CPU-FPGA communication protocol

1. Introduction

The following is a description of the proposed communication protocol between the CPU and the FPGA. The main design goals were the simplicity of the hardware (in terms of area) and the performance. For achieving these two objectives, we divided part of the effort between the hardware implementation and the software tools.

With the present document, we will mainly focus on the hardware side, while briefly detailing what software functionalities are in in turn needed.

The document is organized as follow: section 2 illustrates the interface of the internal buffer and describes the overall functionalities. With section 3, we provide an overall description of the IP Manger interfaces. Section 4 is devoted to the pins configuration and finally sections 5, 6 describe CPU transactions and interrupt requests respectively.

2. Buffer Interface

In order to simplify the design, we decided to implement (by means of a synthesizable VHDL design) the buffer as a full custom dual-port register file. Figure 1 shows the interface. `PORT_0` is dedicated for the communication with CPU, whereas the IP manager controls `PORT_1`. It is worth nothing that there is an additional output port, called `ROW_0`. The purpose of this port is to enable an easy access to the row 0 by the IP manager, which (as discussed in details in section 4) uses this row to perform the correct route for allowing interaction with the selected IP. This port reflects any change in the row 0 of the buffer to whichever device is connected to (in this case, the IP Manger).

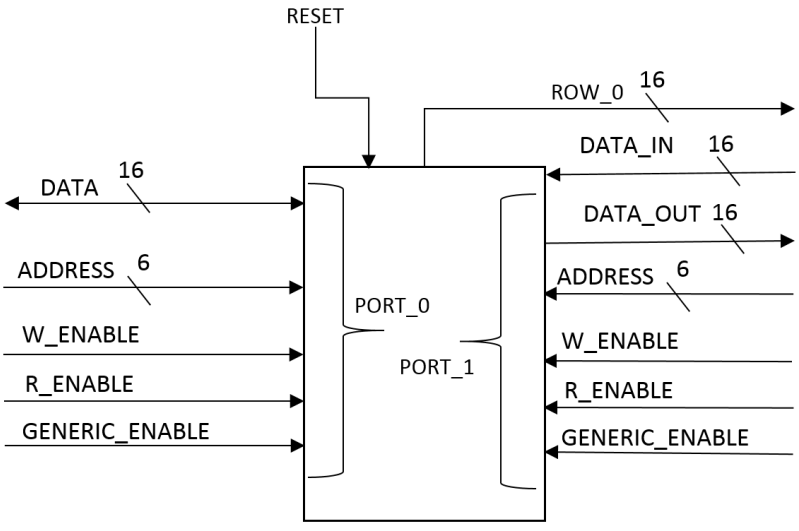


Figure 1 - Buffer interface

Another important aspect is that, as it happens with real register files, the module is completely asynchronous. Indeed, the module is completely asynchronous, as a clocked register file would degrade the overall performance. On the other hand, the IP manager itself is synchronous and is fed with a clock signal from the CPU. In order to read/write on one of the two ports, the generic enable signal must be asserted as well as the required control signal (`W_ENABLE/R_ENABLE`). Then the correct values must be placed on data and address lines. In details, each signal (for both `PORT_0` and `PORT_1`) has the following function:

- `DATA`: input/output port, used for transferring data to/from the buffer.
- `ADDRESS`: input port, used for selecting the right row within the buffer to write/read data.
- `W_ENABLE`: input port, must be asserted in order to enable write operation.
- `R_ENABLE`: input port, must be asserted in order to enable a read operation.
- `GENERIC_ENABLE`: input port, must be asserted in order to start any operation on that port.
- `RESET`: input port, it brings the buffer to the reset state.
- `ROW_0`: output port, it reflects any change in the row 0 of the buffer.

Allowed configurations for each port are shown in table 1:

OPERATION	READ_ENABLE	WRITE_ENABLE	GENERIC_ENABLE
READ	1	0	1
WRITE	0	1	1

Table 1 - Possible configurations of the three signals

Any other configuration will not produce results (e.g., all outputs ports will be at 0).

