DEFINED LOCATIONS FOR THE BNL787 SSP PROGRAMS

USERS' GUIDE

Version 1.00

Suzannah Daviel

TRIUMF

4004 Wesbrook Mall, Vancouver, B.C., Canada

Computer mail address:

SUZ@TRIUMFCL

January 28th, 1987

CHAPTER 1

USE OF BNL787 DEFINED LOCATIONS

1.1 INTRODUCTION

Since most of the SSPs in the BNL787 system control FASTBUS crates containing different types of slave modules, the code loaded into each SSP is likely to be very different. It is therefore necessary to adopt some standard features to enable efficient accessing of the data from all the SSPs in the system. The standard features described here are defined locations in SSP data memory.

This document is intended to be a clarification of the use of the defined locations.

1.1.1 Definition of terms

For the purposes of this note, the word "SSP" refers to any SSP in the system. Anything referred to as "SSP_dependent" means that it may take a different meaning for every SSP, anything "SSP_independent" must have the same meaning for all SSPs in the system.

SSPs in the system apart from the MASTER_SSP are referred to as "SUBSIDUARY SSPs". Examples of these are the DRIFT_CHAMBER_SSP (the SSP reading the the drift chamber TDCs) and the TRIGGER_SSP (the SSP reading the ADCs for the event trigger). *

The FASTBUS system of SSPs is controlled by the "master". For

---

* Note that in the experiment the TRIGGER_SSP may become the master during the event. In this case, the MASTER_SSP will act as a subsiduary SSP for the duration.

subsiduary SSPs, the master will normally be the MASTER_SSP, or, if a direct path to a host computer exists for test purposes, the master may alternatively be the host computer.

Therefore, in this note, the word "master" refers to the controller of the SSP at the time, that is, either the MASTER_SSP, or the host computer.

1.1.2 New version of fixed locations

A number of SSP data memory locations are assigned as "defined" or "fixed" locations for SSP code written for the BNL787 experiment. The present version represents a large change from previous versions.

A number of the locations previously defined have been found to be redundant, and these have been removed or replaced. Because of this, the size of the space reserved for defined locations has been substantially reduced, meaning user code starts at DM400 instead of DM800.

Many of the defined locations have been regrouped into blocks to enable a FASTBUS block read or write to easily access them. Some of the defined locations need to be written only at the beginning of a run, others must be written on every command. Similarly, some locations must be read after every command has been executed, others only at the end of a run. It is more efficient for the master to use one FASTBUS operation to block read or write a group of locations than to use many single word read/write operations.

In common with the previous version, space has been reserved so that extra locations may be added later. Blocks of reserved space in low data memory for SSP_dependent counters and data tables have been removed. Instead, parameters for the start address and length of blocks of memory reserved for these counters and tables have been substituted (see section 2.8).

In this way, the lowest 4K of memory (which is valuable to SSP programmers) is not wasted.

1.1.3 Files of defined locations

All the files mentioned in this document (except where specifically noted) are to be found in the account 787SSP:

The defined locations are listed in the files

BNL787_RES_D.MAR
BNL787_DEF_D.MAR

THESE FILES MUST NOT BE ALTERED (see 1.1.6)

The files BNL787_DEF.MAR and BNL787_RES.MAR are normally identical to BNL787_DEF_D.MAR and BNL787_RES_D.MAR (except as described in Section 1.1.5). It is the files BNL787_DEF.MAR and BNL787_RES.OBJ that are actually used in the cross assembling and linking process for the BNL787 SSP code (see 787DOC:787_SSP_SYS.MEM).

1.1.4 Cross assembling and linking

The file 787DOC:787_SSP_SYS.MEM gives details of cross assembling and linking code for the BNL787 system. A brief summary is given here.

The file 787SSP:BNL787_DEF.MAR is the definition file needed for cross-assembling using the DCROSS command. Its use with BNLSYS version 4.0 and higher is optional, since the CROSS command is sufficient.

