Anexo D

Los bancos de prueba

Para suministrar los patrones de prueba a cada descripción del algoritmo CORDIC, se describieron bancos de prueba. A modo de ejemplo se transcribe un banco de prueba para la descripción funcional algorítmica y uno para cada descripción particular. Se describieron varios bancos de prueba que se utilizaron para efectuar las pruebas con el arcotangente y los ángulos particulares. Dichos bancos de prueba son similares a los que se presentan aquí y se incluyen junto con las descripciones en el CD-ROM adjunto.

El banco de pruebas que se muestra a continuación corresponde a la descripción funcional algorítmica del algoritmo CORDIC. El bucle While suministra al procedimiento `cordic` los ángulos comprendidos entre $-\frac{\pi}{2}$ y $\frac{\pi}{2}$ en modo rotación y la tangente de los ángulos en modo vectorización. Los resultados se almacenan en formato `REAL` provisto por el lenguaje VHDL que a su vez utiliza internamente la representación estándar IEEE 759 de 64 bits. Dichos valores son utilizados por MATLAB™ para realizar las comparaciones.

```
entity TestCordic is
end TestCordic;

architecture TestCordicArch of TestCordic is

      use work.Cordic.all;
      use std.textio.all;

begin

Prueba:
      process
      variable tita,K:real;
      variable sen,cos,atan:real;
      variable xo,yo,zo:real;
      variable linea:line;

      type fi_real is file of real;

      file salidaSen:fi_real open write_mode is "senFunc32_min.r";
      file salidaCos:fi_real open write_mode is "cosFunc32_min.r";
      file salidaATan:fi_real open write_mode is "atanFunc32_min.r";

      begin
```

```
            K:=1.0;
            for i in 0 to Iteraciones-1 loop
                  K:=K*(1.0/sqrt(1.0+2.0**(-2*i)));
            end loop;

            tita:=-math_pi/2.0;
            while tita<=math_pi/2.0 loop

                  wait for 1 ns;

                  cordic(K,0.0,tita,cos,sen,zo,Rotating);
                  cordic(1.0,tan(tita),0.0,xo,yo,atan,Vectoring);

                  write(salidaSen,sen);
                  write(salidaCos,cos);
                  write(salidaAtan,atan);

                  tita:=tita+(math_pi/10800.0);

            end loop;

            wait;

      end process;

end TestCordicArch;
```

Seguidamente se transcriben los bancos de pruebas utilizados para calcular las funciones seno y coseno, correspondiente a las descripciones bit-paralela iterativa y bit-paralela desplegada. Puede observarse la declaración del componente `ParalellIterativeCORDIC` y `ParalellUnrolledCORDIC` cuya descripción fue explicada en el capítulo 4. La descripción iterativa es secuencial, por lo tanto utiliza una señal de reloj como forma de sincronismo. Para la descripción desplegada la señal de reloj no fue necesaria. Sin embargo se incluyó retardos mediante el uso de la construcción `wait for 1 ns`. Dichos retardos deben incluirse luego de las asignaciones secuenciales a las señales, ya que éstas adquieren el valor en el próximo instante de simulación, como se explicó en las secciones 3.6.11.1 y 3.6.15.

