封面格式

行政院及所屬各機關出國報告

（出國類別： 研究 ）

（ AMS-02 計畫高能粒子撞擊試驗及 JIM_CAN 設計確認研討國外公差報告 ）

服務機關：中山科學研究院

出國人職稱：聘用技監、少校技士

姓　　名：劉光新　、葉富銘

出國地區：德國

出國期間：90.11.01～90.11.10

報告日期：91.05.31

CSIPW-91E-E0003

國外公差報告

配合丁肇中院士所主持之反物質質譜儀計畫（Anti-matter spectrometer／Alpha Magnetic spectrometer； AMS）赴瑞士 CERN 及德國 GSI 公司執行 Beam-test "待選零件之抗輻射高能粒子撞擊試驗"，以獲得該等半導體元件，在太空中之可能反應，作為設計質譜儀之主資料蒐集計算機（Main Data Acquisition Computer ； MDC）的參考依據。另外在 JIM_CAN 設計確認研討方面，我們與 CERN/MIT Dr. Cai and Prof. Lebedev 共同制定 JIM_CAN protocols Version 1.3 使得 AMS02 CAN Bus protocol 確定，以利各單元 CAN 軟體撰寫。

由於此次出國在 JIM_CAN protocols 設計確認方面 CERN/MIT Dr. Cai and Prof. Lebedev 希望我們能以實際硬體來驗證無誤後再發表成 AMS-02 CAN BUS 通用 protocols，所以回國後即著手設計第三版 JIM_CAN 軟硬體，於 5 月中完成驗證。因此原因本報告無法提早完成敬請見諒。

中山科學研究院

國外公差心得報告

批示				
在頗有太空電子、零組件設計製造之能量，加強推廣 （副院長 宋大緯 0601 1116）				
公差年度	九十	所屬單位各級主管	政戰部	企劃處
單位	地面雷達組 電子系統所 雷射物理組 第五研究所		已完成資料審查	一、貴單位提報出國報告過於延後，應於規定期限內提出，請加強督導與管制作業。 二、本案請將資料上傳行政院研考會網站，並將報告裝訂五份後送本處技術推廣組三份彙辦，一份紙本及電子檔送交本處科技資料組留存，另紙本一份送計評會解析。
級職	技士 技監			
姓名	葉富銘 劉光新	電子系統研究所 地面雷達組組長 陳鐵鈺 0531 1130 電子系統研究所 所長 荊溪暠 0531 1440	保防 陳 中深 0605 1635 副組長 蔡昇禕 1637	企劃處 副處長 郭永聖 企劃處 技術推廣組組長 沈從正 正 蔣雅倫 副組長 邱兆川

企劃處 91.6.3 收文章

91 356

國外公差人員返國報告主官（管）審查意見表

AMS-02 計畫緣自八十九年四月間討論整體概略功能需求，並釐清工作目標與範圍，於九十年九月技術交流會議，雙方交換 MDC 研製進度檢討，並規劃本次 AMS-02 高能粒子撞擊試驗，以期對設計 MDC 之重要待選零件，了解其抗輻射效應。

雖然本次係 AMS-02 計畫第二次高能粒子撞擊試驗，有第一次經驗作為參考，加上其間多次與中央大學、中研院、CERN 溝通，使得測試系統規劃完善、符合需求，但參與同仁在時程緊湊、成敗壓力下，經常加班趕工，順利完成測試系統開發，而執行團隊平素營運良好更居功厥偉。本院代表、中央大學、中研院與 CERN 工作團隊在測試期間連繫協調週全，應變處置靈活有效，夙夜匪懈 24 小時輪班工作，圓滿協助蒐集測試數據完成，(1)本院自製之 JSBC 板及測試系統在高能粒子撞擊試驗過程中，其行為模式及問題解決方式皆如預期一般，完全、完美的被掌控，並證明 CPC700 與 PPC750 可用於太空。(2)PLL driver 經高能粒子撞擊試驗 雖然沒有 Latch-up 問題 但造成 JSBC CPU 多次 reset 已確定必須更換元件。更詳細資訊研析正由 CERN 判讀中,估計對相關抗輻射效益，具相當程度的參考性，助益 JMDC 研製中，風險降低與可靠度的提昇。(3)另外在 JIM_CAN 設計確認研討方面，我們與 CERN/MIT Dr. Cai and Prof. Lebedev 共同制定 JIM_CAN protocols Version 1.3 使得 AMS02 CAN Bus protocol 確定，以利各單元 CAN 軟體撰寫。

本次公差執行效益顯著，圓滿達成任務需求。基於未來我國太空科技發展需要，建議發展本院設計、製造、測試高性能太空組件之能量。

報　　告　　資　　料　　頁			
1.報告編號：	2.出國類別： 研究	3.完成日期： 91.05.31	4.總頁數： 50
5.報告名稱： AMS-02 計畫高能粒子撞擊試驗及 JIM_CAN 設計確定研討			
6.核准文號	人令文號	(九０)銓鑑字第００七六七八號	
	部令文號		
7.經　　費		新台幣：325,395 元	
8.出(返)國日期		90.11.01 至 90.11.10	
9.公 差 地 點		德國、法蘭克福	
10.公 差 機 構		GSI	
11.附　　記			

附件二

系統識別號

行政院及所屬各機關出國報告提要

出國報告名稱：AMS02 計畫高能粒子撞擊試驗及 JIM_CAN 設計確認研討公差報告

頁數 50 含附件：■是□否

出國計畫主辦機關/聯絡人/電話
中山科學研究院/王中興 /電話 03-4712201#355616
出國人員姓名/服務機關/單位/職稱/電話
劉光新/中科院/光電所/聘用技監/03-4712201#359328
葉富銘/中科院/電子所/少校技士/03-4712201#355614
出國類別：□1 考察□2 進修■3 研究□4 實習□5 其他

出國期間：
90.11.01～90.11.10

出國地區：
德國

報告日期：91.05.31

分類號/目

關鍵詞：AMS，Beam-test，高能粒子撞擊試驗，CAN

內容摘要：(二百至三百字)

配合丁肇中院士所主持之反物質質譜儀計畫(Anti-matter spectrometer／Alpha Magnetic spectrometer； AMS) 赴德國 GSI 公司執行 Beam-test "待選零件之抗輻射高能粒子撞擊試驗"，以獲得該等半導體元件，在太空中之可能反應，作為設計質譜儀之主資料蒐集計算機(Main Data Acquisition Computer ；MDC) 的參考依據。另外在 JIM_CAN 設計確認研討方面，我們與 CERN/MIT Dr. Cai and Prof. Lebedev 共同制定 JIM_CAN protocols Version 1.3 使得 AMS02 CAN Bus protocol 確定，以利各單元 CAN 軟體撰寫。

附件三

行政院及所屬各機關出國報告審核表

出國報告名稱：AMS-02 計畫高能粒子撞擊試驗及 JIM_CAN 設計確認研討國外公差報告	
出國計畫主辦機關名稱：中山科學研究院	
出國人姓名/職稱/服務單位：劉光新/聘用技監/中科院第五研究所 葉富銘/少校技士/中科院電子系統研究所	
出國計畫主辦機關審核意見	☐1. 依限繳交出國報告 ☑2. 格式完整 ☑3. 內容充實完備 ☑4. 建議具參考價值 ☑5. 送本機關參考或研辦 ☑6. 送上級機關參考 ☐7. 退回補正，原因：☐①不符原核定出國計畫 ☐②以外文撰寫或僅以所蒐集外文資料為內容 ☐③內容空洞簡略 ☐④未依行政院所屬各機關出國報告規格辦理 ☐⑤未於資訊網登錄提要資料及傳送出國報告電子檔 ☑8. 其他處理意見：
層轉機關審核意見	☐同意主辦機關審核意見 ☐全部 ☐部分_____（填寫審核意見編號） ☐退回補正，原因：_____（填寫審核意見編號） ☐其他處理意見：

楊治清 1405

許德慶 1480

初水蓮 1411

說明：

一、出國計畫主辦機關即層轉機關時，不需填寫「層轉機關審核意見」。

二、各機關可依需要自行增列審核項目內容，出國報告審核完畢本表請自行保存。

三、審核作業應於出國報告提出後二個月內完成。

目錄

壹、 出國目的及緣由

配合丁肇中院士所主持之反物質質譜儀計畫（Anti-matter spectrometer／Alpha Magnetic spectrometer ；AMS）赴德國 GSI 公司執行 Beam-test "待選零件之抗輻射高能粒子撞擊試驗"，模擬太空環境，測試該等半導體元件在太空中之可能影響，以獲得該等半導體元件太空中之狀況及數據，作為設計該質譜儀中主資料蒐集計算機（Main Data Acquisition Computer；MDC）之參考。JIM_CAN 設計確認研討方面，我們與 CERN/MIT Dr. Cai and Prof. Lebedev 共同制定 JIM_CAN protocols Version 1.3 使得 AMS02 CAN Bus protocol 確定，以利各單元 CAN 軟體撰寫。

貳、 公差心得

一、在丁院士為 AMS02 計畫到中科院主持技術交流會議(Technical Exchange Meeting；TEM）後，即決定 20 種使用於 AMS02 計畫中之零件需進行粒子撞擊試驗（Beam-test），並選定在德國 GSI 公司進行（此公司具有高能粒子加速器），而本次中科院負責其中之 PowerPC 750，CPC700 兩種零件的測試、測試系統之規劃設計及建構等，在院內多位同仁積極努力下，終於在預定測試日期前完成。並於 90.10.27～90.11.10 出差赴瑞士 CERN、德國 GSI 公司執行選用元件之 Beam test。此次任務時間長且 24 小時輪班甚費精神和體力所以分兩梯次接班完成。第一梯次 90.10.27～90.11.05 由邵明義和張和銘負責安裝測試，第二梯次 90.11.01～90.11.10 由劉光新和葉富銘繼續測試及與 CERN 相關人員研討確認 JIM_CAN 協定設計。

二、此次執行 Beam-test 公差，順利達成公差任務，個人有以下心得，供大家分享。

1. 出國前的準備工作：

- ❖ 各項細部工作在本組及中研院團隊摒棄成見及充分合作、日夜趕工之情況下，在最短時間內，完成測試計畫之規劃、設計、製作、測試、模擬、展示.....等準備工作。
- ❖ 在各級長官一再叮嚀下，在院內已將待測之 JSBC 板預備三片，各種接線、界面裝備預備雙份，且將所需之軟體、文件分別儲存、安裝於手提電腦及燒錄於 CD-ROM 中，並分別由本院及中研院人員分批攜帶至瑞士。

- ❖ 在院內時，本院及中研院人員均相互學習及模擬操作，相互backup，除應有的文件外，並且將接頭、訊號線名稱；接點位置等明確標示於線材與設備上，務求完全排除現場人為可能產生的錯誤。
- ❖ 準備相關 JIM_CAN 協定設計文件。

2. 國外安裝測試：
   - ❖ 在經過長途的飛行，與瑞士 CERN 人員會合後，唯恐在德國 GSI 執行 Beam-test 時，發生待測之 JSBC 板或測試系統出現致命問題無法解決，在 CERN 時，本院及中央大學、中研院人員，確實地將本院與 CERN 的裝備、安裝測試多遍，以減少在 GSI 時，連上 BEAM 控制設備時，可能發生的問題，並確保了 Beam-test 順利完成。
   - ❖ 在德國 GSI 執行 Beam-test 測試期間，人員必須在 cave A 內 24 小時待命，但在本院及中央大學、中研院人員分工合作，輪流操作、控制、監看測試狀況下，以最有效率之團隊順利完成任務。
3. 準備週詳，測試資訊收集完整：

   此次測試由於事前測試系統（如附件一）規劃詳盡，測試演練完備，軟硬體搭配完美，所以能夠在執行 beam test 時，不出差錯的收集到非常多的有效資訊，並使用於分析 PPC750 及 CPC700 在 latch-up 及 upset 的行為模式上（如附件二）。
4. 充分討論：

   與 CERN/MIT Dr. Cai and Prof. Lebedev 共同制定 JIM_CAN protocols Version 1.3 使得 AMS02 CAN Bus protocol 確定，以利各單元 CAN 軟體撰寫。Dr. Cai and Prof. Lebedev 希望我們能實際設計完成軟硬體來驗證這協定(JIM_CAN protocols)再發表成全 AMS-02 CAN bus protocol（如附件三 Appendix A）。
5. 工安優先

   由於 cave A 是高能粒子輻射區，為確保進入人員的安全，所有進入人員必須有健康檢查之證明。在執行 Beam-test 測試之前，GSI 人員為此次參與測試者作詳盡之安全講習，提醒測試人員應注意事項，介紹各種安全裝置及措施外，更要求每位人員實地操作進出 cave A 的正確程序。Cave A 的安全管制非常完善，進入 cave A 時，以"輻射量計數器"為開門之鑰，進入時此鑰"歸零"並紀錄使用者身份，出來時用以確認身份並馬上讀取輻射計

量，隨時與安全單位連線，以確保工作人員之安全無虞。另外，每位測試人員隨身攜帶配戴之輻射偵測識別證，記錄測試人員受輻射照射總量，以確保安全。

參、 效益分析

本次公差所執行之高能粒子撞擊試驗及設計確認研討，是為了解所選用之半導體元件，在外太空中受輻射粒子撞擊時之可能反應，以求其因應及解決方法。本次公差之直接效益為“了解所選用之元件，在太空中可能之失效模式”，結論如下(1)本院自製之 JSBC 板及測試系統在高能粒子撞擊試驗過程中，其行為模式及問題解決方式皆如預期一般，完全、完美的被掌控，並證明 CPC700 與 PPC750 可用於太空。(2)PLL driver 經高能粒子撞擊試驗 雖然沒有 Latch-up 問題 但造成 JSBC CPU 多次 reset 已確定必須更換 PLL driver 元件。更詳細資訊研析正由 CERN 判讀中, 估計對相關抗輻射效益，具相當程度的參考性，助益 JMDC 研製中，風險降低與可靠度的提昇。(3)另外在 JIM_CAN 設計確認研討方面，我們與 CERN/MIT Dr. Cai and Prof. Lebedev 共同制定 JIM_CAN protocols Version 1.3 使得 AMS02 CAN Bus protocol 確定，以利各單元 CAN 軟體撰寫。(4)為中科院在丁肇中院士、中央大學張元翰教授及各級長官協助下，直接參與太空領域計畫之設計工作，跨出成功的一步。

