Overview of ipbus_ctrl

IPbus version 2.0v1 Draft 1 (27 November 13)

David Sankey

1 Background

The minimum implementation of an IPbus endpoint in firmware consists of a top level design with some four entities, namely a source of clocks (and synchronous resets), a wrapper for the desired Ethernet MAC, the **ipbus_ctrl** entity and an entity containing the desired slaves. A number of example designs for a selection of (to date) Xilinx development boards are given in https://svnweb.cern.ch/trac/cactus/browser/tags/ipbus_2_0_v1/firmware/example_designs, with an associated readme.txt.

An overview of the implementation of the slaves is given in https://svnweb.cern.ch/trac/cactus/browser/trunk/doc/IPbus_firmware_notes.pdf, along with the slaves.vhd used in the above example designs.

This document provides an overview of the options available with the ipbus_ctrl entity.

2 Overview of functionality

The **ipbus_ctrl** entity implements an UDP/IP v4 transport layer over 1Gbit Ethernet for an IPbus endpoint along with simple arbitration between this network access and other 'out of band' (OOB) access to the endpoint. It supports ARP and ping for ease of commissioning and RARP for IP address resolution, along with a variety of configuration methods. These are either specified by generics, ports on the entity or using an internal configuration space.

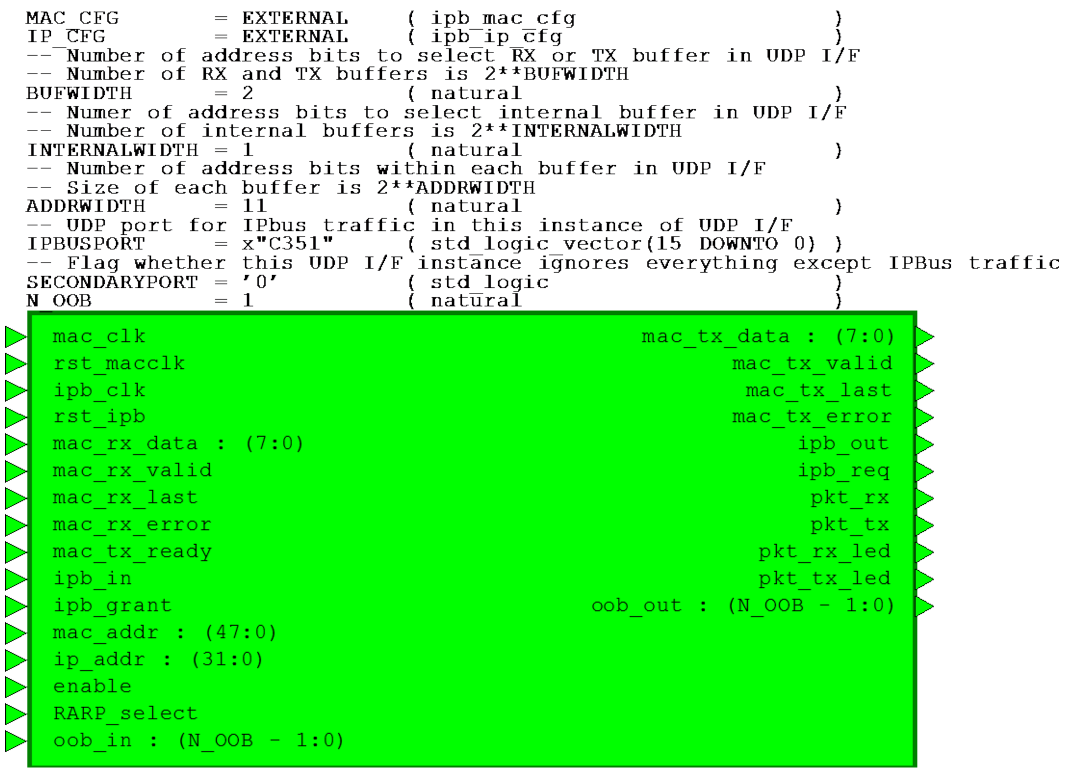
3 Clocks

The transport layer within **ipbus_ctrl** entity runs in the Ethernet MAC clock domain, **mac_clk,** namely 125MHz, with synchronous reset signal (active high), **rst_macclk**. The IPbus logic runs in the IPbus clock domain, **ipb_clk**, typically around 30MHz, with its synchronous reset (again active high) **rst_ipb**. There is no fixed frequency or phase relationship between these two clocks, the only requirement being that **ipb_clk** is slower.