Los patrones de prueba son generados utilizando el tipo de datos `REAL` provisto por VHDL y convertidos a la representación en punto fijo. Los resultados de la simulación se almacenan en archivos de texto. El formato con que se almacenan los valores es como cadena de caracteres compuestas por ‘1’ y ‘0’.

```
entity TestSinCosIterativeRange is
end TestSinCosIterativeRange;

architecture TestSinCosArch of TestSinCosIterativeRange is

      use work.typeconversion.all;
      use std.textio.all;

      constant ext:natural:=2;

      component ParalellIterativeCordic is

            generic(iterations,wide,ext:natural;arctanLUT:STDLVectorW);
            port(
                  clk   :in std_logic;
                  rst   :in std_logic;
```

```
                mode  :in std_logic;
                Xo    :in std_logic_vector(wide-1 downto 0);
                Yo    :in std_logic_vector(wide-1 downto 0);
                Zo    :in std_logic_vector(wide-1 downto 0);

                Xn    :out std_logic_vector(wide-1 downto 0);
                Yn    :out std_logic_vector(wide-1 downto 0);
                Zn    :out std_logic_vector(wide-1 downto 0);
                done  :out std_logic
          );

    end component;


    signal angleRot,sinRot,cosRot,zRot:std_logic_vector(wide-1 downto 0);
    signal ceroIn,K:std_logic_vector(wide-1 downto 0);

    signal clk:std_logic:='0';
    signal rstRot,doneRot:std_logic:='0';

begin

    clk<=not clk after 1 ns;

    ceroIn<=conv_real(0.0,wide,ext);

    ROT: ParalellIterativeCORDIC generic map (iterations,wide,ext,arctanLUT)
          port map(clk,rstRot,'1',K,ceroIn,angleRot
                      ,cosRot,sinRot,zRot,doneRot);
Prueba:
    process
    variable tita,Kr:real;
    variable linea:line;
    variable s:string(1 to wide);

    file salidaSen:text open write_mode is "sen_min.txt";
    file salidaCos:text open write_mode is "cos_min.txt";

    begin

          Kr:=1.0;
          for i in 0 to iterations-1 loop
                Kr:=Kr*(1.0/sqrt(1.0+2.0**(-2*i)));
          end loop;

          K<=conv_real(Kr,wide,ext);

          tita:=-math_pi/2.0;
          while tita<=math_pi/2.0 loop

                wait for 1 ns;
                angleRot<=conv_real(tita,wide,ext);
                rstRot<='1','0' after 2 ns;
                wait until doneRot='1';

                s:=convert(sinRot);
                write(linea,s);
                writeline(salidaSen,linea);
                s:=convert(cosRot);
                write(linea,s);
                writeline(salidaCos,linea);
```

```
                  tita:=tita+(math_pi/10800.0);

            end loop;

            wait;

      end process;

end TestSinCosArch;



entity TestSinCosUnrolledRange is
end TestSinCosUnrolledRange;

architecture TestSinCosArch of TestSinCosUnrolledRange is

      use work.TypeConversion.all;
      use work.ConfTypes.all;
      use std.textio.all;

      constant ext:natural:=2;

      component ParalellUnrolledCORDIC is

            generic(iterations,wide,ext:natural;arctanLUT:STDLVectorW);

            port(
                  mode  :in std_logic;
                  Xo    :in std_logic_vector(wide-1 downto 0);
                  Yo    :in std_logic_vector(wide-1 downto 0);
                  Zo    :in std_logic_vector(wide-1 downto 0);

                  Xn    :out std_logic_vector(wide-1 downto 0);
                  Yn    :out std_logic_vector(wide-1 downto 0);
                  Zn    :out std_logic_vector(wide-1 downto 0)
            );

      end component;


      signal angleRot,sinRot,cosRot,zRot:std_logic_vector(wide-1 downto 0);
      signal ceroIn,K:std_logic_vector(wide-1 downto 0);

begin

      ceroIn<=conv_real(0.0,wide,ext);


      ROT: ParalellUnrolledCORDIC generic map (iterations,wide,ext,arctanLUT)
                        port map('1',K,ceroIn,angleRot,cosRot,sinRot,zRot);
Prueba:
      process
      variable tita,Kr:real;
      variable linea:line;
      variable s:string(1 to wide);

      file salidaSen:text open write_mode is "sen_min.txt";
      file salidaCos:text open write_mode is "cos_min.txt";
```

```
      begin

            Kr:=1.0;
            for i in 0 to iterations-1 loop
                  Kr:=Kr*(1.0/sqrt(1.0+2.0**(-2*i)));
            end loop;

            K<=conv_real(Kr,wide,ext);

            tita:=-math_pi/2.0;
            while tita<=math_pi/2.0 loop
                  wait for 1 ns;

                  angleRot<=conv_real(tita,wide,ext);

                  wait for 1 ns;

                  s:=convert(sinRot);
                  write(linea,s);
                  writeline(salidaSen,linea);
                  s:=convert(cosRot);
                  write(linea,s);
                  writeline(salidaCos,linea);

                  tita:=tita+(math_pi/10800.0);

            end loop;

            wait;

      end process;

end TestSinCosArch;
```