Typer Builder Module Specification

James Francis

MIT 64-30111.0801

Rev. C

June 16, 1993

Contents

Figures

Tables

1 Introduction

1.1 Scope

This document describes the components that make up the Typer Builder Module (TYPER). The Typer Builder Module decodes compressed typer tables.

This document does not describe the overall design of the EDS, nor does it describe the operating details of each component in this module. It assumes that the reader is extremely familiar with the Flight Software Global Design, with the "C" programming language and with the architecture of the EDS.

This document should be of primary interest to Flight Software Engineering, Flight Hardware Engineering, Ground Software Engineering and Software Quality Assurance.

1.2 References

MIT 64-02009 Rev. E .... "EDS-PCA Science Requirements".
MIT 64-03001 Rev. G .... "PCA-EDS Interface Control Document".
MIT 64-02006.01 Rev. D .... "XTE EDS Hardware Functional Description and Requirements".
MIT 64-30102 Rev. E .... "Flight Software Global Design".
MIT 64-30111.0501 Rev. C .... "Recoder Builder Module Specification" (RECODER)
MIT 64-30112.0601 Rev. D .... "Science Thread Module Specification" (SCIENCE)
MIT 64-30110.2001 Rev. C .... "Utility Module Specification" (UTIL)
MIT 64-30110.1701 Rev. D .... "Thread Manager Module Specification" (THREAD)

1.3 Notations

This document assumes the following notations:

UINT8 .... Refers to an 8-bit unsigned value.
UINT16 .... Refers to a 16-bit unsigned value.

1.4 Module Perspective

The following diagram is a copy of the Level 1 data flow diagram from the **Flight Software Global Design**, Section 4.1.2. The Typer Builder Module does not appear directly in this data flow, but has a scope which directly affects items indicated by the shaded areas:

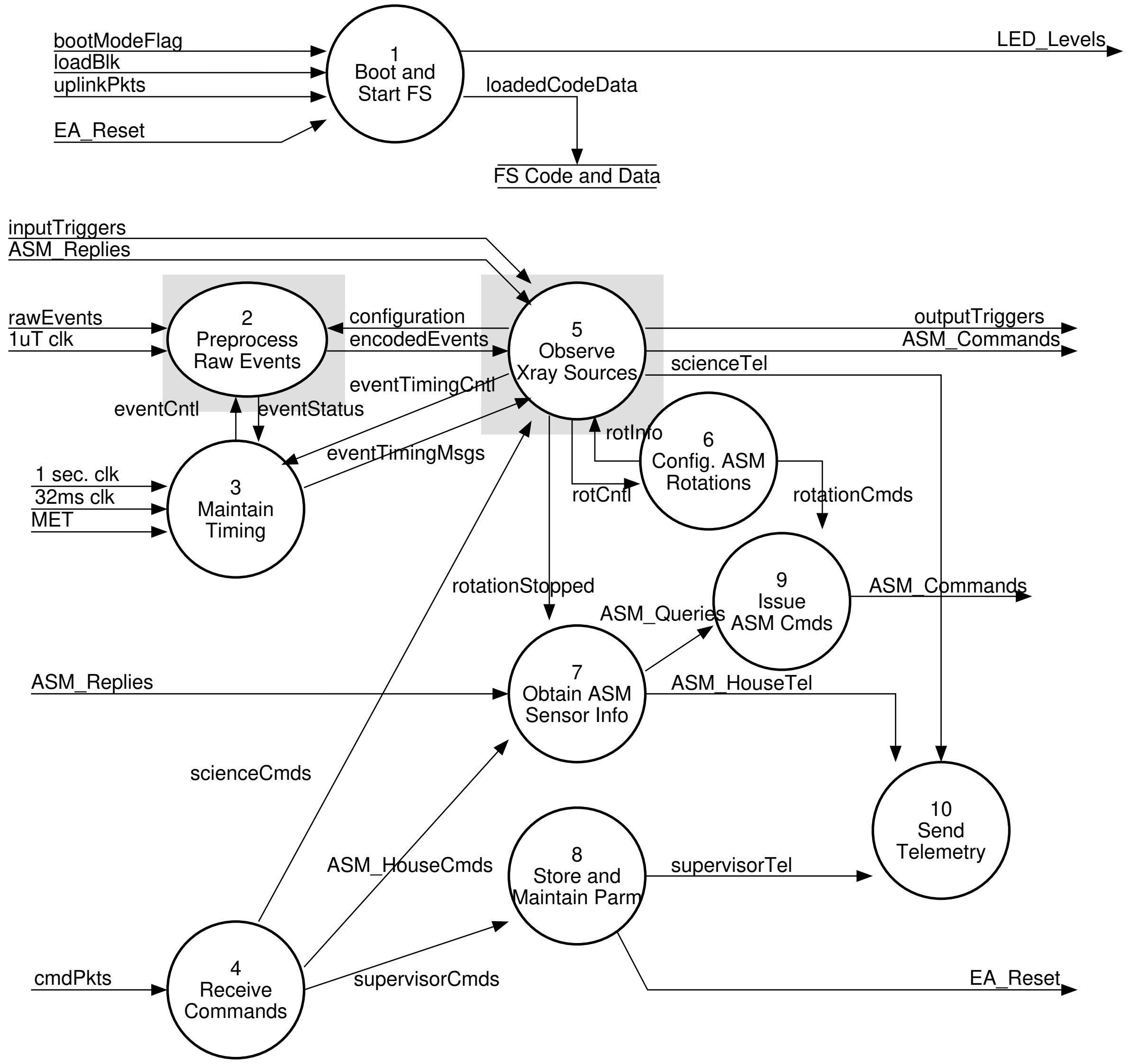

Figure 1 Module Perspective

1.5 Document Overview

Sections 1 and 2 provide an overview of the Typer Builder Module, summarizing the purpose, services, context, and interfaces of the module. Section 3 describes all software structures and constants visible to modules outside of the Typer Builder Module. Section 4 describes some of the details of the Encoded Typer Table format. Section 5 describes the functions provided to other modules by the Typer Builder Module. Section 6 describes some of the implementation details of the module. Section 7 describes which requirements are met by the module. Section 8 describes what types of tests need to be performed to ensure that the implementation of the module is operating properly. Appendix A lists items which are currently undefined, or that need to be reviewed.

1.6 Acronyms

Please refer to the **Flight Software Global Design Specification**, Section 1.5, for a list of acronyms and abbreviations. In addition to the acronyms listed in the Global Design document, this document uses the following:

LLD .... Low-level Discriminator
VLE .... Very-large Event
CD .... Encoded Typer Table Classification Directive

2 Overview

2.1 Purpose

The purpose of the Typer Builder Module is to decode compressed Typer Tables. The event processing hardware uses the Typer Table to select and classify events. The Typer Builder Module defines the encoding format of a compressed table, and provides a single function which decodes a compressed table into its final form.

2.2 Services

The Typer Builder Module provides the following services:

1) Load hardware Typer Table .... The Typer Builder Module is responsible for decoding an Encoded Typer Table into the Front End hardware Typer Lookup Table.

2) Provide Length of Encoded Table..... The Typer Builder Module is responsible for providing the length, in bytes, of an Encoded Typer Table to the rest of the Flight Software.

3) Check hardware Typer Table .... The Typer Builder Module is responsible for checking the integrity of the Front End's hardware Typer Lookup Table, using an Encoded Typer Table for comparison.

2.3 Module Context

The following illustrates the Typer Builder Module relationships with the rest of the Flight Software residing in an EA:

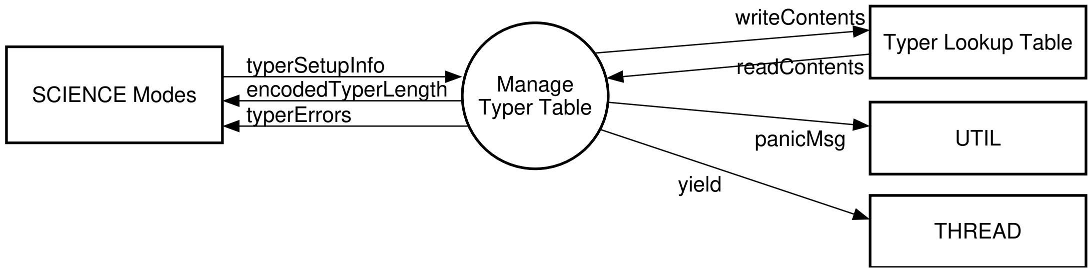

Figure 2 Typer Builder Module Context Diagram

The Science Modes (see **Science Thread Module Specification**) use the Typer Builder Module to load the contents of the Front End's hardware Typer Lookup Table, using an Encoded Typer Table (typerSetupInfo). If the Typer Builder Module detects an error in the specified Encoded Typer Table, it issues a panic message (panicMsg) and reset the EA using the **Utility Module**. The Science Modes also use the Typer Builder Module to obtain the length, in bytes, of the Encoded Typer Table (encodedTyperLength), and to test the integrity of the contents of the Front End hardware's Typer Lookup Table (typerErrors).

