" # b:      %12.8f\n # b real: %12.8f\n", b, getNumber( bpf ) );
    printf(" # c:      %12.8f\n # c real: %12.8f\n", c, getNumber( cpf ) );
    printf(" # d:      %12.8f\n # d real: %12.8f\n", d, getNumber( dpf ) );



    fprintf(fpointer,"%20.15f\t%20.15f\n",getNumber( y1_n ), getNumber( y2_n ));
    for(int i = 0; i<1000; i++){

        y1_ni = y1_n  + multTrunc(hpf,multTrunc( apf, y2_n - y1_n ) +
multTrunc(y2_n, y3_n) + multTrunc( dpf, y4_n));
        y2_ni = y2_n  + multTrunc(hpf, - y1_n + multTrunc( bpf, y2_n ) -
multTrunc( y1_n ,y3_n ) + multTrunc( dpf, y4_n));
        y3_ni = y3_n  + multTrunc(hpf, multTrunc( y2_n , y2_n ) - multTrunc(
cpf, y3_n));
        y4_ni = y4_n  + multTrunc(hpf, - y2_n);

        y1_n = y1_ni;
        y2_n = y2_ni;
        y3_n = y3_ni;
        y4_n = y4_ni;

        fprintf(fpointer,"%20.15f\t%20.15f\n",getNumber( y1_n ), getNumber( y2_n
));
    }

    fclose(fpointer);                                // Cerrar archivo de texto
    return 0;
}
// gcc -o simulation simulation.c
// ./simulation
// gnuplot -e "filename='salida.txt'" graph.gnu
```



Simulación en C HEX

```c
/*
    Autor:       Ciro Fabian Bermudez Marquez
    Descripción: Simulador de diseños en VHDL de 64 bits en punto fijo
*/
/* Librerias*/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>

/* Variables globales */
int _a;         // parte entera
int _b;         // parte fraccionaria
long _power;    // factor de conversion

/*Funciones*/

// Inicializacion A(a,b) representacion en punto fijo
void inicializa( int a, int b ){
    _a = a;                 // _a: parte entera
    _b = b;                 // _b: parte fraccionaria
    _power = (long)1 << _b; // calculo de factor de conversion
}

// Convierte a punto fijo con truncamiento
long setNumber( double v ){
    return ( (long)(v*_power) );
}

// Convierte de vuelta a punto flotante
double getNumber( long r ){
```

```
    return ( (double)r/_power);
}

// Multiplicacion de punto fijo con truncamiento
long multTrunc( long x, long y ){
    __int128 r;
    __int128 a=0;
    __int128 b=0;
    a = x;
    b = y;
    r = a*b;
    r = r >> _b;
    return( r );
}

int main(int argc, char *argv[]){
    FILE *fpointer = fopen("salida.txt","w");       // Archivo de texto

    // Arq: 64 bits entera + frac + 1 = 64
    int entera = 11;
    int frac;
    frac = 64 - 1 - entera;

    // Variables y parametros de simulacion
    // Condiciones iniciales
    double y1_0 = 0.1;
    double y2_0 = 0.1;
    double y3_0 = 0.1;
    double y4_0 = 0.1;

    // Para metros
    double a = 40.0;
    double b = 28.0;
    double c = 4.0;
    double d = 7.0;
    double h = 0.001;

    // Variables para algoritmo en punto fijo
    long y1_n,y2_n,y3_n,y4_n;           // Actual
    long y1_ni,y2_ni,y3_ni,y4_ni;       // Siguiente
    long apf, bpf, cpf, dpf, hpf;

    // Inicializacion de arq
    inicializa( entera, frac);
    printf(" Representacion A(a,b) = A(%d,%d)\n a: entera\tb:
fraccionaria\n",entera,frac);
    printf(" Rango: [%30.20f,%30.20f] = \n", -pow(2.0,entera),pow(2.0,entera)-
pow(2.0,-frac));

    // Conversion a punto fijo

    y1_n = setNumber( y1_0 );
    y2_n = setNumber( y2_0 );
    y3_n = setNumber( y3_0 );
    y4_n = setNumber( y4_0 );
    apf = setNumber( a );
    bpf = setNumber( b );
    cpf = setNumber( c );
```

