HDL Coding for PSEUDO-RANDOM Noise Generators

Inferring Virtex SRL macros results in extremely efficient Linear Feedback Shift Register implementations.

by Mike Gulotta, Field Application Engineer, Xilinx,
mike.gulotta@xilinx.com

Linear Feedback Shift Registers (LFSRs) are a fundamental function in applications such as pseudo-random noise (PN) generators, stream encryption, and error detection/correction. You can achieve extremely efficient LFSR implementations by using the Virtex Shift Register LUT (SRL). And, with today's Virtex-friendly synthesis tools, HDL code can be used to infer the SRL, thereby maintaining code portability.

PN Generators

PN generators are at the heart of every spread spectrum system, and are a good example for demonstrating how you can dramatically reduce FPGA utilization by exploiting the Virtex SRL. In a CDMA system, many PN generators are needed to distinguish channels, base stations, and handsets. Rake receivers, used in CDMA systems, consist of many copies of the same receiver, each called a finger, and each finger requires two PN generators, one for the "I" (In-phase) and one for the "Q" (Quadrature) channel.

Finding a way to improve the FPGA implementation efficiency of a circuit that is copied many times in a system (such as a PN generator), will obviously provide a huge savings.

Linear Feedback Shift Registers

Though the mathematics behind a PN code can be extremely complicated, the LFSR implementation can be relatively simple. A typical LFSR consists of a chain of registers and a modulo-2 adder (XOR gate). Predefined registers are "tapped" and fed to the XOR gate, and the XOR output is fed back to the first register in the chain, as shown in Figure 1. In a CDMA system, the predefined register taps are carefully determined to provided good auto correlation and cross correlation, and are often expressed as a polynomial such as, $P(x) = x^{17} + x^4 + 1$. An LFSR with n registers can sequence through $(2^n - 1)$ states. (See Xilinx XAPP 210 and XAPP 211 for additional information regarding LFSRs and SRLs).

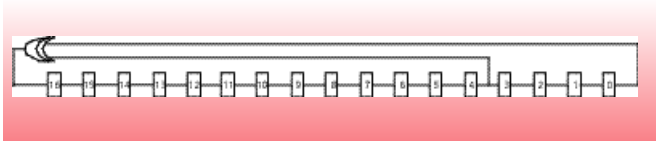

Figure 1 - A typical LFSR.

CDMA system requirements may require additional control of the basic LFSR. "Augmenting" the sequence, by adding an additional state, may be required to achieve 2^n states (instead of 2^n - 1), to maintain an even modulo count. Also, "puncturing" the sequence by periodically skipping a state may be required, if only a subset of the total 2^n states (such as 3 out of 4) are needed.

LFSR HDL Coding

The Virtex FPGA architecture is highly efficient for creating LFSRs. For example, the following code will infer a 64-bit shift register using Virtex SRLs rather than flip-flops (FFs).

VHDL	Verilog
process(clk)	Always @(posedge clk) begin
begin	Y <= {Y[62:0],INPUT};
if clk'event and clk='1' then	end
Y <= Y(62 downto 0) & INPUT;	
end if;	
end process;	

Figure 2 - HDL Code.

Using SRLs instead of FFs, this circuit will cost only one Configurable Logic Block (CLB) instead of 16. With such dramatic savings it is worth looking into ways to use SRLs whenever possible.

However, SRL registers cannot be loaded or read simultaneously, nor can they be asynchronously reset. In a PN generator application it may be necessary to jump out of sequence, which can be done by various techniques such as parallel loading the LFSR with a predetermined state. This can still be satisfied with the SRL by serially filling the LFSR with a predetermined state. To do this, a multiplexer is required in the LFSR feedback path allowing the loop to be broken while the predetermined state is shifted in, as shown in figure 3.

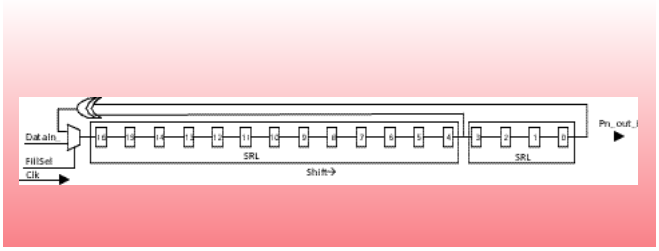

Figure 3 - PN generator.

