# EECS 151/251A FPGA Lab
# Lab 6: Multi-bit Clock-Crossing, FIFOs, UART Piano, Project Intro

Prof. Elad Alon
TAs: Vighnesh Iyer, Bob Zhou
Department of Electrical Engineering and Computer Sciences
College of Engineering, University of California, Berkeley

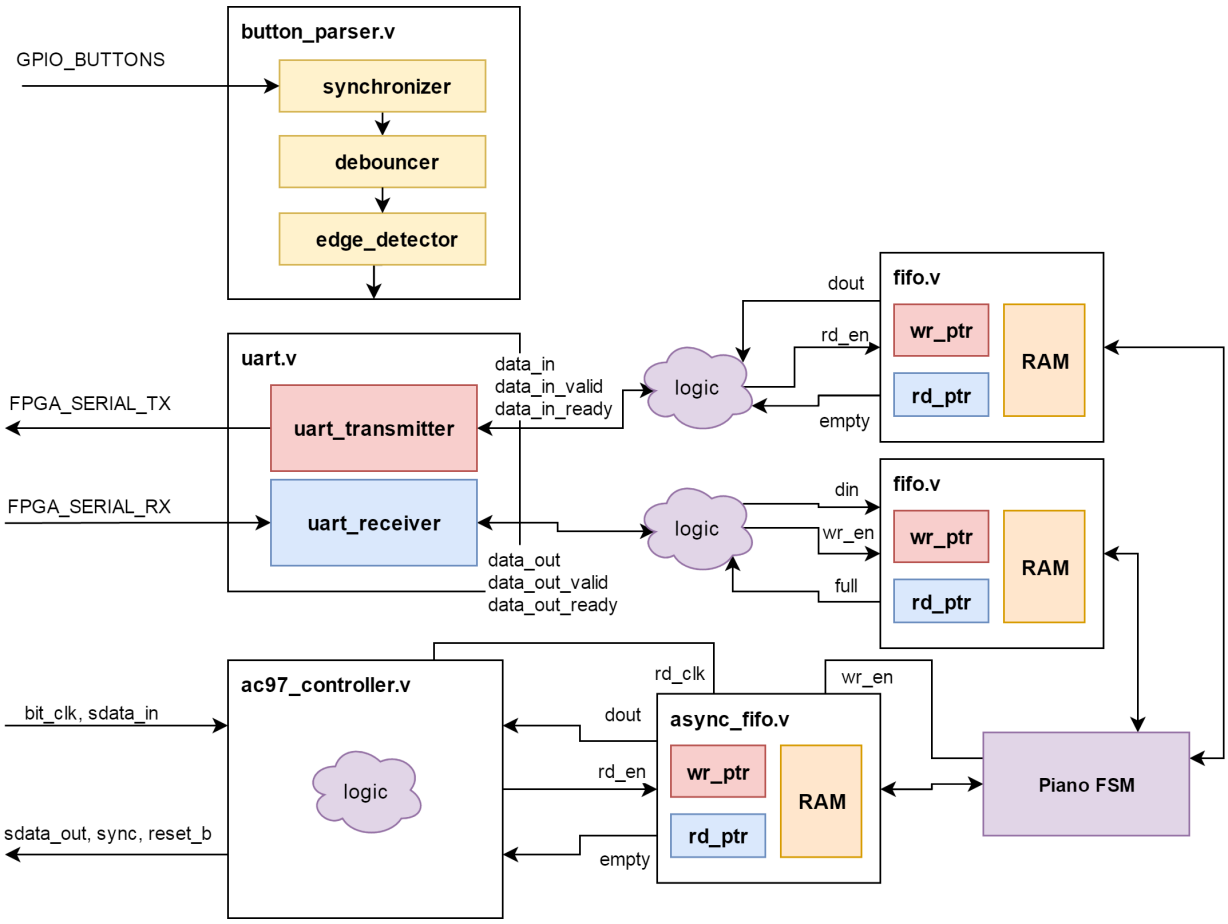
## Contents



## 1 Intro

In this lab, you will integrate the components you created in labs 4 and 5 (UART and AC97 controller). You will begin by building 2 varieties of FIFOs, asynchronous and synchronous, and verifying their functionality using a block-level testbench. You will then modify your AC97 controller to use a FIFO as its PCM data source. Finally, you will create some logic that integrates all these components to form a 'piano'.