```
    dpf = setNumber( d );
    hpf = setNumber( h );
    printf(" # y1_0:      %12.8f\n # y1_0 real: %12.8f\n", y1_0, getNumber( y1_n
) );
    printf(" # y2_0:      %12.8f\n # y2_0 real: %12.8f\n", y2_0, getNumber( y2_n
) );
    printf(" # y2_0:      %12.8f\n # y3_0 real: %12.8f\n", y3_0, getNumber( y3_n
) );
    printf(" # y4_0:      %12.8f\n # y4_0 real: %12.8f\n", y4_0, getNumber( y4_n
) );
    printf(" # a:      %12.8f\n # a real: %12.8f\n", a, getNumber( apf ) );
    printf(" # b:      %12.8f\n # b real: %12.8f\n", b, getNumber( bpf ) );
    printf(" # c:      %12.8f\n # c real: %12.8f\n", c, getNumber( cpf ) );
    printf(" # d:      %12.8f\n # d real: %12.8f\n", d, getNumber( dpf ) );

    printf("%.16lx\n%.16lx\n%.16lx\n%.16lx\n\n", y1_n,y2_n,y3_n,y4_n );
    printf("%.16lx\n%.16lx\n%.16lx\n%.16lx\n%.16lx\n", apf,bpf,cpf,dpf,hpf );
    fprintf(fpointer,"%.16lx\t%.16lx\n", y1_n , y2_n );
    for(int i = 0; i<100; i++){

        y1_ni = y1_n  + multTrunc(hpf,multTrunc( apf, y2_n - y1_n ) +
multTrunc(y2_n, y3_n) + multTrunc( dpf, y4_n));
        y2_ni = y2_n  + multTrunc(hpf, - y1_n + multTrunc( bpf, y2_n ) -
multTrunc( y1_n ,y3_n ) + multTrunc( dpf, y4_n));
        y3_ni = y3_n  + multTrunc(hpf, multTrunc( y2_n , y2_n ) - multTrunc(
cpf, y3_n));
        y4_ni = y4_n  + multTrunc(hpf, - y2_n);

        y1_n = y1_ni;
        y2_n = y2_ni;
        y3_n = y3_ni;
        y4_n = y4_ni;

        fprintf(fpointer,"%.16lx\t%.16lx\n", y1_n , y2_n );
    }

    fclose(fpointer);                               // Cerrar archivo de texto
    return 0;
}
```

Salida

```
y1=0001999999999999
y2=0001999999999999
y3=0001999999999999
y4=0001999999999999

a=0280000000000000
b=01c0000000000000
c=0040000000000000
d=0070000000000000
h=000004189374bc6a
```

```
0001999999999999    0001999999999999
00019c8216c61521    0001a77c45cbbc29
00019fda907fd393    0001b5c0f6e0250f
0001a3a273397594    0001c46a45a4ee39
0001a7d94bc2671f    0001d37add89f9ec
0001ac7ec6b96dfb    0001e2f57d1c06ec
0001b192b009c7c3    0001f2dcf6835938
.                   .
.                   .
.                   .
```

VDHL codes

A continuación se muestran todos los bloques:

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

entity adder is
    generic( n : integer := 64 );
    port(
        T1,T2    : in   std_logic_vector(n-1 downto 0);
        S1       : out  std_logic_vector(n-1 downto 0)
    );
end;

architecture arch of adder is
begin
    S1 <= std_logic_vector( signed(T1) + signed(T2) );
end arch;
```

Código: adder.vhd

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

entity sub is
    generic( n : integer := 64 );
    port(
        T1,T2    : in   std_logic_vector(n-1 downto 0);
        S1       : out  std_logic_vector(n-1 downto 0)
    );
end;

architecture arch of sub is
begin
    S1 <= std_logic_vector( signed(T1) - signed(T2) );
```

