IDN-5

MultiBus Interface for the Butterfly

Ward Harriman

May 1983

Multibus Section

I have designed a multibus interface for the present PNC which uses the present PC boards for the PNC and interfaces to the MultiBus. The major aspects are:

1) Uses present PC boards with 1 added wire. This wire will be an ECN on all existing boards. Therefore, the present PNC boards could be used with the MultiBus interface.

2) There will be a change in the BIOLINK protocol which will allow faster transfers along the BIOLINK than are presently allowed. The changes are implemented in the BIOLINK PLAs.

3) MultiBus Writes to PNC memory are buffered up and acknowledged right away. After the buffering, the MultiBus interface will propigate the write through to the PNC memory. This improves performance of all writes

to the PNC memory.

4) A Prefetch Register is implemented which improves DMA channels performance. In short, any read causes the Prefetch register to also be stuffed. If a subsequent read is from the Prefetch register, the data is delivered and the Prefetch Register is updated.

5) Byte swapping is a function of the Mapping Ram. Thus, PNC memory may be swapped or not swapped as a function of the Mapping Ram. Special function data are never swapped.

6) Byte swapping is a function of the SAR, PNC reads and writes on the MultiBus may be byte swapped or not as desired.

7) Reading and writing MultiBus IO devices has been delegated to a PNC special function. The SAR cannot specify that an IO operation can.

8) A method for determining if the BIOLINK contains a MultiBus interface or a BI1 interface is implemented in the PNC special functions.

9) The PNC micro-interrupt PLAs will be changed as a function of the MultiBus or BI1 card interface selection.

In short, The changes to the present PNC boards to support the off board MultiBus interface are limitted to changing 4 PLAs. Since these are always socketted, changing a PNC from one to the other is a simple matter.

Additional features contained on the MultiBus interface board are:

1) Two Uarts which are NOT part of the MultiBus address space.

2) A dummy switch interface with clock generation and reset control.

Butterfly MultiBus Interface

This document discusses the features and basic architecture of the proposed MultiBus interface for the Butterfly processor node (PNC). There are three aspects of the design which are considered: the PNC as a MultiBus master, the PNC as a MultiBus slave, and the changes involved with implementing this interface.

The MultiBus interface for the Butterfly is a microcoded MultiBus board. The microcode implements a new BIOLINK protocol called MBBIOLINK. Unlike the the BIOLINK, the MBBIOLINK protocol allows for data transfers at specific times rather than having them gated with strobes. This results in more data passing over the MBBIOLINK in a shorter time than is possible with the BIOLINK.

Before discussing the details of the MultiBus interface, it is useful to overview the Butterfly architecture. The viewpoint chosen for this overview is that of the programmer's. In this perspective, the Butterfly is a 68000 based, virtual address machine which is segment oriented. Each process uses a unique set of segments, each segment being specified by a Segment Attribute Register (SAR). This SAR contains several fields which are now discussed.

SAR Format

```
+-----------------------------------------------------------------+
| Node Number | Access Cntrl | Swap | Size | Offset | SS | A22-16 |
+-----------------------------------------------------------------+
 32          24 23          21   20   19  16 15    8 7  6 5       0
```

When the cpu makes an access to a virtual address, the memory management unit (MMU) must perform several functions: ensure that the access is valid, determine the type of access, and translate the virtual address into a physical one. These computations start with SAR selection.

The upper 8 bits of the virtual address are used to select one of the SARs. Once a SAR has been selected, the SubSpace field (SS) is examined to determine the type of segment being accessed. The possible subspaces are: local memory, remote memory, MultiBus, or PNC special functions. (The PNC special functions are refered to as SubSpace 0 or SS0.) Of these four types of accesses, only MultiBus and SS0 are of interest in this document.

As the MMU examines the SS field, it also performs the virtual to physical address translation. The physical address is broken up into three parts: a 64k byte block specified by the SAR field A22-16, a page offset into the block (which is virtual address bits 15-8 added to the SAR Offset field), and a byte offset specified by virtual address bits 7-0. Once the physical address has been computed and the type of access determined, the microengine of the butterfly is interrupted and it performs the type of access required.

The first thing the microengine does is load the physical address from the MMU into the appropriate address register. It then checks for access violations and if there are none, performs the requested function. If the SS field is not SS0, then the access is a read or write and the microengine simply performs the appropriate operation. If the SS field is SS0 then some special function is being requested and the microengine performs the special function which usually involves several reads or writes of memory or IO spaces. In any case, when the function has been completed the microengine acknowledges the completion of the access and returns from interrupt.

The above paragraphs have described how 68000 memory accesses are translated into physical access and butterfly microengine interrupts. The MultiBus interface can now be discussed as viewed from the microengine (here-after, the PNC).

