

ARTISAN®
TECHNOLOGY GROUP

Your **definitive** source for quality pre-owned equipment.

Artisan Technology Group
(217) 352-9330 | sales@artisantg.com | artisantg.com

Full-service, independent repair center
with experienced engineers and technicians on staff.

We buy your excess, underutilized, and idle equipment
along with credit for buybacks and trade-ins.

Custom engineering
so your equipment works exactly as you specify.

- Critical and expedited services
- In stock / Ready-to-ship
- Leasing / Rentals / Demos
- ITAR-certified secure asset solutions

Expert team | Trust guarantee | 100% satisfaction

All trademarks, brand names, and brands appearing herein are the property of their respective owners.

Visit our website - *Click HERE*

FIC 8230

Fast Intelligent Controller

User's Manual, version 1.0
Designation: DOC 8230/UM
PN: 085.209

Version 1.0 - June 1992

CREATIVE ELECTRONIC SYSTEMS S.A.

Creative Electronic Systems

FIC 8230

USER'S MANUAL

The information in this document has been checked carefully and is thought to be entirely reliable. However, no responsibility is assumed in case of inaccuracies. Furthermore, CES reserves the right to change any of the products described herein to improve reliability, function or design. CES neither assumes any liability arising out of the application or use of any product or circuit described herein nor conveys any licence under its patent rights or the rights of others.

WARNING

THIS EQUIPMENT GENERATES, USES AND CAN RADIATE RADIO FREQUENCY ENERGY AND MAY CAUSE INTERFERENCE TO RADIO COMMUNICATIONS IF NOT INSTALLED AND USED IN ACCORDANCE WITH THE INSTRUCTION MANUAL. IT HAS BEEN TESTED AND FOUND TO COMPLY WITH THE LIMITS OF A CLASS A COMPUTING DEVICE PURSUANT TO SUBPART J OF PART 15 OF FCC RULES, WHICH ARE DESIGNED TO PROVIDE REASONABLE PROTECTION AGAINST SUCH INTERFERENCES WHEN OPERATED IN A COMMERCIAL ENVIRONMENT. OPERATION OF THIS EQUIPMENT IN A RESIDENTIAL AREA IS LIKELY TO CAUSE INTERFERENCE; IN WHICH CASE, THE USER AT HIS OWN EXPENSE WILL BE REQUIRED TO TAKE WHATEVER MEASURES ARE NECESSARY TO CORRECT THE INTERFERENCE.

All Rights reserved
Version 1.0 - March 1991

The reproduction of this material, in part or whole, is strictly prohibited. For copy information, please contact:

Creative Electronic Systems,
70 Route du Pont-Butin
P.O. Box 107
CH-1213 Petit-Lancy 1, Geneva, Switzerland.

The information in this document is subject to change without notice. Creative Electronic Systems assumes no responsibility for any error that may appear in this document.

FIC 8230

CONTENTS

Chapter 1 : Introduction 1.1

1.1. Introduction 1
1.2. Features 1
1.3. FIC 8230 - FIC 8231 - FIC 8232 Differences 2
1.4. Ordering Information & Reference options 2
1.5. Power Supply Requirement 3
1.6. Reference Manuals 3
1.7. Block Diagram Description 4
1.7.1. 32-bit Local Bus 4
1.7.2. 8-bit Local Bus 4
1.7.3. 32-bit Global Bus 4
1.7.4. VSB Interface 5
1.7.5. VME Slave Interface 5
1.7.6. VME Master Interface 5
1.8. Addressing Overview 5
1.9. System Clock 6
1.10. External CPU Control - RESET, ABORT and HALT/RUN 6
1.10.1. RESET Generation 6
1.10.1.1. Front Panel RESET 7
1.10.1.2. Software RESET 7
1.10.1.3. Other RESETS 8
1.10.2. ABORT Generation 8
1.10.2.1. Front Panel ABORT 8
1.10.2.2. Software ABORT 9
1.10.3. Halt/Continue CPU Control 9
1.11. Interrupt Handling Structure 9
1.11.1. SBC 68155 Chip Description 9
1.11.2. FIC 8230 and SCB Hardware Implementation 10
1.11.3. Interrupt Sources Description 10
1.12. On Board Time-out 11

Chapter 2 : Local Resources 2.1

2.1. Local Resources 1
2.2. Local Memories 1
2.2.1. Static RAM Memory 1
2.2.2. EPROM Memory 2
2.3. MC 68882 Floating Point Unit 2
2.4. DMA Controller WE 32104 3
2.4.1. Chip Description 3
2.4.2. DMA Hardware Implementation 3
2.4.3. Internal Registers Access (WE 32104) 4
2.4.4. DMA Data Transfer Memory-to-Memory 4
2.4.5. External Options for WE 32104 4
2.4.6. WE 32204 Peripheral Bus 5
2.4.7. Block Mode Support 5
2.5. Serial Controller (Z8030) 5
2.5.1. Chip Description 6
2.5.2. Hardware Implementation 6
2.5.3. Z8030 I/O Mapping 6
2.6. Timer, I/O Port & Interrupt Controller 6
2.6.1. Chip Description 6
2.6.2. Z8036 Hardware Implementation 7

2.6.2.1. Communication FIFO Interrupt Controller 7
2.6.2.2. General Purpose Counter/Timers 7
2.6.2.3. Front Panel I/O Port 7
2.6.3. Interrupt Logic 8
2.6.4. Z8036 I/O Mapping 8
2.7. DP8570 Real-Time Clock 8
2.7.1. Chip Description 8
2.7.2. Hardware Implementation 8
2.7.3. DP8570AI/O Mapping 9
2.8. SCB68154 VME Interrupter 9
2.8.1. Chip Description 9
2.8.2. Hardware Implementation 9
2.9. SCB 68155 Interrupt Handler 9

Chapter 3 : Global Resources **3.1**

3.1. General Philosophy 1
3.1.1. CPU/DMAC Access 2
3.1.2. VME Access to the Global Memory 2
3.1.3. Global Arbitration and Indivisible Cycle 2
3.2. Global Memory 3
3.2.1. OPT8230/1 Static RAM Memory Module 4
3.2.2. OPT8232/6 Dynamic RAM Memory Module 5
3.3. Control and Status Registers 6
3.3.1. Control Registers 6
3.3.1.1. CTLCPU - Adr = $2000'0000 6
3.3.1.2. CTLVSB - Adr = $2000'0004 8
3.3.1.3. CTLVME1 and CTLVME2 8
3.3.2. Status Registers 10
3.3.2.1. STAVSB - Adr = $2000'0400 10
3.3.2.2. STACPU - Adr = $2000'0404 10
3.3.2.3. STAVME - Adr = $2000'0408 10
3.3.2.4. STAFIF - Adr = $2000'040C 11
3.4. FIFO/LIFO Communication 11
3.4.1. FIFO/LIFO Introduction 11
3.4.2. FIFO/LIFO Structure 12
3.4.3. FIFO/LIFO Control Registers 13
3.4.3.1. Memory Buffer 13
3.4.3.2. FIFO Control Registers 13
3.4.3.3. Flag Registers 14
3.4.3.4. FIFO/LIFO I/O Registers 14
3.4.4. Interrupt Generation on the FIFO/LIFO Status 14
3.4.5. FIFO/LIFO Initialization 15

Chapter 4 : VME Slave Interface **4.1**

4.1. General Description 1
4.2. AM Decoding 1
4.3. Standard Access A24 - D16/D32 1
4.4. Extended Access A32 - D16/D32 2
4.5. Data Type & Timing 3
4.6. READ MODIFY WRITE Cycle 3
4.7. Block Mode Support 3

Chapter 5 : VME Master Port 5.1

5.1. Addressing Philosophy 1
5.2. Data Port Size 2
5.2.1. D16 Data Port 2
5.2.2. D32 Data Port 2
5.3. AM Generation & VME Addressing Spaces 2
5.3.1. Extended Addressing Space 2
5.3.2. Standard Addressing Space 3
5.3.3. Short I/O Addressing Space 3
5.3.4. Special Addressing Spaces 4
5.4. VME Arbitration and Requester 4
5.4.1. Arbitration Pad socket 5
5.4.2. ARB 8430/2 Slot 1 Functions & 4 Level Arbiter for the FIC8230 6
5.4.2.1. Introduction 6
5.4.2.2. Installation 6
5.4.2.3. Jumper description 6
5.4.2.4. Bus timer function 6
5.4.2.5. System clock Driver 7
5.4.2.6. VME arbitration 7
5.4.2.7. Fair mode 7
5.4.2.8. Layout 8
5.4.3. On Board 4 Level Requester 8
5.5. RMW and IACK Special Cycles on VME 9
5.5.1. Read Modify Write Cycle 9
5.5.2. Iack Cycles 9
5.6. DMAC Block Mode Access 9
5.7. Special Lines Monitoring & Control 9

Chapter 6 : VSB Master Interface 6.1

6.1. VSB Introduction 1
6.2. VSB Addressing 1
6.3. Transfer Cycles 1
6.3.1. Block Transfer Support 2
6.4. VSB Arbitration 2
6.5. Interrupt Capability 3
6.6. Other Status Bits 3
6.7. VSB Pin Assignment 3

Chapter 7 : General Information 7.1

7.1. Front Panel Description 1
7.1.1. Front Panel LED Indicators 1
7.1.2. Lemo ØØ CRCTL Output 1
7.1.3. RS232 Connectors 1
7.2. General Purpose Connector (P5) 2
7.3. Installation Notes 3

Appendix : EPROM Contents A.1

A.1. Parameter definition file 1
A.2. FIC 8230 environment setup code for pROBE-SDT 8
A.3. pROBE - FIC 8230 parameter definition file 10

Chapter 1

INTRODUCTION

The FIC 8230 is a Motorola MC68020 based card that can access both the VME bus and the VSB auxiliary bus.

1.1. Introduction

This manual provides general information, hardware preparation, installation instructions, functional descriptions and support information for the FIC 8230, 32-bit Microprocessor unit (referred to throughout this document as the FIC 8231). This manual is divided into two parts.

1) ch. 1 --> 7 : Hardware descriptions

2) ch. 8 : Software descriptions

1.2. Features

* MC 68020 (16 and 25 Mhz))
* MC 68881 floating-point co-processor
* 4 DMA channels with 32-bit data/address capacity
* SRAM piggy-back 2-port zero-wait state memory (in CMOS), configurable from 512 Kbytes to 4 Mbytes
* DRAM piggy-back with 2 & 4 Mbytes
* 2 RS232-C ports
* Timers and event counters
* 128 Kbytes, zero-wait-state local memory (SRAM)
* VME master/slave interface
  - master A16: D16/D32, A24: D16/D32, A32: D32
  - slave A24: D16/D32, A32: D16/D32
* VSB master interface, I/O and memory + arbiter
* VME interrupter and interrupt handler; VSB interrupt handler
* 4 independent FIFO communication FIFOs accessible from both CPU and VME
* Card configurable by software
* Minimum static configuration options (jumpers, switches, wraps)
* Real-time clock, 2 timers, plus 40 bytes of RAM with battery back-up

1.3. FIC 8230 - FIC 8231 - FIC 8232 Differences

	FIC8230	FIC8231	FIC8232
CPU	68020 16/24 MHZ	68030 16/24 MHZ	68030 16/24 MHZ
EPROM	128/256 KB MAX.	128/256 KB MAX.	1 MB MAX.
LOCAL RAM	128 KB	128KB	128/512 KB
GLOBAL RAM	512KB-4MB	512KB-4MB	512KB-16MB
16-BIT TIMERS	3	3	6
VME REQUESTER	OPTION	OPTION	ON-BOARD
VME SLAVE BLT	4 LW MAX.	4 LW MAX.	255 LW MAX.
FIFO BLT	NO SUPPORTED	NO SUPPORTED	SUPPORTED
SCSI	GPC ATTACHED	GPC ATTACHED	ON-BOARD
ETHERNET	NO	NO	YES (OPTION)
CPU BUS RETRY	NO	NO	YES
VME SLAVE DYN. MAP.	28-BIT	28-BIT	32-BIT
RTC BATTERY CAPACITY	38 mA	38 mA	200 mA
FRONT PANEL RESET	OPTION	OPTION	ON-BOARD
FRONT PANEL ABORT	OPTION	OPTION	ON-BOARD
MASTER NUMBER	ON-BOARD	ON-BOARD	FRONT PANEL

1.4. Ordering Information & Reference options

Designation	Description
FIC 8230/16	FIC 8230 16 Mhz, 128 Kbytes Private SRAM
FIC 8230/24	FIC 8230 25 Mhz, 128 Kbytes Private SRAM
OPT 8230/3/16	FPU coprocessor for FIC 8230/16 (MC68881 - 16 Mhz)
OPT 8230/3/24	FPU coprocessor for FIC 8230/24 (MC68881 - 24 Mhz)
OPT 8230/32/24	FPU coprocessor for FIC 8230/24 (MC68882 - 24 Mhz)
OPT 8230/2/16	32-bit DMA controller for FIC 8230/16 (WE 32104)
OPT 8230/2/24	32-bit DMA controller for FIC 8230/24 (WE 32204)

Static Memory options:

OPT 8230/1A/16	512 Kbytes zero-wait state SRAM lower piggy-back for FIC8230/16
OPT 8230/1B/16	512 Kbytes zero-wait state SRAM upper piggy-back for FIC8230/16
OPT 8230/1A/24	512 Kbytes zero-wait state SRAM lower piggy-back for FIC8230/24
OPT 8230/1B/24	512 Kbytes zero-wait state SRAM upper piggy-back for FIC8230/24

Dynamic Memory options:

OPT8230/7/16	2 Mbytes dynamic memory piggy-back for FIC 8230/16
OPT8230/7/24	2 Mbytes dynamic memory piggy-back for FIC 8230/24
OPT 8232/6/16	4 Mbytes dynamic memory piggy-back for FIC 8230/16
OPT 8232/6/24	4 Mbytes dynamic memory piggy-back for FIC 8230/24

Arbitrations:

ARB8430/1	Requester
ARB8430/2	Slot 1 + Requester
ARB8430/3	Requester Fair Mode

Other options:

OPT 8230/4	DMA, SCSI on-board controller
OPT 8230/5	Front panel extended console
OPT8230/10	EPROM extension 1 Mbyte

1.5. Power Supply Requirement

The FIC 8230 with the following options:

- 512 Kbytes global Memory
- DMAC
- FPU 68881

consumes :
+ 5V : 3.2 [A] typical 3.8 [A] max.
+12 V : 10 [mA] typical
- 12V : 10 [mA] typical

1.6. Reference Manuals

The following publications may provide helpful information. If they are not shipped with this manual, they may be obtained from our headquarters in Geneva or directly from the manufacturer.

Document	Manufacturer
VMEbus specifications rev. C	Motorola/VITA/IEC
VSBbus specifications rev. C	VITA/IEC
pROBE/pSOS/pRISM	Software Component Group
MC 68020 User's Guide	Motorola
MC 68881 User's Guide	Motorola
WE 32104 DMA controller data sheet	AT&T
Z8036, Z8031 data sheets & technical manuals	Zilog
DP8570 data sheet	NS
SCB 68155, SCB 68154 data sheets	Philips/Signetics

1.7. Block Diagram Description

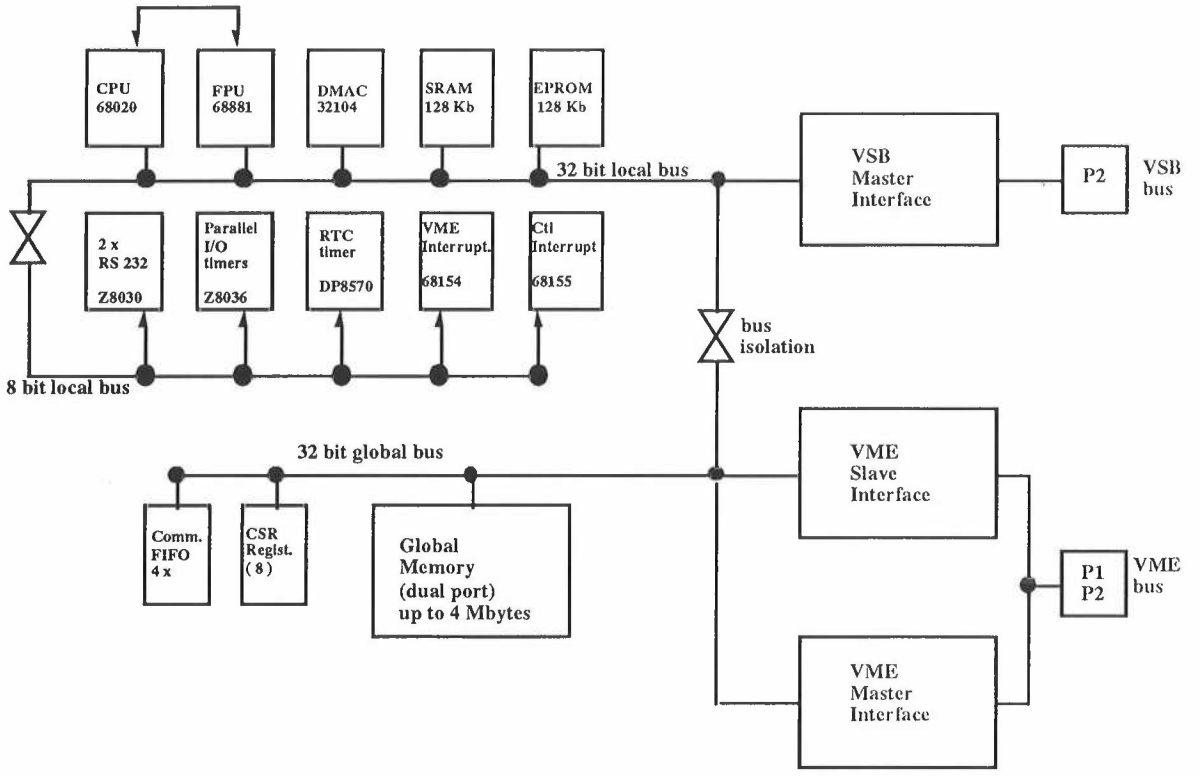


Fig 1.1 : Block Diagram

The FIC 8230 is organized internally around 3 separate busses:

- Local bus 32-bit address/data
- Local bus 8-bit data bus
- Global bus 32-bit address/data bus

All blocks are connected on these busses.

