Version for EDK 14.1 as of January 15, 2013

Goals

- To learn about the Xilinx Fast Simplex Link (FSL) Interface.
- To become familiar with the documentation you will need to refer to when designing your own systems with FSLs.

Requirements

- Module m01: Building a MicroBlaze System in XPS.
- It is suggested that Modules m02 and m03 also be completed before proceeding.

Preparation

- Read the FSL datasheet:

  `$XILINX_EDK/hw/XilinxProcessorIPLib/pcores/fsl_v20_v2_11_e/doc/fsl_v20.pdf`

- Read the FSL Application Note, XAPP529. If you are short on time, start reading at the middle of page 6 where they begin to describe the FSL interface in detail.

Background

The Fast Simplex Link (FSL) is a uni-directional point-to-point connection between two devices. It is an abstraction designed to look like a reliable, ordered communication channel that essentially wraps a FIFO with a standardized interface. The implementation is parameterized to suit the characteristics of the design, depending on whether the user needs it to be synchronous (both devices run on the same clock) or asynchronous (the master and slave run on different clocks), on the size, and on various other parameters that the user can specify.

As a point-to-point simplex link, a single FSL can connect only one master device to one slave device. The master writes into the FSL and the slave reads from it. Since we typically want to be able to communicate in both directions, FSLs are often used in pairs where each of the two connected devices is a master on one of the FSLs and a slave on the other.

FSLs can be used for pseudo-extending the instruction set of a MicroBlaze processor. As an example, instead of the processor having a built in square-root instruction like this

```
sqrt r2, r1
```

a co-processing core could be connected through two FSLs, so that

```
put r1, FSL0
```

writes the operand to the co-processor through the Master FSL connection 0, and

```
get r2, FSL0
```

reads the result back from the Slave FSL 0 coming from the co-processor.

Important Notes

- Note that you can only have one master and one slave on a single FSL.
- Note that one bidirectional (duplex) FSL link is in fact composed of one pair of FSLs. If the FSL link connects devices A and B, one of the FSLs has device A as the master and B as the slave. The other FSL has B as the master and A as the slave.
- Do not write to the FSL when the `Full` signal is high. Doing so will cause the FSL data to be corrupted. In the asynchronous FSL, doing so will overwrite the word in `FSL_S_Data` with the latest data word. This is not a concern when doing writes in C code, because the MicroBlaze functions take care of checking the FSL `Full` and `Exists` lines.

Creating the Hardware

1. Create a copy of the `lab1` folder and rename to `lab10`. This will be the working project directory for this lab.
2. Download the file `m10.zip` and unzip it in the directory containing the `lab10` folder you just created. This file contains an IDCT pcore that we will connect to MicroBlaze using the FSL and the source code for a test application. Go into your `lab10/code/` directory and delete `lab1.c`.
3. Return to your `lab10` folder and double-click on `system.xmp` to open the project.
4. In the Project Repository under the IP Catalog section, you should find the `xil_idct_v1_00_a` user core. Add it to the system. It should now appear in the System Assembly.
5. Add two FSL links which you can find under Bus within the IP Catalog menu.
6. In the Bus Interfaces tab, double click on `fsl_v20_0` and uncheck the Propagate Control Bit option. Do the same for `fsl_v20_1`.
7. Double click on `microblaze_0`. Click Next until you reach the page called Page 4 of 4 - PVR and Buses.
8. Set the Select Stream Interfaces to FSL and change the Number of Stream Links to 2.
9. Connect MicroBlaze as a master on `fsl_v20_0` (using MFSL0) and as a slave on `fsl_v20_1` (using SFSL1).
10. Connect the `xil_idct_0` user core as a master on `fsl_v20_1` (using MFSL) and as a slave on `fsl_v20_0` (using SFSL).
11. In the Ports view, expand the `xil_idct_0` pcore instance. Connect `Clk` to `clock_generator_0` and `Reset` to `net_gnd`.
12. Do the same for `fsl_v20_0` and `fsl_v20_1`. Leave all other connections untouched.

Creating the Bitstream and Software

13. Select Generate Bitsream under the Hardware tab.
14. When bitstream generation is complete, select Export hardware design to SDK... under the Project tab, then choose to Export and launch SDK.
15. When prompted to select a workspace, find the directory of your project (e.g. the lab10 folder) and select it.

16. When SDK launches, create a new Empty Application Xilinx C Project. Name the project fsl_lab10_0. Click Next.

17. Select Create a new Board Support Package project and name this is fsl_lab10_0_bsp. Click Finish.

18. Under fsl_lab10_0, add the system.c file under src.

19. Right click on fsl_lab10_0 and select Clean Project.

20. Once the file fsl_lab10_0.elf.elfcheck has been built, return to XPS.

21. Add the fsl_lab10_0.elf file to the project by selecting Select Elf File under the Project tab, as in lab1. Do this for both the Implementation Elf File and the Simulation Elf File.

Testing the Application

22. Update the Bitstream and download it to the board.

23. Run the project using SDK. Observe the output in the terminal window. You should see the results of a sample application which tests the IDCT operation on 8 sets of data. Read through the sample code (`system.c`) to see how the software interface to the FSL links works. There is a small mistake in the program which is causing the read values to be all-zeros. It is up to you to find the mistake and correct it (*hint: think about how the FSLs are connected*).

24. After your system runs correctly, try to use the blocking read and write FSL operations instead of the non-blocking ones (this is trivial).