When building the Typer Table, the Typer Builder Module writes (writeContents) directly to the Front End's Typer Lookup Table. When the Typer Builder Module checks the integrity of the lookup table, it reads (readContents) the contents of the Front End's Typer Lookup Table directly.

Since the time to build a Typer Lookup Table varies depending on the commanded configuration, the Typer Builder Module uses the **Thread Manager Module** to periodically yield control (yield) during its decoding process.

2.4 Theory of Operation

Typer Table Operation

The hardware uses the Typer Table to map incoming event words to a classification word, and a recoder table select. The table may also tell the hardware to disable the time-tag on an event by event basis, or to discard certain events. The classification word is combined with other information (such as time bin, pulse-height, etc.) to form a classified event word, which is used in later processing. The recoder select is used to control how the pulse height of the event is recoded.

The following diagram illustrates the inputs and outputs of the event word classification process:

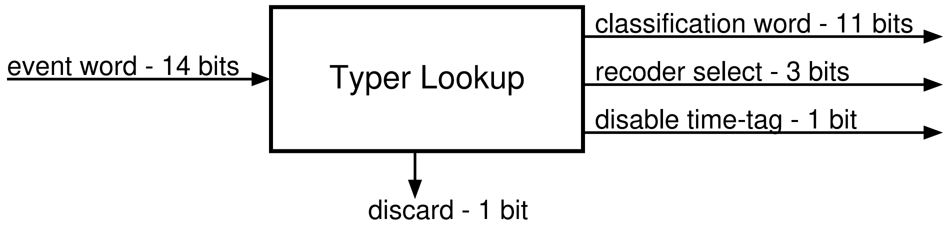

Figure 3 Typer Event Classification

Typer Table Encoding

Typer tables are compressed by the ground and uplinked to an EA's memory. Before an observation starts, the Flight Software uses the Typer Builder Module to decode the uplinked table into the Front End's hardware Typer Lookup Table (see **XTE EDS Hardware Functional Description and Requirements**, "FREGS and Observation Control Registers : Lookup Table RAM").

Typer Table encoding and decoding are very closely tied to the contents of the input and outputs of the Typer Lookup operation, and are described in detail in Section 4.

2.5 Data Structures and Constants Summary

The Typer Builder Module defines the structure of a compressed table. It also uses the typer table definition provided by the hardware.

2.6 Function Summary

The Typer Builder Module defines the following function:

TYPERbuild()	Decodes a compressed typer table.
TYPERencoded_length()	Returns byte length of an Encoded Typer Table
TYPERcheck()	Tests the contents of the Front End's hardware Typer Lookup Table against a passed Encoded Typer Table.

3 Data Structures and Constants

3.1 Hardware

3.1.1 Typer Table (0x040000 flat-address)

This is the lookup table used by the Front End PCA hardware to classify or discard PCA event words. The hardware discards any events which are so tagged by the table. The table contains 16,384 16-bit words (see **XTE EDS Hardware Functional Description and Requirements**, Section "FREGS and Observation Control : Lookup Table RAM" and "Figure 6 : PCA Event Processing").

3.1.2 PCA Typer Input Event Word (Table Address, 14-bits)

The hardware uses this event word as an address into the Typer Table. The format of the input event word is as follows:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	**D2**	**D1**	**D0**	**R3**	**L3**	**R2**	**L2**	**R1**	**L1**	**Vh**	**Vl**	**VLE**	**P**	**Cal**

Cal	Bit 0	Alpha Calibration Flag
Propane	Bit 1	Propane Event Flag
VLE	Bit 2	Very Large Event Flag
Vh-Vl	Bits 3,4	Xenon Veto Layer Pulse Height
[L or R]#	Bits 5 - 10	Xenon Low Level Discriminators (LLD)
D0-D2	Bits 11-13	Detector Id

Figure 4 Typer Input Event Word (table address)

The following lists and describes all of the input word's fields:

Detector ID (3-bits) .... This identifies the detector which produced the event. There are only 5 detectors in the system. Values 0 through 4 correspond to the detectors 1 through 5 respectively.

Xenon LLD (6-bits) .... Each bit corresponds to one of 6 low-level discriminators (LLDs) monitoring separate sets of anodes within the Xenon gas volume. The "L" and "R" designations refer to alternate wires in an anode plane. The numbers refer to anode layers, "1" being closest to the entrance window.

Veto Layer Pulse Height (2-bits)...... This identifies the energy of an event in the Xenon Veto layer.

Alpha Calibration Flag (1-bit) .... This indicates that the event was generated by the Calibration source.

Propane Flag (1-bit) .... This indicates that an event was detected in the Propane Layer.
VLE Flag (1-bit) .... This indicates that a very large, saturating event occurred in a Xenon or Propane Layer.

For more detail, refer to the **XTE EDS Hardware Functional Description and Requirements** document, "Figure 6 : PCA Event Processing".

3.1.3 Typer Table Contents Word (16-bits)

Every incoming event addresses a unique location in the Typer Table. The values stored at these locations have the following format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D	**R2**	**R1**	**R0**	**t10**	**t9**	**t8**	**t7**	**t6**	**t5**	**t4**	**t3**	**t2**	**t1**	**t0**	**T**

T	Bit 0	Disable Time Tag
t0-t10	Bits 1-11	Event Classification
R0,R1,R2	Bits 12,13,14	Recoder Table Selection
D	Bit 15	Discard Event

Figure 5 Typer Table Contents

The "Disable Time Tag" bit, when set, disables the time tag in the final tagged, recoded event. The "Event Classification" bits map the input event to one of a possible 2048 event classes. The "Recoder Table Selection" bits are inputs to the recoder table. The "Discard Event" bit, when set, causes the hardware to discard the current event.

When the hardware processes the "Event Classification" bits, it shifts the bits (t0-t10) left by 2 bits, and "or"s the result with the output of the Recoder Table, the time-stamp, and other information.

For more detail, refer to the **XTE EDS Hardware Functional Description and Requirements** document, "Figure 6 : PCA Event Processing".

3.2 Encoded Typer Table Structure

An encoded Typer Table consists of a Description Header followed by one or more Classification Directive (CD) entries. The details of the encoding algorithm are described in Section 4.

3.2.1 Description Header

An encoded typer table starts with a Description Header. The structure of this header is as follows:

[number of CD entries ...] .... # of Classification Directive Entries (word, little-endian)

3.2.2 Classification Directive Entry

An encoded typer table contains one or more Classification Directive Entries. The structure of these entries is as follows:

[table contents base] .... Base word to use at selected addresses (word, little-endian)
[match pattern ...] .... Selects table addresses (4-bytes)

3.2.2.1 Table Contents Base

The Typer Builder Module "or's" this word with any pass-through information (see Sections 3.2.2.2 and 4.3) and stores the results in the appropriate table locations. Its format is identical to the Typer Table Contents word described in Section 3.1.3.

3.2.2.2 Match Pattern

A Match Pattern contains a set of fields which are used by the decoding algorithm to select zero or more table addresses. It also contains three bit-position values which the decoding algorithm uses, in conjunction with the Table Contents Base (see Section 3.2.2.1 and 4.3), to form the set of values to store in the selected table addresses. The following table illustrates the format of a Match Pattern (see Section 4.2 for more detail) :

	Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
Byte 0	LLD Match Code		LLD Match Bitmap					
			R3	*L3*	*R2*	*L2*	*R1*	*L1*
Byte 1	Special Match Code			Special Match Bitmap				
	Ignore	Or	Not	*Veto H.*	*Veto L.*	*VLE*	*Propane*	*Cal*
Byte 2	LLD Code Pass-through Bit Position				**Spare**	Detector Code		
Byte 3	Veto Energy Pass-through Bit Position				VLE/Prop./Cal. Pass-through Bit Position			

Table 1 Match Pattern Layout

The LLD Match Codes are defined as follows (see Section 4.2.2 for more detail):

TYPER_LLD_CODE_EXACT......................... Match LLD Bitmap exactly
TYPER_LLD_CODE_ONE_BIT .... One and only one matching LLD asserted
TYPER_LLD_CODE_TWO_OR_MORE........ Two or more matching LLDs asserted
TYPER_LLD_CODE_ONES .... Match ones in Bitmap exactly, zeros in the Bitmap are wild.

