CDF Note 452

Specification for a Hardware Event Builder

A.W. Booth, M. Bowden, H. Gonzalez

DRAFT COPY

Contents

1.0 Introduction

1.1 The Goals for a Hardware Event Builder

The goals for the hardware event builder were established through several discussions with people who are very knowledgeable about different aspects of the CDF data acquisition system and about physics data acquisition in general. The reader is directed to the bibliography.

Whilst people had different ideas about which goals were the most important for CDF, all of the goals listed below were felt to be important by at least someone.

a) To read event data or calibration data from front end scanners

b) To reformat the data into YBOS detector bank format

c) To write the data into a level III system

d) To have a throughput of 100 events per second

e) To read data from the cable segment at cable segment bandwidth (40 Mbytes/sec)

f) To have extra memory capacity (to allow for increase in event size)

g) To send and receive messages over FASTBUS

h) To have some degree of parallelism (so that several events may be processed concurrently)

i) To enable readout of unformatted or formatted events by a remote FASTBUS master

1.2 The Role of the Event Builder in the CDF DAQ System

The Event Builder (EVB), as shown in figure 1, is a central element in the DAQ pipeline. It collects event or calibration data from front end scanners and forwards this data to the Level III system. The Event Builder also has the ability to perform some level of data reformatting. The data gather and reformatting functions are currently performed by a software Event Builder executing on the VAX and accessing FASTBUS via the UPI. EVB operations occur in response to control messages from the Buffer Manager. The steps involved in the readout of one event are;

(1) Trigger supervisor broadcasts START-SCAN message to front-end scanners

(2) On receipt of "DONE", the Trigger Supervisor sends a message to the Buffer Manager

(3) The Buffer Manager then sends a PULL EVENT message to the EVB

(4) The Event Builder reads out the designated buffer for all front-end scanners associated with the selected partition and resets scanners

(5) EVB transmits PULL OK message which tells the Buffer Manager that scanner buffers are free for a subsequent event

(6) EVB performs necessary data reformatting and transmits REFORMAT COMPLETE message

(7) Buffer Manager instructs EVB to PUSH EVENT to Level III processor

(8) EVB responds with PUSH OK which indicates that it is ready to accept another event

