Generador de Secuencia Binaria Pseudo Aleatoria

Nota Técnica 12

Cristian Sisterna

Introducción

La generación de una secuencia pseudo aleatoria de números binarios es muy útil en ciertas ambientes de test y desarrollo. Un generador de secuencia binaria pseudo aleatoria, SBSA, (en inglés, Pseudo Random Binary Sequence, PRBS) es un circuito que genera una serie de números binarios de n-bits, un número por ciclo de reloj, sin seguir un patrón determinado, pero que se repite luego de 2^n-1 ciclos de reloj. Por lo general un PRBS se implementa como un registro de desplazamiento de realimentación lineal (en ingles Linear Feedback Shift Register, LDFSR). Dicho de otra manera con el término PRBS se describe lo que el circuito hace, mientras que con el término LFSR se describe como el circuito está implementado.

En esta nota técnica se describe en VHDL un LFSR que genera la secuencia binaria pseudo-aleatoria. Se presentan en VHDL distintos ejemplos de código y su implementación en FPGA. Finalmente se detalla un ejemplo de código VHDL parametrizado a fin de tener un PRBS genérico.

Logic Design

Registro de Desplazamiento de Realimentación Lineal

El registro de desplazamiento con realimentación lineal (linear feedback shift register) es un tipo especial de registro de desplazamiento que utiliza un circuito de realimentación particular para generar el dato de entrada serie al registro de desplazamiento. El resultado de esta implementación es la generación de una secuencia pseudo aleatoria de n-bits de 2^n-1 posibles valores, donde n es el número de registros del registro de desplazamiento. El contenido de cada registro del registro de desplazamiento se desplaza hacia la derecha una posición en cada ciclo de reloj.

El diseño de LFSR se basa en la teoría de campos finitos, desarrollado por Evariste Galois (matemático francés, 1811-1832). Las operaciones de un LFSR se basan en las operaciones en un campo finito con 2^n componentes. Un LFSR de *n-bits* es una cadena de *n* registros con una única entrada, en uno de los registros de los extremos. Este dato de entrada, bit de entrada al registro de desplazamiento, es el resultado realizar la operación lógica XOR (o XNOR) entre ciertos y determinados bits del registro de desplazamiento. Obteniendo los valores lógicos a la salida de cada registro de desplazamiento y uniéndolos de modo de formar un vector o bus de

datos, se obtiene como resultado una secuencia de 2^n-1 números binarios de *n-bits* que cambia con cada flanco de reloj, cuyo ciclo se repite luego de 2^n-1 ciclos de reloj. Por ejemplo, en un LFSR de *n-bits* es posible utilizar una simple compuerta lógica XOR, con determinados bits del registro de desplazamiento como entradas, en el camino de realimentación y usando la salida de la XOR como entrada serie el registro de desplazamiento se genera una secuencia pseudo aleatoria de 2^n-1 números binarios, cuyo ciclo recomienza cada vez que se alcanza el 2^n-1 ciclo de reloj. Por lo tanto, utilizando la teoría del campo finito (que no voy a explicar aquí ☺), se puede demostrar que para cualquier valor de *n* (número de bits del registro de desplazamiento) existe al menos una ecuación de realimentación basada en la operación lógica XOR o XNOR que genera una secuencia pseudo-aleatoria de números binarios cuando de unen las *n* salidas de los n-bits del registro de desplazamiento, que se repite cada 2^n -1 ciclos de reloj (el único estado que no es visitado todos los ceros).

Figura 1 detalla la implementación de un LFSR de 4 bits, que genera el respectivo PRBS.