本院目前專注於軍用武器之研發，所重視的是溫度及振動試驗等，但經由參與本次計畫後，得知高能粒子撞擊試驗之重要性，除了太空科技外，如放射性醫療器材、核電廠等，亦皆有輻射粒子問題的存在，所以這些都是未來本院在多角化、軍民通用方向上，可能可以著墨、貢獻的地方。

肆、 國外工作日程表

日　　期	工　　作　　內　　容
90.11.01(台灣)	出國，飛荷蘭阿姆斯特丹轉機德國
90.11.02(德國)	到高能粒子撞擊實驗室安全講習及進出實驗室實務演練，CAVE 為 control access
90.11.03(德國)	備便 Kr 粒子撞擊試驗 CPC700 及 PPC750 之 Kr 粒子撞擊試驗。
90.11.04(德國)	※ 檢視、討論、分析 CPC700 及 PPC750 在 100、200、400、800MeV，0 度、30 度、60 度 Kr 撞擊測試後，令人滿意之測試結果
90.11.05(德國)	備便 Kr 粒子撞擊試驗 ※ PLL driver 之 Kr 粒子撞擊試驗
90.11.06(德國)	※ 檢視、討論、分析 PLL driver 在 100、200、400、800MeV，0 度、30 度、60 度撞擊測試後 CPU 為何經常 reset。
89.11.07(德國)	※ 與 Dr. Cai and Prof. Lebedev 初步討論 register architecture。
89.11.08(德國)	※ 與 Dr. Cai and Prof. Lebedev 初步制定 JIM_CAN protocols Version 1.3
89.11.09(德國)	※返國　法蘭克福經阿姆斯特丹轉機。
89.11.10(台灣)	安抵國門

伍、 社交活動

日夜趕工待命，無社交活動。

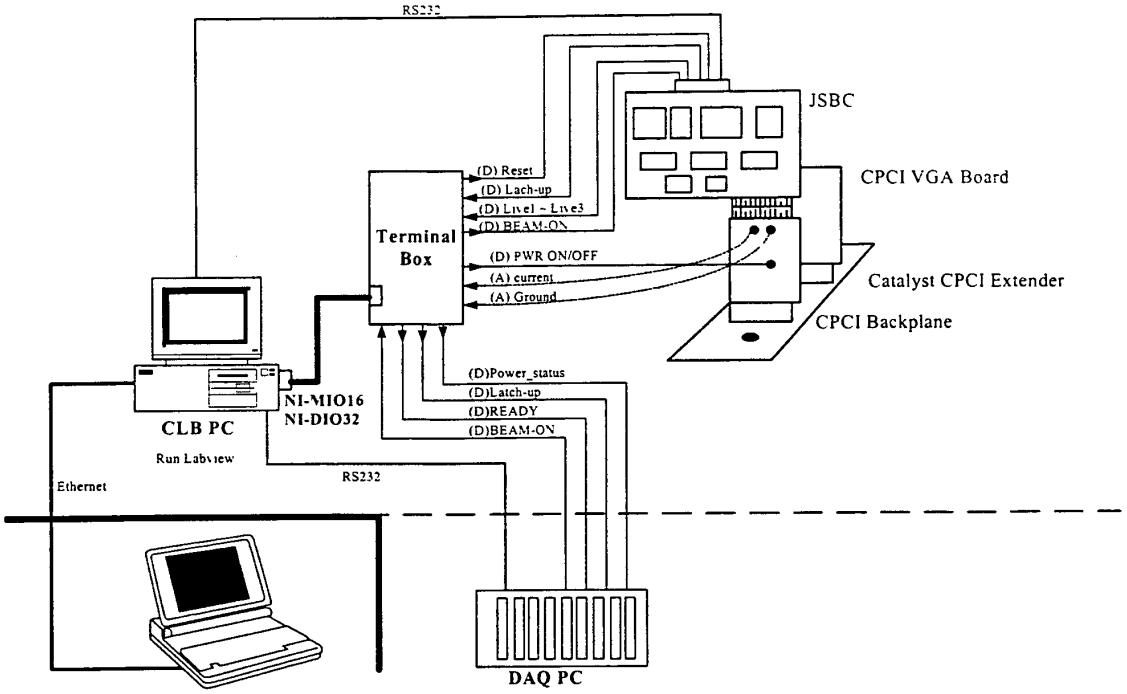
陸、 建議事項

太空科技的研究發展是我國正在進行的重點科技，使用於太空的電子元件必須要有抗輻射、耐高低溫、振動且能在真空中正常運作，因此用於太空的高性能電子元件的取得非常艱難且昂貴，所以才有將商規（工規、軍規）電子元件經過篩選後，使用於太空中的變通做法。

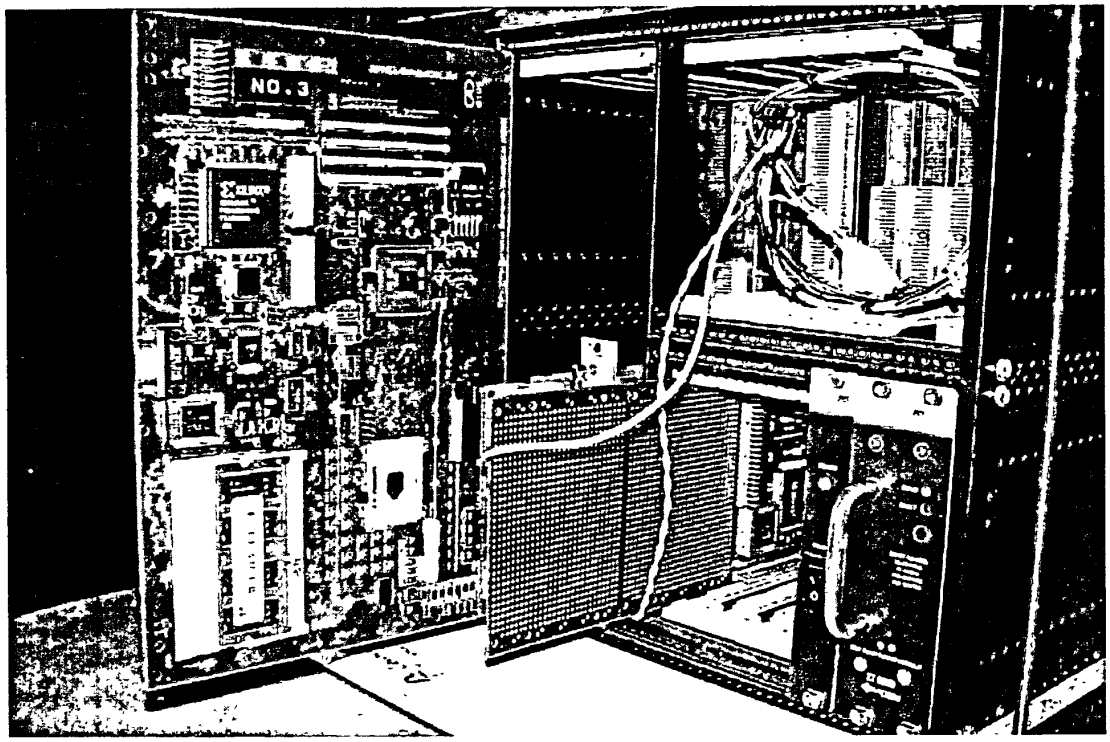
電子、電腦及半導體製造是目前我國最重要且世界一流的產業，要將我國製造各種電子元件使用於太空中，所需的測試項目中，溫度測試及振動測試，本院皆可以支援。真空狀態試驗則可在新竹完成，所欠缺的便是「高能輻射粒子撞擊試驗」，而此試驗所需之硬體設備，非我國目前能解決，故須遠赴德國。

鑑於符合太空規範的高性能電子零組件的缺乏，及太空科技研發是未來的必然趨勢，也是我國的重點科技，故建議本院應利用本次參與 AMS 計畫之便，發展本院設計、製造、測試高性能太空組件之能量。

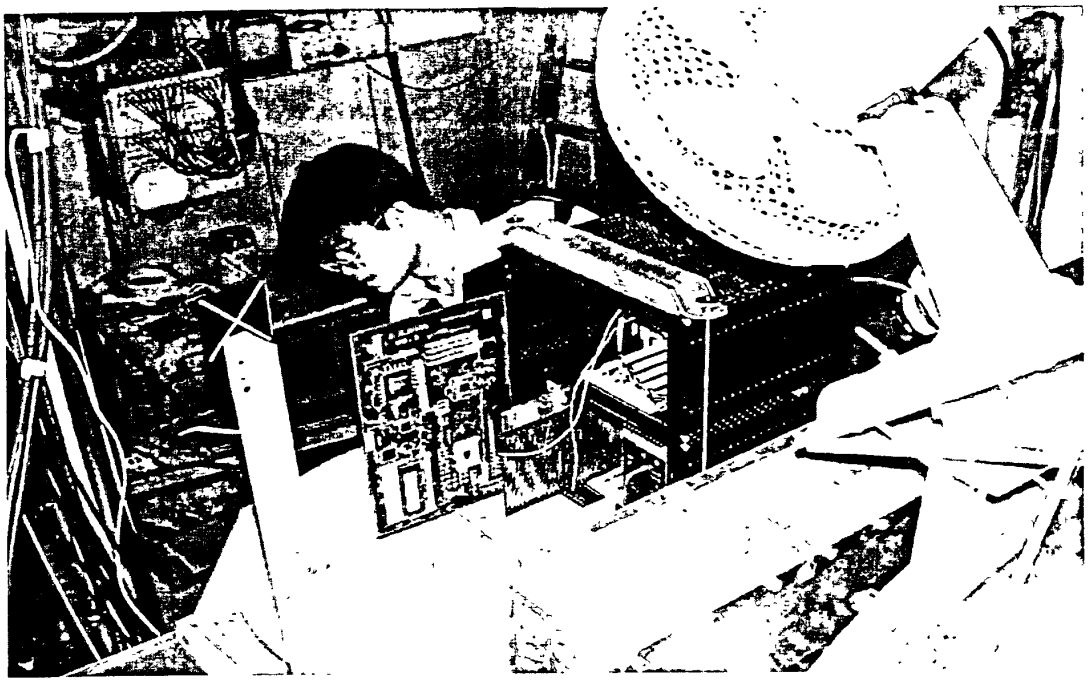
附件一：測試系統架構及安裝



PPC750 & CPC700 Beam Test Set-up



待測之 JSBC 板



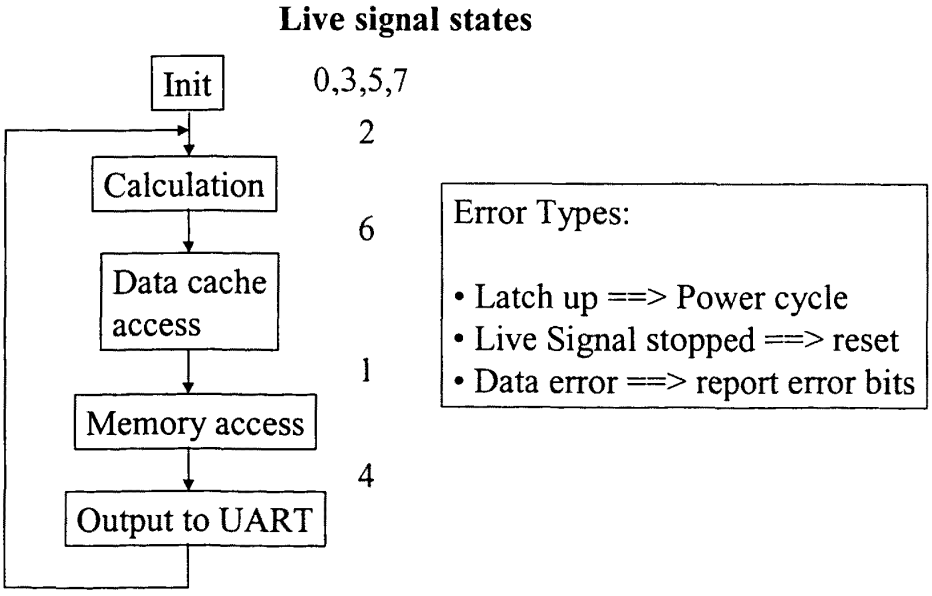
安裝 JSBC 板及機殼上架，連接纜線

附件二：

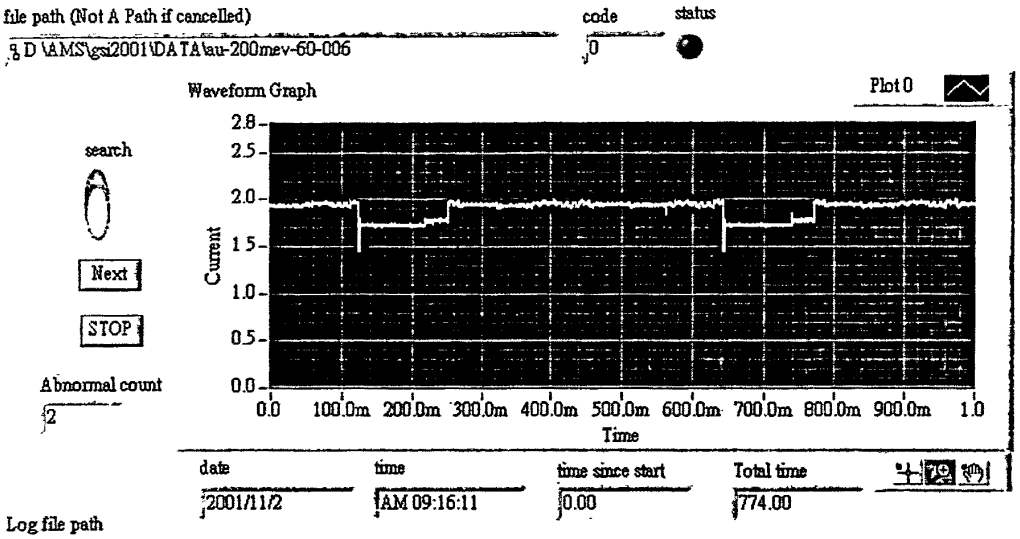
Preliminary results of PPC750 and CPC700 test results