As final remark, it worth to be analysed the `DATA` port belonging to the CPU-FPGA interface.. It is both input and output port, hence its behaviour must be compliant with a digital port driven by a tristate logic buffer. That is, when there is a READ operation (i.e. data flows out of the buffer) the tristate buffer must be activated by appropriate control signals, whereas in case of WRITE operation (i.e. data flows in the buffer) the tristate buffer should be in high impedance. Figure 2 describes a simplified hardware implementation:

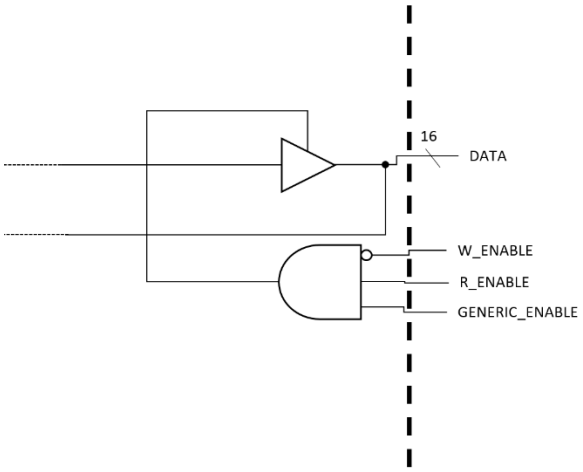


Figure 2 - Simplified hardware implementation of bidirectional port

3. IP Manager interfaces

Main tasks of the IP Manager are the correct selection of the target IP and the interrupt handling. Figure 3 shows the overall architecture of the whole system.

