IPBus: Simple IP-based μTCA Control System

Version 1.3

July 14, 2011

Jeremiah Mans, Erich Frahm, Eric Hazen, Robert Frazier, Dave Newbold, Andrew Rose, Greg Iles.

Overview

This document describes a simple IP-based control system for use with uTCA systems. It assumes the existence of a virtual bus with 32 bit word addressing (allowing up to 2^{34} bytes to be addressed) and 32-bit data transfer. The choice of 32-bit widths is fixed in this protocol, though the target is free to ignore address or data lines if this desired by the given target.

Packet Structure

The structure of standard packets is as follows.

For the initial implementation, the transport is UDP, so each transaction must fit in a single UDP packet. Long block transfers must be split in the software level into individual packets.

A packet consists of a set of transactions. There is no overall header, since that would not fit with the long-term possibility of a TCP implementation which would not be packet-based on the SW side. The number of transactions in a given UDP packet must be deduced from the length of the packet and the contents.

Each transaction carries a transaction identifier, as described below. The transaction id is copied from the transaction request to the response, allowing the software to track the processing of individual transactions in a overlapped I/O structure. The transaction ids do not need to be unique, either across a set of controllers or even from a given controller, beyond the requirements of the software in use. The full identification for a transaction consists of the controller's IP address, the request port, and the transaction id.

Each transaction requires at least a single 32-bit word of the form:

	31 28	**27 17**	**16 8**	**7 3**	**2**	**1 0**
0	VERSION	TRANSACTION ID	WORDS	TYPE	D	RES

The definition of the fields is as follows:

VERSION : Protocol version field. This version has VERSION set to 1.

TRANSACTION ID : Identifier for this transaction for use in internal tracking by software

WORDS : Number of words (relevant for Read, Non-incrementing Read, Write and Non-incrementing Write transactions)

TYPE : Type of the transaction. Specific code values are listed in the descriptions of specific transactions

D : Direction of the data flow [0: controller-to-target (request),1: target-to-controller (response)]

RES: Result code (0: OK, 1: PARTIAL (some words transferred), 2: FAIL, 3: RESERVED) Field should be set to zero in requests

Byte Order Transaction

To optimize the efficiency in standard CPUs, it is suggested that the uTCA firmware handle possible byte-order effects. To achieve this, each UDP packet or TCP transaction should begin with a transaction of the form below. It is guaranteed that this is the only transaction where the upper nibble of the most-significant-byte is 1 (the VERSION number) and the upper nibble of the least-significant-byte is 0xF. It is not permitted to have a transaction with the upper nibble of the most-significant-byte being 0xF and the upper nibble of the least-significant-byte being 1, so if this is observed, the byte order should be exchanged.

Request

	31 28	**27 17**	**16 8**	**7 3**	**2**	**1 0**
0	VERS=1	TRANSACTION ID	WORDS=0	TYPE=0x1F	0	0

Response

	31 28	**27 17**	**16 8**	**7 3**	**2**	**1 0**
0	VERS=1	TRANSACTION ID	WORDS=0	TYPE=0x1F	1	0

Read

Request

	31 24	23 16	15 8	7 0
0	VERS=1	TRANSACTION ID	WORDS	TYPE = 0x3 0 0
1	BASE ADDRESS			

Response

	31 24	23 16	15 8	7 0
0	VERS=1	TRANSACTION ID	WORDS	TYPE = 0x3 1 0
1	Data from BASE ADDRESS			
...	Data from BASE ADDRESS+1			
n	Data from BASE ADDRESS+(WORDS-1)			

Non-incrementing Read

E.g. for use with FIFOs, where you want to read the same address multiple times.

Request

	31 24	23 16	15 8	7 0
0	VERS=1	TRANSACTION ID	WORDS	TYPE = 0x8 0 0
1	BASE ADDRESS			

Response

	31 24	23 16	15 8	7 0
0	VERS=1	TRANSACTION ID	WORDS	TYPE = 0x8 1 0
1	Data from BASE ADDRESS			
...	Data from BASE ADDRESS			
n	Data from BASE ADDRESS			

Write

Request

	31 24	23 16	15 8	7 0		
0	VERS=1	TRANSACTION ID	WORDS	TYPE = 0x4	0	0
1	BASE ADDRESS					
2	Data for BASE ADDRESS					
..	Data for BASE ADDRESS+1					
n	Data for BASE ADDRESS+(WORDS-1)					

Response

	31 24	23 16	15 8	7 0		
0	VERS=1	TRANSACTION ID	WORDS	TYPE = 0x4	1	0

Non-incrementing Write

E.g. for use with FIFOs, where you want to write to the same address multiple times.

