Notes on Firmware Implementation of an IPbus SoC Bus

V1.0 23/5/2012, DMN

Outline

The IPbus system allows the control of hardware via a 'virtual bus', using a standard IP-over-gigabit-Ethernet network connection. IPbus specifies a simple transaction protocol between the hardware and a software controller, which assumes an A32/D32 connection to slave devices connected to the hardware endpoint. Initial implementations of the software and hardware components of the system have been made and tested.

In this document, we give brief details of a firmware implementation of an IPbus endpoint using UDP/IP protocol and a simple synchronous SoC bus. The design has been tested on a range of Xilinx FPGAs with several types of Ethernet connectivity, and is currently being ported to Altera devices. With minor changes, it can be adapted either for minimal resource usage (fits within smallest available Spartan-6 device), or high data transfer performance (~1Gb/s). The firmware is capable of sharing the MAC block with an embedded CPU on an arbitrated basis, allowing it to act effectively as a fast UDP offload engine.

Details of the protocol itself, and of software for controlling large distributed IPbus systems, may be found elsewhere.

Firmware structure

The firmware comprises several components:

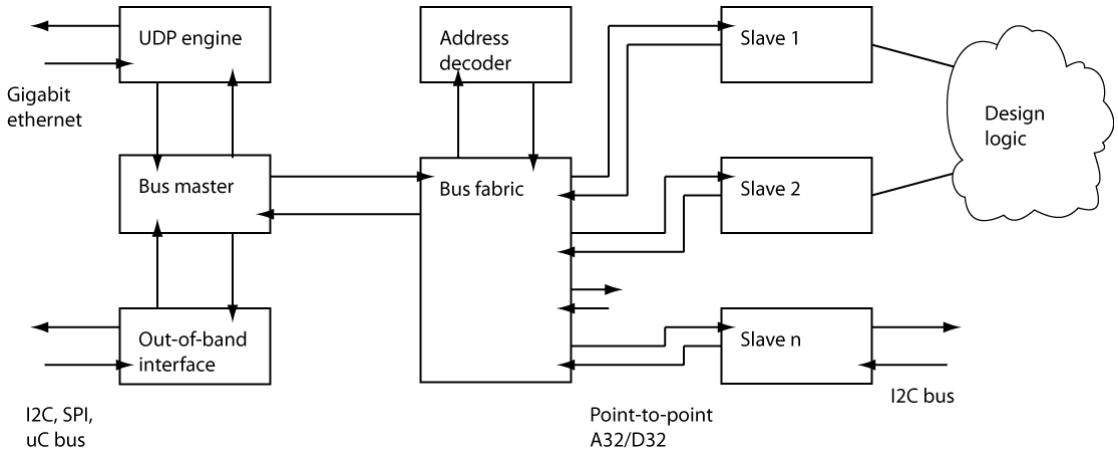
- Interface to Ethernet MAC core, and protocol engines for UDP, ICMP and ARP.
- SoC bus master which decodes an IPbus transaction list
- A optional 'out of band' interface to allow control of the bus via a non-Ethernet route (e.g. via I2C or SPI from an external microcontroller)
- A bus fabric allowing the attachment of multiple slaves to the SoC bus with address decoding.

In addition, some examples of simple slaves are included in the firmware, along with full example designs for various Xilinx evaluation boards.

SoC bus implementation and protocol

Bus topology

The bus is implemented as a set of point-to-point signals. Connection of multiple slaves therefore requires an address decoder and logic to multiplex read data from the slaves back to the master connection. This topology is illustrated below.



The bus fabric arranges for a slave to see activity on the bus when it is being addressed. It is therefore necessary for a slave to decode only its internal address space, and it need have no knowledge of the its position in the overall A32 address space.

Designs need not use either the full A32 or D32 width, and the unused logic should be optimized away in this case.

Clock and bus signals

The SoC bus is fully synchronous, and operates from a single system clock. There is no constraint on the relationship of the bus clock to the 125MHz GbE clock, as the firmware contains handshaking logic. For slaves which do not require wait states, the 32b data path on the SoC bus allows full utilization of the Ethernet interface as long as it runs at >32MHz; this allows straightforward timing closure for complex multi-slave designs. Designs therefore typically drive the IPbus clock at ¼ of the GbE clock (i.e. 31.25MHz)

The bus comprises the following signals:

Signal	Direction	Width	Description
ipb_addr	M -> S	32	Bus address
ipb_wdata	M -> S	32	Data to be written to slave
ipb_write	M -> S	1	Asserted for a write cycle, deasserted for read cycle
ipb_strobe	M -> S	1	Qualifies address and data; assertion marks start of cycle
ipb_rdata	S -> M	32	Data read from slave
ipb_ack	S -> M	1	Acknowledge flag; assertion marks end of cycle
ipb_err	S -> M	1	Bus error flag; assertion marks end of cycle

Bus protocol

There are only two types of bus cycles: read and write. The controller supports RMW cycles as a read immediately followed by a write. No explicit block transfer cycle is supported, as a slave which does not require wait states may fully utilize the bus using normal cycles.