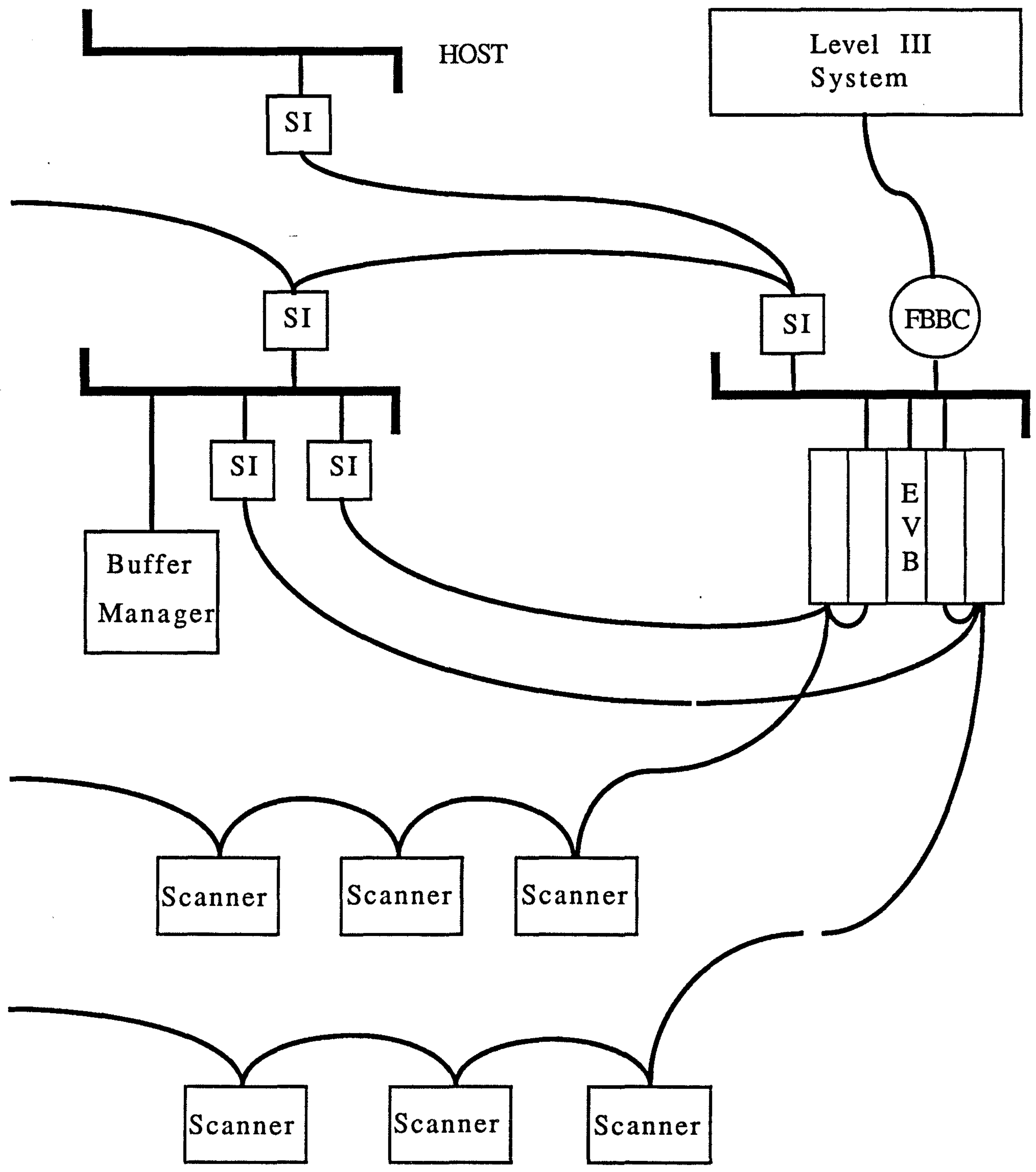

Figure 1 CDF Data Acquisition System

1.3 Event Builder Configuration

The initial configuration of the hardware Event Builder consists of a central controller on the FASTBUS crate segment and an auxiliary controller for each of the two cable segments. Associated with each cable is a reformatter board comprising two reformatting "engines". Since the current CDF system uses two front-end cable segments, a minimum configuration is shown in figure 2. The central controller is a crate segment controller and is responsible for a) communicating with the buffer manager, b) instructing the cable