The DLINK command must always be used for linking BNL787 code, irrespective of whether the command CROSS or DCROSS was used for cross-assembling.

The command :
$DLINK FILE

links the code in the file FILE.OBJ to the files BNL787_RES.OBJ and USER_RES.OBJ Note that the files FILE.OBJ and USER_RES.OBJ reside in the user's own account. The file BNL787_RES_D.MAR not only defines the names of the defined locations, but fills them with an initial value, which is their own address in DM.

****

The file of defined locations for the BNL787 data acquisition system will be BNL787_RES_D.MAR, copied and renamed BNL787_RES.MAR. Therefore all SSP programs must expect that the initial contents of all BNL787 defined locations will be that of BNL787_RES_D.MAR. Thus, the SSP programs and/or the master programs are responsible for writing desired initialization values into BNL787 defined locations.

****

1.1.5 Temporary version of BNL787_RES

When debugging and testing SSP code, often the master is not available to fill the initial contents of the defined locations. For example, if an SSP file is being tested using only SNOOPY (or SSPAR) without the DAQ data producer program, it is very tedious for the programmer always to type in default values of locations that should be filled by the master. For this reason, programmers may wish to make their own version of 787SSP:BNL787_RES.MAR, which is a version of BNL787_RES_D.MAR but with default values supplied for locations that should be written by the master.

An example of this can be found in 787SSP:BNL787_RES_TDC.MAR. This file differs from BNL787_RES_D.MAR only in that suitable values have been supplied as defaults in the (SSP_dependent) locations $MAX_VAR, $MIN_VAR,$MOD_MAX,$RETRY_MAX.

To replace BNL787_RES with this file, it would be cross assembled (using the CROSS command), and the resulting object module renamed BNL787_RES.OBJ. This file would then be used for all DLINK commands, system wide. Clearly, altering BNL787_RES.OBJ must be done with caution, and all those writing SSP code on that system must be informed.

Note that only the contents of SSP dependent locations should be changed as described above. On no account must new locations be added (even if reserved ones are used; see 1.1.6).

1.1.6 Adding defined locations

It is essential that the names and position in DM of defined locations are not changed, except by common agreement. Code written for the SSPs should be relocatable in SSP memory, so it is very easy to expand or contract the size of the defined locations in DM, without affecting existing code.

The author is responsible for updating the files 787SSP:BNL787_RES_D.MAR, BNL787_DEF_D.MAR (and BNL787.INC) with new defined locations, and distributing the new versions to computer systems used by BNL787 SSP programmers (currently ERICH and KAON at TRIUMF, and BNL787 at Brookhaven).

Therefore, suggestions for new BNL787 defined locations should be sent to the author (mail to SUZ@TRIUMFCL), bearing in mind that the labels of BNL787 defined locations should begin with "$", and must not conflict with

- Names of BNLSYS locations
- Names of existing BNL787 defined locations
- The following defined subroutine names (see 1.2.1)
  $INITIALIZE,$START,$BEGIN_SPILL,$EVENT,$END_SPILL,
  $PAUSE_DATA,$CONTINUE

1.1.7 FORTRAN INCLUDE file

The INCLUDE file for the FORTRAN programs that access the BNL787 defined locations (787SSP:BNL787.INC) must agree with the locations defined in BNL787_RES_D.MAR, and therefore must not be changed except as noted in Section 1.1.6.

See Chapter 3 for a description of BNL787.INC.

1.2 OPERATION OF FASTBUS SYSTEM

The BNL787 FASTBUS system consists of a number of FASTBUS crates (or CRATE SEGMENTS), each of which contains at least one SSP which acts as the master of the crate. The remainder of the crate will contain slave modules of different kinds that collect the data from the experiment.

The SSPs are linked together by a CABLE SEGMENT. One of the crates is designated the "MASTER CRATE" and this contains the "MASTER_SSP". The "MASTER CRATE" will also contains an interface to the host computer (CFI interface).