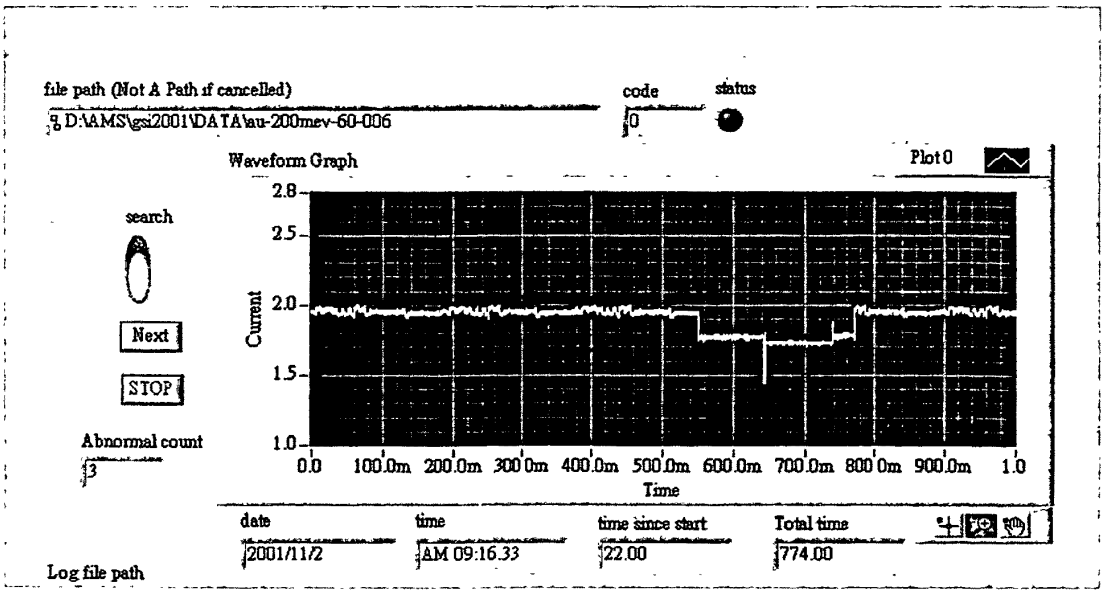
P.J. Hong
M.Y. Shaw
H.M. Chang
Y.H. Chang
C H Lin
C C Wong
L.S Hou

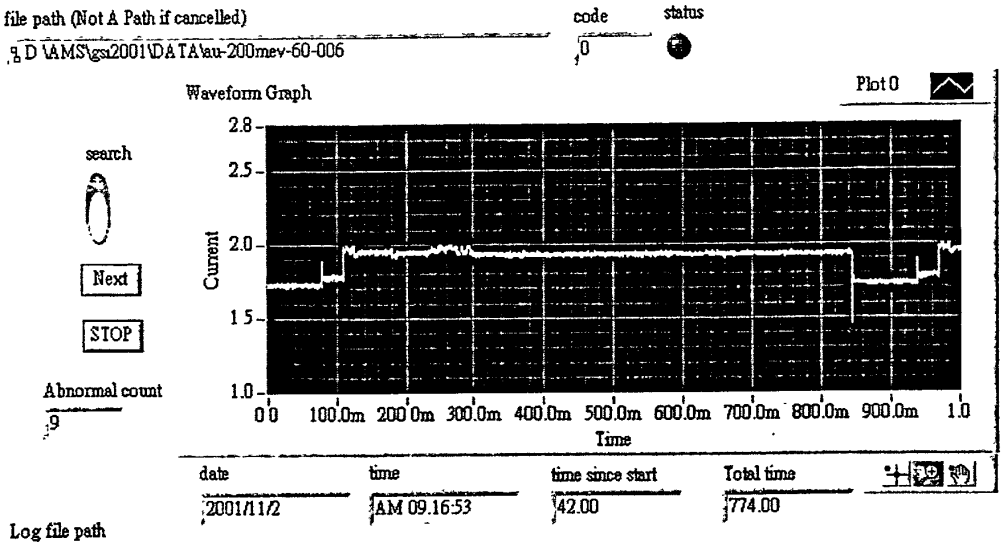
PPC750 test program



Current measurements



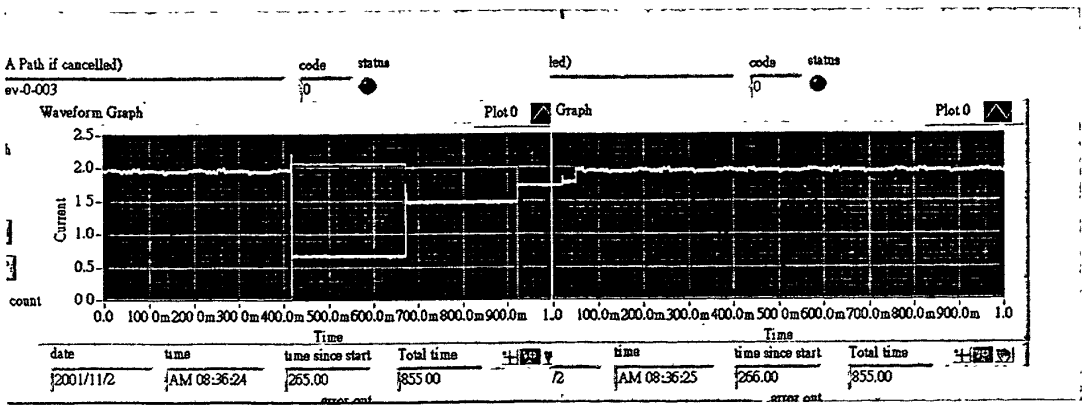


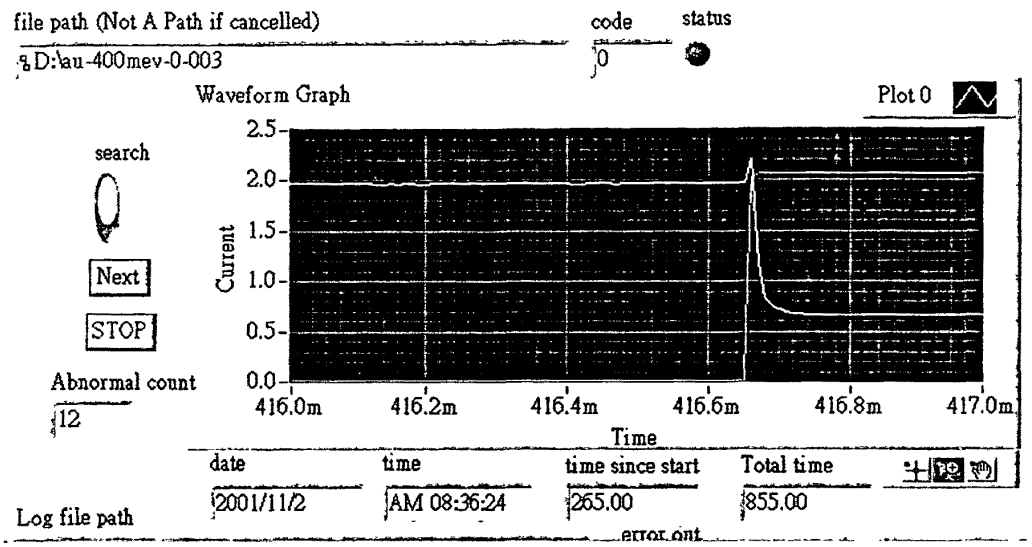


Latch ups

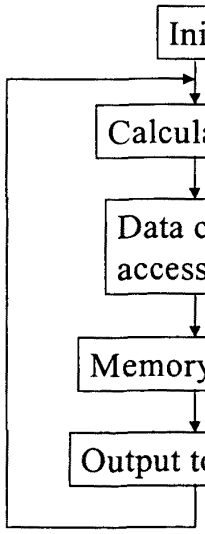
LET	11.	15.5	22.2	25.6	33.3	44.4
SEL(~1M particles/cm2)	0	1	2	1	3	1

No upset seen in Kr beam (LET 2.4 - 7.3)





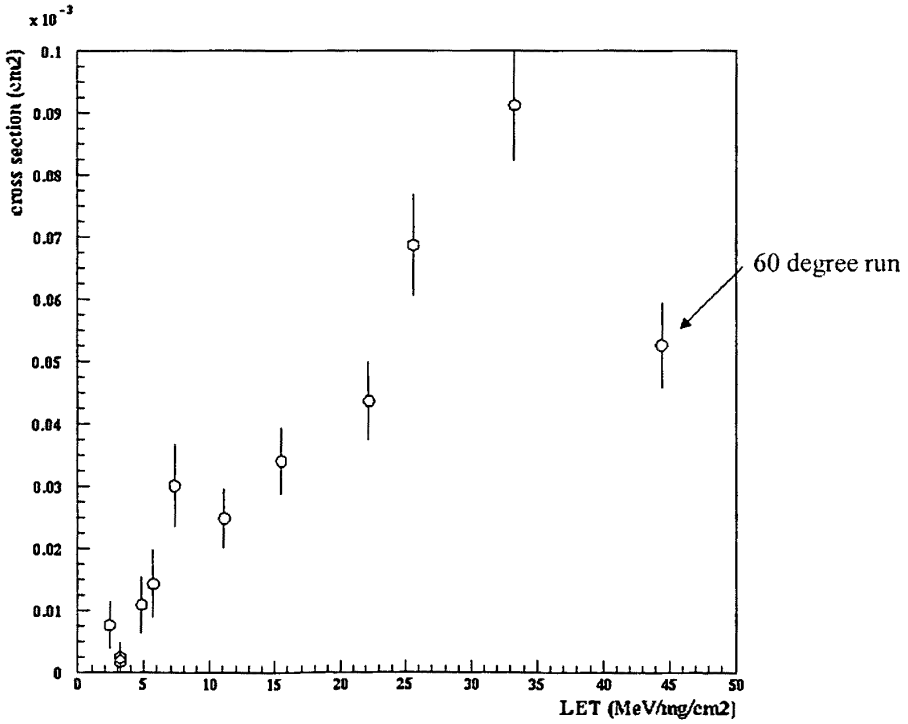
Number of PPC750 Upsets
(~1M particles/cm2)



	Au 800Mev	Au 100 MeV	Kr 100MeV	Total (12 runs)
Init	2	4	1	17
Calculation	9	35	8	140
Data cache access	6	28	5	97
Memory access	4	11	1	32
Output to UART	5	16	2	58
Calculation Error	2	8	4	50
				394

- The upsets occurs in all phases of the program execution, no obvious pattern observed.
- Only 1/8 of the upsets are recoverable. The rests require resetting.

PPC750 SEU Cross section (Beam profile and other corrections not applied yet)

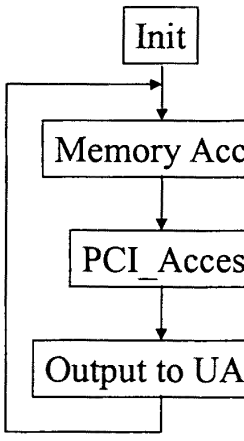


CPC700 Latchup

LET	11.	15.5	22.2	33.3
SEL(~1M particles/cm2)	1	0	0	1

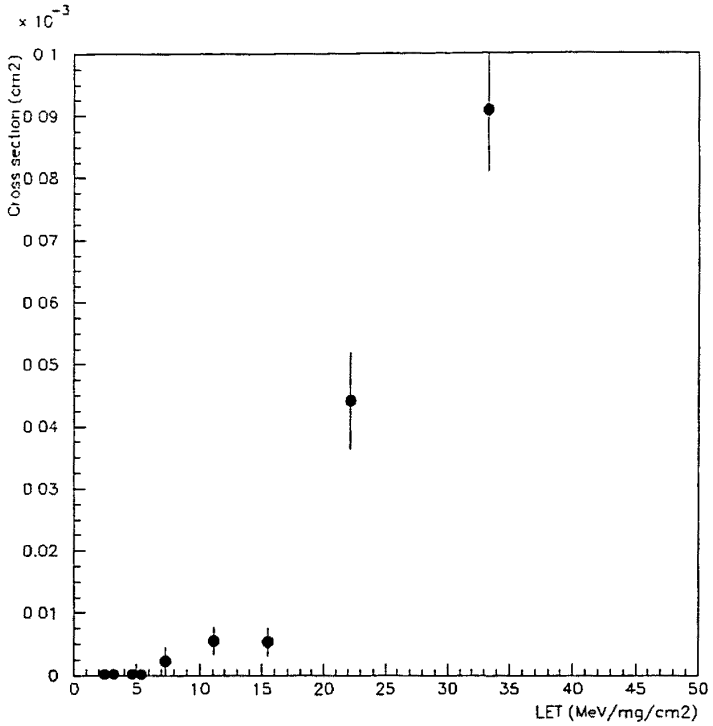
No upset seen in Kr beam (LET 2.4 - 7.3)

CPC700 Upset counts (~ 1M particles/cm2)



	Au 800Mev	Au 100 MeV	Kr 100MeV	Total (all data)
Init	0	6	0	10
Memory Access	1	23	0	36
PCI_Access	3	20	1	30
Output to UART	0	25	0	29
Data error	2	12	0	26
				131

CPC700 SEU Cross section (beam profile not corrected)



Preliminary conclusion:

- PPC750 and CPC700 are both quite radiation hard for our application. We don't expect many SEU due to PPC750 or CPC700 during the flight.
- MAXIM 972 PLL clock driver is latchup free.
- Upsets of MAXIM 972 cause PPC750 to malfunction, the mechanism is not clear yet. The upset cross section is quite large.