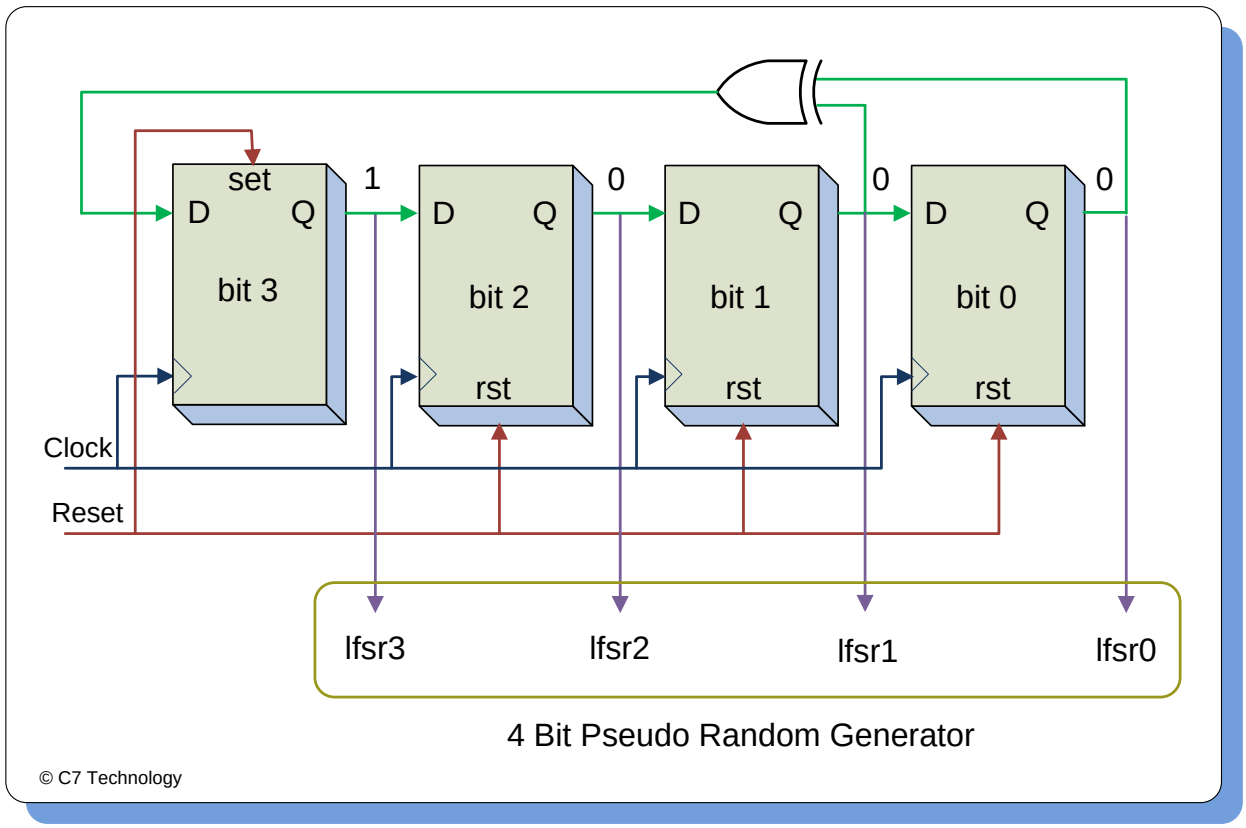

Figura 1 - LFSR de 4 bits

En primer lugar, los registros de desplazamiento se numeran desde la izquierda, lugar del bit más significativo (MSB) a la derecha (LSB)*. El circuito de realimentación está compuesto por una compuerta lógica XOR cuyas entradas son el bit 1 y el bit 0). La salida de la XOR es la entrada serie el LFSR, entrada al bit 3. Ahora bien, cómo trabaja este circuito?

Vamos a suponer que el estado inicial del registro de desplazamiento “1000". En el flanco de subida de sucesivos ciclos de reloj, las siguientes secuencias estarán disponibles en las salidas del registro de:

"0100", "0010", "1001", "1100", "0110", "1011", "0101", "1010 "," 1101 "," 1110 "," 1111 "," 0111 "," 0011 "," 0001 ", …. “1000”, “0100”, ...

De este modo, la salida circula a través de 2^4-1, 15, estados y vuelva a comenzar la misma secuencia nuevamente. La secuencia generada por el LFSR es pseudo-aleatorio, lo que significa que la secuencia muestra una propiedad estadística determinada y parece ser aleatorio.

Se debe tener en cuenta, que el estado "0000" no está incluido en la secuencia y es el único estado que falta de los 16 posibles. Si por alguna razón el LFSR toma el valor "0000", el circuito se quedará en ese estado, pues no tiene forma de salir del mismo.

El valor inicial de la secuencia se denomina comúnmente "semilla" (seed), y se define generalmente como un genérico en la entidad del código VHDL, o una constante en la arquitectura, para facilitar el cambio de la semilla según el valor inicial deseado. En realidad este valor de inicio puede ser cualquier valor excepto el cero, aunque existe un modo de agregar el cero a la secuencia, esto se explica más adelante.

*En esta nomenclatura voy a diferir, creo q por primera vez, con la usada por el bien recordado Peter Alfke en su XAPP052, en la cual usa la nomenclatura al revés. Estuve buscando información de cuál es la manera correcta para nombrar lo registro, y la que uso en este NT creo que es la correcta.

El siguiente código VHDL describe el comportamiento de un LFSR 4-bit.

```
------------------------------------------------------------------
   -- ejemplo 1 de 4-bits LFSR
   -- © C7T
------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;

------------------------------------------------------------------
   -- Enitiy --
------------------------------------------------------------------
entity lfsr_1 is
  generic(initial_value: std_logic_vector(3 downto 0):= "1000";
          lfsr_width   : integer := 4);

  port(
          clk      : in std_logic;
          rst      : in std_logic;
          q_lfsr_4b: out std_logic_vector(lfsr_width-1 downto 0)
      );
end lfsr_1;

------------------------------------------------------------------------------
   -- Architecture --
------------------------------------------------------------------------------
architecture beh of lfsr_1 is

signal q_lfsr_4b_i: std_logic_vector(lfsr_width-1 downto 0);
signal serial_in  : std_logic;

begin
-- Serial Input to the LFSR
serial_in <= q_lfsr_4b_i(1) xor q_lfsr_4b_i(0);

-- Shifter Process
lfsr_cnt_proc: process(rst, clk)
 begin
       if(rst= '1' ) then
              q_lfsr_4b_i <= initial_value;
       elsif (rising_edge(clk)) then
              q_lfsr_4b_i <= serial_in & q_lfsr_4b_i(q_lfsr_4b_i'high downto
                                                    q_lfsr_4b_i'low+1);
       end if;
end process lfsr_cnt_proc;

q_lfsr_4b <= q_lfsr_4b_i;

end architecture beh;

------------------------------------------------------------------------------
   -- EoF --
------------------------------------------------------------------------------
```

Los resultados de la simulación del código VHDL describiendo un LFSR de 4 bits (detallado arriba), pueden verse en Figura 2.