1.7.1. 32-bit Local Bus

This 32-bit bus is the microprocessor bus; the local memory, the DMAC 32104 and the floating point coprocessor are connected to it. It assures a zero wait-state between those elements.

1.7.2. 8-bit Local Bus

This 8-bit bus is derived directly from the 32-bit bus. It supports the peripheral devices, such as the Z8030, Z8036, DP8570, 68154, 68155. This bus is isolated electrically from the processor bus.

1.7.3. 32-bit Global Bus

This 32-bit bus supports the global resources as well as the global memory, the CSR registers and the four communication FIFOs. This bus can be driven by the local bus or by the VME slave interface. An arbitration controls the accesses from the two parts. The Global bus is also used when the CPU/DMAC accesses the VME (master mode).

1.7.4. VSB Interface

This interface allows access to the VME Subsystem Bus (VSB). It is directly connected to the 32-bit local bus. The CPU and the DMAC can access different spaces of the VSB. An arbiter/requester is also implemented on this interface.

1.7.5. VME Slave Interface

This block allows us to map the global resources (1 Mbyte) into the VME addressing space. It supports all data transfer sizes and is able to decode VME in A24 and A32 addressing mode.

It can also generate interrupts in the VME via an "Interrupt Generator".

1.7.6. VME Master Interface

This block allows the CPU and the DMAC to access VME resources . It is able to support data transfer in 8, 16 or 32 bits in all standard VME access modes. It can also be a VME interrupt handler.

1.8. Addressing Overview

The 4 Gbytes addressing range of the CPU and the DMAC is segmented into 4 areas:

1) local resources
2) global resources
3) VSB access
4) VME access

For the exact device address mapping, consult the "FIC8230 parameter definition file", 8230ref.s, in chapter 8.

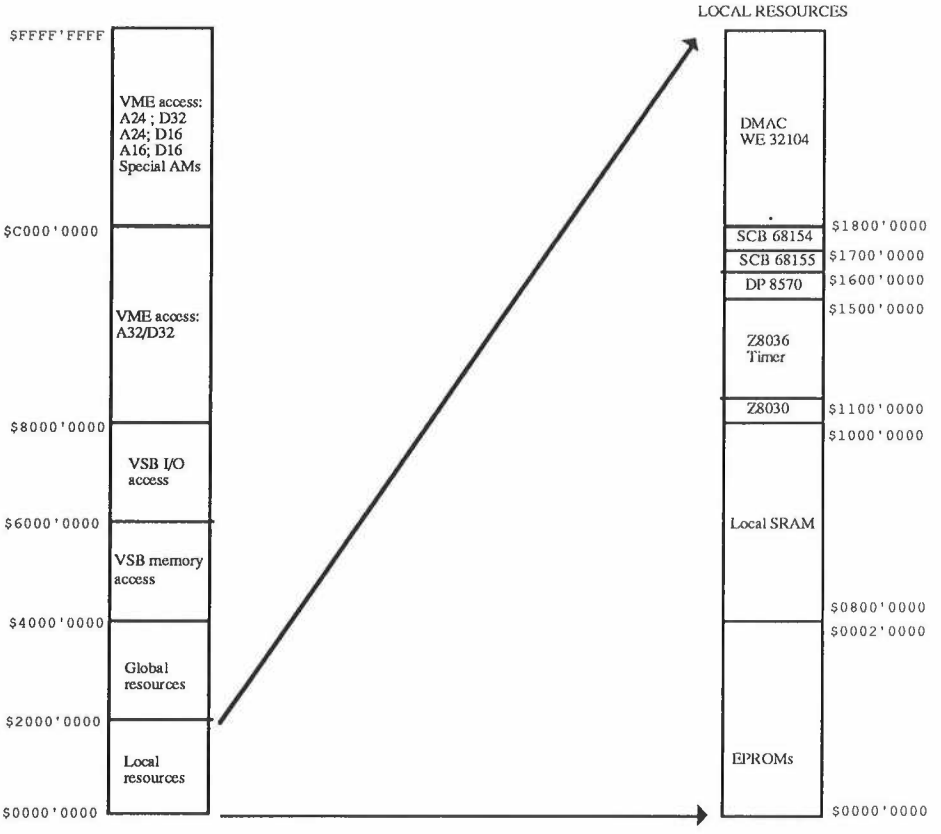


Fig 1.2 : Addressing Overview

1.9. System Clock

The FIC 8230 contains 3 different clock generators:

1) CPU, FPU & DMAC operation
2) Peripheral chips
3) Real-time clock oscillator

1) The CPU/FPU & DMAC clocks are generated by an integrated oscillator with a frequency double that of the CPU clock.

Oscillator for CPU/FPU & DMAC

16.67 Mhz CPU	->	33.33 Mhz oscillator.
20 Mhz CPU	->	40.00 Mhz
25 Mhz CPU	->	50.00 Mhz

This clock is used for internal sequential logic and is divided by two for the CPU/FPU operation. The DMAC is also driven by this clock.

2) The peripheral chips (Z8036, Z8030, DP8570, 68155, 68154) are driven by an independent clock generator that allows the CPU speed to be upgraded without changing parameters in the peripheral devices (especially for timers and baud rate generators).

This 20 Mhz integrated oscillator is divided to generate the 10 Mhz for the timers and the 5 Mhz to drive the Z8030/Z8036 PCK signal.

3) The real-time clock, RTC (DP8570), is equipped with its own oscillator (with battery back-up) to assure completely independent operation.

1.10. External CPU Control - RESET, ABORT and HALT/RUN

The CPU can be externally controlled either via the front panel connection or via the VME slave port. The external CPU controls are:

- RESET
- ABORT (high level interrupt)
- HALT/CONTINUE

1.10.1. RESET Generation

The generation of the RESET will initialize the FIC 8230 to its power-on state. This reset will act on different parts of the FIC 8230.

- The CPU 68020 will stop program execution and jump to addr 0 of the EPROM (restart address).

- The peripheral devices (WE 32104, FPU 68881, SCB 68155 and SCB 68144) are also hardware reset and placed in their wait state.

- The control registers are reset to "0".

- Some devices (Z8036, Z8030, DP 8570, etc) must be reset by software. This must be done in the bootstrap procedure.

- The RESET generation can be done in four different ways:

(1) by the instruction RESET (look at the 68020 description).
(2) by activation of line(s) on the GPC (General Purpose Connector) on the front panel.
(3) by the VME line SYSRESET
(4) by writing the correct pattern into the CPUCTL register from VME or from the CPU itself (software RESET).

1.10.1.1. Front Panel RESET

The front panel RESET can be implemented in two ways. The first way is to directly connect a push-button switch, and the second by using a direct negative TTL pulse signal.

The generation of RESET by a push-button switch is done by using a 3-wire connection. The FIC 8230 has a debouncing logic in its mother board. These 3 lines are cabled on the GPC (General Purpose Connector) as shown in the following diagram.

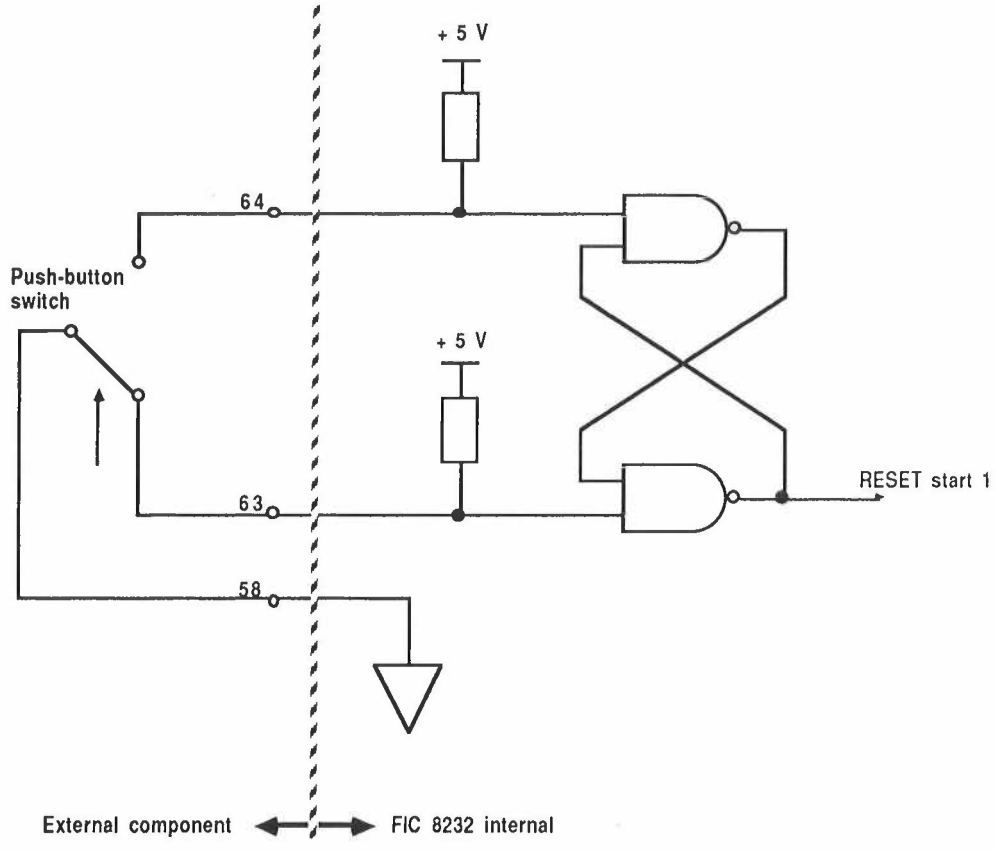


Fig. 1.2. Connection for RESET generated by the GPC

The RESET generation, by means of the direct TTL input, requires the use of the GPC pin No. 60. This input is active "low".

Direct TTL RESET : GPC pin 60

1.10.1.2. Software RESET

The software RESET is done by writing the CTLCPU register with the correct bit pattern. This register is located in the global resources area.

The pattern of bits (0 & 1) uCTL1 and uCTL0 is fixed to 01 to generate a RESET.

Software RESET : WRITE @CTLCPU with ....., X, X, X, X, 01

1.10.1.3. Other RESETS

A RESET is also generated when the VME line: SYSRESET is activated, and the CPU is also capable of generating RESETS by using the proper instruction.

1.10.2. ABORT Generation

The ABORT activator generates a high level interrupt to the CPU. This high level interrupt (level 5) must be initialized ion the SBC 68155 (Interrupt controller) and the corresponding vector number must be stored in that device. The ABORT interrupt is the highest interrupt available after the power-fail detection (AC or DC).

The ABORT activator can be achieved in two ways:

1) By activation of line(s) of the GPC on the front panel.
2) By writing the CPUCTL register with the correct bit pattern.

1.10.2.1. Front Panel ABORT

As in the RESET activator, the front panel ABORT activator can be triggered either with a push-button switch or by a direct negative TTL pulse.

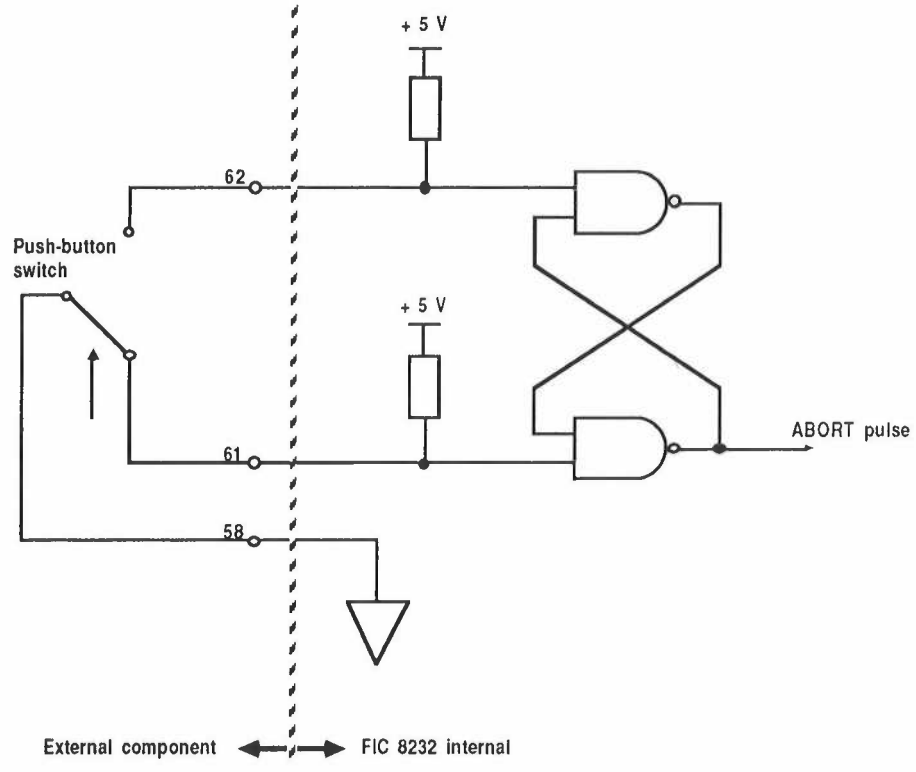


Fig. 1.3. Connection for ABORT generated by the GPC

Direct TTL ABORT : GPC pin 59

1.10.2.2. Software ABORT

The generation of the software ABORT is done by writing the correct pattern bits uCTL1 & uCTL0 in the CPUCTL register.

Prog ABORT : Write @CPUCTL with : X, X, X, X, X, X, 10

1.10.3. Halt/Continue CPU Control

Via the " CPUCTL" register it is possible to "HALT" the CPU and to "CONTINUE" it. This feature is also implemented with the two uCTL bits (0 & 1) as follows:

CPU HALT : Write @CTLCPU with : XXXX'XX11

CPU RUN : Write @CTLCPU with : XXXX'XX00

1.11. Interrupt Handling Structure

The FIC 8230 Interrupt Handler processes various interrupt signals into group and level categories to ensure proper priority and subsequent handling by the MC68020. The interrupt handler logic is is implemented through a Signetics SCB 68155 circuit, controlling 14 interrupt sources.

The different interrupt sources are as follows:

	IRQ Source	Interrupt CPU Level
(Highest priority)	VME ACFAIL, SYSFAIL	7
	VME IRQ 7	7
	8570 IRQ	6
	ABORT	5
	VSB IRQ	4
	Z8036/Z8030	3
	DMAC IRQ	2
	VSB IRQ	1
	VME IRQ 6	6
	VME IRQ 5	5
	VME IRQ 4	4
	VME IRQ 3	3
	VME IRQ 2	2
(Lowest priority)	VME IRQ 1	1

1.11.1. SBC 68155 Chip Description

The Signetics SCB 68155 is an asynchronous interrupt handler for the MC 68000 processor family. Up to 14 sources are prioritized by the SCB 68155 to one of seven levels and are output on the interrupt priority level lines (IPL0-2). The SCB 68155 prioritizes the interrupt in the following manner: local bus requests over system bus (VME) requests with the NMI considered as the highest local interrupt.

(NMI over IRQ7 VME, the local request LRQ6 LRQ1 over VME IRQ6-IRQ1)

During a local interrupt acknowledge sequence, two modes of response are available: vector mode or device-supplies-the-vector mode.

For system bus response, the SCB 68155 works with the VME master interface to acquire STATUS/ID byte from the VME bus.

The 68155 allows each source to be masked and is completely user configurable by appropriate software. The manufacturer data sheet furnishes a complete description.

For hardware compatibility, the 68155 must be initialized with:

LIRQ 7, 6, 5, 4 : vector supplied by the 68155
LIRQ 3,2,1 : vector supplied by the interrupt devices.

1.11.2. FIC 8230 and SCB Hardware Implementation

The SCB 68155 is mapped in the local resources of the FIC 8230. It can be accessed by using a READ/WRITE cycle (MOVE.B instructions).

68155 Mapping

R0 = $ 1600'0000
R1 = $ 1600'0002
R2 = $ 1600'0004
R3 = $ 1600'0006
R4 = $ 1600'0008
R5 = $ 1600'000A
R6 = $ 1600'000C
R7 = $ 1600'000E

1.11.3. Interrupt Sources Description

a) VME IRQ1-7

These interrupt requests are directly connected to the 68155 circuit. They can interrupt the MC68020 (if the corresponding masks are disabled) at the same level as the source.

Special care must be taken with the IRQ7, which is edge sensitive and has the second highest level in the interrupt priority structure.