Each SSP is responsible for the initialization and set up of the modules in the same crate, and for reading out and storing data from their local crate segment on each event. The MASTER_SSP will then read the data from the other SSPs via the cable segment, and will assemble and store it. It will then be read out by the host.

1.2.1 Commands understood by all SSPs

A number of commands have been defined that all the subsiduary SSPs must understand:

ASCII command	Subroutine called	Action may include:
INIT	INITIALIZE	Initialize modules,crate etc
STAR	START	Initialize counters etc.
BSPI	BEGIN_SPILL	Initialize data buffer pointers etc.
EVNT	EVENT	Read data from modules into data buffer
ESPI	END_SPILL	Process data
PAUS	PAUSE	None
CONT	CONTINUE	None

The subroutines called on each command (which may be dummies) are also listed. The subroutines will typically be different for every subsiduary SSP, depending on the crate contents.

1.2.2 Sending a command to an SSP

CAUTION

> The use of SR and front-panel trigger for the SSP described below makes the assumption that there is only one SSP in the system, and the host is acting as the master. In this case, the SSP asserts SR on the crate segment to interrupt the host.

To send any of the above commands, the master must write the 4 character ASCII command word to the location $COMMAND (see 2.2.1) in the SSP, followed by sending SSP GO to start the SSP.

On receipt of SSP GO, the SSPs must lower SR (service request). After every command has been executed (except if SSP front panel trigger is enabled), the SSPs will raise SR to inform the master that they have finished, and halt. In the case of front panel trigger being enabled, a soft halt (see 1.2.3) is set, and SR is not raised.

After the SSP has halted the master must read the return code (see 2.4.2) to check whether the routine has successfully executed.

1.2.3 SSP Soft Halt

The SSP is controlled by a Run bit in CSR 0 and a trigger. The trigger may either come from a bit in CSR 0 or from the front panel input. The SSP will start if the run bit is set and a trigger occurs. A normal HALT instruction will clear the run bit, which may only be set again via FASTBUS. A soft halt instruction stops the SSP but the run bit remains set, so that a subsequent trigger will restart the SSP.

1.3 DATA FLOW IN SSPS DURING A RUN

The flow would be as follows:

The master sends a command (see 1.2.1) INIT to all subsiduary SSPs. On receipt of this command, all SSPs should call a subroutine INITIALIZE. Initialization includes filling the BNL787 defined locations for crate contents (see 2.7). The slave modules should be made ready for data taking. Further actions are SSP_dependent, and may include filling of a YBOS bank containing initial set-up values of the slave modules. If so, the address and length of the bank should be loaded into the SSP_dependent defined locations $SLAVE_REGS and $LSLAVE_REGS (see 2.5.5, 2.5.6)
After the conclusion of Initialize the SSP halts.

The master sends a command STAR to all subsiduary SSPs. This denotes the start of a run. On receipt of this command, all SSPs should call a subroutine START. The action of this subroutine is SSP_dependent. It may include initializing any counters or histograms for the whole run. After the conclusion of START, the SSP halts.

The master sends a command BSPI to all subsiduary SSPs to denote begin of spill. On receipt of this command, all SSPs should call a subroutine BEGIN_SPILL. The action of this subroutine is SSP_dependent. It may include resetting pointers to the data and address buffers (see 2.5), and initialization of any counters for the spill.

If a front panel SSP trigger is being used, the routine BEGIN_SPILL will write the command EVNT to its own $COMMAND location (see 2.2.1) and enable front panel triggers. It then sets soft halt, and waits for an SSP front panel trigger. Otherwise, BEGIN_SPILL will simply halt in the same way as the other subroutines.

On receipt of a hardware trigger to denote an event, the master SSP will cause the subsiduary SSPs to execute the subroutine EVENT, either by sending a SSP front panel trigger, or by writing the command EVNT.