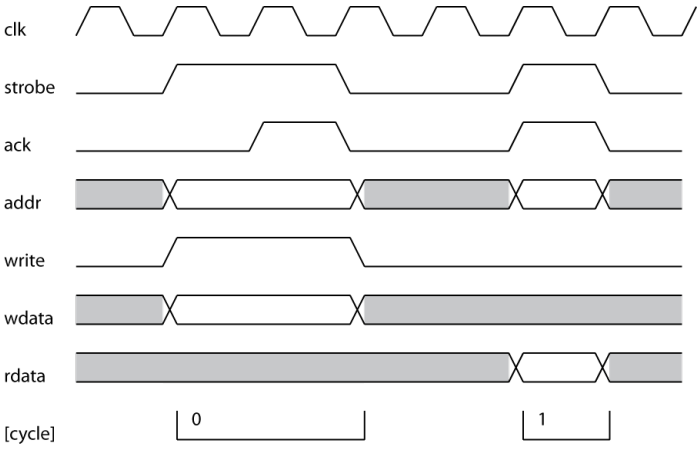
A bus cycle begins with the master driving the address and write bit onto the bus, along with write data if required, and asserting strobe. The cycle ends when the slave asserts ack or err. The slave may respond on the same cycle, or may insert one or more wait states by delaying the ack / err signal. The number of wait states may be different on read and write cycles. The slave is allowed to assert ack on the same cycle as strobe for a

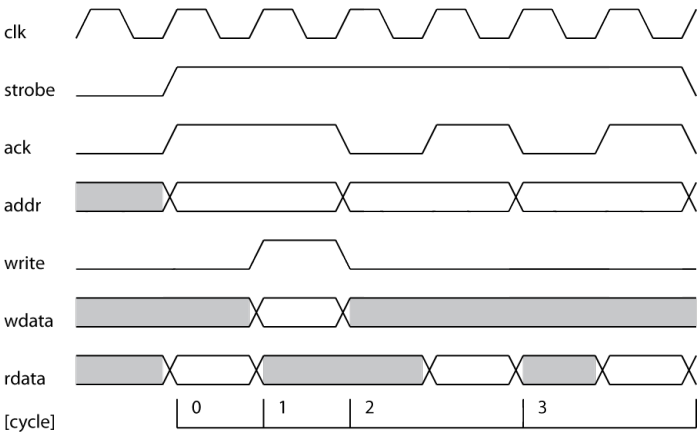
zero wait-state transaction. The master registers the data on the cycle that ack is asserted.
After the termination of the cycle, the master will either deassert strobe, or present new address / data onto the bus, and hold strobe high to signal a new cycle. This protocol is based upon the Wisbone SoC protocol [ref], and is compatible with Wishbone cores. However, there are two important differences:

- The master is *not* required to explicitly deassert strobe between cycles. However, it is *guaranteed* to deassert strobe or begin the new cycle on the clock cycle following ack.

- Slaves are *not* allowed to tie ack high, and *must* deassert ack on the same clock cycle that strobe is deasserted. However, it is allowed to tie ack to strobe, if a zero-wait-state response is always possible.

Timing diagrams of read and write transactions for a slave with and without wait states are given below. The first diagram illustrates a write cycle to a slave with one wait state; the bus idle for two clock cycles; then a read to a slave with zero wait states. The second diagram illustrates a RMW transaction with a slave with zero wait states, followed immediately by two reads from a slave with one wait state.





Bus efficiency

A zero wait-state slave can fully utilize the bus. However, it is not straightforward to write such a slave using registered Xilinx block ram. There are two suggested approaches to solving this issue:

- The firmware includes an example 'peephole' RAM block, which occupies two bus addresses. The first is used for an auto-incrementing address register, the second as a data read / write port. When used in conjunction with non-incrementing read / write IPbus transaction type, this allows efficient sequential access to large memory blocks.

- Depending on the method used to provide the bus clock, it may be possible to use a falling-edge clock to provide zero wait state access to registered RAM.

In order to fully utilize the Ethernet link at ~1000Mb/s, three conditions must be met:

- A zero wait state slave.

- Software capable of handling multiple UDP packets simultaneously in flight; this feature is scheduled for a future release of the software.

- Sufficient buffer space for multiple UDP packets in the protocol engine. Parameterisd buffer size is scheduled for a future release of the firmware.

With the current 'single packet in flight' protocol, throughput of ~200Mb/s has been demonstrated, sufficient for control purposes.

Coupling of slaves to the bus

In order to attach a new slave to the bus, the following steps are required:

- Increase the number of ports on the IPbus_fabric entity via the VHDL generic.

- Modify the address decoder table to specific the address of the new slave.

- Connect the IPbus input / output ports of the slave to the appropriate ports on the IPbus_fabric entity.

Example slaves

The following example slaves are included in the current firmware release:

- A simple register of parameterized width n * 32b with a write-only IO port (one wait state)

- An inferred block RAM with parameterized size (one wait state)

- A bus cycle counter for debugging

- A peephole RAM with zero wait states

- Firmware version register

- IPbus config register allowing the programming of IP / MAC address from an external microcontroller

- Access to configuration, MDIO and statistics functions of the Xilinx MAC core

In addition, the Opencores I2C master core [ref] has been demonstrated to work successfully with the SoC bus.

Additional slaves to be included in future releases include:

- Standard read / write interface to external DDR3 SDRAM

- Standard interface to serdes control registers.

- Standard interface to FPGA internal voltage / temperature monitors