During the IACK cycle, the FIC 8230 fetches the STATUS/ID from VME in 8-bit format and uses it directly for its vector number. To achieve this operation, the VME master interface must work correctly (arbitration, daisy-chain) and the IACK daisy-chain must be correctly installed.

b) LIRQ7 VME ACFAIL and VME SYSFAIL (edge sensitive, active low)

This local interrupt (highest priority) is the ACFAIL (power loss detection) coming from VME. It must be initialized to work on edge sensitive and in active-low state. The vector must be supplied by the 68155 itself. ACFAIL and SYSFAIL are ORed together.

Remark: In some VME backplanes, a glitch can appear on ACFAIL; for this reason software recovery must be implemented before initiating a power fail procedure.

c) LIRQ6 DP8570 (active low, level sensitive)

This local interrupt is generated by the DP 8570 RTC chip. It can signal a DCFAIL, a timer alarm or a periodic interrupt. The vector must be supplied by the 68155.

d) LIRQ5 ABORT (active low, edge sensitive)

This local interrupt is provided by the ABORT logic generation. It can be issued by the Front Panel or by other VME CPUs via the global resources. This interrupt can also be used for a "Mail Box" interrupt structure.

e) LIRQ4, LIRQ1 VSB IRQ (active low, level sensitive)

This local interrupt is generated by the VSB interface bus. The vector can be supplied by the 68155 or eventually with special care fetched directly form the VSB. When the vector number is fetched from VSB, be sure that the VSB autovectoring is supported by all VSB interrupters and devices in the system.

Only one of the two possible LIRQ (LIRQ1 & LIRQ4) must be enabled.
LIRQ4 must be supplied by the 68155.
LIRQ1 must be supplied by the VSB interrupter.

f) LIRQ3 Z8036/Z8030 (level sensitive/active low)

This LIRQ is originated in the ZILOG peripheral devices. Both devices are connected to level 3. The Z8036 has the higher priority. The IRQ from the Z8030 (serial controller) can signal Break detection or an error condition. The Z8036 is the most important interrupter in the system. It signals interrupts from:
- the FIFO's communication STATS/FLAG
- the front panel
- the Z8036's three internal timers.

The internal priority of the Z8036 can be selected by software. The vector has to be supplied by the Z8036 or the Z8030.

g) LIRQ2 DMAC WE 32104 (level sensitive, active low)

This LIRQ is allocated with the DMAC 32104. This device can interrupt the CPU to signal end of transfer, error conditions, etc. The vector must be supplied directly by the DMAC.

Refer to the MC68020 User's Manual for the vector table area, and to the peripheral devices for the interrupt structure.

1.12. On Board Time-out

An on board "bus time-out" allows a cycle to be aborted if there is no acknowledgement after a preset time. This time-out value is defined by using TIMER1 of the RTC DP8570. It can be configured by software directly in the DP8570. The time-out logic generates a BERR to the CPU/DMAC.

time-out [ns]= $[f_{out}]^{-1}$ X 16
(for a 1 Mhz f_{out} from the DP8570A » time-out occurs after 16 µs)

The time-out is associated with a flag located in the "STACPU" register, which is set when a time-out condition occurs. This flag remains asserted until it is cleared by a write cycle in the "CTLCPU" register. This flag is used to determine the source of BERR during the exception routine procedure.At powering-up or during a RESET this flag is reset.The value of this time-out is normally fixed during the bootstrap procedure (see Chapter 8 for example).

Remark: The time-out must be greater or equal to 3.2µs

Chapter 2

LOCAL RESOURCES

2.1. Local Resources

This chapter describes the resources directly connected to the CPU/DMAC bus. These resources can be reached without using the global bus, and are implemented with different bus sizes.

32-bit data : WE 32104, MC 68881, local SRAM
16-bit data : local EPROM 27C512
8-bit data : Z8036, Z8030, DP8570, SN68154, SN68155

For complete information on the VLSI used in the FIC 8230, consult the following original factory data sheets:

WE32204 : AT&T doc reference, 307-733
MC68881 : Motorola user's guide, reference MC68020UM/AD
Z8036 : Zilog data sheet
Z8031 : Zilog data sheet
DP8570A : National Semiconductor data sheet
SCB 68154 : Signetics data sheet
SCB68155 : Signetics data sheet

The following table summarizes the physical address of the FIC 8230 global resources:

Local Resources	Bus Size	Base Address	Size	Remarks
Local SRAM	32	$0800'0000	128 Kbytes	
Local EPROM	16	$0000'0000	128 Kbytes	Restart, Boot
Z8031	8	$1000'0000	32 bytes	Serial ctl
Z8036	8	$1100'0000	64 bytes	
DP8570	8	$1500'0000	32 bytes	RTC
SCB 68155	8	$1600'0000	8 bytes	Interrupt ctl
SCB 68154	8	$1700'0000	2 bytes	VME interrupter
WE32104	32	$1800'0000	1 Kbyte	DMAC 4 ch

The complete address references are located in the " FIC8230 parameter definition file" in Chapter 8.

2.2. Local Memories

The FIC 8230 has a private memory directly connected to the CPU bus and completely independent from the global resources. The CPU accepts all types of transfers (user, supervisor/data, programs) for accessing this memory.
The local memory is composed of:

a) Static RAM memory bank
b) EPROM memory bank

2.2.1. Static RAM Memory

The local static RAM memory is composed of four 32K x 8 memory chips in SOP package. This memory (RAM) allows zero wait state CPU and DMAC access without being disturbed by concurrent VME slave access to the FIC 8230.

This memory is designed to store the operating system kernel, frequently used tasks, and all programs needing maximum undisturbed execution speed.

The local RAM memory is organized as follows:

LOCAL SRAM

Size : 32K x 32-bit (128 Kbytes)
t_{acc} : depending on the CPU clock
base address : $ 0800'0000

The memory is redundant every 128 Kbyte boundary.

2.2.2. EPROM Memory

The EPROM memory is composed of two 27C512 chips. These sockets are standard 28 pin JEDEC, but accept only 27C512 memory. This memory area holds the firmware debugger, initialization procedure and/or OEM firmware application, pSOS, pROBE, pRISM, etc.

Access to the EPROM area is over a 16-bit data path with 2 CPU wait cycles.

To improve system performance, the operating system kernel should be copied to the local SRAM before execution.

EPROM address $Ø contains the restart address. Accordingly, it must always be installed whatever the application.

Depending on the processor speed, the EPROM must satisfy these access time requirements.

EPROM SPEED VERSION

16 Mhz CPU : tacc = 200 [ns]
24 Mhz CPU : tacc = 170 [ns]

The EPROM sockets are located close to the CPU and DMAC controller. The byte allocation (low and high) is as follows:

low byte : close to the DMAC, bottom socket
high byte : close to the CPU, top socket

Remarks: 1) Normally the EPROM contains the following firmware:
- start address + initial stack address
- bootstrap
- firmware monitor

2) The write accesses to the EPROM area are accepted as read cycles and acknowledged with DTACKx.

2.3. MC 68882 Floating Point Unit

The FIC 8230 includes a socket for the MC 68881 floating point coprocessor in order to enhance the system's performance when used in floating point applications, such as in a system running complicated tasks. The 68881 uses the MC 68020 coprocessor interface to provide an execution model transparent logical extension to the MC 68020's instruction set. The 68020 can accommodate 8 coprocessor types in one system via the coprocessor ID bit field, but space is provided for just one MC 68881 on the FIC 8230.

The MC 68881 chip select is decoded from the function codes FCØ-FC2 and the logical addresses LA19-LA16, LA13 all directly from the 68020 CPU.

The coprocessor ID is fixed as ØØØ1 for the FPU 68882.

A status bit in the register "STACPU" signals the presence of the MC 68881 in its socket.

As the FPU 68881 uses the same clock as the CPU, a chip with correctly matched speed must be used.

FPU 68881 -> Coprocessor ID = 1

2.4. DMA Controller WE 32104

The FIC 8230 contains a DMA controller supporting data transfers across the 4 Gbytes addressing range, in 16 and 32-bit data format. This DMAC is able to perform high speed data transfers between memory arrays in the system. This DMAC controller was initially designed for the AT&T 32100/32200 processor family. For a complete description, please refer to the manufacturer's data sheet.

2.4.1. Chip Description

The WE 32104 DMA controller is a memory-mapped peripheral device that performs memory-to-memory, memory-to-peripheral and peripheral-to-memory data transfers very quickly and efficiently. The DMAC contains specialized hardware that allows data transfers at a rate much faster than a typical CPU. In addition, a peripheral bus (ported to the front panel general purpose connector) is provided to couple 8-bit input/output devices to the system bus. The DMAC is implemented in CMOS technology.

Features:

- Full 32-bit address and data busses.
- 8-bit peripheral bus for coupling I/O devices to the system bus.
- Word (32), double word (64 bits) and quad word bus cycles for high system throughput.
- Four independent DMA signals.
- Fixed priority level for each DMA channel.
- Two programmable interrupt vectors per channel.
- Memory fill operations available for writing an arbitrary constant to a block of memory.

2.4.2. DMA Hardware Implementation

The DMAC WE 32104 is directly connected to the CPU bus. The DMAC WE 32204 becomes master of the CPU bus by requesting it directly to the MC 68020. This request procedure uses the original MC 68020 logic with the 3-wire handshake (BR/BG/BGACK). This implementation allows the DMAC to access (in read & write) the full 4 Gbytes addressing range.

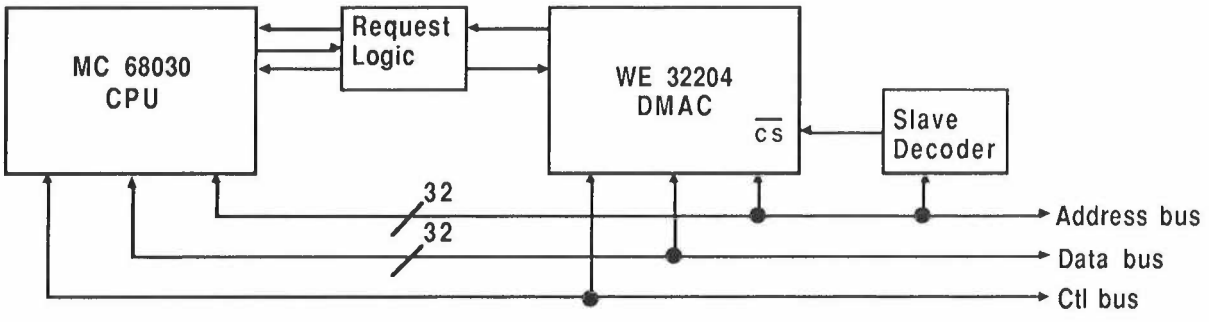


Fig. 2.1. DMAC Implementation

2.4.3. Internal Registers Access (WE 32104)

The DMAC has four independent channels which allow it to service four unrelated transfer requests simultaneously. Each channel has a set of registers that configures and controls its operation. In addition, there is one mask register which is shared between the four channels. The DMAC registers are mapped into the processor's address space. When the DMAC is in the peripheral mode, registers within the DMAC and registers within devices connected to the peripheral bus are accessed by the CPU. This provides the CPU with an access path to the peripheral bus. The DMAC requires a 2 Kbyte address space which is divided into four sections. Sections are provided for the DMAC channel registers, global registers and internal data buffers, and for a programmed input/output path to devices connected on the peripheral bus.

The description of the registers and their internal mapping are explained in detail in the DMAC 32104 data sheet.

The DMAC 32104 internal registers are mapped with the local resources as follows:

DMAC registers base address : $ 1800'0000

Consult the "FIC 8230 parameter definition file" listing in Chapter 8.

2.4.4. DMA Data Transfer Memory-to-Memory

The DMA data transfers between different memory areas must be done with a specific data size. The DMAC data transfer is encoded as follows:

byte : 8-bit
half-word : 16-bit
word : 32-bit
double word : 2 x 32-bit
quad word : 4 x 32-bit

The transfers supported by the hardware are:

	Byte	Half Word	Word	Double Word	Quad Word
Local SRAM	Y	Y	Y	Y	N
Global SRAM	Y	Y	Y	Y	YB
VSB	N	N	Y	Y	YB
VME D16	Y	Y	N	N	N
VME D32	Y	Y	Y	Y	YB

N = not supported.
Y = supported in cycle by cycle.
YB = supported in sequential access.

During memory-to-memory DMA, the source and destination address field must support the data size specified in the DMA CMR register. In case of contravention the SIZERROR logic will abort the current DMAC.

2.4.5. External Options for WE 32104

Two control bits located in the "CTLCPU" register control the following options:

DMA CTL1: This bit <2> allows to condition the procedure on a BERR received from the slave memory during a DMA transfer.
if = Ø: BERR causes the abort of the DMA (normal BERR)
if = 1: BERR causes the automatic retry of the last DMA cycle four times, if by the fourth try the DMA is aborted with a FAULT.

DMA CTL2:	This bit <3> allows the choice of data mode (user or supervisor) during the following DMA transfers:
if = Ø:	DMA transfers in "data user mode"
if = 1:	DMA transfers in "data supervisor mode"

2.4.6. WE 32204 Peripheral Bus

The DMAC WE 32204 has a private 8-bit peripheral bus. This 8-bit bus can be used to connect 8-bit peripherals as well as disk controllers, LAN interfaces, etc. This feature is implemented to add special requirements for an application without adding glue logic between the DMAC and the peripheral. This user friendly bus interface can be configured by software to work in:

1) asynchronous mode
2) synchronous mode with automatic insertion of wait cycle.

Four peripheral devices can be connected to the bus, and they can each work with independent DMA REQ/DONE logic.

The DMAC 32104 uses efficient data packing/unpacking to transfer data between the peripheral (8-bit) and the system bus (32-bit).

The peripheral bus is wired to the front panel general purpose connector with additional + 5V power pins.

It is composed of:

PERADDØ-PERADD4	: peripheral address bus	(5)
PERDØ-PERD7	: peripheral data bus	(8)
PR/W	: peripheral Read/Write	(1)
PCSØ-PCS3	: peripheral device chip select	(4)
PDS	: peripheral bus data strobe	(1)
PDACK	: peripheral bus data acknowledge	(1)
PBRØ-PBR3	: peripheral bus DMA request	(4)
PDONE	: peripheral bus done	(1)

(Refer to paragraph 7.3. "General 64-pin connector ")

Before implementing a device on this bus, carefully consult the AT&T documentation or contact CES in Geneva.

2.4.7. Block Mode Support

When the DMAC is initialized with a Quad-Word data size, the DMAC uses a block transfer to move the operands. This allows to transfer data in a more efficient manner. Depending on the address field, the block transfer is supported as follows:

Local memory	:	not supported -> do not use quad-word data size
Global memory	:	supported
VME access std/ext	:	generated with the corresponding VME AM code
VSB access	:	supported

* Refer to Chapter 3, paragraph 3.4.3.2.

2.5. Serial Controller (Z8030)

The Zilog Z8030A chip is used as a serial communication controller. Its primary use is to allow a direct terminal and host connection for control and debugging. Only the asynchronous protocols are implemented on board.

2.5.1. Chip Description

The Z8030 Z-SCC serial communication controller is a dual channel, multi-protocol data communication peripheral primarily designed for use with the Zilog Z-bus. The Z-SCC functions as a serial to parallel, parallel to serial converter/controller. The Z-SCC can be software configured to satisfy a wide variety of serial communication applications. The devices contain sophisticated functions including:

- on-chip baud generators
- break detection and generators

2.5.2. Hardware Implementation

The Z8030 Z-SCC is implemented on the FIC 8230 with a 5 Mhz clock. This parameter is important to set up the baud rate generator.

PCK_{8030} = 5 Mhz

The terminal and the host port are cabled to the front panel connector with only three signals per channel with RS232 levels. A terminal can be directly connected these connectors as follows:

pin 3 : Tx
pin 8 : Rx
pin 2,5,9,10 : Ground GND

If needed, the Z-SCC is able to generate interrupts to the CPU. This interrupt generation is cabled on level 3. An internal daisy-chain prioritizes the three Zilog peripherals during the IACK cycle. This daisy-chain sets the Z-SCC Z8030 to the lowest priority.

Int level Z8030 = 3

2.5.3. Z8030 I/O Mapping

Consult the listing of "FIC 8230 parameter definition file" in Chapter 8..

2.6. Timer, I/O Port & Interrupt Controller

Zilog's Z8036 Z-CIO chip is used as a general purpose interface for different parts of the FIC 8230. It is used for the following:

a) Interrupt controller for the communication FIFOs
b) General purpose timer
c) Front panel interrupt
d) Front panel parallel I/O port

2.6.1. Chip Description

The Z8036 Z-CIO Counter/Timer and parallel I/O element is a general purpose peripheral circuit satisfying most counter/timer and parallel I/O needs encountered in system designs. This versatile device contains three I/O ports and three counter/timers. Many software options tailor its configuration to specific applications. The use of the device is simplified by making all internal registers readable and writable. In addition, each register is given in its own unique address so that it can be accessed directly - No special sequential operations are required.

Features:

- 2 independent 8-bit double buffered bi-directional I/O ports.
- 4 handshake modes, including "3 wires".
- flexible pattern recognition logic, programmable as an interrupt controller.
- 3 independent timer/counters with external access.

2.6.2. Z8036 Hardware Implementation

The Z8036 Z-CIO is implemented on the FIC 8230 with a 5 Mhz clock. This parameter is important when the internal timers are used.