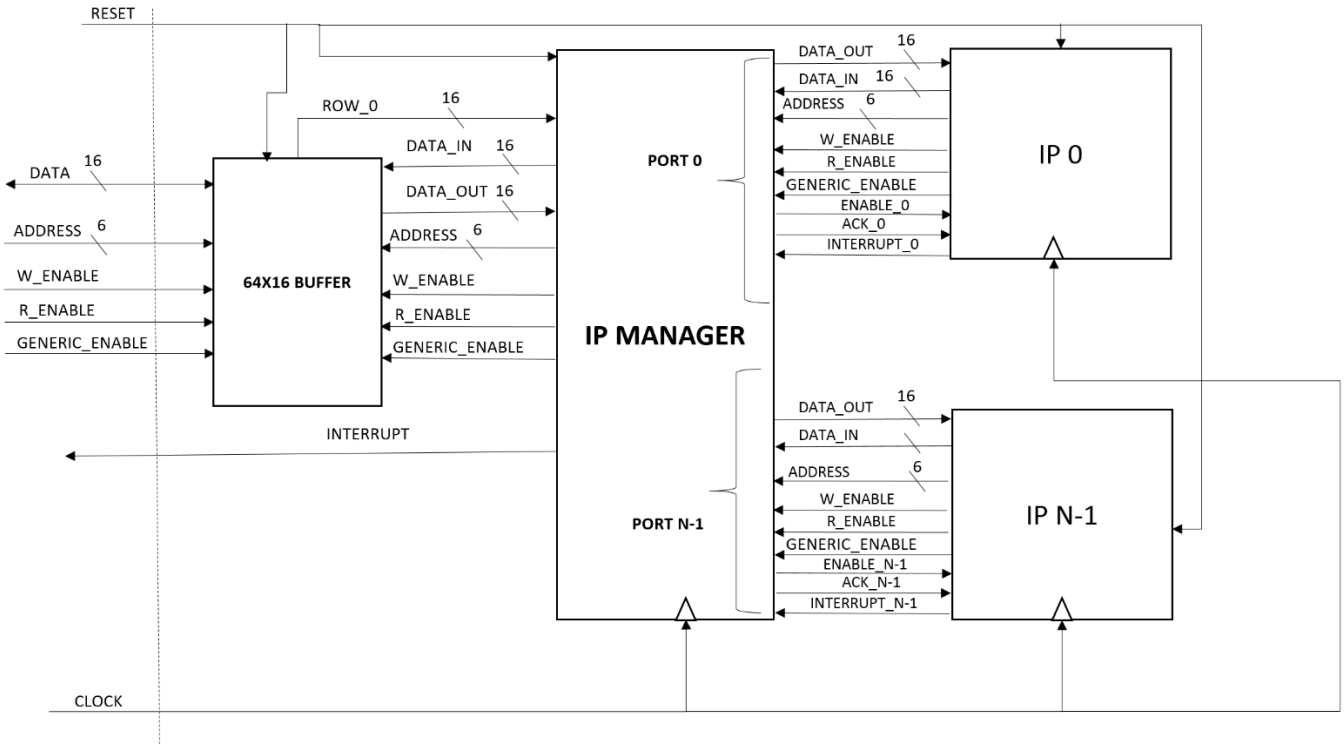


Figure 3 - System architecture

As shown in the figure, the IP manger has a standard interface with the buffer. This interface is used to redirect data to/from the target IP and to access the buffer itself (e.g. to write the row 0). However, it is provided with a direct access (read-only) to the internal element to speed-up the routing process whenever a new transaction begins. It is important to underline that for each instantiated IP there are a set of standard signals and some protocol-specific signals. The former are the very same signals used for accessing the buffer, whilst the latter are used for handling interrupt requests. For sake of clarity, we report the function of the IP-specific signals for the generic IP x:

- `ENABLE_x`: input port for IP x, output port for IP Manger. The IP Manager set this signal at the beginning of a transaction only. The signal is cleared at the end of a transaction. For the whole duration of the transaction, the signal must be maintained asserted. From the IP x viewpoint, as long as the signal is zero the IP itself is not activated and all the outputs are set to 0.
- `ACK_x`: input port for IP x, output port for IP Manger. Asserted by the IP Manager to acknowledge the interrupt request of IP x. It must be maintained asserted for the whole duration of the transaction. The IP x clears on the reception the `INTERRUPT_0` signal.
- `INTERRUPT_x`: output port for IP x, input port for IP Manager. Asserted by IP x to raise an interrupt request. It must be maintained asserted until it receives `ACK_x` from IP Manager.
- `INTERRUPT`: output for the IP Manager, used for signalling that an interrupt arrived from one of the IPs.

For the remaining signals, refer to previous section.

4. Pins Configuration

The present section provides a short description of which pins of the CPU-FPGA connection[1] are used and how should be programmed by the CPU. The signals that should be connected to such bus are depicted in figure 3.

Specifically, they are those crossed by a vertical dotted straight line, in the left-hand side of the figure. In table 2 there are shown such associations:

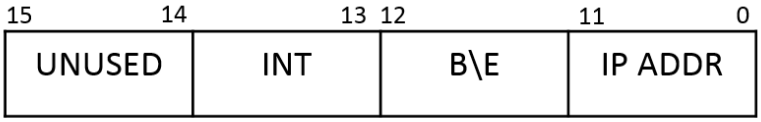
Signal Design Name	FPGA-CPU Connection Name	Programmed
RESET	CPU_FPGA_RST	OUTPUT
DATA	CPU_FPGA_BUS_D0:15	INPUT/OUTPUT
ADDRESS	CPU_FPGA_BUS_A0:5	OUTPUT
W_ENABLE	CPU_FPGA_BUS_NWE	OUTPUT
R_ENABLE	CPU_FPGA_BUS_NOE	OUTPUT
GENERIC_ENABLE	CPU_FPGA_BUS_NE	OUTPUT
INTERRUPT	CPU_FPGA_INT_N	INPUT
CLOCK	CPU_FPGA_CLK	OUTPUT

Table 2 - CPU-FPGA connections

It is worth to be noticed that CPU_FPGA_BUS_D0:15 must be programmed as input during a read operation, while as output during a write operation.

5. CPU Transaction

Whenever the CPU wants to begin a transaction with one of the IPs present on the FPGA, it writes at address 0 of the buffer a data packet compliant with the following specifications:



Bit(s)	Purpose	Value(s)
Bit 15	Unused	Unused
Bit 14	Unused	Unused
Bit 13	Interrupt ACK from the CPU	Interrupt = 1, Normal = 0
Bit 12	Signals the begin/end of a transaction	Begin = 1, End = 0
Bit 11-0	The physical address of the target IP.	From 0 up to N-1

[1] CPU-FPGA connection is detailed in this document (page 8, section 2 FPGA-CPU connection): https://www.dropbox.com/sh/2d5nhfbgwcb4ncz/AABQsdNgy8qTYCukBO9wQoR7a/FP%20-%20FPGA?dl=0&preview=FP1+-+The+SEcube%E2%84%A2+FPGA+Configuration+-+Getting+Started+-+rel+1.4.pdf

