Performance y uso de recursos de contadores basados en Linear FeedBack Shift-Registers (LFSRs). Diseño de una megafunción parametrizable usando ALTERA.

Autores: Guillermo A. Jaquenod (*) y Marisa R. De Giusti (**)

Resumen:

Los contadores conforman un caso muy habitual de máquina secuencial sincrónica, siendo usados en tareas de barrido de memoria, conteo de eventos, generación de retardos, división de frecuencias, etc. En ciertos casos no es condición indispensable que la secuencia de conteo sea natural (0,1,2..), sino sólo que dicha secuencia sea predecible, y en este caso ciertas alternativas (códigos Gray, contadores Johnson, generadores de secuencias pseudo-aleatorios) pueden resultar ventajosas. Este artículo analiza la realización de contadores pseudo-aleatorios usando Linear-Feedback Shift Registers, describe su implementación física usando dispositivos MAX de ALTERA, y compara la performance y uso de recursos resultante en comparación a soluciones basadas en contadores binarios.

(*) Profesor Titular, Facultad de Ingeniería, Universidad Nacional del Centro de la Provincia de Buenos Aires.
Dirección postal: A.Del Valle 5737 – (7400) OLAVARRÍA – ARGENTINA
Correo Electrónico: <*chipi@satlink.com*>

(**) Investigador Adjunto sin Director – Comisión de Investigaciones Científicas de la Provincia de Buenos Aires. Profesor Titular, Facultad de Ingeniería, Universidad Nacional de La Plata.
Dirección postal: Calle 24 # 709 – (1900) LA PLATA – ARGENTINA
Correo Electrónico: <*marisadg@volta.ing.unlp.edu.ar*>

1. Introducción

Los contadores conforman un caso muy habitual de máquina secuencial sincrónica, siendo usados en tareas de barrido de memoria, retardos, división de frecuencias, etc. y en ciertos casos no es condición indispensable que la secuencia de conteo sea binaria natural (0,1,2..), sino sólo que dicha secuencia sea predecible:

- En ciertas aplicaciones de barrido de memorias (P.Ej: captura de datos en un analizador lógico, generadores de direcciones en memorias tipo FIFO), tanto las direcciones de escritura como las de lectura son generadas por el mismo o similares contadores, por lo que no importa en qué lugar se almacenan los datos, sino que sean leídos en el mismo orden en que fueron escritos.
- En el caso de la generación de retardos o división de frecuencias, a partir de un estado inicial sólo importa detectar el estado final que corresponde a la cantidad de ciclos de reloj deseados, sin importar el significado numérico del número a detectar.

Cuando la performance de los contadores se analiza sólo desde el punto de vista del empleo de flipflops, surge que los contadores binarios convencionales hacen uso óptimo de los registros, pues usando N flipflops es posible generar hasta 2^N estados distintos; sin embargo, si este análisis se realiza sobre el circuito completo se nota que los contadores binarios requieren circuitos combinatorios complejos o de gran fan-in para la generación de los sucesivos estados.

Las alternativas tradicionales más usuales al empleo de contadores de secuencia binaria son:

- Contadores de secuencia Gray: también ofrecen 2^N estados distintos usando N flipflops, donde sólo un flipflop conmuta de estado en cada ciclo de reloj. Su uso óptimo de flipflops queda opacado por la gran cantidad de lógica necesaria para sintetizar la tabla de transición de estados.

- Contadores en anillo: son básicamente registros de desplazamiento donde un único bit "activo" (ya sea un '1' o un '0') recorre circularmente las sucesivas etapas del registro. El aprovechamiento de los flipflops es el peor posible (N estados requieren N flipflops) pero la lógica combinatoria necesaria es nula; sólo se requiere poder inicializar al sistema con un único bit activo en la cadena.
- Contadores Johnson: son registros de desplazamiento donde la salida de la última etapa se realimenta negada en la entrada y se fuerza un dado estado inicial (todos ceros o todos unos, por ejemplo). El uso de flipflops es algo mejor (N flipflops permiten obtener 2N estados distintos), la lógica combinatoria necesaria es nula y sólo se requiere poder inicializar al sistema al estado inicial.