The PNC as MultiBus Master

The PNC may become a MultiBus master for three types of transactions: memory reads/writes, IO reads/writes, and interrupt acknowledges (vector fetches). For IO reads/writes, the PNC can support both 8 bit and 16 bit IO address devices. For interrupt acknowledges, the PNC can use either the two or three cycle version, the two cycle version being more common.

In the case of memory reads/writes, the MultiBus supports either 20 or 24 bits of address. The MMU, however, supports only 22 bits of physical address. This conflict has several solutions. The first is to ignore the problem and simply fix the two most significant bits to some predetermined value; perhaps via jumpers or dip switches. Another solution is to support only the 20 bit version of the MultiBus although this seems short sighted. The proposed solution is to use the 'node number' in the SAR as the most significant 8 bits of MultiBus address. Although this makes SARs have multiple formats, the format of the SAR specified by the subspace (SS) field.

When the 68000 accesses the MultiBus, the SS field of the SAR specifies a MultiBus operation. The PNC interrupts into the

MultiBus micro-routines and gates the low 16 bits of physical address into the MultiBus address register. It then gates the Node Number field into the high 8 bits of the address register and examines the IO bit. If the Swap bit is set, the PNC will swap data for the read or write. Byte and word operations are supported.

MultiBus interrupts can be fielded by the 68000 at levels three or four. Any MultiBus interrupt may be connected to either level, and the priority of the interrupts may be re-arranged and enabled/disabled under macro-code control. When the 68000 acknowledges an interrupt at level 3 or 4, the micro-engine determines which MultiBus interrupt is being acknowledged. It then reads an 8 bit vector from main memory and if it is not NULL (not zero) then this 8 bit vector is returned to the 68000. If the vector from memory is NULL then the micro-engine performs an interrupt acknowledge operation on the MultiBus in order to obtain the 8 bit vector from the IO device.

Interrupt Mapping and Control

```
  Interrupt Request Lines
  |7 |6 |5 |4 |3 |2 |1 |0
  |  |  |  |  |  |  |  |
 ,-----------------------,
 |   Interrupt Control   |
 |    & Mapping Ram      |
 '-----------------------'
   | |  |        | |  |
   | |  |        | |  '->-level 3 Interrupt
   |3|  |        |3|____
   | |  |        |______Who at level 3
   | |  |
   | |  |
   | |  '->-level 4 Interrupt
   | |____
   |______Who at level 4
```

MultiBus IO operations are implemented using PNC SSO functions. The Parameter block will contain an opcode, a 16 bit MultiBus address, and a data field. If the operation is a write, the data field will be written as specified by the opcode. Otherwise, a read takes place under opcode control and returns the data read int he data field.

In order to initiate any operation on the MultiBus, the PNC must first become the MultiBus master. Mastership is gained by setting a MultiBus Bus Request bit. Some time later, the PNC is told that it is MultiBus master and may begin an access. After asserting the necessary data and address signals, the PNC will assert the operation strobe on the MultiBus. When transfer acknowledge (XACK) is observed, the PNC will return any necessary data to the 68000 and release control of the MultiBus. The PNC always has the option of performing multiple accesses on the MultiBus, but this is done only for special functions.

Since the MultiBus may have general purpose CPUs, the MultiBus interface must provide for locked operations. The locked operation supported is a 'clear then add' to memory locations (called PNC_am). The butterfly programmer may perform a PNC_am operation in any memory including MultiBus memory. The MultiBus CPU programmer may perform the operation anywhere but MultiBus memory.

The PNC as MultiBus Slave

MultiBus masters may access PNC resources in two ways: they may read and write memory, and they may request SSO functions. The addresses to which the PNC will respond are programmed under macro-code control. The 24 bit MultiBus address space is divided into 256 64k byte segments. Each segment may be either a 64k block of PNC memory (SS3), or PNC special functions (SSO), or no response. The following block diagram shows the implementation of the decoding logic.

PNC as Slave Address Decoding and Mapping

```
-------------------------------------------
          MultiBus Address Bus
-------------------------------------------
       | |MBA23-16                 |  |MBA15-0
       |8|                         |  |
,-----------------,                |16|
| Address Decode |                 |  |
| & Mapping Ram  |                 |  |
'-----------------'                |  |
  | |       |2|                    |  |
  | |       | |_                   |  |
  |6|       |___ SST               |  |
  | |            11  No Response   |  |
  | |            10  SSO Function  |  |
  | |            01  Byte Swapped  |  |
  | |            00  No Swapping   |  |
  | |A21-16                        |  |A15-0
,------------------------------------------,
|               PNC MEMORY                 |
|            256k to 4M bytes              |
'------------------------------------------'
```

A read or write to PNC memory proceeds as follows. The MultiBus master asserts an address on the bus. The upper 8 bits are used to look up a byte in the Address Decode Ram and SST is 'Ox'. The PNC is interrupted and performs a memory read or write as requested; data being swapped or not as requested.