On receipt of the command EVNT, a subroutine EVENT will be called. This will read the data from the slave modules into a data buffer, where it is stored in YBOS format. EVNT commands will be given repeatedly until either the end of spill occurs, or the buffer becomes full. If the buffer becomes full, a special return code is sent (see 2.4.2). On concluding subroutine EVENT, the SSP will halt unless SSP front panel triggers are enabled, in which case it will set the soft halt.

The master will signal the end of spill by sending the command ESPI. On receipt of this command, a subroutine END_SPILL will be called. The action of END_SPILL is SSP_dependent, and may include processing the raw data. On concluding subroutine END_SPILL, the SSP will halt.

At this point, the master may read all the pieces of the event from the subsiduary SSPs, and combine them in buffer. The commands BSPI,EVNT,ESPI would then be repeated until the end of run.

The commands PAUS and CONT are dummies presently.

1.4 DATA AND ADDRESS BUFFER FORMAT

All the SSPs in the system must collect data from the slaves in their crate segment, and produce a data buffer and address buffer that can be read out by the master at the end of spill. The data buffer may contain raw or processed data. The gathering of the data and filling of the address and data buffers is SSP_dependent, but the final format of the contents of the data and event buffers should be the same for all SSPs.

1.4.1 Event format

The data from each event should be in YBOS format (see Table 1). Each event will typically consist of a number of banks for different types of data, possibly with associated error banks (also in YBOS format).

Each of the data banks is assigned a unique name. Associated with each data bank name is an error bank name (unless an error bank is never needed). If no error has occurred, no error bank will be produced, and the data bank will appear alone. If an error has occurred, an error bank will be added to the event. In some cases, an error bank may occur without the associated data bank. Only one error bank of each name may appear per event, however there may be as many or more different error banks present in the event, as there are types of data banks present. Each error bank name is assigned a unique bit in the error_code_word in BADD (see 1.4.3.1).

The length of the event is the total length (in 32-bit words) of all the banks for that event.

TABLE 1
Format of a YBOS bank

WORD	CONTENTS
1	Name of bank (4 ASCII characters)
2	Number of bank
3	Index to next bank
4	Exclusive length in words of the following datawords (ND)
5	Type of bank
6	....
7	....
	Data for this event
	....
	etc.

1.4.2 Data buffer

The data buffer (known as BDAT) will thus consist of a list of events ordered in increasing event number (Table 2). The length of BDAT will in general be different for every spill, depending on the number of events collected, and the length of the individual events.

TABLE 2
Format of Data Buffer BDAT

WORD	CONTENTS	
1	Exclusive length of BDAT (in 32-bit words)	
2	Start of data for event N .... End of data for event N	EVENT N
	Start of data for event N+1 .... End of data for event N+1	EVENT N+1
	etc.	

1.4.3 Address buffer

Associated with BDAT is an address buffer BADD (see Table 3) that contains four words for each event stored in BDAT. BADD also is of variable length. Thus if 20 events are contained in BDAT, the length of BADD would be 80 words.

TABLE 3
The format of the address buffer BADD

WORD	CONTENTS	
1	Event number of event N	
2	Length (in 32-bit words) of event N	EVENT N
3	Address in BDAT of start of event N	
4	Error code word for event N	
5	Event number of event N+1	
6	Length (in 32-bit words) of event N+1	EVENT N+1
7	Address in BDAT of start of event N+1	
8	Error code word for event N+1	
9	etc.	

1.4.3.1 Error_code_word in BADD -

The fourth word in BADD for each event is the error_code_word. Each bit of this word is uniquely assigned to an error bank (see 1.4.1). If a particular error bank is present in the event, the bit of the error_code_word assigned to that error bank will be set. If no error banks are present in the event, the error_code_word will be zero. The purpose of the error_code_word is to provide a way of knowing quickly whether an error bank is present in that event.