```
end arch;
```

Código: sub.vhd

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

entity mult is
    generic( n  :   integer := 64);
    port(
        A   : in    std_logic_vector(n-1 downto 0);
        B   : in    std_logic_vector(n-1 downto 0);
        M   : out   std_logic_vector(n-1 downto 0)
    );
end;

architecture arch of mult is
signal temp : std_logic_vector(2*n-1 downto 0);
begin
    temp <= std_logic_vector(signed(A)*signed(B));
    M <= temp(115 downto 52);
end arch;

-- El formato es  A(a ,b ) = (11,52)
--                Ap(ap,bp) = (22,104)
-- lim_izq = bp + a = 104 + 11 = 115
-- lim_der = bp - 1 - (b-1) = 104 -1 - (52-1) = 52
```

Código: mult.vhd

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

entity ff_hab is
    generic(n : integer := 64);
    port(
        RST :   in  std_logic;
        CLK :   in  std_logic;
        HAB :   in  std_logic_vector(1 downto 0);
        D   :   in  std_logic_vector(n-1 downto 0);
        Q   :   out std_logic_vector(n-1 downto 0)
    );
end;

architecture ff of ff_hab is
signal Qn, Qp : std_logic_vector(n-1 downto 0);
begin
--  Qn <= Qp when HAB = '0' else D;
    with HAB select
```

```
    Qn <=                    (others => '0') when "00",   -- Reset
                                          D when "01",   -- Pasar
                                         Qp when others; -- Mantener
    process(RST, CLK)
    begin
        if RST = '1' then
            Qp <= (others => '0');
        elsif rising_edge(CLK) then
            Qp <= Qn;
        end if;
    end process;
    Q <= Qp;
end ff;
```

Código: ff_hab.vhd

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

entity rom is
    generic( n : integer := 64 );        -- tamaño de palabra
    port(
        a,b,c,d,h,w0,x0,y0,z0   : out   std_logic_vector(n-1 downto 0)
    );
end rom;

architecture arch of rom is
begin
     a <= "0000001010000000000000000000000000000000000000000000000000000000";
-- 40.000
     b <= "0000000111000000000000000000000000000000000000000000000000000000";
-- 28.000
     c <= "0000000001000000000000000000000000000000000000000000000000000000";
-- 4.000
     d <= "0000000001110000000000000000000000000000000000000000000000000000";
-- 7.000
     h <= "0000000000000000000001000001100010010011011101001011110001101010";
-- 0.001
     w0 <= "0000000000000001100110011001100110011001100110011001100110011001";
-- 0.100
     x0 <= "0000000000000001100110011001100110011001100110011001100110011001";
-- 0.100
     y0 <= "0000000000000001100110011001100110011001100110011001100110011001";
-- 0.100
     z0 <= "0000000000000001100110011001100110011001100110011001100110011001";
-- 0.100

end arch;
```

Código: rom.vhd

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

entity mux is
    generic( n : integer := 64);  -- Tamanio de palabra
    port(
        W0  : in std_logic_vector(n-1 downto 0);
        Wn_1: in std_logic_vector(n-1 downto 0);
        SEL : in std_logic;
        Wn  : out std_logic_vector(n-1 downto 0)
    );
end;