附件三 JIM_CAN Hardware and Protocols

TABLE OF CONTENTS

DOCUMENT CHANGE RECORD

No.	History	Date	Remark
01	DRAFT	June 29, 2001	
02	DRAFT	August 31,2001	
03	DRAFT	October 31,2001	
04	DRAFT	April 30,2002	

1 Overview

1.1 Feature

- 32 bit CompactPCI(CPCI) board compliant with PICMG 2.2 standard
- 32 bit memory mapped address
- Interrupt is provided via PCI bus
- Microcontroller Intel 8051 clocked 11.059 MHz
- Two Intel AN82527 controller clocked with 16 MHz
- 32K bytes Dual-Port-RAM(DPRAM)
- 128k bytes flash memory
- Two CAN bus interface according to ISO 11898 physical layer
- One RS232C interface on–board

1.2 Block Diagram

The block diagram of prototype version is shown in Figure 1.1. The detailed function description of each block is given in the following sections.

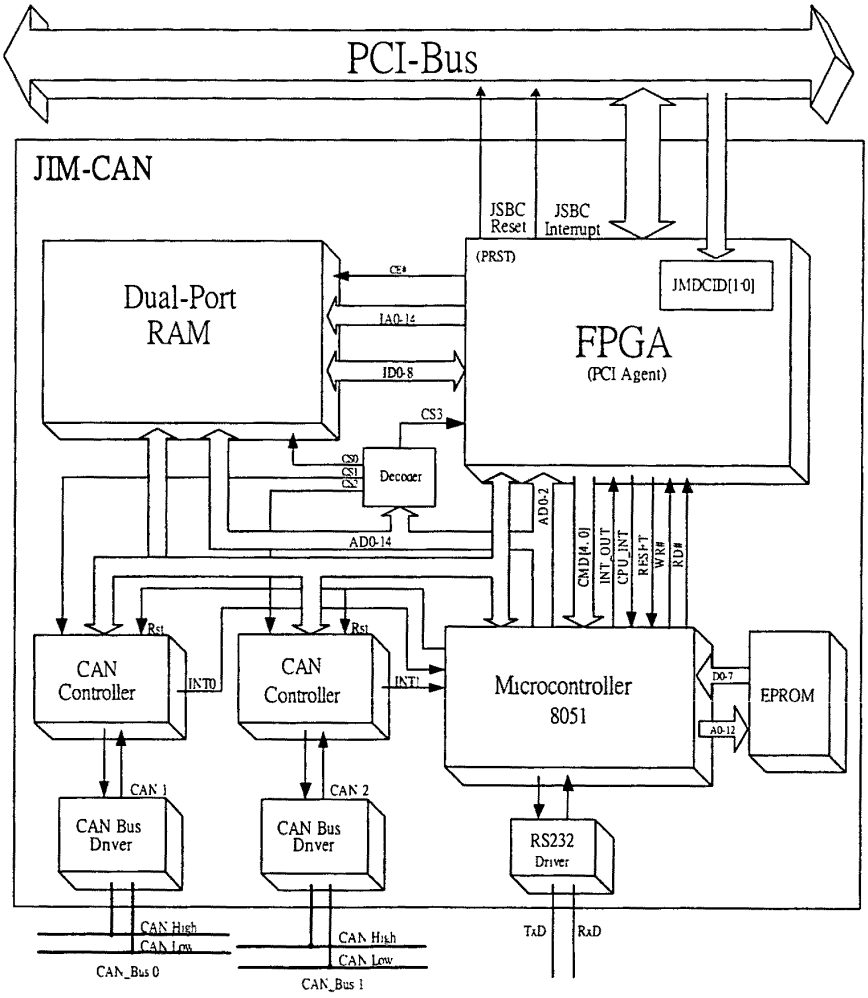


Figure 1.1 Prototype version of JIM-CAN Board

1.3 Dual-port RAM

In current prototype version, One IDT7007 (32K x 8) dual-port RAM is used as data buffer between JSBC and local *u*P (8051). The DPRAM is divided into several segments to hold data to/from different kind of master/slave node. By this mean, the loading of JSBC could be brought down effectively. The flight version will use the same dual-port memory as other JIM boards. It will be either IDT70V28 or CY7C028V (64k x 16).

1.4 PCI AGENT

The PCI agent implemented in an anti-fuse FPGA, Actel A54SX, is compliant with PCI 2.2 Specification. The configuration and PCI agent space of PCI agent is shown below:

PCI Configuration Space

Address	Register	Value
CFG–0x00	Vendor ID	0X414D ('AM')
CFG-0x02	Device ID	0X5305 ('S-5')
CFG-0x2C	Sub-Vendor ID	0X4A43 ('JC')
CFG-0x2E	Sub-System ID	0X414E ('AN')
CFG-0x0B	Base-Class Code	0X0C (Serial bus)
CFG-0x0A	Sub-Class Code	0X85
CFG-0x09	Prog. I/F	0X00
CFG-0x08	Rev. ID	0X22 (Rev. 2.2)
CFG-0x10	Base address register #0 of DPM access	By PnP BIOS
CFG-0x14	Base address register #1 of Register access	By PnP BIOS

PCI Agent Space

Address	Register
Dual port memory (DPM) BAR0+0x00000000 ~ BAR0+0x00007FFF	Dual-port RAM 32KB address space.
BAR1+0x00	Control Register (CR)
BAR1+0x04	Status Register (SR)
BAR1+0x08	Interrupt Register (IR)
BAR1+0x0C	Error Register (ER)

For detailed definition of control, status, interrupt and error registers, please refer to "Appendix A: JIM-CAN Protocols version 1.3".

1.5 Microcontroller

The adoption of 8051 here has several advantages. First, it can reduce the burden of JSBC. Second, when power-up, 8051 can read the Geographic ID (GID) from the board. Third, when 8051 decode a Boot command from CAN bus, it can issue a reset signal to notify JSBC that the reset process must be taken immediately.

1.6 CAN Bus Controller

The Intel AN82527 serial communications controller is a highly integrated device that performs serial communication according to the CAN protocol. The CAN protocol uses a multi-master (contention based) bus configuration for the transfer of 'communication objects' between nodes of the network. This multi-master bus is also referred to as CSMA/CR or Carrier Sense, Multiple Access, with Collision Resolution.

The 82527 provides storage for 15 message objects of 8-byte data length. Each message object can be configured as either transmit or receive except for the last message object.

1.7 CAN Bus Driver

The Philips PCA82C250 is the interface between the CAN protocol controller and the physical bus. The device provides differential transmit capability to the bus and differential receive capability to the CAN controller. It is fully compatible with the ISO 11898 standard with data rate up to 1 Mbps.

1.8 EPROM

In prototype version, one 128K X 8 bit flash memory is used to store the program of 8051. Flight version will use two radiation hardened 32K X 8 bit PROM.

1.9 RS232C

The JIM-CAN also implements a RS232C interface on the board. We can communicate with the monitor program through RS232 port to dialogue the hardware function. By this way, we can easily to check whether the JIM-CAN is in a good condition or not.

2. Memory Space

The PC communicates with the board via DPRAM. The memory space of DPRAM is divided into three sections. The first section is local RAM buffer for 8051. The second section is RX/TX buffer for different master/slave nodes. The last section is register area for handshakes for Host CPU and 8051.

2.1 Memory map seen from PCI

In prototype version, the memory space on the PC side is 32k bytes. The BAR0 and BAR1, are configuration registers, contain base address for accessing memory and registers. The BAR0 is used for memory access and is assigned by the CPCI PnP Bios. The BAR1 is used for registers access (Control, Status, Command/Interrupt and Error).

BAR0 Offset	**Usage**	**Real Size**
0x00000	8051 RAM Buffer	2 Kbytes
0x0800	JMDC_A TX Buffer (Request)	2 Kbytes
0x1000	JMDC_A RX Buffer (Request)	2 Kbytes
0x1800	JMDC_A TX Buffer (Reply)	2 Kbytes
0x2000	JMDC_A RX Buffer(Reply)	2 Kbytes
0x2800	JMDC_B TX Buffer (Request)	2 Kbytes
0x3000	JMDC_B RX Buffer (Request)	2 Kbytes
0x3800	JMDC_B TX Buffer (Reply)	2 Kbytes
0x4000	JMDC_B RX Buffer(Reply)	2 Kbytes
0x4800	JMDC_C TX Buffer (Request)	2 Kbytes
0x5000	JMDC_C RX Buffer (Request)	2 Kbytes
0x5800	JMDC_C TX Buffer (Reply)	2 Kbytes
0x6000	JMDC_C RX Buffer(Reply)	2 Kbytes
0x6800	USCM TX Buffer	2 Kbytes
0x7000	USCM RX Buffer	2 Kbytes
BAR1 Offset		
0x00	Control Register	4 bytes
0x04	Status Register	4 bytes
0x08	Command/Interrupt Register	4 bytes
0x0C	Error Register	4 bytes

Notes:

- 8051 RAM buffer should be accessed only from 8051.
- Control Register is Read/Write from PCI and Read Only from 8051 with auto clear.
- Status Register is Read only from PCI with auto clear and Read/Write from 8051.
- Error Register is Read only from PCI with auto clear and Read/Write from 8051.
- Host should check its own ID at initialization and use buffers for other three JMDCs according to ID order.

JMDC0	A=1	B=2	C=3
JMDC1	A=0	B=2	C=3
JMDC2	A=0	B=1	C=3
JMDC3	A=0	B=1	C=2

2.2 Memory map seen from 8051

Because there maybe at least one masters active in the same time, we must preserve fixed space for holding data to/from the can bus. The default size is 2k bytes each for TX/RX buffer.

Address	Usage	Real Size
0x0000	8051 RAM Buffer	2 Kbytes
0x0800	JMDC_A TX Buffer (Request)	2 Kbytes
0x1000	JMDC_A RX Buffer (Request)	2 Kbytes
0x1800	JMDC_A TX Buffer (Reply)	2 Kbytes
0x2000	JMDC_A RX Buffer(Reply)	2 Kbytes
0x2800	JMDC_B TX Buffer (Request)	2 Kbytes
0x3000	JMDC_B RX Buffer (Request)	2 Kbytes
0x3800	JMDC_B TX Buffer (Reply)	2 Kbytes
0x4000	JMDC_B RX Buffer(Reply)	2 Kbytes
0x4800	JMDC_C TX Buffer (Request)	2 Kbytes
0x5000	JMDC_C RX Buffer (Request)	2 Kbytes
0x5800	JMDC_C TX Buffer (Reply)	2 Kbytes
0x6000	JMDC_C RX Buffer(Reply)	2 Kbytes
0x6800	USCM TX Buffer	2 Kbytes
0x7000	USCM RX Buffer	2 Kbytes
0x8000	CAN Controller 1	256 bytes
0x8100	CAN Controller 2	256 bytes
0x8200	Status Register	4 bytes
0x8204	Control Register	4 bytes
0x8208	Command/Interrupt Register	4 bytes
0x820C	Error Register	4 bytes

Notes:

- 8051 RAM buffer should be accessed only from 8051.
- Control Register is Read/Write from 8051 and Read Only from PCI with auto clear.

- Status Register is Read only from 8051 with auto clear and Read/Write from PCI.
- Error Register is Read only from PCI with auto clear and Read/Write from 8051.
- 8051 should check its own ID at initialization and use buffers for other JMDCs according to the ID order.

2.3 CAN controller

Two Can controllers are installed on JIM-CAN. They are accessed with CS1 and CS2 signals The 1st CAN controller is mapped in the memory area from 0x8000h to 0x80FFh, the 2nd CAN controller is mapped in the memory area from 0x8100h to 0x81FFh of 8051. With an access to the memory area the corresponding CAN controller is automatically selected. The detailed register description can be found in the data sheet.
Both CAN controllers are clocked with 16 MHz. A low-level signal comes from 8051 Port 1 bit 1 resets the CAN controllers.

CAN Controller	Base Address	Interrupt at 8051	Hardware Reset
1st CAN Controller	0x8000h	Int0	P1.1
2nd CAN Controller	0x8100h	Int1	P1.1

3. Configurations

3.1 Jumper setting

The following figure shows a diagram of JIM-CAM with the user configurable jumpers.

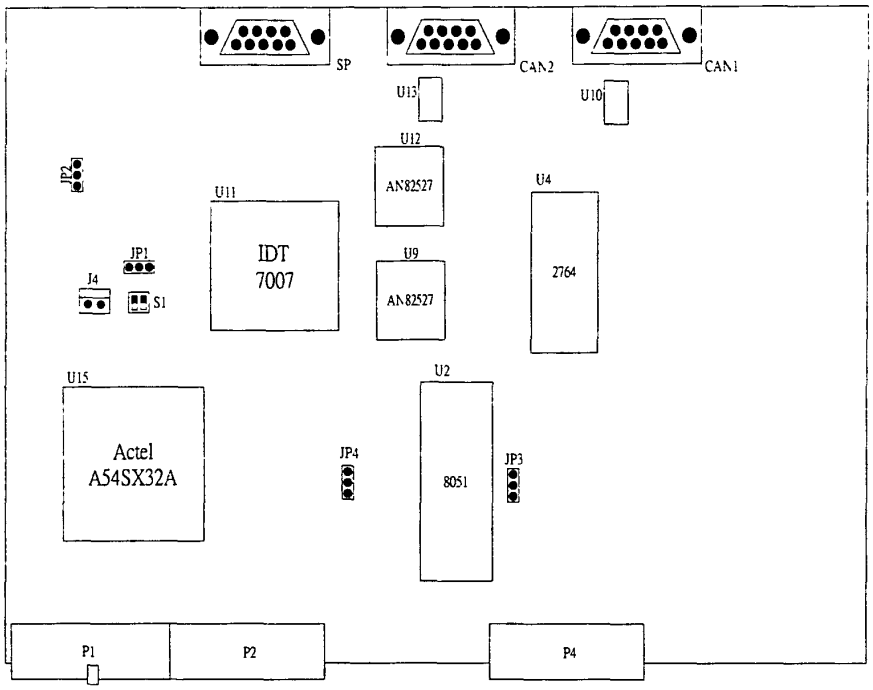


Figure 3.1 Jumpers and connectors

3.1.1 +3.3 V power supply

+3.3 V	JP1
On-Board	1,2
from CPCI	2,3

3.1.2 +2.5 V power supply

+2.5 V	JP2
On-Board	1,2
from CPCI	2,3

3.1.3 program source

Program	JP3
External	1,2
Internal	2,3

3.1.4 reset source

Reset	JP4
Reset Buttom	1,2
PCI Agent	2,3

3.1.5 +5 V power supply

+ 5 V	J4
External	ON

3.2 Pin assignment

The pin assignment is for prototype only. Flight version of connector and pin assignment will be defined later.