controllers when to read out an event, c) keeping a record of which reformatter is reformatting which event, and d) setting up the FASTBUS master interface on the crate segment for the push of an event to level III.

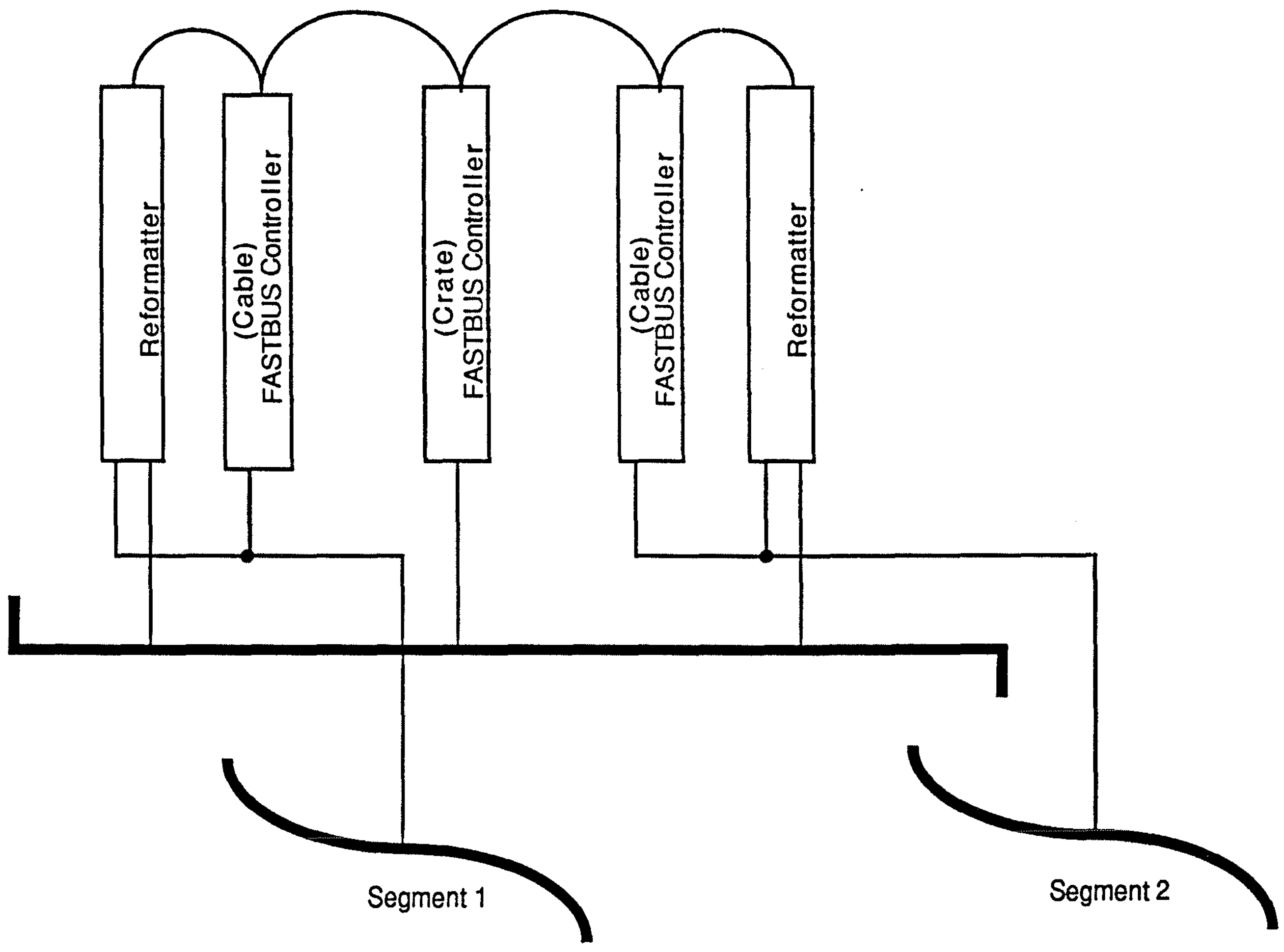
Reformatter
(Cable) FASTBUS Controller
(Crate) FASTBUS Controller
(Cable) FASTBUS Controller
Reformatter
Segment 1
Segment 2

Figure 2 CDF Event Builder (Initial Configuration)

Each of the cable segment controllers are responsible for setting up their

FASTBUS master interfaces to read out events and for controlling their associated reformatter board during the read out .