El empleo de generadores de secuencias pseudo-aleatorias de longitud máxima ofrece en muchos casos una alternativa ventajosa, pues permite el aprovechamiento cuasi óptimo de los flipflops usados (con N flipflops es posible generar hasta 2^N–1 estados distintos) a la vez que requiere una lógica combinatoria casi nula. Estos sistemas secuenciales se basan en un registro de desplazamiento realimentado donde el valor a ingresar en la primer etapa resulta de OR-exclusivo (o del NOR exclusivo) del valor de la última etapa con el de ciertas etapas intermedias, y por ello son llamados LFSR (por Linear Feedback Shift Registers). Si bien el diseño estándar de los LFSR basados en XOR excluye al estado en que todos los registros valen 0 (o todos 1, si el diseño se basa en XNOR), estos circuitos pueden modificarse para abarcar los 2^N estados distintos, con un simple agregado a la lógica de realimentación.

La decisión del tipo de contador a emplear no depende sólo de la aplicación sino también de la familia de lógica programable a usar; por ejemplo, la familia FLEX de ALTERA ofrece prestaciones especiales ("LE Counter Operating Mode") que simplifican y optimizan enormemente el diseño de contadores binarios, pero en familias sin estas prestaciones el uso de contadores LFSR puede minimizar el consumo de registros y de lógica, y posibilitar una frecuencia de conteo mucho más alta que la obtenible mediante otras soluciones.

Este artículo describe la implementación física de contadores LFSR usando dispositivos MAX de ALTERA, y compara la performance y uso de recursos resultante en comparación al uso de contadores binarios.

2. Circuitos básicos de LFSRs

La figura 1 muestra el esquema de un LFSR de 4 bits, donde la realimentación a la entrada de la primer etapa es calculada en base al NOR-exclusivo de las salidas de la tercer y cuarta etapa.

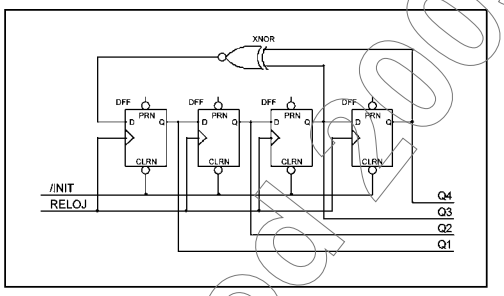

FIGURA 1

Si se aplican pulsos de reloj y se analizan los sucesivos valores de las salidas Q[4..1] se puede observar que (en hexadecimal) ellos son 1, 3, 7, E, D, B, 6, C, 9, 2, 5, A, 4, 8, 0 para volver a 1 luego de 15 (2^4-1) ciclos de reloj, y así sucesivamente.

En este caso deben notarse dos circunstancias:
1. en operación normal el LFSR no atraviesa el estado F (todos '1'), que es el estado "prohibido" de los LFSR basados en XNOR.
2. debe garantizarse que el estado de arranque no sea F, porque de entrar el LFSR en ese estado quedaría permanentemente allí.

La decisión de qué otros bits realimentar, además del último, se sustenta en una teoría matemática compleja (teoría de campos finitos de Galois), y para lo cual, a los fines prácticos, se anexa en el Apéndice I una tabla que indica, para LFSRs de distinta longitud ("N") qué realimentación de bits produce una secuencia de largo máximo 2^N-1.

3. Circuitos LFSR con 2^N estados

Se ha visto que un LFSR convencional basado en XNOR, en su modo normal de operación no atraviesa el estado "11...11", y que de estar en ese estado queda en él de modo permanente.

Si se desea que la secuencia del LFSR pase de 2^N-1 a 2^N estados deberá incluirse al estado "11.11", definiendo algún modo de ingresar a ese estado desde el modo "normal" y poder salir de él al ciclo siguiente.
Para ello deben tenerse en cuenta dos hechos:

1. Por usar un registro de desplazamiento, la única forma para que el LFSR quede cargado con "todos unos" es detectar cuando todos los registros menos el último valen '1' para ingresar un '1' en la entrada (momento en el cual la función XNOR de un LFSR estándar intenta ingresar un '0').
2. De igual modo, para que un LFSR, ahora cargado con todos unos, no quede bloqueado en ese estado, debe ingresarse un '0' en su entrada (aunque al estar cargado con "todos 1" la función XNOR intente ingresar un '1').