architecture arch of mux is
begin
    Wn <= W0 when SEL = '0' else Wn_1;
end arch;
```

Código: mux.vhd

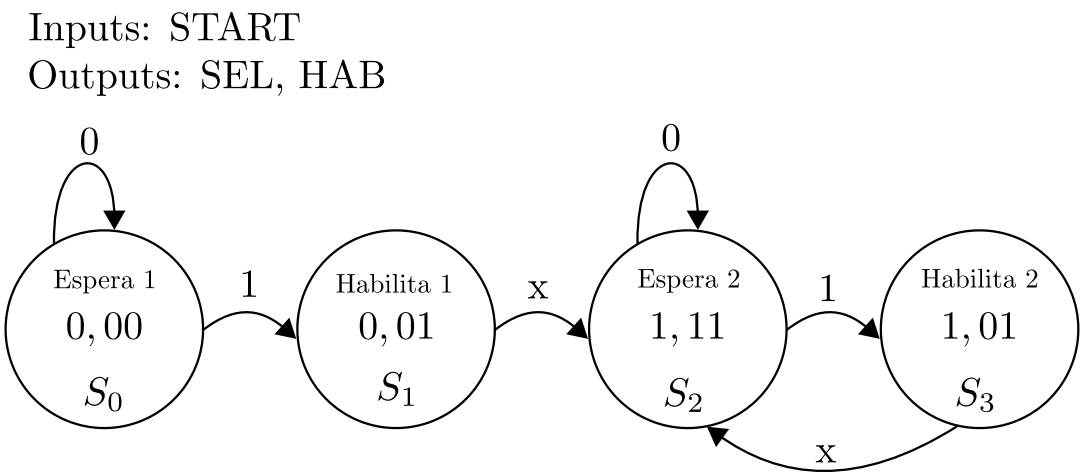
Diseño de maquina de estado

Los registros funcionan con la siguiente lógica:

HAB	Accion
`00`	Reset
`01`	Pasar dato
`10` o `11`	Mantener dato

Los mux funcionan con la siguiente lógica:

SEL	Salida
`0`	Condición inicial
`1`	Retroalimentación



La maquina de estado funciona teniendo en cuenta la señal de entrada `START` la cual el usuario controla con un switch de la FPGA o puede utilizarse como señal interna de comienzo para interconectarse con otros sistemas. Al encenderse el sistema, `SEL` esta en `0` y `HAB` en `00`, por tanto en la salida de los multiplaxores se encuentra la condición inicial y los registros inicializan con cero, casi instantaneamente se realizan todas las operaciones del sistema en paralelo pero no es hasta que `START` este en `1` que pasa al siguiente estado y se almacenan los resultados en los registros, `SEL` se mantiene en `0` para no alterar la operación actual. Despues de esa primera iteración se pasa al siguiente estado donde `SEL` es igual a `1` y `HAB` a `11`, los registros pasan a mantener su estado actual, y a la salida de los multiplexores ahora tenemos los valores de los registros, otra vez, de manera casi inmediata se realiza el cálculo y dependiendo si `START` es igual `1` o `0` se guada el resultado en los registros o mantiene su valor. De esta manera podemos calcular cada iteración dependiendo unicamente de la señal `START`,

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

entity cu is
  port(
    RST     : in   std_logic;
    CLK     : in   std_logic;
    START   : in   std_logic;
    HAB     : out  std_logic_vector(1 downto 0);
    SEL     : out  std_logic
  );
end;

architecture fsm of cu is
    signal Qp, Qn  : std_logic_vector(1 downto 0); -- porque son 6 estados
begin

  process(Qp,START)
  begin
    case Qp is
     when "00" => SEL <= '0'; HAB <= "00";           -- espera 1
        if START = '1' then
          Qn <= "01";
        else
          Qn <= Qp;
        end if;
     when "01" => SEL <= '0'; HAB <= "01";           -- habilita 1
          Qn <= "10";
     when "10" => SEL <= '1'; HAB <= "11";           -- espera 2
        if START = '1' then
          Qn <= "11";
        else
          Qn <= Qp;
        end if;
     when "11" => SEL <= '1'; HAB <= "01";            -- habilita 2
        Qn <= "10";
     when others => SEL <= '0';  HAB <= "11";         -- default
        Qn <= "00";
    end case;
  end process;

  -- Registros para estados
  process(RST,CLK)
```

```
  begin
    if RST = '1' then
      Qp <= (others => '0');
    elsif rising_edge(CLK) then
      Qp <= Qn;
    end if;
  end process;

end fsm;
```

Código: cu.vhd

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

entity sundar_system is
    generic( n : integer := 64 );  -- Tamanio de palabra
    port(
        CLK     : in  std_logic;
        RST     : in  std_logic;
        START   : in std_logic;
        --w0      : in  std_logic_vector(n-1 downto 0);
        --X0      : in  std_logic_vector(n-1 downto 0);
        --Y0      : in  std_logic_vector(n-1 downto 0);
        --Z0      : in  std_logic_vector(n-1 downto 0);
        Wn_out    : out  std_logic_vector(n-1 downto 0);
        Xn_out    : out  std_logic_vector(n-1 downto 0);
        Yn_out    : out  std_logic_vector(n-1 downto 0);
        Zn_out    : out  std_logic_vector(n-1 downto 0)
    );
end;