PCK_{8036} = 5 Mhz

The Z 8036 is used for three distinct applications in the FIC 8230.

a) Communication FIFO interrupt controller
b) General purpose Timer/Counters
c) Front panel I/O port

The Z8036 registers can be mapped in the byte boundary. This feature allows the user to access 16-bit registers by using instructions such as - MOVE.W

2.6.2.1. Communication FIFO Interrupt Controller

The Port A of the Z8036 Z-CIO is used as a general purpose interrupt controller. It allows interrupts to be generated on the following FIFO flag states:

FULL or Not FULL
EMPTY or Not Empty

For each of the four FIFOs, these flags are updated by the FIFO sequence logic at each FIFO access. Up to 8 different vectors can be generated on the FIFO conditions.

In this case, the Z8036#1 port A is set up as an "OR priority encoded vector mode".
Please refer to the software example in Chapter 8.

2.6.2.2. General Purpose Counter/Timers

The internal counter is used in operating systems for periodic interruptions. The three timers can be used in different modes with or without external triggering (via the front panel connectors).

2.6.2.3. Front Panel I/O Port

The port B and C are brought out to the front panel general purpose connectors (64 pins). These ports can be used in different manners, for example:

- I/O port with/without handshake
- Front panel interruption
- Control/status to/from the counter timers

The following pins are connected to the Z8036

GP Connectors 64 Pins

pin	51 ->Z8036	PCØ	}	
	53	PC1	}	Port C
	55	PC2	}	
	57	PC3	}	
pin	35->Z8036	PBØ	}	
	37	PB1	}	
	39	PB2	}	
	41	PB3	}	Port B [with on-board pull-up resistors]
	43	PB4	}	
	45	PB5	}	
	47	PB6	}	
	49	PB7	}	

NOTE: The even numbered pins (36, 38,....58) are grounded.
The port B I/Os are pulled up with a 4.7 kΩ resistor.

2.6.3. Interrupt Logic

The Z8036 is able to generate interrupts to the 68020 on level 3. It is the highest priority device at this level. The INT pin must be initialized in "OPEN DRAIN" and active low.

For further details, please refer to Chapter 1.10. "Interrupt Structure".

INT level Z8036 = 3

2.6.4. Z8036 I/O Mapping

If the RJA bit in the MIR register is set to 1, the address references are byte-aligned.
Consult "FIC 8230 parameter definition file" in Chapter 8.

2.7. DP8570 Real-Time Clock

The DP8570 is used in the FIC 8230 to provide four distinct functions:

a) Real-time clock calendar
b) Oscillator for Time-out logic
c) Power fail detection
d) Memory with battery back-up

2.7.1. Chip Description

The DP8570 is a multi-function device designed with low power CMOS technology. It is composed of 2 X 10 Mhz 16-bit multi-mode timers, a real-time clock with its own oscillator (32.768 khz), a power failure detector and a 44 byte CMOS RAM with battery back-up.

The DP8570 has a sophisticated interrupt structure, which provides for four basic types of interrupts: Periodic, Alarm/Compare, Timer and Power Fail interrupts. An interrupt mask and status register enables the masking and easy determination of each interrupt.

2.7.2. Hardware Implementation

The DP8570 is implemented in the FIC 8230 with the following features:

The TCK (Timer clock) is set at 10 Mhz (whatever the CPU clock) and the real-time clock is set with a 32.768 khz quartz.

TCK_{8570} = 10 Mhz
RTC_{osc} = 32.768 khz

Timer #1 is used (via its T1 output pin) as the clock generator for the time-out logic. This timer must only be used for this purpose. The timer is used as a divider to generate the correct frequency for the desired time-out value.

time-out = (1÷ fT1) X 16

The power-fail logic is trimmed to trigger at 4.2 (v) typ. This high value allows enough time to save some important parameters, in the battery backed-up memory.

U_{pfail} = 4.2 v (typ)

The battery assures a ten-year retention, without data loss. This battery must be leaded (against oxidation) and changed when the DP8570 signals the end of the battery.

Battery type: Renata 338A

The interrupt generation is given by the "INT" pin. This must be initialized in "active low" and "push-pull". All the DP8570 interrupts must use this pin (the MFO output pin is not used). The DP8570 interrupts the CPU at level 6 This high level was chosen because of the power-fail logic detection (see Chapter 1.10 "Interrupt Structure"). The memory locations 1D, 1E and 1F are reserved for CES.

INT level 8570 = 6

2.7.3. DP8570AI/O Mapping

The 32 bytes of the DP8570 are mapped as follows:

register ØØ : MSR = $ 1500'0000
register Ø1 : ... = $ 1500'0001
...
register 1F : = $1500'001F

2.8. SCB68154 VME Interrupter

The VME interrupter allows the generation of interrupts on the VME. This feature is useful in a system where it is not possible to implement the "message passing" structure. This logic is implemented with a Signetics SCB68154 device.

2.8.1. Chip Description

The Signetics SCB68154 interrupt generator provides an interface between an interrupting device and the VME system bus.
The SCB68154 has three primary functions:

1) Generates bus interrupt requests.
2) Resides in the VME interrupt daisy-chain.
3) Allows a status/ID byte to be supplied to the interrupt handler if needed.

The SCB68154 has seven interrupt request levels, which are selected by using the interrupt request register. The local master writes to the interrupt request register to generate an interrupt request on any interrupt request level. The interrupt request register may be read to determine if an interrupt has been acknowledged.

For additional details, please refer to the SCB68154 data sheet.

2.8.2. Hardware Implementation

The SCB68154 is physically implemented on the local bus. The CPU 68030 can access its two registers via memory locations as follows:

RØ: Int vector register = $ 1700'0000
R1 : Int request register = $ 1700'0002

The vector output bus is independently connected to the VME bus and all control signals are independent of the VME Master & Slave Interfaces (this means that the FIC 8230 can generate a VME interrupt to itself and acknowledge it, for test purposes).

2.9. SCB 68155 Interrupt Handler

(Consult paragraph 1.10)

Chapter 3

GLOBAL RESOURCES

3.1. General Philosophy

The global resources form 3 different groups:

- Global memory
- Communication FIFOs
- Control and Status registers

These resources are accessible from the CPU and the DMAC as well as through the VME slave port. Access from the CPU is supported in BYTE, WORD and LWORD modes; VME access supports D16 and D32. The global resources are mapped as follows:

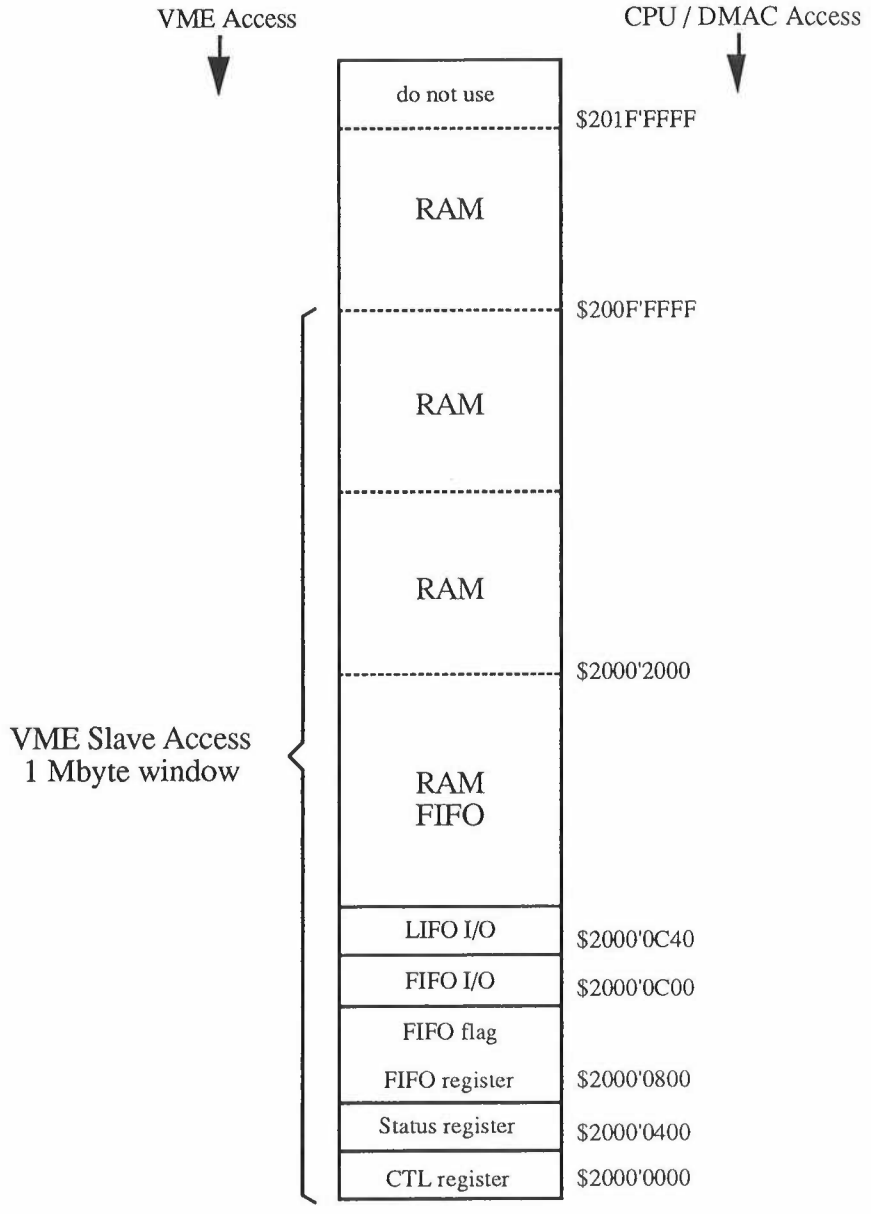


Fig 3.1 : Global Resources Mapping

3.1.1. CPU/DMAC Access

Access from the CPU to the global resources is mapped from the address $2000'0000 to $201F'FFFF. The CPU can access the memory (global RAM) in BYTE, WORD and LWORD modes in both user/supervisor and data/program mode.

CPU and DMAC Mapping

$ 2000'0000 - $ 2000'03FF : Control registers
$ 2000'0400 - $ 2000'07FF : Status registers
$ 2000'0800 - $ 2000'0BFF : FIFO ctl/status registers
$ 2000'0C00 - $ 2000'0EFF : FIFO I/O port
$ 2000'1000 - $ 2000'1FFF : FIFO memory
$ 2000'2000 - $ 201F'FFFF : RAM memory

The global resources are "redundant" modulo 2 Mbytes. This means that access to address $2000'0000 = $2020'0000 = $2040'0000 and so on.

3.1.2. VME Access to the Global Memory

access from the VME port to the global memory is through a 1 Mbyte WINDOW mapped in the VME addressing range. This mapping is described in Chapter 4 "VME Slave Port". This window provides access to the first Mbyte of the global memory. The VME port supports all VME data access, including RMW cycles and unaligned transfers (ULT).

Block transfers from VME or the internal DMAC WE32104 are supported in QUAD WORD.

3.1.3. Global Arbitration and Indivisible Cycle

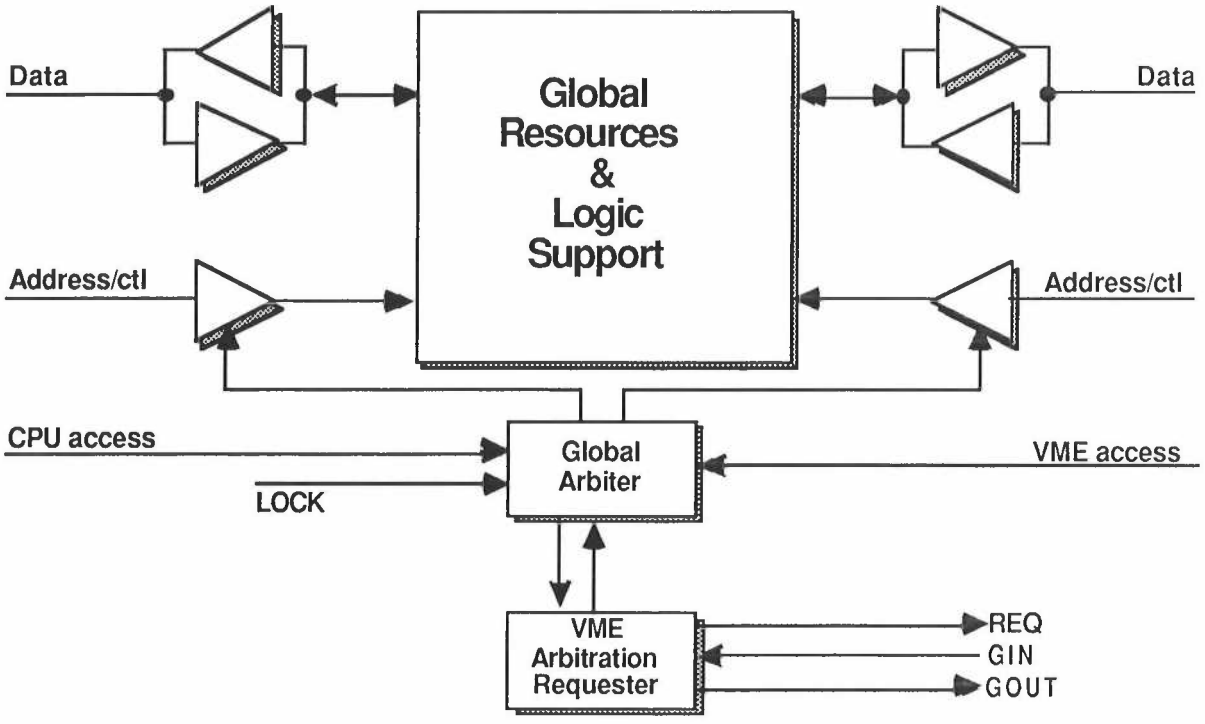


Fig 3.2 : Global Arbitration Resources Diagram

Access to the global resources is arbitrated by a "Global arbiter". This arbiter is designed to allow a "zero wait state" on the CPU port if no request is coming from the VME port. It works on a "release when done" mode on the VME port.

The arbitration on the two bus interfaces to the global resources (VME and CPU) supports the indivisible cycle as follows:

CPU ->	Global Memory	VME -> Global Memory
TAS		RMW cycle B, W, LW
CAS	aligned and non-aligned	
CAS2	aligned and non-aligned	

Note: Be careful that the VME RMW (Read-Modify-Write) cycles are executed correctly (AS* must stay active during the two Read and Write cycles independently of the BBSY* line).

3.2. Global Memory

The global memory is used as the main memory system. The FIC 8230 can be equipped with from 512 Kbytes (minimum) to 4 Mbytes (maximum).

The global memory is mounted in a special 68-pin socket. This socket can be equipped with the following:

1) OPT8230/1	: Static RAM	512 Kbytes capacity
3) OPT8232/6	: Dynamic RAM	4 Mbytes capacity
4) OPT8232/7	: Dynamic RAM	16 Mbytes capacity

The three memory modules are built using "Surface Mounted Device" (SMD) and CMOS technology for a higher density with a reasonable power consumption.

The first 8 Kbytes of global memory are reserved for a different use:

Memory reserved	: from CPU : $ 2000'0000	-> $ 2000'1FFE
	: from VME : $ xxx0'0000	-> $ xxx0'1FFE

The use of this area as standard memory is perfectly possible; however it must be done carefully and with full knowledge of the possible consequences.

Memory Size Options

	512 KB	1 MB
CPU Speed	1 X OPT8230/1	2 X OPT8230/1
16 Mhz	100 ns	100 ns
	Ø wait	Ø wait
24 MHz	70 ns	70 ns
	Ø wait	Ø wait
	4MB	16 MB
CPU Speed	1 X OPT8230/6	2 X OPT8232/9
16/24 MHz	100 ns	100 ns
	1 or 2 wait	1 or 2 wait

3.2.1. OPT8230/1 Static RAM Memory Module

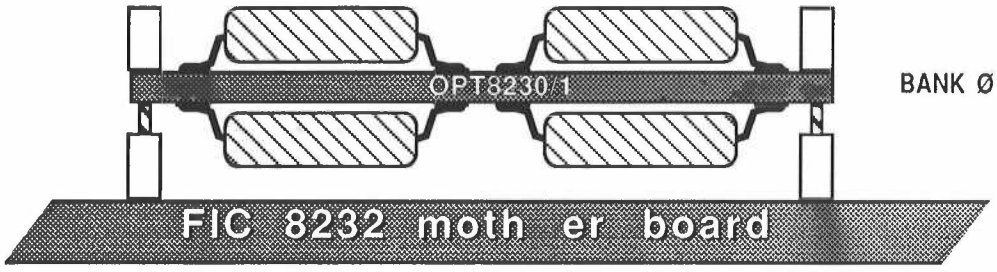
The OPT8230/1 static RAM memory module is built around CMOS static RAM chips in SO package (SMD). Each module is equipped with 16 memories (32K X 8) organized in 128 K X 32.

Depending on the CPU speed, the memory module must be equipped with the correct speed grade to guarantee a Ø wait state operation.

