SIEMENS

SIMATIC 575

Interboard Communications Specification

User Manual

Order Number: PPX:575–8103–2
Second Edition

⚠ DANGER

DANGER indicates an imminently hazardous situation that, if not avoided, will result in death or serious injury.

DANGER is limited to the most extreme situations.

⚠ WARNING

WARNING indicates a potentially hazardous situation that, if not avoided, could result in death or serious injury, and/or property damage.

⚠ CAUTION

CAUTION indicates a potentially hazardous situation that, if not avoided, could result in minor or moderate injury, and/or damage to property.

CAUTION is also used for property-damage-only accidents.

MANUAL PUBLICATION HISTORY

SIMATIC 575 Interboard Communications Specification
Order Manual Number: PPX:575–8103–2

Refer to this history in all correspondence and/or discussion about this manual.

Event	Date	Description
Original Issue	03/93	Original Issue (2801373–0001)
Second Edition	06/95	Second Edition (2801373–0002)

LIST OF EFFECTIVE PAGES

Pages	Description	Pages	Description
Cover/Copyright	Second		
History/Effective Pages	Second		
1 — 86	Second		
Registration	Second		

Customer Response

We would like to know what you think about our user manuals so that we can serve you better. How would you rate the quality of our manuals?

	Excellent	Good	Fair	Poor
Accuracy	_______	_______	_______	_______
Organization	_______	_______	_______	_______
Clarity	_______	_______	_______	_______
Completeness	_______	_______	_______	_______
Graphics	_______	_______	_______	_______
Examples	_______	_______	_______	_______
Overall design	_______	_______	_______	_______
Size	_______	_______	_______	_______
Index	_______	_______	_______	_______

Would you be interested in giving us more detailed comments about our manuals?

☐ **Yes!** Please send me a questionnaire.

☐ **No.** Thanks anyway.

Your Name: ______________________________

Title: ______________________________

Telephone Number: (_____)______________________________

Company Name: ______________________________

Company Address: ______________________________

____

____

Manual Name: SIMATIC 575 Interboard Communications Specification **Edition:** Second
Manual Assembly Number: 2589734–0004 **Date:** 6/95
Order Number: PPX:575–8103–2

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO.3 JOHNSON CITY, TN

POSTAGE WILL BE PAID BY ADDRESSEE

ATTN: Technical Communications M/S 519
SIEMENS INDUSTRIAL AUTOMATION INC.
3000 BILL GARLAND RD
P O BOX 1255
JOHNSON CITY TN 37605–1255

FOLD

PREFACE

This document describes the methods and algorithms used for boards in the SIMATIC® 575 system to communicate with each other. The organization of this document is as follows:

Section 1, Introduction	Describes the basic concepts of SIMATIC 575 interboard communications.
Section 2, Power-Up/Reset Processing	Describes the power-up/reset sequence including how boards are logged into the system and power-fail recovery.
Section 3, RLL locks	Describes RLL locks.
Section 4, Interboard Communications	Describes how boards pass messages to one another.
Section 5, Mode Change	Describes how to perform mode changes.
Section 6, Memory Management	Describes how boards can obtain memory from the System Heap.
Section 7, Interboard Messages	Describes each of the IBC messages.
Section 8, Interboard Communications Utilities	Describes a set of routines that implement interboard communications in the SIMATIC 575 system.
Section 9, Data Dictionary	Contains definitions of the data types, record structures, and variables used in the SIMATIC 575 system.
Section 10, GSDA Format Changes	Contains the changes made to the GSDA data structure and in what release of software those changes were made.
Section 11, IBC Error Codes	Contains the error codes and error descriptions for the messaging system.

1. INTRODUCTION

This section describes the basic concepts for interboard communication (IBC) within a SIMATIC 575 system. The SIMATIC 575 system is an open architecture Programmable Logic Controller(PLC) which is built on the concept of multiple CPUs operating together in a VMEbus backplane. Each of the CPUs in the SIMATIC 575 system have SIMATIC® 545 functional capabilities with extensions for interlocking applications (running on each CPU) and interboard communication. Provisions are made for third-party boards to participate in system operation and communicate across the VMEbus backplane as necessary to satisfy the requirements for the particular system application.

1.1 Scope

The scope of this specification applies to the messaging system within the SIMATIC 575 chassis only.

1.2 System Data Structures

When the SIMATIC 575 powers up and after all boards have completed power up diagnostics and de-asserted SYSFAIL*, the primary CPU initializes the system data structures which are used for system configuration (Board Log-in) and for interboard communication via the VMEbus backplane. The primary CPU then sequences through a procedure (described in Section 2, Power-Up/Reset Processing) which allows all boards to log into the various tables contained in the system data structures.

The relationship between the data structures is shown in Figure 1-1 System Data Structures. Each structure's details are explained in Section 9 Data Dictionary, which contains definitions of the data types, record structures, and variables used in the SIMATIC 575 system. For any changes to the GSDA format see Section 10 GSDA Format Changes.

The first structure of interest is the Global System Data Area (GSDA). This data structure resides at VME A24 address 0 (standard non-privileged data access) if the 575 CPU is in AUTO-CONFIGURED mode. If the 575 CPU is in USER-CONFIGURED mode the GSDA data structure resides at the VME A24 BASE-ADDRESS determined by the selection of SW3 and SW4 on the Primary 575's 505-remote I/O annex card (see the 575 System Manual PPX:575-8101-5). The GSDA contains variables that indicate the next available entry into other data structures and pointers to these structures. These entry indexes and pointers are used during board login at power-up/reset time. The format of GSDA is shown in Figure 1-2 Global System Data Area (GSDA). See Record GSDA in Section 9, Data Dictionary, for a description of the elements. The four most important pointers contained within GSDA are listed here.

BRDTBL	Points to the Board Table which contains an entry for each master board that logs into the SIMATIC 575 system. The format of BRDTBL entry is shown in Figure 1-4 Board Table Entry Format (BRDTBL). See Record BRDTBLEN in Section 9, Data Dictionary, for a description of a table entry. The primary areas of interest in this entry are the protocol type indicator, the pointer to the boards interrupt location if supported, interrupt location access method along with associated masks, and the pointer to the board's interboard communication area (IBCAREA). The board's IBCAREA contains head and tail pointers to a linked list of messages intended for that board. These are the principal elements used for interboard communication and are discussed further in Section 4. In addition, an 128 byte buffer(BTEPWRKA) is provided as a private work area.
SDT	Points to the System Descriptor Table which contains a list of the boards and the VME Medium Address (A24) space assigned to the board. The format of an SDT entry is shown in Figure 1-6 System Descriptor Table Entry Format (SDT). See Record SDTEN in Section 9, Data Dictionary, for a description of a table entry.
APPTBL	Points to the Application Table which contains a list of the Applications and the boards on which they execute. Application dependencies on other Applications and a pointer to the Application's G Memory are also contained in this table. The format of an APPTBL entry is shown in Figure 1-5 Application Table Entry Format (APPTBL). See Record APPTBLEN in Section 9, Data Dictionary, for a description of a table entry. Applications and G Memory are discussed further in Paragraph 1.3, Applications.
SBRDTBL	Points to the Slave Board Table which contains an entry for each slave board that logs into the SIMATIC 575 system. The format of SBRDTBL entry is shown in Figure 1-7 Slave Board Table Entry Format (SBRDTBL). See Record SBRDTBLE in Section 9, Data Dictionary, for a description of a table entry. The entry has a pointer to the Slave Board Specific Table which may or may not exist depending upon the Slave board function.

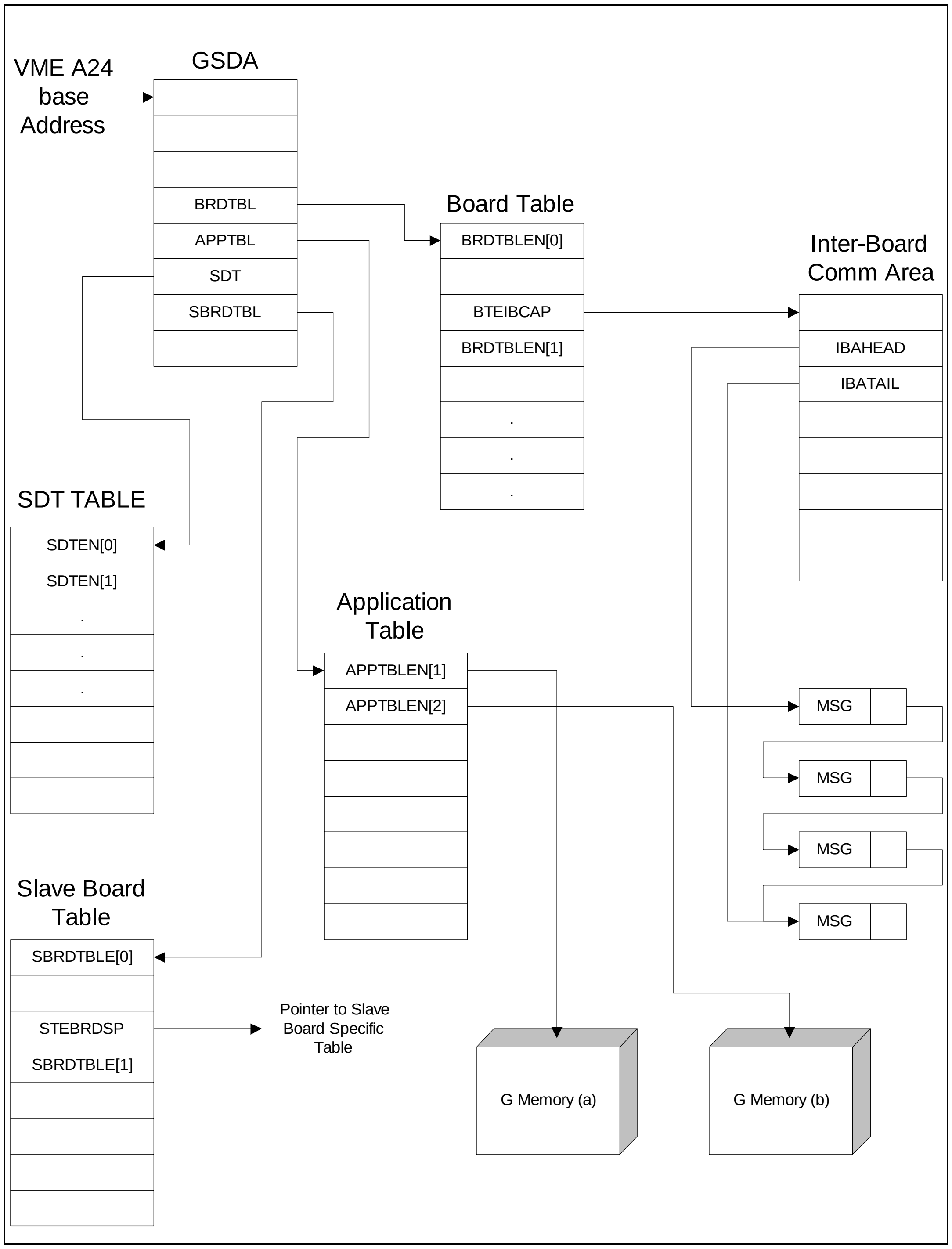

Figure 1-1 System Data Structures

Offset	31 – 24	23 – 16	15 – 08	07 – 00
0000	GSDAFMT[0]		GSDAFMT[1]	
0004	BATGOOD	filler	PURDONE	PURPHASE
0008	PURCMD			
000C	PURRESP			
0010	SDTNMLAR	SDTNSLAR	SDTNSDT	
0014	SDT			
0018	filler			BRDTBLMU
001C	BRDTBL			
0020	APPTBL			
0024	filler			SYSTODMU
0028	SYSTOD			
002C	(see Figure 1-3)			
0030	filler	SBRDTMUX	SBRDNSBR	
0034	SBRDTBL			
0038	filler	SYSIOCMU	SYSIOCNE	
003C	SYSIOCTB			
0040	IAPPLMAP			
0044	SYSCONFG	filler		
0048	PFSAVAR[0]			
.	.			
.	.			
.	.			
0084	PFSAVAR[15]			
0088	filler (120 bytes)			

Figure 1-2 Global System Data Area (GSDA)

Offset	31 – 28	27 – 24	23 – 20	19 – 16	15 – 12	11 – 08	07 – 04	03 – 00
0000	Y	Y	M	M	D	D	h	h
0004	m	m	s	s	f	f	0	d

Figure 1-3 System Time of Day Format (SYSTOD)

Offset	31 – 24	23 – 16	15 – 08	07 – 00
0000	BTELAR	BTEBID	BTMEMGOOD	BTEPFRDN
0004	BTEIBCPR		BTEPUTO	
0008	BTESDT		BTEDIAG	
000C	BTEIBCAP			
0010	BTEINTRP			
0014	BTEINTAC		BTEINTAM	BTEINTOM
0018	BTEFLAGS			
001C	BTESDESC (20 Bytes)			
.	.			
.	.			
002C	.			
0030	BTEHDESC (20 Bytes)			
.	.			
.	.			
0040	.			
0044	BTEPWRKA (128 Bytes)			
.	.			
.	.			
00C0	.			

Figure 1-4 Board Table Entry Format (BRDTBL)

Offset	31 – 24	23 – 16	15 – 08	07 – 00
0000	ATEMUTEX	ATEAPPID	ATELAR	filler
0004	ATETYPE		ATEOPMOD	
0008	ATEABRDS			
000C	ATEGMEMS			
0010	ATEGMEME			
0014	ATEREQ			
0018	ATEOPT			
001C	ATEMDLOK			
0020	ATEDESC (40 Bytes)			
.	.			
.	.			
0044	.			

Figure 1-5 Application Table Entry Format (APPTBL)

Offset	31 – 24	23 – 16	15 – 08	07 – 00
0000	filler		SELAR	SEBRDID
0004	ATEOPT			
0008	ATEMDLOK			

Figure 1-6 System Descriptor Table Entry Format (SDT)

Offset	31 24	23 16	15 08	07 00
0000	STELAR	STEBID	STESDT	
0004	STEBRDSP			
0008	STEDESC (40 Bytes)			
.	.			
.	.			
002C	.			

Figure 1-7 Slave Board Table Entry Format (SBRDTBL)

For a board to log into these data structures, it must follow one of the procedures detailed in either:

- Section 2.3.4 Power-up/reset -- Third-Party Masters using IRQ6.
- Section 2.3.6 Power-up/reset -- Boards with preassigned LARs.
- Section 2.3.7 Boards which cannot log in any other way.

A board must be able to access and interpret these data structures in order to participate in the interboard communication facility of the SIMATIC 575 system. The board does not have to log into the structures to send messages to and receive responses from another board or application within the system.

However, if a board is to be the recipient of a message, it must log into the board table and indicate support for one of the protocols described in Section 4, Interboard Communications. (See type IBCPROT in Section 9, Data Dictionary, for the protocol definitions.) A board that logs into the board table must also make an entry into the SDT and indicate where its shared memory resides.

1.3 Applications

An application is a set of related programs residing in a board. A board may contain multiple applications although the SIMATIC 575 CPU contains only one application per board. A board may also have no applications.

1.3.1 Application Assignment

A logical identifier (application ID - which is a letter from A through Z) is assigned to each application and is used in the RLL instructions referencing G-Memory. The application is tied to a particular board by variable ATELAR in the Application Table Entry.

1.3.2 Application Dependencies

Task Code 92 can be used to specify three types of application dependencies:

Required	The set of Applications which must be present in order for this application to transition to RUN.
Optional	The set of Applications which this application may reference, even though they may not be present in a given configuration. Reference to G Memory on a non-installed SIMATIC 575 CPU is allowed in RLL and is compiled to set bit 6 of STW1 and load reason code 0001 (Uninstalled application referenced) into STW200. This optional feature allows the same program to be used with different system configurations with skips around the unused RLL.
Mode-Locked	The set of Applications which must be in the same mode as this application. In order for any of the mode-locked applications to change modes, all applications must be able to make the change.

1.3.3 G Memory

Each application may present 0 to 32,768 words (SIMATIC 575 always presents 32K words) of contiguous VME A24-space memory which may be used as non-interlocked shared memory (G Memory). Access to G Memory is provided for most word-oriented RLL instructions. Other boards may access G Memory for any installed application by using the G Memory start address (variable ATEGMEMS in the Application Table Entry) as G1 and indexing to the desired offset.

1.4 System Mode Change

Each SIMATIC 575's operational mode (RUN/PROGRAM/SINGLE SCAN/RESTARTS) can be independently locked or unlocked to follow/not follow the mode of any other SIMATIC 575 CPU. The locked mode of a group of applications can be changed from any SIMATIC 575 whose application is locked to follow the group mode. A coordinated mode change is required and all locked applications must be able to enter the requested mode before the mode change can occur. If an application is not mode locked, it may change modes independently of other applications.

2. POWER-UP/RESET PROCESSING

This section describes the procedures used to initialize the system following a power-up/reset.

2.1 Introduction.

During power-up/reset the system is initialized. System initialization consists of the following:

1. Logging boards into the system.
2. Performing power-fail recovery.
3. Initiating system execution.

2.2 Power-Fail Recovery.

One of the requirements of the 575 system is that it survive power outages without losing any programs or data. To do this, the SIMATIC 575 CPU boards have battery-backup on all RAM areas. The system also provides an "early-power-fail" interrupt (EPF) which occurs at least 4 milliseconds before power fails completely.