architecture arch of sundar_system is
    signal sel : std_logic;
    signal hab : std_logic_vector(1 downto 0);
    signal wn_retro, xn_retro, yn_retro, zn_retro : std_logic_vector(n-1 downto
0);
    signal wn, xn, yn, zn : std_logic_vector(n-1 downto 0);
    signal s1,s2,s3,s4,s5 : std_logic_vector(n-1 downto 0);
    signal a1,a2,a3,a4,a5,a6 : std_logic_vector(n-1 downto 0);
    signal m1,m2,m3,m4,m5,m6,m7,m8,m9,m10,m11 : std_logic_vector(n-1 downto 0);
    signal a_rom, b_rom, c_rom, d_rom, h_rom : std_logic_vector(n-1 downto 0);
    signal w0_rom, x0_rom, y0_rom, z0_rom : std_logic_vector(n-1 downto 0);
begin

    -- Control Unit
    cu_mod  : entity work.cu  port map(RST,CLK,START,hab,sel);


    -- Mux
    mux_w   : entity work.mux    generic map(n => 64) port
map(w0_rom,wn_retro,sel,wn);        -- listo
    mux_x   : entity work.mux    generic map(n => 64) port
map(x0_rom,xn_retro,sel,xn);        -- listo
```

```
    mux_y   : entity work.mux    generic map(n => 64) port
map(y0_rom,yn_retro,sel,yn);        -- listo
    mux_z   : entity work.mux    generic map(n => 64) port
map(z0_rom,zn_retro,sel,zn);        -- listo


    -- Rom de parametros
    param   : entity work.rom   generic map(n => 64) port
map(a_rom,b_rom,c_rom,d_rom,h_rom,w0_rom,x0_rom,y0_rom,z0_rom);

    -- Bloque wn
    sub_1   : entity work.sub    generic map(n => 64) port map(xn,wn,s1);
  -- Listo
    mult_1  : entity work.mult   generic map(n => 64) port map(s1,a_rom,m1);
 -- Listo
    mult_2  : entity work.mult   generic map(n => 64) port map(xn,yn,m2);
  -- Listo
    add_1   : entity work.adder  generic map(n => 64) port map(m1,m2,a1);
  -- Listo
    mult_3  : entity work.mult   generic map(n => 64) port map(d_rom,zn,m3);
 -- Listo
        -- Integrador wn
    add_2   : entity work.adder  generic map(n => 64) port map(a1,m3,a2);
  -- Listo
    mult_8  : entity work.mult   generic map(n => 64) port map(a2,h_rom,m8);
 -- Listo
    add_4   : entity work.adder  generic map(n => 64) port map(m8,wn,a4);
  -- Listo
    reg_1   : entity work.ff_hab generic map(n => 64) port
map(RST,CLK,hab,a4,wn_retro);      -- Listo


    -- Bloque xn
    sub_2   : entity work.sub    generic map(n => 64) port map(m3,wn,s2);
   -- Listo
    mult_4  : entity work.mult   generic map(n => 64) port map(b_rom,xn,m4);
  -- Listo
    mult_5  : entity work.mult   generic map(n => 64) port map(wn,yn,m5);
   -- Listo
    sub_3   : entity work.sub    generic map(n => 64) port map(m4,m5,s3);
   -- Listo
        -- Integrador xn
    add_3   : entity work.adder  generic map(n => 64) port map(s2,s3,a3);
-- Listo
    mult_9  : entity work.mult   generic map(n => 64) port map(a3,h_rom,m9);
   -- Listo
    add_5   : entity work.adder  generic map(n => 64) port map(m9,xn,a5);
-- Listo
    reg_2   : entity work.ff_hab generic map(n => 64) port
map(RST,CLK,hab,a5,xn_retro);      -- Listo


    -- Bloque yn
    mult_6  : entity work.mult   generic map(n => 64) port map(xn,xn,m6);
     -- Listo
    mult_7  : entity work.mult   generic map(n => 64) port map(c_rom,yn,m7);
    -- Listo
        -- Integrador yn
```

```
    sub_4   : entity work.sub    generic map(n => 64) port map(m6,m7,s4);
      -- Listo
    mult_10 : entity work.mult   generic map(n => 64) port map(s4,h_rom,m10);
      -- Listo
    add_6   : entity work.adder  generic map(n => 64) port map(m10,yn,a6);
 -- Listo
    reg_3   : entity work.ff_hab generic map(n => 64) port
map(RST,CLK,HAB,a6,yn_retro);    -- Listo


