Practicum 7

[Home] [Back]

Contents

Introduction

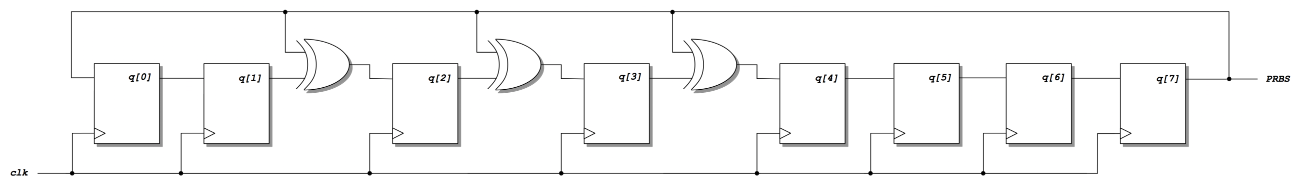
[Contents]

In this practicum we implement and test on real FPGA hardware a **Pseudo-Random Bit Sequence (PRBS) generator** using an **8-bit Linear-Feedback Shift-Register (LFSR)**.

The pseudo-random bit sequence generated by the circuit can be then used to display an **eye-diagram** at the oscilloscope in order to qualify the quality of terminations and transmission lines.

The RTL code of the digital block has been already discussed and simulated in:

https://github.com/lpacher/lae/tree/master/fpga/labs/lab9



Practicum aims

[Contents]

This practicum should exercise the following concepts:

- review the working principle of a PRBS generator using a LFSR
- implement and test the circuit on real FPGA hardware
- display an eye-diagram at the oscilloscope

Navigate to the practicum directory

[Contents]

As a first step, open a **terminal** window and change to the practicum directory:

```
% cd Desktop/lae/fpga/practicum/7_eye_diagram
```

List the content of the directory:

```
% ls -l
% ls -la
```

Setting up the work area

[Contents]

Copy from the `.solutions/` directory the main `Makefile` already prepared for you:

```
% cp .solutions/Makefile .
```

Create a new fresh working area:

```
% make area
```

Additionally, recursively copy from the `.solutions/` directory the following design sources and scripts already prepared for you:

```
% cp -r .solutions/rtl/      .
% cp -r .solutions/bench/    .
% cp -r .solutions/scripts/  .
% cp -r .solutions/xdc/      .
```

Compile the PLL IP core

[Contents]

A Phase-Locked Loop (PLL) IP core is used in RTL to **filter the jitter on the external input clock** fed to the core logic. The main **Xilinx Core Instance (XCI)** XML file containing the configuration of the IP has been already prepared for you.

Create a new `cores/PLL/` directory to contain IP sources that will be generated by the Vivado IP flow:

```
% mkdir cores/PLL
```

Copy from the `.solutions/cores/PLL/` directory the main XCI configuration file:

```
% cp .solutions/cores/PLL/PLL.xci  cores/PLL/
```

Finally, **compile the IP** using `make` as follows:

```
% make ip xci=cores/PLL/PLL.xci
```

At the end of the flow verify that all IP sources are in place:

```
% ls -l cores/PLL/
```

Review RTL sources (optional)

[Contents]

The proposed block is a shift-register with a special "feedback" that causes register outputs to assume binary values that "seems" random. A parameterized tick-counter can be used to slow-down the data processing, while the PLL is used to filter the input clock.

If needed, review in your text-editor application the main RTL module `rtl/LFSR.v` before continuing.

Simulate the design (optional)

[Contents]

Before mapping the RTL code into real FPGA hardware it is recommended to run a behavioral simulation of the proposed RTL code in order to verify that all RTL and IP sources are in place and to review the functionality of the digital block:

```
% make sim mode=gui
```

Implement the design on target FPGA

[Contents]

Inspect the content of the main **Xilinx Design Constraints (XDC)** file used to implement the design on real FPGA hardware already prepared for you:

```
% cat xdc/LFSR.xdc
```

If not already in place, copy the file from the `.solutions/` directory as follows:

```
% cp .solutions/xdc/LFSR.xdc  xdc/
```

Identify all pins that have been used to map top-level RTL ports.