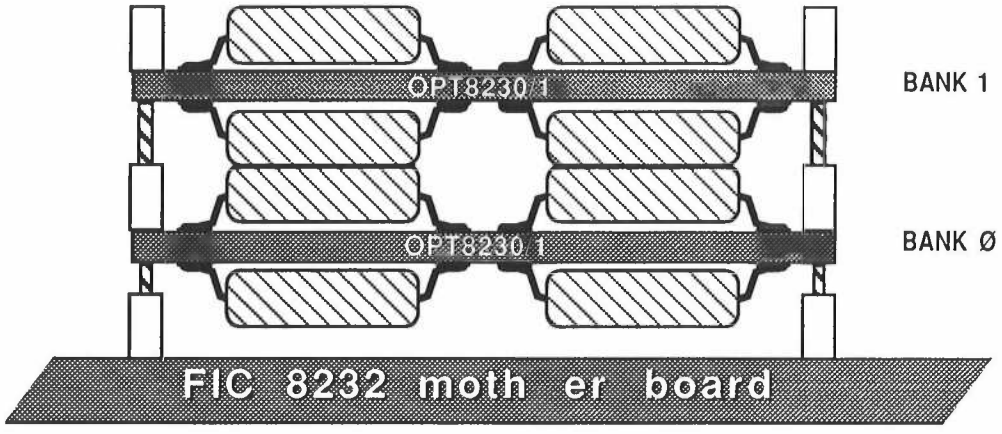
16 Mhz CPU : 100 ns
24 Mhz CPU : 70 ns

Two of these memories can be plugged in the memory socket of the FIC 8230. The two memory modules are assigned as BANKØ and BANK1.

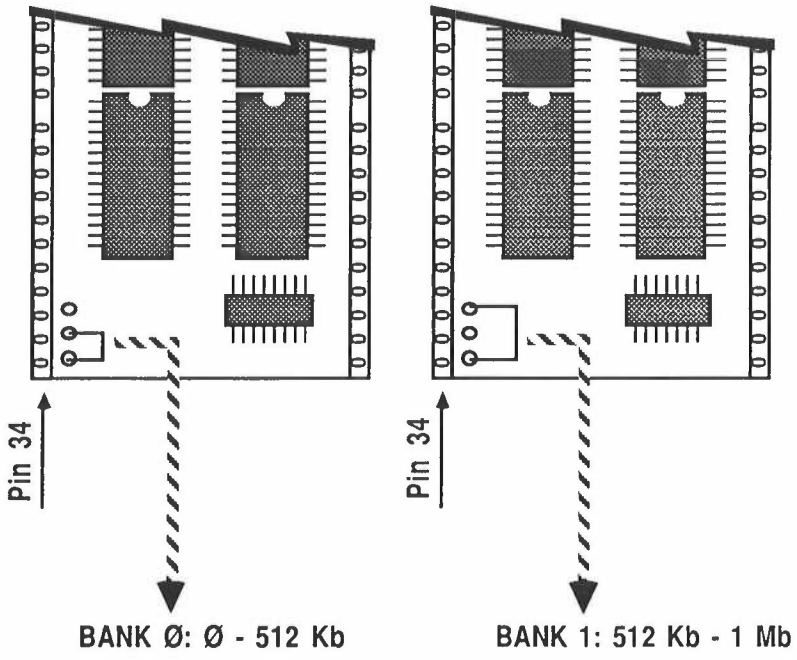
Option 512 K
1 x OPT8230/1 module



Option 1 Mega
2 x OPT8230/1 module



The OPT8230/1 memory module must be labeled as BANK Ø or BANK 1. The hardware setting allows the memory module to be mapped as the first or the second 512 Kbyte bank. (Address 0-512Kb or 512Kb-1Mb)



WARNING: The OPT8230/1 is packaged in 68-pin piggy-back modules so it must be inserted at the bottom of the 78-pin connector on the FIC8230 (Refer to paragraph 7.1 "FIC8232 Implementation").

Pin 34 of the OPT8230/1 ------> Pin 39 of the 78-pin FIC8232 connector

3.2.2. OPT8232/6 Dynamic RAM Memory Module

The OPT8232/6 dynamic RAM memory module is built from CMOS dynamic RAM chips (1 Mbit capacity) in SOJ package. The module is equipped with 32 memories (256K X 4) organized in 1M X 32 (4 Mbytes). The address multiplexing and refresh logic are implemented directly on the memory module. From a hardware point of view, the memory appears as a static RAM. The arbitration between the refresh cycle and the access (R/W) is implemented on the module, and an "ACK" acknowledge signal allows to delay the current cycle to reach the CPU/DMAC specification and the VME slave access.

Equipped with a 4 Mbyte dynamic RAM option, the CPU runs with a 1 wait-state.

WARNING: Only 1 OPT8232/6 memory module can be plugged on the FIC 8232.
The OPT8232/6 is packaged in 76-pin piggy-back modules so it must be inserted at the bottom of the 78-pin connector on the FIC8232 (Refer to paragraph 7.1 "FIC8232 Implementation").

Pin 38 of the OPT8230/6 ------> Pin 39 of the 78-pin FIC8232 connector

3.3. Control and Status Registers

The control and status registers contain the important parameters of the FIC 8230. These registers are accessible from both the CPU and VME. The control registers are implemented to replace the hardware options (links, jumper, wire-wrap).

Status registers are "read-only" whereas control registers are Read/Write. These registers are mapped at the base of the global resources area as described below:

$ 2000'0000	: CTLCPU	: control option CPU	(R/W)
$ 2000'0004	: CTLVSB	: control option VSB	(R/W)
$ 2000'0008	: CTLVME1	: control option VME port 1	(R/W)
$ 2000'000C	: CTLVME2	: ctl option VME port 2	(R/W)
$ 2000'0400	: STAVSB	: status general VSB	(R only)
$ 2000'0404	: STACPU	: status general CPU	(R only)
$ 2000'0408	: STAVME	: status general VME	(R only)
$ 2000'040C	: STAFIF	: status general communication FIFO	(R only)

The control/status registers form three different groups:

CPU : General information
VSB : Interface VSB master
VME : Interface bus VME, Master & Slave

The RAM redundant with these registers is masked by the hardware, and can not be used.

Access to the registers from the VME port is identical to the access to the global memory.

Access to the registers must be done in LWORD data from the CPU or correctly aligned byte access.

3.3.1. Control Registers

These registers contain the configuration options as well as defining operation modes of the system. At power on or during a RESET, all these registers are zeroed.

These registers are READ/WRITE, a copy of the register value is stored in the global RAM area.(when a register is written, the value is also written through into the global memory)

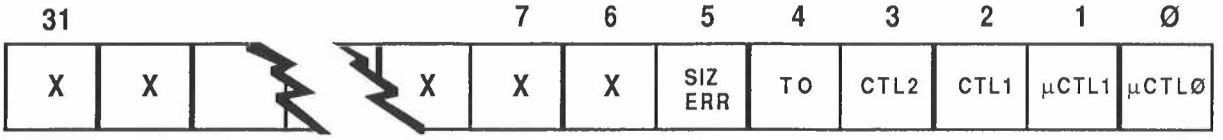
Consequently, at RESET or power on (registers set to Ø), the CPU must write Ø in the register's address to ensure that the RAM "shadow" is also zero.

Because these registers contain important parameters, the start-up value for an application can be stored in and recovered from the CMOS memory with a battery back-up in the DP8570A (see paragraph 2.7.).

3.3.1.1. CTLCPU - Adr = $2000'0000

This 6-bit register can be divided in three parts:

1) - CPU 68020 external control
2) - DMAC options
3) - Cycle exception flag set/reset



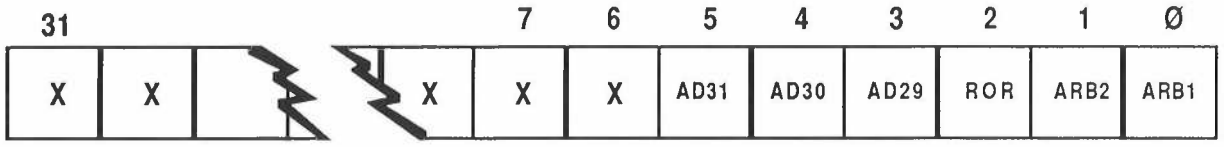
Bit Description:

<1, Ø>	µCTL1, µCTLØ	These two bits are used to control the state of the CPU from an external processor over the VME bus. The combinations of these two bits generate the following action: <Ø,Ø> : No action, or CPU HALT -> RUN. <Ø,1> : activates the RESET line. <1,Ø> : generates an ABORT. <1,1> : Put the CPU in the HALT state. These bits must be twiddled very carefully.
< 3,2 >	CTL2, CTL1	These two bits are used to select the option for the DMAC WE32104 operation. Bit <2> CTL1 allows the user to choose the action required in case of a BERR (or a time-out) during a cycle issued by the DMAC 32204. CTL1 = Ø : BERR causes an effective error (FAULT). CTL1 = 1 : BERR causes a RERUN of this cycle four times, before aborting the DMA with a FAULT. Bit <3> CTL2 allows the user to choose the priority level for the subsequent DMA transfers. CTL2 = Ø : User mode CTL2 = 1 : Supervisor mode
< 5,4 >	SIZERR TO	These two bits allow the user to SET/RESET the corresponding flag: Time-out and size error by writing the desired value (Ø or 1).

NOTE: During any modification to bits <5,4,3,2>, bits <1,Ø> must be set to "Ø", otherwise the action described for configuration of these bits will be executed immediately. (Whenever it is issued by the CPU itself)

3.3.1.2. CTLVSB - Adr = $2000'0004

This 6-bit register contains options for the VSB master port.



Bit Description:

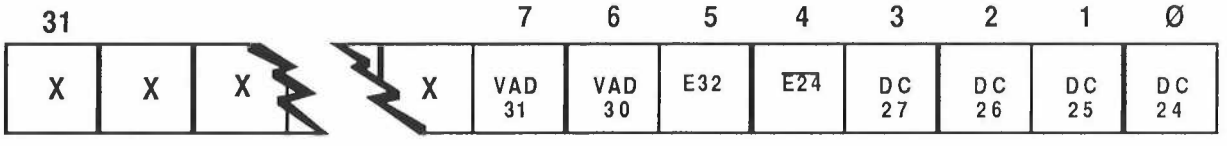
<1,Ø>	ARB2, ARB1	These two bits define the function of the VSB arbitration/requester. if <Ø,Ø> : FIC 8232 is the VSB arbiter if it is in VSB slot N^{O} Ø if <1,Ø> : FIC 8232 is the VSB arbiter wherever it is. if <Ø,1> : FIC 8232 is not the VSB arbiter.
<2>	ROR	This bit defines the release mode when the unit is VSB master. ROR = 1 : work in release when done. ROR = Ø : work in release on request.
<5,4,3>	AD31,30,29	These 3 bits map the higher address bit in the VSB cycles.

See Chapter 6 for additional explanations.

3.3.1.3. CTLVME1 and CTLVME2

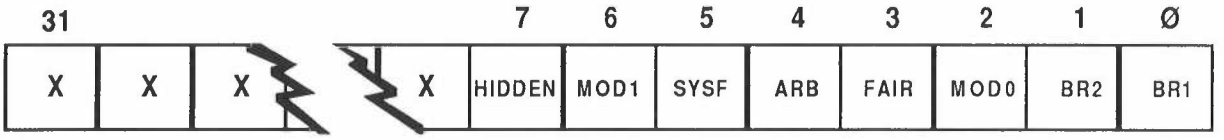
These two registers define options for the VME interfaces (Master & Slave).

CTLVME1 : Adr = $ 2000'0008
CTLVME2 : Adr = $ 2000'000C



CTLVME1

<3-Ø>	DC27,26,25,24	These 4 bits furnish part of the address to map the VME slave access in the VME extended addressing range (A32 : D16/D32).
<4>	/E24	This bit allows to enable/disable the VME slave access in VME standard addressing (A24: D16/D32). /E24 = Ø : A24 enabled /E24 = 1 : A24 disabled
<5>	E32	This bit allows us to enable/disable the VME slave access in VME extended addressing (A32: D16/D32). E32 = Ø : A 32 disabled E32 = 1 : A32 enabled
<7,6>	VAD31, VAD30	These 2 bits allow the user to generate addresses A31 & A30 when the CPU accesses VME in extended addressing.



CTLVME2

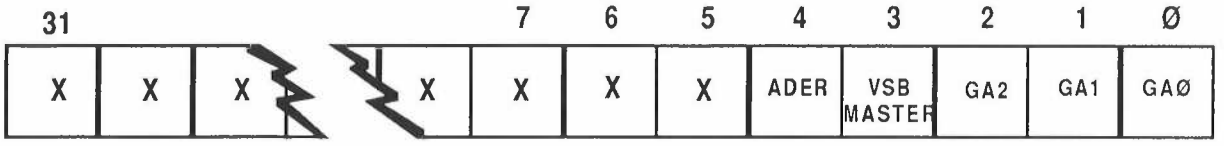
<1,0>	BR2,BR1	These 2 bits define the level of BREQ used during the VME request arbitration. <Ø,Ø> ---> BREQ level 3 <Ø,1> ---> BREQ level 2 <1,Ø> ---> BREQ level 1 <1,1> ---> BREQ level Ø
<2>	ROR	These two bits define the mode of release during the VME request arbitration. <Ø> ---> RWD mode <1> ---> ROR mode
<3,4>	SP1, SP2	Bits reserved for future distributed arbitration.
<5>	SYSF	This bit allows to generate the VME "SYSFAIL" line. <1> : VME SYSFAIL active <Ø> : VME SYSFAIL inactive

For a complete explanation of these control bits, refer to the corresponding chapter: Chapter 4 for VME slave and Chapter 5 for VME master.

3.3.2. Status Registers

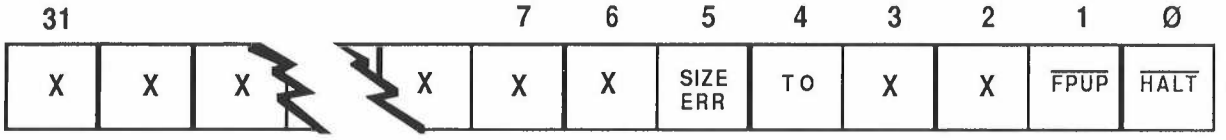
The four status registers give the user the important statuses of the FIC 8230. These registers are "Read Only" and can be accessed from both the CPU and the VME slave port.

3.3.2.1. STAVSB - Adr = $2000'0400



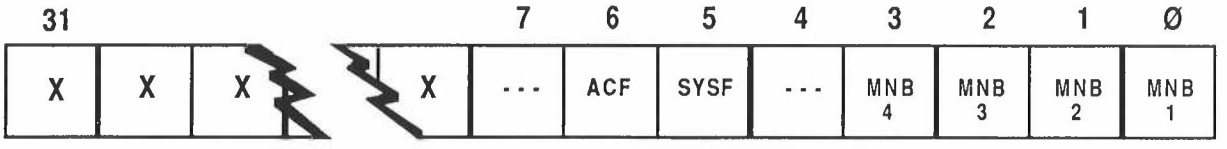
<2,1,Ø>	GA2,1,Ø	These three bits contain the number of the VSB slot into which the FIC 8230 is connected.
<3>	VSBM	VSB Master. This bit is active when the FIC 8232 unit is VSB master. (BUSY = 1)
<4>	ADER	VSB Address error. This bit is active if during the last VSB cycle DSACK 1 & Ø were = ØØ when AC -> 1. It is reset at the beginning of each VSB cycle.

3.3.2.2. STACPU - Adr = $2000'0404



<Ø>	/HALT	This bit is active (=Ø) when the 68020 is in the halt state.
<1>	/FPUP	This bit is active (=Ø) when the 68881 is present in its socket.
<2,3>	not used	
<4>	TO	Time-out flag. This bit is active when a time-out has occurred during a past cycle.This bit is used to define the source of a BERR. It can be reset with the CTLCPU register (See Chapter 3.3.1.1).
<5>	SIZERR	Size error during a DMA WE 32104. (See Chapter 2.3.4)

3.3.2.3. STAVME - Adr = $2000'0408

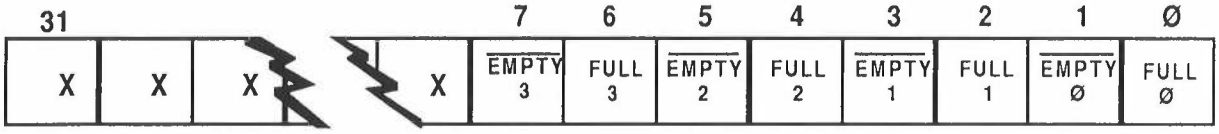


This register contains status relative to the VME interface (Master/Slave).

<3-Ø>	MNB,3,2,1,Ø	Master Number. These bits reflect the position of the front panel rotary switch defining the master number.
<5>	SYSF	Sysfail VME. Shows status of the VME SYSFAIL line. SYSF = 1 ---> VME SYSFAIL active.

3.3.2.4. STAFIF - Adr = $2000'040C

These 8-bit registers allow to read the "Empty and Full" flags of the four communication FIFOs. These 8 bits are also accessible via the Z8036 interrupt controller. See paragraph 3.4.



<Ø,2,4,6>	FULL # X	These bits show when FIFO # X is full.
		1 = FULL
		Ø = not FULL

<1,3,5,7>	EMPT # X	These bits show when FIFO # X is empty.
		1 = not EMPTY
		Ø = EMPTY

3.4. FIFO/LIFO Communication

Four communication FIFOs are implemented in the global resources. These FIFOs are accessible from both the CPU and the VME slave port in READ and WRITE. Each FIFO is 255 words deep and 32 bits wide and can work both in first-in first-out (FIFO)and last-in first-out (LIFO) modes.

3.4.1. FIFO/LIFO Introduction

The FIFOs are provided in the FIC 8230 to solve the problems of interprocessor interrupts in a multi-processor VME environment. The FIFO structure allows the user to implement efficient "message passing" between several processors, without limitation in the number of processors.