1.4 Greater Throughput

Each Reformatter board is already double buffered and is organised so that it can perform simultaneously two of three operations: readout event, reformat event and push event to level III. However, with the initial configuration for the event builder (as shown in figure 2 above), and the mean event size as projected in ref. 1, the possible throughput is in the range 30-50 events per second accounting for fluctuations in event size and trigger rate. (This assumes that level III can accept data at 20Mb/sec).

In order to achieve greater throughput, it is necessary to add more reformatting boards (i.e. more buffering). Figure 3 gives an indication of what can be expected from several reformatter boards. If required, it should be possible to fill a FASTBUS crate with reformatter boards.

2.0 Event Builder Architecture

2.1 Controller Board

The following description of the controller board refers to both the crate and cable segment controllers. They are logically the same but differ in their electrical connection to FASTBUS. The controller board in the CDF event builder is based on the controller board of the CERN ALEPH Event Builder [6]. The essential differences are in the FASTBUS slave circuitry (CDF requires more interrupt receivers , for example), in the front panel bus (the CDF EVB front panel bus is primarily for control), and the ALEPH board incorporates a LAN controller (there is no such requirement at CDF[7]). These and other minor changes are documented.

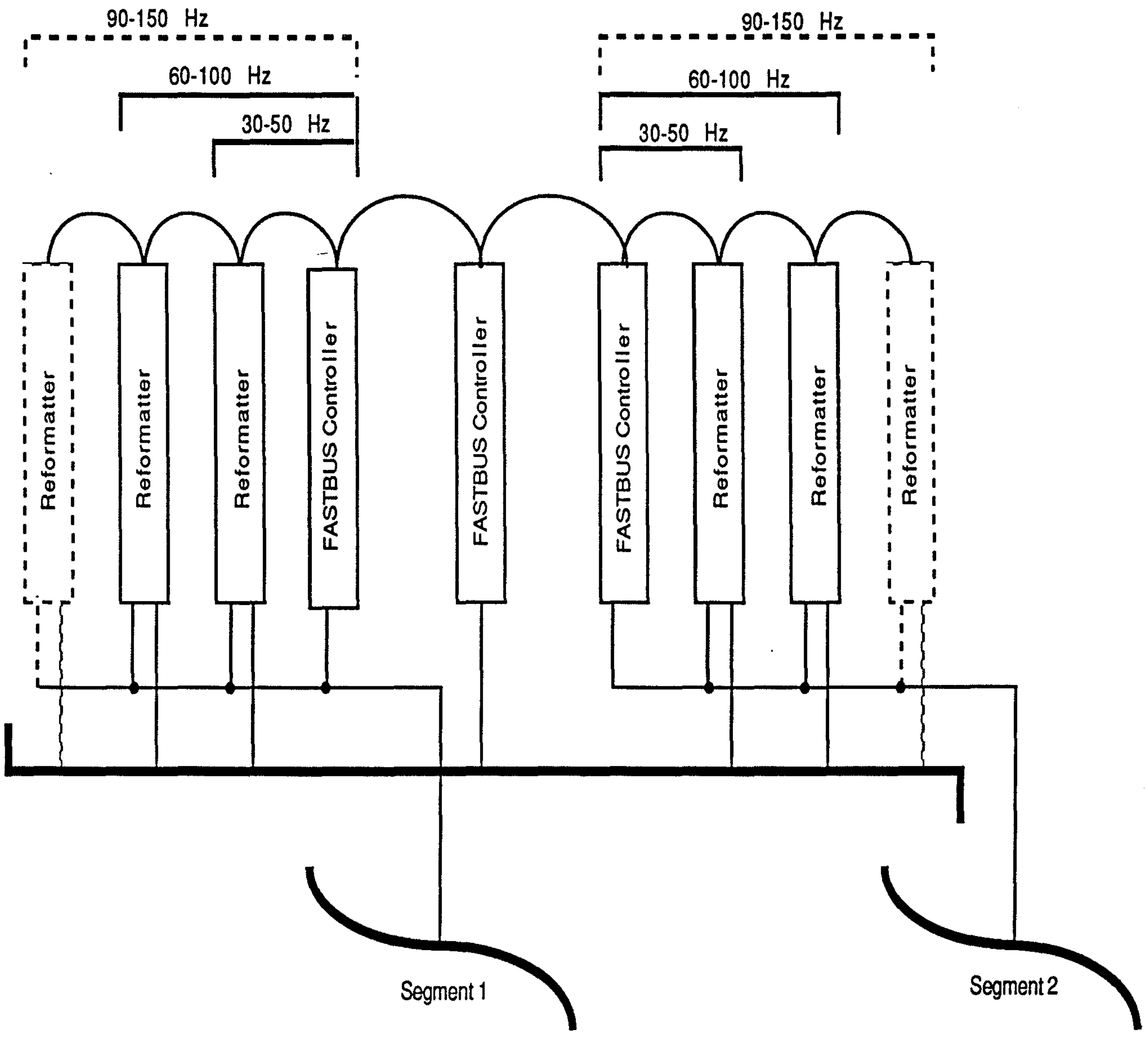

Figure 3 Multi-buffering EVB

2.1.1 Application Processor

The controller board has a Motorola MC68020 microprocessor, which was selected for its high computational throughput and a 32bit input/output capability (hence a desirable candidate for FASTBUS). An attractive additional feature is its ability to support coprocessor hardware which enables FASTBUS to appear as a 'native' bus to the MC68020.

2.1.2 Program Memory

The 68020 program memory is composed of 8 32K x 8 static RAMs. Access to this memory complies with the minimum 68020 bus cycle timing. The 68020 has access to program memory at all times.

2.1.3 FASTBUS Control Sequencer

2.1.3.1 AM2910 Sequencer

The FASTBUS master port is controlled by an AM2910 sequencer (figure 4). The sequencer executes 88 bit microcode instructions with a 75 ns cycle time. The AM2910 allows addressing of up to 4k words of microprogram.

FASTBUS operations are initiated by the 68020 writing 3 or 4 words through the coprocessor interface to the register file of the ALU(29116's); these words are typically primary address, secondary address, word count and microcode jump address (opcode).

Low level FASTBUS functions are performed by discrete logic under direction of the sequencer. This includes skewing of control signals, bus arbitration and block transfer synchronization.

2.1.3.2 ALU

The FASTBUS master port includes a dual 16-bit ALU (AM29116). In the FASTBUS context this is particularly useful for performing bit tests and making low level decisions. It has a 32 word register file which is also used for coprocessor parameter passing.