4 Generics

Figure 1 is a symbolic overview of the **ipbus_ctrl** interface, showing the ports and generics.

MAC_CFG and **IP_CFG** determine whether the MAC address and IP address (*pace* RARP mode) are taken from the **mac_address** and **ip_addr** ports (respective generic set to **EXTERNAL**, the default) or the internal configuration space (**INTERNAL**). For a description of the configuration space see the notes section below.



```
MAC_CFG        = EXTERNAL    ( ipb_mac_cfg                  )
IP_CFG         = EXTERNAL    ( ipb_ip_cfg                   )
-- Number of address bits to select RX or TX buffer in UDP I/F
-- Number of RX and TX buffers is 2**BUFWIDTH
BUFWIDTH       = 2           ( natural                      )
-- Numer of address bits to select internal buffer in UDP I/F
-- Number of internal buffers is 2**INTERNALWIDTH
INTERNALWIDTH  = 1           ( natural                      )
-- Number of address bits within each buffer in UDP I/F
-- Size of each buffer is 2**ADDRWIDTH
ADDRWIDTH      = 11          ( natural                      )
-- UDP port for IPbus traffic in this instance of UDP I/F
IPBUSPORT      = x"C351"     ( std_logic_vector(15 DOWNTO 0) )
-- Flag whether this UDP I/F instance ignores everything except IPBus traffic
SECONDARYPORT  = '0'         ( std_logic                    )
N_OOB          = 1           ( natural                      )
```

Figure 1 Symbolic view of ipbus_ctrl interface

BUFWIDTH determines the number of simultaneous packets in flight supported by the interface, specifically the number of bits in the dual-port RAM address field to select the packet, so number of packets is 2 to the power of this number. The default value is currently 4, giving 16 packets in flight.

INTERNALWIDTH is similarly the number of bits in the address field for 'other' packets (ARP, ping, status *etc.*). The default value is the minimum, namely 1 (2 packets).

ADDRWIDTH is number of address bits within each packet, the size of the packet in bytes given by 2 to the power of this number. Default is 11, giving a 2kbyte packet, *i.e.* standard Ethernet packets. The maximum is 13 for 8kbyte packets, slightly smaller than the default for jumbo frames (along with suitable configuration of MAC and switches to support jumbo frames).

The total size in bytes of each of the two dual-port RAMs between the transport layer and the endpoint is given by 2 to the power of (**BUFWIDTH+ADDRWIDTH**) and that of the internal dual-port RAM by 2 to the power of (**BUFWIDTH+INTERNALWIDTH**), all these RAMs being instantiated by inference (see the notes section below if trying to reduce this to the minimum).

IPBUSPORT is the UDP port number for this IPbus endpoint, default being **x"C351"**, 50001.

SECONDARYPORT flags whether this IPbus endpoint responds only to traffic to the above UDP port, so as to coexist with another object sharing the Ethernet MAC. Default is **'0'**, where the endpoint responds to ARP and ping (and in RARP mode, issues RARP requests).

N_OOB is the number of out of bound interfaces to this IPbus endpoint (SPI *etc*.), default none.