Error_code_word bits are assigned by the author. The following bits have been assigned so far:

- Bit 0 Error bank for drift chamber (bank name DCT) TDC data
- Bit 1 Error bank for drift chamber raw TDC data
- Bit 2 Error bank for drift chamber sorted TDC data
- Bit 3 Error bank for drift chamber ADC data
- Bit 4 Error bank for drift chamber CAMAC data

Please contact the author (SUZ@TRIUMFDC) to request error_code_word bits for new error banks. The assignments of the error_code_word bits (and bit-masks) are in the FORTRAN include file 787SSP:ERROR_BANK_BITS.INC. An array containing the error bank names whose index corresponds to the bit assignments is also provided.

CHAPTER 2

DESCRIPTION OF EACH LOCATION

2.1 INTRODUCTION

The BNL787 defined locations (in 787SSP:BNL787_RES_D.MAR) are described below. Space is reserved for extra locations that may be added later.

All code written for BNL787 experiment must use the defined SSP_independent locations for the purposes described below. Those locations whose use is SSP_dependent may be left empty if not needed. However, if they are used, they should be used as described below, and not diverted to some other use.

2.2 WORDS WRITTEN ON EVERY COMMAND

The following words are input to the SSP by the master. Two words, $COMMAND and $MASTER_ID must be written on every command. The other words may be written less often. $COMMAND and $MASTER_ID, must be written on every command.

2.2.1 $COMMAND

In order to start the SSP running a program, the master writes a command word to the location $COMMAND in the SSP, followed by an SSP GO to start the SSP running. The SSP checks the location $COMMAND to see which subroutine it must execute.

The command word consists of a 4 character ASCII string. The commands that have been defined so far are listed in section 1.2.1 .

2.2.2 $MASTER_ID

This word is a 4 character ASCII string that is to be filled by the master who wrote the command word in $COMMAND. So far, the following IDs have been defined:

- o VAX - ID of host computer
- o MSTR - ID of MASTER_SSP
- o TRIG - ID of TRIGGER_SSP

Note that subsidiary SSPs that never act as the master (see 1.1.1) do not need a MASTER_ID.

2.2.3 $TRIGGER

This word is to be filled by the master. Each bit of this word may be defined to represent a hardware configuration of the experimental trigger external to the SSP. The definition of the bits are SSP_independent.

2.2.4 $PR_EVENT

This word is the number of the present event. It is to be incremented and written by the master every time the EVENT command is executed.

2.2.5 $SPILL

This word is the spill number, and is to written by the master at the beginning of each spill. The begin of spill is signified to the SSP by sending the BEGIN_SPILL command.

2.2.6 $RUN

This word is the run number, and is to written by the master at the beginning of each run. The begin of run is signified to the SSP by sending the START command.

2.3 WORDS WRITTEN PRIOR TO INIT

The following commands must be written prior to the INIT command being given. They need not be written again, as they must remain unchanged throughout the run.

2.3.1 $OPTION

This word is to be filled by the master. Each bit of this word can be used to enable a software option in the SSP. The definition of the bits are SSP_dependent. If the option word is equal to zero, no options are enabled. The option word must be filled by the master before the command INIT is given, and must remain unchanged for the other commands. If $OPTION is changed, the initialization must be repeated.

Failure to ensure that $OPTION remains unchanged throughout the run may cause routines to fail, since $OPTION bits may affect what slave modules are to be initialized.

A FORTRAN INCLUDE file (in 787SSP:) should be provided by the SSP programmer that gives the assignments of the (SSP-dependent) bits in $OPTION. For example, the file DC_OPTION_BITS.INC contains the bit assignments (and bit-masks) for the drift chamber SSP.

2.3.2 $MAX_VAR

This word is used to set a limit on the total length of the data obtained by variable length reads during one event. Any extra words will be discarded, and the counter $OVERFLOW (see 2.6.6) should be incremented. An error bank may be produced.

What this maximum number actually refers to is SSP_dependent. For example, if a fixed number of words is read every event, $MAX_VAR will be ignored. If part of the event is a variable length read, and part fixed, $MAX_VAR will refer to the maximum length of the variable part only.