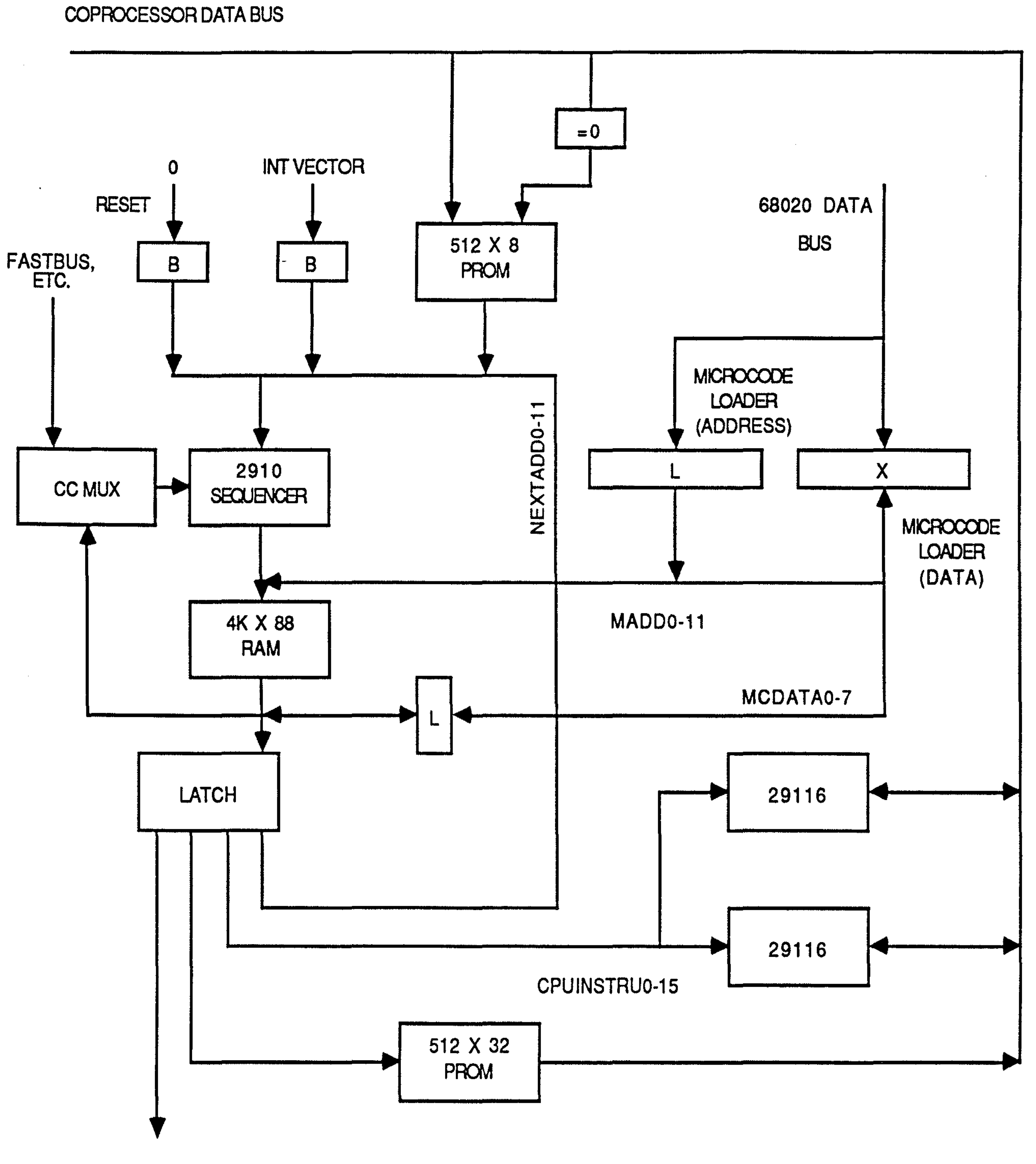

Figure 4 Sequencer

2.1.3.3 Microcode Memory / Pipeline Register

The microcode memory is addressable from either the 68020 (as 11k x 8) for downloading, or the sequencer (as 1K X 88) . The 68020 determines which path is enabled. Memory output is latched by a pipeline register which drives the sequencer and peripheral logic.

2.1.3.4 FASTBUS Control / Status Encoder

FASTBUS control, status and mode select lines drive the sequencer status encoder.

FASTBUS control logic is driven directly from microcode, with the exception of time-critical functions such as bus arbitration and control signal skewing.

2.1.3.5 Timers

Two bus timers are available; a 1 second timer for arbitration and WAIT timeouts and a 2500 nsec timer for address/data response timeouts.

2.1.4 FASTBUS Master Interface

2.1.4.1 Coprocessor Concept

The FASTBUS Master port is implemented as a 68020 coprocessor. This means that it can be considered as a piece of special purpose hardware attached to the main processor (in the same way as a floating point unit). The coprocessor concept allows the capabilities and performance of a general purpose processor to be enhanced for a particular application . The interactions between the main processor and the coprocessor that are necessary for the coprocessor to provide a given service, are transparent to the programmer. That is, no knowledge of the communication protocol between the main processor and the coprocessor is required of the programmer since the protocol is implemented in hardware. This differs from the more traditional approach to dealing with standard peripheral hardware, which is generally accessed through interface registers which are mapped into the memory space of the main processor, and where the programmer is responsible for writing code (using standard processor instructions) to handle that peripheral.

The coprocessor therefore adds new instructions and registers so that FASTBUS commands become part of the instruction set of an enhanced 68020, and FASTBUS itself appears as a 'native' bus to the 68020. To illustrate how the coprocessor mechanism functions let us consider as an example, a FASTBUS single word write to data space. This typically appears in a FORTRAN program as :-

```
CALL  FBSWWD  (Prim_Add,Sec_Add,Write_data)
```

This is compiled[12] into a number of assembly language instructions:-

```
MOVEA.L     Prim_Add,A0
MOVEA.L     Sec_Add,A1
MOVE.L      Write_data,D0
FBSWWD      A0,A1,D0
```

These instructions are then assembled into object code by an assembler[13] which can recognize that FBSWWD is a FASTBUS coprocessor instruction and so must be assembled into an "F-line" instruction . During execution, when the 68020 encounters an F-line instruction, it initiates communication with a coprocessor. Communication with the FASTBUS coprocessor is achieved by the 68020 addressing one of the FASTBUS coprocessor interface registers (CIR), which are mapped into CPU address space. Some protocol then takes place in which the 68020 writes to the "command" register in the coprocessor and then reads a "response" register to determine whether the coprocessor needs further service. In this way the FASTBUS parameters are passed to the coprocessor, which then performs the operation on FASTBUS. The 68020 waits for completion of the FASTBUS cycle (except for block transfers).

2.1.5 FASTBUS Slave Interface

2.2 Reformatter board

The reformatter board consists of a MC68020 and two reformatting "engines". Each reformatting engine is capable of "spying" on the FASTBUS data bus and consists of 0.5 Mbyte of Static RAM, a DMA table ~~and a FIFO~~. One reformatter board can be simultaneously performing any two of the three functions; readout, table building (reformatting) and pushing (writing to level III).

2.2.1 Processor

The 68020 is used for building a table of pointers to the data so that the data can be assembled into YBOS format as it is pushed to level III. The 68020 is also used for communicating with the controller board.

2.2.2 Event Memory, FIFO's and Multiplexors

The memory is accessible from the 68020 (or anything that controls its bus through the Front Panel bus interface) and as a FASTBUS slave. Data transfers between the memory and FASTBUS will be buffered by FIFO's to accommodate asynchronous FASTBUS block transfers and achieve block transfer rates of approximately 100 ns/word. Multiplexors are provided to enable word swapping (for example, when 16 bit MX data has an odd word count).

2.2.3 DMA Table

The DMA table is used to hold pointers to the actual event data so that when the data is pushed to level III, it goes in the order required by YBOS.

2.3 Front Panel Interfaces

2.3.1 Front Panel Bus

The front panel bus is used for communication between the 68020 processors on the controller and reformatter boards.

2.3.2 Logic Analyzer Port

Connectors located on the circuit board behind the front panel permit direct connection of a Kontron logic analyzer to the buffered 68020 bus.