Since the time between the EPF interrupt and power loss is short(4 milliseconds), the boards may not have time to complete all critical operations which may have been in progress at the time of the EPF interrupt. For this reason, the system implements something called power-fail recovery. For the 575, this means that only the processor context is saved at the time of the EPF. The critical operations are finished when power is restored, i.e., the next time the system powers up.

During power-up/reset processing, the Primary 575 indicates that power-fail recovery is to be done. Each board restores the processor context that was saved by the EPF handler, finishes up any critical operations that may have been in progress, and tells the Primary 575 that they are done. The Primary 575 waits for all boards to finish their power-fail recovery processing and then initiates system execution.

In order for power-fail recovery to work, the following restrictions apply:

1. All G-memory areas must reside in battery backed-up RAM.
2. All IBC message buffers must reside in battery backed-up RAM.
3. The power-up/reset handler and any power-up diagnostics must not modify any memory used by the operational software until after power-fail recovery completes.

The SIMATIC 575 provides a mechanism which boards which do not contain battery backed-up RAM can use to allocate memory from battery backed-up RAM.

It should be noted that power-fail recovery cannot always be performed. The conditions under which power-fail recovery is not performed are:

1. A board which was in the system when power failed is removed.
2. An application which was in the system when power failed is deleted.
3. A board which was in the system when power failed detects that its memory is scrambled. This would occur if a board is removed from the chassis and then replaced.
4. The position of two boards are swapped with the battery on.
5. The battery is off or has been turned off while power was off.
6. The system was in a failed state when power failed.

For conditions 1 through 4 the system will enter fault mode. For condition 5 the system will perform a battery bad power-up, clearing all memory, and for condition 6 the system will power up in the fault mode.

2.3 Power-Up Reset Processing.

2.3.1 Phases of power-up/reset.

Power-up/reset is broken into phases. The phase is indicated by variable PURPHASE which resides in the Global System Data Area (GSDA). The different phases of power-up/reset are represented in Figure 2-1 and Figure 2-2. The phases are as follows:

PURPHASE	00_{16}	This is the initial phase in which the boards perform their power-up diagnostics. This phase lasts until SYSFAIL* has been deasserted by all boards for at least 50 milliseconds.
PURPHASE	01_{16}	Standard board login phase. During this phase, the system logs in boards which use the VME IRQ6 IACK daisy-chain method of logging in. This is the method used by SIMATIC 575 CPUs and SIMATIC VME I/O modules. Masters using this method assert VME IRQ6 and present vector $40 or $FE in their STATUS/ID byte. When the master's IRQ6 interrupt request is acknowledged the master must log itself into the system tables and then set PURDONE, indicating to the primary CPU that it can proceed with log-in of the next board, if any. SIMATIC compatible VME I/O modules assert VME IRQ6 and present vector $41 in their STATUS/ID byte. The primary CPU logs these boards into the system tables when it recognizes their interrupt request. This process continues until all boards using the IRQ6 method have logged in. Because of the way the IRQ6 IACK works, boards using this method are logged into the system in relative slot order.
PURPHASE	02_{16}	Preassigned LAR login phase. During this phase, boards which have preassigned LARs are logged into the system. This phase is intended for boards which cannot use the standard board login sequence. The system allows a board to indicate a delay time to be used on a subsequent power up. This phase will last at least 200 milliseconds longer than the longest requested delay time. Boards using this method must acquire exclusive control of the Global System Data Area (GSDA). This is done using the Board Table Mutex (BRDTBLMU)
PURPHASE	03_{16}	Power-fail recovery. During this phase, all boards perform their power-fail recovery processing.
PURPHASE	04_{16}	Fault-Restart - The 575 system is performing a fault restart (i.e., entering fault mode). Do not access any variable in the GSDA area except PURHASE while in this phase.
PURPHASE	FF_{16}	Power-up/reset complete. The system begins operation. While the system is operating, boards which did not log in during power-up/reset may log themselves into the system. On subsequent power up, boards that previously logged in during this phase must log in during PURPHASE = 02_{16}. (See paragraph Section 2.3.7, Boards which cannot log in any other way.)