The depth of 255 words offers a sufficiently large buffer to run in a real-time environment, without spending time in retries or reservation procedures. The 32-bit word is wide enough to incorporate the source signature, the type of interrupt and other real-time parameters in a one word message.

When a CPU (local or external VME unit) writes using LIFO addressing, the word is stocked at the beginning of the FIFO. This feature allows to send "high priority messages" to the FIC 8232, bypassing the preceding messages stocked in the queue. In addition, if only the LIFO addresses are used, the FIFO/LIFO works as a stack.

Drivers can be made of FIFOs in multi-processing or in distributed processing.
For example:

- message passing
- virtual interrupts
- resource sharing
- task calling sequence

The four FIFOs of the FIC 8230 can be used individually. This allows to define priority in queue (for message passing) or to define a FIFO for message passing and another one for data buffer reservation. Each FIFO is able to interrupt the processor (CPU) with its two individual flags: FULL and EMPTY. Those 8 sources of interrupt (2 per FIFO) have to be supported by the program or by the operating system.

The FIFO data buffers use the global memory ($2000'1000 ---> $2000'1FFE). This allows access to the FIFO data buffers directly as a standard memory without passing through the FIFO/LIFO registers.

In case of error conditions (e.g. Read FIFO if it is empty or write FIFO if it is full), the FIFO control logic will generate a BERR. This last condition will be treated in the appropriate manner (e.g. Retry last cycle).

The FIFO structure allows to solve a lot of multi-processing problems in a most efficient way.

3.4.2. FIFO/LIFO Structure

The FIFOs are implemented in the FIC 8230 by using a fast state machine which controls the FIFO and the LIFO processing in a section of the global memory.

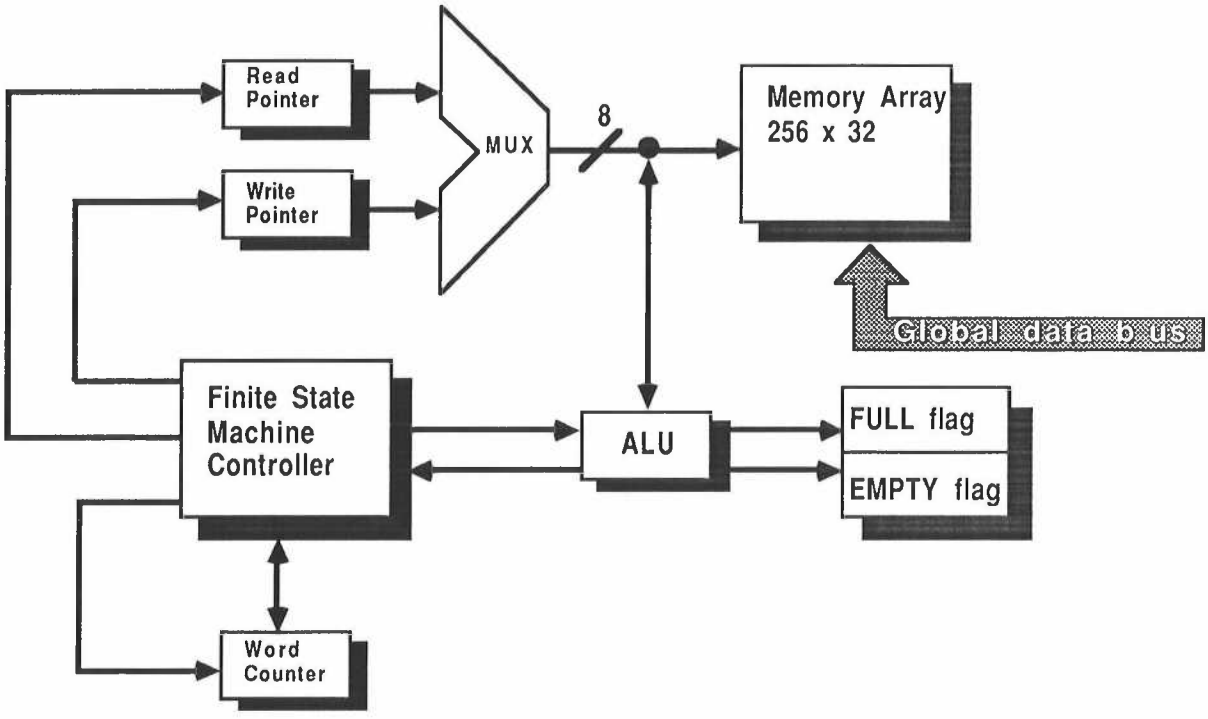


Fig 3.3 : FIFO Logic Block Diagram

Each FIFO/LIFO is controlled by three registers (which are also directly accessible as a memory location).

- Write pointer WRPT#X
- Read pointer RDPT# X
- Word counter WCNT# X

Each time a word is written in the FIFO, the word counter is incremented, the data is written in the memory location pointed by the WRPT and this register is incremented by 1. The FIFO flags are also updated at each FIFO access.

If the FIFO is accessed via its LIFO address, the data is stored in the memory location pointed by the RDPT-1. The WCNT and the flags are also updated.

Access to the FIFO/LIFO is independent of the data format (D16, D32).
Access to the FIFO/LIFO is done via FIFO # X (or LIFO # X) location as Read or Write.

WRITE causes a data word to be written into the FIFO# X
READ causes a data word to be read from the FIFO# X

3.4.3. FIFO/LIFO Control Registers

The FIFO/LIFO logic contains 4 register spaces:

1. Memory buffer (R/W)
2. FIFO/LIFO registers (RW)
3. Flag registers (Read only or Write only)
4. FIFO/LIFO I/O registers (R/W)

3.4.3.1. Memory Buffer

Global memory is used for the FIFO/LIFO memory buffers. This part of the global memory can be used as a standard memory or accessed directly for FIFO/LIFO fast reading or for FIFO/LIFO simulation.

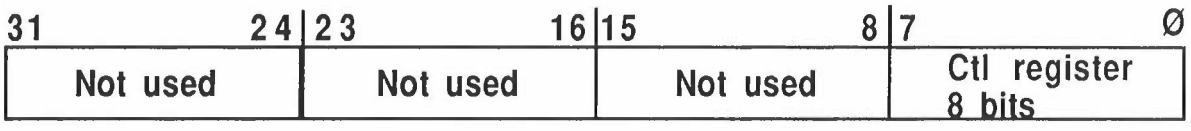
memory buffer FIFO#Ø : $2000'1000 --> $2000'13FF
FIFO#1 : $2000'1400 --> $2000'17FF
FIFO#2 : $2000'1800 --> $2000'1BFF
FIFO#3 : $2000'1C00 --> $2000'1FFF

3.4.3.2. FIFO Control Registers

These registers allow the FIFO/LIFO sequencer to control the FIFO memory buffer. They are also accessible by the CPU/VME slave for external control or for start-up initialization.

These registers must be reset during the start-up initialization to assure correct operating.

The registers are 8 bit wide and Read/Write.



The control registers are stored in an individual static RAM memory.
The control registers are mapped as follows:

Write pointers:

WRPT#Ø : $2000'0810
WRPT#1 : $2000'0814
WRPT#2 : $2000'0818
WRPT#3 : $2000'081C

Read pointers:

RDPT#Ø : $2000'0820
RDPT#1 : $2000'0824
RDPT#2 : $2000'0828
RDPT#3 : $2000'082C

Word counters:

WCNT#Ø : $2000'0830
WCNT#1 : $2000'0834
WCNT#2 : $2000'0838
WCNT#3 : $2000'083C

3.4.3.3. Flag Registers

The flag registers are composed of 5 register addresses. One of the registers (read-only) allows to read the 8 flags of the FIFOs. This "STATIF' register is described in paragraph 3.3.2.4.

The other four registers allow to modify each flag for each FIFO with a write cycle.

Adr = FIF_FLAG #X DØØ ---> FULL flag
DØ1 ---> EMPTY flag

Flag registers:

SET 1

FIF_FLAG#Ø : $2000'0840
FIF_FLAG#1 : $2000'0844
FIF_FLAG#2 : $2000'0848
FIF_FLAG#3 : $2000'084C

3.4.3.4. FIFO/LIFO I/O Registers

These registers control access to the FIFO/LIFO. They can be accessed in read or write and all data formats (byte, words, Lwords). However, each cycle will always be a 32-bit access.

Block transfers are only supported on these registers if SET 2 FIFO is selected on the FIC8232 (Refer to paragraph 7.4 "Fifo Addresses SET1/SET2 selection").

The LIFO address can only be used in Write mode, Read is always done in FIFO mode.

FIFO#Ø : $2000'0C00
FIFO#1 : $2000'0C04
FIFO#2 : $2000'0C08
FIFO#3 : $2000'0C0C

LIFO#Ø : $2000'0C40
LIFO#1 : $2000'0C44
LIFO#2 : $2000'0C48
LIFO#3 : $2000'0C4C

3.4.4. Interrupt Generation on the FIFO/LIFO Status

Each FIFO/LIFO generates 2 flags marked FULL and EMPTY. The combination of these 2 flags gives the following status:

FULL	EMPTY	FIFO Status
Ø	Ø	The FIFO is empty (WCNT = Ø)
Ø	1	The FIFO is not empty 0 < WCNT<255
1	Ø	Invalid state, normally not used in standard applications.
1	1	The FIFO is full WCNT = 255

Depending on the application, the interrupt must be generated if :

- at least one word is stored in the FIFO/LIFO buffer
- a FIFO/LIFO is Empty
- a FIFO/LIFO is Full

The interrupt generation on a FIFO/LIFO is executed via A Zilog Z8036 interrupt controller. The 8 flags are directly connected to port A of the Z8036#1 and Z8036#2.

To work correctly, the Z8036#1 or Z8036#2 have to be initialized as an interrupt controller. This is done by defining the following condition on port A :

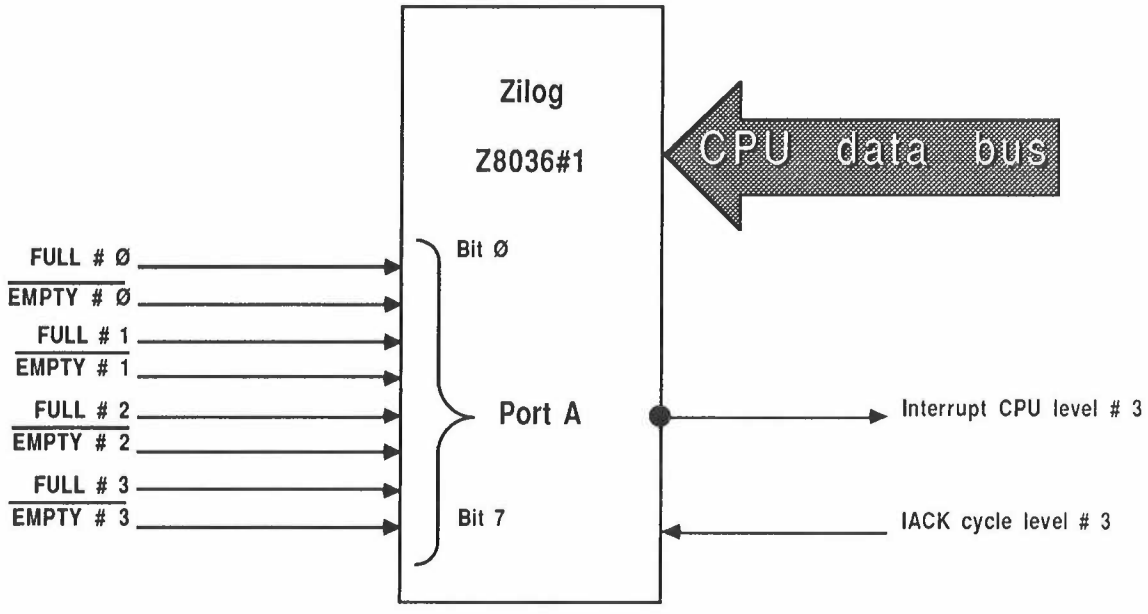
- input port with "OR" priority encoded vector mode
- correct polarity on each flag (depending on the application)

The interrupt condition can be set for each flag of each FIFO.

Each flag condition is associated directly to a vector number.

For more details on the use of this circuit, refer to the Zilog Z8036 data sheet.

See also the Software example in Chapter 8.



3.4.5. FIFO/LIFO Initialization

After a Reset or at power on, the FIFOs must be initialized as follows:

1) write $Ø into WRPT#X, RDPT#X, WCNT#X
2) set the FULL#X & EMPTY#X flags to Ø
3) Initialize Z8036 for interruption on the FIFO flags

Chapter 4

VME SLAVE INTERFACE

4.1. General Description

The VME slave port allows to map the global resources in the VME address space. the window size is fixed at 1 Mbyte whatever the addressing type or the memory present in the global resources.

The mapping is done by decoding :

1) the accepted AM codes (fixed in a PAL)
2) bits in the register CTLVME1
4) the Master No. (Rotary switch)

The FIC 8230 can map its global resources in 2 different addressing ranges in VME. In standard access: 24 address bit and in extended access: 32 address bits. They both can be enabled or disabled under program control by using E32 & E24 control bits in the CTLVME1 register.

The "Master N°" is the only static option (hardware selected) of the FIC 8230. It defines one of 16 1 Mbyte address ranges through which the FIC 8230 global resources are accessible from VME.

This parameter can also be read by the CPU in the STACPU register (Useful for signature in the "message passing" interrupt). This parameter is also displayed on the front panel hexadecimal rotary switch.

The decoding in VME is done by using:

1) Enable/disable addressing control bits /E24 & E32
2) The AM code
3) The high addresses corresponding for A24 & A32

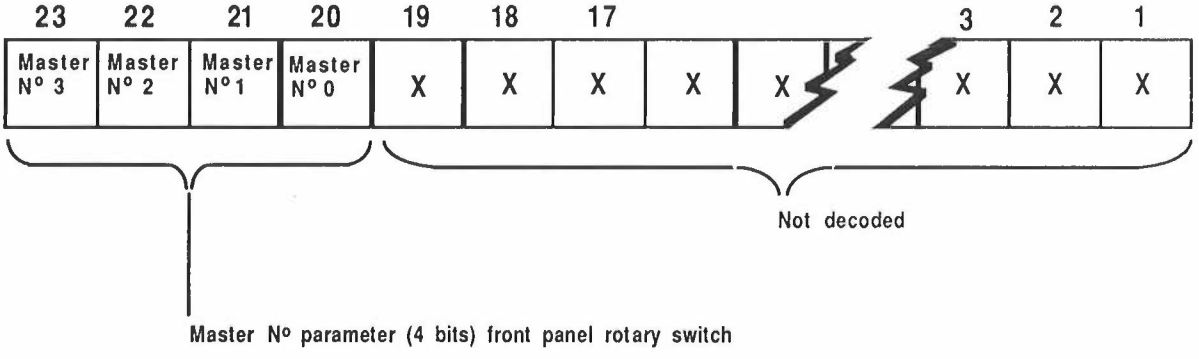
4.2. AM Decoding

The VME address modifiers are decoded by a PAL. This programmable array logic can be reconfigured by the user. This PAL is set to decode the following AM codes:

AM =	$39, S3D	: Standard access	A24
AM =	$9, $D	: Extended access	A32
AM =	$3B, $3F, $B, $F	: Block transfer	A24 & A32

4.3. Standard Access A24 - D16/D32

Standard access maps the global resources into a 1 Mbyte region in the 16 Mbytes VME addressing range. The higher address bits are furnished by the "Master No "Code.



The /E24 bit (VMECTL1 register) is used to enable or disable this type of decoding:

Ø = enabled
1 = disabled

WARNING: This bit is defined in negative logic. This allows to to enable this decoding after a Reset or after Power on.

Examples:

1) if Master N° = $4 and global resources size = 1 Mbyte and /E24 = Ø and AM = $3D or $39
----------> VME addresses decoded = $40'0000 - $4F'FFFF

2) if Master N° = $A and global resources size = 2 Mbytes and /E24 = Ø and AM = $3D or $39
----------> VME addresses decoded = $A0'0000 - $BF'FFFF

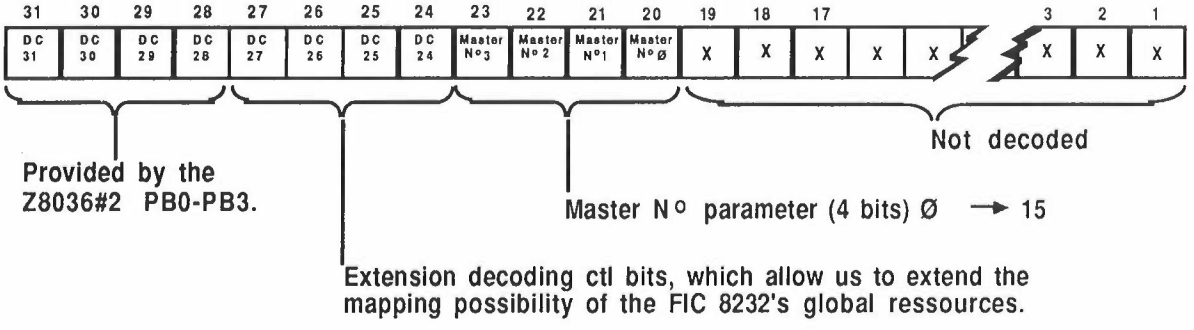
Bit /E24 is used to enable or disable this type of decoding:
Ø = enabled
1 = disabled

WARNING: This bit is defined in negative logic . This allows to enable this decoding after a Reset or Power on.