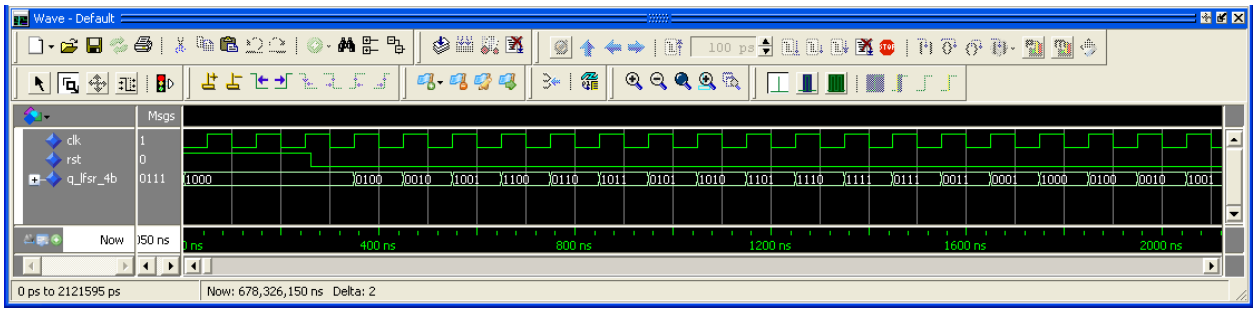

Figura 2 - Resultados de la Simulación de un LFSR de 4 bits

En las formas de ondas de la simulación se puede ver claramente la secuencia pseudo-aleatoria de la salida, llamada q_lfsr_4b, y como comienza el ciclo nuevamente una vez que pasan los 2^n-1 ciclos de reloj.

Otra manera de describir el LFSR de 4 bits, de un modo más compacto es usando un foor-loop. El siguiente código detalla el código VHDL respectivo.

```
-----------------------------------------------------------------------
   -- ejemplo 2 de 4-bits LFSR
   -- © C7T
-----------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;

-----------------------------------------------------------------------
   -- Enitiy --
-----------------------------------------------------------------------
entity lfsr_loop is
  generic(initial_value: std_logic_vector(3 downto 0):= "1000";
          lfsr_width   : integer := 4);

  port(
          clk      : in std_logic;
          rst      : in std_logic;
          q_lfsr_4b: out std_logic_vector(lfsr_width-1 downto 0)
      );
end lfsr_loop;

-----------------------------------------------------------------------
   -- Architecture --
-----------------------------------------------------------------------
architecture beh of lfsr_loop is
signal q_lfsr_4b_i: std_logic_vector(lfsr_width-1 downto 0);
begin
```

```
-- Shifter Process
lfsr_cnt_proc: process(rst, clk)
 begin
       if(rst= '1' ) then
              q_lfsr_4b_i <= initial_value;
       elsif (rising_edge(clk)) then
          -- shift operation: b3->b2, b2->b1, b1->b0
              shifter_loop: for i in 3 downto 1 loop
                     q_lfsr_4b_i(i-1) <= q_lfsr_4b_i(i);
              end loop shifter_loop;
          -- Serial Input to the b3 of the LFSR
      q_lfsr_4b_i(3) <= q_lfsr_4b_i(1) xor q_lfsr_4b_i(0);
       end if;
end process lfsr_cnt_proc;
q_lfsr_4b <= q_lfsr_4b_i;
end architecture beh;

-----------------------------------------------------------------------
   -- EoF --
-----------------------------------------------------------------------
```

Aplicaciones

Entre las aplicaciones más importantes de este circuito se pueden mencionar las siguientes:

- Datos circuitos de compresión, comúnmente cíclico de verificación de redundancia (CRC)
- Circuitos requieren números pseudo-aleatorios binarios
- Mezcle y descodificar los circuitos
- Codificación y Modulación

Implementación en FPGA

Una gran ventaja en la implementación de un LFSR es la reducida cantidad de recursos necesarios para la construcción del respectivo circuito. Por ejemplo, para implementar un generador pseudoaleatorio de 32-bits solo se necesitan 32 registros y 3 compuertas lógicas XOR. De este modo, se puede construir, con my bajos recursos, un LFSR de 32-bit con ciclos de 2^{32}-1 estados. Por otra parte, si se quisiera construir un contador binario de 32 bits la cantidad de recursos es mucho. La siguiente tabla muestra el resultado de la implementación en una FPGA, Cyclone IV, de un LFSR de 32-bit y un contador de bits 32.

Tabla 1 - Comparación entre 32-bits LFSR y Contador de 32 bits

Circuito	Niveles de Lógica	Recursos		Freq Max (MHz)
		Register Only	Comb. w/Reg	
32b-LFSR	1	29	3	545,85
32b-Binary Counter	31	0	32	304,41

De la tabla anterior, se puede observar que el uso de un LFSR tiene dos ventajas principales, una es la reducida cantidad de recursos utilizados, y la otra, que es muy importante para ciertos casos, la frecuencia máxima es mucho mayor que la frecuencia máxima de un contador binario (sólo un nivel de lógica por la compuerta XOR de 2 entradas). Por lo tanto, un LFSR puede sustituir eficazmente otros contadores en aplicaciones en las que el orden de los estados de recuento no es importante. *Utilizar un LFSR es una técnica de diseño inteligente y optimizada.*

Bits a Realimentar del Registro de Desplazamiento

El circuito de realimentación del LFSR depende del número de bits del LFSR, y los bits a tomar como entrada a la compuerta lógica XOR ya están determinados sobre una base ad hoc (de la teoría de campo finito). La expresión de la realimentación comúnmente es muy simple y puede implicar hasta cuatro operaciones lógicas XOR o XNOR.