De lo anterior surge que para ampliar los estados posibles en un LFSR a 2^N basta con detectar cuándo los N-1 registros menos significativos valen '1' y en ese caso invertir la función propuesta por la compuerta XNOR de realimentación.

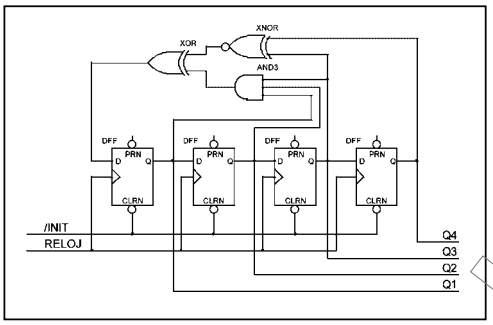

FIGURA 2

La figura 2 muestra el esquema de un LFSR de 4 bits, pero ahora con 2^N estados, donde a la realimentación se ha agregado una AND para la detección del valor '1' en los N-1 primeros registros, que a través de una XOR puede invertir el nuevo valor a ingresar al LFSR. Si se aplican pulsos de reloj y se analizan los sucesivos valores de las salidas Q[4..1] se observa que ellos son 1, 3, 7, F, E, D, B, 6, C, 9, 2, 5, A, 4, 8, 0 para volver a 1 luego de 16 (2^4) ciclos de reloj. Ahora también aparece el estado "F", que no estaba presente en el circuito de la figura 1.

4. Diseño de LFSRs usando dispositivos MAX de ALTERA

Para explotar al máximo un diseño usando una dada familia es fundamental analizar las prestaciones especiales de dicha familia y la complejidad de las funciones a resolver.

En el caso de las familias MAX7000/MAX3000A de ALTERA, la figura 3 muestra el esquema general de una macrocelda de esta familia, donde se han obviado ciertos detalles (P.Ej: fast inputs).

Cada macrocelda resuelve una función combinatoria mediante la Suma de Productos (SOP), contando para ellos con hasta 5 términos producto propios por macrocelda; y la salida de esta función combinatoria puede salir directamente, o pasar a través de un flipflop en el caso de sistemas secuenciales.

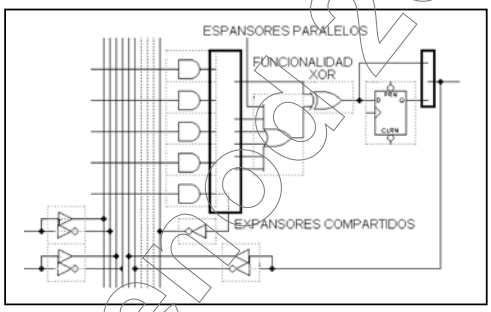

FIGURA 3

Este comportamiento básico queda altamente potenciado por ciertas funcionalidades adicionales:

- Funcionalidad XOR: la salida del término "suma de productos" puede pasar a través de una compuerta XOR, cuya otra entrada puede estar fija al valor '0', al valor '1' (en que opera como negador), o provenir de un término producto, caso en el que posibilita el uso de algoritmos de síntesis mucho más poderosos que los disponibles para SOP (como Karnaugh o Quine McCluskey).
- Expansores paralelos: en el caso de funciones que requieren la suma de muchos términos producto, una macrocelda puede integrar a la entrada de la compuerta OR términos suma provenientes de otras macroceldas, hasta totalizar la suma de 20 términos producto.
- Expansores compartidos: si un término producto no usado en una dada macrocelda se realimenta negado, este término producto (que por De Morgan se transforma en un término suma) puede ser usado por otro término producto de esta u otra macrocelda para generar expresiones complejas.
- Tipo de flipflop: macroscópicamente, el flipflop D puede ser re-sintetizado para comportarse como un flipflop T, JK o RS.

4.1 LFSRs de secuencia 2^N-1

El diseño de un LFSR de N etapas comprende dos bloques, el shift register y la lógica de realimentación:

El diseño del Shift-Register es elemental, pues dado que cada macrocelda MAX incluye un flipflop cuya salida se realimenta hacia dentro del chip, y que esta salida es la entrada a la etapa siguiente, solo se requerirán N macroceldas para construir un shift register de N etapas.