2.3.3 Status Display

Front panel indicators are provided for the following modes;

FASTBUS Master on Crate Segment (green)

FASTBUS Master on Cable Segment (green)

FASTBUS Slave on Crate Segment (yellow)

FASTBUS Slave on Cable Segment (yellow)

Front Panel Master (green)
set by 68020 to indicate it is controlling the FP bus.

ACTIVE (green)
set by 68020 to indicate that it is currently processing an event.

Module Fault (red)
set by power-up or module reset.
cleared by 68020 on completion of self-test.

A 32-element Liquid Crystal Display Module is also envisaged for the front panel bus to display more meaningful messages.

2.4 Auxiliary Board

The cable segment connection is made through the EVB auxiliary port. Signals at the auxiliary connector are identical to those at the crate segment connector, i.e. single-ended ECL FASTBUS. To implement a FASTBUS cable segment, an auxiliary board is used to convert these signals to differential ECL using hybrid cable segment drivers and standard 10KH differential receivers.

3.0 FASTBUS CSR Assignments

Contents of these registers will be described in greater detail in the next release. Basic FASTBUS CSR assignments for the CDF EVB are:

CSR 00000000_{16} - General Control/Status Register

	Read Significance	Write Significance
Bit 00	Error flag	Set error flag
02	Running	RUN *
06	FASTBUS crate segment error	Set error
07	FASTBUS cable segment error	Set error
14	FASTBUS parity error	Set error
15	Active	
16	Module ID = $C8_{16}$	Clear Errors *
17	"	
18	"	
19	"	
20	"	
21	"	
22	"	Clear bit 06
23	"	Clear bit 07
24	"	
25	"	
26	"	
27	"	
28	"	

29	"	
30	"	RESET *
31	"	Clear data *

* Operation involves 68020 interrupt routine

CSR 00000001_{16} - Crate/cable segment error control

	Read Significance	Write Significance
Bit 03	FASTBUS master error	
04	Device retry flag	
05	Network retry flag	
06	Device retry overflow	
07	Network retry overflow	

Device retry on SS=6,7. Network retry on SS=1,3.

CSR 00000002_{16} - Auxiliary Control/Status Register

	Read Significance	Write Significance
Bit04	Interrupts enabled	Enable interrupts
08	Non-existent address	Set error
20	Interrupt pending	Disable interrupts

CSR 00000008_{16} - Arbitration level
(Crate and cable segments)

CSR 00000009_{16} - Timer control

	Read Significance	Write Significance
Bit04	Long timer enabled	Enable Long timer
05	Wait timer enabled	Enable Wait timer
06	Address timer enabled	Enable Address timer
07	Data timer enabled	Enable Data timer

- CSR $0000000A_{16}$ - Interrupt Destination Primary Address (Buffer Manager)
- CSR $0000000B_{16}$ - Interrupt Destination Secondary Address (Buffer Manager)
- CSR $0000000C_{16}$ - Interrupt Destination Primary Address (Buffer Manager)
- CSR $0000000D_{16}$ - Interrupt Destination Secondary Address (Buffer Manager)
- CSR $0000000E_{16}$ - Interrupt Destination Primary Address (Buffer Manager)
- CSR $0000000F_{16}$ - Interrupt Destination Secondary Address (Buffer Manager)
- CSR $000000Ax_{16}$ - Interrupt Message block (Source)
- CSR $0000010x_{16}$ - Interrupt Message block (Receiver)

- CSR $000000Bx_{16}$ - Interrupt Message block (Source)
- CSR $0000011x_{16}$ - Interrupt Message block (Receiver)
- CSR $000000Cx_{16}$ - Interrupt Message block (Source)
- CSR $0000012x_{16}$ - Interrupt Message block (Receiver)

EVB data memory is accessible in data space at $00xxxxxx_{16}$. EVB program memory is not directly accessible from FASTBUS but may be copied to/from data memory in response to an interrupt message.

In a multi-board EVB system, all control messages from the Buffer Manager are sent to the central controller which then allocates reformatter modules as needed.

4.0 Event Builder Operation

Consider the situation where the EVB has been initialised (loaded with scanlists etc.), and is now waiting for a message from the Buffer Manager (BM). What follows is a typical sequence of events.

- BM sends a message to the EVB saying "PULL EVENT"
- 68020 on the crate controller board is interrupted
- 68020 decodes the message
- 68020 checks its "scoreboard" to see which reformatting engine is free
- 68020 sets a bit in a CSR register of the reformatter board enabling one of the two "engines" to start spying on the data
- 68020 sends a message over the front panel bus to the 68020's on the cable segment controller boards to "PULL EVENT"
- 68020's on the controller boards now set up their FASTBUS Master Interface's to perform primary and secondary address cycles to scanners on their respective cables
- When the cable segment controllers have finished "PULLING", they send a message over the front panel bus to the central controller saying "PULL COMPLETE"
- When the central controller has received a "PULL COMPLETE" message from both cables, it sends a message over FASTBUS to the BM saying "PULL COMPLETE"
- the central controller updates its scoreboard
- the central controller then sends a message over the front panel bus to the 68020's on the reformatter boards to "START PROCESSING", which means they start building the DMA table of pointers so that the data will be ready to go to level III in YBOS format.
- When the cable reformatting 68020's have finished "PROCESSING"

they send a message over the front panel bus to the central controller saying "PROCESSING COMPLETE"