3.2.1. CAN bus and RS232C interface

CAN bus interface		RS232C interface	
Pin No.	Signal	Pin No.	Signal
1		1	
2	CAN High	2	TxD
3	GND	3	RxD
4		4	
5		5	GND
6	GND	6	
7	CAN Low	7	
8		8	
9		9	

4. Power Issue

In prototype version, there are two regulators used to convert +5V into +3.3V and + 2.5V. These two voltages are used for PCI agent only. With an external +5V power source, JIM-CAN can work alone without plug into CPCI enclosure.

In flight version, only one regulator will be adopted to provide + 2.5V for PCI agent and the +3.3V source will be supplied from CPCI bus.

The total power consumption is estimated as follow:

Voltage	**Current**	Total Power Consumption	# of DC/DC	Latch-up Protection
+5V +3.3V	0.35A (Typical) 0.42 A (Max) 0.40A (Typical) 0.48A (Max)	3.07 W (Typical) 3.684 W (Max)	+5V → +3.3V +5V → +2.5V	None

Appendix A: JIM-CAN Protocols Version 1.3

A.1 Memory Map Seen from PCI

BAR0 Offset	Usage	Real Size
0x00000	8051 RAM Buffer	2 Kbytes
0x0800	JMDC_A TX Buffer (Request)	2 Kbytes
0x1000	JMDC_A RX Buffer (Request)	2 Kbytes
0x1800	JMDC_A TX Buffer (Reply)	2 Kbytes
0x2000	JMDC_A RX Buffer(Reply)	2 Kbytes
0x2800	JMDC_B TX Buffer (Request)	2 Kbytes
0x3000	JMDC_B RX Buffer (Request)	2 Kbytes
0x3800	JMDC_B TX Buffer (Reply)	2 Kbytes
0x4000	JMDC_B RX Buffer(Reply)	2 Kbytes
0x4800	JMDC_C TX Buffer (Request)	2 Kbytes
0x5000	JMDC_C RX Buffer (Request)	2 Kbytes
0x5800	JMDC_C TX Buffer (Reply)	2 Kbytes
0x6000	JMDC_C RX Buffer(Reply)	2 Kbytes
0x6800	USCM TX Buffer	2 Kbytes
0x7000	USCM RX Buffer	2 Kbytes
BAR1 Offset		
0x00	Control Register	4 bytes
0x04	Status Register	4 bytes
0x08	Command/Interrupt Register	4 bytes
0x0C	Error Register	4 bytes

Notes:

- 8051 RAM buffer should be accessed only from 8051.
- Control Register is Read/Write from PCI and Read Only from 8051 with auto clear.
- Status Register is Read Only from PCI with auto clear and Read/Write from 8051.
- Error Register is Read Only from PCI with auto clear and Read/Write from 8051.
- Host should check its own ID at initialization and use buffers for other three JMDCs according to ID order.

JMDC0	**A=1**	**B=2**	**C=3**
JMDC1	**A=0**	**B=2**	**C=3**
JMDC2	**A=0**	**B=1**	**C=3**
JMDC3	**A=0**	**B=1**	**C=2**

A.2 Memory Map Seen from 8051

Address	Usage	Real Size
0x0000	8051 RAM Buffer	2 Kbytes
0x0800	JMDC_A TX Buffer (Request)	2 Kbytes
0x1000	JMDC_A RX Buffer (Request)	2 Kbytes
0x1800	JMDC_A TX Buffer (Reply)	2 Kbytes
0x2000	JMDC_A RX Buffer(Reply)	2 Kbytes
0x2800	JMDC_B TX Buffer (Request)	2 Kbytes
0x3000	JMDC_B RX Buffer (Request)	2 Kbytes
0x3800	JMDC_B TX Buffer (Reply)	2 Kbytes
0x4000	JMDC_B RX Buffer(Reply)	2 Kbytes
0x4800	JMDC_C TX Buffer (Request)	2 Kbytes
0x5000	JMDC_C RX Buffer (Request)	2 Kbytes
0x5800	JMDC_C TX Buffer (Reply)	2 Kbytes
0x6000	JMDC_C RX Buffer(Reply)	2 Kbytes
0x6800	USCM TX Buffer	2 Kbytes
0x7000	USCM RX Buffer	2 Kbytes
0x8000	CAN Controller 1	256 bytes
0x8100	CAN Controller 2	256 bytes
0x8200	Status Register	4 bytes
0x8204	Control Register	4 bytes
0x8208	Command/Interrupt Register	4 bytes
0x820C	Error Register	4 bytes

Notes:

- 8051 RAM buffer should be accessed only from 8051.
- Control Register is Read/Write from 8051 and Read Only from PCI with auto clear.
- Status Register is Read Only from 8051 with auto clear and Read/Write from PCI.
- Error Register is Read Only from PCI with auto clear and Read/Write from 8051.
- 8051 should check its own ID at initialization and use buffers for other JMDCs according to the ID order.

A.3 Registers Bit Map

A.3.1 Bit map for Control (seen from PCI)

Bit map for Status (seen from 8051)

Bits	Descriptions	Host	8051
0	Set New Data for JMDC_A TX (Request)	R/W	R&C
1	Set Read Done for JMDC_A RX (Request)	R/W	R&C
2	Set New Data for JMDC_A TX (Reply)	R/W	R&C
3	Set Read Done for JMDC_A RX (Reply)	R/W	R&C
4	Set New Data for JMDC_B TX (Request)	R/W	R&C
5	Set Read Done for JMDC_B RX (Request)	R/W	R&C
6	Set New Data for JMDC_B TX (Reply)	R/W	R&C
7	Set Read Done for JMDC_B RX (Reply)	R/W	R&C
8	Set New Data for JMDC_C TX (Request)	R/W	R&C
9	Set Read Done for JMDC_C RX (Request)	R/W	R&C
10	Set New Data for JMDC_C TX (Reply)	R/W	R&C
11	Set Read Done for JMDC_C RX (Reply)	R/W	R&C
12	Set New Data for USCM TX	R/W	R&C
13	Set Read Done for UCSM RX	R/W	R&C
14-15	Reserved	R/W	R&C
16-23	Reserved	R/W	R&C
24-31	Reserved	R/W	R&C

Notes:

- R/W : read or write.
- R&C : read and clear.
- R : read only.

A.3.2 Bit map for Status Register (seen from PCI)

Bit map for Control Register (seen from 8051)

Bits	Descriptions	HOST	8051
0-1	JMDC_A TX Status (Request)	R&C	R/W
2-3	JMDC_A RX Status (Request)	R&C	R/W
4-5	JMDC_A TX Status (Reply)	R&C	R/W
6-7	JMDC_A RX Status (Reply)	R&C	R/W
8-9	JMDC_B TX Status (Request)	R&C	R/W
10-11	JMDC_B RX Status (Request)	R&C	R/W
12-13	JMDC_B TX Status (Reply)	R&C	R/W
14-15	JMDC_B RX Status (Reply)	R&C	R/W
16-17	JMDC_C TX Status (Request)	R&C	R/W
18-19	JMDC_C RX Status (Request)	R&C	R/W
20-21	JMDC_C TX Status (Reply)	R&C	R/W
22-23	JMDC_C RX Status (Reply)	R&C	R/W
24-25	USCM TX Status	R&C	R/W
26-27	USCM RX Status	R&C	R/W
28-29	Reserved	R&C	R/W
30-31	Boot command	R&C	R/W

Bits	Boot command Descriptions
00	Reserved
01	Boot from JMDC_A
10	Boot from JMDC_B
11	Boot from JMDC_C

Bits	TX/RX Status Descriptions	
00	Reserved	Reserved
01	JMDC_X Ready	USCM Ready
10	JMDC_X Busy	USCM Busy
11	JMDC_X Error	USCM Error

Notes:

- Status code 00 = reserved
- Status code 01 = Tx done or Rx completed
- Status code 10 = Tx or Rx undergoing
- Status code 11 = Abnormal

A.3.3 Bit map for Command/Interrupt Register (seen from both sides)

Bits	Descriptions	Host	8051	Signal
0	Host_RST_8051	R/W	R&C	Pulse
1	Reserved	R	R/W	latch
2	Host_IRQ_8051	R/W	R&C	Pulse
3	En_Host_IRQ_8051	R	R/W	latch
4	Reserved	R/W	R&C	Pulse
5	Reserved	R	R/W	latch
6	Reserved	R/W	R&C	Pulse
7	Reserved	R	R/W	latch
8	8051_Interrupt_Host	R&C	R/W	Pulse
9	En_8051_Interrupt_Host	R/W	R	latch
10	8051_Ready_Interrupt_Host	R&C	R/W	Pulse
11	En_8051_Ready_Interrupt_Host	R/W	R	latch
12	8051_Error_Interrupt_Host	R&C	R/W	Pulse
13	En_8051_Error_Interrupt_Host	R/W	R	latch
14	Boot_Command_Interrupt_Host	R&C	R/W	Pulse
15	En_Boot_Command_Interrupt_Host	R/W	R	latch
16	Reserved	R&C	R/W	Pulse
17	Reserved	R/W	R	latch
18	Reserved	R&C	R/W	Pulse
19	Reserved	R/W	R	latch
20-23	Reserved	R/W	R	latch
24-29	Reserved for 8051 command	R	R/W	latch
30-31	JMDC ID (Hardware setting)	R	R	latch

A.3.4 Bit map for Error Register (seen from both sides)

Bits	Descriptions	Host	8051
0-1	Error for JMDC_A TX (Request)	R&C	R/W
2-3	Error for JMDC_A RX (Request)	R&C	R/W
4-5	Error for JMDC_A TX (Reply)	R&C	R/W
6-7	Error for JMDC_A RX (Reply)	R&C	R/W
8-9	Error for JMDC_B TX (Request)	R&C	R/W
10-11	Error for JMDC_B RX (Request)	R&C	R/W
12-13	Error for JMDC_B TX (Reply)	R&C	R/W
14-15	Error for JMDC_B RX (Reply)	R&C	R/W
16-17	Error for JMDC_C TX (Request)	R&C	R/W
18-19	Error for JMDC_C RX (Request)	R&C	R/W
20-21	Error for JMDC_C TX (Reply)	R&C	R/W
22-23	Error for JMDC_C RX (Reply)	R&C	R/W
24-25	Error for USCM TX	R&C	R/W
26-27	Error for USCM RX	R&C	R/W
28-29	Reserved	R&C	R/W

30-31	Reserved	R&C	R/W

Notes:

- Error code 00 = no error
- Error code 01 = Timeout or Controller error
- Error code 10 = Error received
- Error code 11 = Fail received

A.-4 Default Message Buffer Assignment

Message Buffers	**Assignment**	CAN ID mask		
		R/R	Source	Destination
1	JMDC_A TX (Request)	1	JMDC	JMDC_A
2	JMDC_A RX (Request)	1	JMDC_A	JMDC
3	JMDC_A TX (Reply)	0	JMDC	JMDC_A
4	JMDC_A RX (Reply)	0	JMDC_A	JMDC
5	JMDC_B TX (Request)	1	JMDC	JMDC_B
6	JMDC_B RX (Request)	1	JMDC_B	JMDC
7	JMDC_B TX (Reply)	0	JMDC	JMDC_B
8	JMDC_B RX (Reply)	0	JMDC_B	JMDC
9	JMDC_C TX (Request)	1	JMDC	JMDC_C
10	JMDC_C RX (Request)	1	JMDC_C	JMDC
11	JMDC_C TX (Reply)	0	JMDC	JMDC_C
12	JMDC_C RX (Reply)	0	JMDC_C	JMDC
13	USCM TX Buffer	1	JMDC	USCM
14	USCM RX Buffer	0	USCM	JMDC
15	Reserved	-	-	JMDC

A.5 << AMS CAN Block >> Format

A.5.1 definition of CAN ID

<< AMS CAN Block >> is transmitted as one or several <<AMS CAN packets>> also known as CAN Data Frames. The extended 29-bit ID's are used, their formats are shown below. Do not forget that the 7 most significant bits(ID28~ID22) must no be all equal to 1. (L. Lebedev, "Data Formats and Communication Protocols", Nov. 2001)

it	**D**	Mask	Description	**Values**
31	28	0	0	Not used, must be 0
30	27	0	FIRST/BC1	CAN Packet sequence flag or special character
29	26	0	LAST/BC2	
28	25	1	R/R	Request/Reply flag
27	24	0	R/W	Read/Write flag
26	23	1	BUS_REQ	CAN bus for request
25	22	0	BUS_REP	CAN bus for reply
24	21	1	NEW	"new born" ID flag
23	20	1	S7	Source:
22	19	1	S6	
21	18	1	S5	
20	17	1	S4	
19	16	1	S3	
18	15	1	S2	
17	14	1	S1	
16	13	1	S0	
15	12	1	D7	Destination:
14	11	1	D6	
13	10	1	D5	
12	9	1	D4	
11	8	1	D3	
10	7	1	D2	
9	6	1	D1	
8	5	1	D0	
7	4	0		Not used
6	3	0		
5	2	0		
4	1	0		
3	0	0		
2				
1				
0				