The bit-field combinations of the Special Match Code are defined as follows (see Section 4.2.3 for more detail):

TYPER_SPEC_CODE_EXACT....................... (000)........ Match Special Bitmap exactly
TYPER_SPEC_CODE_NOT .... (001)........ Match all but Special Bitmap
TYPER_SPEC_CODE_OR............................... (010)........ Any 1 matches, and all 0s match
TYPER_SPEC_CODE_NOT_OR .... (011)........ All 1s mismatch, or any 0 mismatch
TYPER_SPEC_CODE_IGN .... (100)........ All 1s match, 0s are wild
TYPER_SPEC_CODE_NOT_IGN.................. (101)........ Any 1 mismatch, 0s are wild
TYPER_SPEC_CODE_OR_IGN .... (110)........ Any 1 match, 0s are wild
TYPER_SPEC_CODE_NOT_OR_IGN .... (111)........ No 1s match, 0s are wild

The Detector Codes are defined as follows (see Section 4.2.1 for more detail):

TYPER_MATCH_DET_PCU_1....................... (1)............ Match Detector Id 0 (PCU 1)
TYPER_MATCH_DET_PCU_2....................... (2)............ Match Detector Id 1 (PCU 2)
TYPER_MATCH_DET_PCU_3....................... (3)............ Match Detector Id 2 (PCU 3)
TYPER_MATCH_DET_PCU_4....................... (4)............ Match Detector Id 3 (PCU 4)
TYPER_MATCH_DET_PCU_5....................... (5)............ Match Detector Id 4 (PCU 5)
TYPER_MATCH_DET_PCU_ALL................. (7)............ Match all Detector Ids (PCUs 1-5)
NOTE: There is no way for a Match Pattern to

select Typer Table addresses which do not correspond to a PCU (i.e. detector ids 5, 6, and 7).

The Pass-through Position values range from 0 to 15 and indicate the bit-position of the respective field within the Typer Contents Word's Event Classification/Recoder Selection bits (bits 1-14, see Section 3.1.3). A value of 15 indicates that the field should not be passed through to the contents word (i.e. shifted off the end of the contents word). For example, if the Veto Position value is 0, then the Veto bits will appear in bits 1 and 2 of the Typer Table Contents Word, and, therefore, in bits 0 and 1 of Typer's Output Word.

3.3 Constants

3.3.1 TYPER_CHECK_DEFAULT (0x0ffff)

This defines the default value to use for the Typer Table error count, offset and value (see TYPERcheck(), Section 5.3).

3.3.2 TYPER_NUM_DETECTORS (5)

This constant defines the number of PCUs recognized by the EDS.

3.3.3 TYPER_NUM_DETECTOR_IDS (8)

This constant defines the number of possible Detector Ids supported by the 3 detector id bits passed into the Typer Table.

3.3.4 TYPER_NUM_LLDS (6)

This constant defines the number of Low Level Discriminators (LLDs) in a given PCU.

3.3.5 TYPER_NUM_LLD_COMBOS (64)

This constant defines the total number of possible LLD bit combinations, given the number of LLDs (see Section 3.3.4).

3.3.6 TYPER_NUM_SPECIAL_BITS (5)

This constant defines the total number of possible Special Flag bits passed into the Typer Table (Veto Energy (2 bits), VLE (1 bit), Propane Flag (1 bit) and Alpha Calibration Flag (1 bit)).

3.3.7 TYPER_NUM_SPECIAL_COMBOS (32)

This constant defines the total number of possible combinations of Special Flag bits, given the number of Special Flag bits (see Section 3.3.6).

3.3.8 TYPER_ADDR_CALIB_BIT (0)

This constant defines the bit position of the Alpha Calibration Flag within the address inputs to the Typer Table.

3.3.9 TYPER_ADDR_PROPANE_BIT (1)

This constant defines the bit position of the Propane Flag within the address inputs to the Typer Table.

3.3.10 TYPER_ADDR_VLE_BIT (2)

This constant defines the bit position of the Very Large Event Flag within the address inputs to the Typer Table.

3.3.11 TYPER_ADDR_VETO_LOW_BIT (3)

This constant defines the bit position of the low bit of the Veto Layer Energy within the address inputs to the Typer Table.

3.3.12 TYPER_ADDR_VETO_HIGH_BIT (4)

This constant defines the bit position of the high bit of the Veto Layer Energy within the address inputs to the Typer Table.

3.3.13 TYPER_ADDR_VETO_MASK (0x0018)

This value, when "and"ed with a Typer Table address, masks off all non-Veto Energy bits within the given address.

3.3.14 TYPER_ADDR_VETO_SHIFT (3)

This value defines the number of bits by which to right shift a Typer Table address to place the Veto Energy bits at bit position 0.

3.3.15 TYPER_ADDR_SPECIAL_MASK (0x001f)

This value, when "and"ed with a Typer Table address, masks off all non-Special Flag bits within the given address.

3.3.16 TYPER_ADDR_SPECIAL_SHFT (0)

This value defines the number of bits by which to right shift a Typer Table address to place the first of the Special Flag bits at bit position 0.

3.3.17 TYPER_ADDR_LLD_MASK (0x007e0)

This value, when "and"ed with a Typer Table address, masks off all non-LLD bits within the given address.

3.3.18 TYPER_ADDR_LLD_SHFT (5)

This value defines the number of bits by which to right shift a Typer Table address to place the first of the LLD bits at bit position 0.

3.3.19 TYPER_ADDR_DET_ID_MASK (0x03800)

This value, when "and"ed with a Typer Table address, masks off all non-Detector Id bits within the given address.

3.3.20 TYPER_ADDR_DET_ID_SHFT (11)

This value defines the number of bits by which to right shift a Typer Table address to place the first of the Detector Id bits at bit position 0.

3.3.21 TYPER_CONT_TIME_DISABLE_MASK (0x00001)

This mask selects the Time-Tag disable bit within the Typer Table Contents word.

3.3.22 TYPER_CONT_EVENT_CLASS_MASK (0x00ffe)

This mask selects the bits within the Typer Table Contents word that correspond to the Event Classification value output by the Typer Lookup Table.

3.3.23 TYPER_CONT_EVENT_CLASS_SHFT (1)

This value defines the number of bits by which to right shift a Typer Table Contents word to place the first of the Event Classification bits at bit position 0.

3.3.24 TYPER_CONT_RECODER_SEL_MASK (0x07000)

This mask selects the bits within the Typer Table Contents word that correspond to the Recoder Selection value output by the Typer Lookup Table.

3.3.25 TYPER_CONT_RECODER_SEL_SHFT (12)

This value defines the number of bits by which to right shift a Typer Table Contents word to place the first of the Recoder Selection bits at bit position 0.

3.3.26 TYPER_CONT_DISCARD_MASK (0x08000)

This mask selects the Event Discard bit within the Typer Table Contents word.

3.3.27 TYPER_CONT_OUTPUT_MASK (0x07fff)

This mask selects all bits except the Event Discard bit within the Typer Table Contents word.

3.3.28 TYPER_MATCH_DETECTOR_MASK (0x007)

This mask selects the detector code within byte 2 of a Classification Directive's match pattern.

3.3.29 TYPER_MATCH_DET_PCU_1 (1)

This is the detector code, in a Classification Directive's match pattern, which selects Typer Table addresses which correspond to PCU 1 (detector id 0).

3.3.30 TYPER_MATCH_DET_PCU_2 (2)

This is the detector code, in a Classification Directive's match pattern, which selects Typer Table addresses which correspond to PCU 2 (detector id 1).

3.3.31 TYPER_MATCH_DET_PCU_3 (3)

This is the detector code, in a Classification Directive's match pattern, which selects Typer Table addresses which correspond to PCU 3 (detector id 2).

3.3.32 TYPER_MATCH_DET_PCU_4 (4)

This is the detector code, in a Classification Directive's match pattern, which selects Typer Table addresses which correspond to PCU 4 (detector id 3).

3.3.33 TYPER_MATCH_DET_PCU_5 (5)

This is the detector code, in a Classification Directive's match pattern, which selects Typer Table addresses which correspond to PCU 5 (detector id 4).

3.3.34 TYPER_MATCH_DET_PCU_ALL (7)

This is the detector code, in a Classification Directive's match pattern, which selects Typer Table addresses which correspond to any PCU (detector ids 0-4).

3.3.35 TYPER_MATCH_LLD_CODE_MASK (0x0c0)

This mask selects the LLD Match code in byte 0 of a Classification Directive's match pattern.

3.3.36 TYPER_MATCH_LLD_MAP_MASK (0x03f)

This mask selects the LLD Bitmap in byte 0 of a Classification Directive's match pattern.