La siguiente tabla detalla los bits del LFSR que forman parte de la realimentación. Siempre la salida de la XOR o XNOR está conectada a la entrada serie del LFSR.

Tabla 2 - Bits a realimentar en un LFSR

Tamaño del LFSR	Realimentación
2	q1 XOR q0
3	q1 XOR q0

4	q1 XOR q0
5	q2 XOR q0
6	q1 XOR q0
7	q3 XOR q0
8	q4 XOR q3 XOR q2 XOR q0
12	q6 XOR q4 XOR q1 XOR q0
16	q5 XOR q4 XOR q3 XOR q0
32	q22 XOR q2 XOR q1 XOR q0
64	q4 XOR q3 XOR q1 XOR q0
128	q29 XOR q17 XOR q2 XOR q0

Las expresiones detalladas en la tabla de arriba para cada LFSR proveen lo que se llama "máxima longitud del LFSR", por lo que utilizando la expresión correspondiente en el circuito de realimentación, generará la secuencia binaria pseudo aleatoria de 2^n-1 valores, cuyo ciclo dura 2^n-1 ciclos de reloj, y se repite indefinidamente. Se debe aclarar que por cada determinado número de registros puede haber otra expresión que origine una máxima longitud del LFSR, pero con mayor cantidad de bits en la expresión. También, puede haber otras expresiones que generen solo un número determinado de valores, mucho menos que los 2^n -1 posibles.

Cualquiera de los generadores LFSR generan una secuencia pseudo-aleatoria de unos y ceros, la secuencia no exactamente aleatoria, ya que se repite, y sigue una cierto comportamiento explicado en la teoría de los campos finitos. Sin embargo a los fines prácticos, la secuencia generada puede considerarse aleatoria. For ejemplo, un LFSR de 60 bits con un reloj corriendo a 50MHz, repite la secuencia después de 731 años!, tiempo suficientemente lago como para ser considerado aleatorio. En el caso de un LFSR de 63 bits, con un reloj de 50MHz, el ciclo se repite luego de 5849 años! . . .

El diagrama general de un LFSR se detalla en la siguiente figura.