Para evaluar la complejidad de la función de realimentación debe observarse la tabla del Apéndice 1, que muestra que en la mayoría de los casos la función de realimentación es el XOR/XNOR de dos señales, en ciertos casos de 4 señales, y en sólo un caso (N=37) de seis señales.

- En el caso de un XOR de dos señales, esta función se resuelve mediante la suma de dos términos producto, bastando con los recursos combinatorios de la primer macrocelda
- En el caso que sea el XOR de cuatro señales (16 minitérminos), esta función requeriría la suma de ocho términos producto, pero al emplear la funcionalidad XOR propia de las MAX esta función puede resolverse mediante sólo cinco términos producto (uno de ellos controlando dicha funcionalidad), por lo que también bastan los recursos combinatorios de la primer macrocelda.
- En el caso N=37, donde es necesario realizar el XOR de seis señales (taps 37, 5, 4, 3, 2 y 1), esta función también puede resolverse mediante sólo 37 macroceldas, pero ahora sintetizando un flipflop T para la primer etapa (con lo que resuelve el feedback de la etapa 1) y 16 expansores compartidos para la síntesis de los 16 minitérminos que requiere el XOR de las 5 señales restantes de feedback.

Se observa entonces que, para LFSRs de hasta 66 etapas (ciclo de conteo de $1{,}36 \times 10^{39}$), sólo se requieren tantas macroceldas como etapas tenga el LFSR.

4.2 LFSRs de secuencia 2^N

La síntesis de LFSRs con secuencia de largo 2N no introduce cambios para la síntesis de las etapas 2 y subsiguientes, pero eleva la complejidad de la función de realimentación a causa de la necesidad de un término más en el cómputo del XOR/XNOR, término que resulta del AND de N-1 señales de realimentación.

En este caso la síntesis de LFSRs de elevada cantidad de etapas comienza a tener limitaciones definidas por las 32 señales como máximo que pueden realimentarse hacia cada LAB desde el PIA, con lo que la complejidad de la síntesis pasa por un problema de cableado.

5. Un LFSR de largo parametrizable

Si se desea contar con una función "universal" para el uso de LFSRs, puede realizarse una descripción en AHDL usando parámetros que definan la cantidad de etapas, el tipo de realimentación, y si se desea realizar un LFSR con secuencia de largo 2^{N-1} o de 2^N.

En el Apéndice II se adjunta un listado de dicha función, donde puede notarse como el uso de sentencias IF..GENERATE permite que el "hardware" resulte parametrizable.

6. Comparaciones de performance

Para poder comparar la performance de una solución usando LFSRs de secuencia 2^N-1 y 2^N frente a un contador binario se compilaron las tres alternativas usando el MAX+plus II v.10.1, para un mismo dispositivo (EPM7256BTC100-5) y con las mismas opciones de síntesis lógica, en modo NORMAL. Para la síntesis del contador se empleó la megafunción LPM_COUNTER (optimizada por ALTERA) empleando sólo las entradas de reloj e inicialización asincrónica. Para el diseño del LFSR se empleó la solución basada en XNOR, también teniendo como entradas sólo las señales de reloj e inicialización asincrónica.

- En el caso de los LFSR de secuencia 2^N-1, y para aquellos casos en que la realimentación se realiza mediante el XNOR de dos o cuatro señales (todos menos el caso N=37), la solución emplea tantas macroceldas como etapas, y permite una frecuencia máxima de operación de 188,67MHz; sólo en el caso de un XNOR de 6 etapas (N=37) la solución debe emplear 16 expansores compartidos, lo que provoca que pese a seguir siendo usados N elementos lógicos, la frecuencia máxima de operación caiga a 116.27MHz.
- En el caso de los LFSR de secuencia 2^N, el compilador sólo encuentra soluciones ruteables para N<33. Para N desde 3 hasta 7 se usan N elementos lógicos, y la velocidad es comparable al caso previo, con una ligera disminución de Fmax desde 188,67MHz hasta 185,18MHz. Desde N=8 hasta N=32 esta solución deja de ser competitiva, pues se usan N+2 elementos lógicos con una Fmax de 114,94MHz.