In details, this is what happens during a read or write transaction:

The CPU writes at address 0 the control word for the IP manger. In particular, the IP manager will always assume that the IP address that is written from bit 11 up to bit 0 is already the physical address of the IP (i.e. one of the output port) and it will not perform any further translation. Hence, it is up to the software environment to properly connect the right IP to the right port and to configure correctly the device drivers. Figure 4 shows the association that the software tool is supposed to do:

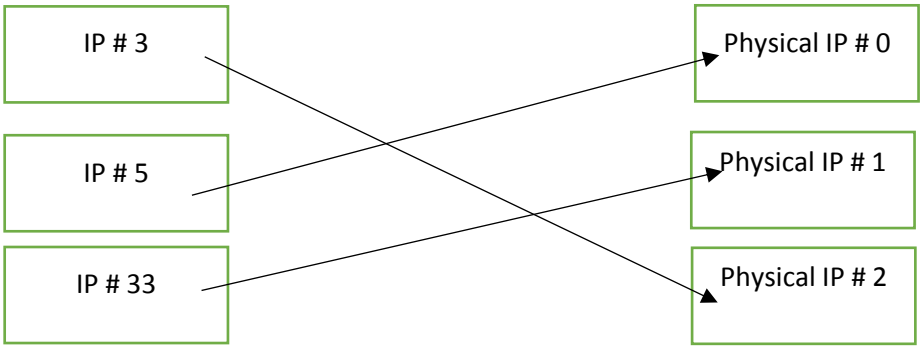


Figure 4 - Association IP number with physical IP

If the physical address of the control world is at all 0s, it means that the CPU wants to interact directly with the IP Manager for some kind of configurations (exploiting bits 15-14, left unused).

It is important to signal the start and the end of a transaction because this information will be exploited by the IP manager to understand whether it can signal to the CPU that an interrupt arose (more details in section 6).

Of course, both CPU and IP have to drive the buffer control signals according to the specifications of section 2 in order to access the buffer.

Form the viewpoint of the IP manger, the behaviour is also the same whether it is a read or a write transaction. Depending on the IP address, it will enable one of the IPs (asserting the `ENABLE_x` signal of Figure 3). The IP manager does not care about how many packets have to be written or read, nor whether is a read/write transaction, since it is IP-dependent. It must only keep the port associated to the target IP open as long as the value written in the IP address field do not change. The target IP will be enabled if the bit 12 is set (signalling the beginning of a transaction) and it will be not enabled as long as bit 12 is at zero (even if there is a change in the bits 11-0). Of course, even if not enabled for the current transaction, each IP has the capability of raising interrupt request if it has been programmed (during a previous transaction) for doing so.

To better clarify the transaction mechanism, it is reported a sequence diagram (figure 4) describing a normal transaction and the format of the control word:

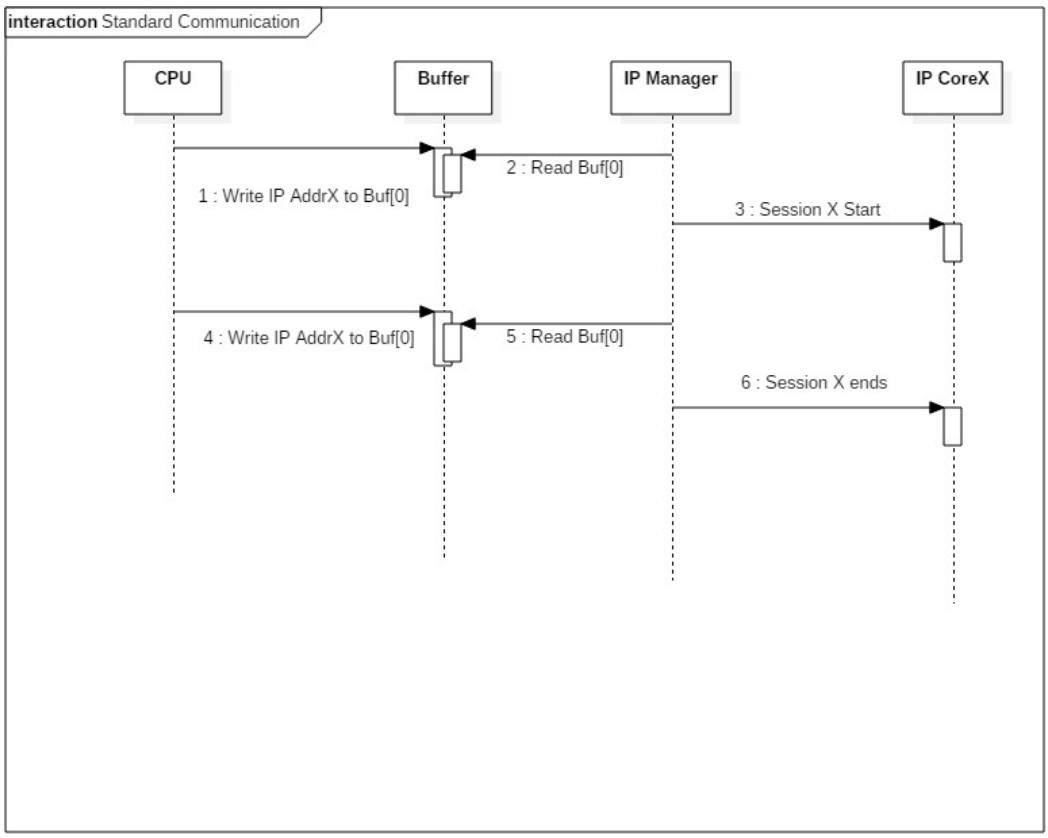


Figure 4 - Normal transaction

The format for the previous transactions (assuming that the target IP is the one connected to port 0):

1) Write IP AddrX to buffer 0(begin):
   - Bit 15,14: x (unused)
   - Bit 13: 0
   - Bit 12: 1
   - Bit 11-0: 001 (in hexadecimal, IP ADDR 0 is dedicated to the IP Manager)

2) Write IP Addrx to buffer 0(end):
   - Bit 15, 14: x (unused)
   - Bit 13: 0
   - Bit 12: 0
   - Bit 11-0: 001

6. Interrupt Handling

The association depicted in Figure 3 will be driven by priority of the IP (specified by the user). Highest priority will be always associated to IP manager port 0 (properly connected to the right IP), and the lowest to port N-1. The IP manager will always assume that IP connected to port 0 is the one with the higher priority.

Since the architecture is a master (CPU) slave (FPGA) architecture, the IP manager must guarantee that an interrupt request from one or more of IPs do not interrupt a transaction started by the master. For doing so, the manager leverages bit 12. Only when it is set to zero it signals to the CPU, through the `INTERRUPT` signal, that an interrupt service routine has to start. At the very same time, it writes to the address 0 of the buffer the IP address of the IP requesting the interrupt.

Before starting any transaction, the CPU reads the content of address 0 so that it knows which is the requesting IP. At this point, the CPU starts a typical transaction, with the exception that bit 13 now is set. Doing so the manager knows that the interrupt has been received, and it can clear `INTERRUPT` signal and sets the `ACK_x` signal to the requesting IP. This signal is cleared once the requesting IP clears its `INT_x` signal (done if and only if it receives the `ACK_x`).

It is worth noting that, if an interrupt request arrives while there is still an ongoing transaction, the manager must maintain the interrupt request alive. Indeed, the requesting IP will not clear the `INT_x` signal until it receives the `ACK_x` signal. The manager will raise this signal if and only if there is not any other ongoing transaction.

As final remark, since there could be several interrupts requests at the same time (with possibly different priorities), the IP Manager must guarantee that all requests are served. In particular, the IP Manger continuously rises the `INTERRUPT` signal as long as there are still pending requests. Hence, the `ACK_x` signal must be sent to only one IP at a time (according to the priority level).