This word may be used to estimate the maximum event size expected (needed for checking for free space in the data buffer). It may be convenient (for debugging) to load this location with a default value (this can be done in BNL787_RES.MAR, see 1.1.5)

2.3.3 $MIN_VAR

This word is used to set a minimum value on the length of the data obtained by variable length reads during one event. If the minumum is not reached, the counter $UNDERFLOW (see 2.6.7) will be incremented. The event should not be discarded, but an error bank may be produced.

What this minimum number actually refers to is SSP_dependent. For example, if a fixed number of words is read every event, $MIN_VAR will be ignored. If part of the event is a variable length read, and part fixed, $MIN_VAR will refer to the minimum length of the variable part only.

It may be convenient (for debugging) to load this location with a default value (this can be done in BNL787_RES.MAR, see 1.1.5)

2.3.4 $MOD_MAX

This word is used to set a limit on the number of datawords obtained from any one module during one event. The data from the module must be of variable length. The exact use of $MOD_MAX is SSP_dependent; it is ignored for fixed length reads.

If $MOD_MAX is exceeded, an error bank may be added to the event. Counting of events where $MOD_MAX is exceeded, is SSP_dependent.

It may be convenient (for debugging) to load this location with a default value (this can be done in BNL787_RES.MAR, see 1.1.5)

2.3.5 $RETRY_MAX

If a slave module is busy when a master attempts to read the data, an SS=1 (busy) response is received. The master is expected to retry sometime later with every expectation of success. $RETRY_MAX contains the maximum number of retries on receipt of this condition before giving up, and continuing with the next module. The time delay between retries is SSP_dependent - for example a loop may be used to produce a time delay.

An error bank may be added to the event if the number of retries is exceeded. Counting of events where this occurs is SSP_dependent.
It may be convenient (for debugging) to load this location with a default value (this can be done in BNL787_RES.MAR, see 1.1.5)

2.4 WORDS READ AFTER EVERY COMMAND

The following words must be read by the master on completion of every command:

2.4.1 $MODE

The mode word of 4 ASCII characters must be written by the SSP on receipt of one of the commands listed in 1.2.1. The mode words listed below have been defined so far:

ASCII Mode word written by SSP		ASCII command read from $COMMAND
INIM	denotes receipt of	INIT
STAM	" "	STAR
BSPM	" "	BSPI
EVNM	" "	EVNT
ESPM	" "	ESPI
PAUM	" "	PAUS
CONM	" "	CONT
GOIN	denotes that a command has been received, SSP has started.	
KRAP	denotes that an unrecognised command has been received	

The mode word "GOIN" is usually quickly overwritten by one of the other mode words, but is useful for debugging purposes.

2.4.2 $RET_CODE

The return code from the subroutines is written into this location. It consists of 4 ASCII characters. The return codes defined so far are as follows:

- o SUCC
  The subroutine has executed successfully. The next command can be issued. All the subroutines defined in 1.2.1 must be able to return this code.

- o FAIL
  The subroutine has failed for some reason. The FAIL return code is considered a fatal error, and the reason for the failure must be investigated and corrected before continuing. More information on the reason for failure can be found in $MORE_INFO (see 2.5.7). All the subroutines defined in 1.2.1

must be able to return this code.

- DFUL
  This return code must be returned by EVENT if the data buffer becomes full. It is not considered a fatal error. The condition can be corrected by sending a BEGIN_SPILL command, which resets the data and address buffer pointers. In some implementations, other subroutines may return this code if desired.

- AFUL
  This return code must be returned by EVENT if the address buffer becomes full It is not considered a fatal error. The condition can be corrected by sending a BEGIN_SPILL command, which resets the data and address buffer pointers. In some implementations, other subroutines may return this code if desired.