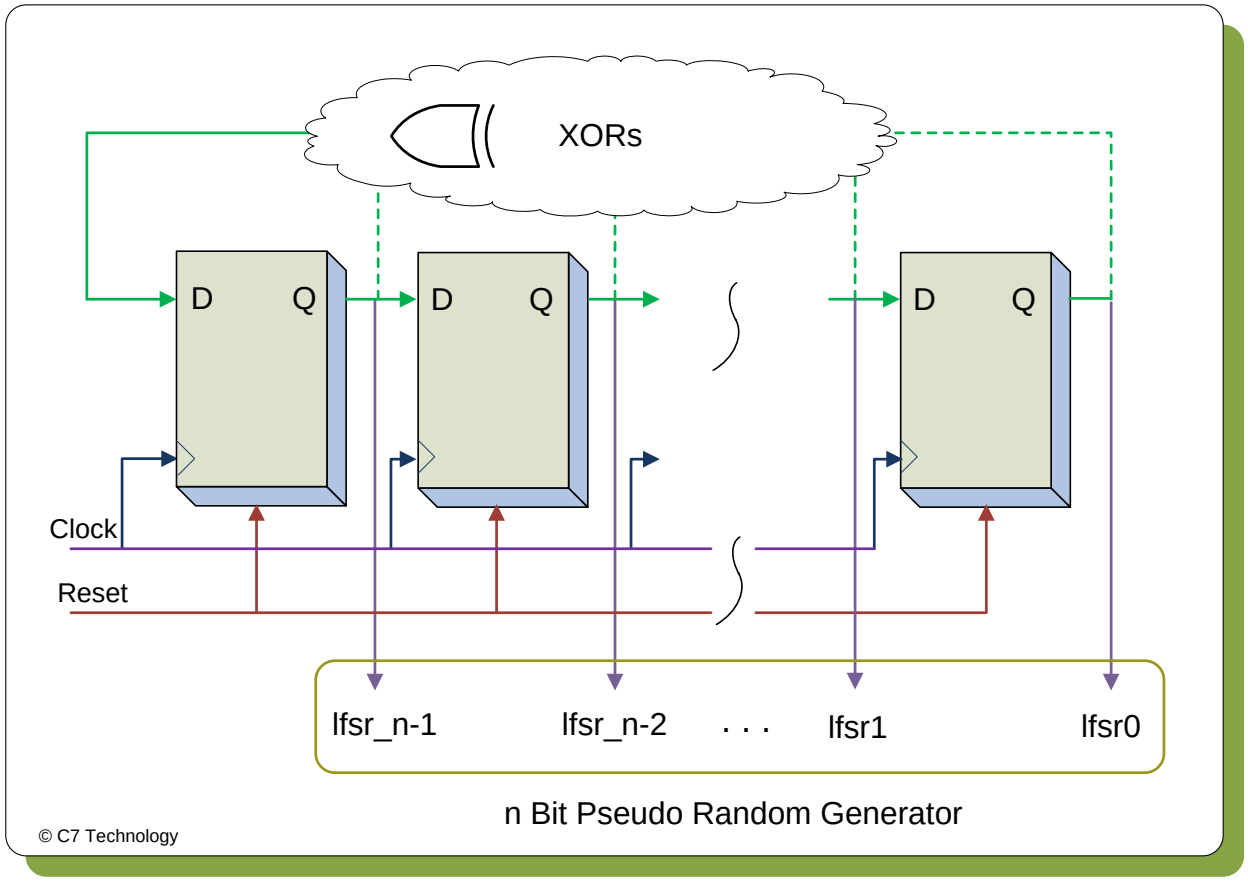

Figura 3- Diagrama general de un LFSR

XOR or XNOR Feeback

El circuito de realimentación puede estar basado en compuertas lógicas XOR o XNOR. Estas compuertas son intercambiables, aunque el LFSR genere los mismos valores, la secuencia sería distinta.

Estado de Bloqueo

Independientemente del número de registros del LFSR, siempre hay un estado en el cual el LFSR se bloquea y no puede salir de ese estado. Para el caso de realimentación con compuertas XOR ese estado de bloqueo se produce cuando todos los registros del LFSR tienen valor '0 '. Es decir, el valor todos '0' *no es parte* de la secuencia pseudo aleatoria. Para el caso del uso de compuertas XNOR en el circuito de realimentación, el estado de bloqueo es de todos '1'. Este

estado de bloqueo se debe a que si por alguna razón, por ejemplo Single Event Upset, el LFSR entra en el estado de todos ‘0’, no hay manera de que salga de ese estado.

Evitando Caer en Estado de Bloqueo

Cuando se utiliza puertas XOR en el circuito de realimentación, el LFSR no secuencia a través de todos los valores binarios en donde todos los bits están en 0 lógico. Por lo tanto, si todos los bits en el LFSR son ‘0’, se seguirán desplazándose todos los ‘0s’ indefinidamente. Por lo que es muy importante, *que se inicialice correctamente* el LFSR, evitando el estado de inicio de todos los bits en ‘0’. Lo mismo ocurre cuando se usa una compuerte XNOR en el circuito de realimentación y todos los bits del LFSR son ‘1’, se seguirán desplazándose todos los ‘1 indefinidamente.

Si se desea realizar un diseño robusto de un LFSR, se debe proporcionar un modo de subsanar este problema en caso de que se presente. Hay tres posibles diseños para como solución del problema:

- Uso de una señal de inicialización, que podría ser un reinicio por ejemplo, para preajustar individualmente o borrar los registros del LFSR con un valor inicial conocido (seed), y distinto del todos ‘0’ o todos ‘1’.
- Uso de una señal de carga, para cargar un valor inicial, llamado semilla (seed), en el LFSR.
- Agregar un circuito extra de lógica para detectar la condición prohibida, solucionar el problema y aún más, que esta condición de todos ‘0’ o todos ‘1’ ser también parte del ciclo generador de PRBS. En este caso el LFSR pasará por todos los 2^n estados.

Agregando el estado todos ‘1’ o todos ‘0’ al LFSR

Para muchas aplicaciones una secuencia -1 estados distintos no puede ser un problema, pero para otras aplicaciones, tales como el modelado de un contador de 4 bits (16 valores distintos), todos los estados son necesarios en la secuencia. Para añadir el estado n-ésimo de la secuencia, el camino de realimentación debe ser modificado con circuitería adicional para asegurar que todos los 2^n valores binarios estén incluidos en la secuencia.

El circuito, o la lógica, a agregar debería ser capaz de incluir el estado de todos ‘0’ entre el estado "00 ... 01» y el estado «10 ... 00" de la secuencia pseudo-aleatoria. Normalmente,

después del estado “00 ... 01", la siguiente entrada serie al LFSR será un '1 ', entonces el estado siguiente de estado “00 ... 01” será el estado "10 ... 00 ". En otras palabras, es necesario detectar cuando todos los bits, excepto el bit 0, del LFSR son cero, y usar esta detección de junto con la realimentación habitual para entrar, permanecer y luego salir del estado todos ‘0’.

Usando el siguiente diagrama se explica debajo el principio de funcionamiento del sistema.