    -- Bloque yn
    mult_11 : entity work.mult   generic map(n => 64) port map(xn,h_rom,m11);
      -- Listo
    sub_5   : entity work.sub    generic map(n => 64) port map(zn,m11,s5);
     -- Listo
    reg_4   : entity work.ff_hab generic map(n => 64) port
map(RST,CLK,HAB,s5,zn_retro);       -- Listo

    Wn_out <= wn_retro;
    Xn_out <= xn_retro;
    Yn_out <= yn_retro;
    Zn_out <= zn_retro;

end arch;
```

Código: sundar_system.vhd

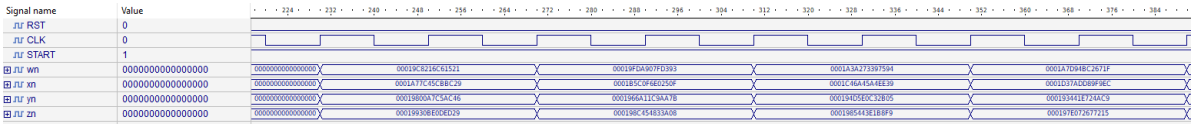
```
library ieee;
use ieee.std_logic_1164.all;

entity tb_sundar_system is
    generic(n : integer := 64);
end entity;

architecture tb of tb_sundar_system is
    signal      RST,CLK,START       :   std_logic := '0';
    signal      wn,xn,yn,zn         :   std_logic_vector(n-1 downto 0);
begin

    UUT : entity work.sundar_system generic map(n => 64) port
map(CLK,RST,START,wn,xn,yn,zn);
    RST <= '1', '0' after 20 ns;
    CLK <= not CLK after 10 ns;             -- La mitad del perido que es 20 ns


    START <= '1' after 200 ns, '0' after 2000 ns;

    -- NOTA: Simular por almenos 1000 ns
    -- wait for 10 ns;
end architecture;
```

Código: tb_sundar_system .vhd

Simulación VHDL

Los valores de las condiciones iniciales y de los parametros en hexadecimal son los siguientes

a_rom	0280000000000000
b_rom	01C0000000000000
c_rom	0040000000000000
d_rom	0070000000000000
h_rom	000004189374BC6A
w0_rom	0001999999999999
x0_rom	0001999999999999
y0_rom	0001999999999999
z0_rom	0001999999999999



Estos son las primeras iteraciones la simulación en C en hexadecimal, como podemos notar son exactamente los mismos resultados, por lo tanto la arquitectura esta lista para pasarse a la FPGA.

```
00019c8216c61521    0001a77c45cbbc29
00019fda907fd393    0001b5c0f6e0250f
0001a3a273397594    0001c46a45a4ee39
0001a7d94bc2671f    0001d37add89f9ec
```

Notas personales

- Para poder realizar verdaderamente un analisis de punto fijo del sistema es necesario analizar operación por operación para descubrir cuales son los límites máximos y mínimos, además de eso ejecutar el programa un tiempo razonable, no basta con analizar los límites de las variables de estado, por esta razón modifique el programa de análisis de punto fijo de matlab, además, noté que cambiar el orden de las operaciones modifica ligeramente el atractor, afortunadamente no su cualidad de ser caótico, por lo anterior también se modifican los límites máximos y mínimos ligeramente. Una vez determinados los límites se selecciona el mínimo número de bits para la parte entera, en este caso particular el límite es |1583| por lo tanto se seleccionaron 11 bits para la parte entera, la parte fraccionaria se elige seleccionando una arquitectura, de 16, 32 o 64 bits y despues restando la parte entera y el bit de signo a la arquitectura seleccionada. Esto es conveniente porque nos permite hacer una simulación en C antes de saltar a la descripción de VHDL y evitar perder tiempo buscando errores. Se pueden cometer errores de sintaxis, los cuales se solucionan facilmente pero no errores de diseño.
- En el código de **mult.vhd** estan las ecuaciones para no perder tiempo al extraer los bit en punto fijo,

Analisis de SCM (Single constant multiplication)

h = 0.01