Here is an overview of the entire system in `ml505top` we are going to build. You may find it useful to refer to this block diagram while doing this lab.




In this lab, you will be building the FIFOs, modifying your AC97 controller to use one, and designing the piano FSM. You will then construct a system-level testbench to verify functionality in simulation.

## 2 Building a Synchronous FIFO

A FIFO (first in, first out) data buffer is a circuit that has two interfaces: a read side and a write side. The FIFO we will build in this section will have both the read and write side clocked by the same clock; this circuit is known as a synchronous FIFO.

### 2.1 FIFO Functionality

A FIFO is implemented with a circular buffer (2D reg) and two pointers: a read pointer and a write pointer. These pointers address the buffer inside the FIFO, and they indicate where the next read or write operation should be performed. When the FIFO is reset, these pointers are set to the same value.

When a write to the FIFO is performed, the write pointer increments and the data provided to the FIFO is written to the buffer. When a read from the FIFO is performed, the read pointer increments, and the data present at the read pointer's location is sent out of the FIFO.

A comparison between the values of the read and write pointers indicate whether the FIFO is full or empty. You can choose to implement this logic as you please. The `Electronics` section of the FIFO Wikipedia article will likely aid you in creating your FIFO.

Here is a general diagram of the FIFO you should create from page 103 of the Xilinx FIFO IP Manual.

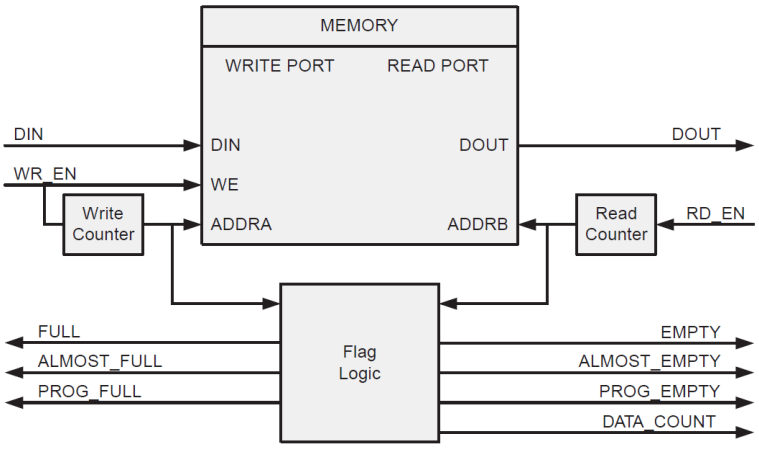



*Figure 5-4:* **Functional Implementation of a Common Clock FIFO using Block RAM or Distributed RAM**

The interface of our FIFO will contain a subset of the signals enumerated in the diagram above.

## 2.2 FIFO Interface

Take a look at the FIFO skeleton in `labs_sp17/lab6/src/fifo.v`.

Our FIFO is parameterized by these parameters:

- `data_width` - This parameter represents the number of bits per entry in the FIFO.
- `fifo_depth` - This parameter represents the number of entries in the FIFO.
- `addr_width` - This parameter is automatically filled by the log2 macro to be the number of bits for your read and write pointers.

The common FIFO signals are:

- `clk` - Clock used for both read and write interfaces of the FIFO.
- `rst` - Reset synchronous to the clock; should cause the read and write pointers to be reset.

The FIFO write interface consists of three signals:

- `wr_en` - When this signal is high, on the rising edge of the clock, the data on `din` will be written to the FIFO.
- `[data_width-1:0] din` - The data to be written to the FIFO should be present on this net.
- `full` - When this signal is high, it indicates that the FIFO is full.