2.4.3 $PROGID

A unique ID of 4 ASCII characters should be assigned to every SSP program that will be loaded into one of the SSPs in the BNL787 experiment. (It is likely that every SSP will be loaded with a different program). This program ID should be loaded into the location $PROGID after the command GO is given, so that the master can easily check which code has been loaded into any particular SSP.

No SSP program IDs have yet been assigned. It would be appreciated if SSP programmers would assign IDs to their own programs, and communicate them to the author (SUZ@TRIUMFCL) to ensure that they remain unique.

2.5 BUFFER LOCATIONS AND SIZES

The first four words (address and length of data buffers) would normally be read by the Master after the end of spill (ESPI) command has been given. The last two words need only be read if an error return code is generated.

2.5.1 $ABADD

This location should be filled with the address in DM of the start of the address buffer BADD (see 1.4.3) from which the master is to read the addresses of the data stored in BDAT.

2.5.2 $LBADD

This location should be filled with length in (32-bit words) of the address buffer BADD (which is of variable length, see 1.4.3) .

2.5.3 $ABDAT

This location should be filled with the address in DM of the start of the data buffer BDAT (see 1.4.2) from which the master can read out the data.

2.5.4 $LBDAT

This location should be filled with length in (32-bit words) of the data buffer BDAT (see 1.4.2), i.e. the length of the data buffer that contains data.

2.5.5 $SLAVE_REGS

This location should be filled with the address in DM of the start of a YBOS bank that contains the initial set-up values of the slave registers in the crate, or any information of a similar nature. The use and contents of this bank is SSP_dependent.

The bank may read by the master at the beginning of run and written onto magnetic tape, thereby providing some record of the setup parameters of the slave modules for that run.

If no such bank is in use, the location $LSLAVE_REGS (2.5.6) should be set to zero.

2.5.6 $LSLAVE_REGS

This location should be filled with the length (in 32-bit words) of the (SSP_dependent) YBOS bank whose address is loaded into $SLAVE_REGS (see 2.5.5). If no such bank is in use, the location $LSLAVE_REGS should be set to zero.

2.5.7 $MORE_INFO

This location contains the address of a storage block in DM (known as MORE_INFORMATION) that contains status or error information. In the event of a return code of FAIL (see 2.4.2), MORE_INFORMATION should contain more information on the reason for failure, so that the condition may be corrected.

2.5.8 $LMORE_INFO

Since the block MORE_INFORMATION (see 2.5.7) is, in general, of variable length, the length (in 32-bit words) of the contents of MORE_INFORMATION is stored in $LMORE_INFO.

2.6 COUNTERS READ AT END OF RUN

The following counters are incremented throughout the run, and are only cleared on a START command.

2.6.1 $THISPNT

The pointer to the location in BDAT (see 1.4.2) of the data of the current event being read out is stored in $THISPNT. Obviously, this value must be updated every time subroutine Event is called.

2.6.2 $THIS_EV

This location is used as an internal counter of event number by the SSP. Since the master writes the event number each time it gives the command EVENT, it is useful to have an internal counter in each SSP to check that the event numbers are consecutive. In this way, missing events can be detected and counted (in $MISSEV see 2.6.5).

2.6.3 $TOT_EVNTS

This location is used to count the number of events that have been collected during the run.

2.6.4 $NEVBAD

This location is used to count the number of bad events that have occurred during the run. The definition of a bad event is SSP_dependent; for example it may be defined as any event that contains an error bank.

2.6.5 $MISSEV

An internal event counter ($THIS_EV - see 2.6.2) is used to count the events for comparison with the value $PR_EVENT (see 2.2.4) written by the master. If an event is discovered to be missing, the counter $MISSEV is incremented.

2.6.6 $OVERFLOW

If an event is cut short because $MAX_VAR (2.3.2) has been exceeded for the variable part of the event, the location $OVERFLOW will be incremented.

2.6.7 $UNDERFLOW

If the variable part of an event is less than $MIN_VAR (2.3.3), the location $OVERFLOW will be incremented.