- En el caso de los contadores el MAX+plus II configura a los flipflops como FlipFlop T, y para el rango N<=36 la solución emplea tantas macroceldas como etapas; la máxima frecuencia de operación para N=8 es de 185,18MHz, frecuencia que decae suavemente hasta N=36, donde es de 156,25MHz. A partir de allí, la cantidad finita de caminos de entrada a cada LAB provoca que el compilador deba fraccionar la lógica combinatoria, con lo que el uso de macroceldas se incrementa sobre el número de etapas (N=37 requiere 41 Les, N=66 requiere 73 Les), y la frecuencia cae a 79,36MHz (N=37) para llegar a 51,28MHz para N=66.

De la comparación surge una clara ventaja de los LFSRs de largo 2^N-1 frente a los contadores cuando se emplean hasta 36 etapas, ventaja que se acrecienta cuanto mayor es la cantidad de etapas; para N=37 la máxima velocidad de operación de un LFSR es cerca de tres veces, y para N=66 más de cuatro veces la velocidad de un contador binario.
Vale notar que en el caso de las familias MAX el uso de LFSRs basados en XNOR tiene ciertas ventajas frente a los basados en XOR: en el primer caso para la señal de inicialización de los flipflops (.clrn) puede usarse un recurso de cableado global, en tanto que en el segundo caso la señal de inicialización (ahora .prn) consume un término producto; por ello, de usarse realimentación XOR todos los LFSR con feedback cuádruple también tendrían una frecuencia máxima de operación de 128,2MHz.

7. Conclusiones

El uso de LFSRs con secuencia de largo 2^N-1 tiene indudables ventajas frente al empleo de contadores de secuencia binaria, las que se acrecientan en el caso de elevada cantidad de etapas. Para el caso de LFSRs con secuencia de largo 2^N estas ventajas disminuyen ante la necesidad de realizar una función lógica de gran fan-in para construir el circuito necesario para entrar y salir del estado adicional, siendo de posible utilidad sólo en el caso de LFSRs de menos de 8 etapas.

8. Bibliografía:

[1]. Alfke, P. "Efficient Shift Registers, LFSR Counters, and Long Pseudo Random Sequence Generators". Application Note XAPP 052, Version 1.1, XILINX Corp., 1996.

[2]. Altera. "1999 Device Data Book", A-DB-0599-01, Altera Corp., 1999, USA.

[3]. Wakerly, J. "Diseño Digital. Principios y Prácticas". Prentice Hall, ISBN 968-880-244-1.

APÉNDICE I

Tabla se selección de los bits de realimentación, para LFSRs desde 3 hasta 66 etapas, que generan secuencias de largo máximo:

N	Realimentacion	N	Realimentacion	N	Realimentacion	N	Realimentacion
3	3, 2	19	19, 6, 2, 1	35	35, 33	51	51, 50, 36, 35
4	4, 3	20	20, 17	36	36, 25	52	52, 49
5	5, 3	21	21, 19	37	37,5,4,3,2,1	53	53, 52, 38, 37
6	6, 5	22	22, 21	38	38, 6, 5, 1	54	54, 53, 18, 17
7	7, 6	23	23, 18	39	39, 35	55	55, 31
8	8, 6, 5, 4	24	24, 23, 22, 17	40	40, 38, 21, 19	56	56, 55, 35, 34
9	9, 5	25	25, 22	41	41, 38	57	57, 50
10	10, 7	26	26, 6, 2, 1	42	42, 41, 20, 19	58	58, 39
11	11, 9	27	27, 5, 2, 1	43	43, 42, 38, 37	59	59, 58, 38, 37
12	12, 6, 4, 1	28	28, 25	44	44, 43, 18, 17	60	60, 59
13	13, 4, 3, 1	29	29, 27	45	45, 44, 42, 41	61	61, 60, 46, 45
14	14, 5, 3, 1	30	30, 6, 4, 1	46	46, 45, 26, 25	62	62, 61, 6, 5
15	15, 14	31	31, 28	47	47, 42	63	63, 62
16	16, 15, 13, 4	32	32, 22, 2, 1	48	48, 47, 21, 20	64	64, 63, 61, 60
17	17, 14	33	33, 20	49	49, 40	65	65, 47
18	18, 11	34	34, 27, 2, 1	50	50, 49, 24, 23	66	66, 65, 57, 56