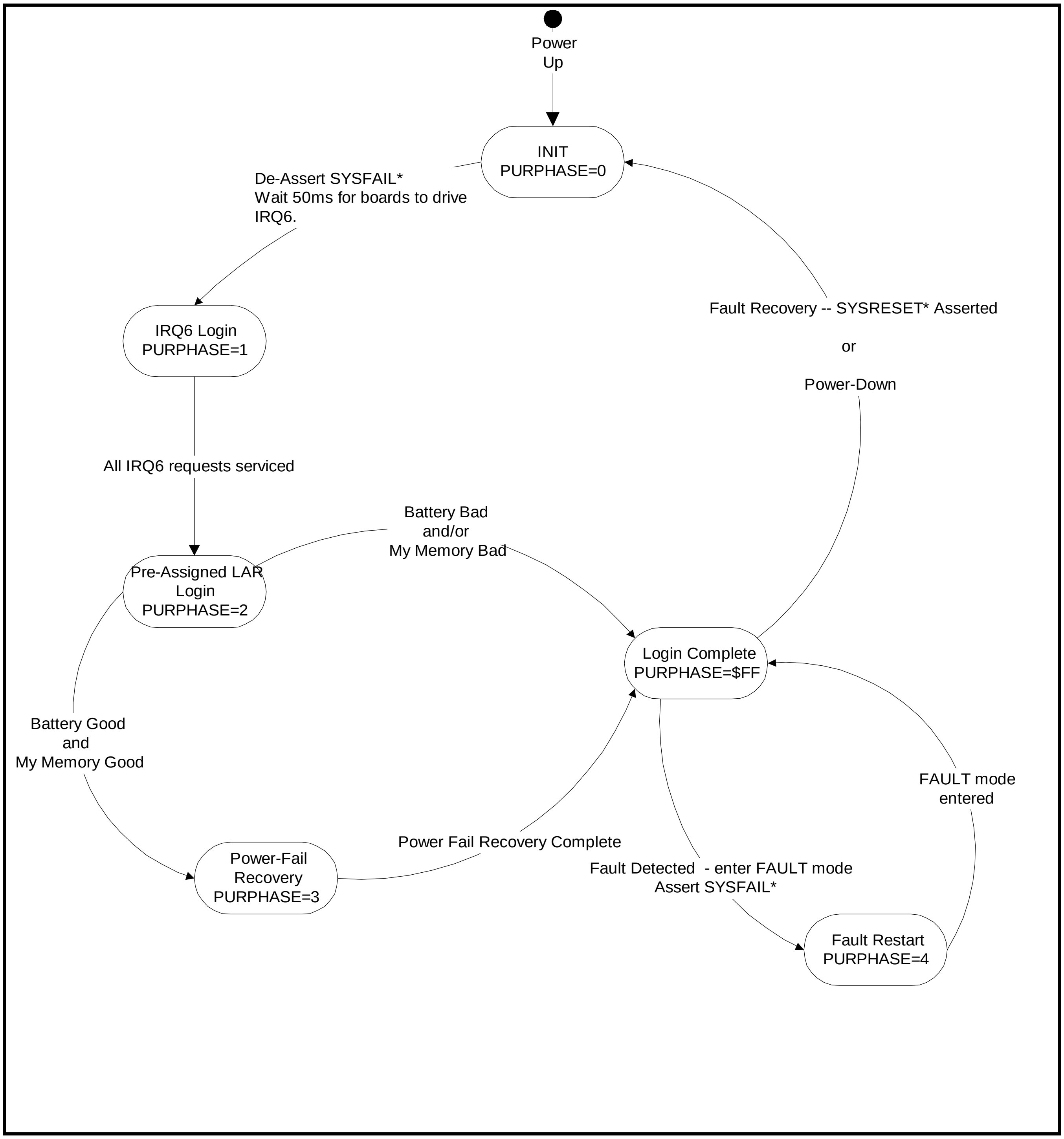

Figure 2-1 PURPHASE STATES - Assert SYSRESET*

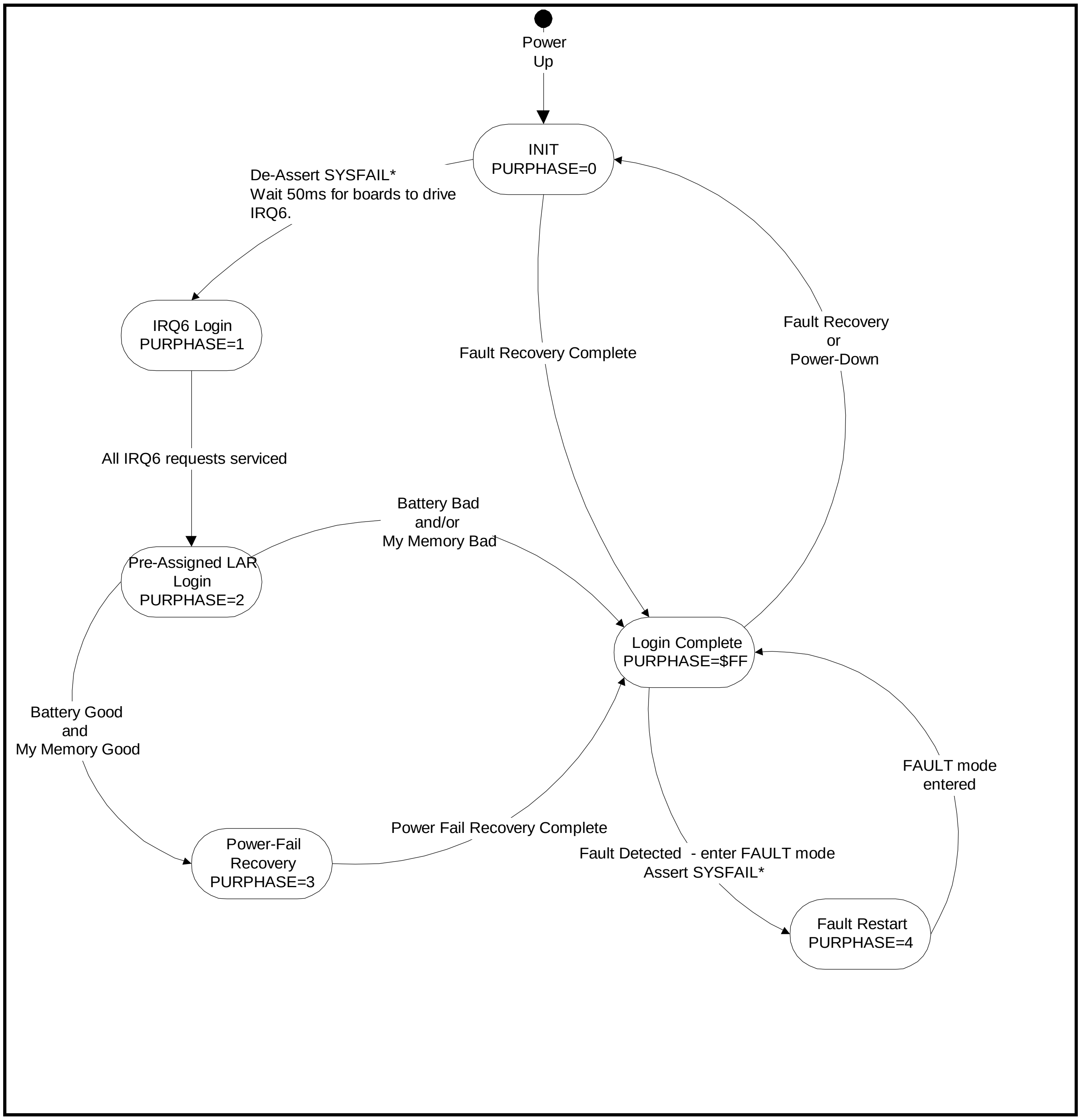

Figure 2-2 PURPHASE STATES - Do Not Assert SYSRESET*

2.3.2 Power-up/reset -- Primary 575 Processing

System initialization is controlled by the Primary 575 CPU. A 575 CPU is considered to be the Primary 575 CPU if it is set for **AUTO-CONFIGURED** mode and is installed in slot 1 (system controller slot) or if it is set for **USER-CONFIGURED** mode and the **PRIMARY-575** option has been selected for this 575 CPU. The Primary 575 CPU's processing is shown in the following pseudo-code.

```
/----------------------------------------------------------------------------\
  I$RESET -- PRIMARY CPU

  begin

     /*****************************************************/
     /*  NOTE THAT SYSFAIL* IS ASSERTED DUE TO POWER-UP.  */
     /*****************************************************/

     /*
     ** Map into VME space.
     */
     Major_Version := 0001₁₆;
     Minor_Version := 0001₁₆;
     if AUTO_CONFIGURED then
         map my GCSR at address 0000₁₆ of VME A16-space;
         map GSDA at address 000000₁₆ of VME A24-space;
         map my shared RAM at address 000000₁₆ of VME A24-space;
     else
         Major_Version := Major_Version + 8000₁₆;
         map my GCSR at BASE_ADDRESS * 4000₁₆ of VME A16-space;
         map GSDA at address BASE_ADDRESS * 400000₁₆ of VME A24-space;
         map my shared RAM at BASE_ADDRESS * 400000₁₆ of VME_A24 space;
     endif;
     GSDA->GSDAFMT[0] := Major_Version;
     GSDA->GSDAFMT[1] := Minor_Version;
     GSDA->SDT        := VME A24 address of SDT;
     GSDA->BRDTBL     := VME A24 address of BRDTBL;
     GSDA->APPTBL     := VME A24 address of APPTBL;
     GSDA->SBRDTBL    := VME A24 address of SBRDTBL;
     GSDA->SYSIOCTB   := VME A24 address of SYSIOCTB;
     GSDA->SYSCONFG   := System_Configuration;

/*
     ** Set power-up recovery to initial phase, perform initial diagnostics, and
     ** determine the state of the battery.
     */
     GSDA->PURPHASE := 00₁₆;
     if (this CPU in slot 1) then
        GSDA->BATGOOD := battery status (OK = FF₁₆, bad = 00₁₆);
     endif;
     /*
     ** If the battery is good, my memory is OK, and the system is configured
     **    the same as when power failed then make copies of the
     **    system tables.  The copies are used later to determine whether or
     **    not we can perform power-fail recovery.
     **
     ** If the battery is not good or my memory is has not bœn maintained then
     **    we just clear everything.
     */
     if ( GSDA->BATGOOD and
          (my memory is good) and
          (SDT[MY_SDT].SESMEMS equals start address of my shared memory) )
then
        if not OLD_TABLES_BUILT then
           copy SDT to OLD_SDT;
           copy BRDTBL to OLD_BRDTBL;
           copy APPTBL to OLD_APPTBL;
           OLD_TABLES_BUILT := true;
           MY_MEMORY_GOOD := true;
        endif;
     else
        OLD_TABLES_BUILT := false;
        clear all my memory;
        MY_MEMORY_GOOD := false;
     endif;

     /*
     ** Clear the system tables in preparation for board log-in.
     */
     GSDA->PURCMD   := 00₁₆;    /* Clear power-up-reset command.   */
     GSDA->BRDTBLMU := 00₁₆;   /* Unlock the board table mutex.   */
     GSDA->SDTNMLAR := 00₁₆;   /* Next VME MASTER LAR.            */
     GSDA->SDTNSLAR := 10₁₆;   /* Next VME SLAVE LAR.             */
     GSDA->SDTNSDT  := 00₁₆;   /* Index of next entry in the SDT. */
     clear SDT;
     clear BRDTBL;
     clear APPTBL;
     clear SBRDTBL;

     /*
     ** Set my LAR and log myself into the System Data Table (SDT).
     */
     MY_LAR                    := GSDA->SDTNMLAR;
     MY_SDT                    := GSDA->SDTNSDT;
     GSDA->SDTNMLAR            := GSDA->SDTNMLAR + 1;
     GSDA->SDTNSDT             := GSDA->SDTNSDT + 1;
     GSDA->SDT[MY_SDT].SELAR   := MY_LAR;
     GSDA->SDT[MY_SDT].SEBRDID := my board ID;
     GSDA->SDT[MY_SDT].SESMEMS := start address of my shared memory;
     GSDA->SDT[MY_SDT].SESMEME := end address of my shared memory;
```

```
/*
** Log myself into the Board Table (BRDTBL).
*/
GSDA->BRDTBL[MY_LAR].BTELAR   := MY_LAR;
GSDA->BRDTBL[MY_LAR].BTEBID   := my board ID;
GSDA->BRDTBL[MY_LAR].BTEMGOOD := MY_MEMORY_GOOD;
GSDA->BRDTBL[MY_LAR].BTEIBCPR := standard IBC message protocol;
GSDA->BRDTBL[MY_LAR].BTESDT   := MY_SDT;
GSDA->BRDTBL[MY_LAR].BTEDIAG  := diagnostic result status word
GSDA->BRDTBL[MY_LAR].BTEIBCAP := loc(My_IBC_Area);
GSDA->BRDTBL[MY_LAR].BTEINTRP := interrupter address;
GSDA->BRDTBL[MY_LAR].BTEINTAC := Interrupt access code;
GSDA->BRDTBL[MY_LAR].BTEINTAM := Interrupt_AND_Mask;
GSDA->BRDTBL[MY_LAR].BTEINTOM := Interrupt_OR_Mask;
GSDA->BRDTBL[MY_LAR].BTESDESC := Software Description;
GSDA->BRDTBL[MY_LAR].BTEHDESC := Hardware Description;
GSDA->BRDTBL[MY_LAR].BTEFLAGS := 0;
if ( I can not perform power fail recovery) then
    GSDA->BRDTBL[MY_LAR].BTEFLAGS.BTEF_PFRE := 1;
endif;
GSDA->BRDTBL[MY_LAR].BTEIBCAP->IBAMUTEX := 0; /* Unlock My_IBC_Area */

/*
** Log myself into the Application Table (APPTBL).
*/
if (any applications defined on my board) then
   initialize the APPTBL entries for my applications;
endif;

/*
** Phase 1 -- Log in boards using the IRQ6 IACK daisy-chain method.
*/
GSDA->PURPHASE := 01₁₆;
de-assert my SYFAIL* output;
wait for SYSFAIL* to be de-asserted from VMEbus ;
wait for the 50 ms;  /* Allow time for boards using PURPHASE 1 to assert IRQ6 */

while IRQ6 is asserted
   GSDA->PURDONE := 0;
   vector := acknowledge IRQ6 interrupt and read vector from requester;
   if (vector = 40₁₆ or vector = FE₁₆)
   then
      /*
      **  Allow master board to log itself in.
      */
      wait for PURDONE = FF₁₆;

   else    /* vector 41₁₆ - SIMATIC VME I/O board. */
      /*
      ** Log in a SIMATIC VME I/O slave board.
      */
      MODULE_LOGIN_ADDR := VME A16 address 0120₁₆;
      module_id         := *((byte *)MODULE_LOGIN_ADDR) & 0F₁₆;
      if module_id <> 3 then
         /* Invalid module ID for SIMATIC VME I/O module */
         enter FAULT mode
      else
         /*
         ** Configure SIMATIC VME I/O module.
         */
         MOD_LAR                        := SDTNSLAR & 0F₁₆;
         PAR                            := MOD_LAR | F0₁₆;
         PAR_LAR                        := (PAR << 8) | MOD_LAR;
         *( (word *) MODULE_LOGIN_ADDR ) := PAR_LAR;
         /*
         ** Log SIMATIC VME I/O module into System Descriptor Table.
         */
         GSDA->SDT[GSDA->SDTNSDT].SELAR   := GSDA->SDTNSLAR;
         GSDA->SDT[GSDA->SDTNSDT].SEBRDID := slave_ioboard_id;
         GSDA->SDT[GSDA->SDTNSDT].SESMEMS := PAR<<16 + $0000_{16}$;
         GSDA->SDT[GSDA->SDTNSDT].SESMEME := PAR<<16 + $FFFF_{16}$;

         /*
         ** Log SIMATIC VME I/O module into Slave Board Table.
         */
         GSDA->SBRDTBL[GSDA->SBRDNSBR].STELAR   := GSDA->SDTNSLAR;
         GSDA->SBRDTBL[GSDA->SBRDNSBR].STEBID   := module_id;
         GSDA->SBRDTBL[GSDA->SBRDNSBR].STESDT   := GSDA->SDTNSDT;
         GSDA->SBRDTBL[GSDA->SBRDNSBR].STEBRDSP := loc(GSDA->SYSIOTB[GSDA->SYSIOCNE]);
         GSDA->SBRDTBL[GSDA->SBRDNSBR].STEDESC  := Module Description;

         /*
         ** Log SIMATIC VME I/O module into System I/O Configuration Table.
         */
         GSDA->SYSIOCTB[GSDA->SYSIOCNE].SIOC_OWN := no_owner;
         GSDA->SYSIOCTB[GSDA->SYSIOCNE].SIOC_SDT := GSDA->SDTNSDT;

         /*
         ** Increment Table Indexes
         */
         GSDA->SDTNSDT  := GSDA->SDTNSDT + 1;
         GSDA->SDTNSLAR := GSDA->SDTNSLAR + 1;
         GSDA->SBRDNSBR := GSDA->SBRDNSBR + 1;
         GSDA->SYSIOCNE := GSDA->SYSIOCNE + 1;
      endif;
   endif;
endwhile;

/*
** Phase 2 -- Let boards with pre-assigned LARs log themselves in.
**
**    To determine the time to allow this phase to execute, we search
**    OLD_BRDTBL for defined entries and set PHASE_2_DELAY to the
**    maximum value specified in field BTEPUTO(seconds)+200 milliseconds.
*/
GSDA->PURPHASE := $02_{16}$;

PHASE_2_DELAY := 0 milliseconds;
for LAR := 0 to maxval(LAR)
   if OLD_BRDTBL[LAR].BTELAR = LAR then
      PHASE_2_DELAY := max(PHASE_2_DELAY, 1000 * OLD_BRDTBL[LAR].BTEPUTO)
   endif
endfor;
delay for PHASE_2_DELAY + 200 milliseconds;
```

```
/*
** Phase 3 -- Power-fail recovery.
**
**    The first thing we do is determine if we can even do power-fail
**    recovery.  This is done in several parts:
**
**      1. Compare the entries in the OLD_BRDTBL/OLD_SDT to BRDTBL/SDT.
**      2. Compare the entries in the OLD_APPTBL to APPTBL.
**
**    If anything doesn't match then we won't perform power-fail
**    recovery.
*/
get GSDA->BRDTBLMU exclusive;
if ( OLD_TABLES_BUILT )
  DO_POWER_FAIL_RECOVERY := true;
  LAR := 0;
  while ( LAR <= maxval(LAR) and DO_POWER_FAIL_RECOVERY)
     if (OLD_BRDTBL[LAR].BTELAR <> FF16) and   /* Slot was not Empty */
        (OLD_BRDTBL[LAR].BTEFLAGS[BTEF_PFR] = 0) then
        IF (GSDA->BRDTBL[LAR].BTELAR <> LAR)                              or
           (not GSDA->BRDTBL[LAR].BTEMGOOD)                               or
           (OLD_GSDA->BRDTBL[LAR].BTEBID <> GSDA->BRDTBL[LAR].BTEBID)     or
           (OLD_BRDTBL[LAR].BTEIBCAP <> GSDA->BRDTBL[LAR].BTEIBCAP)       or
           (OLD_SDT[OLD_BRDTBL[LAR].BTESDT].SESMEMS <>
                            GSDA->SDT[GSDA->BRDTBL[LAR].BTESDT].SESMEMS) or
           (OLD_SDT[OLD_BRDTBL[LAR].BTESDT].SESMEME <>
                            GSDA->SDT[GSDA->BRDTBL[LAR].BTESDT].SESMEME) then
           DO_POWER_FAIL_RECOVERY := false;
           Reinitialize VME heap;
           Release GSDA->BRDTBLMU;
           Fail system with an ABNORMAL_POWER_FAIL error;
        endif;
     endif;
     LAR := LAR + 1;
  endwhile;

  APP := 1;
  while ( APP <= maxval(APP_ID) and DO_POWER_FAIL_RECOVERY )
     if (OLD_APPTBL[APP].ATEAPPID <> 0) and  /* was present */
        (OLD_BRDTBL[OLD_APTBL[APP].LAR].BTEFLAGS[BTEF_PFR] = 0) then
        if (OLD_APPTBL[APP].ATEAPPID <> GSDA->APPTBL[APP].ATEAPPID) or
           (OLD_APPTBL[APP].ATETYPE  <> GSDA->APPTBL[APP].ATETYPE)  or
           (OLD_APPTBL[APP].ATEGMEMS <> GSDA->APPTBL[APP].ATEGMEMS) or
           (OLD_APPTBL[APP].ATEGMEME <> GSDA->APPTBL[APP].ATEGMEME) or
           (OLD_APPTBL[APP].ATEABRDS <> GSDA->APPTBL[APP].ATEABRDS) or
           (OLD_APPTBL[APP].ATEREQ   <> GSDA->APPTBL[APP].ATEREQ)   or
           (OLD_APPTBL[APP].ATEOPT   <> GSDA->APPTBL[APP].ATEOPT)   or
           (OLD_APPTBL[APP].ATEMDLOK <> GSDA->APPTBL[APP].ATEMDLOK) then
           DO_POWER_FAIL_RECOVERY := false;
           Reinitialize VME heap;
           Release BRDTBLMU;
           Fail system with an ABNORMAL_POWER_FAIL error;
        endif;
     endif;
     APP := APP + 1;
  endwhile;

else /* old tables were not built */
  DO_POWER_FAIL_RECOVERY := false;
endif;
Release GSDA->BRDTBLMU;
```

```
    /*
    ** If everything matched then initiate power-fail recovery and wait for
    **    everyone to finish.
    */
    if DO_POWER_FAIL_RECOVERY then
       GSDA->PURPHASE := 03$_{16}$;
       perform my board’s power-fail recovery procedure;
       for LAR := 1 to max(LAR)
          if GSDA->BRDTBL[LAR].BTELAR <> EMPTY and GSDA->BRDTBL[LAR].BTEMGOOD = GOOD then
             wait for flag GSDA->BRDTBL[LAR].BTEPFRDN to be set;
          endif;
       endfor;
    endif;
    OLD_TABLES_BUILT := false;  /* Copy new tables to old tables on next power-up */

    /*
    ** Power-up/reset complete.
    */
    GSDA->PURPHASE := FF$_{16}$;
    begin normal system operation;
 end.
\-----------------------------------------------------------------------------/
```

2.3.3 Power-up/reset -- Secondary 575 Processing

A secondary 575 CPU is any 575 CPU other than the Primary 575 CPU. The secondary 575 CPUs, log into the system during Phase 1 of power-up/reset. A 575 CPU determines that it must login as a Secondary 575 CPU by detecting that it is **AUTO-CONFIGURED** and is not installed in slot 1 or by detecting that it is **USER-CONFIGURED** and that the **SECONDARY-575** option is true. Secondary 575 processing in shown in the following pseudo-code.

```
 /-----------------------------------------------------------------------------\
  I$RESET -- Secondary 575 CPU

  begin
       /*****************************************************/
       /*  NOTE THAT SYSFAIL* IS ASSERTED DUE TO POWER-UP.  */
       /*****************************************************/

       /*
       ** Determine VMEbus base address for GSDA and GCSRs.
       */
       Major_Version = 0001₁₆;
       Minor_Version = 0001₁₆;
       if AUTO_CONFIGURED then
           GCSR address is 0000₁₆ in A16 address space;
           GSDA address is 000000₁₆ in A24 address space;
       else
           Major_Version := Major_Version + 8000₁₆;
           GCSR address is configured BASE_ADDRESS * 4000₁₆ in A16 address space;
           GSDA address is configured BASE_ADDRESS * 400000₁₆ in A24 address space;
       endif;

       /*
       ** If my role has changed since I was last powered-up, indicate that I don't
       **     have a good memory.
       */
       if ( I was a Primary 575 or the system controller last time I powered up ) then
           MY_MEMORY_GOOD := false;
       endif

       /*
       ** De-assert SYSFAIL* output and then wait for SYSFAIL* input to de-assert.
       */
       de-assert my SYSFAIL* output;
       wait for SYSFAIL* to be de-asserted from VMEbus;

       /*
       ** Request login rights.
       ** NOTE:   GSDA->PURPHASE equals 01₁₆ at this point.
       */
       assert IRQ6 interrupt request with STATUS/ID equal to 40₁₆ (or FE₁₆);
       wait for IRQ6 interrupt acknowledge;

       /*
       ** Fail the system if the Primary 575 CPU's configuration does not
       **    agree with my configuration.
       */
       if (GSDA->GSDAFMT[0] <> Major_Version) then
          Fail the system due to configuration error;
       endif;
```

```
/* Allocate my shared memory.  Start address must be greater than the base address
** of the GSDA and address range must not overlay memory allocated by any
** other board in the system.
*/
My_Mem_Start := allocate start address of my shared memory;
My_Mem_End   := My_Mem_Start + shared memory size;

/*
** Allocate my GCSR starting address.
*/
My_GCSR := allocate start address of my GCSR;  /* A16 base address + 16 * SDTNMLAR;

/*
** If the battery is good, my memory is OK, and the system is configured
**    the same as I am then prepare for a battery good restart.  Otherwise,
**    prepare for a bad battery restart.
*/
if (GSDA->BATGOOD and my memory is good) and
   (GSDA->SDTNMLAR == MY_LAR) and
   (GSDA->SDT[0].SESMEMS == GSDA) then
   MY_MEMORY_GOOD := true;
else
   MY_MEMORY_GOOD := false;
   Clear all my shared memory;
endif;     /*

/*
** Set my LAR to SDTNMLAR and log myself into the system data table.
*/
MY_LAR := GSDA->SDTNMLAR;
GSDA->SDT[GSDA->SDTNSDT].SELAR := MY_LAR;
GSDA->SDT[GSDA->SDTNSDT].SEBRDID := my board ID;
GSDA->SDT[GSDA->SDTNSDT].SESMEMS := My_Mem_Start;
GSDA->SDT[GSDA->SDTNSDT].SESMEME := My_Mem_End;

/*
** Log myself into the board table.
*/
GSDA->BRDTBL[MY_LAR].BTELAR := MY_LAR;
GSDA->BRDTBL[MY_LAR].BTEBID := my board ID;
GSDA->BRDTBL[MY_LAR].BTEMGOOD := MY_MEMORY_GOOD;
GSDA->BRDTBL[MY_LAR].BTEIBCPR := standard IBC message protocol;
GSDA->BRDTBL[MY_LAR].BTESDT := 0;
GSDA->BRDTBL[MY_LAR].BTEDIAG := diagnostic result status word
GSDA->BRDTBL[MY_LAR].BTEIBCAP := loc(my IBCAREA);
GSDA->BRDTBL[MY_LAR].BTEINTRP := interrupter address;
GSDA->BRDTBL[MY_LAR].BTEINTAC := Interrupt access code;
GSDA->BRDTBL[MY_LAR].BTEINTAM := Interrupt_AND_Mask;
GSDA->BRDTBL[MY_LAR].BTEINTOM := Interrupt_OR_Mask;
GSDA->BRDTBL[MY_LAR].BTESDESC := Software Description;
GSDA->BRDTBL[MY_LAR].BTEHDESC := Hardware Description;
GSDA->BRDTBL[MY_LAR].BTEFLAGS := 0;
if ( I can not perform power fail recovery) then
   GSDA->BRDTBL[MY_LAR].BTEFLAGS.BTEF_PFRE := 1;
endif;
GSDA->BRDTBL[MY_LAR].BTEIBCAP->IBAMUTEX := 0; /* Unlock My_IBC_Area */

/*
** Log myself into the Application table.
*/
if (any applications defined on my board) then
   initialize the APPTBL entries for my applications;
endif;
```

```
     /*
     ** Tell the Primary 575 that I'm done.
     */
     GSDA->SDTNMLAR := GSDA->SDTNMLAR + 1;     /* Next VME MASTER LAR.            */
     GSDA->SDTNSDT  := GSDA->SDTNSDT + 1;      /* Index of next entry in the SDT.  */
     GSDA->PURDONE  := FF16;   /* Tell the master that I have completed login.     */

     /*
     ** Complete my initialization.
     */
     if ( not MY_MEMORY_GOOD ) then
        clear all my memory;
     endif;
     /*
     ** Wait for Power-up/reset Phase 3 or FF16.
     */
     wait until GSDA->PURPHASE >= 0316;

     /*
     ** If this is Phase 3 do power fail recovery.
     */
     if GSDA->PURPHASE = 0316 then
        do power-fail recovery;
        GSDA->BRDTBL[MY_LAR].BTEPFRDN := FF16;
     endif;

     /*
     ** Wait for Power-up/reset to complete.
     */
     wait until GSDA->PURPHASE = FF16;
     begin normal system operation;
  end.
 \-----------------------------------------------------------------------------/
```

2.3.4 Power-up/reset -- Third-Party Masters using IRQ6.

These boards log themselves into the system during Phase 1 of power-up/reset. These boards must do the following:

1. De-assert SYSFAIL* output from my board.
2. Wait for SYSFAIL* to be de-asserted from the VMEbus.
3. Assert IRQ6 within 50ms with vector 40_{16} or FE_{16} in the STATUS/ID byte.
4. Wait for the IRQ6 interrupt acknowledge.
5. Check GSDAFMT[0] to ensure that it matches the GSDA data structure version that my board was programmed from (See Section 10 GSDA Format Changes). If it does not match fail the system.
6. Search through the SDT for a place to map my shared memory (if I can). Boards which can control where in A24-space their shared memory is to be mapped under software control should map their memory as low as possible. Users with boards which control where in A24-space their shared memory is to be mapped with dip-switches or jumpers should map their memory as high as possible. You are responsible for ensuring that no two boards' memory areas overlap. A24-space from $F00000_{16}$ to $FFFFFF_{16}$ is reserved for SIMATIC VME I/O modules. Masters and other slave boards should not map into this area.

   Boards which cannot directly control where their memory gets mapped should at least verify that they aren't overlaying someone else.
7. Log into entry SDT[SDTNSDT] of the SDT.
8. Increment SDTNSDT.
9. Log into entry BRDTBL[SDTNMLAR] of the BRDTBL.
10. Increment SDTNMLAR.
11. Log any applications into the APPTBL.
12. Set flag BTEMGOOD in my BRDTBL entry to indicate whether power-fail recovery is to be done.
13. If this master has slave boards to log in then for each slave board:
    - Enter the slave board into SBRDTBL[SBRDNSBR] and SDT[SDTNSDT].
    - Increment SBRDNSBR and SDTNSDT
    (Use steps 1 through 9 defined for the Primary 575 in Section 2.3.5 as a guide for the logging in slave boards.)
14. Initialize flag BTEFLAGS.BTEF_PFR in my BRDTBL entry to indicate whether or not this board on subsequent power up will participate in power-fail recovery.
15. Set flag PURDONE to FF_{16}.
16. Wait for PURPHASE >= 03_{16}.
17. If PURPHASE = 03_{16} then do the following:
    a. Do power-fail recovery processing.
    b. Set flag BTEPFRDN in my BRDTBL entry to FF_{16}.
18. Wait for PURPHASE = FF_{16}.
19. Start up.