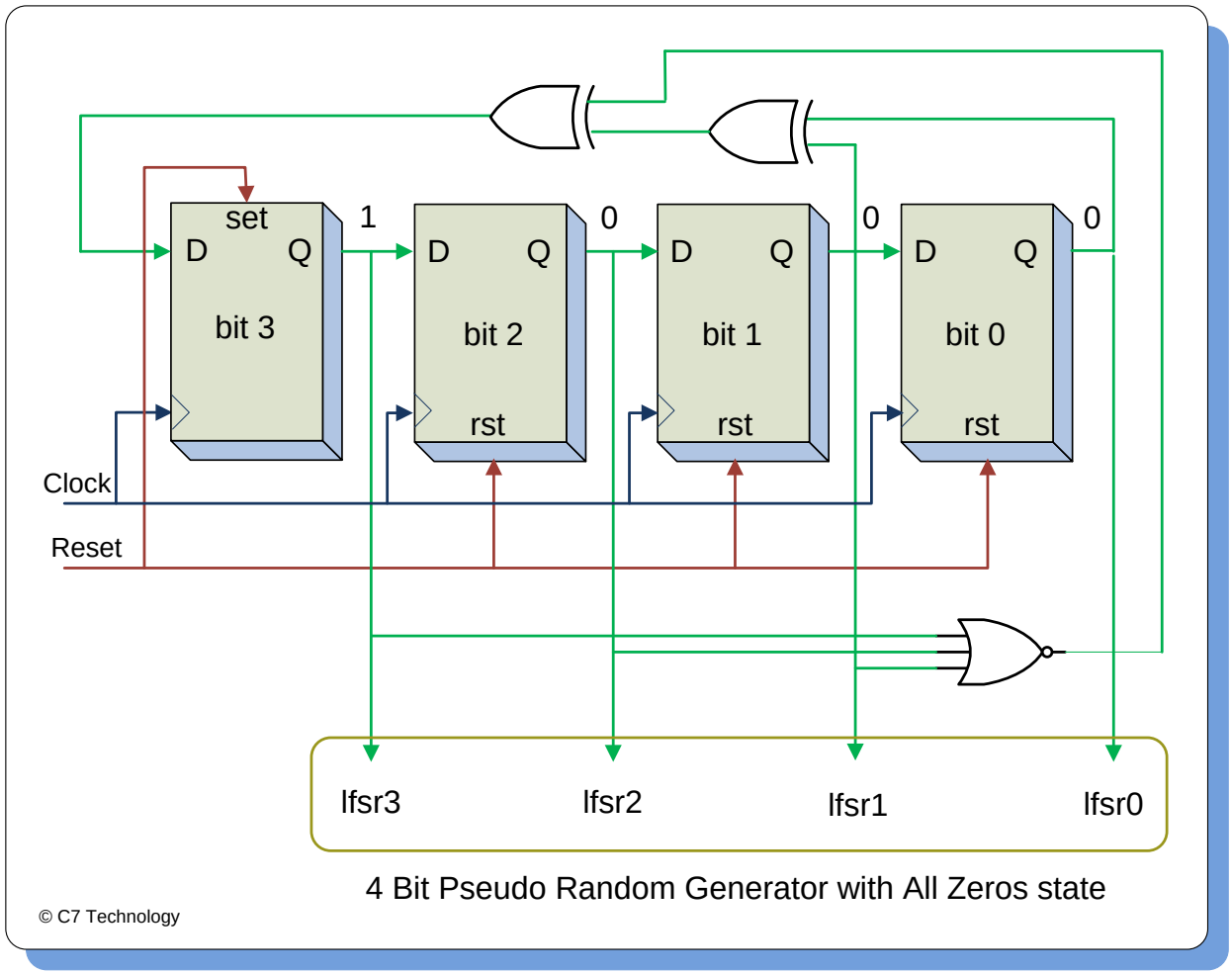

Figura 4 - Diagrama de un LFSR con Detección para la Inclusión del Estdo "0000"

Básicamente, en un LFSR de *n* bits es necesario agregar una compuerta XOR de dos entradas. Siendo uno de la entrada la salida de una compuerta NOR de n-1 entradas, a la que se conectan

todos los bits del registro de desplazamiento, excepto el bit 0, y la otra entrada de XOR es la salida de la XOR de realimentación.

Expresando lo dicho en una forma lógica se obtiene la siguiente expresion:

$$LFSR_all_states = LFSR_Feedback \ \mathbf{XOR} \ (q_{n-1} \ \mathbf{NOR} \ q_{n-2} \ \ldots\ldots \ \mathbf{NOR} \ q_1)$$

Analizando la expresión, se puede ver:

• La expresión $(q_{n-1} \ \mathbf{NOR} \ q_{n-2} \ \ldots\ldots \ \mathbf{NOR} \ q_1)$ sólo puede ser verdadera cuando el LFSR se encuentra en el estado "00. . . 01 " o el estado " 00. . . 00 ".

• Cuando la expresión $(q_{n-1} \ \mathbf{NOR} \ q_{n-2} \ \ldots\ldots \ \mathbf{NOR} \ q_1)$ es falsa, entonces $LFSR_all_states = LFSR_Feedback$, ya que $LFSR_Feedback$ XOR ‘0’ es igual a $LFSR_Feedback$. Esto implica que el circuito funcionara como normalmente lo hace.

• Si el estado actual del LFSR es "00. . . 01 ", el valor de la expresión $(q_{n-1} \ \mathbf{NOR} \ q_{n-2} \ \ldots\ldots \ \mathbf{NOR} \ q_1)$ será '1 'y luego el valor de $LFSR_all_states$ será ‘1’ XOR ‘1’, que es igual a '0'. Por lo tanto, un '0 'se desplaza a la entrada en serie del LFSR en el siguiente flanco activo del reloj. Así, el siguiente estado del LFSR será "00. . . 00 " (después del estado “00. . . 01”).