In order to invoke a SSO function from the MultiBus, two problems must be solved. First, a parameter block must be supplied to the PNC. This parameter block specifies required data (for example, a queue pointer), and a location for returned or final status. Second, the PNC must be informed as to which SSO function is to be performed. The butterfly programmer specifies the parameter block and SSO function selection by writing the address of the parameter block into a PNC magic location in SSO. The MultiBus programmer's job is much the same.

In the MultiBus interface, it was decided to take certain short cuts which simplify the microcoding and hardware efforts at the expense of consistency with the butterfly approach. While the parameter blocks used by the butterfly programmer may reside anywhere in PNC local memory, those used by the MultiBus programmer are restricted to a 64k block of PNC local memory known as 'F8' memory. The butterfly programmer allocates a parameter block and passes a pointer to it to the MultiBus programmer.

The MultiBus programmer sets up the parameter block and invokes the SSO function by writing to a MultiBus address which the PNC responds to as Special. The exact SSO functions specified by the lower 4 bits of the MultiBus address. The address of the parameter block in 'F8' memory is the word written. Data swapping is never invoked during SSO functions. An example is in order.

Example PNC SSO Function Invocation

```
/* the butterfly programmer needs to allocate a
 * parameter control block for the MultiBus CPU.
 * It must be in F8 memory.
 */

MultiBus_Pointer = make_MB_addr(F8_alloc(MB_post_pb));

/* the butterfly programmer then hands to pointer
 * to the MultiBus CPU...
 */

event_parameter_block = MultiBus_Pointer;    /* the effect */

/* the MultiBus programmer then sets up the parameters
 */

event_parameter_block -> event_handle = an_event_handle;
event_parameter_block -> event_data   = an_event_data;

/* then he invokes the special function, say opcode
 * 'PNC_Post'.
 */

PNC_Post = (short)event_parameter_block;
```

Changes to the Present PNC.

The major goal of the MultiBus interface design is to minimize the changes in the present PNC PC board. The result is a PC board which can have two different sets of PLAs. One set is the old version of the PLAs and implements the BIOLINK protocol. The second set is a new set of PLAs which allows the PNC to communicate with the MBBIOLINK (MultiBus Butterfly Input/Output LINK). There will be a single version of the microcode which contains all the code for both BIOLINK and MBBIOLINK bus protocols. The PLAs will choose which set of micro-interrupt routines are used for IO transfers.

As described above, there will be significant PNC microcode enhancements. Most of these are either PNC SSO functions, or MBBIOLINK micro-interrupt handlers. None of them seriously impact the existing code and therefore have little chance of breaking anything in the more complex areas of the PNC.

The Architecture

The MultiBus interface is implemented on a MultiBus form factor card. This board contains a FSM which implements the MBBIOLINK protocol. The MultiBus board and the PNC communicate over the 60 pin ribbon cable presently used by the BIOLINK.

When the MultiBus master writes to the PNC memory, the write is buffered on the MultiBus interface board and immediately acknowledged. The FSM then writes the data into PNC memory. Thus MultiBus writes to PNC memory are pipelined and result in a faster block DMA transfer.

When the MultiBus master reads from PNC memory, the FSM performs the read and returns the data to the MultiBus master. It then prefetches a second word. If the MultiBus master then reads from the location contained in the Prefetch unit, the data is immediately returned to the MultiBus master and another prefetch takes place. Thus MultiBus reads from PNC memory are pipelined and result in a faster block DMA transfer.

The major difference between the BIOLINK and MBBIOLINK is in the transfer of data. In the BIOLINK, data transfers are controlled by two signals called Address Strobe and Data Strobe. In the MBBIOLINK, both the PNC and MultiBus interface board understand the timing of all transfers. This eliminates the overhead involved with synchronizing and waiting for the AS or DS signals.

In addition to the basic transfers of data, the MBBIOLINK must provide for controlling the various Rams and Control register. However, since the reading/writing of these resources has a low frequency the MBBIOLINK uses DS to control the actual transfers. This allows the PNC to control timing directly, an advantage when difficult data setup and hold times are present.

Conclusions.

The decision to place the MultiBus interface on a separate card has several advantages. First, there is a performance enhancement because there is more real estate available. Second, it does not affect the PNC PC board and therefore reduces the risk of layout. Third, the MultiBus board may contain the 'fake switch', 'clock', 'reset', and console/debugger IO. In short, the reduced risk and increased functionality of the off board MultiBus interface appears a better choice than the previously proposed PNC with MultiBus interface on board.