> **QUESTION**
>
> On which board pin has been mapped the `PRBS` Verilog output port ?
>
> ____

Run the FPGA implementation flow in ***Non Project mode*** from the command line:

```
% make build
```

Once done, verify that the **bitstream file** has been properly generated:

```
% ls -l work/build/outputs/  | grep .bit
```

Install and debug the firmware

[Contents]

Connect the board to the USB port of your personal computer using a **USB A to micro USB cable**. Verify that the **POWER** status LED turns on. Once the board has been recognized by the operating system **upload the firmware** from the command line using:

```
% make install
```

Observe the pseudo-random bit sequence at the oscilloscope.

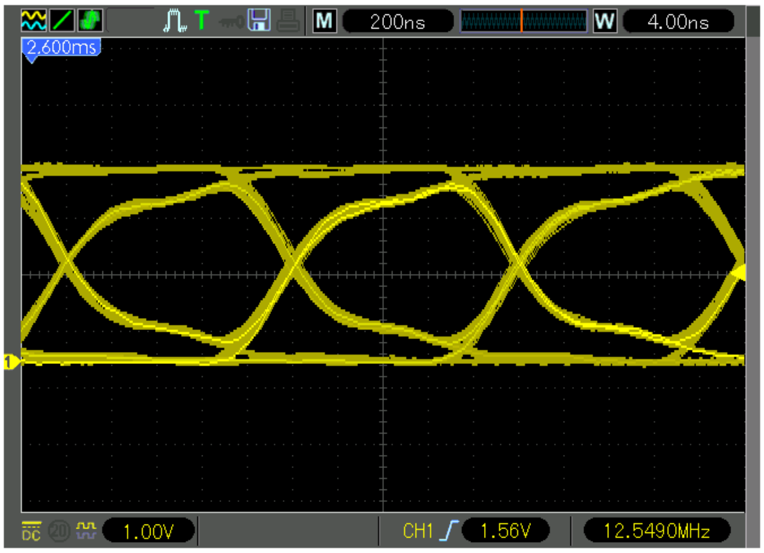
> **QUESTION**
>
> Which is the rate of the pseudo-random bit sequence ? Compare your answer with the behavioral simulation.
>
> ____

Display the eye diagram at the oscilloscope

[Contents]

The pseudo-random serial output generated by the circuit can be used to display a so called **eye-diagram**. This is a well-known technique extensively employed to qualify the **goodness and reliability of a data-transmission system**.

In order to display the eye diagram simply trigger on the pseudo-random serial output and set the oscilloscope acquisition mode to **infinite persistence**. The **overlay** between all low-to-high and high-to-low transitions generates the diagram. The quality of the data transmission is then quantified by the ***opening*** of the "eye", while **RMS and peak-to-peak jitter** can be measured by sampling and histogramming the arrival time.



Exercises

[Contents]

EXERCISE 1

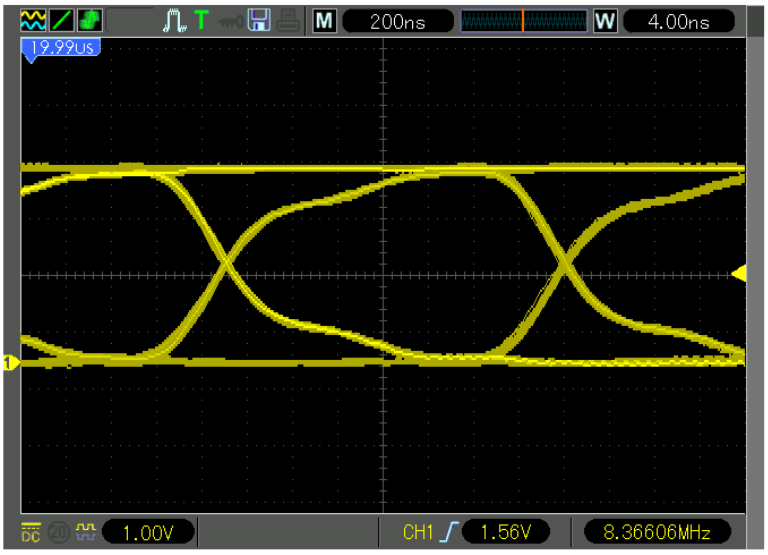
Modify the `rtl/LFSR.v` code in order to decrease the frequency of the "tick" generated by the tick-counter by changing the `MAX` value. As an exemple:

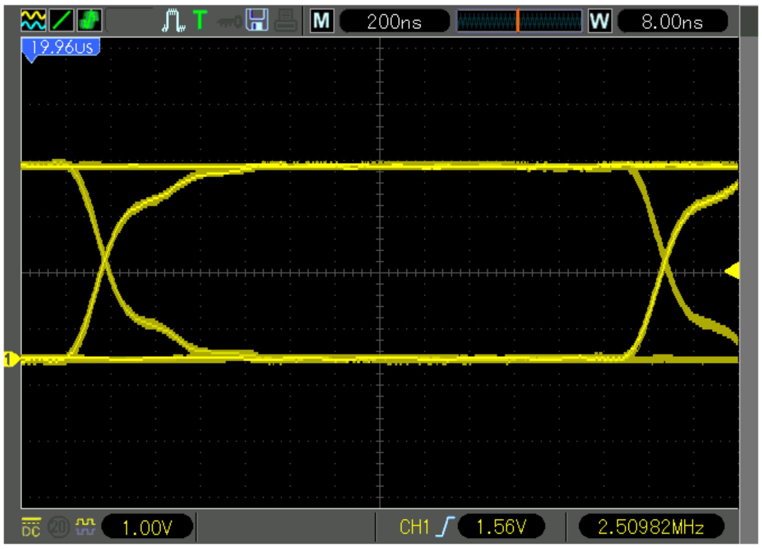
```
TickCounterRst #(.MAX(3)) TickCounter_inst (.clk(clk), .rst(~pll_locked), .tick(enable)) ;
```

Once done, save the file and try to re-run the flows from scratch up to FPGA programming with:

```
% make clean
% make build install
```

Display the new eye diagram after your changes.





QUESTION

What happens to the "eye" by lowering the frequency of the "tick" ?

____

EXERCISE 2

Modify the constraints file `xdc/LFSR.xdc` in order to map the `PRBS` output on a PMOD pin **without a 200 ohm series resistor**.

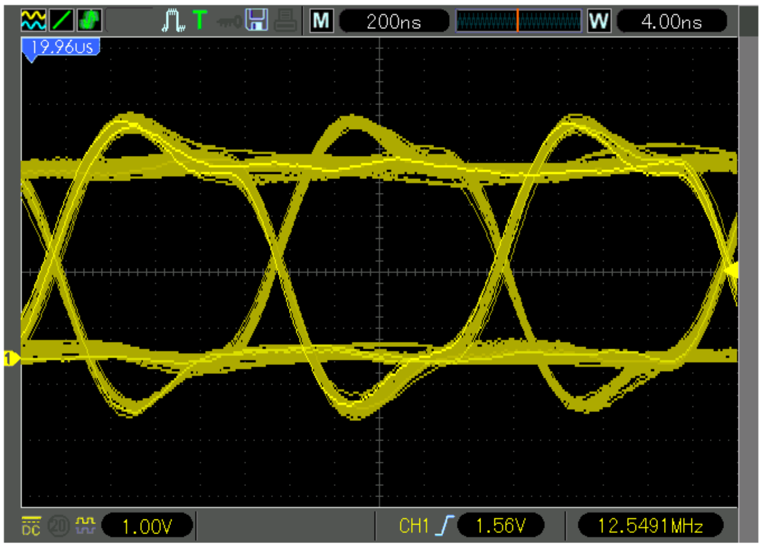
As an example:

```
% set_property -dict { PACKAGE_PIN E15  IOSTANDARD LVCMOS33 } [get_ports PRBS]
```

Once done, save the file and try to re-run the flows from scratch up to FPGA programming with:

```
% make clean
% make build install
```

Display the new eye diagram after your changes. Observe the additional "ringing" in the waveforms due to the lack of a series resistance.



EXERCISE 3

Change the default `SEED` value of the LFSR from `8'hFF` to `8'h00` in the `rtl/LFSR.v` module declaration:

```
module LFSR #(parameter [7:0] SEED = 8'h00) (

...
...

endmodule
```

Once done, save the file and try to re-run the flows from scratch up to FPGA programming with:

```
% make clean
% make build install
```

> QUESTION
>
> What happens to the pseudo-random bit sequence generated by the LFSR ?
>
> ____