A sequence diagram (figure 5) along with the formats of the control word are shown here (assuming that the requesting IP is connected to port 0):

1) Write IP AddrX to buffer 0(begin):
    - Bit 15,14: x (unused)
    - Bit 13: 1
    - Bit 12: 1
    - Bit 11-0: 001 (in hexadecimal, IP ADDR 0 is dedicated to the IP Manager)

2) Write IP Addrx to buffer 0(end):
    - Bit 15, 14: x (unused)
    - Bit 13: 1
    - Bit 12: 0
    - Bit 11-0: 001

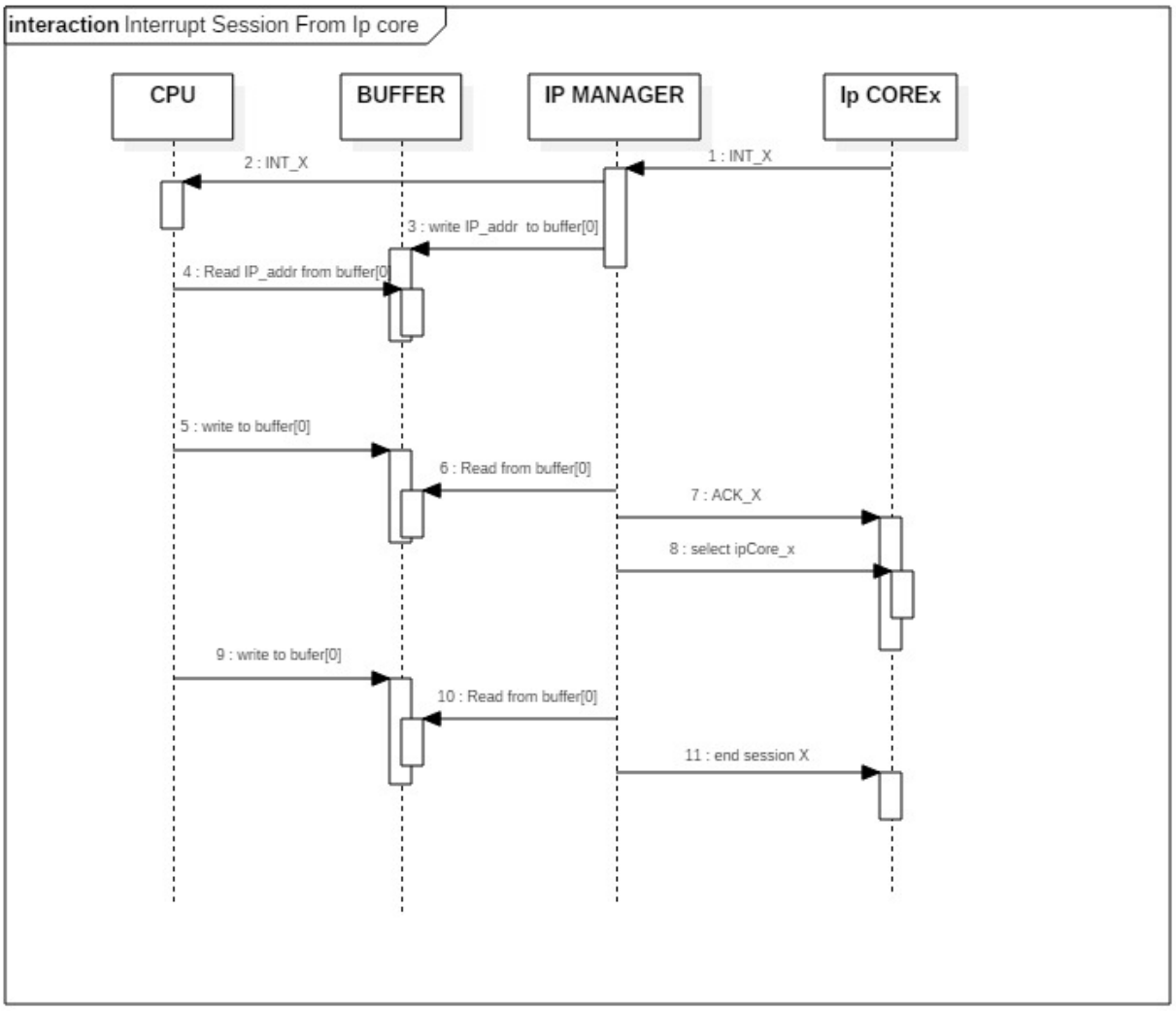


Figure 5 - Sequence diagram for interrupt transaction