Request

	31 24	23 16	15 8	7 0		
0	VERS=1	TRANSACTION ID	WORDS	TYPE = 0x9	0	0
1	BASE ADDRESS					
2	Data for BASE ADDRESS					
..	Data for BASE ADDRESS					
n	Data for BASE ADDRESS					

Response

	31 24	23 16	15 8	7 0		
0	VERS=1	TRANSACTION ID	WORDS	TYPE = 0x9	1	0

Read/Modify/Write Bits (RMWbits)

The RMWbits transaction is useful for setting or clearing bits. The algorithm performed is the following:

X <= (X & A) | B

Only single-word RMWbits transactions are defined.

Request

	31 **24**	**23** **16**	**15** **8**	**7** **0**
0	VERS=1	TRANSACTION ID	WORDS=1	TYPE = 0x5 | 0 | 0
1	BASE ADDRESS			
2	AND TERM			
3	OR TERM			

Response

	31 **24**	**23** **16**	**15** **8**	**7** **0**
0	VERS=1	TRANSACTION ID	WORDS=1	TYPE = 0x5 | 1 | 0
1	CONTENT OF REGISTER AFTER TRANSACTION			

Read/Modify/Write Sum (RMWsum)

The RMWsum transaction is useful for adding (or subtracting, using two's complement) values from a register.

X <= (X + A)

Only single-word RMWsum transactions are defined.

Request

	31	**24**	**23**	**16**	**15**	**8**	**7**		**0**
0	VERS=1	TRANSACTION ID			WORDS=1		TYPE = 0x6	0	0
1	BASE ADDRESS								
2	ADDEND								

Response

	31	**24**	**23**	**16**	**15**	**8**	**7**		**0**
0	VERS=1	TRANSACTION ID			WORDS=1		TYPE = 0x6	1	0
1	CONTENT OF REGISTER AFTER TRANSACTION								

Get Reserved Address Information

Each target is allowed to determine the location, length, and data width of its reserved address data. This information should be returned by this transaction.

Request

	31 24	23 16	15 8	7 0
0	VERS=1	TRANSACTION ID	WORDS=0	TYPE = 0x1E 0 0

Response

	31 24	23 16	15 8	7 0
0	VERS=1	TRANSACTION ID	2	TYPE = 0x1E 1 0
1	BASE ADDRESS OF RESERVED AREA			
2	RESERVED SPACE SIZE		RESERVED=0	DATA WIDTH

Reserved Addresses

For the purposes of uniformly identifying chips, developers, and firmware versions, a block of address space shall be reserved for identification strings and codes. Specification details to be added in a subsequent version of this document.

The “Get Reserved Address Information” transaction shall be used to extract the information about the base address and data width for any given target. If the reserved size and data width are set to zero, no reserved/identification records are available.

Implementations of the Protocol

Multiple implementations of the protocol on the target side are possible, depending on the hardware in question. The choices between parallel and serial connections and between microcontroller C-code and HDL will depend on the hardware and application. The known complete or planned implementations are described here.

FPGA Implementation with HDL (HCAL)

For use in the HCAL development project, an implementation of the protocol has been developed which uses the Verilog HDL and the UDP protocol. For this implementation, the protocol is contained within an IP block which uses the built-in MAC and Gigabit serializer capabilities of the Virtex 5 FPGA and which provides a control interface to the rest of the chip using a simple interface allowing the insertion of wait-states where necessary.

The client interface is a simple asynchronous parallel bus interface. The timing and behavior is similar to a simplified VME and the signals are given the same names as in VME.

- `addr` [31:0] (from Core) : Address bus (word-based addressing)
- `data_o`[31:0] (from Core) : Write data bus
- `data_i`[31:0] (into Core) : Read data bus
- `strobe` (from Core) : indicates active transfer (active high)
- `write` (from Core) : indicates that the current transfer is a write (active high)
- `dtack` (into Core): acknowledge successful transfer (active high)
- `berr` (into Core) : failed transfer (active high)

`write` will always be set before strobe (at least one internal clock cycle before). `strobe` will be held until `dtack` or `berr` is seen. Once `strobe` is deasserted, `dtack` and `berr` should also be deasserted to allow the next cycle to occur.

The maximum theoretical byte rate from gigabit ethernet is 125 MB/s. Each transaction generates at least 8 bytes, either incoming or outgoing and neglecting the Ethernet, IP, and UDP overheads. Thus, the maximum transaction rate is 15.6 M bus cycles/sec. The FPGA client should have a cycle capability of at least 50 MHz if the maximum performance of the interface is to be achieved.