Skid Buffer

Sai Mohan Kilambi

October 20, 2023

Contents

1 Introduction

This design document covers the concept of a skid buffer and a pipeline skid buffer. Skid-buffers have many names: Carloni buffer, Elastic buffer. The following description is heavily borrowed from [1]. Infact its verbatim. I have only added more description and the thought process.

2 Theory of Operation

Valid-ready based handshaking is a very popular way of communicating between two blocks. In this style of communication, the downstream block accepts new data only when it is ready to accept data and the upstream block has data to provide. Only when valid and ready signals are both high does an actual transfer happen. However in big chips, the two blocks can be very far apart and this can become the critical path.

In such cases, one straight forward way to be to pipeline all the valid, ready and data signals. But this creates latency. Lets think about this. Lets says we have 2-stage pipelining between the sender and receiver. If receiver block pulls its ready low, then it will will take 2 clock cycles for the sender to realize that the ready has gone low. The receiver needs to know of this pipelining to account for two data samples that will still show up at the receiver. Now when the receiver pulls its ready back high, it can immediately start absorbing data from the buffer but it will take 2 cycles for the sender to know that the ready is back high. After that it will take another 2 cycles for data to come to the receiver.

This in itself is ok if a large block of data is transferred when a valid transaction happens. But the problem is that the receiver now needs to be aware of how much pipelining exists between the sender and receiver and adjust the buffer size accordingly. This is not a very smart way of pipelining handshaking and associated data signals.

Also note that data needs to be written into the register and read from the register in the same clock cycle. Now if the receiver puts it ready down, then sender should not send a new data sample in the same clock cycle. But this will create a combinational chain

The solution to this is a skid buffer. The idea behind a skid buffer is this.

1. When the receiver is ready and sender has data to send data, the skid buffer should just be able to pass data through.

2. If the receiver puts its ready down, then the data from the sender will skid into the buffer inside the skid-buffer.

3. Here there are two options:

   The sender stops sending data until the receiver comes back online by setting its ready high.

   There is a circular buffer mode. In which the new data from sender shifts into the buffer and old data from the buffer moves into the data out register. Of course, whatever was there in the data out register is replaced. So for the input side there is nothing stalling it.

We can divide the operation of the skid buffer into two parts.

1. Data path portion.

2. Control portion. This will have the state-machine.

2.1 Datapath Section

In the datapath we need to think of two things.

1. Under what conditions should the skid buffer accept an input. That is to say under what conditions does the "i_ready" signal go high?

2. Under what conditions should the skid buffer present a valid output? That is to say under what conditions does "o_valid" go high?

And why are we specifically interested in these two signals? Because from the skid-buffer's point of view, these are the two control signals under its control.

So when would the "i_ready" signal be high?

1. When the output register is not holding any previous sample that it has not moved out.

2. When there is a sample in the output buffer but the internal buffer is empty so we can queue.

3. In the "Circular Buffer Mode" (CBM), input is always ready.

From the below state-machine description you will find that "i_ready" will be high whenever the state is **NOT** FULL or we are in the CBM mode.

When would "o_valid" be high?

1. Whenever you are not in an empty state. This would mean that there is some thing in the output buffer and/or in the internal buffer.

So from the state point of view this is high when state is **NOT** empty.

2.2 Control Section

So for the control section we will need to mark some states that define the state of the output buffer and the internal buffer.

State definitions:

1. EMPTY: No data inside the output buffer or the internal buffer.

2. LOADED: Data in the output buffer but not in the internal buffer.

3. FULL: Data present in both the output buffer and internal buffer. So either skid-buffer cannot accept anymore data until one of the buffers empties out or in CBM mode, we keep pushing data out of the output buffer and move internal buffer data to the output. This is irrespective of the fact that the downstream block can accept data or not. When it cannot, we are obviously losing data.

So now lets talk about the transitions.

1. Initial state is always EMPTY. This is when "i_ready" is high.

References

[1] Charles Eric LaForest. *Skid Buffer*. URL: http://fpgacpu.ca/fpga/Pipeline_Skid_Buffer.html.