4.4. Extended Access A32 - D16/D32

Extended access maps the global resources into 1 Mbyte in the 4 Gbyte VME addressing range. The higher address bits (A31 --> A20) are furnished by using :

1) Master N° code
2) Fixed value
3) 4 programmable bits in the VMECTL1 register



This decoding allows to map the 1 Mbyte window into the first 256 Mbytes VME extended addressing range. The decoding is enabled if the "E32" control bit is active (this bit is set to Ø during a Reset or Power on). It must be set by the CPU or via the VME slave port (A24) after initialization of control bits DC27, 26, 25, 24. These parameters can also be dynamically modified under software control.

WARNING: The control bits /E24 and E32 can also be written from the VME slave port. Be careful not to disable both address decoding, otherwise the FIC 823 global resources will not be accessible via the VME port (the FIC would be completely disconnected from VME).

4.5. Data Type & Timing

The FIC 8230 VME slave port supports all VME data types, whatever addressing range is used. The internal swap logic aligns the word transfer in the 32-bit global memory.

Access can be done in:

Byte	D08	R, W, RMW
Word	D16	R, W, RMW
Lword	D32	R, W, RMW
Unaligned	Transfer	(ULT) R, W

The timing of a VME slave access to the FIC 8230 global resources depends on the following parameters:

1) The memory or FIFO access time
2) Any simultaneous CPU or DMAC access
3) The refresh cycle (only with DRAMs)

CPU or DMAC simultaneous or pending access to global resources delays VME slave cycles. This delay varies depending on the type of access. (read/write or indivisible cycle)

4.6. READ MODIFY WRITE Cycle

The global resources of the FIC 8232 support the READ MODIFY WRITE cycle in standard VME specifications.

4.7. Block Mode Support

The FIC 8230 VME slave interface supports VME block transfer capability with some limitations. The AM codes decoded for this feature are the same as the VME specifications.

This feature allows a higher bandwidth in data movement between several FICs,by using the DMAC 32104. The block transfer is limited to:

1) The data size must be a LWORD -> D32: BLT
2) The initial address must be in 4 LWORD boundary.
3) The data transfer is supported up to 4 LWORDs.

Using the DMAC with the Quad word data size achieves a transfer speed twice as fast as in a cycle by cycle manner.

Chapter 5

VME MASTER PORT

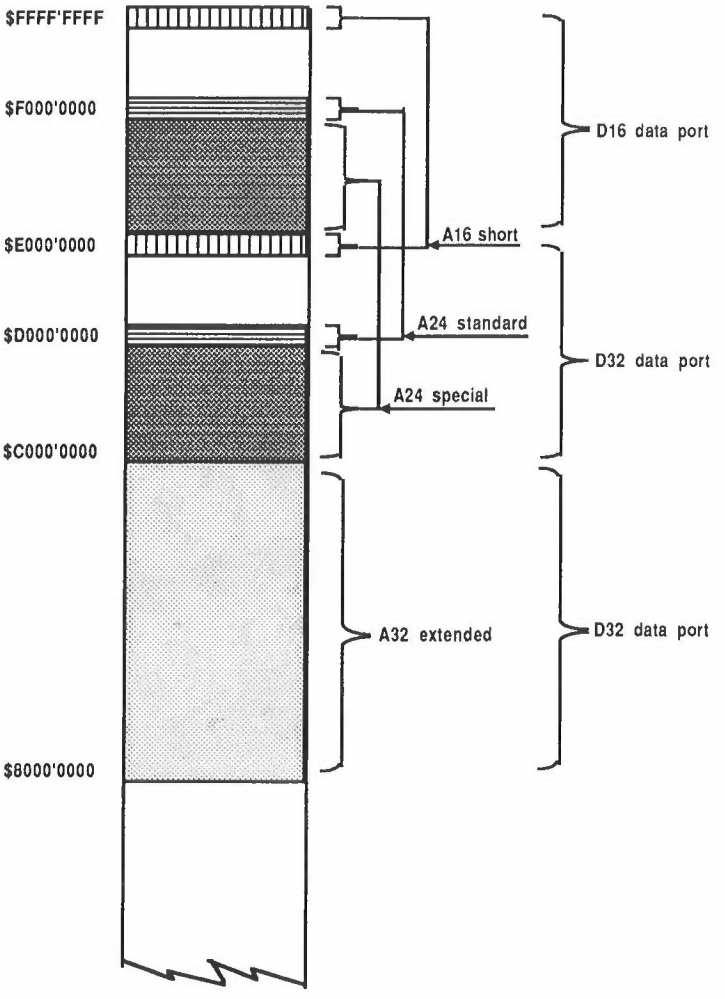
5.1. Addressing Philosophy

Access to VME is directly mapped in the 4 Gbyte CPU/DMAC addressing space. The cycles with the VME address are routed to the VME master port and transferred in the appropriate manner to the VME bus. Only cycles destined for VME are transferred to the VME bus. Before generating a VME cycle, an arbitration procedure is done to assure the VME bus sharing.

2 Gbytes are allowed for VME access. This address space is partitioned in different areas for generating the different VME cycles.

$8000 0000 - $BFFFFFFF	A32 : D32 extended
$F000 0000 - $F0FF FFFF	A24 : D16 standard
$D000 0000 - $D0FF FFFF	A24 : D32 standard
$FFFF 0000 - $FFFF FFFF	A16 : D16 short
$DFFF 0000 - $DFFF FFFF	A16 : D32 short

Special VME cycles (with special AM codes) are also mapped in this space. (see paragraph 5.3.4.).



5.2. Data Port Size

The FIC 8230 is able to generate VME cycles in two different data port sizes. This assures maximum compatibility with the existing VME slave, without altering data transfer efficiency. A24 and A16 VME accesses can be generated by addressing different addresses for 16-bit or 32-bit data ports.

examples:

MOVE.L DØ, $ FFFF 0000 ; generates a 16-bit data transfer in the VME short I/O address (2 x 16 bit word)

MOVE.L DØ, $ DFFF 0000 ; generates a 32-bit data transfer in the VME short I/O address.

5.2.1. D16 Data Port

Data port size : $E000 0000 - $FFFF'FFFF

All access references for this area are transformed in D16 or D08 VME cycles, whatever the data size.

NOTE: The indivisibility of an LWORD cycle generated in the D16 data port is not guaranteed by hardware.

5.2.2. D32 Data Port

This data port size : $C000'0000 - $DFFF'FFFF

All access references in this area are transformed in D08, D16 or D32 VME cycles, depending on the data size format. The FIC 8230 never generates unaligned transfers (UAT).

5.3. AM Generation & VME Addressing Spaces

The FIC 8230 is able to generate VME cycles in 4 different addressing spaces. The AM code generated with a VME cycle depends on:

1) VME addressing space
2) CPU function code

The VME addressing spaces are defined as follows:

1) Extended A32
2) Standard A24
3) Short A16
4) Special A24

5.3.1. Extended Addressing Space

For its extended addressing space, VME uses a 32 bit address range (4 Gbytes).The access to this addressing space is generated when the FIC 8230 accesses the corresponding CPU/DMAC address.

A32 : D32 : $ 8000 0000 - $ BFFF FFFF

This 1 Gbyte window is mapped directly in the VME A32:D32 4 Gbyte space with a 2 bit mapping located in the VME CTL1 register VAD31 & VAD30. (see paragraph. 3.3.1.3)

ex : if VAD31, VAD30 = ØØ (Reset condition)

MOVE.L dØ, $ 8000 0000 -----> will generate a VME cycle A32 : D32 at the VME address $0000 00000

if VAD31, VAD30 = 01(after a set-up in the VMECTL1)

MOVE.L dØ, $ 8000 0000 -----> will generate a VME cycle A32 : D32 at the VME address $ 4000 0000

The AM code generated when the CPU/DMAC accesses this area is defined as follows:

FIC 8230 Master Access	AM Codes Generated	
CPU supervisor data	ØD	
CPU user data	Ø9	
CPU supervisor program	ØE	
CPU user program	ØA	
DMAC supervisor data	ØD	
DMAC user data	Ø9	
DMAC quad word transfer size	ØF	Block transfer

Note: See Chapter 5.6 for Block transfers

5.3.2. Standard Addressing Space

The VME standard addressing space uses a 24 bit address range (16 Mbytes). This space is mapped twice into the FIC 8230 address range, to implement 16-bit or 32-bit data transfers, as follows :

16 bit data port : $ F000 0000 - F0FF FFFF
32 bit data port : $ D000 0000 - D0FF FFFF

ex. MOVE.W,B,L dØ, $ F000 0120; generates VME cycle standard at addr = $120 assuming 16 bit data transfers are accepted.

MOVE.B,W,L dØ, $ D000 0120; generates VME cycle standard at addr = $120 assuming that 32 bit data transfers are accepted.

The generated AM codes are defined as follows:

FIC 8230 Master Access	16 bit port size	32 bit port size
CPU supervisor data	3D	3D
CPU user data	39	39
CPU supervisor program	3E	3E
CPU user program	3A	3A
DMAC supervisor data	3D	3D
DMAC user data	39	39
DMAC quad word transfer size	Not supported	3F

5.3.3. Short I/O Addressing Space

The VME short I/O addressing space uses a 16-bit address range (64 Kbytes). This space is mapped twice into the FIC 8230 address range, to implement 16-bit or 32-bit data transfers, as follows :

16 bit data port : $ FFFF 0000 - FFFF FFFF
32 bit data port : $ DFFF 0000 - DFFF FFFF

ex.: MOVE.W,B,L dØ, $ FFFF 0120; generates a VME short cycle at addr = $120 assuming 16 bit data transfers are accepted

MOVE.B,W,L dØ, $ DFFF 0120; generates a VME short cycle at addr = $120 assuming that 32 bit data transfers are accepted.

The generated AM codes are defined as follows:

FIC 8230 Master Access	16 bit port size	32 bit port size
CPU supervisor data	2D	2D
CPU user data	29	29
DMAC supervisor data	2D	2D
DMAC user data	29	29

5.3.4. Special Addressing Spaces

The FIC 8230 is able to generate special AM codes (user defined or reserved) with VME cycles. These special addressing spaces are labelled from Ø to 7 and are mapped in the CPU/DMAC addressing space for both data sizes as follows:

	D16 port	D32 port
Spec Ø	$E000'0000 - E1FF'FFFF	$C000'0000 - C1FF'FFFF
Spec 1	$E200'0000 - E3FF'FFFF	$C200'0000 - C3FF'FFFF
Spec 2	$E400'0000 - E5FF'FFFF	$C400'0000 - C5FF'FFFF
Spec 3	$E600'0000 - E7FF'FFFF	$C600'0000 - C7FF'FFFF
Spec 4	$E800'0000 - E9FF'FFFF	$C800'0000 - C9FF'FFFF
Spec 5	$EA00'0000 - EBFF'FFFF	$CA00'0000 - CBFF'FFFF
Spec 6	$EC00'0000 - EDFF'FFFF	$CC00'0000 - CDFF'FFFF
Spec 7	$EE00'0000 - EFFF'FFFF	$CE00'0000 - CFFF'FFFF

This special addressing space can be used to define new AM codes for multi-processing environments.

The AM codes generated with these special addressing spaces are defined as follows:

	Special Access Numbers: Ø	1	2	3	4	5	6	7
CPU supervisor data	00	01	02	03	10	11	12	13
CPU user data	04	05	06	07	14	15	16	17
CPU supervisor program	18	19	1A	1B	20	21	22	23
CPU user program	1C	1D	1E	1F	24	25	26	27
DMAC supervisor data	00	01	02	03	10	11	12	13
DMAC user data	04	05	06	07	14	15	16	17
DMAC quad word	$00_{/04}$	$01_{/05}$	$02_{/06}$	$03_{/07}$	$10_{/14}$	$11_{/15}$	$12_{/16}$	$13_{/17}$

REMARK: Some of the generated AM codes (00 ---> 07 & 20 --- > 27) are labelled "RESERVED" in the VME specification C1. The use of this AM code must be done with care according to the VME system used.

5.4. VME Arbitration and Requester

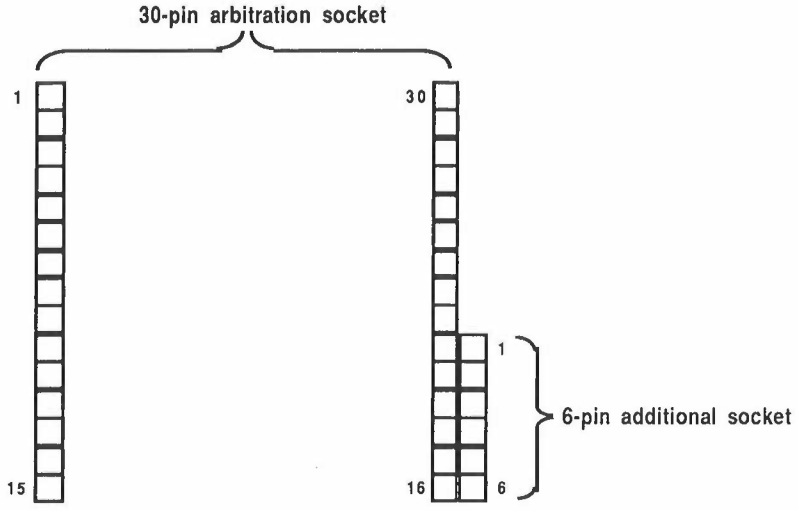
The arbitration of the FIC 8230 is done by a small piggy-back board located at the top right corner of the main PC board. This socket can be equipped with the 3 different following arbiter/requester piggy-backs:

1) ARB8430/1 Requester
2) ARB8430/2 Slot 1 + Requester
3) ARB8430/3 Requester Fair Mode

.The FIC 8230 is equipped in factory with the ARB8430/1

5.4.1. Arbitration Pad socket

The arbitration pad socket is composed of:
- 30-pin standard socket
- 6-pin additional socket



The 30-pin standard socket contains all the signals used in the requester module for the VME standard arbitration and for the distributed parallel arbitration (CERN specs).

The signals are:

pin 1	VCC	:	+ 5V Power supply
pin 2 - 6	MAPØ-MAP4	:	These pins are used to set control options or to define the arbitration level in the distributed arbitration level.
pin 7	HPRI	:	High priority request
pin 8	MWB	:	Master want bus (wired to +5V)
pin 9	MGB	:	Master grant bus (not used)
pin 10	RMCDET	:	Read modify Write Cycle detection
pin 11	SYSRES*	:	Sysreset VME line
pin 12	SYSK	:	Sysclock VME line
pin 13	BBSY*	:	BBSY* VME line
pin 14	GND	:	Ground
pin 15	not used		
pin 16	not used		
pin 17	SYSCLK	:	Sysclock VME
pin 18	BCLR*	:	Bus clear VME
pin 19	BR3	:	Bus request 3 VME
pin 20	BR 2	:	Bus request 2 VME
pin 21	BR1	:	Bus request 1 VME
pin 22	BRØ	:	Bus request Ø VME
pin 23	BGOUT3	:	Bus grant out VME 3
pin 24	BGIN3	:	Bus grant in VME 3
pin 25	BGOUT2	:	Bus grant out VME 2
pin 26	BGIN2	:	Bus grant in VME 2
pin 27	BGOUT1	:	Bus grant out VME 1
pin 28	BGIN1	:	Bus grant in VME 1
pin 29	BGOUTØ	:	Bus grand out VME Ø
pin 30	BGINØ	:	Bus grant in VME Ø

The additional 6-pin pads are used for the slot 1 functions and are defined as follows:

Direct VME lines
pin 1 AS*
pin 2 IACKIN*
pin 3 IACK*
pin 4 DS1*
pin 5 DSØ*
pin 6 BERR*

5.4.2. ARB 8430/2 Slot 1 Functions & 4 Level Arbiter for the FIC8230

5.4.2.1. Introduction

The "Slot 1 arbiter" ARB 8430/2 offers the following functions :

1) The 4 levels arbitration requester as in the 8230ARB/A
2) The 'FAIR' mode option
3) The 16 Mhz oscillator for the VME SYSCLK line
4) The Power/up and the manual VME SYSRESET generation
5) The bus timer module (BTO)
6) The 4 level VME arbiter working in PRI & RRS modes

The typical slot 1 functions can be enabled or disabled with a jumper [J01].

5.4.2.2. Installation

To install the "Slot 1 arbiter" ARB 8430/2, proceed as follows :

1) Remove the 8230ARB/A
2) Plug the ARB 8430/2 into the socket
3) If the jumper J01 is installed, Take care that :

   - No other VME board is VME arbiter
   - No other VME board drives SYSCLK VME

5.4.2.3. Jumper description

J01: This jumper enables the Slot 1 function on board. When that jumper is removed the ARB 8430/2 is equivalent to the 8230ARB/A.

J01 installed: Slot 1 functions enabled

J02: This jumper authorizes the generation of VME SYSRESET when the front panel push button is activated.

J02 installed: front panel RESET generates SYSRESET

5.4.2.4. Bus timer function

The timer module of the ARB 8430 generates a BERR if no reply is received after a certain period of time.
The timing can be adjusted with a trimmer from 8 µs to 15 µs. The BTO function is enabled with the J01 jumper (as in the arbiter).

BTO (8-15 -> 10 µs)

The Trimmer "PT1" allows us to adjust the time-out value (Factory set : 10 µs)

5.4.2.5. System clock Driver

When jumper J01 is installed a 16 Mhz clock is provided for SYSCLK* VME .