2.3.5 Power-up/reset -- Slave Boards using IRQ6 IACK daisy-chain.

The Primary 575 will log these boards into the system during Phase 1 of power-up/reset. The Primary 575 expects slave boards that assert IRQ6 to be SIMATIC VME I/O or compatible modules. If not, the system will enter fatal error. Non SIMATIC compatible VME slave modules must wait to be logged in by an assigned master instead of driving IRQ6.

Note: Some third-party VMEbus slaves require configuration by a VMEbus master prior to their de-assertion of SYSFAIL*. The 575 CPU can not perform this configuration operation since the user program is not executed until after SYSFAIL* has been de-asserted by all boards. If you plan to use such a slave you must also include a non-575 master in your system configuration.

A24-space from $F00000_{16}$ to $FFFFFF_{16}$ is reserved for SIMATIC VME I/O modules. Masters and other slave boards should not map into this area.

To be logged in by the Primary 575, the slave board must do the following:

1. Wait for SYSFAIL* to be de-asserted.
2. Assert IRQ6 within 50ms with vector 41_{16} in the STATUS/ID byte.
3. Wait for the System Controller to acknowledge the interrupt.

The Primary 575 must do the following:

1. Set Slave Board LAR to SDTNSLAR.
2. Generate next PAR_LAR using LAR. (This step determines the VME A24 address space of the slave module.)
3. Write PAR_LAR to address 0120_{16} in A16 space. (This step tells the slave module what address space to respond to.)
4. Increment SDTNSLAR.
5. Log into entry SDT[SDTNSDT] of the SDT.
6. Increment SDTNSDT.
7. Log into entry SBRDTBL[SBRDTNSBR] of the SBRDTBL.
8. Increment SBRDTNSBR.
9. Log into entry SYSIOCTB[SYSIOCEN] of SYSIOCTB.
10. Increment SYSIOCEN.
11. Set flag PURDONE.

2.3.6 Power-up/reset -- Boards with preassigned LARs.

These boards log themselves into the system during Phase 2 of power-up/reset. These boards must do the following:

1. Wait for SYSFAIL* to be de-asserted.
2. Check GSDAFMT[0] to ensure that it matches the GSDA data structure version that the board logging in was programmed from (See Section 10 GSDA Format Changes). If it does not match - fail the system.
3. Wait for PURPHASE = 02_{16}.
4. Gain exclusive control of the Board Table Mutex (BRDTBLMU).
5. Verify that the BRDTBL entry that I think I have is not being used. If it is being used then about all you can do is find an unused entry in the BRDTBL and use it. (Be sure to clear BTEMGOOD so that the System Controller will not try to initiate power-fail recovery.)
6. Search through the SDT for a place to map my shared memory (if I can). Boards which can control where in A24-space their shared memory is to be mapped under software control should map their memory as low as possible. Users with boards which control where in A24-space their shared memory is to be mapped with dip-switches or jumpers should map their memory as high as possible. They are also responsible for ensuring that no two boards' memory areas overlap.
   Boards which can't directly control where their memory gets mapped should at least verify that they aren't overlaying anyone else.
7. Log into entry SDT[SDTNSDT] of the SDT.
8. Increment SDTNSDT.
9. Log into entry BRDTBL[MY_LAR] of the BRDTBL.
10. Log any applications into the APPTBL.
11. Set flag BTEMGOOD in my BRDTBL entry to indicate power-fail recovery is to be done otherwise clear it.
12. If this master has slave boards to log in then for each slave board:
    a. Enter the slave board into SBRDTBL[SBRDNSBR] and SDT[SDTNSDT]. (Use steps 1 through 9 defined for the Primary 575 in Section 2.3.5, Power-up/reset -- Slave Boards using IRQ6 IACK daisy-chain., as a guide for the logging in slave boards.)
13. Initialize variable BTEPUTO in my BRDTBL entry to indicate the number of seconds that the System Controller will wait for the board to log itself in during power-up/reset.
14. Initialize flag BTEFLAGS.BTEF_PFR in my BRDTBL entry to indicate whether or not this board on subsequent power up will participate in power-fail recovery.
15. Release BRDTBLMU.
16. Wait for PURPHASE = 03_{16} or PURPHASE = FF_{16}.
17. If PURPHASE = 03_{16} then do the following:
    a. Do power-fail recovery processing.
    b. Set flag BTEPFRDN in my BRDTBL entry.
18. Wait for PURPHASE = FF_{16}.
19. Start up.

2.3.7 Boards which cannot log in any other way.

The SIMATIC 575 also provides a mechanism for boards which cannot log into the system any other way to log themselves in. They basically wait until the rest of the system starts up and then log themselves in. Processing for these boards is as follows:

1. Wait for SYSFAIL* to be de-asserted.
2. Check GSDAFMT[0] to ensure that it matches the GSDA data structure version that the board logging in was programmed from (See Section 10 GSDA Format Changes). If it does not match - don't login.
3. Wait for PURPHASE = FF_{16}.
4. Gain exclusive control of the Board Table Mutex (BRDTBLMU).
5. Find an unused entry in the Board Table. (Boards using this method should begin their search at the end of the Board Table). Set MY_LAR to the index of this entry.
6. Search through the SDT for a place to map my shared memory (if I can). Boards which can control where in A24-space their shared memory is to be mapped under software control should map their memory as low as possible. Users with boards which control where in A24-space their shared memory is to be mapped with dip-switches or jumpers should map their memory as high as possible. They are also responsible for ensuring that no two boards' memory areas overlap.
   Boards which can't directly control where their memory gets mapped should at least verify that they aren't overlaying anyone else.
7. Log into entry SDT[SDTNSDT] of the SDT.
8. Increment SDTNSDT.
9. Log into entry BRDTBL[MY_LAR] of the BRDTBL.
10. If this master has slave boards to log in then for each slave board:
    a. Enter the slave board into SBRDTBL[SBRDNSBR] and SDT[SDTNSDT]. (Use steps 1 through 9 defined for the Primary 575 in Paragraph Power-up/reset -- Slave Boards using IRQ6 IACK daisy-chain. as a guide for the logging in slave boards.)
11. Initialize variable BTEPUTO in my BRDTBL entry to indicate the number of seconds that the System Controller will wait for the board to log itself in during power-up/reset.
12. Initialize flag BTEFLAGS.BTEF_PFR in my BRDTBL entry to indicate whether or not this board will participate in power-fail recovery on subsequent power up.
13. Release BRDTBLMU.
14. Start up.

Boards which log themselves into the system after power-up/reset and DO NOT wish to participate in Power-Fail-Recovery on the subsequent power-up must set the BRDTBLE[MY_LAR].BTEFLAG.BTEF_PFR bit to 1, i.e., DO NOT participate in power-fail-recovery. On a subsequent power-up this board must wait until PURPHASE is equal to FF_{16} to login.

Boards which log themselves into the system after power-up/reset and wish to participate in Power-Fail-Recovery on the subsequent power-up must clear the BRDTBLE[MY_LAR].BTEFLAG.BTEF_PFR bit and write the number of seconds for the Primary 575 to wait for it to login in entry BRDTBLE[MY_LAR].BTEPUTO. On a subsequent power up this board must now login during PURPHASE equal to 02_{16}.

3. RLL LOCKS

This section describes RLL locks.

3.1 Introduction.

An RLL lock is a longword that normally resides in G-memory and is used by applications to control access to shared resources. Refer to type RLL_Lock in Section 9, Data Dictionary, for the format of an RLL lock. RLL locks are manipulated with the LOCK and UNLOCK RLL box instructions.

An RLL lock may be in one of three states as follows:

FREE	In this state, anyone may obtain the lock.
HELD SHARED	In this state, applications requesting the lock EXCLUSIVE are blocked. Applications requesting the lock SHARED are allowed to proceed (up to a maximum of 255).
HELD EXCLUSIVE	In this state, all other requests for the lock are blocked.

An RLL lock is defined in such a way as to prevent applications requesting the lock SHARED from starving an application requesting the lock EXCLUSIVE.

3.2 Lock Manipulation Procedures.

This paragraph describes the procedures that applications should use to manipulate RLL locks. The procedures are given as pseudo-code for performing operations on the locks. There are three procedures described as follows:

S$LOCKS	Obtain a lock SHARED.
S$LOCKX	Obtain a lock EXCLUSIVE.
S$UNLOCK	Release a lock.

In the following pseudo-code descriptions, two of the Interboard Communication Utilities, MU$GET and MU$REL, described in Section 8, Interboard Communications Utilities, are used. These utilities assume the existence of the following functions:

get_time()	Function that returns the current system time in clock ticks.
test_and_set(mutex)	Function that will set the most-significant bit of its parameter using an indivisible "read-modify-write" bus cycle. No other bits in the parameter may be modified. Function test_and_set returns the value of the bit before the instruction executed (i.e., "true" if the bit was set and "false" if the bit was clear).

3.2.1 S$LOCKS - Obtain Shared Control Of An RLL Lock

Procedure S$LOCKS obtains SHARED control of an RLL lock. The parameters of this procedure are as follows.

L	input / output	The RLL lock to be obtained.
TIMEOUT	input only	Time-out in clock ticks. 0 indicates that S$LOCKS is to try to obtain the lock one time. Maxval(clock_ticks) indicates that S$LOCKS is to try to obtain the lock forever.
OBTAINED	output	Flag indicating whether or not the lock was successfully obtained.

Procedure S$LOCKS will attempt to obtain SHARED control of an RLL lock. S$LOCKS will attempt to obtain the lock for up to TIMEOUT clock ticks before reporting failure.

If the lock is obtained, S$LOCKS will check flag RLREQEX to determine if another application is trying to obtain the lock EXCLUSIVE. If the flag is set, S$LOCKS will release the lock and try again to obtain the lock until either it obtains the lock with flag RLREQEX clear or the time-out expires. The purpose of the RLREQEX flag is to prevent applications that are requesting the lock SHARED from starving an application requesting the lock EXCLUSIVE.

Function S$LOCKS - Obtain Shared Control Of An RLL Lock

```
/*----------------------------------------------------------------------*\
S$LOCKS(inout  L        : RLL_Lock,
        in     TIMEOUT  : clock_ticks,
        output OBTAINED : boolean);

var
   ELAPSED_TIME : clock_ticks;
   START_TIME   : clock_ticks;
   GOT_MUTEX    : boolean;

begin
   ELAPSED_TIME := 0;
   START_TIME := get_time();
   repeat
      MU$GET(L.RLMUTEX, TIMEOUT - ELAPSED_TIME, GOT_MUTEX)
      if GOT_MUTEX then
         if L.RLREQEX or (L.RLHELDCN = maxval(RLL_Lock.RLHELDCN)) then
            /*
            ** Someone is trying to obtain the lock exclusive or the
            ** lock is held by the maximum number of holders.  Release
            ** the mutex and try again.
            */
            MU$REL(L.RLMUTEX)
         else
            /*
            ** We can get the lock.  Increment the held count, release
            ** the lock, and return success.
            */
            L.RLHELDCN := L.RLHELDCN + 1;
            MU$REL(L.RLMUTEX);
            OBTAINED :=true;
            return
         endif
      endif;
      if TIMEOUT <> maxval(clock_ticks) then
         ELAPSED_TIME := get_time() - START_TIME
      endif
   until ELAPSED_TIME >= TIMEOUT;
   /*
   ** We did not obtain the lock, return failure.
   */
   OBTAINED := false;
   return
end.
\*----------------------------------------------------------------------*/
```

3.2.2 S$LOCKX - Obtain Exclusive Control Of An RLL Lock

Procedure S$LOCKX is used to obtain EXCLUSIVE control of an RLL Lock. The parameters of this procedure are as follows:

L	input / output	The RLL lock to be obtained.
TIMEOUT	input only	Time-out in clock ticks. Zero (0) indicates that S$LOCKX is to try to obtain the lock one time. Maxval(clock_ticks) indicates that S$LOCKX is to try to obtain the lock forever.
OBTAINED	output	Flag indicating whether or not the lock was successfully obtained.

Procedure S$LOCKX will attempt to obtain EXCLUSIVE control of an RLL lock. S$LOCKX will try to obtain the lock for up to TIMEOUT clock ticks before reporting failure.

While trying to obtain the lock, S$LOCKX will set the "requesting EXCLUSIVE" flag (RLREQEX) of the lock. This is done to prevent applications requesting SHARED control of the lock from obtaining it and thus starving applications requesting EXCLUSIVE control of the lock.

Function S$LOCKX - Obtain Exclusive Control Of An RLL Lock

```
/*----------------------------------------------------------------------*\
S$LOCKX(inout  L        : RLL_Lock,
        in     TIMEOUT  : clock_ticks,
        output OBTAINED : boolean);

import
   MY_APP_ID           : APP_ID;

var
   ELAPSED_TIME        : clock_ticks;
   REQUESTED_EXCLUSIVE : boolean;
   START_TIME          : clock_ticks;
   GOT_MUTEX           : boolean;

begin
   REQUESTED_EXCLUSIVE := false;
   ELAPSED_TIME := 0;
   START_TIME := get_time();
```

```
repeat
      MU$GET(L.RLMUTEX, TIMEOUT - ELAPSED_TIME, GOT_MUTEX)
      if GOT_MUTEX then
         if L.RLREQEX and (not REQUESTED_EXCLUSIVE) then
            /*
            ** Someone else has requested the lock exclusive, release
            ** the mutex and try again.
            */
            MU$REL(L.RLMUTEX);
         else
            if L.RLHELDCN <> 0 then
               /*
               ** One or more applications already hold the lock
               ** shared.  Indicate that we are requesting the lock
               ** exclusive and release the mutex.
               */
               L.RLREQEX := true;
               MU$REL(L.RLMUTEX);
               REQUESTED_EXCLUSIVE := true
            else
               /*
               ** We obtained the lock.  Return success.
               */
               L.RLHOLDER := MY_APP_ID;
               L.RLHELDCN := 1;
               L.RLREQEX := false;
               OBTAINED := true;
               return
            endif
         endif
      endif;
      if TIMEOUT <> maxval(clock_ticks) then
         ELAPSED_TIME := get_time() - START_TIME
      endif
   until ELAPSED_TIME >= TIMEOUT;
   /*
   ** We did not obtain the lock.  If we indicated that we requested
   ** it exclusive then we must clear the "requested exclusive" flag
   ** (RLREQEX).
   */
   if REQUESTED_EXCLUSIVE then
      L.RLREQEX := false
   endif;
   /*
   ** Return failure.
   */
   OBTAINED := false;
   return
end.
\*-----------------------------------------------------------------------*/
```

3.2.3 S$UNLOCK - Release An RLL Lock

Procedure S$UNLOCK is used to release an RLL Lock that was obtained using S$LOCKS or S$LOCKX. The parameters for S$UNLOCK are as follows.

L	input / output	The RLL lock to be released.

Procedure S$UNLOCK will release an RLL lock that was previously obtained with either S$LOCKS or S$LOCKX. It is an error for an application to release a lock that it does not hold.

```
/*-----------------------------------------------------------------------*\
S$UNLOCK(inout L : RLL_Lock);

import
   MY_APP_ID  : APP_ID;

var
   GOT_LOCK   : boolean;

begin
   if L.RLMUTEX and (L.RLHOLDER = MY_APP_ID) then
      /*
      ** We hold the lock exclusive.
      */
      L.RLHELDCN := 0;
      L.RLHOLDER := 0
   else
      /*
      ** We hold the lock shared.
      */
      MU$GET(L.RLMUTEX, maxval(clock_ticks), GOT_LOCK);
      L.RLHELDCN := L.RLHELDCN - 1
   endif;
   MU$REL(L.RLMUTEX)
end.
\*-----------------------------------------------------------------------*/
```

4. INTERBOARD COMMUNICATIONS

This section describes the data structures and algorithms used for interboard communications.

A board must be able to access and interpret the data structures discussed in Section 1, Introduction, in order to participate in the interboard communication facility of the SIMATIC 575 system.

Each entry in the Board Table contains variable BTEIBCAP, which is a pointer to that board's interboard communication area (IBCAREA), and variable BTEIBCPR, which is a protocol type indicator.

The board's IBCAREA contains head and tail pointers to a linked list of messages intended for that board. See Record IBCAREA in Section 9, Data Dictionary, for a description of each field. Figure 4-1 shows the format for an IBCAREA record.

For a board to send a message to another board and receive a response, it must be able to read the various tables.

For a board to receive messages from other boards in the chassis, it must log into the various tables and indicate the protocol supported in its BTEIBCPR field.

To support the standard interrupt communications protocol, a board must provide an interrupt location that is accessed as a byte using one of the methods indicated by variable BTEINTAC in the board table.

If the other protocols are sufficient, it is not necessary to provide this location.

Offset	31 24	23 16	15 08	07 00
0000	IBAMUTEX	filler		
0004	IBAHEAD			
0008	IBATAIL			

Figure 4-1 Interboard Communication Area Header Format (IBCAREA)

Variable BTEIBCPR specifies one of three types of protocol to send messages to a board. See type IBCPROT in Section 9, Data Dictionary, for the allowed values. The three types are:

- Standard interrupt communications protocol. Indicates that messages and responses are queued in the board's IBCAREA and an interrupt is generated.
- Standard polled communications protocol. Indicates that messages and responses are queued in the board's IBCAREA. The board must poll field IBAHEAD to determine if anything is queued.
- No messages, polled responses. Indicates that the board cannot accept messages. For any messages that it sends, it must poll the "is response" field (IBMISRES) of the message to determine when the response is ready.

