Notes about the implementation

Monideep Bora

January 24, 2021

1 Introduction

- This implementation is based on the theory found in [1], [2] and [3]
- The FIFO being only 32 locations deep, it has been implemented using registers. FiFO's of bigger sizes can be implemented using intrinsic memory elements embedded inside the FPGA. However, these implementations are vendor-specific. The width and depth are made flexible using generics.

2 Some special features

- **Synchronous Reset**: Conventionally, the data in memory is not to be initialized during run-time. However, a synchronous reset signal has been provided to reset the indices and counter.
- **Almost Full/Empty flags**: The FIFO is equipped with *full* and *empty* flags which can be read by a upstream entity to check if read-write operations are within limits. In addition, as per the US Patent [4], *almost-full* and *almost-empty* flags are added which get asserted when a certain predefined threshold using generics. These help in saving cycles, not having to check the flags after each read or write. For example, in the current implementation, the *almost-full* flag is set after the 24$^{\text{th}}$ location is written to. After that, the rest of the 8 locations until 32 can be written to without checking the *full* flag by the upstream entity. The *almost-empty* flag works similarly. The thresholds can be set using generics.
- **Diagnostic Testbench**: The testbench writes an incrementing value starting from 1 and going to 32, to all the 32 locations of the FIFO. AAfter that, it reads back the values starting from the index 0 to 32, verifying them at each location using *assert* constructs. There values read or written and their corresponding locations are printed to console in verbose.
- **Registered full and empty flags**: The *full* and *empty* flags serve as *output* from the FIFO entity to be interfaced to an upstream processor entity. Since they need to be read inside the FIFO entity, therefore, registered versions of them are used which are later interfaced to the output ports. A possible workaround would be to make the flags as *inout*, rather than *output*.

3 Execution

3.1 Using Modelsim

The *simulation.tcl* script can be used to run the testbench for the FIFO implementation. The waveforms are loaded in the *wave.do* file. If *Modelsim* is in the curent path, the below command can be executed from the root project directory.

```
vsim -do simulation.tcl
```

3.2 Using ghdl

The *Makefile* found in the root directory runs an instance of ghdl (if installed and in $PATH), analyzes, elaborates and exetues the design in the console. The below command can be run:

```
make all
```

References

[1] P. J. Ashenden, *Digital Design: An Embedded Systems Approach Using VHDL*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2008, pp. 263–264, ISBN: 0123695287.

[2] D. M. Harris and S. L. Harris, "Chapter 5 - digital building blocks," in *Digital Design and Computer Architecture*, D. M. Harris and S. L. Harris, Eds., Burlington: Morgan Kaufmann, 2007, pp. 233–286, ISBN: 978-0-12-370497-9. DOI: https://doi.org/10.1016/B978-012370497-9/50006-8.

[3] *Nandland:vhdl register based fifo*, https://www.nandland.com/vhdl/modules/module-fifo-regs-with-flags.html, Accessed: January 24, 2021.

[4] G. A. Kreifels, *Us patent us4891788a: Fifo with almost full/almost empty flag*, https://patents.google.com/patent/US4891788A/en, 1987.