DLC	FIRST /BC1	LAST /BC2	#Req/Rep	#Rd/WR	Meanings	Header	Data
!=0	0	0	Request(0)	Read(0)	Request: read(I)	May be	May be
!=0	0	0	Request(0)	write(1)	Request: write(I)	May be	May be
!=0	0	0	Reply(1)	Read(0)	Reply: read(I)	No	Yes
!=0	0	0	Reply(1)	write(1)	Reply: write(I)	impossible	
!=0	0	1	Request(0)	Read(0)	Request: read(L)	May be	May be
!=0	0	1	Request(0)	write(1)	Request: write(L)	May be	May be
!=0	0	1	Reply(1)	Read(0)	Reply: read(L)	No	Yes
!=0	0	1	Reply(1)	write(1)	Reply: write(L)	impossible	
!=0	1	0	Request(0)	Read(0)	Request: read(F)	Yes	May be
!=0	1	0	Request(0)	write(1)	Request: write(F)	Yes	May be
!=0	1	0	Reply(1)	Read(0)	Reply: read(F)	No	Yes
!=0	1	0	Reply(1)	write(1)	Reply: write(F)	impossible	
!=0	1	1	Request(0)	Read(0)	Request: read(F&L)	Yes	May be
!=0	1	1	Request(0)	write(1)	Request: write(F&L)	Yes	May be
!=0	1	1	Reply(1)	Read(0)	Reply: read(F&L)	No	Yes
!=0	1	1	Reply(1)	write(1)	Reply: write(F&L)	impossible	
=0	0	0	Request(0)	Read(0)	Request : Next	No	No
=0	0	0	Request(0)	write(1)	Request : Next	No	No
=0	0	0	Reply(1)	Read(0)	Reply : Next	No	No
=0	0	0	Reply(1)	write(1)	Reply : Next	No	No
=0	0	1	Request(0)	Read(0)	Request : Error	impossible	
=0	0	1	Request(0)	write(1)	Request : Error	impossible	
=0	0	1	Reply(1)	Read(0)	Reply : Error	No	No
=0	0	1	Reply(1)	write(1)	Reply : Error	No	No
=0	1	0	Request(0)	Read(0)	Request :Abort	No	No
=0	1	0	Request(0)	write(1)	Request : Abort	No	No
=0	1	0	Reply(1)	Read(0)	Reply : Abort	No	No
=0	1	0	Reply(1)	write(1)	Reply : Abort	No	No
=0	1	1	Request(0)	Read(0)	Request : End	Not used	
=0	1	1	Request(0)	write(1)	Request : End	Not used	
=0	1	1	Reply(1)	Read(0)	Reply : End	No	No
=0	1	1	Reply(1)	write(1)	Reply : End	No	No

A.5.2 AMS CAN Block Header

AMS Data Block Primary Header (Block ID)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
B1	B0	A8	A7	A6	A5	A3	A3	A2	A1	A0	T4	T3	T2	T1	T0
Block Type		Node address									Data Type				

Mapping

29-bit Identifier

--		ID25	ID24	--	ID12	ID11	ID10	ID9	ID8	ID7	ID6	ID5	--
--		B1	B0	--	D7	D6	D5	D4	D3	D2	D1	D0	--
--		Block Type		--	Destination Address (8bits)								--

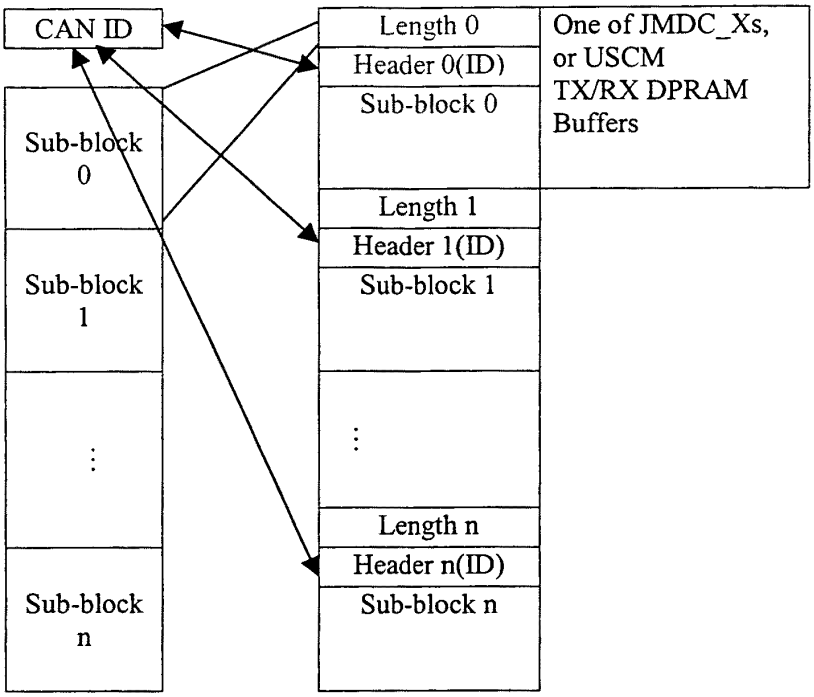
AMS CAN Block Header

7	6	5	4	3	2	1	0
B1	B0		T4	T3	T2	T1	T0
Block Type			Data Type				

A6 Protocol between Host and 8051

A.6.1 Block Structure

Data can be cut into sub-blocks since the limitation of buffer size. The format of sub-block is shown below.



A.6.2 Generic description

A. Procedure of host sending out one sub-block via TX(i) buffer

1. Host writes data length, sub-block header (CAN ID), and data into the corresponding TX(i) buffer.
2. Host writes a "set new data" bit of this TX (i) in Control register.
3. Host writes Host_IRQ_8051 in Command/Interrupt register. Set "1" to this bit and then hardware will generate a pulse automatically. This pulse will interrupt 8051.
4. After got an interrupt, 8051 reads Status register to check which TX(i) buffer got the "new data". This "set new data" bit will be automatically cleared by hardware after reading the Status register.
5. 8051 reads length, sub-block header (CAN ID), and data from the buffer of TX(i) and sets "Busy" bits in the Control register of TX(i).
6. If the length of sub-block of TX(i) exceeds one packet length of CAN (8Bytes), 8051 will cut sub-block into several packets and sends packets one after another. After transmitting each packet, 8051 waits for reply "next", "end", "error" or "fail".

7. When all of the sub-block data have been transmitted, 8051 sets "Ready" bits in Control register of this TX(i) buffer and sends an interrupt to host to inform host by setting 8051_Ready_Interrupt _Host bit in the Command/Interrupt register.
8. On errors (controller error, received "error" replay, or received "fail" reply), 8051 writes "error type" in Error register of TX(i) and "error" code in the Control register of TX(i). Then 8051 sends an interrupt to host by setting 8051_Error_Interrupt _Host bit.
9. After host got an interrupt, host reads Status register to checks "TX(i) status" and also reads "error type" bits from Error register if "error" bit is set.
10. Host prepares to send another data.

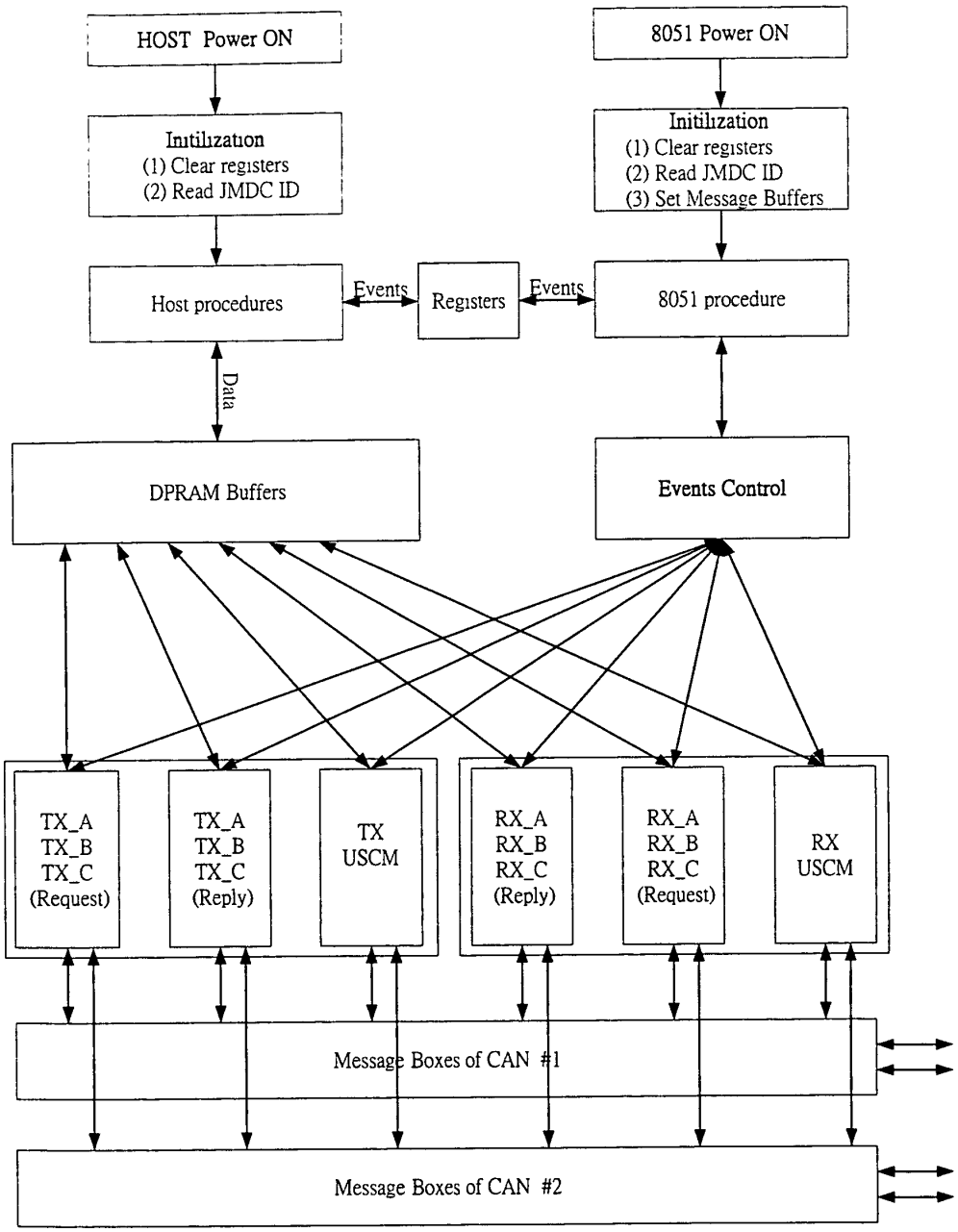
B. Procedure of host getting one sub-block via RX(j) buffer from 8051

1. When 8051 received one packet, it moves the packet to the corresponding RX(i) buffer. If it is the first packet, it should leave the first unit empty for length and save the CAN ID to the second location.
2. 8051 sets "Busy" bits of this RX(j) buffer in the Control register.
3. 8051 checks the CAN ID and data type of AMS CAN header. If it is a "boot" command, it goes to the special procedure for "boot" command. (?) (TBD) else goes to step 4.
4. After successfully receiving each packet, 8051 checks the value of CAN ID.DLC. If CAN_ID.DLC is not equal to zero, then 8051 will send "next" reply to source. Otherwise, the packet is recognized as the reply:token for acknowledge TX(j) transmitting data.
5. 8051 packs packets into sub-block of RX(i) buffer. Repeat 4 until the end of RX(j) buffer or the last packet.
6. 8051 should write the total length to the first location.
7. 8051 sets "Ready" bits in Control register of the RX(j) and sends an interrupt 8051_Ready_Interrupt _Host to host.
8. After got an interrupt, host checks which buffer got "new data".
9. Host reads length, sub-block header, and data from RX(j) buffer.
10. Host sets "read done" bit and Host_IRQ_8051 bit in Command/Interrupt register to interrupt 8051.
11. 8051 clears Status register bits of this RX(j) buffer. Repeat steps 1-11.
12. On error, 8051 writes "error code" to Error register and sets both "error status" bits and "error type" bits for this RX buffer in the Control register. Then 8051 sends an interrupt to host.
13. Host checks "error" bit. If "error" bit is set, host reads "error type" from Error register. Host will decide what to do.

C. Procedure of Receiving "Boot" Command

Host will decide what to do. (To Be Determined.)

A.6.3 Software flow (preliminary)



JIM_CAN software pseu-program is shown in the page 27-29. They include: **TX_Events()** .**RX_Events()**, **Do_packet_TX(i)**, **Received_B_C_RX(i)**, and **Received_F_L_RX(i)** procedures. These procedures are used to explain whole software flow we are going to implement the ASM02 JIM_CAN software in 8051. They are just preliminary design for handling the AMS02 CAN bus communications. Because the AMS02 CAN protocol allows for multiple masters access. It will be more complicated than AMS01 CAN protocol, only one master (MCC). In order to communicate with the other three bus masters(JMDCs) and several USCMs, an event control procedure is applied to deal with the CAN bus events and Host computer events by checking event registers (Control/Status/Interrupt/Error), the dedicated TX(i) buffer, RX(i) buffer and dedicated CAN message buffer. For more detail descriptions of TX(i)/RX(i) buffer map and control/ status/ interrupt register, .., etc., please refer to the section 1-5 of this document.

In the event control procedure, each event generated by Host computer or CAN controllers has the same priority. They share the CPU time resource by turns. Therefore a data structure is needed to store the job status of every TX(i) buffer event and RX(i) buffer event. The **Do_packet_TX(i)** procedure is applied to implement the data transmission of Request:Write, Reply:Read, Request:Read. Each TX(i) event has its own Finite State Machine but share the same transmission procedure. When call this procedure, just one CAN packet is transmitted and then will go to next State by next turn. The next state of TX(i) event may depend on the RX(i) Reply:token which is obtained by **Received_B_C_RX(i)**. The **Received_F_L_RX(i)** procedure is used to handle packet data collection for one sub-block of RX(i) buffer and auto reply "next", "error", "abort", and "end" when received ID.DLC !=0. The main goal of **Received_B_C_RX(i)** procedure is to recognize the meaning of replying message for changing TX(i) state when received ID.DLC ==0.

