Linear feedback shift register design example

1 Introduction

This example shows the solution of the simplified version of the first homework. In this example the length of the shift register is 4 bits, it shifts right one bit in each clock cycle when enabled, and the feedback function is $SI = Q_0 \ XOR \ Q_2$ (Q0 is the LSb of the shift register).

2 Determining the maximum state change cycle

The linear feedback shift register generates a repeating sequence of numbers. The states of the shift register can be divided into cycles and each shift register state belongs to only one cycle. In our case there are 16 possible states ($2^4 = 16$).

When the number of states is small, the cycles can be determined by hand. Because the shift register shifts right, the last three bits of the current state will be the first three bits of the next state. The output of the feedback function ($SI$) can be calculated using the current state of the shift register and its result will be the MSb of the next state. When a cycle ends, the first state of the next cycle can be chosen from the not yet used states.

current state: $Q^i = \{Q_3, Q_2, Q_1, Q_0\} \rightarrow$ next state: $Q^{i+1} = \{SI, Q_3, Q_2, Q_1\}$

In case of large number of states, determining the cycles by hand is difficult. Therefore it is better to write a small program for this purpose. The following C program displays the state sequence of each cycle.

```c
#include <stdio.h>

#define SHR_LENGTH 4

char bin1[SHR_LENGTH + 1];
char bin2[SHR_LENGTH + 1];
char isStateUsed[1 << SHR_LENGTH];

char *dec2bin(unsigned long dec, char *bin)
{
    unsigned long i;
    bin[SHR_LENGTH] = 0;
    for (i = 0; i < SHR_LENGTH; i++)
    {
        bin[SHR_LENGTH - 1 - i] = '0' + (dec & 0x01);
        dec >>= 1;
    }
    return bin;
}
```
//Main function.
int main()
{
    unsigned long q0, q1, q2, q3, si, shr, next, length, i;
    unsigned long cycle = 0;

    //Initialize the isStateUsed[] array.
    for (i = 0; i < (1 << SHR_LENGTH); i++)
        isStateUsed[i] = 0;

    //Discover the cycles.
    while (1)
    {
        //Search for the first not yet used state.
        shr = 0;
        while (shr < (1 << SHR_LENGTH))
        {
            if (isStateUsed[shr] == 0)
                break;
            shr++;
        }

        //Check if all states have been used.
        if (shr == (1 << SHR_LENGTH))
            break;

        //Update the cycle number.
        cycle++;
        //Set the cycle length to zero.
        length = 0;

        printf("Cycle %d:
", cycle);
        printf("------------------------------
");

        //Discover the states of the current cycle.
        while (isStateUsed[shr] == 0)
        {
            //Calculate the value of each bit of the shift register.
            q0 = shr & 0x01;
            q1 = (shr >> 1) & 0x01;
            q2 = (shr >> 2) & 0x01;
            q3 = (shr >> 3) & 0x01;

            //Calculate the output value of the feedback function.
            si = q0 ^ q2;

            //Calculate the next state of the shift register (right shift!).
            next = (shr >> 1) | (si << 3);

            //Display the state transition.
            printf("%s (%02d) -> %s (%s), shr, dec2bin(next, bin2));

            //The current state has been used.
            isStateUsed[shr] = 1;
            //Set the next state of the shift register.
            shr = next;
            //Increment the length of the current cycle.
            length++;
        }

        //Display the length of the current cycle.
        printf("Length of cycle %d: %d state(s).
", cycle, length);
    }

    //Display the number of discovered cycles.
    printf("Total number of cycles: %d
", cycle);
    return 0;
}

The linear feedback shift register in this example has 4 cycles as shown in the table below. The first cycle has 1 state, the second and the third cycles have 6 states and the fourth cycle has 3 states. We can choose the second or the third cycle as our main cycle. Let’s choose the second cycle.
3 Designing the self-correcting network

The self-correcting network is a combinational logic which monitors the states of the shift register and resets the shift register into the main cycle (not necessarily in one step) if it is outside of the main cycle. In order to get the correction logic, we need a Karnaugh-map where the indices of the cycles are written into. The states of the main cycle are indicated by M and the states of the invalid cycles are indicated by 1, 3 and 4.

The correction logic must satisfy the following requirements:

- The states of the main cycle must not be covered by the minterms of the correction function, because the shift register must not be reset if it is in the main cycle.
- At least one state from each invalid cycle must be covered by the minterms of the correction function.
- The correction function should be as simple as possible.

One possible solution for the correction logic is \( \text{CORR} = Q_0\cdot Q_1 + /Q_0\cdot /Q_1\cdot /Q_2\cdot /Q_3 \).
4 Block diagram of the design

When the output of correction function is asserted, a state from the main cycle must be loaded into the shift register. Therefore the CORR signal is connected to the load input of the shift register and a constant value (main cycle state) is connected to the parallel data input of the shift register.

In order to test the design, it is necessary to be able to load any value into the shift register. This feature requires some extra logic. A multiplexer is needed to select between the main cycle state (constant) and the 4 bit data input, and its output is connected to the parallel data input of the shift register. The active high LOAD signal is connected to the select input of the multiplexer and ORed with the CORR signal. Therefore, when the LOAD input is high, the shift register loads the data from the 4 bit data input. When the LOAD signal is low, the correction network can load the main cycle state into the shift register.
5 Verilog implementation

5.1 Main module

module main(  
    input  wire       clk,       //Clock input.  
    input  wire       rst,       //Synchronous reset input.  
    input  wire       load,      //Synchronous load input.  
    input  wire [3:0] din,       //4 bit data input.  
    output wire       corr       //Output of the correction logic.  
);  
//Feedback function.  
wire shr_si = dout[0] ^ dout[2];  
//Correction logic.  
assign corr = (dout[0] & dout[1]) | (dout == 4'b0000);  
//Load signal for the shift register.  
wire shr_ld = load | corr;  
//Data to be loaded into the shift register.  
wire [3:0] shr_din = (load) ? din : 4'b0001;  
//Instantiating the shift register module.  
shr shr(  
    .clk(clk),       //Clock input.  
    .rst(rst),       //Synchronous reset input.  
    .ld(shr_ld),     //Synchronous load input.  
    .si(shr_si),     //1 bit serial data input.  
    .din(shr_din),   //4 bit parallel data input.  
    .q(dout)         //4 bit data output.  
);  
endmodule

5.2 Shift register module

module shr(  
    input  wire       clk,       //Clock input.  
    input  wire       rst,       //Synchronous reset input.  
    input  wire       ld,        //Synchronous load input.  
    input  wire       si,        //1 bit serial data input.  
    input  wire [3:0] din,       //4 bit parallel data input.  
    output reg [3:0] q           //4 bit data output.  
);  
always @(posedge clk)  
begin  
    if (rst)  
        q <= 4'b0000;       //Reset  
    else  
        if (ld)  
            q <= din;       //Load  
        else  
            q <= {si, q[3:1]};       //Right shift  
end  
endmodule