To send messages to and receive responses from another board or application within the system the board must be able to perform the functions detailed in the following utility routines described in Section 8, Interboard Communications Utilities:

C$ENQMSG	Enque Message To IBC Area.
C$REPLY	Send Reply To An IBC Message.
C$SEND	Send Interboard Message.
C$SENDAA	Send Application-To-Application Message.
C$SENDAB	Send Application-To-Board Message.
C$SENDBA	Send Board-To-Application Message.
C$SENDBB	Send Board-To-Board Message.

In summary, a board must be able to:

- Format a message as shown in Figure 4-2 and defined by Record IBCMSG in Section 9, Data Dictionary. If the sending board is not logged into the system, then the Sending LAR is set to FF_{16} and the sending Application ID is set to 0.
- Locate the IBCAREA which is pointed to by variable BTEIBCAP in the board table entry for the destination board. Use the destination board's LAR to index into the board table. If the destination is an Application, the board LAR is in variable ATELAR of the application's Application Table entry.
- Obtain the IBCAREA Mutex and enqueue the message on the destination boards linked list by entering a pointer to the source message into both the IBCAREA's tail pointer and into the next message field of the message that was at the tail of the link. (See C$ENQMSG in Section 8, Interboard Communications Utilities)
- Determine the protocol type from variable BTEIBCPR in the board table entry for the destination board.
    - ◊ If the destination board supports standard interrupt communications protocol, locate the interrupt register for the destination board, perform the appropriate access to generate the interrupt, and then release the IBCAREA Mutex.
    - ◊ If the destination board supports standard polled communications protocol, release the IBCAREA Mutex.
- Wait for response by one of the following:
    - ◊ If the sending board supports standard interrupt communications protocol, wait for interrupt.
    - ◊ If the sending board supports standard polled communications protocol, poll field IBAHEAD for response on message queue.
    - ◊ If the sending board does not log in or supports "no messages, polled responses", poll for Variable IBMISRSP in the message structure to change from 0 to 1.

Table 4-1 summarizes the SIMATIC 575 interboard messages and Section 7, Interboard Messages, describes each of the messages. Section 11, IBC Error Codes, contains a list of error codes and their description.

Offset	31 24	23 16	15 08	07 00
0000	IBMNEXT			
0004	IBMRWRKA (16 Bytes)			
.	.			
.	.			
0010	.			
0014	IBMSWRKA (16 Bytes)			
.	.			
.	.			
0020	.			
0024	IBMSLAR	IBMSAPP	IBMRLAR	IBMRAPP
0028	IBMISRSP	filler	IBMTYPE	
002C	IBMRESP		IBMDSIZE	
0030	IBMDLNT		filler	
0034	IBMREQDP			
0038	IBMRSPDP			

Figure 4-2 Message Entry Format (IBCMSG)

Message Code	Message
0001	Configure port.
0002	Read port configuration.
0003	Connect to port.
0004	Disconnect from port.
0005	Read port connections.
001B	Allocate memory from system heap.
001C	Release memory to system heap.
001D	Report available memory.
001E	Execute application task.

Table 4-1 Interboard Message Type (IBMTYPE) Summary

5. MODE CHANGE

5.1 Initiating a Mode Change

To initiate a mode change, send the appropriate taskcode (see below) using message $001E_{16}$ (Execute Application Task) to the proper SIMATIC 575 CPU application. This application will coordinate the mode change between all participating applications. When the mode change is complete the acknowledgment is sent via message $001E_{16}$ to the initiator. A complete description of the following mode change task codes can be found in the SIMATIC 575 Task Code Specification (PPX:575-8104).

32 go to run,
33 go to program,
34 Execute Power Up Restart,
35 Execute Complete Restart,
36 Execute Partial Restart.

5.2 Detecting a Mode Change

The mode of any SIMATIC 575 application can be determined by polling the application's mode in the Application Table (see Record APPTBLEN in Section 9, Data Dictionary, for a description of the Application Table entry) or by periodically sending task code 30 to that application.

6. MEMORY MANAGEMENT

This section describes the management of System Heap by the Primary 575. It also recommends methods to obtain and release battery backed System Heap for each board that needs it.

6.1 Building System Heap

On a bad battery power up, the Primary 575 initializes a System Heap which consists of a memory pool residing on the Primary 575.

All System Heap allocations are taken from this pool. This pool is retained in its pre power-fail state for all subsequent battery good power cycles.

If the system fails on power-up due to an abnormal power failure (e.g., the system tables did not match, etc.) system heap is reclaimed by the Primary 575. Otherwise the system heap is left in its pre power-fail state.

6.2 Using System Heap

System Heap is intended to be used by boards that have no battery backed memory of their own for maintaining the following items necessary for power-fail recovery processing:

1. Inter-board message buffers. Since a given message may result in a critical operation being performed by the destination board, and since power could fail during message processing, message buffers must remain intact until power-fail recovery processing has completed.
2. Information necessary to re-log into the board into the same place of the following tables on subsequent power-up/resets.
    a. The Application Table.
    b. The Board Table.
    c. The System Descriptor Table.
3. Any other information necessary to be maintained through a power cycle.

6.3 Allocating System Heap

A board which needs battery backed memory may request allocation from the Primary 575 at anytime after power-fail recovery processing completes (PURPHASE set to FF_{16}). This request is accomplished by sending the inter-board message $001B_{16}$, Allocate Memory from System Heap, to the Primary 575.

This message must be in non-volatile memory, so for boards that have none, a block of 128 bytes is provided in each entry of the board table (BTEPWKRA). This memory is battery backed and can be used as a message buffer to request more memory from the heap if necessary. To use this buffer, a board must be logged into the board table.

Allocation of System Heap should be viewed as a high overhead function and only used the first time a board appears in the system. Dynamic allocation of memory from this pool should be minimized with the board requesting one block of memory for all it's needs and then performing dynamic buffer management within that block locally.

6.4 Finding Allocated System Heap

Boards may find their allocated battery backed heap following a battery good power up by using the report memory allocation command of the PURCMD facility while they have control of the Board Table during login. The appropriate information can then be copied from the allocated memory to the various tables before indicating "power-up done" during the login procedure.

If a board does not participate in log in and/or power-fail recovery processing but still needs battery backed memory from System Heap, it can still find it's previously allocated memory. This is accomplished after power fail recovery processing completes by sending the inter-board message $001D_{16}$, Report Available Memory, to the Primary 575.

6.5 Releasing System Heap

If a board no longer requires an allocated block of System Heap, it may release it by sending inter-board message $001C_{16}$, Release Memory to System Heap, to the Primary 575. This can be sent after power fail recovery completes. No provisions are made to release memory using the PURCMD facility.

7. INTERBOARD MESSAGES

This section describes each of the SIMATIC 575 interboard messages. The message descriptions are in numerical order by message type. Each description contains the following components:

Sent by	Specifies 'who' can send the message. In general, a message can be sent from a board or from an application on a board. Some messages restrict the possible senders.
Sent to	Specifies 'who' can process the message. In general, a given message can be sent to a board or to an application on a board. Some messages can only be processed by the Primary 575 CPU (board).
Type	Specifies the value for the message type (IBMTYPE) field.
Request	Specifies the request format as a sequence of letters and spaces. Each letter corresponds to 4 data bits (e.g., cc represents an 8-bit byte - 2 consecutive 4-bit units composing a single field, pppp represents a 16-bit word). A sequence one or more of the same letter represents a single field, e.g., qqqq represents a Port_ID field. Spaces are used to improve readability of the specification. They do not occur in the actual message. Optional fields are shown enclosed in brackets, i.e., [cc]. The braket is not included when the field is present.
Response	Specifies the format of the message's response data. Specification convention is as for the Request component.
Errors	Lists the errors that the message may return. A description of error codes that can be returned is contained in Section 11, IBC Error Codes.
Parameters	Describes the parameters of the Request and/or Response components.

7.1 Message 0001_{16} - Configure Port

This message is used to set up a port for operation.

Sent by	Anyone.
Sent to	Board containing the port to be configured.
Type	0001_{16}
Request	qqqq pppp bbbbbbbb cc pp ss ff
Response	none.
Errors	No such port. Specified protocol not supported or invalid. Specified BAUD rate not supported or invalid. Specified data-bits/parity not supported or invalid. Specified flow control not supported or invalid.

Parameters

qqqq Port ID (LAR / Port number) (see PORT_ID in Section 9, Data Dictionary).

pppp Protocol as follows

0000_{16}	None
0001_{16}	TTY
0002_{16}	NITP/TBP Secondary (Task Codes)
0003_{16}	NITP/TBP Host (Future)
0004_{16} - $FFFF_{16}$	undefined

Refer to document SIMATIC 575 Task Code Specification (PPX:575-8104) for a description of NITP and TBP protocols. TBP and NITP are request/response protocols used in task code communications. TTY indicates that the port is used for unstructured communications (e.g., the port is a printer port).

bbbbbbbb BAUD rate in bits/second as a binary integer, i.e., 19200 for 19.2K baud

cc Character size not including any parity or stop bits. Typically 7 or 8.

pp Parity as follows:

00	No parity
01	EVEN parity.
02	ODD parity.
03	MARK (1) parity.
04	SPACE (0) parity.

ss Number of stop bits as follows:

00	Bit synchronous.
01	1 stop bit.
02	1.5 stop bits.
03	2 stop bits.

ff Flow control options (bit mask) as follows:

bit 2^7	DSR/DTR	If not selected then if DSR is deasserted, the operation is aborted.
bit 2^6	XON/XOFF	Cannot be selected if TBP is selected.

Zero, one or both bits can be set.

7.2 Message 0002_{16} - Read Port Configuration

Sent by	Anyone.
Sent to	Board containing the port.
Type	0002_{16}
Request	qqqq
Response	qqqq pppp bbbbbbbb cc pp ss ff
Errors	No such port.

Parameters

qqqq Port ID.

pppp Protocol as follows

0000_{16} = None
0001_{16} = TTY.
0002_{16} = NITP/TBP Secondary (Task Codes)
0003_{16} = NITP/TBP Host (Future)
0004_{16} - FFFF_{16} undefined

Refer to document SIMATIC 575 Task Code Specification (PPX:575-8104) for a description of NITP and TBP protocols. TBP and NITP are request/response protocols used in task code communications. TTY indicates that the port is used for unstructured communications (e.g., the port is a printer port).

bbbbbbbb BAUD rate in bits/second, as a binary integer..

cc Character size not including any parity and stop bits. Typically 7 or 8.

pp Parity ss follows

00 - No parity.
01 - EVEN parity.
02 - ODD parity.
03 - MARK parity (MSB = 1)
04 - SPACE parity (MSB = 0)

ss Number of stop bits as follows

00 - Bit synchronous.
01 - 1 stop bit.
02 - 1.5 stop bits.
03 - 2 stop bits.

ff Flow control options (bit mask) as follows:

bit 2^7	DSR/DTR	If not selected then if DSR is deasserted, the operation is aborted.
bit 2^6	XON/XOFF	Cannot be selected if TBP is selected.

Zero, one or both bits can be set.

7.3 Message 0003_{16} - Connect To Port

This message is used to connect an application to a port.

Sent by	Anyone.
Sent to	Board containing the port.
Type	0003_{16}
Request	qqqq aa [cc]
Response	none
Errors	No such port. No such application. Connection failed. Timeout not supported.

Parameters	qqqq	Port ID.
	aa	App ID of the application to be connected to the port. If not coded or 00_{16} then the requestor is connected to the port. (The purpose of this parameter is to allow the user to statically configure his port connections).
	cc	Connection type as follows 00_{16} EXCLUSIVE connection. Only a single application may have an exclusive connection to a port. 01_{16} SHARED connection. Multiple applications may have shared connections to a port. If not coded, then EXCLUSIVE is assummed. Currently, the only protocol that supports shared connections is TTY. For protocols not supporting shared connections, a request for a shared connection is treated like a request for an exclusive connection.

7.4 Message 0004_{16} - Disconnect From Port

This message is used to cancel a port connection.

Sent by	Application wishing to disconnect from the port.	
Sent to	Board containing the port.	
Type	0004_{16}	
Request	qqqq [aa]	
Response	none.	
Errors	No such port.	
	Application was not connected to the port.	
Parameters	qqqq	Port ID.
	aa	App ID of the application whose connection is to be canceled. If not coded or if 00_{16} then the requesting application is disconnected.

7.5 Message 0005_{16} - Read Port Connections

This message is used to read a port connection.

Sent by	Anyone.
Sent to	Board containing the port.
Type	0005_{16}
Request	qqqq
Response	[aa cc [aa cc] ...]
Errors	No such port.

Parameters	qqqq	Port ID.
	aa	App ID of the application currently connected to the port.
	cc	Connection type as follows 00_{16} - EXCLUSIVE connection. 01_{16} - SHARED connection.

7.6 Message $001B_{16}$ - Allocate Memory From System Heap

This message is used for a board or application to obtain memory from System Heap.

Sent by	Anyone.
Sent to	Board containing Primary 575 function.
Type	$001B_{16}$
Request	nnnnnnnn
Response	pppppppp
Errors	Memory not available.

Parameters	nnnnnnnn	Number of bytes of memory to be allocated.
	pppppppp	VME pointer to the first usable byte for the block of memory that was allocated. This is the first byte after the header. (See record SHEAPHDR in Section 9, Data Dictionary).

7.7 Message 001C$_{16}$ - Release Memory To System Heap

This message allows a board or application to release memory that was previously allocated by message $001B, "Allocate Memory From System Heap".

Sent by	Anyone	
Sent to	Board containing Primary 575 function.	
Type	001C$_{16}$	
Request	[pppppppp]	
Response	none	
Errors	Specified memory was not allocated by message 001B$_{16}$, "Allocate Memory From System Heap".	
Parameters	pppppppp	VME pointer to the first usable byte of the block of memory to be released. If not coded then all System Heap allocated to the board or application is released.

7.8 Message 001D$_{16}$ - Report Available Memory

This message is used to determine the amount of memory available to be allocated from System Heap.

Sent by	Anyone.
Sent to	Board containing Primary 575 function.
Type	001D$_{16}$
Request	none
Response	nnnnnnnn llllllll aaaaaaaa ffffffff
Errors	none

Parameters	nnnnnnnn	Total number of usable bytes of System Heap available.
	llllllll	Size of the largest block of usable bytes of System Heap available to be allocated.
	aaaaaaaa	Total number of usable bytes of System Heap that is allocated to the board or application.
	ffffffff	VME pointer to the first block of System Heap that is allocated to the board or application. Points to the first usable byte. If multiple blocks are allocated, the sender is responsible for finding other blocks using SHEAPHDR.

7.9 Message 001E$_{16}$ - Execute Application Task

This message is used to request execution of an application-defined task by another board or application. For 575 CPUs or 575 applications, this is a Series 500 task code.

Note	When a port is configured for NITP or TBP protocol, all messages received on the port are passed to the connected application as "Execute Application Task" messages.

Sent by	Anyone.	
Sent to	Anyone.	
Type	001E$_{16}$	
Request	qqqq 00 dd...dd	
Response	qqqq 00 dd...dd	
Errors	Message not supported.	
Parameters	qqqq	Port id of the sending board.
	dd...dd	Request/response data. The format of the data is dependent on the receiver. For a SIMATIC 575 CPU, the data is defined by document SIMATIC 575 Task Code Specification (PPX:575-8104).

8. INTERBOARD COMMUNICATIONS UTILITIES

This section describes utility routines referenced by the pseudo-code descriptions of the interboard messages.

8.1 C$ENQMSG - Enque Message To IBC Area

Procedure C$ENQMSG is called by function C$SEND to enqueue a message on a destination board's IBC Message Queue. This procedure has two parameters:

input	DEST_BRD	Pointer to destination board's Board Table Entry.
input	MSG	Pointer to the message to be sent.

The pseudo-code for procedure C$ENQMSG follows:

```
/-----------------------------------------------------------------------\
C$ENQMSG(in DEST_BRD : VME_ptr to BRDTBLEN,
         in MSG      : VME_ptr to IBCMSG);

var
   DEST_QUE : VME_ptr to IBCAREA;
   LAST_MSG : VME_ptr to IBCMSG;
   OBTAINED : boolean;

begin
   MSG->IBMNEXT := nil;
   DEST_QUE := DEST_BRD->BTEIBCAP;
   MU$GET(DEST_QUE->IBAMUTEX, no_timeout, OBTAINED);
   LAST_MSG := DEST_QUE->IBATAIL;
   if LAST_MSG = nil then
      DEST_QUE->IBAHEAD := MSG
   else
      LAST_MSG->IBMNEXT := MSG
   endif;
   DEST_QUE->IBATAIL := MSG
end.
\-----------------------------------------------------------------------/
```

8.2 C$REPLY - Send Reply To An IBC Message

Procedure C$REPLY will send a reply to an IBC message. This is done by setting the "is response" flag (field IBMISRSP) of the IBC message and, if the board supports it, enqueing the message on the board's IBC Message Queue. C$REPLY has a single parameter:

input	MSG	Pointer to the message to which the reply is to be sent.