APÉNDICE II

Listado de una megafunción parametrizable en AHDL para la síntesis de LFSRs de 3 hasta 66 etapas con realimentacion por XNOR que generan secuencias de largo máximo 2^N-1:

```
--------------------------------------------------------------------------------
PARAMETERS (LFSRTAPS=8);      -- number of taps of the LFSR Must be in the range 3..66
--------------------------------------------------------------------------------
ASSERT (LFSRTAPS >= 3)
       REPORT         "LFSRTAPS (%) must be >= 3" LFSRTAPS
       SEVERITY              ERROR;

ASSERT (LFSRTAPS <= 66)
       REPORT         "LFSRTAPS (%) must be <= 66" LFSRTAPS
       SEVERITY              ERROR;
--------------------------------------------------------------------------------
SUBDESIGN lfsrn (
 n_init,clock : INPUT; -- for synchronous control
 taps[LFSRTAPS..1] : OUTPUT;  -- synchronous LFSR output
)
--------------------------------------------------------------------------------
VARIABLE
 fftaps[LFSRTAPS..1] : DFF;
 xorout0 : NODE;
--------------------------------------------------------------------------------
BEGIN
------
 fftaps[].clk = clock; -- all FF operate synchronously
------
 fftaps[].clrn  = GLOBAL(n_init); -- the initial state is 0..0
------
 FOR i IN 2 TO LFSRTAPS
   GENERATE fftaps[i].d = fftaps [i-1].q; -- chain of flipflops creating a ShiftRegister
 END GENERATE;
%------
- for each N-taps LFSR, the Max.Length sequence is obtained with different feedback
-------%
 IF (LFSRTAPS==3) GENERATE xorout0 = fftaps[3].q XOR fftaps[2].q; END GENERATE;
 IF (LFSRTAPS==4) GENERATE xorout0 = fftaps[4].q XOR fftaps[3].q; END GENERATE;
 IF (LFSRTAPS==5) GENERATE xorout0 = fftaps[5].q XOR fftaps[3].q; END GENERATE;
 IF (LFSRTAPS==6) GENERATE xorout0 = fftaps[6].q XOR fftaps[5].q; END GENERATE;
 IF (LFSRTAPS==7) GENERATE xorout0 = fftaps[7].q XOR fftaps[6].q; END GENERATE;
 IF (LFSRTAPS==8) GENERATE
   xorout0 = fftaps[8].q XOR fftaps[6].q XOR fftaps[5].q XOR fftaps[4].q;
   END GENERATE;
 IF (LFSRTAPS==9) GENERATE xorout0 = fftaps[9].q XOR fftaps[5].q; END GENERATE;
 IF (LFSRTAPS==10) GENERATE xorout0 = fftaps[10].q XOR fftaps[7].q; END GENERATE;
```

```
IF (LFSRTAPS==11) GENERATE xorout0 = fftaps[11].q XOR fftaps[9].q; END GENERATE;
IF (LFSRTAPS==12) GENERATE
  xorout0 = fftaps[12].q XOR fftaps[6].q XOR fftaps[4].q XOR fftaps[1].q;
  END GENERATE;
IF (LFSRTAPS==13) GENERATE
  xorout0 = fftaps[13].q XOR fftaps[4].q XOR fftaps[3].q XOR fftaps[1].q;
  END GENERATE;
IF (LFSRTAPS==14) GENERATE
  xorout0 = fftaps[14].q XOR fftaps[5].q XOR fftaps[3].q XOR fftaps[1].q;
END GENERATE;
IF (LFSRTAPS==15) GENERATE xorout0 = fftaps[15].q XOR fftaps[14].q; END GENERATE;
IF (LFSRTAPS==16) GENERATE