- When the central controller has received a "PROCESSING COMPLETE" message from both cables, it sends a message over FASTBUS to the BM saying "PROCESSING COMPLETE"
- the central controller updates its scoreboard
- the event builder is then in a state of waiting for another message from the BM
- the BM sends a message to the EVB saying "PUSH EVENT"
- the EVB central controller is interrupted and decodes the message
- the central controller now sets up its FASTBUS Master port for a write to Level III
- the central controller sends a message to one reformatter board to "write its data" to level III (1/2 of the event)
- when this is complete, the central controller sends a message to a reformatter board (on the other cable) to write its half of the event to level III
- when this is complete, the central controller sends a "PUSH OK" message to the BM

5.0 Software

Functional Requirements

(i) communication with other pipeline stage elements
(ii) communication with other EVB modules (over the front panel bus)
(iii) communication with the host computer
(iv) downloading of scanlists and control tables
(v) reformatting
(vi) error handling
(vii) diagnostics

Considering each of the above in greater detail;

5.1 Communication with Other Pipeline Stages

a) Receiving messages

Any FASTBUS master can write an interrupt message to the EVB by geographically addressing the module, performing a secondary address to CSR $1XX_{16}$ and then a block transfer of up to 16 words. Single-word writes to secondary addresses in the $100\text{-}1FF_{16}$ range will perform the same function. On release of the AS/AK lock, the 68020 is interrupted to read the message from the register file and take appropriate action.

b) sending messages

When the 68020 wishes to send a message to any other pipeline element it must first load the primary address of that device into CSR A_{16}, secondary address into CSR B_{16} and the actual message into CSR $A0\text{-}AF_{16}$. The 68020 then invokes the sequencer which acquires mastership of FASTBUS and enables the block transfer.

5.2 Communication with Other Event Builders

Although the EVB consists of several FASTBUS boards, the crate segment controller is the overall controller of the EVB. It is with this module that all other pipeline elements communicate, over FASTBUS. However, this EVB controller communicates with other EVB modules over a private front panel bus. Typically, the EVB controller writes into the memory space of a "slave" module, releases the local 68020 processor and then writes to the FP bus slave port control/status register to generate a processor interrupt. The 68020 of the "slave" EVB module will then execute the requested interrupt program (e.g., read out its half of the event).

The EVB controller can return at some later time (probably after receipt of a Buffer Manager interrupt message) to inspect the status of the slave EVB. Interrupt routine completion is normally indicated by setting a bit in the FP bus slave port control/status register.

The EVB controller therefore has the choice of operating the "slave" as either a loosely coupled auxiliary processor (as outlined above), or in a tightly coupled mode (by directly accessing "slave" module memory and FASTBUS sequencer with the 68020 halted).

5.3 Communication with the Host Computer

Communication with the host computer is accomplished through FASTBUS interrupt messages or indirectly through the RS232 serial port.

5.4 Downloading of Scanlists and Control Tables

There are three ways in which scanlists and control tables can be loaded into the 68020's program memory:

a) Over the RS232 link

The serial data would be transmitted to the 8052 which would perform serial/parallel conversion and then via the parallel port load the data into the 68020 program memory.

b) From FASTBUS

A remote FASTBUS master can address the event builder and write a new scanlist (or control table) into data memory and then write a message into the slave register file. The 68020 will then be interrupted to read the new scanlist from data memory into its program memory.

c) Over the front panel bus

A front panel master (FPM) could halt the 68020 and then write a new scanlist directly into the 68020's program memory. When the 68020 comes back to life it would service an interrupt to make any necessary adjustments.

5.5 Reformatting

Reformatting is accomplished by a two step process; firstly a 68020 processor scans the event data and constructs a table of pointers to that data. This table is used in the second step, which is a DMA operation where the data is pushed to level III in the order specified by the table. This means that the data is moved once only, i.e. when it is pushed to level III.

5.6 Exception Handling

5.6.1 Retry Philosophy

The 68020 control program incorporates an error handling strategy which essentially attempts to recover from any error situation, but under certain circumstances will send an error message to the Buffer Manager. There is a class of errors that the sequencer can resolve, a class of errors that the 68020 can resolve, and a class of errors where the 68020 requires help or information from other pipeline stages. See section 6 of this document.

5.6.2 Arbitration Requests

START-SCAN broadcasts from the Trigger Supervisor and Interrupt messages from the Buffer Manager share the FASTBUS segments with event data. The Event Builder must detect arbitration requests (AR) on either FASTBUS segment and release the bus. For all operations other than block transfers, the AR line is ignored. If AR is present, the sequencer waits for the pipeline to empty (FIFO empty and all DK's returned), releases the bus and interrupts the 68020. The 68020 will read the current memory address to compute a new secondary address. Continuation of the transfer is queued as if it were a separate operation begining at the point where the previous transfer was interrupted. Maximum bus access latency is approximately 20 μsec.

5.6.3 BUSY / WAIT

The EVB is able to handle WT and SS=1 (BUSY) responses from attached devices. It will also generate WT for cases where it is addressed as a slave and cannot respond immediately.