The pseudo-code for C$REPLY follows:

```
/----------------------------------------------------------------------\
C$REPLY(in MSG : VME_ptr to IBCMSG);

begin
   MSG->IBMISRSP := true;
   if MSG->IBMSLAR <> FF$_{16}$ then
      C$SEND(MSG->IBMSLAR, MSG)
   endif;
   return
end.
\----------------------------------------------------------------------/
```

8.3 C$SEND - Send Interboard Message

Procedure C$SEND is called by procedures C$SENDAA, C$SENDAB, C$SENDBA, C$SENDBB, and C$REPLY to send an IBC message. The method used to send the message is dependent on the IBC message protocol specified in the destination board's Board Table entry (field IBCPROT). C$SEND has three parameters, as follows:

input	LAR	LAR of the board to which the IBC message is to be sent.
input	MSG	Pointer to the IBC message to be sent.
output	RESULT	Result of the send operation. (See 11, IBC Error Codes.)

The pseudo-code for C$SEND follows:

```
/-----------------------------------------------------------------------\
C$SEND(in  DEST_LAR : LAR
       in  MSG      : VME_ptr to IBCMSG)
       out RESULT   : uint16);

var
   DEST_BRD     : VME_ptr to BRDTBLEN;
   DEST_QUE     : VME_ptr to IBCAREA;
   DEST_INTR_16 : Short_VME_ptr to byte;
   DEST_INTR_24 : VME_ptr to byte;

begin
   DEST_BRD := loc(GSDA->BRDTBL[DEST_LAR]);
   if DEST_BRD->BTELAR <> DEST_LAR then
      RESULT := DESTINATION_BOARD_NOT_LOGGED_IN;
      return
   endif;
   DEST_QUE := DEST_BRD->BTEIBCAP;
   case DEST_BRD->BTEIBCPR of
      IBCPSTD :   /* Standard interboard messages with interrupt */
         C$ENQMSG(DEST_BRD, MSG);
         /* Determine method of interrupt */
         case DEST_BRD->BTEINTAC of
            A16RMW :    /* A16 space Read-modify-write cycle access */
               DEST_INTR := A16_Address(DEST_BRD->BTEINTRP);
               begin ATOMIC operation;
                  *DEST_INTR := (*DEST_INTR & DEST_BRD->BTEINTRAM) | DEST_BRD->BTEINTROM;
               end ATOMIC operation;

            A16W :      /* A16 space write cycle access */
               DEST_INTR  := A16_Address(DEST_BRD->BTEINTRP);
               *DEST_INTR := DEST_BRD->BTEINTROM;

            A24RMW :    /* A24 space Read-modify-write cycle access */
               DEST_INTR := A24_Address(DEST_BRD->BTEINTRP);
               begin ATOMIC operation;
                  *DEST_INTR := (*DEST_INTR & DEST_BRD->BTEINTRAM) | DEST_BRD->BTEINTROM;
               end ATOMIC operation;

            A24W :      /* A24 space write cycle access */
               DEST_INTR := A24_Address(DEST_BRD->BTEINTRP);
               *DEST_INTR := DEST_BRD->BTEINTROM;

            Otherwise : /* No interrupt supported */
               MU$REL(DEST_QUE->IBAMUTEX);
               RESULT := UNKNOWN_INTERRUPT_ACCESS_CODE_SPECIFIED_IN_BRD_TBL;
               return;
         endcase;
         MU$REL(DEST_QUE->IBAMUTEX);
      IBCPSTDP :  /* Standard interboard messages w/o interrupt */
         C$ENQMSG(DEST_BRD, MSG);
         MU$REL(DEST_QUE->IBAMUTEX);

      otherwise : /* Board cannot accept messages */
         RESULT := UNKNOWN_PROTOCOL_SPECIFIED_IN_BRDTBL_ENTRY;
         return;
   endcase;
   RESULT := NO_ERROR;
   return
end.
\---------------------------------------------------------------------/
```

8.4 C$SENDAA - Send Application-To-Application Message

Procedure C$SENDAA is used to send an IBC message from one application to another application. C$SENDAA will fill in the message header and enqueue the message on the receiver's IBC Message Queue. C$SENDAA has three parameters:

input	DEST_APP	APP_ID of application to which the IBC message is to be sent.
input	MSG	Pointer to the IBC message to be sent.
output	RESULT	Result of the send operation. (See 11, IBC Error Codes.)

The pseudo-code for C$SENDAA follows:

```
/-----------------------------------------------------------------------\
C$SENDAA(in  DEST_APP : APP_ID,
         in  MSG      : VME_ptr to IBCMSG)
         out RESULT   : uint16);

import
   MY_APP_ID : APP_ID;
   MY_LAR    : LAR;

begin
   if APPTBL[DEST_APP].ATEAPPID <> DEST_APP then
      RESULT := APPLICATION_IS_NOT_DEFINED");
      return
   endif;

   MSG->IBMISRSP := false;
   MSG->IBMSLAR := MY_LAR;
   MSG->IBMSAPP := MY_APP_ID;
   MSG->IBMRLAR := APPTBL[DEST_APP].ATELAR;
   MSG->IBMRAPP := DEST_APP;
   C$SEND(MSG->IBMRLAR, MSG, RESULT);
   return
end.
\-----------------------------------------------------------------------/
```

8.5 C$SENDAB - Send Application-To-Board Message

Procedure C$SENDAB is used to send an IBC message from an application to a board. C$SENDAB will fill in the message header and enqueue the message on the receiver's IBC Message Queue. C$SENDAB has three parameters:

input	DEST_LAR	LAR of the board to which the IBC message is to be sent.
input	MSG	Pointer to the IBC message to be sent.
output	RESULT	Result of the send operation. (See 11, IBC Error Codes.)

The pseudo-code for C$SENDAB follows:

```
/-----------------------------------------------------------------------\
C$SENDAB(in  DEST_LAR : LAR,
         in  MSG      : VME_ptr to IBCMSG)
         out RESULT   : uint16);

import
   MY_APP_ID : APP_ID;
   MY_LAR    : LAR;

begin
   MSG->IBMISRSP := false;
   MSG->IBMSLAR := MY_LAR;
   MSG->IBMSAPP := MY_APP_ID;
   MSG->IBMRLAR := DEST_LAR;
   MSG->IBMRAPP := 0;
   C$SEND(MSG->IBMRLAR, MSG, RESULT);
   return
end.
\-----------------------------------------------------------------------/
```

8.6 C$SENDBA - Send Board-To-Application Message

Procedure C$SENDBA is used to send an IBC message from a board to an application. C$SENDBA will fill in the message header and enqueue the message on the receiver's IBC Message Queue. This procedure has three parameters:

input	DEST_APP	APP_ID of application to which the IBC message is to be sent.
input	MSG	Pointer to the IBC message to be sent.
output	RESULT	Result of the send operation. (See 11, IBC Error Codes.)

The pseudo-code for C$SENDBA follows:

```
/-------------------------------------------------------------------------\
C$SENDBA(in  DEST_APP : APP_ID,
         in  MSG      : VME_ptr to IBCMSG,
         out RESULT   : uint16);

import
   MY_LAR    : LAR;

begin
   if GSDA->APPTBL[DEST_APP].ATEAPPID <> DEST_APP then
      RESULT := APPLICATION_IS_NOT_DEFINED;
      return
   endif;

   MSG->IBMISRSP := false;
   MSG->IBMSLAR := MY_LAR;
   MSG->IBMSAPP := 0;
   MSG->IBMRLAR := GSDA->APPTBL[DEST_APP].ATELAR;
   MSG->IBMRAPP := DEST_APP;
   C$SEND(MSG->IBMRLAR, MSG, RESULT);
   return
end.
\-------------------------------------------------------------------------/
```

8.7 C$SENDBB - Send Board-To-Board Message

Procedure C$SENDBB is used to send an IBC message from one board to another board. C$SENDBB will fill in the message header and enqueue the message on the receiver's IBC message queue. This procedure has three parameters:

input	DEST_LAR	LAR of the board to which the IBC message is to be sent.
input	MSG	Pointer to the IBC message to be sent.
output	RESULT	Result of the send operation. (See 11, IBC Error Codes.)

The pseudo-code follows:

```
/-----------------------------------------------------------------------\
C$SENDBB(in  DEST_LAR : LAR,
         in  MSG      : VME_ptr to IBCMSG,
         out RESULT   : uint16);

import
   MY_LAR    : LAR;

begin
   MSG->IBMISRSP := false;
   MSG->IBMSLAR := MY_LAR;
   MSG->IBMSAPP := 0;
   MSG->IBMRLAR := DEST_LAR;
   MSG->IBMRAPP := 0;
   C$SEND(MSG->IBMRLAR, MSG, RESULT);
   return
end.
\-----------------------------------------------------------------------/
```

8.8 MU$GET - Obtain A Mutex

Function MU$GET is used to obtain a mutex. This is done by performing a "test_and_set" on the mutex until either we obtain the mutex or we time out.

Note: To prevent excessive VMEbus traffic, a short (16 microsecond) delay is done between each attempt to obtain the mutex.

Procedure MU$GET has three parameters:

inout	MUTEX	Mutex to be obtained
input	TIMEOUT	Maximum amount of time that we are to wait for the mutex to become available. A value of maxval(clock_ticks) indicates "wait forever".
output	OBTAINED	Flag indicating whether or not we obtained the mutex.

The pseudo-code follows:

```
/----------------------------------------------------------------------\
MU$GET(inout  MUTEX    : mutex,
       in     TIMEOUT  : clock_ticks,
       output OBTAINED : boolean);

var
   START_TIME   : clock_ticks;
   ELAPSED_TIME : clock_ticks;

begin
   OBTAINED := false;
   ELAPSED_TIME := 0;
   START_TIME := get_time();
   repeat
      if not test_and_set(MUTEX) then
         OBTAINED := true
      else
         delay for 16 microseconds
      endif;
      if TIMEOUT <> maxval(clock_ticks) then
         ELAPSED_TIME := get_time() - START_TIME
      endif;
   until OBTAINED or (ELAPSED_TIME > TIMEOUT);
   return
end.
\----------------------------------------------------------------------/
```

8.9 MU$REL - Release A Mutex

Procedure MU$REL is used to release a mutex. MU$REL has a single parameter:

inout	MUTEX	Mutex to be released.

The pseudo-code follows:

```
/----------------------------------------------------------------------\
MU$REL(inout MUTEX : mutex);

begin
   MUTEX := 00₁₆;
   return
end.
\----------------------------------------------------------------------/
```

9. DATA DICTIONARY

This section contains the type definitions, record definitions, and variables that were defined in this document. The entries are in alphabetical order.

Note: Offsets are given as hexadecimal values.

9.1 Type APP_ID : Application ID.

An application ID is an 8-bit value in the range [0 ... 26] used to identify an application. The value 0 indicates either "current application" or "no application" depending on the context in which it occurs. It is used to index the Application Table Entry (see APPTBL).

```
type APP_ID : uint8 range [0 ... 26];
```

Externally, applications are referenced by a letter with 'A' corresponding to APP_ID = 1, 'B' corresponding to APP_ID = 2, etc.

9.2 Type APPOPMOD : Application Operational Mode.

Indicates the operational mode of an application.

```
type APPOPMOD : sint16 values
                                        { AOMFAULT = -1,
                                          AOMPGM = 0,
                                          AOMRUN = 1 };
```

where

AOMFAULT	Failed. A hardware or software fault has been detected.
AOMPGM	Program. This state indicates that the application is not executing.
AOMRUN	Run. This state indicates that the application is executing.

9.3 Variable APPTBL : VME_ptr To The Application Table.

Points to the Application Table. Variable APPTBL resides in the Global System Data Area (GSDA).

```
APPTBL : VME_ptr to APPTBLEN[1..26];
```

9.4 Record APPTBLEN : Application Table Entry.

Describes an entry in the Application Table (see APPTBL). Refer to Figure 1-5 for a graphical representation of the APPTBLEN record type.

```
          record APPTBLEN
0000           ATEMUTEX      : mutex;
0001           ATEAPPID      : APP_ID;
0002           ATELAR        : LAR;
0003           filler        : uint8; // must be zero.
0004           ATETYPE       : APPTYPE;
0006           ATEOPMOD      : APPOPMOD;
0008           ATEABRDS      : packed boolean[0..31];
000C           ATEGMEMS      : VME_ptr;
0010           ATEGMEME      : VME_ptr;
0014           ATEREQ        : packed boolean[0..31];/* bit 0 reserved */
0018           ATEOPT        : packed boolean[0..31];/* bit 0 reserved */
001C           ATEMDLOK       : packed boolean[0..31];/* bit 0 reserved */
0020           ATEDESC       : char[1..40]
0048      endrecord
```

where

ATEMUTEX	Mutex used to control access to the entry. This mutex must be obtained beforecreating, deleting, or modifying the application table entry.
ATEAPPID	Application ID. 0 indicates that this application is not present.
ATELAR	LAR of the board to which all applications-directed messages are to be sent.
ATETYPE	Type of application.
ATEOPMOD	Application's operational mode.
ATEABRDS	Attached boards map. Bit-map indicating which boards in addition to ATELAR are attached to this application with LAR = 0 corresponding to the msbit. (This field is provided for future expansion and will always be 0 for now).
ATEGMEMS	Address of the start of the application's G-memory area. 0 indicates that the application does not have any G-memory.
ATEGMEME	Address of the end of the application's G-memory area. 0 indicates that the application does not have any G-memory.
ATEREQ	Required applications map. Bit-map indicating which other applications are required to be present for this application to run. Indexed by APP_ID.
ATEOPT	Optional applications map. Bit-map indicating which other applications this application may reference but are not required to be present for this application to run. Indexed by APP_ID.
ATEMDLOK	Locked modes list. Bit-map indicating which applications' operational modes are locked to this application's operational mode. A request to change the operational mode of any of the locked applications will result in the operational mode of all locked applications being changed. Indexed by APP_ID.
ATEDESC	ASCII string(zero terminated) describing the application.

9.5 Type APPTYPE : Application Type Code.

Specifies the type of an application.

type APPTYPE : uint16 values { APPRLLSF = 0001_{16}, ... };

where

APPRLLSF : Series 500 RLL / SF/Loop program.

Currently, the only defined APPTYPE is 0001_{16}. Siemens Industrial Automation, Inc. reserves type codes 0001_{16} through $7FFF_{16}$ for registered application types. 8000_{16} through $FFFF_{16}$ are reserved for customer use.

9.6 Variable BATGOOD : "Battery Good" Flag.

Indicates the status of the battery. Variable BATGOOD resides in the Global System Data Area (GSDA).

BATGOOD : boolean;

9.7 Type BOARD_TYPE : Board ID Code.

Specifies the type of board installed.

type BOARD_TYPE: uint8 values

{ 00_{16} - 02_{16} reserved
03_{16} : SIMATIC VME I/O
04_{16} : SIMATIC 575 CPU
05_{16} - 06_{16} reserved
07_{16} : General Purpose CPU
08_{16} - $7F_{16}$ Reserved for Siemens definition
80_{16} - FF_{16} User defined }

9.8 Type BOOLEAN : Boolean.

Specifies a boolean variable that can take on one of two values, true or false.

false = 0
true = non-zero

9.9 Variable BRDTBL : VME_ptr To The Board Table.

Points to the Board Table. Variable BRDTBL resides in the Global System Data Area (GSDA).

BRDTBL : VME_ptr to BRDTBLEN[0..15];

9.10 Record BRDTBLEN : Board Table Entry.

Describes an entry in the Board Table (see BRDTBL). Figure 1-4 provides a graphical representation of this data structure.

```
         record BRDTBLEN
0000           BTELAR       : LAR;
0001           BTEBID       : BOARD_TYPE;
0002           BTEMGOOD     : BOOLEAN;
0003           BTEPFRDN     : BOOLEAN;
0004           BTEIBCPR     : IBCPROT;
0006           BTEPUTO      : uint16;
0008           BTESDT       : uint16;
000A           BTEDIAG      : uint16;
000C           BTEIBCAP     : VME_ptr to IBCAREA;
0010           BTEINTRP     : uint32;
0014           BTEINTAC     : INTRCODE;
0016           BTEINTAM     : uint8;
0017           BTEINTOM     : uint8;
0018           BTEFLAGS     : packed boolean[0..31]
001C           BTESDESC     : char[1..20]
0030           BTEHDESC     : char[1..20]
0044           BTEPWRKA     : char[1..128]
00C4     endrecord
```

where

BTELAR	Board's LAR. FF_{16} indicates that the board is not present.
BTEBID	Board's type (see BOARD_TYPE for definitions).
BTEMGOOD	Flag used during power-up/reset to indicate whether or not the board's memory is still good.
BTEPFRDN	Flag used during power-up/reset to indicate when the board has completed power-fail recovery processing.
BTEIBCPR	Interboard communications protocol to be used (see Type IBCPROT).
BTEPUTO	Number of seconds that the System Controller will wait for the board to log itself in during power-up/reset.
BTESDT	Index of the board's entry in the System Descriptor Table (see variable SDT).
BTEDIAG	Diagnostic word that indicates reason for failure. Value of 1 indicates no failure.
BTEIBCAP	Pointer to the board's Interboard Communications Area (see record IBCAREA), if any. If BTEIBCAP is 0 then this board cannot receive messages.
BTEINTRP	Byte offset into either VME A-16 or A-24 address space. Points to a location on the board that causes an interrupt if accessed using the method indicated by variable BTEINTAC.
BTEINTAC	Interrupt Location Access Code that indicates how to initiate a board interrupt to signal that a message has been enqueued on the boards message queue. (see Type INTRCODE)
BTEINTAM	Mask to be "ANDed" with the value read from the interrupt location if the access code indicates read-modify-write.

