Using the Valid-Ready pipeline protocol

Rev. 1.1

Introduction

In order to facilitate the flow of data in a pipeline, many of our IP cores use a simple valid-ready pipeline protocol. The protocol is fully synchronous, and makes use of 4 basic signals: c*lk*, *data*, v*alid* and r*eady*. The signal c*lk* is the system clock, d*ata* is the data to be transferred and *valid/ready* perform a simple handshake.

A module in a pipeline may have any number of valid-ready interfaces, but typically it will have two - an upstream interface and a downstream interface. The IP core 'PIPELINE REGISTER' is an example of a module with two valid-ready interfaces.

The fundamental rule of the pipeline protocol is as follows:

"Data is transferred at a pipeline interface on a rising clock-edge when valid and ready are both active high"

This is the basic rule that must be adhered to for all elements in the pipeline. Even with this simple rule in place, there are many types of scenarios in which data can flow and be stalled. The next sections attempt to address some of these situations in more detail.

Valid-Ready Signalling

Normally, a module in a pipeline will have at least one input and one output interface. In this table, a module with two such interfaces is described: an upstream interface (prefixed 'datain') and a downstream interface (prefixed 'dataout').

Signal Name	***I/O***	***Description***	***Active state***
clk	in	Synchronous system clock	rising-edge
datain	in	Input data word	data
datain_val	in	Indicates that the module upstream has valid data to transfer	high
datain_rdy	out	Indicates that the module is ready to accept data	high
dataout	out	Output data word	data
dataout_val	out	Indicates that the module has valid data at it's output	high
dataout_rdy	in	Indicates that the module downstream is ready to accept data	high

Valid-Ready using FIFO Naming Conventions

Another way of looking at the valid-ready protocol is in terms of common FIFO naming conventions. The signals *datain* and *dataout* are equivalent to the data paths into and out of the FIFO. The signal *datain_val* would be the *WRITE* strobe of the FIFO. Likewise, *dataout_val* would be an inverted version of the *EMPTY* flag. The *READ* strobe of the FIFO is equivalent to *dataout_rdy* while *datain_rdy* would be an inverted version of the *FULL* flag. Figure 1 below shows this connectivity in more detail.

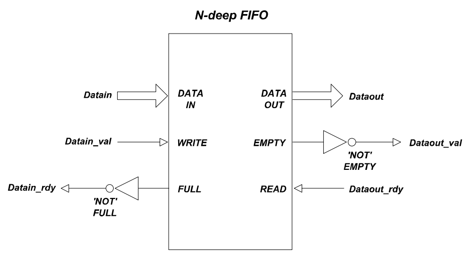

Figure 1: Valid-Ready signalling in terms of FIFO nomenclature

Valid-Ready Functional Timing Examples

Ready-before-Valid Signalling

Ready-before-valid signalling (Figure 2) occurs when an interface asserts ready high before there is valid data available. This would occur when a pipeline element has the capacity to absorb data before any data arrives. An example of this would be a FIFO or a buffer with a number of empty pipeline stages available. Generally it's good design practise to use ready-before-valid signalling (if the architecture permits) as it ensures maximum throughput down a pipeline. If ready-before-valid signalling is used throughout a pipeline then the pipeline will be self flushing.

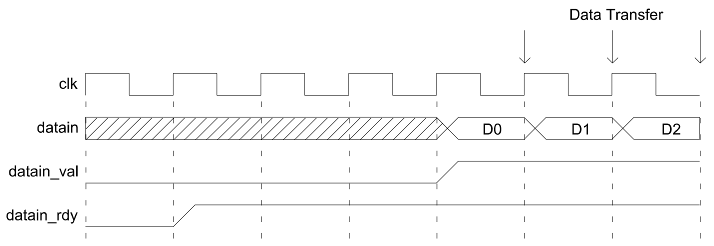

Figure 2: Ready-before-Valid signalling

Valid-before-Ready Signalling

With valid-before-ready signalling (Figure 3), a pipeline element asserts the valid signal first before the downstream element responds with a ready signal. In some designs, it may be necessary to use this type of signalling. An example would be an arbitration between various data paths where the selection of a particular data path is dependent on the type of data (E.g. an MPEG transport stream). In this case, a particular data path may not be routed until there's valid data at it's input.

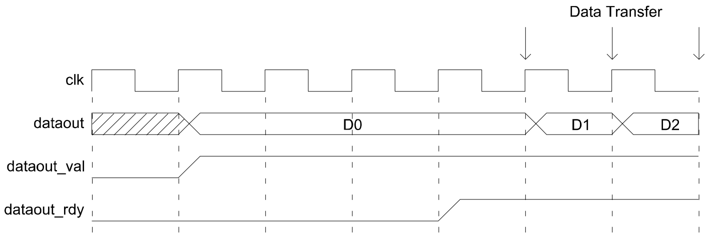

Figure 3: Valid-before-Ready signalling

Valid-Ready protocol 'Stalemate' situations

In some scenarios, a 'stalemate' situation may arise between elements in a pipeline using the valid-ready protocol. This occurs when the upstream element uses ready-before-valid and the downstream interface uses valid-before-ready. In this situation, the upstream interface will wait indefinitely for a ready before asserting it's valid and likewise, the downstream will wait indefinitely for valid before asserting it's ready. The result is a stalemate with the pipeline locked.