3.3.37 TYPER_LLD_CODE_EXACT (0x000)

This LLD Match code selects Typer Table Addresses whose LLD field is the same as the LLD Bitmap specified in the directive.

3.3.38 TYPER_LLD_CODE_ONE_BIT (0x040)

This LLD Match code selects Typer Table Addresses whose LLD field contains a single bit set to 1, which is also set to 1 in the LLD Bitmap.

3.3.39 TYPER_LLD_CODE_TWO_OR_MORE (0x080)

This LLD Match code selects Typer Table Addresses whose LLD field contains two or more bits which are set to 1, and which are also set to 1 in the LLD Bitmap.

3.3.40 TYPER_LLD_CODE_ONES (0x0c0)

This LLD Match code selects Typer Table Addresses whose LLD field contains 1s where ever there are 1s in the LLD Bitmap. The LLD field may have any value where there are 0s in the LLD Bitmap (i.e. a bitmap of 0 selects all LLD combinations).

3.3.41 TYPER_MATCH_SPECIAL_IGN_FLG (0x080)

This mask selects the Ignore Flag in the Special Match Code in byte 1 of a Classification Directive's match pattern.

3.3.42 TYPER_MATCH_SPECIAL_OR_FLG (0x040)

This mask selects the Or Flag in the Special Match Code in byte 1 of a Classification Directive's match pattern.

3.3.43 TYPER_MATCH_SPECIAL_NOT_FLG (0x020)

This mask selects the Not Flag in the Special Match Code in byte 1 of a Classification Directive's match pattern.

3.3.44 TYPER_MATCH_SPECIAL_CODE_MASK (0x0e0)

This mask selects all bits in the Special Match Code in byte 1 of a Classification Directive's match pattern.

3.3.45 TYPER_MATCH_SPECIAL_MAP_MASK (0x01f)

This mask selects all bits in the Special Match Bitmap in byte 1 of a Classification Directive's match pattern.

3.3.46 TYPER_SPEC_CODE_EXACT (0x000)

This Special Match code (Ign = 0, Or = 0, Not = 0) selects Typer Table Addresses whose Veto/VLE/Propane/Calibration bits are equal to those in the directive's Special Match bitmap.

3.3.47 TYPER_SPEC_CODE_NOT (0x020)

This Special Match code (Ign = 0, Or = 0, Not = 1) selects any Typer Table Addresses whose Veto/VLE/Propane/Calibration bits are not equal to those in the directive's Special Match bitmap.

3.3.48 TYPER_SPEC_CODE_OR (0x040)

This Special Match code (Ign = 0, Or = 1, Not = 0) selects any Typer Table Addresses whose Veto/VLE/Propane/Calibration are zero where there are zeros in the directive's Special Match bitmap and which has any 1 which matches the bitmap.

3.3.49 TYPER_SPEC_CODE_NOT_OR (0x060)

This Special Match code (Ign = 0, Or = 1, Not = 1) selects any Typer Table Addresses whose Veto/VLE/Propane/Calibration has a 1 where there is a zero in the directive's Special Match bitmap or which does not have any 1s which match the bitmap.

3.3.50 TYPER_SPEC_CODE_IGN (0x080)

This Special Match code (Ign = 1, Or = 0, Not = 0) selects any Typer Table Addresses whose Veto/VLE/Propane/Calibration has 1s where ever there are 1s in the directive's Special Match bitmap (0s are ignored).

3.3.51 TYPER_SPEC_CODE_NOT_IGN (0x0a0)

This Special Match code (Ign = 1, Or = 0, Not = 1) selects any Typer Table Addresses whose Veto/VLE/Propane/Calibration does not have a 1s where ever there are 1s in the directive's Special Match bitmap (0s are ignored).

3.3.52 TYPER_SPEC_CODE_OR_IGN (0x0c0)

This Special Match code (Ign = 1, Or = 1, Not = 0) selects any Typer Table Addresses whose Veto/VLE/Propane/Calibration has a 1 where there's a 1 in the directive's Special Match bitmap (0s are ignored).

3.3.53 TYPER_SPEC_CODE_NOT_OR_IGN (0x0e0)

This Special Match code (Ign = 1, Or = 1, Not = 1) selects any Typer Table Addresses whose Veto/VLE/Propane/Calibration does not have a 1 where there's a 1 in the directive's Special Match bitmap (0s are ignored).

3.3.54 TYPER_MATCH_POS_LLD_MASK (0x0f0)

This mask selects the LLD Position field within byte 2 of the Classification Directive.

3.3.55 TYPER_MATCH_POS_LLD_SHFT (4)

This value defines the number of bits by which to right shift byte 2 of the Classification Directive to place the LLD Pass-through position bits at bit position 0.

3.3.56 TYPER_MATCH_POS_SPEC_MASK (0x00f)

This mask selects the Special Pass-through Position field within byte 3 of the Classification Directive. NOTE: The Veto bits are positioned separately (see Section 3.3.58).

3.3.57 TYPER_MATCH_POS_SPEC_SHFT (0)

This value defines the number of bits by which to right shift byte 3 of the Classification Directive to place the Special Pass-through position bits at bit position 0.

3.3.58 TYPER_MATCH_POS_VETO_MASK (0x0f0)

This mask selects the Veto Pass-through Position field within byte 3 of the Classification Directive.

3.3.59 TYPER_MATCH_POS_VETO_SHFT (4)

This value defines the number of bits by which to right shift byte 3 of the Classification Directive to place the Veto Pass-through position bits at bit position 0.

3.3.60 TYPER_SEL_END (0x0ff)

This value is used to terminate arrays of field selections (detector, LLD, special).

3.3.61 MSG_TYPER_DECODE_ENCODED_NULL

This message code is passed to UTILpanic() if the pointer to the Encoded Table passed to the Typer Module's decode routine, typer_decode() is NULL. The message should read something like:

"PANIC: ---typer_decode() -- NULL Encoded Table"
where %d is the Detector Match code that was contained in the Match Pattern.

3.3.62 MSG_TYPER_SELECT_DETECTOR

This message code is passed to UTILpanic() if the Detector Match Code in the Match Pattern passed to the internal function typer_select_detector() is invalid (0 or 6). The message should read something like:

"PANIC: ---typer_select_detector([%d]) -- Invalid Detector Match Code"
where %d is the Detector Match code that was contained in the Match Pattern.

3.3.63 MSG_TYPER_SELECT_LLDS

This message code is passed to UTILpanic() if the LLD Match Code in the Match Pattern passed to the internal function typer_select_llds() is invalid. The message should read something like:

"PANIC: ---typer_select_llds([%d]) -- Invalid LLD Match Code"
where %d is the LLD Match code that was contained in the Match Pattern.

3.3.64 MSG_TYPER_LLD_ONE_BIT

This message code is passed to UTILpanic() if the Match Pattern passed to the internal function typer_lld_one_bit() contained an LLD Match Bitmap which does not have any bits set (i.e. is zero). The message should read something like:

"PANIC: ---typer_lld_one_bit() -- LLD Bitmap is Zero"

3.3.65 MSG_TYPER_LLD_TWO_OR_MORE

This message code is passed to UTILpanic() if the Match Pattern passed to the internal function typer_lld_two_or_more() contained an LLD Match Bitmap which does not have two or more bits set (i.e. is 0, 1, 2, 4, 8, 16, or 32) or is greater than or equal to TYPER_NUM_LLD_COMBOS (see Section 3.3.5). The message should read something like:

"PANIC: ---typer_lld_two_or_more(%d) -- LLD Bitmap invalid"
where %d is the LLD Match Bitmap that was contained in the Match Pattern.

3.3.66 MSG_TYPER_SPECIAL_TEST

This message code is passed to UTILpanic() if the operator code passed to the internal typer_special_test() was invalid. This error cannot be generated by a bad Match Pattern and can only be generated by some code or data corruption in the EA. The message should read something like:

"PANIC: ---typer_special_test(%d) -- Bad Operator Code"
where %d is the Operator Code that was passed to typer_special_test().

4 Table Encoding

4.1 Decoding Algorithm

The table decoding algorithm starts by setting the entire Typer Table to discard all events. It then processes each Classification Directive (CD) entry in the encoded table. For each directive, it selects the Typer Table addresses specified by the Match Pattern, and fills the selected locations with the Table Contents Word specified in the directive. The directives are processed in the order in which they appear in the encoded table. Directives which match previously filled Typer Table locations will overwrite the previous values stored at the matched table locations. The following section describes how the Match Patterns select Typer Table addresses.