BTEINTOM	Mask to be "ORed" with the value read from the interrupt location if the access code indicates read-modify-write or the value to be written if access code indicates write.
BTEFLAGS	Board attribute flags as follows
	bit 2^{31} — BTEF_PFR. Participant in Power-Fail- Recovery. If cleared and board was present at power-down then on battery good power up, board will participate in power fail recovery. Otherwise, board does not participate.
	bit 2^{30} — BTEF_PFRE. Power-Fail-Recovery-Error. A power fail recovery error was detected during login.
	bits 2^{29} - 2^{0} — Reserved -- set to 0.
BTESDESC	ASCII string (zero terminated) describing the software on the board. SIMATIC 575 CPU's record the Software Configuration Number in this field.
BTEHDESC	ASCII string (zero terminated) describing the hardware of the board. SIMATIC 575 CPU's record "575" in this field.
BTEPWRKA	Private work area. This area is provided for the board's private use. Intended purpose is as a guaranteed battery backed IBC message header/text area and/or Power-up Recovery save area.

9.11 Variable BRDTBLMU : Board Table Mutex.

Used to control access to the Board Table (BRDTBL). Variable BRDTBLMU resides in the Global System Data Area (GSDA).

BRDTBLMU : mutex;

Before a board attempts to log itself into the system, it should obtain BRDTBLMU.

9.12 Type CLOCK_TICKS : Clock Ticks.

Indicates a number of clock ticks. This is the data type returned by function "get_time".

9.13 Record GCSR : Global Control And Status Register.

Describes the GCSR area for SIMATIC 575 CPU's. GCSR is an 8 byte status region that provides facilities that allow other boards to interrupt the destination board.

```
            record GCSR
0000            filler1         : uint8; // even addresses not defined.
0001            GCSR_R0         : uint8;
0002            filler2         : uint8; // even addresses not defined.
0003            GCSR_R1         : uint8;
0004            filler3         : uint8; // even addresses not defined.
0005            GCSR_BID        : uint8;
0006            filler4         : uint8; // even addresses not defined.
0007            GCSR_GR0        : uint8;
0008            filler5         : uint8; // even addresses not defined.
0009            GCSR_GR1        : uint8;
000A            filler6         : uint8; // even addresses not defined.
000A            GCSR_GR2        : uint8;
000C            filler7         : uint8; // even addresses not defined.
000B            GCSR_GR3        : uint8;
000E            filler8         : uint8; // even addresses not defined.
000F            GCSR_GR4        : uint8;
0010        endrecord
```

where

GCSR_R0 Location monitor masks, chip ID as follows

bits 2^7 - 2^4	GCR0_LM	Location monitor masks. Not used.
bits 2^3 - 2^0	GCR0_CID	Chip ID. Not used.

GCSR_R1 Interrupt / control flags as follows

bit 2^7	GCR1_RH	Reset and hold. Allows other VME masters to reset the board. (read/write).
bit 2^6	GCR1_SCN	System Controller flag. Set if the board is the System Controller. (read only).
bit 2^5	GCR1_ISF	Inhibit SYSFAIL. Allows other VME masters to cause the board to release its input to the SYSFAIL* line. (read/write).
bit 2^4	GCR1_FL	Board Fail. Shows the status of the BRDFAIL* signal. (read only).
bits 2^3 - 2^2	Undefined	
bit 2^1	GCR1_SHP	High-priority signal interrupt. Not used.
bit 2^0	GCR1_SLP	Low-priority signal interrupt. When set causes a low-priority signal interrupt on the board. This interrupt is used to indicate that an IBC message is present on the board's IBC Message Queue

GCSR_BID	Board ID.
GCSR_GRn	General purpose control and status registers. Not used.

GCSR is located in short VME address space (A16).

9.14 Record GSDA : Global System Data Area.

Defines the global system data area. This is the primary data area used in interboard communications since it contains pointers to all the other interboard communications data structures. See Figure 1-2 for a graphical representation of this data structure.

When the Primary 575 CPU is configured as **AUTO-CONFIGURED** the GSDA data structure resides on the Primary 575 CPU at VME A24 address 000000_{16}.

If the Primary 575 CPU is configured as **USER-CONFIGURED** then the GSDA resides at a VME A24 base address determined by the selection of SW3 and SW4 on the 505 Remote I/O CPU Annex Card. See the 575 System Manual (PPX:575-8101-5) for more information.

```
               record GSDA
0000              GSDAFMT        : uint16[0 .. 1];
0004              BATGOOD        : BOOLEAN;
0005              filler1        : uint8;
0006              PURDONE        : BOOLEAN;
0007              PURPHASE       : uint8;
0008              PURCMD         : uint32;
000C              PURRESP        : uint32;
0010              SDTNMLAR       : uint8;
0011              SDTNSLAR       : uint8;
0012              SDTNSDT        : uint16;
0014              SDT            : VME_ptr to SDTENT[0 .. 31];
0018              filler2        : uint8[0 .. 2]; // always 0.
001B              BRDTBLMU       : mutex;
001C              BRDTBL         : VME_ptr to BRDTBLEN[0 .. 15];
0020              APPTBL         : VME_ptr to APPTBLEN[1 .. 26];
0024              filler3        : uint8[0 .. 2]; // always 0.
0027              SYSTODMU       : mutex;
0028              SYSTOD         : Time_of_Day
0030              filler4        : uint8;
0031              SBRDTMUX       : mutex;
0032              SBRDNSBR       : uint16;
0034              SBRDTBL        : VME_ptr to SBRDTBLE[0..15];
0038              filler5        : uint8; // always 0.
0039              SYSIOCMU       : mutex;
003A              SYSIOCNE       : uint16;
003C              SYSIOCTB       : VME_ptr to SYSIOCEN[0..15];
0040              IAPPLMAP       : packed BOOLEAN [0..31];
0044              SYSCONFG       : uint8;
0045              filler6        : uint8[0 .. 2];
0048              PFSAVAR        : uint32[0 .. 15];
0088              filler7        : uint32[0 .. 46] /* reserved for future expansion */
0144       endrecord
```

9.15 Variable GSDAFMT : Global System Data Area Format Version Number.

Indicates the format version of the GSDA. Variable GSDAFMT resides in the Global System Data Area.

```
GSDAFMT : uint16[0..1];
```

GSDAFMT indicates the format of the Global System Data Area and related data structures such as the Board Table and the Application Table. GSDAFMT[0] is the major version number and is incremented whenever a major change has been made that makes prior version of the GSDA area incompatible with each other. All boards in the system should check GSDAFMT[0] to ensure that it matches what they expect. See Section 10, GSDA Format Changes, for the correct value for GSDAFMT[0].

GSDAFMT[1] is the minor version number. It is incremented whenever a minor change has been made which should not affect the operation of boards running with older software having the same value for GSDAFMT[0].

9.16 Variable IAPPLMAP : Installed Application Map

Indicates which applications logged in on the last powerup. Variable IAPPLMAP resides in the Global System Data Area.

```
IAPPLMAP : packed BOOLEAN [0..31] /* bit 0 reserved */
```

IAPPLMAP is indexed by application id. APP_ID = 1 corresponds to bit 1. This map is built at the end of power-up / reset phase 2. This map is used to help determine which applications have been installed in a battery good system.

9.17 Record IBCAREA : Interboard Communications Area.

Describes the Interboard Communications Area located on each board. A board's IBCAREA is pointed to by field BTEIBCAP in the board's Board Table entry (see record BRDTBLEN).

```
          record IBCAREA
  +0000           IBAMUTEX : mutex;
  +0001           filler : uint8[3]; // reserved, must be zero.
  +0004           IBAHEAD : VME_ptr to IBCMSG;
  +0008           IBATAIL : VME_ptr to IBCMSG;
  +000C   endrecord
```

where

IBAMUTEX	IBC message mutual exclusion semaphore.
IBAHEAD	Pointer to first (oldest) entry on the board's IBC Message Queue.
IBATAIL	Pointer to last (youngest) entry on the board's IBC Message Queue.

9.18 Record IBCMSG : Interboard Message.

Describes the buffer used to send interboard messages.

```
          record IBCMSG
0000           IBMNEXT       : VME_ptr to IBCMSG;
0004           IBMRWRKA      : uint8[16];
0014           IBMSWRKA      : uint8[16];
0024           IBMSLAR       : LAR;
0025           IBMSAPP       : APP_ID;
0026           IBMRLAR       : LAR;
0027           IBMRAPP       : APP_ID;
0028           IBMISRSP      : BOOLEAN;
0029           filler        : uint8;
002A           IBMTYPE       : uint16;
002C           IBMRESP       : uint16;
002E           IBMDSIZE      : uint16;
0030           IBMDLNT       : uint16;
0032           filler        : uint16;
0034           IBMREQDP      : VME_ptr to Request Data Area
0038           IBMRSPDP      : VME_ptr to Response Data Area
003C      endrecord;
```

where

IBMNEXT	Ptr to next entry on the board's IBC Message Queue.
IBMSWRKA	16-byte sender workarea.
IBMRWRKA	16-byte receiver workarea.
IBMSLAR	LAR of the sending board.
IBMSAPP	Sender's Application ID if message was initiated by an application. 0 if message was not initiated by an application.
IBMRLAR	LAR of the destination board
IBMRAPP	Destination Application ID if the message is directed to an application. 0 if message is directed to the board.
IBMISRSP	"Is response" flag. If non-zero then this is a message response. If 0 then this is a message.
IBMTYPE	Message type.
IBMRESP	Response code.
IBMDSIZE	Size of the response data area (pointed to by IBMRSPDP) in bytes. This is the maximum size of any response that may be returned.
IBMDLNT	Actual length of the request/response data in bytes.
IBMREQDP	Pointer to the request data area.
IBMRSPDP	Pointer to the response data area. This pointer may be the same as IBMREQDP. This would be the case if the response data is to be returned in the request data buffer.

9.19 Type IBCPROT : Interboard Communications Protocol.

Specifies the protocol to send messages to a board.

```
type IBCPROT : uint16 values {
                    IBCPSTD = 0,
                    IBCPSTDP = 1,
                    IBCRPOLL = 2 }
```

where

IBCPSTD	Standard communications protocol. Indicates that messages and responses are queued in the board's IBCAREA and an interrupt is generated by accessing the location pointed to by field BTEINTRP using the access method indicated by field BTEINTAC.
IBCPSTDP	Standard polled communications protocol. Indicates that messages and responses are queued in the board's IBCAREA. The board must poll field IBAHEAD to determine if anything is queued.
IBCRPOLL	No messages, polled responses. Indicates that the board cannot accept messages. For any messages that it sends, it must poll the "is response" field (IBMISRES) of the message to determine when the response is ready.

9.20 Type INTRCODE : Interrupt Location Access Code.

Specifies the access method to initiate an interrupt to a board indicating an IBC message has been sent to the board. The location to be accessed is pointed to by variable BTEINTRP (see record BRDTBLEN).

```
type INTRCODE : uint16 values {
                    A16RMW = 0,
                    A16W = 1,
                    A24RMW = 2,
                    A24W = 3 }
```

where

A16RMW	Location is accessed with a supervisor-mode read-modify-write cycle to the VME A16 address space. The value read shall be modified by anding it with field BTEINTAM and oring this result with field BTEINTOM of the destination board's board table entry.
A16W	Location is accessed with a supervisor-mode write cycle to the VME A16 address space. The value to be written is contained in field BTEINTOM.
A24RMW	Location is accessed with a supervisor-mode read-modify-write cycle to the VME A24 address space. The value read shall be modified by anding it with field BTEINTAM and oring this result with field BTEINTOM of the destination board's board table entry.
A24W	Location is accessed with a supervisor-mode write cycle to the VME A24 address space. The value to be written is contained in field BTEINTOM.

9.21 Type LAR : Board Logical Address.

A LAR is an 8-bit value specifying the logical address of a board. The Primary 575 has LAR = 00_{16}. The next VME master has LAR = 01_{16}, etc. The first VME slave card has LAR = 10_{16}, the next VME slave has LAR = 11_{16}, etc. The LAR is used to index the entries in the Board Table (see BRDTBL). LAR = FF_{16} indicates that the board is not present.

type LAR : uint8 range [00_{16} ... $1F_{16}$, FF_{16}];

9.22 Type MUTEX : Mutual Exclusion Semaphore.

Byte used to control access to shared data structures.

type mutex : byte;

Bit 2^7 of a mutex indicates its status as follows:

0 : Mutex is not held.

1 : Mutex is held.

Two operations are defined on a mutex. They are

MU$GET : Acquire a mutex using an indivisible read-modify-write operation.

MU$REL : Release a mutex.

9.23 Variable MY_APP_ID : Application ID Of The Current Application.

Used to reference the current application's Application ID.

MY_APP_ID : APP_ID;

9.24 Variable MY_LAR : LAR Of The Current Board.

Used to reference the current board's LAR.

MY_LAR : LAR;

9.25 Variable PFSAVAR : Power Fail Save Area

This area is maintained through a power failure. Each master is allocated 1 longword to use as it see fit. The index into this area is given by PFSAVAR[LAR]. Variable PFSAVAR resides in the Global System Data Area (GSDA).

PFSAVAR : uint32[0 .. 15];

9.26 Record PORT_ID : Port ID.

Identifies a port in the SIMATIC 575 system. PORT_ID is referred to as the "qqqq" field in the SIMATIC 575 Task Code Specification (PPX:575-8104). Its primary usage in this system is to identify the source of task codes for port-lockout, generic-upload/download, etc.

type PORT_ID : uint16;

The encoding of PORT_ID is as follows:

2^{15} ... 2^{8}	2^{7} ... 2^{0}			
0xxxxxxx	**0xxxxxxx**	Reserved values that are not used by 575.		
00LLLLLL	**10PPPPPP**	PORT_ID references a local communication port on the PLC, where:		
			LLLLLL	Board's LAR [00_{16} ... $0F_{16}$].
			PPPPPP	Port number (0 = first port).
00000000	**11AAAAAA**	PORT_ID references an application, where:		
			AAAAAA	APP_ID [01_{16} ... $1A_{16}$].
01LLLLLL	**1CCCBBBB**	PORT_ID references a remote port on an RBC, where:		
			LLLLLL	LAR of the board to which the RBC is attached.
			CCC	Remote channel number.
			BBBB	Base number.
1CCCCCCC	**BBBBSSSS**	PORT_ID refers to a module position on a Remote I/O channel, where:		
			CCCCCCC	Channel number -- always 00_{16} for 575.
			BBBB	Base number.
			SSSS	Slot number.
11111110	**LLLLLLLL**	PORT_ID refers to a board, where:		
			LLLLLLLL	Board's LAR -- 00_{16} - $0F_{16}$.

9.27 Type PTR : Pointer.

Specifies a pointer to an object. Zero (0) is recognized as the nil pointer, i.e., a pointer that doesn't point to anything.

9.28 Variable PURCMD : Primary 575 Command.

Used during power-up/reset to pass a command to the Primary 575. Variable PURCMD resides in the Global System Data Area (GSDA).

PURCMD : sint32;

A board must hold the Board Table mutex (BRDTBLMU) before it can pass commands to the Primary 575. All commands have the most-significant bit of the command longword set. The Primary 575 polls this bit to determine when a command is present, so the sending board should first load the command into PURCMD and then set the msbit. When the Primary 575 has finished processing a command, it places the response in variable PURRESP and then clears PURCMD.

The following Primary 575 commands are supported.

$81LLAA00_{16}$	Report memory allocation. "LL" and "AA" are the LAR and APP_ID for which memory allocation is to be reported. If "AA" is "00" then memory allocation is reported for the board, only. When PURCMD becomes 0 the VME_ptr to the first block of allocated memory for LLAA is returned in PURRESP. If no memory is allocated to LLAA then PURRESP will contain 0. (Memory is allocated using IBC message $001B_{16}$.)

NOTE: If more than one non-logged in board asks for heap then "AA" must be unique.

9.29 Variable PURDONE : "Board Login Complete" Flag.

Indicates that a board using the IRQ6 IACK daisy-chain method of logging in has completed logging itself in. Variable PURDONE resides in the Global System Data Area (GSDA).

PURDONE : boolean;

9.30 Variable PURPHASE : Power-up/Reset Phase.

Indicates the phase of power-up/reset. Variable PURPHASE resides in the Global System Data Area (GSDA).

PURPHASE : sint8;

The values taken on by PURPHASE during power-up/reset are as follows:

00_{16} - Initial phase.
01_{16} - Logging in boards using IRQ6 IACK daisy-chain.
02_{16} - Logging in boards with previously assigned LARs.
03_{16} - Doing power-fail recovery.
04_{16} - Doing a fault restart.
FF_{16} - Power-up/reset complete.

9.31 Type RLL_LOCK : RLL Lock.

A data structure in shared memory used to control access to a shared data-structure. RLL_Locks are manipulated with the LOCK and UNLOCK RLL box instructions.

type RLL_Lock : sint32;

The format of an RLL_Lock is as follows:

2^{31} 2^{24}	2^{23} 2^{16}	2^{15} 2^{8}	2^{7} 2^{0}		
xxxxxxxx	....	....	....	RLMUTEX	Mutex used to indicate that the lock is held or is being manipulated. (80_{16} = held or being manipulated). The mutex must be acquired using an indivisible read-modify-write instruction.
....	x0000000	....	....	RLREQEX	Flag indicating that someone is requesting EXCLUSIVE control of the lock. (1 = requesting EXCLUSIVE). This bit is used to prevent applications requesting the lock SHARED from starving applications requesting the lock EXCLUSIVE.
....	....	xxxxxxxx	....	RLHELDCN	Number of lock holders.
....	....	....	xxxxxxxx	RLHOLDER	App ID of the application holding the lock EXCLUSIVE.

