Title of the Project

16-bit Linear-Feedback Shift Register

Group Elements

Bruno Aguiar nmec: 80177
José Moreira nmec: 79671

Brief Description of the Functionality

A 16-bit LFSR has many applications. One can be used as a counter, in cryptography, circuit testing, communications and can be used to generate pseudo-random sequences, so it can be helpful to generate an approximation of white noise.

The output stream, although, is deterministic, if we know the present state and the xor gates are also known, then we can predict the next state.

The user must assign an initial state different then all zeros. If not, the counter will remain "locked-up". He or she must also specify the LFSR period. The LFSR period is the number of LFSR cycles and the period needed to reach the maximal LFSR is 2^(16)-1, or 65535 cycles.

After that, the output stream is calculated in a software and hardware manner and the time required to do that is presented in the Vitis Serial Terminal, in microseconds.

The output stream is also displayed in the hexadecimal displays.

The implemented algorithm

The architecture to perform a 16-bit Fibonacci LFSR is composed of 16 registers and 3 XORs. The taps (bit positions that will affect the next state) are represented in the following polynomial x^16 + x^15 + x^13 + x^4 + 1, and in hexadecimal: 0xD008.

We decided to implement a coprocessor with the AXI Stream Interface, which means that we don't have to deal with the address bus, and we didn't use the DMA because the data bus is 32-bit length, so we can fit the 16-bit period and the 16-bit initial state in the same word sent to the coprocessor by the MicroBlaze processor.

Hardware Implementation

The following code shows how we processed the period and initial state of the LFSR as `s_data` being the initial state of the LFSR and `s_tmp` being the LFSR period:

```
elsif (S_AXIS_TVALID = '1') then
      if (s_ready = '1') then
            s_validOut <= '0';
            if (s_cnt = 0) then
                  s_data <= S_AXIS_TDATA(15 downto 0);
                  s_tmp <= to_integer(unsigned(S_AXIS_TDATA(31 downto
            16)));
                  s_cnt <= s_cnt + 1;
            end if;
       end if;
```

In the VHDL implementation of the algorithm, we decided to do the register shifting by making the 15 most significant bits of the LFSR state as the least significant bits and make the xor calculation as the most significant bit. This will calculate the next state of the stream and will stop when the counter `s_cnt` reaches the period value. In the next clock cycle the output stream is sent back to the processor:

```
if (s_cnt > 0) and (s_cnt <= s_tmp) then
      s_data <= (s_data(0) xor s_data(1) xor s_data(3) xor
s_data(12)) & s_data(15 downto 1);
      s_validOut <= '0';
      s_cnt <= s_cnt + 1;
end if;

if  (s_cnt > s_tmp)  then
      s_dataOut(s_data'range) <= s_data;
      s_cnt <= 0;
      s_validOut <= '1';
end if;
```

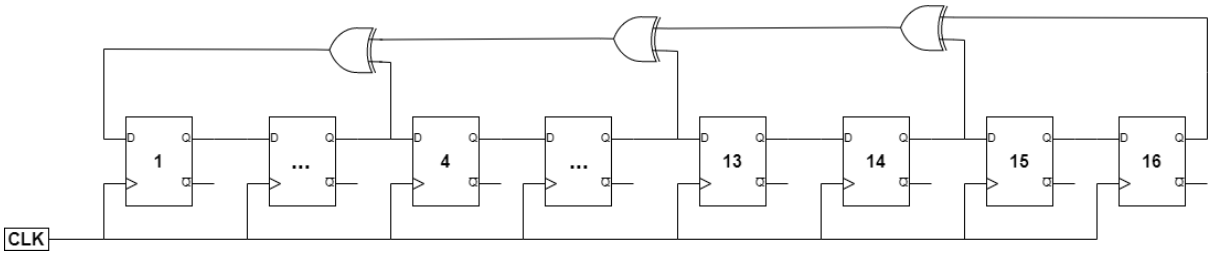
Software Implementation

The software implementation is pretty straightforward, as it's very similar to hardware level implementation:

```
unsigned short softwareLFSR(unsigned short lfsr, unsigned int
period)
{
      unsigned short bit;
      while(period != 0) {
            bit = ( (lfsr >> 0) ^ (lfsr >> 1) ^ (lfsr >> 3) ^ (lfsr
      >> 12) ) & 1;
             lfsr = (lfsr >> 1) | (bit << 15);
            period--;
      }
      return lfsr;
}
```

Coprocessor's Block Diagram

This represents the diagram of the 16-bit LFSR implemented in the project, as described earlier:



Percentage of the workload:

Bruno Aguiar: 50%
José Moreira 50%

Self-Evaluation:

We consider that this project went as expected. The software and the hardware implementation give the same output stream as expected, compared to the testbench simulation output stream. The difference in performance is also substantial, especially for large `period` values, as it is 75x faster in hardware rather than software, for the maximum period value (65535).
We could have added more features such as interruptions for when the coprocessor finished the computations and a DMA to send different initial states and periods, but, overall, the project delivers the basics for a LFSR implementation with a solid optimization in hardware (and in software as well).

NOTA: Dentro da pasta "vitis" está o workspace de vitis (que inclui a implementação em software), o Block Design do projeto e a implementação em hardware do Coprocessador.