ASIC FPGA Prototyping - Random Number Generation

Jared Botte

1 Overview

In this lab, we will be implementing a Linear-Feedback Shift Register (LFSR), which will allow us to create a Pseudo Random Number Generator (PRNG).

2 Setup

Create a new directory and copy over the `makefile`.

```
mkdir random-number-generator
cd random-number-generator
mkdir source
cp $TOOLSDIR/makefile .
```

3 What is a Linear-Feedback Shift Register?

A Linear-Feedback Shift Register (LFSR) is a shift register in which the shifted in bit is determined by bits currently in the shift register. The specific bits in the shift register that are used to determine the shifted in bit are called taps. These taps are combined using XOR operations.

The selection of these taps is critical to create a LFSR with a maximal period (number of bits before repeating its sequence).

There are many videos/articles about LFSRs online. You are encouraged to do some external research and learn about them. Here is a good article, and here is a good video.

4 How do we use it to get a Pseudo Random Number?

To get a random number, we can use the last n bits of the LFSR as our random number. A 24-bit LFSR has a maximal period of 2^24 - 1 = 16,777,215. This means that our random numbers will not repeat in a sequence for over 16 Million cycles. That should be enough for any application we are going to implement.

It is important to note that the output is deterministic meaning that if you know the taps and a value, you can easily determine the next numbers. There are also mathematical ways to reverse the operation and derive the taps given enough values. For this reason, it's not a secure method of generating random numbers.

Lastly, as you probably can figure out, the starting value of our LFSR is very important. Since the order of the random numbers is a simple mathematical function, starting from the same value every time will give you the same random sequence of numbers. For this reason, we must provide a starting number, or seed, to randomize the starting point in the sequence each time. In our case, we can simply use a 24-bit counter which will be connected to our 50MHz clock. This means that we'll cycle through our possible values 3 times per second, and we'll use user input to determine the seed.

5 Implementing an LFSR

To implement a PRNG, we must first create our LFSR. This design will be a combination of the StP and PtS shift registers, as we need the ability to load in a value (the seed) and get the current output value.

Create a module `lfsr` with the following ports:

name	direction	width	description
clk	input	1 bit	clock
n_rst	input	1 bit	active low reset
shift_enable	input	1 bit	signal to enable shifting
load_enable	input	1 bit	signal to load in a seed
seed	input	24 bits	seed input values
value	output	24 bits	current value of the LFSR

Note: If using an XOR LFSR, the design may never be seeded to start with an all 0 value. Doing this will cause the LFSR to be stuck in an infinite loop and never produce anything except zeros.

The taps in this design should be bits 23, 22, 21, and 16. These four bits should be XORed together to generate the next MSB shifted in.

Also note that when using an LFSR, you won't want to use the whole value for your random number, you'll want to use a subset of the generated number to get your value. You'll also want to shift multiple times between values to add some extra randomness.

An example of this is Tetris on the NES. Your next Tetromino (Tetris game piece) is determined by an LFSR, which is shifted each frame. Therefore, the frame at which your piece is placed at the bottom of the board determines what your next piece will be.

6 Generating Pseudo-Random Numbers

Now we'll create a PRNG module that uses the LFSR to give us a random number.