The FIFO read interface consists of three signals:

- `rd_en` - When this signal is high, on the rising edge of the clock, the FIFO should present the data indexed by the write pointer on `dout`
- `[data_width-1:0] dout` - The data that was read from the FIFO after the rising edge on which `rd_en` was asserted
- `empty` - When this signal is high, it indicates that the FIFO is empty.

### FIFO Timing

The FIFO that you design should conform to the specs above. To further, clarify here are the read and write timing diagrams from the Xilinx FIFO IP Manual. These diagrams can be found on pages 105 and 107. Your FIFO should behave similarly.

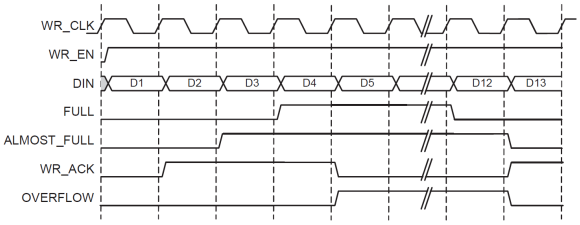



*Figure 5-6:* **Write Operation for a FIFO with Independent Clocks**

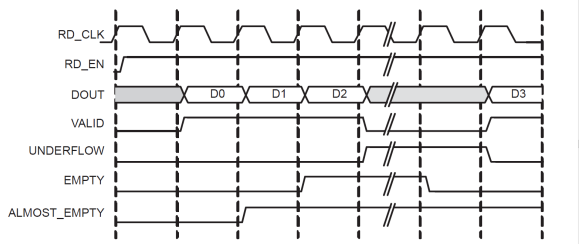



*Figure 5-7:* **Standard Read Operation for a FIFO with Independent Clocks**

Your FIFO doesn't need to support the `ALMOST_FULL`, `WR_ACK`, or `OVERFLOW` signals on the write interface and it doesn't need to support the `VALID`, `UNDERFLOW`, or `ALMOST_EMPTY` signals on the read interface.

### FIFO Testing

We have provided a testbench for your synchronous FIFO which can be found in `hardware/src/fifo_testbench.v` This testbench can test either the synchronous or the asynchronous FIFO you will create later in the project. To change which DUT is tested, comment out or reenable the defines at the top of the testbench (`SYNC_FIFO_TEST`, `ASYNC_FIFO_TEST`).

The testbench we have provided performs the following test sequence, which you should understand well.

1. Checks initial conditions after reset (FIFO not full and is empty)
2. Generates random data which will be used for testing

3. Pushes the data into the FIFO, and checks at every step that the FIFO is no longer empty
4. When the last piece of data has been pushed into the FIFO, it checks that the FIFO is not empty and is full
5. Verifies that cycling the clock and trying to overflow the FIFO doesn't cause any corruption of data or corruption of the full and empty flags
6. Reads the data from the FIFO, and checks at every step that the FIFO is no longer full
7. When the last piece of data has been read from the FIFO, it checks that the FIFO is not full and is empty
8. Verifies that cycling the clock and trying to underflow the FIFO doesn't cause any corruption of data or corruption of the full and empty flags
9. Checks that the data read from the FIFO matches the data that was originally written to the FIFO
10. Prints out test debug info

This testbench tests one particular way of interfacing with the FIFO. Of course, it is not comprehensive, and there are conditions and access patterns it does not test. We HIGHLY recommend adding some more tests to this testbench to verify your FIFO performs as expected. Here are a few tests to try:

- Several times in a row, write to, then read from the FIFO with no clock cycle delays. This will test the FIFO in a way that it's likely to be used when buffering user I/O.
- Try writing and reading from the FIFO on the same cycle. This will require you to use fork/join to run two threads in parallel. Make sure that no data gets corrupted.

# 3 Conclusion + Checkoff

You are done with lab 6! Please write down any and all feedback and criticism of this lab and share it with the TA. This is a brand new lab and we welcome everyone's input so that it can be improved.

## 3.1 Checkoff Tasks