4.2 Table Address Selection

The Typer Table consists of an array of 16,384 table addresses. Each address corresponds to one possible Typer Input Event Word. Each input word to the Typer Table (table address) can be thought of as having 3 fields:

1) Detector Id (1 out of 5 possible detectors)
2) LLD Set (any combination of 6 On/Off inputs)
3) Special Events (1 out of a possible of 32 combinations)

The "Special Event" field is comprised of combinations of the following inputs:

1) Xenon Veto Layer Energy Level (None, Low, Medium or High)
2) Alpha Calibration Flag (On or Off)
3) Propane Flag (On or Off)
4) Very Large Event Flag (On or Off)

A Match Pattern selects a Typer Table address if the address "matches" all 3 of the pattern's input fields. A single Match Pattern can select more than one table address. The rules for matching the address against each field vary.

4.2.1 Detector Match

Description

The Match Pattern uses an encoded field to match Typer Table addresses associated with one or more detectors. If the field has a value from 1 to 5, then it matches addresses associated with the specified detector. If the field is set to 7, then it matches all addresses associated with any of the 5 detectors. Fields with a value of 0 or 6 are invalid, and will result in a panic message and EA reset.

Examples

Detector Field
0 .... Illegal value.
1 .... Match only addresses associated with detector 1.
2 .... Match only addresses associated with detector 2.
3 .... Match only addresses associated with detector 3.
4 .... Match only addresses associated with detector 4.
5 .... Match only addresses associated with detector 5.
6 .... Illegal value.
7 .... Match addresses associated with any of the 5 detectors.

4.2.2 LLD Match

Description

The Match Pattern has an LLD match entry, which consists of two parts. The first part contains an LLD bitmap, where each bit corresponds to one LLD. The second part is a code which governs how the bitmap is to be interpreted. The code can have one of the following values:

0 .... Match only addresses which exactly match the LLD bitmap. The LLD is On where there is a one in the bitmap, and is Off where there is a zero.

1 .... Match addresses which have exactly one of the specified LLDs set. The LLDs are specified by ones in the bitmap. LLDs corresponding to zeros in the bitmap must be Off.

2 .... Match addresses which have two or more of the specified LLDs set. The LLDs are specified by ones in the bitmap. LLDs corresponding to zeros in the bitmap must be Off.

3 .... Match addresses which have all of the specified LLDs set. The LLDs are specified as ones in the bitmap. LLDs corresponding to zeros in the bitmap may be On or Off.

Examples

Code	L1	R1	L2	R2	L3	R3	
0	0	0	0	0	0	0	Match only addresses where all LLDs are Off.
0	0	1	0	0	1	0	Match only addresses where R1 and L3 are On. All other LLDs must be Off.
1	1	1	0	0	1	1	Match only addresses where **either** L1, R1, L3, or R3 are On. Only one of the LLDs may be On. All other LLDs must be Off.
2	1	1	0	0	1	1	Match only addresses where **two or more** of the following LLDs are On; L1, R1, L3, R3. All other LLDs must be Off.
3	1	1	0	0	1	1	Match all addresses where L1, R1, L3, and R3 are On. L2 and R2 may be On or Off.
3	0	0	0	0	0	0	Match all LLD addresses.

4.2.3 Special Event Match

Description

The Match Pattern has a four field match entry to match Typer Table addresses associated with "special events." The first three fields are operator codes. The fourth field is a "special event" bitmap. The three operator codes govern how the "special event" bitmap is interpreted. Addresses may match only when their "special event" inputs conform to the requirements specified by the bitmap and operator codes. The four fields are as follows:

Ignore Flag .... If clear, then only select addresses whose inputs corresponding to the zeros in the bitmap are Off.
If set, ignore the state of the inputs corresponding to the zeros in the bitmap.

Or Flag .... If clear, then only select addresses in which all inputs corresponding to the ones in the bitmap are On.
If set, then select only addresses in which one or more of the inputs corresponding to the ones in the bitmap are On.

Not Flag .... If clear, only match the selected addresses.
If set, only match unselected addresses.

Special Event BitMap .... This consists of the following bit fields:
(A) .... Alpha Calibration Flag
(P) .... Propane Flag
(L) .... VLE Flag
(Vh) .... Veto Height, most significant bit
(Vl) .... Veto Height, least significant bit

Examples

I	O	N		Vh	Vl	L	P	A		Logic	Description
0	0	0		0	0	1	0	0		(!Vh & !Vl & L & !P & !A)	Match VLE-only events.
0	0	1		0	0	1	0	0		!(!Vh & !Vl & L & !P & !A)	Match all except VLE-only events.
0	1	0		1	1	0	0	0		((Vh | Vl) & !L & !P & !A)	Match veto-only events.
0	1	1		1	1	0	0	0		!((Vh | Vl) & !L & !P & !A)	Match all except veto-only events.
1	0	0		1	1	0	0	0		(Vh & Vl & 1)	Match any high energy veto events.
1	0	1		1	1	0	0	0		!(Vh & Vl & 1)	Match all except high energy veto events.
1	1	0		1	1	0	0	0		((Vh | Vl) & 1)	Match any veto event.
1	1	1		1	1	0	0	0		!((Vh | Vl) & 1)	Match any non-veto event.
0	0	0		0	0	0	0	0		(!Vh & !Vl & !L & !P & !A)	Match events with no Veto, VLE, Propane or Alpha Calibration.
1	0	0		0	0	0	0	0		(1)	Match all events.

4.3 Table Contents Generation

This section describes how the Typer Builder Module forms Typer Table Contents words (see Section 3.1.3). For each Typer Table address selected by a given directive's Match Pattern (see Section 4.2), the Typer Builder Module shifts the selected address's LLD, Veto Energy and VLE/Propane/Alpha Calibration fields left by the amount specified in the directive's Match Pattern plus 1 (i.e. the extra 1 is to position the field relative to the Event Classifcation bits), and "or's" the result with the directive's Table Contents Base (NOTE: The VLE, Propane Flag, and Alpha Calibration bits are treated as a single field). The module then stores the result in the addressed location. This allows the ground to use a single Match Pattern to generate a series of unique Contents Words by passing through certain fields of interest.

Example

This example will generate a pass-through lookup table for all valid Detector entries to the Event Classification bits in the Output Word. It uses 5 Match Patterns, each of which select all LLD, and Special Code combinations for a single detector. Each specify a Contents Base value which disables the Time-Tag and which explicitly positions the Detector Id within the stored contents word.

	Base	**LLD Bit-map**	**LLD Code**	**Spec. Bitmap**	**Spec. Code**	**Detector Code**	**LLD Pass-through**	**Spec. Pass-thorugh**	**Veto Pass-thorugh**
Directive 0	0x0001	000000	Match all 1s, ignore 0s	00000	Ignore Only	1	5	0	3
Directive 1	0x1001	000000	Match all 1s, ignore 0s	00000	Ignore Only	1	5	0	3
Directive 2	0x2000	000000	Match all 1s, ignore 0s	00000	Ignore Only	1	5	0	3
Directive 3	0x3001	000000	Match all 1s, ignore 0s	00000	Ignore Only	1	5	0	3
Directive 4	0x4001	000000	Match all 1s, ignore 0s	00000	Ignore Only	1	5	0	3

Table 2 Contents Generation Example

5 Functions

5.1 TYPERbuild() - Decompress a Compressed Typer Table

Usage

void TYPERbuild (UINT8 **encoded*);

Description

TYPERbuild() expands an encoded Typer Table, pointed to by "*encoded*", directly into the Front End hardware's Typer Lookup Table. This function performs some sanity checking on the data pointed to by "*encoded*".

If an error is encountered while decoding the Encoded Typer Table, the decoding functions called by TYPERbuild() will issue a panic message and reset the EA, using the **Utility Module** function UTILpanic(). The following lists the possible error messages:

MSG_TYPER_DECODE_ENCODED_NULL ...... "*encoded*" is NULL.
MSG_TYPER_SELECT_DETECTOR .... Illegal Detector Match Code.
MSG_TYPER_SELECT_LLDS .... Illegal LLD Match Code.
MSG_TYPER_LLD_ONE_BIT .... Bitmap is zero.
MSG_TYPER_LLD_TWO_OR_MORE .... Bitmap is invalid.
MSG_TYPER_SPECIAL_TEST .... Illegal Special Match Code.

Return Values

NONE - FUNCTION IS VOID.

Caveats

1) "*encoded*" must point to data which conforms to the structure described in Section 3.2.

2) The DSP MUST be held in a reset state while calling this function.

See Also

Utility Module Specification

5.2 TYPERencoded_length() - Return Length of Encoded Table

Usage

UINT16 TYPERencoded_length (UINT8 **encoded*);

Description

TYPERencoded_length() computes and returns the length, in bytes, of the Encoded Typer Table, pointed to by "*encoded*".

Return Values

Length, in bytes, of Encoded Typer Table pointed to by "*encoded*"

Caveats

1) "*encoded*" must point to data which conforms to the structure described in Section 3.2.