An RLL Lock may be in one of three states as follows:

FREE	In this state, the lock may be obtained by anyone. All fields contain 0.
HELD SHARED	In this state, applications requesting the lock EXCLUSIVE will be blocked but other applications requesting the lock SHARED (up to a maximum of 255) will be allowed to proceed (if RLREQEX is not set). In this state, field RLHOLDER = 00_{16} and field RLHELDCN contains a value in the range [1 ... 255] indicating the number of holders.
HELD EXCLUSIVE	In this state, any requests for the lock are blocked. Field RLMUTEX is set, field RLHOLDER contains the application ID of the application holding the lock, and field RLHELDCN = 1.

9.32 Variable SBRDNSBR : INDEX Of Next Entry In SBRDTBL.

Indicates the index of the next entry in the Slave Board Table. Variable SBRDNSBR resides in the Global System Data Area (GSDA).

SBRDNSBR : uint16;

Variable SBRDNSBR should be incremented by the master board after it has completed logging it's slave board into the system.

9.33 Variable SBRDTBL : VME_ptr To The Slave Board Table

Points to the Slave Board Table. Variable SBRDTBL resides in the Global System Data Area (GSDA).

SBRDTBL : VME_ptr to SBRDTBLE[0..15];

9.34 Record SBRDTBLE : Slave Board Table Entry.

Describes an entry in the Slave Board Table (see SBRDTBL).

```
                record SBRDTBLE
0000                STELAR       : LAR;
0001                STEBID       : BOARD_TYPE;
0002                STESDT       : uint16;
0004                STEBRDSP     : VME_ptr to entry in BRDSPTBL;
0008                STESDESC     : char[1..40]
0030₁₆          endrecord;
```

where

STELAR	Board's LAR. FF_{16} indicates that the board is not present.
STEBID	Board's type(see BOARD_TYPE for possible values).
STESDT	Index of the board's entry in the System Descriptor Table (see variable SDT).
STEBRDSP	VME pointer to an entry in the Board Specific Table (BRDSPTBL). For SIMATIC 575 this table is called the System I/O configuration table (SYSIOCTB) and resides in the Global System Data Area.
STESDESC	ASCII string (zero terminated) describing the Slave board. SIMATIC VME Slave board example "IO 16Xs 16Ys".

9.35 Variable SBRDTMUX : Slave Board Table Mutex.

Used to control access to the Slave Board Table (SBRDTBL). Variable SBRDTMUX resides in the Global System Data Area (GSDA).

```
SBRDTMUX : mutex;
```

A master board must obtain SBRDTMUX before it can log its slave boards into the Slave board table.

9.36 Variable SDT : VME_ptr To The System Descriptor Table.

Points to the System Descriptor Table. The SDT contains a list of the boards and the range of VME addresses assigned to the board. Variable SDT resides in the Global System Data Area (GSDA).

```
SDT : VME_ptr to SDTENT[0..31];
```

9.37 Record SDTENT : System Descriptor Table Entry.

Describes an entry in the System Descriptor Table (SDT).

```
            record SDTENT
0000            filler          : uint16; // reserved, must be zero.
0002            SELAR           : LAR;
0003            SEBRDID         : BOARD_TYPE;
0004            SESMEMS         : VME_ptr;
0008            SESMEME         : VME_ptr;
000C        endrecord.
```

where

SELAR	Board LAR.
SEBRDID	Board ID. Used to identify the type of board(see BOARD_TYPE).
SESMEMS	Address of start of shared memory.
SESMEME	Address of end of shared memory.

9.38 Variable SDTNMLAR : Next VME Master LAR.

Indicates the LAR of the next available VME master. SDTNMLAR is used in the board login process during power-up/reset for boards using the IRQ6 IACK daisy-chain method of logging in. Variable SDTNMLAR resides in the Global System Data Area (GSDA).

```
SDTNMLAR : LAR;
```

9.39 Variable SDTNSDT : Index Of Next Entry In The SDT.

Indicates the index of the next entry in the System Descriptor Table. Variable SDTNSDT resides in the Global System Data Area (GSDA).

```
SDTNSDT : uint16;
```

Variable SDTNSDT should be incremented by each board after it has completed logging itself into the system.

9.40 Variable SDTNSLAR : Next VME Slave LAR.

Indicates the LAR of the next available VME slave. SDTNSLAR is used in the board login process during power-up/reset for boards using the IRQ6 IACK daisy-chain method of logging in. Variable SDTNSLAR resides in the Global System Data Area (GSDA).

```
SDTNSLAR : LAR;
```

9.41 Record SHEAPHDR : Header For a Block Of Memory In System Heap.

Defines the header prepended to blocks of memory allocated from System Heap by message $001B_{16}$, Allocate Memory From System Heap.

```
          record SHEAPHDR
0000           SHHLAR     : LAR;
0001           SHHAPP     : APP_ID;
0002           filler     : uint16;
0004           SHHNEXT    : VME_ptr to SHEAPHDR;
0008           SHHSIZE    : uint32
000C      endrecord.
```

where

SHHLAR	LAR of the board to which the block of memory is allocated.
SHHAPP	App ID of the application to which the block of memory is allocated. 00_{16} indicates that the block is allocated to the board rather than to an application.
SHHNEXT	VME_ptr to the next block of allocated memory.
SHHSIZE	Size of the block of memory, including the System Heap Header.

9.42 Type SHORT_VME_ptr : Pointer Into Short VME Space (A16 space).

Specifies a 32-bit pointer into short VME address space (A16). A Short_VME_ptr is an unsigned 32-bit integer (uint32), of which only the least-significant 16-bits are meaningful. It contains the byte offset of an object in VME A16 (short) address space.

9.43 Type sint8 : 8-bit Signed Integer.

Specifies an 8-bit two's-complement signed integer in the range [-128 ... 127]. The bit significance (for a positive number) is as follows:

S	2^6	2^5	2^4	2^3	2^2	2^1	2^0

9.44 Type sint16 : 16-bit Signed Integer.

Specifies a 16-bit two's-complement signed integer in the range [-32768 ... 32767]. The order of bytes and bit significance (for a positive number) is as follows:

0000	S	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8
0001	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

9.45 Type sint32 : 32-bit Signed Integer.

Specifies a 32-bit two's-complement signed integer in the range [-2147483648 ... 2147483647]. The order of bytes and bit significance (for a positive number) is as follows:

0000	S	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}	2^{25}	2^{24}
0001	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}
0002	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8
0003	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

9.46 Variable SYSCONFG : Primary 575 Configuration Options

Variable SYSCONFG contains the configuration options selected for the Primary 575. This variable should be treated as a READ ONLY variable by all other masters. Primary 575 configuration options are contained in an 8-bit byte. The bits of this byte are defined as follows

bits 2^7 - 2^5	Reserved, always 000_2.
bits 2^4 - 2^3	Base Address[1]

2^4	2^3	A24 Base Address (GSDA)	A16 Base Address ($GCSR_0$)
0	0	000000_{16}	0000_{16}
0	1	400000_{16}	4000_{16}
1	0	800000_{16}	8000_{16}
1	1	$C00000_{16}$	$C000_{16}$

bit 2^2	0 : Primary 575 will assert SYSRESET* during FAULT recovery.[2] 1 : Primary 575 will not assert SYSRESET* during FAULT recovery.
bit 2^1	0 : Auto-Configured. 1 : User-Configured.
bit 2^0	Reserved, always 0.

Note 1 - Default value is 00_2 when Auto-Configured is selected.
Note 2 - Default value when Auto-Configured is selected.

9.47 Record SYSIOCEN : System I/O Configuration Table Entry

Describes an entry in the System I/O Configuration Table (SYSIOCTB). Variable SYSIOCTB resides in the Global System Data Area.

```
          record SYSIOCEN
0000          SIOC_OWN      : uint8;
0001          SIOC_CC       : uint8;
0002          SIOC_SDT      : uint16;
0004      endrecord.
```

where

SIOC_OWN	Owner of the board. SIOC_OWN shall reference the first application that configures an output point for the entry, whether or not the board referenced by the entry contains outputs. When SIOC_OWN is non-zero an attempt to configure an output for the entry is invalid.

0	No owner.
1 .. 26	Application ID of owner.
27 .. 255	Invalid.

SIOC_CC	Indicates the number of applications that have configurations for this board.
SIOC_SDT	Index to SDT entry corresponding to this entry. Zero indicates no board installed.

9.48 Variable SYSIOCMU : System I/O Configuration Table Mutex.

Used to control access to the System I/O Configuration Table (SYSIOCTB). Variable SBRDTMUX resides in the Global System Data Area (GSDA).

```
SBRDTMUX : mutex;
```

This Mutex must be held in order to update the System I/O Configuration Table.

9.49 Variable SYSIOCNE : Index Of Next Entry In SYSIOCTB.

Indicates the index of the next entry in the System I/O Configuration Table. Variable SYSIOCNE resides in the Global System Data Area (GSDA).

```
SYSIOCNE : uint16;
```

Variable SYSIOCNE should be incremented by the master board after each I/O configuration command.

9.50 Variable SYSIOCTB : VME_ptr To The SYSIOCEN.

Points to the System I/O Configuration Table. Variable SYSIOCTB resides in the Global System Data Area (GSDA).

```
SYSIOCTB : VME_ptr to SYSIOCEN[0..31];
```

9.51 Variable SYSTOD : System Time-Of-Day.

Contains the system time-of-day. SYSTOD is located in the GSDA. It is maintained by the System Controller.

```
SYSTOD : Time_of_Day;
```

9.52 Variable SYSTODMU : System Time-Of-Day Access Mutex.

Used to control access to the system time-of-day (SYSTOD). SYSTODMU must be obtained by a board before reading the time-of-day.

```
SYSTODMU : mutex;
```

9.53 Record TIME_OF_DAY : Time Of Day.

Describes the time-of-day.

```
             record Time_of_Day
0000             Year            : BCD[2] range [00 .. 99];
0001             Month           : BCD[2] values { 01 = Jan, ... 12 = Dec };
0002             Day             : BCD[2] range [01 ... 31];
0003             Hours           : BCD[2] range [01 ... 23];
0004             Minute          : BCD[2] range [00 ... 59];
0005             Second          : BCD[2] range [00 ... 59];
0006             .01 seconds     : BCD[2] range [00 ... 99];  // resolution is .10 second on 575
0007             Day_of_Week     : BCD[2] values { 01 = Sun, ..., 07 = Sat }
0008         endrecord;
```

9.54 Type uint8 : 8-bit Unsigned Integer.

Specifies an 8-bit unsigned integer in the range [0 ... 255].

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

9.55 Type uint16 : 16-bit Unsigned Integer.

Specifies a 16-bit unsigned integer in the range [0 ... 65535].

0000	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8
0001	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

9.56 Type uint32 : 32-bit Unsigned Integer.

Specifies a 32-bit unsigned integer in the range [0 ... 4294967295].

0000	2^{31}	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}	2^{25}	2^{24}
0001	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}
0002	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8
0003	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

9.57 Type VME_ptr : Pointer To An Object In Globally Accessible Memory.

Specifies a pointer to an object in standard VME address space (A24). A VME pointer is an unsigned 32-bit integer (uint32), of which only the least-significant 24-bits are meaningful. The VME_ptr contains the byte offset of an object in A24 (standard) VME address space. A value of 0 is the nil pointer and indicates that the pointer doesn't point to anything.

10. GSDA FORMAT CHANGES

This section describes each change that has been made to the GSDA data structure.

575 Software Release	GSDAFMT[0] Major Version	GSDAFMT[1] Minor Version
1.0.0	0000_{16}	0000_{16}
2.0.0	0001_{16}	0000_{16}
3.1.0[1]	0001_{16}	0001_{16}
3.1.0[2]	8001_{16}	0001_{16}

Note 1: Configuration mode is Auto-Configured.
Note 2: Configuration mode is User-Configured.

10.1 GSDA Format Changes For SIMATIC 575 Release 1.0

This was the original release.

10.2 GSDA Format Changes For SIMATIC 575 Release 2.0

Offsets into GSDA area for user code are the same for both version 0 and version 1 of GSDAFMT. However; there have been changes that make version 1 of the GSDA data structure incompatible with the previous version. These changes are explained in this section.

Variable IAPPLMAP was added to the end of the GSDA area.

A filler of 256 bytes was added to the end of the GSDA area to reduce the risk of another major change to the GSDA area.

The Time_of_Day data structure was changed to conform to that of the SIMATIC 545. This change affects only the format of variable SYSTOD.

The Time_of_Day data structure format for GSDA version 0 was as follows:

```
        record Time_of_Day
0000        Year            : BCD[4];  // e.g., 1994
0002        Month           : BCD[2] values { 01 = Jan, ... 12 = Dec };
0003        Day             : BCD[2] range [01 ... 31];
0004        Hours           : BCD[2] range [01 ... 23];
0005        Minute          : BCD[2] range [00 ... 59]
0006        Second          : BCD[2] range [00 ... 59];
0007        Day_of_Week     : BCD[2] values { 01 = Sun, ..., 07 = Sat }
0008    endrecord;
```

The Time_of_Day data structure format for GSDA version 1 follows:

```
        record Time_of_Day
0000        Year            : BCD[2];  // e.g., 94
0001        Month           : BCD[2] values { 01 = Jan, ... 12 = Dec };
0002        Day             : BCD[2] range [01 ... 31];
0003        Hours           : BCD[2] range [01 ... 23];
0004        Minute          : BCD[2] range [00 ... 59]
0005        Second          : BCD[2] range [00 ... 59];
0006        .01 seconds     : BCD[2] range [00 ... 99];  // resolution is .10 seconds
0007        Day_of_Week     : BCD[2] values { 01 = Sun, ..., 07 = Sat }
0008    endrecord;
```

10.3 GSDA Format Changes For SIMATIC 575 Release 3.1.

If the 575 system is operating in **AUTO-CONFIGURED** mode, the GSDA data structure is compatible with version 1 of the GSDA. In this configuration GSDAFMT[0] still equals 1 and the minor version number is incremented to 1 (i.e., GSDAFMT[1] = 1).

In **USER-CONFIGURED** mode the GSDA data structure can reside on any 4 megabyte boundary. To prevent a CPU using version 1 of the GSDA (i.e., assumming the GSDA base address was at 0) from comming up, the most significant bit of the major version number was set (i.e., GSDAFMT[0] = 8001_{16}) indicating that the two data structures are no longer compatible.

The length of the GSDA area did not change in this release. However, two new variables were added to the reserved area. The first new variable is SYSCONFG. This variable is a READ ONLY byte indicating the configuration of the Primary 575 CPU. The second new variable is PFSAVAR which is an array of sixteen 32-bit longwords. This battery backed-up array is maintained over a power cycle and may be used by masters (LAR 00_{16} to $0F_{16}$, respectively) to save up to 32 bits of power-fail context in the global system table. A 575 CPU uses its entry in this table to save the power-fail value of the system time tick for verification during power-up. A second copy of this value is also saved in the 575's local RAM. By comparing these two values on a subsequent power-up, the 575 CPU can reliably detect a board swap-out while power was off.

11. IBC ERROR CODES

This section describes the error codes received in the messaging system. All values are given as hexidecimal numbers.

Error Number	Error Code Description
00	No error.
70	Invalid message type (IBMTYPE)
71	Unimplemented message type (IBMTYPE)
72	Unknown protocol specified in Brd Tbl entry.
73	Destination brd not logged in
74	Attempt to send IBC msg to MY_LAR
75	Attempt to get mutex failed
76	Attempt to release mutex failed (was released).
77	Attempt to allocate from free VME sys heap failed
78	Attempt to free unallocated VME sys heap memory
79	TC operation failed.
7A	Invalid port number.
7B	Attempt to assign port rejected.
7C	Insufficient request size. (Msg 1E)
7D	Insufficient dst/rsp buffer size for result.
7E	Unknown Interrupt Access Code specified in Brd Tbl
7F	IBC msg rcvd is valid for only the system cntrlr.
80	No such application.
81	Task Code Request Block Not Available
82	Task Code Buffer Not Available
83	Message sent to wrong board

Customer Response

We would like to know what you think about our user manuals so that we can serve you better. How would you rate the quality of our manuals?

	Excellent	Good	Fair	Poor
Accuracy	_______	_______	_______	_______
Organization	_______	_______	_______	_______
Clarity	_______	_______	_______	_______
Completeness	_______	_______	_______	_______
Graphics	_______	_______	_______	_______
Examples	_______	_______	_______	_______
Overall design	_______	_______	_______	_______
Size	_______	_______	_______	_______
Index	_______	_______	_______	_______

Would you be interested in giving us more detailed comments about our manuals?

☐ **Yes!** Please send me a questionnaire.

☐ **No.** Thanks anyway.

Your Name:	________________________________
Title:	________________________________
Telephone Number:	(_____)__________________________
Company Name:	________________________________
Company Address:	________________________________
	________________________________
	________________________________

Manual Name: SIMATIC 575 Interboard Communications Specification **Edition:** Second
Manual Assembly Number: 2589734–0004 **Date:** 6/95
Order Number: PPX:575–8103–2

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO.3 JOHNSON CITY, TN

POSTAGE WILL BE PAID BY ADDRESSEE

ATTN: Technical Communications M/S 519
SIEMENS INDUSTRIAL AUTOMATION INC.
3000 BILL GARLAND RD
P O BOX 1255
JOHNSON CITY TN 37605–1255

FOLD