• Cuando el estado actual del LFSR es "00. . . 00 ", el valor de $LFSR_Feedback$ es '0 ', y el valor de la expresión $(q_{n-1} \ \mathbf{NOR} \ q_{n-2} \ \ldots\ldots \ \mathbf{NOR} \ q_1)$ es igual a '1'. Por lo tanto, $LFSR_all_states$ será el valor de ‘0’ XOR ‘1’, que es '1'. Así, un ‘1’ se desplaza a la entrada en serie del LFSR en el siguiente flanco activo del reloj, después del estado “00. . . 00”. Así, el siguiente estado del LFSR será "10. . . 00 ".

• Una vez que el LFSR se encuentra en estado "10. . . 00 ", la secuencia normal se repetirá hasta que llegue otra vez el estado" 00. . . 01 ", entonces las operaciones descritas anteriormente comienzan de nuevo.

Se puede concluir que la adición de una puerta XOR, y una NOR de n-1 entradas en el camino de realimentación modifica el comportamiento del LFSR permitiendo el estado todos ‘0’. Técnicamente hablando esta operación destruye la linealidad y el circuito ya no es "lineal". Esta versión modificada del LFSR a veces se llama el *contador de Bruijn*.

Para validar lo que se acaba de describir la siguiente figura se muestra el resultado de la simulación del LFSR de 4 bits con el estado de todos los ceros incluidos. Este circuito tiene ahora un ciclo de 16 estados.

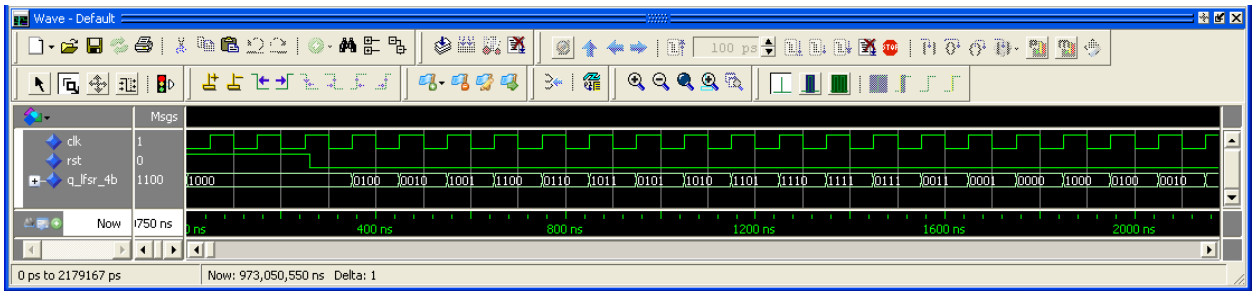

Figura 5 - Simulación con la inclusión del Estado "0000"

Otra gran ventaja de este LFSR, que incluye el estado de todos los ceros, es que ahora el LFSR se puede inicializar en "00. . . 00 ", por ejemplo con el reset del sistema.

El código VHDL que describe el LFSR que agrega el estado todos '0' es el siguiente.

```
------------------------------------------------------------
   -- ejemplo 3 de 4-bits LFSR
   -- © C7T
------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;

------------------------------------------------------------
   -- Enitiy --
------------------------------------------------------------
entity lfsr_all_zeros is
  generic(initial_value: std_logic_vector(3 downto 0):= "1000";
          lfsr_width   : integer := 4);

  port(
          clk      : in std_logic;
          rst      : in std_logic;
          q_lfsr_4b: out std_logic_vector(lfsr_width-1 downto 0)
      );
end lfsr_all_zeros;

------------------------------------------------------------------------------------------
   -- Architecture --
------------------------------------------------------------------------------------------
architecture beh of lfsr_all_zeros is

signal q_lfsr_4b_i  : std_logic_vector(lfsr_width-1 downto 0);
signal nor_det_zeros: std_logic;
signal serial_in    : std_logic;
signal feedback     : std_logic;

begin
-- Serial Input to the LFSR
serial_in <= feedback XOR nor_det_zeros;

-- NOR for the n-1 to n1 bits
nor_det_zeros <=  '1' when q_lfsr_4b_i(lfsr_width-1 downto 1)="000" else
```

```
                                          '0';
-- Feedback XOR from the feedback equation
feedback <= q_lfsr_4b_i(1) xor q_lfsr_4b_i(0);

-- Shifter Process
lfsr_cnt_proc: process(rst, clk)
 begin
       if(rst= '1' ) then
              q_lfsr_4b_i <= initial_value;
       elsif (rising_edge(clk)) then
              q_lfsr_4b_i <= serial_in & q_lfsr_4b_i(q_lfsr_4b_i'high downto
q_lfsr_4b_i'low+1);
       end if;
end process lfsr_cnt_proc;

q_lfsr_4b <= q_lfsr_4b_i;

end architecture beh;

-------------------------------------------------------------------
   -- EoF --
-------------------------------------------------------------------
```

Código VHDL Parametrizado

Finalmente se detalla a continuación el código VHDL que describe en forma genérica un LFSR que puede ser instanciado como un LFSR de 2 a 8 bits. También tiene como opción la inclusión o no del estado de todos '0'.

Las diferentes expresiones de la lógica XOR para registros de desplazamientos de 2 a 8 bits son listadas en un arreglo. Cada fila del arreglo corresponde a un tamaño específico del registro de desplazamiento e indica cuales bits del registro se deben usar en la expresión.