5 Ports

5.1 Ports in `mac_clk` domain

`mac_rx_data`, **`mac_rx_valid`**, **`mac_rx_last`**, **`mac_rx_error`** and **`mac_tx_ready`** on input along with **`mac_tx_data`**, **`mac_tx_last`** and **`mac_tx_error`** (never asserted) on output, handle the traffic from and to the Ethernet MAC, satisfying the behaviour corresponding to the Xilinx AXI4_Stream user interface as described in Chapter 7 of the Xilinx UG777 Tri-Mode Ethernet MAC User Guide, as is implemented in Xilinx Spartan 6 and Virtex 6 on. In principle this will work for all speeds supported by this MAC, although all the example designs only implement 1Gbit Ethernet.

With a suitable shim to manipulate the signals, as in `eth_v5_gmii.vhd`, the interface will also cope with the previous Xilinx V5 TEMAC interface (Xilinx UG194). Porting to the Altera interfaces should not be insurmountable.

Finally **`mac_address`** and **`ip_addr`** are assumed stable and are accessed without buffering in the **`mac_clk`** domain.

5.2 Ports in `ipb_clk` domain

Input ports **`enable`** and **`RARP_select`** enable the UDP interface and RARP mode respectively when set to **'1'** (another means of achieving this is through the configuration space described below.. When **`enable`** is set to **'0'** all incoming Ethernet packets are ignored, but out of band access is still enabled.

Input port **`ipb_in`** and output **`ipb_out`** are the bus signals from and to the slaves, format defined in ipbus_package.vhd.

`pkt_rx` and **`pkt_tx`**, and their stretched versions, **`pkt_rx_led`** and **`pkt_tx_led`**, indicate an IPbus packet starting to be decoded by the endpoint and it being finished, *i.e.* state transitions at the IPbus level, not any network level.

`oob_in` and **`oob_out`** are the bus signals from and to the out of band interfaces, format defined in ipbus_trans_decl.vhd.

Finally input port **`ipb_grant`** and **`ipb_req`** allow the possibility of arbitration between multiple bus masters as an alternative to the out of band interface. In standard use this is not enabled and **`ipb_grant`** defaults to **'1'**.

6 RARP mode

Each Ethernet MAC should have a globally unique address associated with it and in general it will be this MAC address that is used. There then remains the need for programmatically assigning the IP address in anything other than the smallest systems.

Within xTCA there is the possibility of assigning the IP address over IPMI, for example through the configuration space, but there remains the need for a network-based method of IP address assignment.

Modern systems tend to use DHCP for this purpose, but this is quite taxing on an FPGA. Instead IPbus uses the older, simpler, RFC903 Reverse ARP mechanism.

Here the IPbus endpoint issues RARP requests until such time as a RARP daemon responds. The initial request is issued within 1s of the UDP interface being enabled (**`enable`** going high), actual time determined by the bottom two bits of the MAC address, with subsequent requests in the absence of a response issued at random intervals of up to 8s.

Note that until such time as a valid RARP response is received the endpoint will ignore all other network traffic. Out of band access is still enabled.

7 Notes

7.1 Configuration space

At present the configuration space is defined as **`std_logic_vector(127 downto 0)`**.

The MAC address is stored in bits 79 down to 32, IP address in bits 127 down to 96, enable as bit 80, RARP enable as bit 81. Enable and RARP mode are enabled by an **`or`** of this signals and the ports defined above.

7.2 Minimising block RAM usage

As stated above, the block RAMs are inferred as dual-port block RAM. The RAM between the Ethernet domain and the IPbus logic have asymmetric port widths, 8 bit on the Ethernet side and 32 bit on the IPbus side. Precision Synthesis correctly infers asymmetric port widths from the HDL, whereas Xilinx XST only infers symmetric port widths.

In general this is not an issue, except for the RAM on the rx side, `udp_dualportram_rx.vhd`, where XST instantiates this as a minimum of 4 block RAMs, even where the RAM would fit in a single block. Hence for the smallest RAM footprint when using XST for synthesis when this dual-port RAM would fit in less than 4 actual block RAMs this module should be replaced by a suitable IP core.