2.7 CRATE AND CABLE SEGMENTS

The following should be filled on receipt of command INIT, and may be read out by the master and written on tape at the beginning of run.

2.7.1 $CRATE

This marks the start of 27 words of storage for the crate contents.

The number of modules found in the crate are contained in location $CRATE, followed by a list (in consecutive locations) of one word per module arranged in order of increasing geographic address, such that the manufacturer’s ID of the module is stored in bits 16-31, and the geographic address in bits 0-15.

These locations should normally be filled during crate initialization.

2.7.2 $CABLE

This marks the start of 16 words of storage for the Manufacturer's IDs of modules on the cable segment (which are expected to be SSPs).

The number of modules found on the cable segment is stored at location $CABLE, followed by a list (in consecutive locations) of one word per module found, in order of increasing geographical address, such that the manufacturer's ID is stored in bits 16-31, and the geographical address in bits 0-15.

Only the master will fill these locations.

2.8 POINTERS TO HISTOGRAMS, TABLES ETC.

These are provided so that the master can read out any SSP_dependent counters and/or histograms using a block read. The starting address and length (in 32-bit words) of the data block are all that is required.

2.8.1 $COUNT

This should be filled with the address of a block of memory which is used for SSP_dependent counters.

2.8.2 $LCOUNT

This should be filled with the length (in 32-bit words) of the block of memory starting at $COUNT used for SSP_dependent counters. If no SSP_dependent counters are defined, this location should be zeroed.

2.8.3 $HIST

This should be filled with the address of a block of memory which is used for SSP_dependent histograms.

2.8.4 $LHIST

This should be filled with the length (in 32-bit words) of the block of memory starting at $HIST used for SSP_dependent histograms. If no

SSP_dependent histograms are defined, this location should be zeroed.

2.8.5 $TABLE

This should be filled with the address of a block of memory which is used for SSP_dependent tables.

2.8.6 $LTABLE

This should be filled with the length (in 32-bit words) of the block of memory starting at $TABLE used for SSP_dependent tables. If no SSP_dependent tables are defined, this location should be zeroed.

CHAPTER 3

PARAMETERS FOR DATA PRODUCER

Parameters needed to interface the FASTBUS data producer for DAQ with the SSP are provided in the FORTRAN INCLUDE file 787SSP:BNL787.INC.

If changes are made in the BNL787 defined locations (see Section 1.1.6), the FORTRAN INCLUDE file must be changed accordingly.

3.1 SSP DM PARAMETERS

The parameters named DM_P_* (in BNL787.INC) correspond directly to the BNL787 fixed locations in the SSP data memory, which are described in detail in Chapter 2 above.

Note that these parameters contains the byte address in DM. Therefore, to obtain the word address, divide by 4.

3.2 SIZE PARAMETERS

3.2.1 DM_NW_MASTER

This contains the length of a block (in 32-bit words) to be written by the MASTER on the command INIT. The block should include all BNL787 defined locations that are ever written to the SSP by the MASTER.

3.2.2 DM_NW_MASTER_SHORT

This contains the length of a block (in 32-bit words) to be written by the MASTER on commands other than the INIT command. This parameter represents the minimum number of words that must be written on giving a command.

3.2.3 DM_NW_STATUS

This contains the length of a block (in 32-bit words) to be read by the MASTER after a command has executed. It includes all words necessary to read a status report from the SSP, i.e. it includes all words up to and including $LMORE_INFO, but excludes counter and histogram addresses that may be read only once per run.

Note that a short status length parameter is not required, since it is only necessary to read one word ($RET_CODE) for this information.

3.2.4 DM_NW_787_FIXED

This contains the total length (in 32-bit words) of the area in DM occupied by the block BNL787 fixed locations. An array DMDAT of length DM_NW_787_FIXED is provided to correspond to the contents of the BNL787 defined locations. For example, the location $COMMAND would correspond to the array element DMDAT(DM_P_COMMAND/4).