The following verilog code implements an LFSR with several input controls that may be used to accommodate a PN generator application. The ShiftEn signal may be used to stall (augment) and/or puncture the sequence, and the FillSel and DataIn signals may be used to jump out of sequence. The `define compiler directive along with the Reset signal provide code portability by allowing the code to infer SRLs if targeting Virtex FPGAs, or to infer typical asynchronous reset FFs if targeting another technology. The number of taps are fixed, however the tap points and LFSR length are parameterized.

Conclusion

The Virtex architecture is very efficient for creating PN generators by using the Virtex Shift Register LUT (SRL). The SRL can also be used in many other applications such as pipeline balancers, filters, dividers, and waveform generators. In large systems, such as CDMA, the overall FPGA utilization can be reduced considerably by

```
/*************************************************************

The following is example code that implements an LFSR that can be used as
part of a pn generator.

The number of taps are fixed,however the tap points are parameratized.  The
LFSR length is also parameritized.  This code is not intended to be technology
specific.  When targeting Xilinx (Virtex) however, all the latest synthesis tools
(Leonardo, Synplify, and FPGA Express) will infer the Shift Register LUTS
(SRL16) resulting in a very efficient implementation.

Control signals have been provided to allow external circuitry to control such
things as filling,puncturing,stalling (augmentation),etc.  Only minimal simula-
tions have been run on this code as it is intended to be used for reference pur-
poses only.

A compiler directive can be used to steer the following code to infer typical FF's
(w/async resets),or, infer Virtex SLR16E elements.  Controlling the compiler
flow can be done by uncommenting the following line.  This can also be done
from a top level module.

*************************************************************/
// `define non_Virtex_device  // Comment out to infer Virtex SRL16s.
module pn_gen_lg_srl (clk,pn_out_i,ShiftEn,FillSel,DataIn_i,RESET);
parameter Width = 17;  // LFSR length (ie. number of storage elements)
// Parameratize l channel LFSR taps.
// l(x) = X**17 + X**4 + 1
parameter l_tap4 = 4;  // l channel LFSR,single tap.
//Ports
input   clk,DataIn_i,FillSel,ShiftEn,RESET;
output  pn_out_i;

// l channel ////////////////////
reg [l_tap4-1:0] srl1_i;
reg [Width-1:l_tap4] srl2_i;
wire  lfsr_in_i,par_out_i;

assign   pn_out_i = srl1_i[0];
assign   par_out_i = srl2_i[l_tap4] ^ srl1_i[0];
assign   lfsr_in_i = FillSel ? DataIn_i :par_out_i;

`ifdef non_Virtex_device // compiler directive, if defined,will infer asyn reset FF
always @(posedge clk or negedge RESET) begin
   if (!RESET) begin
      srl1_i <= 0;
      srl2_i <= 0;
   end
   else
`else  // compiler directive, if not defined,will infer SRL16.

always @(posedge clk)  begin

`endif

   if (ShiftEn) begin
      srl2_i <= {lfsr_in_i,srl2_i[Width-1:l_tap4+1]};
      srl1_i <= {srl2_i[l_tap4],srl1_i[l_tap4-1:1]};
   end
end

endmodule
```

When targeting Virtex,all the latest synthesis tools (Leonardo, Synplify, and FPGA Express) will infer the SRL16E resulting in a very efficient implementation. The compiler directive, used to steer the code to infer FFs or Virtex SLR16E elements, may not be supported by all synthesis tools. Controlling the compiler flow can be done by un-commenting the first line. (This can also be done from a top level module). Only minimal simulations have been run on this code as it is intended to be used for reference purposes only.

taking advantage of the SRL, which can lead to smaller, fewer, and less expensive parts. With only a basic understanding of the SRL, along with today's Virtex-friendly synthesis tools, these savings can be accomplished easily and without sacrificing code portability.