  xorout0 = fftaps[16].q XOR fftaps[15].q XOR fftaps[13].q XOR fftaps[4].q;
  END GENERATE;
IF (LFSRTAPS==17) GENERATE xorout0 = fftaps[17].q XOR fftaps[14].q; END GENERATE;
IF (LFSRTAPS==18) GENERATE xorout0 = fftaps[18].q XOR fftaps[11].q; END GENERATE;
IF (LFSRTAPS==19) GENERATE
  xorout0 = fftaps[19].q XOR fftaps[6].q XOR fftaps[2].q XOR fftaps[1].q;
  END GENERATE;
IF (LFSRTAPS==20) GENERATE xorout0 = fftaps[20].q XOR fftaps[17].q; END GENERATE;
IF (LFSRTAPS==21) GENERATE xorout0 = fftaps[21].q XOR fftaps[19].q; END GENERATE;
IF (LFSRTAPS==22) GENERATE xorout0 = fftaps[22].q XOR fftaps[21].q; END GENERATE;
IF (LFSRTAPS==23) GENERATE xorout0 = fftaps[23].q XOR fftaps[18].q; END GENERATE;
IF (LFSRTAPS==24) GENERATE
  xorout0 = fftaps[24].q XOR fftaps[23].q XOR fftaps[22].q XOR fftaps[17].q;
  END GENERATE;
IF (LFSRTAPS==25) GENERATE xorout0 = fftaps[25].q XOR fftaps[22].q; END GENERATE;
IF (LFSRTAPS==26) GENERATE
  xorout0 = fftaps[26].q XOR fftaps[6].q XOR fftaps[2].q XOR fftaps[1].q;
  END GENERATE;
IF (LFSRTAPS==27) GENERATE
  xorout0 = fftaps[27].q XOR fftaps[5].q XOR fftaps[2].q XOR fftaps[1].q;
  END GENERATE;
IF (LFSRTAPS==28) GENERATE xorout0 = fftaps[28].q XOR fftaps[25].q; END GENERATE;
IF (LFSRTAPS==29) GENERATE xorout0 = fftaps[29].q XOR fftaps[27].q; END GENERATE;
IF (LFSRTAPS==30) GENERATE
  xorout0 = fftaps[30].q XOR fftaps[6].q XOR fftaps[4].q XOR fftaps[1].q;
  END GENERATE;
IF (LFSRTAPS==31) GENERATE xorout0 = fftaps[31].q XOR fftaps[28].q; END GENERATE;
IF (LFSRTAPS==32) GENERATE
  xorout0 = fftaps[32].q XOR fftaps[22].q XOR fftaps[2].q XOR fftaps[1].q;
  END GENERATE;
IF (LFSRTAPS==33) GENERATE xorout0 = fftaps[33].q XOR fftaps[20].q; END GENERATE;
IF (LFSRTAPS==34) GENERATE
  xorout0 = fftaps[34].q XOR fftaps[27].q XOR fftaps[2].q XOR fftaps[1].q;
  END GENERATE;
IF (LFSRTAPS==35) GENERATE xorout0 = fftaps[35].q XOR fftaps[33].q; END GENERATE;
IF (LFSRTAPS==36) GENERATE xorout0 = fftaps[36].q XOR fftaps[25].q; END GENERATE;
IF (LFSRTAPS==37) GENERATE xorout0 = fftaps[37].q XOR fftaps[5].q XOR fftaps[4].q
                                XOR fftaps[3].q XOR fftaps[2].q XOR fftaps[1].q;
  END GENERATE;
IF (LFSRTAPS==38) GENERATE
  xorout0 = fftaps[38].q XOR fftaps[6].q XOR fftaps[5].q XOR fftaps[1].q;
  END GENERATE;
IF (LFSRTAPS==39) GENERATE xorout0 = fftaps[39].q XOR fftaps[35].q; END GENERATE;
IF (LFSRTAPS==40) GENERATE
  xorout0 = fftaps[40].q XOR fftaps[38].q XOR fftaps[21].q XOR fftaps[19].q;
  END GENERATE;
IF (LFSRTAPS==41) GENERATE xorout0 = fftaps[41].q XOR fftaps[38].q; END GENERATE;
IF (LFSRTAPS==42) GENERATE
  xorout0 = fftaps[42].q XOR fftaps[41].q XOR fftaps[20].q XOR fftaps[19].q;
  END GENERATE;