See Also

NONE.

5.3 TYPERcheck() - Check Integrity of Front End's Typer Table

Usage

UINT16 TYPERcheck(UINT8 **encoded*, UINT16 **err_offset*, UINT16 **err_val*);

Description

TYPERcheck() decodes the Encoded Typer Table, pointed to by "*encoded*" into the Science Buffer, and performs a word-by-word comparison of the newly decoded table with the Front End hardware's Typer Lookup Table. If a discrepancy is found, TYPERcheck() fills the contents of "*err_offset*" with the word index of the hardware Typer Table that had the error, and fills the contents of "*err_val*" with the actual value discovered at that location. It then increments an error counter. If more errors are found, TYPERcheck() again increments the counter, without reporting the new offset and value which disagreed with the Encoded Typer Table. After the entire table has been checked, TYPERcheck() returns the value of the error counter.

If no errors are found, TYPER_CHECK_DEFAULT will be stored in the locations pointed to by "*err_offset*" and "*err_val*" and TYPERcheck() will return 0.

If an error is encountered while decoding the Encoded Typer Table, the decoding functions called by TYPERcheck() will issue a panic message and reset the EA, using the **Utility Module** function UTILpanic(). The following lists the possible error messages:

MSG_TYPER_DECODE_ENCODED_NULL ...... "*encoded*" is NULL.
MSG_TYPER_SELECT_DETECTOR .... Illegal Detector Match Code.
MSG_TYPER_SELECT_LLDS .... Illegal LLD Match Code.
MSG_TYPER_LLD_ONE_BIT .... Bitmap is zero.
MSG_TYPER_LLD_TWO_OR_MORE .... Bitmap is invalid.
MSG_TYPER_SPECIAL_TEST .... Illegal Special Match Code.

Return Values

The total number of discrepancies between the Encoded Typer Table and the Front End hardware's Typer Lookup Table.

Caveats

1) TYPERbuild() must be called before calling this function.

2) The DSP MUST be held in a reset state while calling this function.

3) The contents of the Science Buffer will be corrupted by calling this function.

See Also

Utility Module Specification

6 Detailed Design Considerations

6.1 Data Flow

The following diagram illustrates the level 1 data flow within the Typer Builder Module:

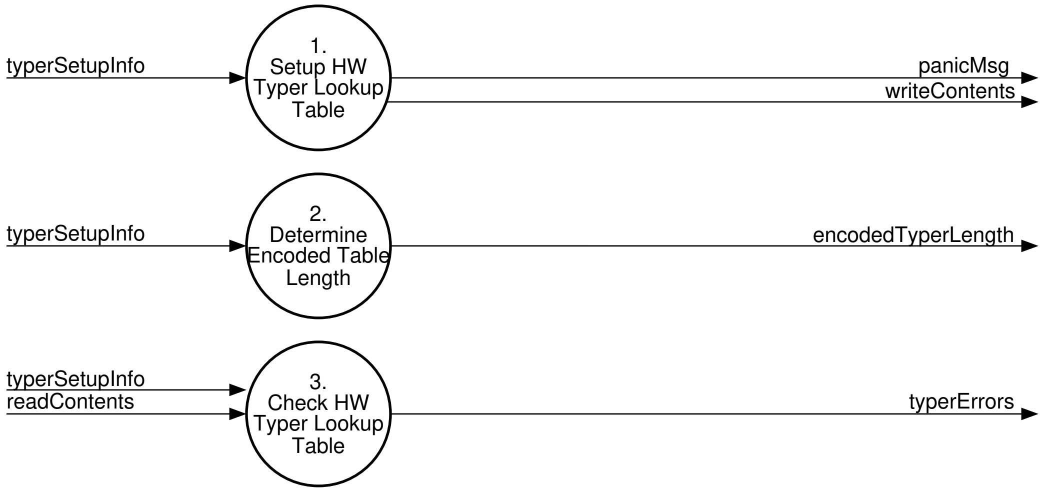

Figure 6 Typer Builder Module Level 1 Data Flow

Before the start of an observation, the Typer Builder Module is called to decode (bubble 1) an encoded Typer Table (typerSetupInfo) into the Front End Hardware's Typer Lookup Table (writeContents), and to determine (bubble 2) the length (encodedTyperLength), in bytes, of the encoded Typer Table (typerSetupInfo). If an error is encountered while building the Front End's Typer Table, the Typer Builder Module issues a panic message and resets the EA (panicMsg).

After the observation has terminated, the Typer Builder Module is called to test the integrity of the Front End's Typer Lookup Table (bubble 3). The Typer Builder Module decodes the encoded Typer Table (typerSetupInfo) into the Science Data Buffer (not shown) and then compares the buffer with the contents of the Front End's Typer Table (readContents). The Typer Builder Module then reports any differences (typerErrors) to the caller.

6.2 Control Considerations

6.2.1 Table Index Generation Algorithm

This section describes the top level algorithm used by the Typer Builder Module to generate indexes into the Typer Table using a given Match Pattern. An Encoded Typer Table consists of an array of Classification Directives. Each directive specifies a Table Contents Base to "or" with any pass-through information and then store into the Typer Table, and a Match Pattern which specifies which locations in the table are to receive the word, and the bit-positions of the pass-through information (see Section 3.2). The Typer Builder Module's decoding algorithm starts by initializing the table with a Contents Word which causes all X-ray events to be discarded. It then runs through the array of Classification Directives, processing each directive in turn. For each directive, the decoding algorithm generates a list of selected detectors, LLD bit combinations and Veto, VLE, Propane, and Alpha Calibration bit combinations. It then iterates through all combinations of the selected fields and forms a typer table index and contents word and stores the contents word in the selected table memory location. The decoding algorithm then proceeds to the next Classification Directive.

The Typer Table indexes are generated using the three components of the directive's Match Pattern; the Detector Match, LLD Match, and Special Event Match (see Sections 4.2.1, 4.2.2, and 4.2.3). Each component may specify one or more identifiers associated with a portion of the PCA Typer Input Event Word (see Section 3.1.2), which can then be used to generate an index into the Typer Table. The decoding algorithm consists of a set of four nested loops. The top loop iterates through each Classification Directive. The second loop iterates through all Detector Identifiers which are selected by the Detector Match portion of the current directive's Match Pattern. The third loop iterates through all LLD Identifiers which are selected by the LLD Match portion of the current directive's Match Pattern, and the fourth loop iterates through all Special Code Identifiers which are selected by the Special Code Match portion of the current directive's Match Pattern. Inside the inner-most loop, the decoding algorithm forms a Typer Table index and contents word, using the current Detector, LLD, Special Code Identifiers, the current directive's Table Contents Base and the Detector, LLD, Special Code and Veto pass-through bit positions. It then stores the formed word into the indexed Typer Table memory location.

The following diagram illustrates this algorithm:

- Initialize Typer Table to discard all events
- FOR each Classification Directive in the Encoded Typer Table
  - FOR each detector specified in the Detector Match
    - FOR each LLD Identifier specified in the LLDMatch
      - FOR each Special Code Identifier specified in the Special CodeMatch
        - generate a Typer Table Index from the Detector, LLD and Special Code Identifiers
        - form and store the contents word in the selected Typer Table location

Figure 7 Table Index Generation Algorithm

6.2.2 Detector Identifier Selection (max. 5 per Match Pattern)

This section describes the algorithm used by the Typer Builder Module to generate Detector Identifiers, used to index into the Typer Table, using a given Match Pattern. A Detector Identifier is a 3-bit code which represents the Detector Id bits as presented by the PCA Typer Input Event Word (see Section 3.1.2, bits 11-12). Match Patterns contain a 3-bit code (Detector Match, see Section 4.2.1) which can select locations associated with a single specified detector, or locations associated with any of the detectors. If the code specifies a specific detector, only the Detector Identifer associated with the specified detector is supplied. If the code specifies all of the detectors, all Detector Identifiers are generated.

6.2.3 LLD Identifier Selection (max. 64 per Match Pattern)

This section describes the algorithm used by the Typer Builder Module to generate LLD Identifiers, used to index into the Typer Table, using a given Match Pattern. An LLD Identifier is a 6-bit code which represents the LLD bits as presented by the PCA Typer Input Event Word (see Section 3.1.2, bits 5-10). A Match Pattern contains a 2-bit LLD Selection Code and a 6-bit LLD Bitmap (LLD Match, see Section 4.2.2). The LLD Selection Code is used to select LLD Identifers by comparing the Identifiers with the supplied LLD BitMap. The following describes each of the LLD Selection Codes:

0 .... Exact Match.
The LLD Identifier equals the LLD BitMap.