This proposal "JIM_CAN protocol between Host computer and 8051 microprocessor" is based on documents of "Data Formats and communication protocols" and the discussion of Dr. Cai, and Dr. Liu in the meeting at CSIST, November 2001. Some conditions of "Error", "Fail", "Abort" are still not very clear defined yet. We need more discussion with Dr. Cai and Prof. A. Lebedev. In other words, this protocol will be updated according to system requirement.

```
TX_Events( )    // this routine may be called by interrupt event or polling loop
{       For all TX(i) do
          If Host_Control_register.TX(i) == "Set new data"   then
            TX(i).mode <= TX_data; Do_packet_TX(i);
          If 8051_Control_register.TX(i) == "Busy" then
            Do_packet_TX(i);
        End for;
}
Do_packet_TX(i)
{
        Switch(TX(i).mode)
        Case   TX_data:
               If (block type of TX(i) is request:write or reply:read) then
                 { If (First packet or First&Last packet) then
                      If ( request:write ) then
                        TX_CAN(TX(i).ID+TX(i).Header+Data(TX(i).pointer));
                      Else { TX_CAN(TX(i).ID+Data(Tx(i).pointer)); }
                      Tx(i).pointer <= Tx(i).pointer +DLC;
                      TX(i).length <= TX(i).length –DLE;
                      8051_Control_register.TX(i) <= "Busy";
                      TX(i).mod <= Wait_Reply;
                   Else If (Intermediate packet) then
                          TX_CAN(TX(i).ID+Data(Tx(i).pointer));
                          Tx(i).pointer <= Tx(i).pointer +DLC;
                          TX(i).length <= TX(i).length –DLE;
                          TX(i).mod <= Wait_Reply;
                        Else //Last packet
                          TX_CAN(TX(i).ID+Data(Tx(i).pointer));
                          TX(i).mod <= End; }
               Else If   ( block type of TX(i) is request:read ) then
               {       TX_CAN(TX(i).ID+TX(i).Header+Data(TX(i).pointer));
                       TX(i).mod <= End; }
               Break;
        Case   Wait_Reply:
               If TX(i).length >0   then
               { If (block type of TX(i) is request:write or reply:read   ) then
                   If (RX(i).status.token = "request:next" or "reply:next") then
                       TX(i).mod <= TX_data;
                   Else {
                       Int_reg.8051_error_interrupt_host <= '1';
                       TX(i).mod <= Abnormal; }
               }
               Else { 8051_Control_register.TX(i) <= "Ready";
                       Int_reg.8051_ready_interrupt_host <= '1';
                       TX(i).mod <= TX_data; }
               Break;
        Case   End :
               If   RX(i).status.token == "Reply: End" then
                  { 8051_Control_register.TX(i) <= "Ready";
                   Int_reg.8051_ready_interrupt_host <= '1';
                    TX(i).mod <= TX_data;   }
```

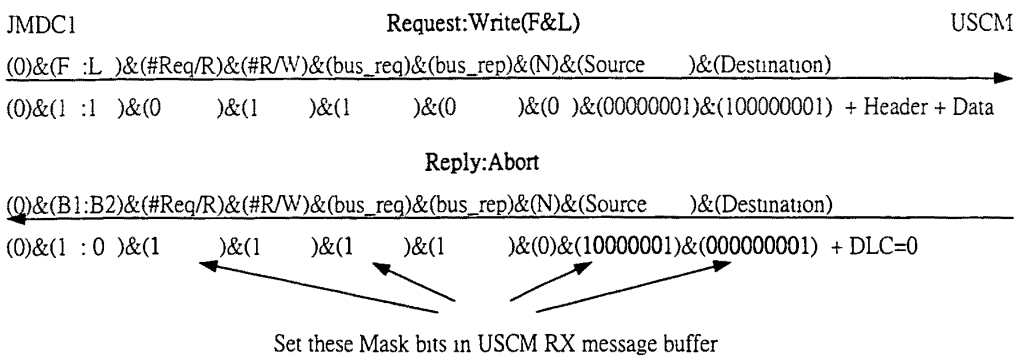
```
            Else  If  RX(i).status.token == "Error" then
                        TX(1).mod <= Abnormal;
            Break;
        Case  Abnormal:
              8051_Control_register.TX(i) <= "Error";
              Error_register <= Error Type;
              Int_reg.8051_error_interrupt_host <= '1';
              ....
              Break;
}

RX_Events( )    // this routine may be called by interrupt event or polling loop
{  For all RX(i) do
      If Host_Control_register.TX(i) == "Set Read done" then
       {  If RX(i).can.packet_status == ( Error or Time_out Or Abort ) then
          { Reply (Error or Abort);
            8051_Control_register.RX(i) <= "Error";
            Error_register.RX(i) <= Error Type; }
            Int_reg.8051_Error_interrupt_host <= 1; //Generate interrupt to Host;
          Else If RX(i).can.packet.status == OK then
              If  RX(i).packet.ID.DLC != 0 then  Received_F_L_RX(i);
              Else  Received_BC_RX(i);
       }
     End for;
}
Received_F_L_RX(i)
   {
      Switch()
      Case ( RX(i).packet.ID.first == 1 and RX(i).packet.ID.last == 0 ) :
            Move CAN_block_header and CAN_data to DPRAM(RX(i).pointer);
            Update RX(i).pointer and R(x).data_length= R(x).data_length+DLC;
            If CAN_block_header.data_type == "boot" then
              8051_Control_register.Boot <= boot.RX(i);
              Int_reg.Boot_interrupt_host <= 1; //Generate interrupt to Host;
            Else
              If ( can ID is request:write or reply:read ) then
                  8051_Control_register.RX(i) <= "Busy"; Reply "Next" ;
              Else If (can ID is request: read ) then
                  8051_Control_register.RX(i) <= "Ready";
                  Int_reg.8051_ready_interrupt_host <= 1;
            Break;
      Case ( RX(i).packet.ID.first == 1 and RX(i).packet.ID.last == 1 ) :
            Move CAN_block_header and CAN_data to DPRAM(RX(i).pointer);
            Update RX(i).pointer and RX(i).data_length= RX(i).data_length+DLC;
            8051_Control_register.RX(i) <= "Ready";
            Int_reg.8051_ready_interrupt_host <= 1; // interrupt Host;
            Break;
      Case ( RX(i).packet.ID.first == 0 and RX(i).packet.ID.last == 0 ) :
            Move CAN_block_header to DPRAM(RX(i).pointer);
            Update RX(i).pointer and RX(i).data_length= RX(i).data_length+DLC;
            If RX(i).data_length > Buffer_length -8 then
```

```
                {Move RX(i).data_length to DPRAM(RX(i).length);
                 RX(i).pointer<=8; RX(i).data_length=0;
                 8051_Control_register.RX(i) <= "Ready";
                 Int_reg.8051_ready_interrupt_host <= '1'; // interrupt Host;
                }
              Reply "Next";
              Break;
      Case ( RX(i).packet.ID.first == 0 and RX(i).packet.ID.last == 1 ) :
              Move   CAN_block_header to DPRAM(RX(i).pointer);
              Update RX(i).pointer and RX(i).data_length= RX(i).data_length+DLC;
              Move RX(i).data_length to DPRAM(RX(i).length);
              8051_Control_register.RX(i) <= "Ready";
              Int_reg.8051_ready_interrupt_host <= '1'; // interrupt Host;
              Break;
}
Received_B_C_RX(i)
{
      Case  ( RX(i).packet.ID.BC1 ==0 and RX(i).packet.ID.BC2 == 0 ) :
              If RX(i).packet.ID.request == 1 then
                 RX(i).status.token <=   "Request: Next";
              Else
                 RX(i).status.token <= "Reply: Next"   ;
              Break;
      Case  ( RX(i).packet.ID.BC1 ==0 and RX(i).packet.ID.BC2 ==1 ) :
              If RX(i).packet.ID.request == 1 then
                 RX(i).status.token <=   "Impossible";
              Else
                 RX(i).status.token <= "Reply: Error"   ;
                              Break;
      Case  ( RX(i).packet.ID.BC1 ==1 and RX(i).packet.ID.BC2 == 0 ) :
              If RX(i).packet.ID.request == 1 then
                 RX(i).status.token <=   "Request: Abort";
              Else
                 RX(i).status.token <= "Reply: Abort"   ;
              Break;
      Case  ( RX(i).packet.ID.BC1 ==1 and RX(i).packet.ID.BC2 ==1 ) :
              If RX(i).packet.ID.request == 1 then
                 RX(i).status.token <=   "not used";
              Else
                 RX(i).status.token <= "Reply: End"   ;
              Break;
   }
```

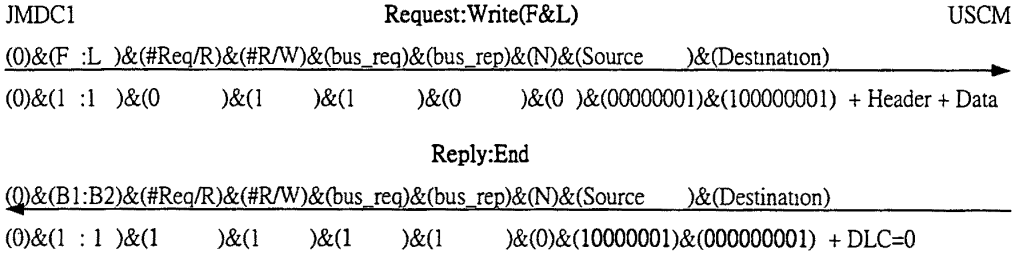
A.7 Examples of Communication Protocol via CAN Bus

A. Write Request(1)

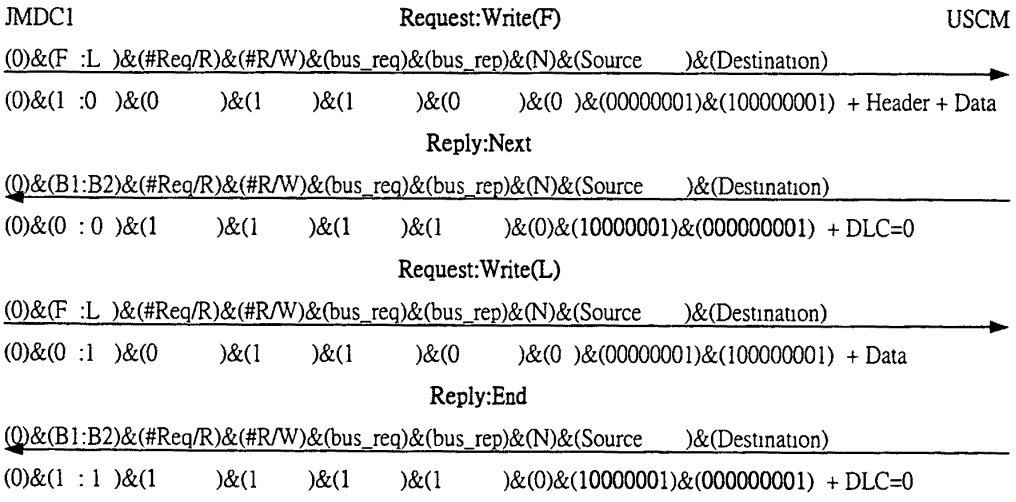
CASE 1:



CASE 2:



CASE 3:



CASE 4:

JMDC1 Request:Write(F) USCM

(0)&(F :L)&(#Req/R)&(#R/W)&(bus_req)&(bus_rep)&(N)&(Source)&(Destination) →

(0)&(1 :0)&(0)&(1)&(1)&(0)&(0)&(00000001)&(100000001)+Header+Data[0. 6]

Reply:Next

(0)&(B1:B2)&(#Req/R)&(#R/W)&(bus_req)&(bus_rep)&(N)&(Source)&(Destination) ←

(0)&(0 : 0)&(1)&(1)&(1)&(1)&(0)&(10000001)&(000000001) + DLC=0

Request:Write(I)

(0)&(F :L)&(#Req/R)&(#R/W)&(bus_req)&(bus_rep)&(N)&(Source)&(Destination) →

(0)&(0 :0)&(0)&(1)&(1)&(0)&(0)&(00000001)&(100000001) + Data[7..14]

Reply:Next

(0)&(B1:B2)&(#Req/R)&(#R/W)&(bus_req)&(bus_rep)&(N)&(Source)&(Destination) ←

(0)&(0 . 0)&(1)&(1)&(1)&(1)&(0)&(10000001)&(000000001) + DLC=0

Request:Write(I)

(0)&(F :L)&(#Req/R)&(#R/W)&(bus_req)&(bus_rep)&(N)&(Source)&(Destination) →

(0)&(0 :0)&(0)&(1)&(1)&(0)&(0)&(00000001)&(100000001) + Data[15..23]

Reply:Next

(0)&(B1:B2)&(#Req/R)&(#R/W)&(bus_req)&(bus_rep)&(N)&(Source)&(Destination) ←

(0)&(0 : 0)&(1)&(1)&(1)&(1)&(0)&(10000001)&(000000001) + DLC=0

Request:Write(L)

(0)&(F :L)&(#Req/R)&(#R/W)&(bus_req)&(bus_rep)&(N)&(Source)&(Destination) →

(0)&(0 :1)&(0)&(1)&(1)&(0)&(0)&(00000001)&(100000001) + Data[24..31]

Reply:End

(0)&(B1:B2)&(#Req/R)&(#R/W)&(bus_req)&(bus_rep)&(N)&(Source)&(Destination) ←

(0)&(1 : 1)&(1)&(1)&(1)&(1)&(0)&(10000001)&(000000001) + DLC=0

CASE 5:

JMDC1 Request:Write(F) USCM

(0)&(F :L)&(#Req/R)&(#R/W)&(bus_req)&(bus_rep)&(N)&(Source)&(Destination) →

(0)&(1 :0)&(0)&(1)&(1)&(0)&(0)&(00000001)&(100000001)+Header+Data[0..6]