IF (LFSRTAPS==43) GENERATE
  xorout0 = fftaps[43].q XOR fftaps[42].q XOR fftaps[38].q XOR fftaps[37].q;
  END GENERATE;
IF (LFSRTAPS==44) GENERATE
  xorout0 = fftaps[44].q XOR fftaps[43].q XOR fftaps[18].q XOR fftaps[17].q;
  END GENERATE;
IF (LFSRTAPS==45) GENERATE
  xorout0 = fftaps[45].q XOR fftaps[44].q XOR fftaps[42].q XOR fftaps[41].q;
  END GENERATE;
IF (LFSRTAPS==46) GENERATE
  xorout0 = fftaps[46].q XOR fftaps[45].q XOR fftaps[26].q XOR fftaps[25].q;
  END GENERATE;
IF (LFSRTAPS==47) GENERATE xorout0 = fftaps[47].q XOR fftaps[42].q; END GENERATE;
IF (LFSRTAPS==48) GENERATE
  xorout0 = fftaps[48].q XOR fftaps[47].q XOR fftaps[21].q XOR fftaps[20].q;
  END GENERATE;
IF (LFSRTAPS==49) GENERATE xorout0 = fftaps[49].q XOR fftaps[40].q; END GENERATE;
IF (LFSRTAPS==50) GENERATE
  xorout0 = fftaps[50].q XOR fftaps[49].q XOR fftaps[24].q XOR fftaps[23].q;
```

```
  END GENERATE;
 IF (LFSRTAPS==51) GENERATE
   xorout0 = fftaps[51].q XOR fftaps[50].q XOR fftaps[36].q XOR fftaps[35].q;
   END GENERATE;
 IF (LFSRTAPS==52) GENERATE xorout0 = fftaps[52].q XOR fftaps[49].q; END GENERATE;
 IF (LFSRTAPS==53) GENERATE
   xorout0 = fftaps[53].q XOR fftaps[52].q XOR fftaps[38].q XOR fftaps[37].q;
   END GENERATE;
 IF (LFSRTAPS==54) GENERATE
   xorout0 = fftaps[54].q XOR fftaps[53].q XOR fftaps[18].q XOR fftaps[17].q;
   END GENERATE;
 IF (LFSRTAPS==55) GENERATE xorout0 = fftaps[55].q XOR fftaps[31].q; END GENERATE;
 IF (LFSRTAPS==56) GENERATE
   xorout0 = fftaps[56].q XOR fftaps[55].q XOR fftaps[35].q XOR fftaps[34].q;
   END GENERATE;
 IF (LFSRTAPS==57) GENERATE xorout0 = fftaps[57].q XOR fftaps[50].q; END GENERATE;
 IF (LFSRTAPS==58) GENERATE xorout0 = fftaps[58].q XOR fftaps[39].q; END GENERATE;
 IF (LFSRTAPS==59) GENERATE
   xorout0 = fftaps[59].q XOR fftaps[58].q XOR fftaps[38].q XOR fftaps[37].q;
   END GENERATE;
 IF (LFSRTAPS==60) GENERATE xorout0 = fftaps[60].q XOR fftaps[59].q; END GENERATE;
 IF (LFSRTAPS==61) GENERATE
   xorout0 = fftaps[61].q XOR fftaps[60].q XOR fftaps[46].q XOR fftaps[45].q;
   END GENERATE;
 IF (LFSRTAPS==62) GENERATE
   xorout0 = fftaps[62].q XOR fftaps[61].q XOR fftaps[6].q XOR fftaps[5].q;
   END GENERATE;
 IF (LFSRTAPS==63) GENERATE xorout0 = fftaps[63].q XOR fftaps[62].q; END GENERATE;
 IF (LFSRTAPS==64) GENERATE
   xorout0 = fftaps[64].q XOR fftaps[63].q XOR fftaps[61].q XOR fftaps[60].q;
   END GENERATE;
 IF (LFSRTAPS==65) GENERATE xorout0 = fftaps[65].q XOR fftaps[47].q; END GENERATE;
 IF (LFSRTAPS==66) GENERATE
   xorout0 = fftaps[66].q XOR fftaps[65].q XOR fftaps[57].q XOR fftaps[56].q;
   END GENERATE;
------
 fftaps[1].d = !xorout0; -- for a XNOR feedback xorout0 must be negated
------
 taps[] = fftaps[].q;

END;
```