1 .... Only One Bit Set
The LLD Identifier has only one bit set to 1, and the corresponding bit in the LLD BitMap is also set to 1. Bits in the Bitmap which are 0 must also be 0 in the LLD Identifier.

2 .... Two or more Bits Set
The LLD Identifer has two or more bits set to 1, where the

corresponding bits in the LLD Bitmap are also set to 1. Bits in the Bitmap which are 0 must also be 0 in the LLD Identifier.

3 .... All Asserted Bits Match
The LLD Identifier has all bits set to 1 where the LLD Bitmap also has bits set to 1. Bits in the Bitmap which are 0 may be either 0 or 1 in the Identifier.

The first case (0) is easy, the selected LLD Identifier is the specified LLD Bitmap.

The second case (1) can be implemented by iterating through each LLD Identifier that has only one bit set (6 possible). Start with an LLD Identifier of 0x01 and "and" it with the specified LLD Bitmap. If the result isn't zero, then that LLD Identifier is selected. If it is zero, then that LLD Identifier is not selected. Then shift the Identifier left by one bit and try again. Perform the procedure until the Identifier is greater than the LLD Bitmap.

The third case (2) can be handled by iterating through all possible LLD combinations, skipping 0, 1, 2, 4, 8, 16 and 32, (57 combinations), and testing each case against the criteria. First, ensure that the Identifier doesn't have any bits set to 1 which are not also set in the LLD Bitmap. "Or" the Identifier with the Bitmap. If the result is not the same as the Bitmap, then the Identifier has bits set to 1 which are not set in the LLD Bitmap, and the Identifier cannot be selected. If the result is the same as Bitmap, the Identifier is selected. Then increment the LLD Identifier and try again. Perform the procedure until the Identifier is greater than the specified Bitmap (since all zero bits in Bitmap must also be zero in the Identifier, Identifiers greater than Bitmap won't match).

The fourth case (3) can be handled by iterating through all possible LLD combinations (64), testing each case against the criteria. Start with an LLD Identifier of 0. "And" the Identifier with the Bitmap and compare the result to the Bitmap. If the result is equal to the Bitmap, then the Identifier is selected. If the result is not equal to the Bitmap, then the Identifier is not selected. Then increment the Identifier and try again. Perform the procedure until the Identifier exeeds its maximum value (63).

6.2.4 Special Code Identifier Selection (max. 32 per Match Pattern)

This section describes the algorithm used by the Typer Builder Module to generate Special Code Identifiers, used to index into the Typer Table, using a given Match Pattern. A Special Code Identifier is a 5-bit code which represents the Special Code bits as presented by the PCA Typer Input Event Word (see Section 3.1.2, bits 0-4). A Match Pattern contains 3 single-bit Special Code Operators and a 5-bit Special Code Bitmap (Special Match, see Section 4.2.3). The Special Code Operators are used to select Special Code Identifers by comparing the Identifiers with the supplied Special Code BitMap. The following describes each of the Special Code Operators:

Ignore Flag .... If 0, then only select addresses whose inputs corresponding to the zeros in the bitmap are 0.
If 1, ignore the state of the inputs corresponding to the 0s in the bitmap.

Or Flag .... If 0, then only select addresses in which all inputs corresponding to the 1s in the bitmap are 1.
If 1, then select addresses in which one or more of the inputs corresponding to the 1s in the bitmap are 1.

Not Flag .... If 0, only match the selected addresses.
If 1, only match unselected addresses.

The operator flags can be thought of as forming an operator code, which can take on one of the following 8 values:

ION	Match Test Code	Match
000	0	Identifier which is equal to Bitmap (Identifier == Bitmap)
001	1	Identifiers which are not equal to Bitmap (Identifier != Bitmap)
010	2	Identifiers which have 0s where there are 0s in the Bitmap and which have any 1 where there is a 1 in Bitmap ((Identifier != 0) && (Identifier & ~Bitmap) == 0)
011	3	Identifiers which have a 1 where there is a 0 in the Bitmap or which don't have any 1s where there are 1s in the Bitmap. ((Identifier == 0) || (Identifier & ~Bitmap) != 0)
100	4	Identifiers which have all 1s where there are 1s in Bitmap ((Bitmap == 0) || (Identifier & Bitmap) == Bitmap)
101	5	Identifiers which don't have all 1s where there are 1s in Bitmap ((Bitmap != 0) && (Identifier & Bitmap) != Bitmap)
110	6	Identifiers which have any 1 where there's a 1 in Bitmap ((Identifier & Bitmap) != 0)
111	7	Identifiers which don't have any 1s where there are 1s in Bitmap ((Identifier & Bitmap) == 0)

The Special Code Identifier selection algorithm consists of combining the Special Code Operator Flags into a single Match Test Code, and by testing each possible Identifier (32) using the criteria specified by the code (see above).

6.2.5 Table Index Formation

This section describes how the Typer Builder Module forms a Typer Table Index based on a Detector Identifier, LLD Identifier, and Special Code Identifier.

Since each component Identifier is associated with a specific set of bits in the PCA Typer Input Event Word (see Sections 3.1.2, 6.2.2, 6.2.3, and 6.2.4), the Typer Table index is formed by shifting each component Identifier to the associated bit position, and "or"ing the selected Identifiers. The result is the Typer Table index associated with all of the selected Identifers.

6.2.6 Table Contents Formation

This section describes how the Typer Builder Module forms the final output word to store in a given Typer Table address using the Detector, LLD, Special Code, Veto bit pass-through positions, and Classication Directive's Contents Base. For each selected address, the detector id bits, LLD bits, Alpha Calibration/Propane/VLE bits and Veto bits are shifted left by the specified amounts. The results are then "or'd" together. The "or'd" result is then shifted left by 1 bit to position all of the fields to bit 0 relative to the event type/recoder selection bits in the contents word. The algorithm then masks off the top bit (reject event bit) and "or"s the result with the Classification Directive's Contents Base to form the final value to store in the selected Typer Table address.

NOTE: A shift value of 14 or 15 effectively disables pass-through for the specified field. A shift value of 0, places the field at bit 1 in the contents word (i.e. bit 0 in the actual word output by the Typer Lookup Table). The time-tag disable and event reject bits must be specified explicitly by the directive's Contents Base. By default, a location not selected by any directive will cause the corresponding events to be rejected.

6.3 Module Structure

6.3.1 Global Function Structures

This section describes the structure of the Typer Builder Module's globally accessible functions. Bolded boxes and text refer to function provided by modules other than the Typer Builder Module:

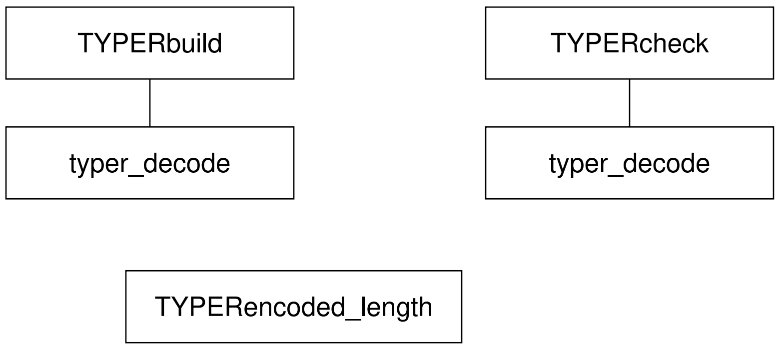

Figure 8 Typer Builder Module Global Function Structures

The following lists the functions shown in Figure 8. The details of each function's operation and arguments are described in the Typer Builder Module's detailed design document.

