Linear Feedback Shift Register (LFSR) Counters

ECE 5440/6370
Spring 2010

Yuhua Chen
LFSR Counters

- Primer
  - An LFSR is a special type of counter
  - Counting Sequence is Pseudo Random
  - Typically has $2^n - 1$ states (primitive form)
  - Can be made to have $2^n$ states with "extended sequence" logic
LFSR Counters Implementation Styles

- **Up / Down Counters**
  - Down counters (start w/ 111… typically)
    - Use XOR gates for feedback terms
    - Primitive forms do not have an all zeros state
  - Up counters (start w/ 000… typically)
    - Use XNOR gates for feedback terms
    - Primitive forms do not have an all ones state

- **Flip-flop Feedback Construction**
  - One-to-many – One output to many inputs
  - Many-to-one - Many outputs to one input
Many to One Examples

Different taps sometimes possible

XOR vs. XNOR
LFSR Counters

- One-to-many is fastest, since it can be implemented in two levels of logic

4-bit LFSR Schematic
# XNOR Version Counting Sequence

<table>
<thead>
<tr>
<th>binary</th>
<th>hex</th>
<th>decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0001</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0010</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>0101</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>1010</td>
<td>A</td>
<td>4</td>
</tr>
<tr>
<td>0100</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>1001</td>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>0011</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>0110</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>1101</td>
<td>D</td>
<td>9</td>
</tr>
<tr>
<td>1011</td>
<td>B</td>
<td>10</td>
</tr>
<tr>
<td>0111</td>
<td>7</td>
<td>11</td>
</tr>
<tr>
<td>1111</td>
<td>F</td>
<td>12</td>
</tr>
<tr>
<td>1110</td>
<td>E</td>
<td>13</td>
</tr>
<tr>
<td>1100</td>
<td>C</td>
<td>14</td>
</tr>
<tr>
<td>1000</td>
<td>8</td>
<td>15</td>
</tr>
</tbody>
</table>
LFSR Counters

Comparison to other counter types

PROS:
- Requires very little logic to implement
- Even long counters are very efficient
  - Low gate count
  - High speed
- Easy to test for faults - typically only need 2*n clocks

CONS:
- Primitive forms must be initialized to valid state
- Some applications require binary count sequences
- Not easy to predict count sequence – tough to roll your own…
Fortunately, there’s any easy way...

- Use the LFSR Testbench Utility
- Careful… LSB is on the left
- Select the Verilog tab for code…
Recommended Settings

- When counting sequences are longer than 256 states, it will be cumbersome to use the tool to determine the terminal count (last count before the pattern repeats).
- Use **many-to-one** because patterns are more predictable.
- When feedback is NXOR, the terminal count will have the characteristic of MSB = 1, all other bits are zero.
- When feedback is XOR, the terminal count will have the characteristic of MSB = 0, all other bits are one.
- See examples on the next page…
Many-to-one XNOR Example
Many-to-one XOR Example
Decoding LFSR States

- To be useful, the LFSR flop outputs must be decoded
- Typically, we want to decode the terminal count (last counter state before the counting sequence repeats)
- However, when used for “timer” applications where the exact number of counts required is not $2^n$ or $2^n-1$, a shorter count sequence is needed
- The steps are:
  - Combinatorially (assign statement) decode the LFSR pattern corresponding to one count less than the desired length
  - Synchronously reset the counter to its initial state when decoded state is active
- For example, assume a simple, XNOR, many-to-one 8-bit LFSR and we want a counter sequence length of 240 counts.
- Using the LFSR test bench, we determine that the LFSR pattern after 239 counts is 9Ch
Simple, 8-Bit, Many-to-one NXOR Decoding Example
Decoding LFSR States

- Add the assign statement and reset logic to the module created by the LFSR testbench.
- Add synchronous reset logic.

```verilog
decoding_lfsr_states
module LFSR8_8E(reset_, clock, q, lfsr_to);
input clock, reset_;
output [7:0] q, lfsr_to;
reg [7:0] LFSR;
wire lfsr_to;
assign lfsr_to = (LFSR == 8'h9C);

always @(posedge clock or negedge reset_)
begin
  if (!reset_) LFSR[7:0] <= 8'h00;
  else begin
    if (lfsr_to) LFSR[7:0] <= 8'h00;
    else begin
      LFSR[7:1] <= LFSR[6:0];
    end
  end
end
assign q = LFSR;
endmodule
```
LFSR Counters

More LFSR Information