5.7 Interrupt Handlers

The 68020 receives interrupts at 7 possible levels. Level 7 is a non-maskable interrupt used for system reset. All interrupts are auto-vectored except level 4 which receives a vector mapped from the slave NTA register.

Interrupt Level	Function
7	RESET
6	FP bus interface
5	undefined
4	FASTBUS slave operation
3	undefined (sequencer)
2	undefined (sequencer)
1	FASTBUS error

5.8 Diagnostics

Some diagnostic software already exists for testing the controller board of the event builder. VAX-based diagnostic software is presently being designed.

6.0 System Integration

6.1 Overview

See figure 1 of this specification, and for a more detailed description of the way in which the event builder integrates into the data acquisition system as a whole, see CDF 378. In terms of system integration, the main points brought out by CDF 378 are as follows:-

a) Regardless of the degree of formatting by the event builder, the event must be transformed into detector banks prior to its delivery to any consumer

b) The event readout is a list-driven operation

c) The readout list must be sent to the EVB by the partition Run_control at the initialization of each readout sequence

d) The event data is always read over FASTBUS from the front-end scanners

e) The EVB must allow . . . . FASTBUS event data transfer to overlap the event formatting

f) Both the Trigger Supervisor and the EVB must gain sufficient use of the FASTBUS to permit events to be digitized and transmitted to level III at the system design throughput of 100Hz.

g) The Trigger Supervisor must be configured to allow FASTBUS arbitration for the "start scan" to continue in the event of collision with the EVB.

h) Even though the EVB may consist of several hardware modules, it must appear to the online system as a single entity (especially to the Buffer Manager)

NAME	MEANING
PULL EVENT	Gain mastership of FASTBUS cable segment and read data from front end scanners. If the event builder is also reformatting the data, then reformatting will commence at the end of the readout.
PUSH EVENT	Gain mastership of FASTBUS crate segment and write data to level III.
RESET SCANNERS	This command halts execution of all scanners, sets status="empty" for all front end buffers and returns all scanners to an initial state waiting for the next start scan broadcast.

Messages sent TO the Buffer Manager

NAME	MEANING
PULL OK	This message is sent to the Buffer Manager when the event builder has completed readout of MEP/SSP/Scanner buffer memory.
FORMATTING COMPLETE	This message is sent to the Buffer Manager when the event builder has finished reformatting the event.
PUSH OK	This message is sent to the Buffer Manager when transfer of event data to level III is complete.
EVENT ERROR	This message is sent to the Buffer Manager when a class of error is encountered which the sequencer and 68020 are unable to resolve.
MESSAGE	This message is sent to the Buffer Manager when

the	ERROR	event builder receives a message which it is unable to interpret.
	EVB STATUS	This message is sent in response to a RESET SCANNER command.

6.4 Event Readout by Host VAX

It is possible for the host VAX to readout an event from the Event Builder in either scanner bank or detector bank format. Although the exact procedure for accomplishing this has not been decided, there are several possibilites. An interrupt message could be sent to the EVB followed by a read of data space, or, a particular CSR bit could be set, again followed by a read from data space, or two distinct data space addresses could be associated with scanner bank data and detector bank data. The format of the actual scanner data will be the same as if the VAX had read the scanner directly through a UPI.

7.0 System Development

A Language Resources 68020 system with an IBM-AT or XT will be used for assembly level programming and symbolic debugging. Application programs will be developed in FORTRAN and/or assembly. Schematics will be generated on a Daisy CAE workstation. Netlists will be generated using "wiremaster".

Acknowledgements

We would like to thank Ed Barsotti for his advice and encouragement - his experience has been a great asset. We would also like to thank Sam Segler and Kathy Turner for input on the philosophy of the design and for providing additional resources. We are grateful to Sandro Marchioro and George Mcpherson for many fruitful discussions and invaluable assistance with ALEPH Event builder Schematics, wirelists, PAL/PROM listings etc. Wolfgang Von Ruden's ALEPH team have helped expedite this project. Thanks also to Hans von der Schmitt for the FORTRAN Compiler and Horst Von Eicken for the Motorola 68K Assembler. Not least, we would like to thank Roy Schwitters, Alvin Tollestrup, John Cooper and all our colleagues at CDF who have helped ensure that this specification meets CDF needs.

Bibliography

1. van Ingen Event Builder System Integration Specification, CDF Note 378,1986.
2. Elias CDF Hardware Event builder considerations,1986.
3. Day Buffer Manager Software Design, CDF Note 326,1985.
4. Segler & Turner Private Communication,1986.
5. Vidal Private Communication,1986.
6. Benetta,Marchioro, McPherson,von Ruden The ALEPH Event Builder,CERN,1985.
7. CDF Data Acquisition Board Meeting, March 1986.
8. Carroll Comments on Level 3 Reformatting ,1986.
9. Carroll YBOS Scanner Bank Format,CDF Note 264,1984.
10. Quarrie et al CDF Event Structure,CDF Note 152,1983.
11. Quarrie Dataflow within the CDF Data Acquisition System, CDF Note 183,1983.
12. von der Schmitt RTF68K FORTRAN Compiler, CERN,1986.
13. von Eicken M68MIL Cross Macro Assembler,CERN/DD 83-12, CERN, 1983.