La opción del número de registros del LFSR es configurada usando un *generic* (`lfsr_width`). Del mismo -modo otro *generic* es usado para la inclusión o no del estado de todos '0'. En este caso el valor de este *generic* (`all_zeros_state`), que es un *boolean*, se usa un *if-generate*. Si el valor del generic es verdadero (true), la lógica dentro del if-generate correspondiente (`gen_zero`) será generada para poder de ese modo agregar el estado de todos '0' a la secuencia generada. En caso de que el *generic* `all_zeros_state` sea falso, false, no lógica será agregada y el simple circuito de realimentación original será usado.

Nota: este ejemplo es una modificación del descrito en el libro "RTL GHardware Design Using VHDL", de Pong Chu.

```
------------------------------------------------------------
   -- ejemplo 4 de LFSR Parametrizado
   -- C7T
------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;

-----------------------------------------------------------------------
   -- Enitiy --
-----------------------------------------------------------------------
entity lfsr_generic is
  generic(
      lfsr_width     : natural := 4;
      all_zeros_state: boolean:= false);

  port(
         clk      : in std_logic;
         rst      : in std_logic;
         q_lfsr_4b: out std_logic_vector(lfsr_width-1 downto 0)
       );
end lfsr_generic;

-------------------------------------------------------------------------------------------
   -- Architecture --
-------------------------------------------------------------------------------------------
architecture beh of lfsr_generic is
-- constant declarations
constant max_width    : natural := 8;  -- max # of registers in the LFSR
constant initial_value: std_logic_vector(lfsr_width-1 downto 0) := -- seed
                                                    (0=>'1', others =>'0');
type feedback_array_type is array (2 to max_width) of
                                        std_logic_vector(max_width-1 downto 0);
constant feedback_equation: feedback_array_type :=
                           (2 => (1 | 0 => '1', others => '0'),
                            3 => (1 | 0 => '1', others => '0'),
                            4 => (1 | 0 => '1', others => '0'),
                            5 => (2 | 0 => '1', others => '0'),
                            6 => (1 | 0 => '1', others => '0'),
                            7 => (3 | 0 => '1', others => '0'),
                            8 => (4 | 3 | 2 | 0 => '1', others => '0'));
-- signal declarations
signal q_lfsr_4b_i  : std_logic_vector(lfsr_width-1 downto 0);
signal nor_detect_0s: std_logic;
signal serial_in    : std_logic;
signal feedback     : std_logic;

begin

-------------------------------------------------------------

-- Process to get feedback XOR from the feedback equation --
feedb_proc: process(q_lfsr_4b_i)
-- process local declarations
constant tap_constant: std_logic_vector(max_width-1 downto 0)
                                          := feedback_equation(lfsr_width);
variable tmp: std_logic;
begin
```

```
       tmp := '0';
       bit_xor: for i in 0 to lfsr_width-1 loop
              tmp := tmp xor (q_lfsr_4b_i(i) and tap_constant(i));
       end loop bit_xor;
       feedback <= tmp;
end process feedb_proc;
-------------------------------------------------------------

-------------------------------------------------------------
-- Shifter Process
lfsr_cnt_proc: process(rst, clk)
 begin
       if(rst= '1' ) then
              q_lfsr_4b_i <= initial_value;
       elsif (rising_edge(clk)) then
          -- shift operation: b3->b2, b2->b1, b1->b0
              shifter_loop: for i in 3 downto 1 loop
                     q_lfsr_4b_i(i-1) <= q_lfsr_4b_i(i);
              end loop shifter_loop;
          -- Serial Input to the b3 of the LFSR
      q_lfsr_4b_i(q_lfsr_4b_i'high) <= serial_in;
       end if;
end process lfsr_cnt_proc;
q_lfsr_4b <= q_lfsr_4b_i;

-------------------------------------------------------------
-- piece of code to be generated when the 'all zeros'
-- state is needed
-------------------------------------------------------------
gen_zero: if (all_zeros_state = TRUE) generate
       nor_detect_0s <=  '1' when q_lfsr_4b_i(lfsr_width-1 downto 1)=
                            (q_lfsr_4b_i(lfsr_width-1 downto 1)'range=>'0') else
                         '0';
       -- Serial Input to the LFSR
   serial_in <= feedback XOR nor_detect_0s;
end generate gen_zero;
-------------------------------------------------------------

-------------------------------------------------------------
-- when the 'all zeros' state is not needed
-------------------------------------------------------------
gen_no_zeros: if (all_zeros_state = FALSE) generate
       -- Serial Input to the LFSR
   serial_in <= feedback;
end generate gen_no_zeros;
-------------------------------------------------------------
end architecture beh;

-------------------------------------------------------------
   -- EoF --
-------------------------------------------------------------
```

Conclusión

Aspectos prácticos de diseño e implementación de LFSRs y detallado código VHDL han sido descritos en esta nota técnica. Espero sean de utilidad.

C7 Technology

http://www.c7technology.com

Copyright © 2016
All rights reserved

Este documente puede ser copiado y distribuido libremente.
Si es copiado por favor mantenga el logo e información de la compañía.
Si parte del documento es cortado y pegado, por favor mencione *C7 Technology* como fuente de información. Gracias.