Care must be taken to match valid-ready interfaces correctly. If possible, modules within a pipeline should be designed to assert both ready and valid without 'snooping' the condition of the upstream or downstream interfaces (i.e. adopt a valid-before-ready *and* ready-before-valid approach). In this way, data throughput is maximized and stalemate situations are avoided.

Stalled Pipeline

Figure 4 demonstrates a situation where the data path is initially stalled waiting for a ready signal. The ready is asserted for one clock-cycle resulting in a single data transfer. The pipeline is subsequently stalled, waiting for a ready signal. The upstream pipeline element is using valid-before-ready signalling.

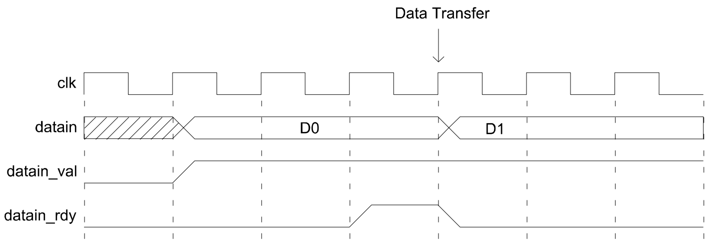

Figure 4: Full pipeline with stalling

Empty Pipeline

Figure 5 demonstrates the transfer of a single data item down the pipeline. After data transfer, the pipeline becomes empty. The downstream element is using ready-before-valid signalling and asserts ready high for the duration.

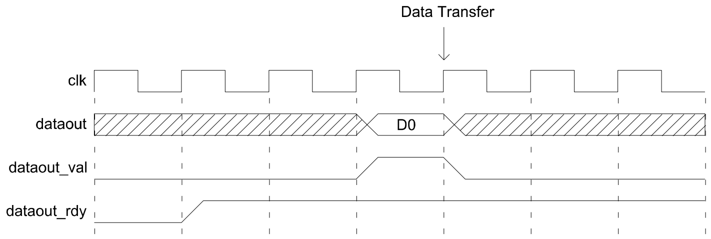

Figure 5: Pipeline empty

Valid-Ready Connectivity

The following block diagrams give examples of connectivity between pipeline elements that use the valid-ready protocol.

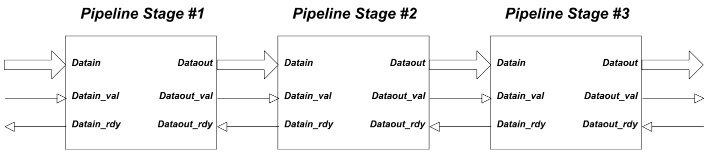

Figure 6: Pipeline elements connected in series

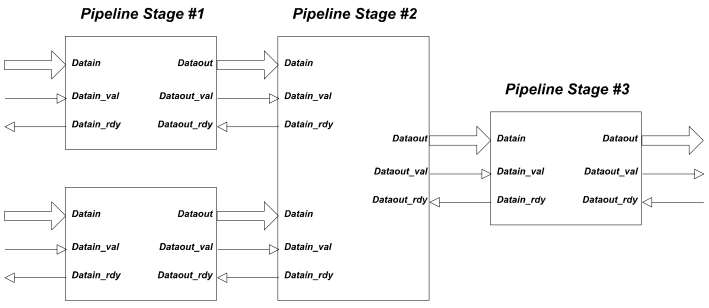

Figure 7: Pipeline 2:1 Mux

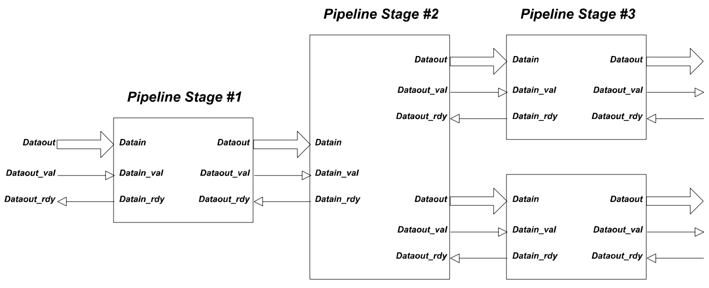

Figure 8: Pipeline 1:2 De-mux

Conclusion

The valid-ready pipeline protocol is a versatile protocol used to manage the data-flow in a pipeline. The protocol is used by many of the IP Cores within the ZIPcores library. It uses four basic signals: a synchronous clock, data, valid and ready. With these four signals, any number of pipeline elements may be connected together and in various configurations.

We have seen that there are different methods of implementing the valid-ready handshake signals such as the valid-before-ready and ready-before-valid implementations. Careful design of the interfaces between pipeline elements will ensure maximum data throughput and will avoid potential pipeline stalemate situations.

Revision History

Revision	***Change description***	***Date***
1.0	Initial revision	30/06/08
1.1	Added FIFO naming convention example	10/06/09