Practicum 9

[Home] [Back]

Contents

- Introduction
- Practicum aims
- Navigate to the practicum directory
- Setting up the work area
- Compile the PLL IP core
- Inspect RTL sources
- Simulate the design (optional)
- Implement the design on target FPGA
- Install and debug the firmware
- Data analysis

Introduction

[Contents]

In this practicum we implement and test on real FPGA hardware a **Digitally-Controlled Delay Line (DCDL)** using a **Serial-In Parallel-Out (SIPO) shift register** and a multiplexer. The circuit can be then used to add a **4-bit programmable delay** on an input signal by changing the amount of delay through **slide-switches** available on the Arty A7 board. Optionally, a **PLL core** can be used to **multiply the input clock** in order to increase the resolution of the delay. An **auxiliary PWM generator** has been also included in the RTL code in order to test the circuit without the need of an external waveform generator.

Practicum aims

[Contents]

This practicum should exercise the following concepts:

- review how to describe a shift register in Verilog HDL
- implement and test on real FPGA hardware a programmable delay line
- use a PLL IP core for clock multiplication
- verify the linear characteristics delay value vs. MUX selection code

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/9_delay_line
```

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]

As usual a PLL IP core is used in RTL to **filter the jitter on the external input clock** fed to the core logic. Additionally, we use the PLL to **double the clock frequency** up to 200 MHz in order to increase the resolution of the fine-delay. Both 100 MHz and 200 MHz clocks are generated by the core and can be used in 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/
```

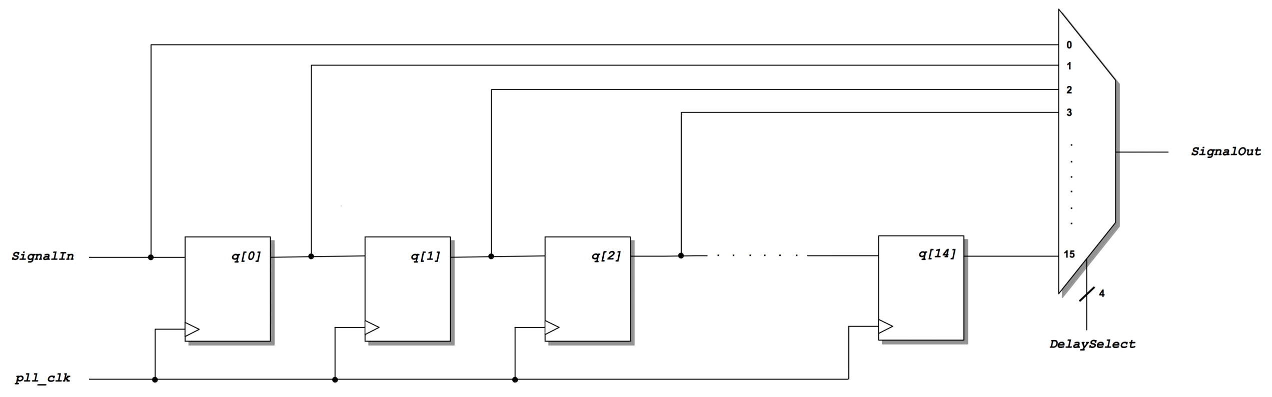
Inspect the **Verilog instantiation template** generated for you by Vivado in order to understand the port list of the IP that we are going to use in the RTL code:

```
% cat cores/PLL/PLL.veo
```

Inspect RTL sources

[Contents]

The circuit that you are going to implement and test on real hardware is the following:



An auxiliary 8-bit PWM generator (not shown in the schematic) has been also included in the RTL code in order to test the functionality of the programmable delay-line without the need of an external waveform generator.

Open with a text editor application the main `rtl/DelayLine.v` RTL code already prepared for you and try to understand the working principle of the proposed digital design.

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:

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

Debug your simulation results.

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/DelayLine.xdc
```

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

```
% cp .solutions/xdc/DelayLine.xdc  xdc/
```

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

> **QUESTION**
>
> On which board pin have been mapped `SignalIn` , `SignalOut` and `PWM` Verilog ports ?
>
> ____

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
```

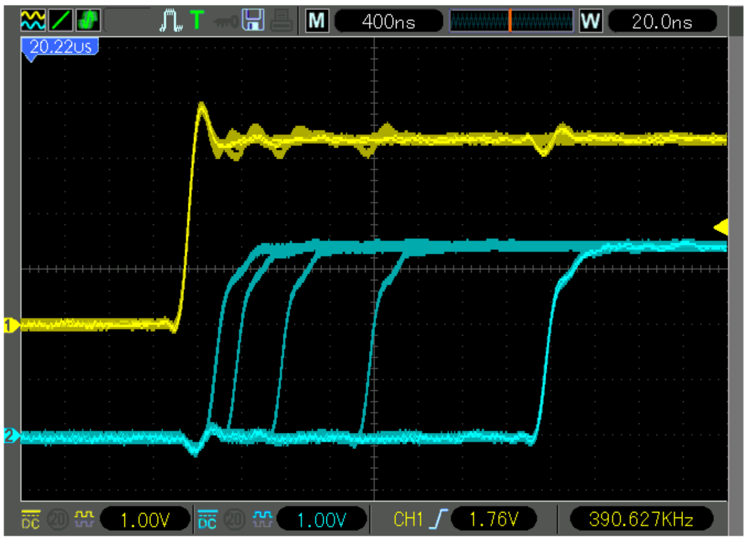
Install and debug the firmware

[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 `PWM` output signal at the oscilloscope. Then use a jumper wire to feed the PWM signal as input for the delay line and change the amount of delay using on-board slide-switches. Verify at the oscilloscope the expected functionality of the firmware.



Data analysis

[Contents]

Explore all possible MUX selection codes and derive the *delay vs. MUX code* characteristic. For each MUX code measure at the oscilloscope the delay inserted between input and output signals.

MUX code	delay
0	...
1	...
2	...
...	...
...	...

Use **ROOT** for your data analysis and verify the expected linearity of the characteristic with a fit.

Sample **ROOT un-named scripts** have been already prepared for you as a reference starting point for your analysis, you can copy them from the `.solutions/bin/` directory as follows:

```
% cp -r .solutions/bin/  .
```

Ask to the teacher if you are not confident in using the ROOT software.

Exercise

[Contents]

By default the PLL output clock used in the core logic runs at 100 MHz. However the PLL IP core has been compiled such that it can also **multiply the clock up to 200 MHz** in order to **increase the resolution of the delay**.

Modify the `rtl/DelayLine.v` code in order to run the core logic at 200 MHz instead of 100 MHz as follows:

```
PLL  PLL_inst ( .CLK_IN(clk), . CLK_OUT_100(UNCONNECTED), .CLK_OUT_200(pll_clk), .LOCKED(pll_locked) ) ;
```

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

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

Verify at the oscilloscope the functionality of the new firmware. If you have time re-measure the *delay vs. MUX code* characteristic and perform a linear fit on experimental data using ROOT.