TYPERbuild .... Load Front End Typer Lookup Table
TYPERencoded_length .... Provide length (in bytes) of Encoded Typer Table
TYPERcheck .... Verify contents of Front End Typer Lookup Table

6.3.2 Internal Function Structures

This section describes the structure of the Typer Builder Module's internal functions. Bolded boxes and text refer to function provided by modules other than the Typer Builder Module:

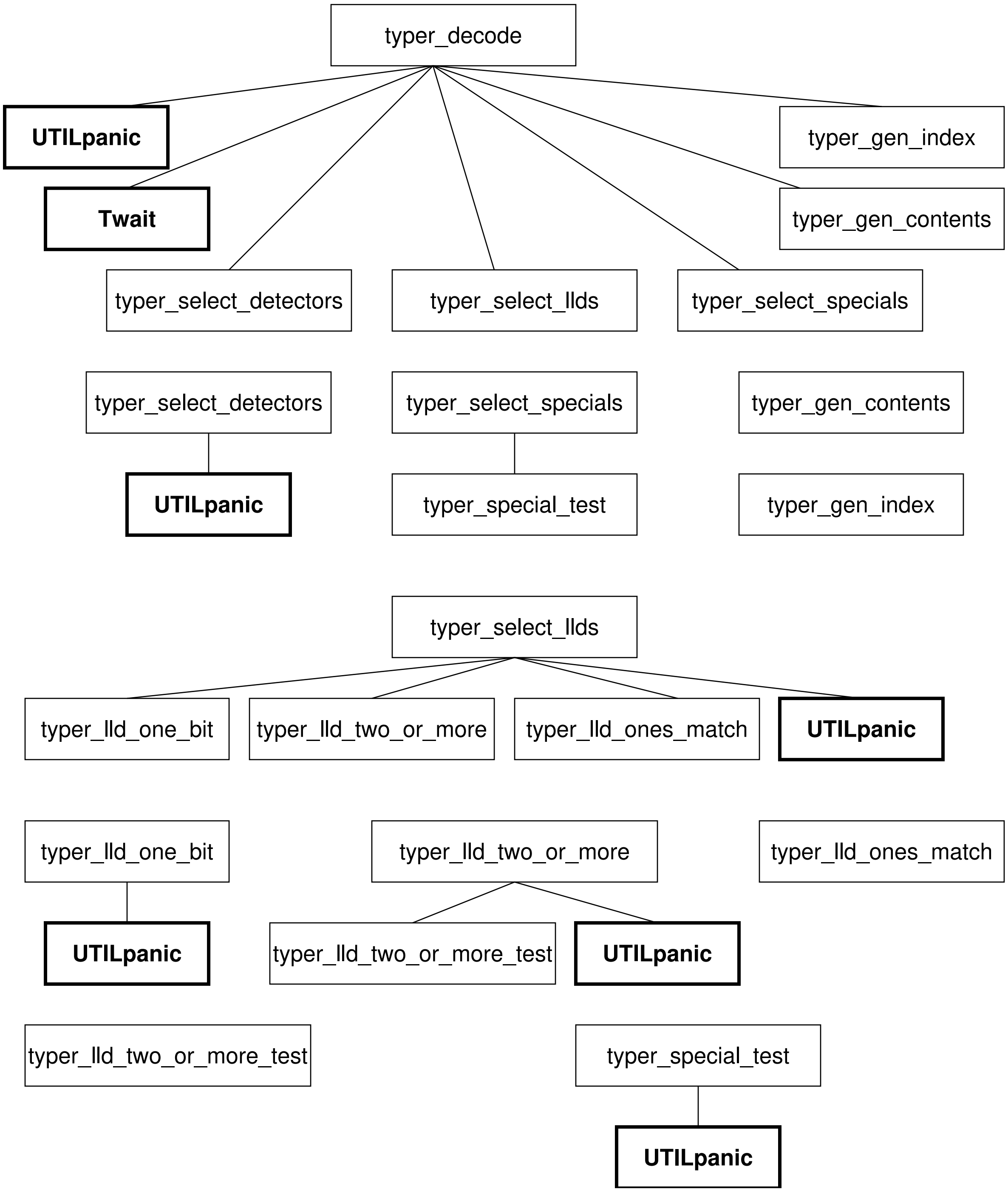

Figure 9 Typer Builder Module Internal Function Structures

typer_decode	Decode Encoded Typer Table into a specified buffer
typer_select_detectors	Generate array of Detector Ids selected by Match Pattern
typer_select_llds	Generate array of LLD codes selected by Match Pattern
typer_select_specials	Generate array of Veto/VLE/Propane/Calibration codes selected by Match Pattern.
typer_gen_index	Generate Typer Table index based on Detector Id, LLD bits and Veto/VLE/Propane/Calibration bits.
typer_gen_contents	Generate Typer Table contents based on Detector Id, LLD bits, Veto bits, VLE/Propane/Calibration bits and a Classification Directive's pass-through bit positions and Contents Base.
typer_lld_one_bit	Generate array of LLD codes which have one 1 bit which matches the specified bitmap.
typer_lld_two_or_more	Generate array of LLD codes which have two or more 1 bits which match the specified bitmap.
typer_lld_ones_match	Generate array of LLD codes which have 1s wherever there's a 1 in the specified bitmap (zeros in the bitmap are wild).
typer_lld_two_or_more_test	Determine whether or not the specified LLD Identifier has two or more bits set.
typer_special_test	Determine whether or not the specified Veto/VLE/Propane/Calibration bit pattern is selected by the specified Special Code and bitmap.

7 Requirements Traceability

7.1 Science Requirements

The Typer Builder Module addresses all PCA Science Requirements associated with the ability of the ground to configure the PCA Event selection and classification based on the following event inputs: detector id, LLD measurement chain, propane flag, very-large-event (VLE) flag, veto-event pulse height, and calibration flag. The requirements which have been addressed by this document are listed in the **Requirements Traceablity Matrix**.

This module does not address the PCA Science Requirements associated with Xenon pulse height recoding. These are handled in the **Recoder Builder Module Specification**.

8 Testing Considerations

8.1 Unit Testing

To perform unit tests on this module, the following equipment is required:

a. An Event Analyzer equipped with the Intermetrics ROM Monitor Debug Board

b. A workstation running the Intermetrics debugger

c. An RS-232 serial cable between the ROM Monitor's serial port and the workstation

To unit test this module, the tester must link and locate this module's functions to some EA RAM memory location. Using a resident monitor/debugger, the tester should first verify the operation of all functions internal to this module, and then verify the interactions between all functions internal to the module. These tests should verify all branches, calls, and legal and illegal parameters. The tester should then verify all calls made to functions outside of this module, and ensure that the passed arguments are always within the limits specified by the corresponding module specifications. The tester should also verify that the caller always handles values returned by these functions in an appropriate manner.

In addition to these tests, the tester should perform the following:

a. Generate a set of Match Patterns which contain all possible Detector Match values (6), using constants for the LLD Match and Special Code Match components of the Match Pattern and ensure that typer_decode() properly fills the appropriate regions of the Typer Table, or issues a panic message as appropriate.

b. Generate a set of Match Patterns which contain all possible LLD Match Codes and Bitmaps (256), using constants for the Detector Match and Special Code Match and ensure that typer_decode() properly fills the appropriate regions of the Typer Table, or issues a panic message as appropriate.

c. Generate a set of Match Patterns which contain all possible Special Code Match Operators and Bitmaps (256), using constants for the Detector Match and LLD Match and ensure that typer_decode() properly fills the appropriate regions of the Typer Table, or issues a panic message as appropriate.

d. Generate a variety of Match Patterns using combinations of Detector Match, LLD Match, and Special Code Matches and ensure that typer_decode() properly fills the appropriate regions of the Typer Table, or issues a panic message as appropriate. Some of these Match Patterns should generate maximum, minimum and intermediate values for the component Identifiers to ensure that the Typer Table index generation does not overlap the Identifiers when generating the index.

8.2 Integration Testing

For integration tests, the following equipment is required:

a. An Event Analyzer

b. A populated System Manager board

c. A logic analyzer

d. Hardware which can:

  1. Drive the MinorTick and MajorTick clock signals (S/C Simulator)

  2. Receive telemetry packets from the EA (S/C Simulator).

  3. Block telemetry packets from the EA (i.e. stop receiving packets).

  4. Send simulated PCA events to the EA (PCA-Simulator).

The integration tests should verify the operation of the Typer Builder Module in conjunction with the Science Thread Module and with the event processing hardware. These tests should include:

a. Verify that the Typer Builder Module properly builds Typer Tables under the control of the Science Thread.

b. Verify that the Typer Builder Module properly detects single-bit corruptions of the built Typer Table.

c. Verify that the Typer Builder Module properly detects outright corruption of the built Typer Table.

d. Verify that the Science Buffer is not in use while the Typer Builder Module checks the Typer Table, and that the DSP is held in a reset state.

e. Verify the operation of the Front End Typer Table using fabricated PCA event words and all Science Modes.

f. Ensure that the Typer Table properly discards undesired events.

g. Ensure that the Typer Table properly disables the Time-tag portion of the processed event words.

h. Ensure that the Typer Table properly selects the desired Virtual Recoder Table.

i. Test the Typer Table using the PCA Standard Data Modes, as defined in the **EDS-PCA Science Requirements Specification**.

j. Exercise the Typer Table using all PCA Science Modes and using a variety of configurations. Ensure that the Typer Table can be configured successfully, such that all event classification requirements for all modes, defined in the **PCA-EDS Science Requirements Specification**, are met.

k. Generate a Classification Directive which selects all entries in the Typer Table and ensure that it takes no more than 1 second to process the directive.

Appendix A

A.1 Items Under Review

This section lists items and issues which need to be resolved:

Document Reference	Section	Resolution Expected by
NONE		