Reply:Next

(0)&(B1:B2)&(#Req/R)&(R/#W)&(bus_req)&(bus_rep)&(N)&(Source)&(Destination) ←

(0)&(0 : 0)&(1)&(1)&(1)&(1)&(0)&(10000001)&(000000001) + DLC=0

Request:Write(I)

(0)&(F :L)&(#Req/R)&(#R/W)&(bus_req)&(bus_rep)&(N)&(Source)&(Destination) →

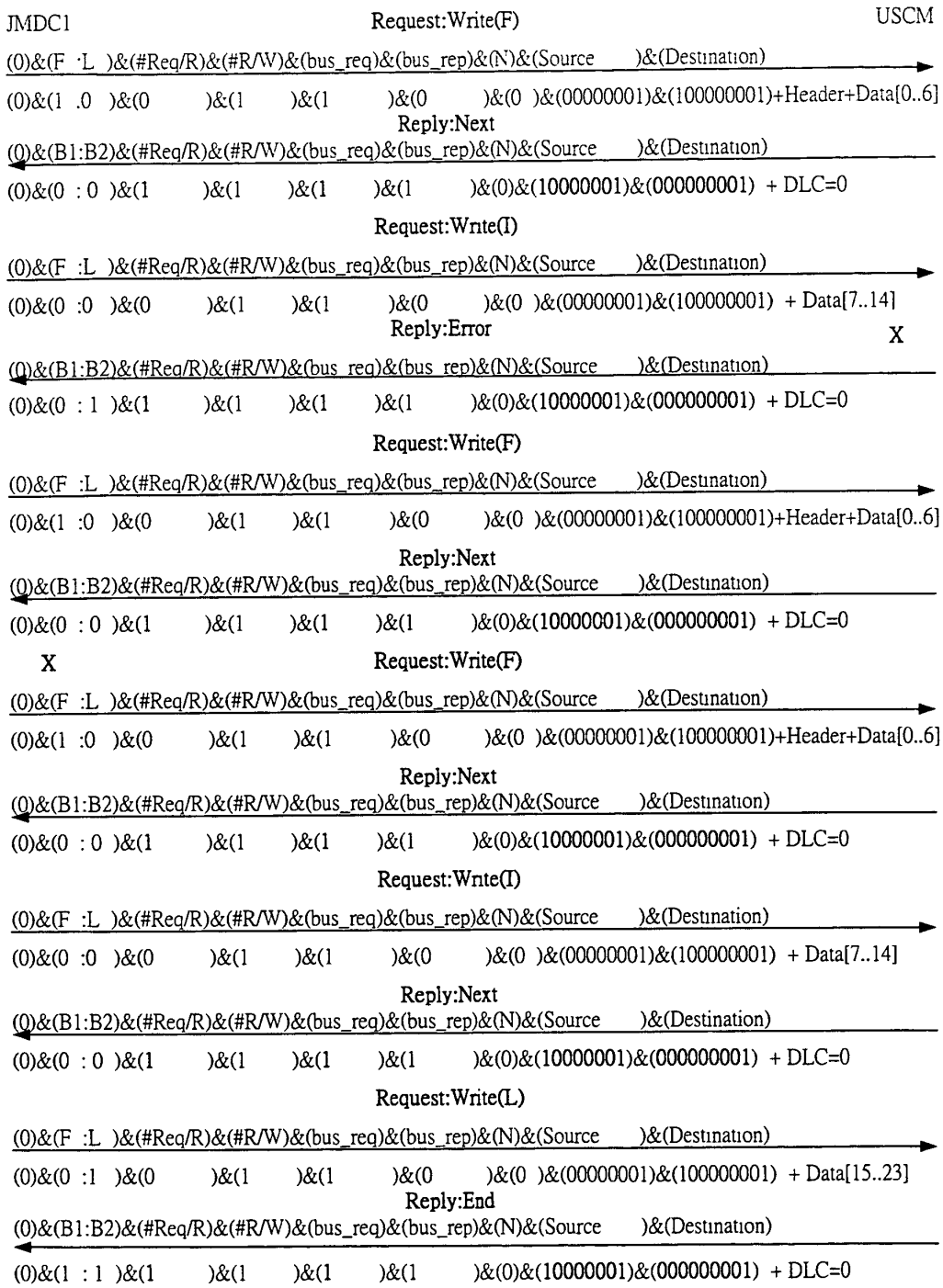
(0)&(0 :0)&(0)&(1)&(1)&(0)&(0)&(00000001)&(100000001) + Data[7..14]

Reply:Abort

(0)&(B1:B2)&(#Req/R)&(#R/W)&(bus_req)&(bus_rep)&(N)&(Source)&(Destination) ←

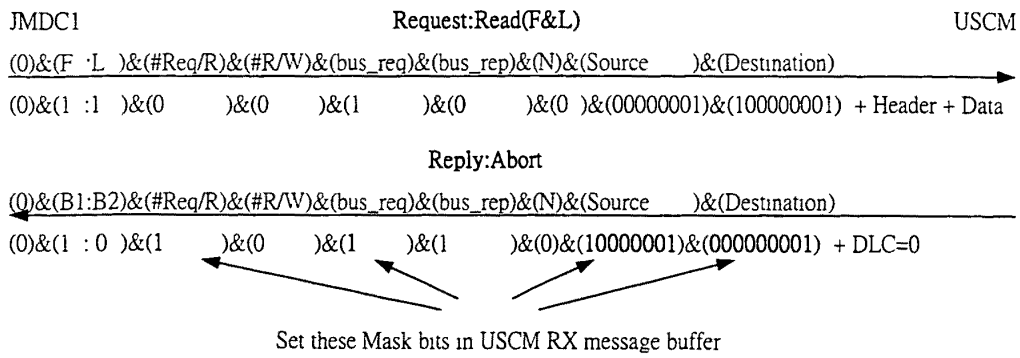
(0)&(1 : 0)&(1)&(1)&(1)&(1)&(0)&(10000001)&(000000001) + DLC=0

CASE 6:

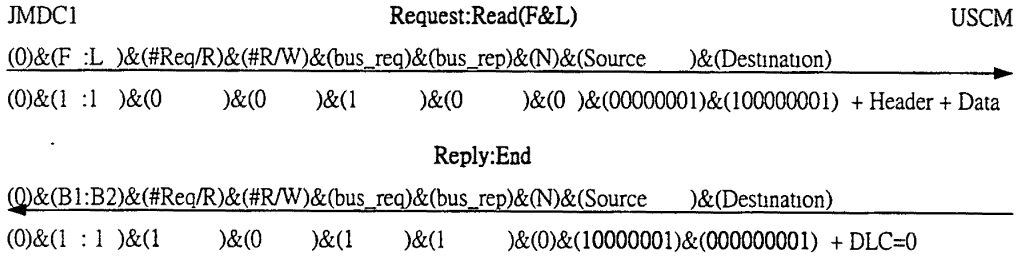

JMDC1
Request:Write(F)
USCM
(0)&(F :L)&(#Req/R)&(#R/W)&(bus_req)&(bus_rep)&(N)&(Source)&(Destination)
(0)&(1 :0)&(0)&(1)&(1)&(0)&(0)&(00000001)&(100000001)+Header+Data[0..6]
Reply:Next
(0)&(B1:B2)&(#Req/R)&(#R/W)&(bus_req)&(bus_rep)&(N)&(Source)&(Destination)
(0)&(0 : 0)&(1)&(1)&(1)&(1)&(0)&(10000001)&(000000001) + DLC=0
Request:Write(I)
(0)&(F :L)&(#Req/R)&(#R/W)&(bus_req)&(bus_rep)&(N)&(Source)&(Destination)
(0)&(0 :0)&(0)&(1)&(1)&(0)&(0)&(00000001)&(100000001) + Data[7..14]
Reply:Error
X
(0)&(B1:B2)&(#Req/R)&(#R/W)&(bus_req)&(bus_rep)&(N)&(Source)&(Destination)
(0)&(0 : 1)&(1)&(1)&(1)&(1)&(0)&(10000001)&(000000001) + DLC=0
Request:Write(F)
(0)&(F :L)&(#Req/R)&(#R/W)&(bus_req)&(bus_rep)&(N)&(Source)&(Destination)
(0)&(1 :0)&(0)&(1)&(1)&(0)&(0)&(00000001)&(100000001)+Header+Data[0..6]
Reply:Next
(0)&(B1:B2)&(#Req/R)&(#R/W)&(bus_req)&(bus_rep)&(N)&(Source)&(Destination)
(0)&(0 : 0)&(1)&(1)&(1)&(1)&(0)&(10000001)&(000000001) + DLC=0
X
Request:Write(F)
(0)&(F :L)&(#Req/R)&(#R/W)&(bus_req)&(bus_rep)&(N)&(Source)&(Destination)
(0)&(1 :0)&(0)&(1)&(1)&(0)&(0)&(00000001)&(100000001)+Header+Data[0..6]
Reply:Next
(0)&(B1:B2)&(#Req/R)&(#R/W)&(bus_req)&(bus_rep)&(N)&(Source)&(Destination)
(0)&(0 : 0)&(1)&(1)&(1)&(1)&(0)&(10000001)&(000000001) + DLC=0
Request:Write(I)
(0)&(F :L)&(#Req/R)&(#R/W)&(bus_req)&(bus_rep)&(N)&(Source)&(Destination)
(0)&(0 :0)&(0)&(1)&(1)&(0)&(0)&(00000001)&(100000001) + Data[7..14]
Reply:Next
(0)&(B1:B2)&(#Req/R)&(#R/W)&(bus_req)&(bus_rep)&(N)&(Source)&(Destination)
(0)&(0 : 0)&(1)&(1)&(1)&(1)&(0)&(10000001)&(000000001) + DLC=0
Request:Write(L)
(0)&(F :L)&(#Req/R)&(#R/W)&(bus_req)&(bus_rep)&(N)&(Source)&(Destination)
(0)&(0 :1)&(0)&(1)&(1)&(0)&(0)&(00000001)&(100000001) + Data[15..23]
Reply:End
(0)&(B1:B2)&(#Req/R)&(#R/W)&(bus_req)&(bus_rep)&(N)&(Source)&(Destination)
(0)&(1 : 1)&(1)&(1)&(1)&(1)&(0)&(10000001)&(000000001) + DLC=0

B. (Write and then) Read Request(1)

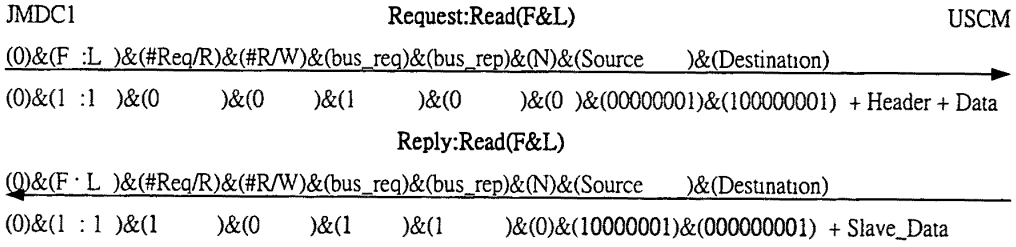
CASE 1:



CASE 2:



CASE 3:



CASE 4:

JMDC1 Request:Read(F&L) USCM

(0)&(F .L)&(#Req/R)&(#R/W)&(bus_req)&(bus_rep)&(N)&(Source)&(Destination) →

(0)&(1 :1)&(0)&(0)&(1)&(0)&(0)&(00000001)&(100000001)+Header+Data[0 .6]

Reply:Read(F)

(0)&(F .L)&(#Req/R)&(#R/W)&(bus_req)&(bus_rep)&(N)&(Source)&(Destination) ←

(0)&(1 . 0)&(1)&(0)&(1)&(1)&(0)&(10000001)&(000000001) + USCM_data[0..7]

Request:Next

(0)&(B1 :B2)&(#Req/R)&(#R/W)&(bus_req)&(bus_rep)&(N)&(Source)&(Destination) →

(0)&(0 :0)&(0)&(0)&(1)&(0)&(0)&(00000001)&(100000001) + DLC=0

Reply:Read(L)

(0)&(F :L)&(#Req/R)&(#R/W)&(bus_req)&(bus_rep)&(N)&(Source)&(Destination) ←

(0)&(0 : 1)&(1)&(0)&(1)&(1)&(0)&(10000001)&(000000001) + USCM_data[8..15]

CASE 5:

JMDC1 Request:Read(F&L) USCM

(0)&(F :L)&(#Req/R)&(#R/W)&(bus_req)&(bus_rep)&(N)&(Source)&(Destination) →

(0)&(1 :1)&(0)&(0)&(1)&(0)&(0)&(00000001)&(100000001)+Header+Data[0..6]

Reply:Read(F)

(0)&(F :L)&(#Req/R)&(#R/W)&(bus_req)&(bus_rep)&(N)&(Source)&(Destination) ←

(0)&(1 : 0)&(1)&(0)&(1)&(1)&(0)&(10000001)&(000000001) + USCM_data[0..7]

Request:Next

(0)&(B1 :B2)&(#Req/R)&(#R/W)&(bus_req)&(bus_rep)&(N)&(Source)&(Destination) →

(0)&(0 :0)&(0)&(0)&(1)&(0)&(0)&(00000001)&(100000001) + DLC=0

Reply:Read(I)

(0)&(F :L)&(#Req/R)&(#R/W)&(bus_req)&(bus_rep)&(N)&(Source)&(Destination) ←

(0)&(0 : 0)&(1)&(0)&(1)&(1)&(0)&(10000001)&(000000001) + USCM_data[8..15]

Request:Next

(0)&(B1 .B2)&(#Req/R)&(#R/W)&(bus_req)&(bus_rep)&(N)&(Source)&(Destination) →

(0)&(0 .0)&(0)&(0)&(1)&(0)&(0)&(00000001)&(100000001) + DLC=0

Reply:Read(L)

(0)&(F :L)&(#Req/R)&(#R/W)&(bus_req)&(bus_rep)&(N)&(Source)&(Destination) ←

(0)&(0 : 1)&(1)&(0)&(1)&(1)&(0)&(10000001)&(000000001) + USCM_data[16..23]