5.4.2.6. VME arbitration

The VME arbiter on the ARB 8430 supports the two VME arbitration.

Prioritized	(PRI)
Round Robin Select	(RRS)

Note: Read ch 3.3.1 of the VME specification rev C for explanations of these two modes.

The selection of these two modes is done under software control in the register CTLVME2 bit #4.
That bit is specified reserved . (See ch 3.3.1.3 of the FIC 8230 User's Manual).

CTLVME2 bit #4	: <0> = PRI mode
	: <1> = RRS mode

The VME BCLR line is driven only with the PRI mode.

The arbiter also contains a 'time-out' logic that causes it to withdraw a bus grant if BBSY* is not driven low
by a REQUESTER within 4 [µs]. That abort sequence has no effect on the functionality of the system.

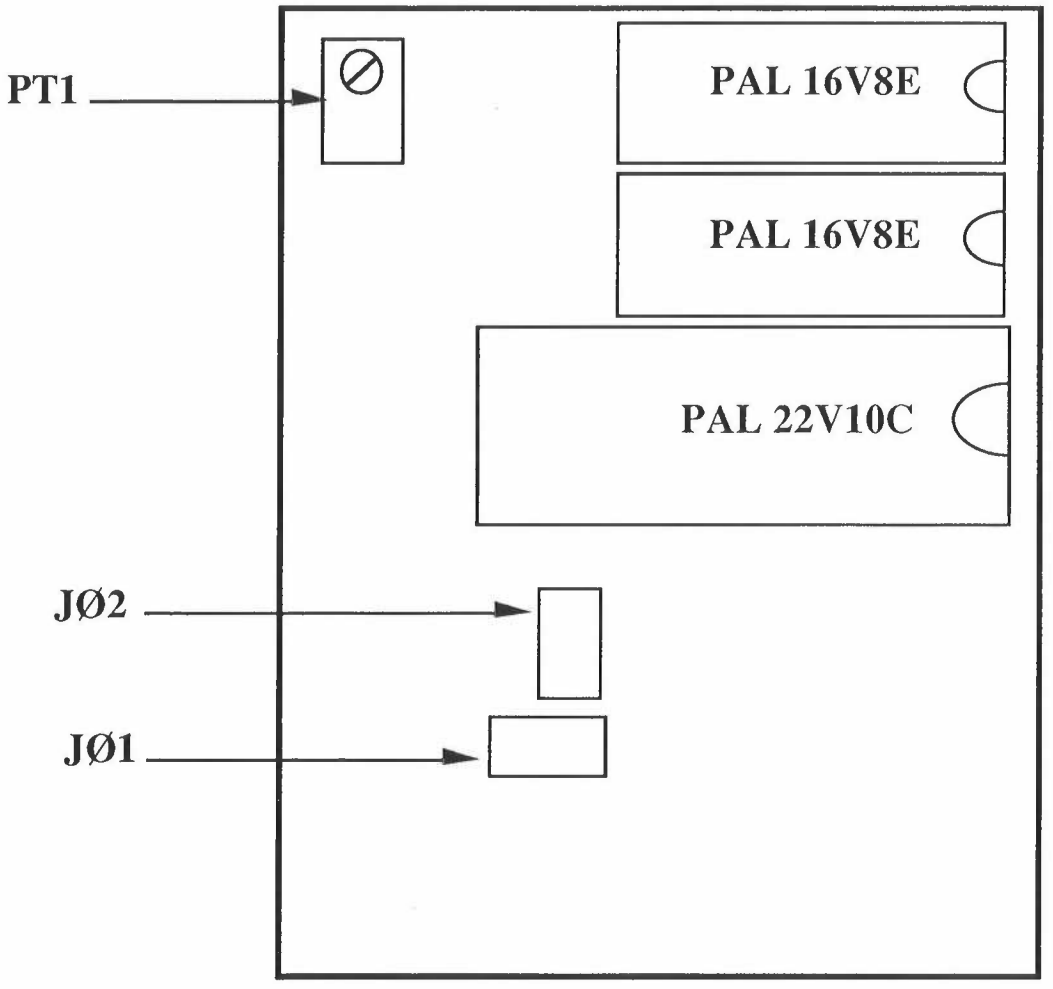
5.4.2.7. Fair mode

The VME Arbitration Requester has been implemented with 'FAIR MODE'. This option is chosen by setting Bit 3 in the VMECTL2 register.

CTLVME2 bit #3	: <0> = FAIR MODE OFF
	: <1> = FAIR MODE ON

5.4.2.8. Layout

ARB8230/2 : SLOT 1 ARBITER



5.4.3. On Board 4 Level Requester

The FIC 8230 VME requester is a 4 level Requester. The features are the following:

* Four level requester (BR3-BRØ)
* Release when Done / Release on Request / No Release modes
* Hidden Requester
* Fairnless Requester

All these features are software controlled through the VMECTL2 register (see paragraph 3.3.1.3).

When the system is running, the Bus request level must be changed with care and only with software protection. The level is normally chosen during the bootstrap procedure.

After a Reset, the options are defined as follows:

LEVEL 3, RWD, No Hidden, No Fairnless.

5.5. RMW and IACK Special Cycles on VME

The FIC 8230 generates these special cycles in VME specifications:

READ MODIFY WRITE : on TAS, CAS, (CAS2) instructions
IACK cycle : on interrupt acknowledge procedure, to get the status ID from the interrupter module.

5.5.1. Read Modify Write Cycle

The FIC 8230 generates RMW cycles on the VME bus when the CPU executes TAS, CAS, (CAS2) special instructions. These instructions must be used with special care taking in consideration the hardware limitations of the VME bus. The RMW cycles can be generated in all VME address spaces and in all data transfer sizes: D08, D16,D32. These instructions are supported on the VME with an RMW cycle as follows:

TAS : in all VME address spaces
CAS <Byte> : in all VME address spaces
CAS < Word > : in all VME address spaces on word boundary (A1=Ø)
CAS < Lword > : only on Lword boundary on the 32-bit data port
CAS2 : cannot be supported on VME (VME hardware limitation)

5.5.2. Iack Cycles

The VME IACK (D08) cycles generated by the FIC 8230 are controlled by the SCB 68155 and issued on the VME bus in the standard manner.

WARNING: Make sure that the daisy-chain is correctly set.

5.6. DMAC Block Mode Access

The DMAC 32204 issues a block data transfer in the VME when the DMAC is programmed to work in quad word data size. (See DMAC data sheet, page 7).This feature is only available on the 32-bit data port. The block transfer procedure reads or writes 4 Lwords at each address cycle. The VME address space can be either the source or the destination of the DMA. The FIC 8230 generates the BLT AM code for this kind of transfer.

5.7. Special Lines Monitoring & Control

The VME line SYSFAIL* can be monitored or activated through the VMECTL 1 register and the VMESTA register. The ACFAIL* can be monitored through the VMESTA register.

The assertion of the VME lines SYSFAIL* or ACFAIL* generates an interrupt level 7 to the 68030 CPU.

Chapter 6

VSB MASTER INTERFACE

6.1. VSB Introduction

The VSB (VME Subsystem Bus) is implemented on the FIC 8230 to attach the extended memory and special I/O for fast data acquisition or monitoring. This bus allows fast access time and provides "smart" features, such as auto-vectoring and geographical addressing.

Both the CPU and the DMAC can access resources via the VSB interface.

The FIC 8230 can also handle the interrupts coming from the VSB interrupt line.

For additional explanations, please consult the IEC VSB specifications.

6.2. VSB Addressing

The FIC 8230 is able to generate cycles in 3 VSB address spaces, mapped through two 512 Mbyte windows, as follows:

SAS : System Address space ($4000 0000 - $5FFF'FFFF)
IOAS : IO Address space ($6000'0000 - $7FFF'FFFF)
CPU : CPU Address space (IACK on VSB)

The 32-bit VSB address is completed with the 3 bits located in the VSBCTL register: AD31, AD30, AD29.

This method allows to directly access 512 Mbytes of each VSB space (IOAS, SAS).

Example:

a) If AD31, 30, 29 = ØØØ (in the VSBCTL register):

- MOVE.X DØ, $4000'1230 --> generates SAS cycles with address = $0000'1230

- MOVE.X DØ, $6012'0000 --> generates IOAS cycles with address = $0012'0000

b) If AD31, 30, 29 = 101

- MOVE.X DØ, $4000'1230 --> generates SAS cycles with address = $A000'1230

6.3. Transfer Cycles

The data transfer cycles between the FIC 8230 and the VSB slave devices are carried out in accordance with the VSB specifications revision C. They can be either Single Transfer cycles or Block transfers if the DMAC is the original master device.

The FIC 8232 does not issue Address-only cycles.

The transfer size can be Byte/Word or LWORD.

Normal termination of a VSB cycle is signalled with an acknowledge ("ACK") coming directly from the slave device. In some cases, an error ("ERR") can abort the current VSB cycle. The source of this BERR can be determined with the two following status bits:

- TO : STACPU bit <5> : Time-out FIC 8230
- ADRE : STAVSB bit <4> : Address error

The address error status bit is cleared at the beginning of each VSB cycle. It is set if AC becomes active while ASACKØ & ASACK1 = Ø.

The combination of these two status bits gives:

TO	ADRE	Cause of BERR
Ø	Ø	The error is directly received from the VSB slave.
Ø	1	The VSB address is not decoded in the System.
1	Ø	A time-out has occurred during the DS handshake.
1	1	A time-out has occurred during the Address broadcast phase.

The FIC 8230 supports slave device data sizes as follows:

CPU 68020 supports D08, D16, D32
DMAC 32104 supports D32 port only

6.3.1. Block Transfer Support

The DMAC 32104 can perform block transfers on the VSB. This block transfer is limited to 4 x 32-bit words after each address broadcast phase. The 32104 does not support "block transfer handshake". When it begins a block transfer, it always finishes it, whatever the state of the AC during the data handshake.

This limitation is not important as slaves supporting block access should have address counters with a greater capacity.

6.4. VSB Arbitration

The FIC 8230 uses VSB serial arbitration. The on board logic can be a SER REQUESTER or a SER ARBITER. These two functions are normally defined by the slot number (VSB geographic addressing).

slot N^{O} = Ø : Arbiter
slot N^{O} ≠ Ø : Requester

To assure maximum flexibility, two bits in the CTLVSB register allows to modify this set-up. These two bits (ARB2 and ARB1) encode the following options:

ØØ ---> std arbitration mode slot Ø ---> Arbiter (Otherwise ---> Requester)
1.Ø ---> VSB Serial arbiter whatever the slot N^{O}
Ø.1 ---> VSB Serial Requester whatever the slot N^{O}

The VSB serial requester can work in two different modes selected by bit 2 of the CTLVSB register.

Release on request ---> if bit # 2 = Ø
Release when done ---> if bit # 2 = 1

6.5. Interrupt Capability

The FIC 8230 monitors the VSB IRQ line; thus the CPU can be interrupted by a request coming directly from the VSB.

This interrupt is routed to the Interrupt controller SCB 68155 at LIRQ 1 or 4.

VSB int ---> level 1 or level 4

Only one of these two levels must be able to interrupt the CPU ; thus at least one of the two levels must be masked in the SCB 68155.

The Interrupt acknowledge sequence is done in two different ways, depending on the level enabled.

Level 4 : The vector associated to the Interrupt is fetched from the SCB 68155. This mode corresponds to the IHP.

Level 1 : The associated vector is fetched from the VSB with an Interrupt acknowledge sequence (IHV). When using this mode, the VSB slave devices must be able to arbitrate between themselves to supply the vector.

6.6. Other Status Bits

The VSB slot N^{O} can be read in the STAVSB register bits #Ø, #1 and #2.

Bit #3 of the STAVSB reflects the BBSY line issued by the local SER REQUESTER.

6.7. VSB Pin Assignment

The VSB uses the P2 extension connectors, rows A and C. The VSB backplane must have a proper terminator, as specified in the VSB specifications.

VSB devices cannot be connected with ribbon cables.

PIN	ROW A	ROW C
1	AD00	AD01
2	AD02	AD03
3	AD04	AD05
4	AD06	AD07
5	AD08	AD09
6	AD10	AD11
7	AD12	AD13
8	AD14	AD15
9	AD16	AD17
10	AD18	AD19
11	AD20	AD21
12	AD22	AD23
13	AD24	AD25
14	AD26	AD27
15	AD28	AD29
16	AD30	AD31
17	GND	GND
18	IRQ*	GND
19	DS*	GND
20	WR*	GND

PIN	ROW A	ROW C
21	SPACE0	SIZE0
22	SPACE1	PAS*
23	LOCK*	SIZE1
24	ERR*	GND
25	GND	ACK*
26	GND	AC
27	GND	ASACK1*
28	GA0	ASACK0*
29	GA1	CACHE*
30	GA2	WAIT*
31	BGIN*	BUSY*
32	BREQ*	BGOUT*

Chapter 7

GENERAL INFORMATION

7.1. Front Panel Description

The FIC 8230 front panel is composed of:

- 1 x small display of 12 LEDs
- 2 x 10-pole flat cable connector for RS-232 connection
- 1 x 64-pole flat cable connector as a general purpose connector (GPC)
- 1 x Lemo ØØ connector for the Cheapernet LAN connection

7.1.1. Front Panel LED Indicators

The font panel display is composed of the following 12 LED indicators:

VME MASTER	: On when the FIC 8232 accesses the VMEbus (master)
VME SLAVE	: On when the VME slave interface accesses the global resources
VSB CONTROL	: On when the FIC 8232 is arbiter of the VSB bus
VSB MASTER	: On when the FIC 8232 is master of the VSB (BUSY * VSB line)
CPU HALT	: On when the CPU 68030 "halt" line is asserted
INT PEND	: On when an interrupt is pending on the 68030
DMAC MASTER	: On when the DMAC WE32204 has granted the CPU bus
FIFO ACCESS	: On when one of the FIFOs are accessed
MBN8	: These 4 LEDs reflect the state of the rotary switch defining the master number of this unit.
MBN4	:
MBN2	:
MBN1	:

7.1.2. Lemo ØØ CRCTL Output

This socket allows to trigger a logic analyzer on the VMDIS 8003 when the FIC 8230 is a master on the VMEbus. This active high TTL output is equivalent to its BBSY* signal. It can be useful for debugging in multiprocessor systems.

7.1.3. RS232 Connectors

These two 10-pole flat cable connectors are assigned as follows:

Top connector	: Terminal channel 1 Z8030A
Bottom connector	: Host channel 2 Z8030A

Pin assignment (for both connectors):

pin 2, 5, 9 & 10	: GND
pin 3	: Tx
pin 8	: RX

pin 1 is located at the bottom right.

7.2. General Purpose Connector (P5)

The 64-pin flat cable connector is assigned for:

RESET, ABORT : see paragraph 1.9
Front panel I/O, Interrupt : see paragraph 2.5
DMAC peripheral Bus : see paragraph 2.3.6

The pin # 1 is located at the bottom right.

DMAC port: (Pin 1 --> 34)

PIN

01 : Peripheral data # Ø
02 : GND
03 : Peripheral data # 1
04 : Peripheral data # 2
05 : Peripheral data # 3
06 : GND
07 : Peripheral data # 4
08 : Peripheral data # 5
09 : Peripheral data # 6
10 : Peripheral data # 7
11 : Peripheral Address # Ø
12 : GND
13 : Peripheral Address # 1
14 : Peripheral Address # 2
15 : Peripheral Address # 3
16 : Peripheral Address # 4
17 : Peripheral Request # Ø
18 : GND
19 : Peripheral Request # 1
20 : Peripheral Request # 2
21 : Peripheral Request # 3
22 : Peripheral /CS # Ø
23 : Peripheral /CS # 1
24 : GND
25 : Peripheral /CS # 2
26 : Peripheral /CS # 3
27 : Peripheral Data Strobe
28 : GND
29 : Peripheral /DTACK
30 : GND
31 : Peripheral /WRITE
32 : + 5 (V) VCC
33 : Peripheral DONE
34 : + 5 (V) VCC

Z8036#1 Port (pin 35 --> pin 58) : see paragraph 2.5.2.3

ABORT/RESET Port (pin 59 --> pin 64)

59 : Direct/ABORT (active/low)
60 : Direct/RESET (active/low)
61 : ABORT switch
62 : /ABORT switch
63 : RESET switch
64 : /RESET switch

7.3. Installation Notes

Before inserting the unit into a VME backplane, the following must be done:

1) Make sure that the VME address of the unit is correctly selected with the front panel rotary switch. (Refer to Chapter 4)

2) The P2 extension bus (rows A and C) must be free of local busses or implemented with a VSB backplane.

3) The BGIN/BGOUT daisy-chain must be correctly set up.

4) The VME crate should not be powered on.

After power on:

1) Plug the RS232 cable in the "Terminal Connector".
   The communication on this port is done with: 9600 baud/8-bit/no parity.

2) Consult the software references for the Firmware monitor implemented on board.

Appendix

EPROM CONTENTS

A.1. FIC 8230 parameter definition file

```
        ttl FIC8230 parameter definition file
*       ====================================
*       FIC8230 parameter definition file
*       -addresses seen from CPU 68020 & DMAC 32104
*       ====================================
*       v0.0 10-Jun-87 jb  Original version
*       v0.1 29-jun-87 jb  Minor corrections
*       v0.2 22-jul-87 rb  correct ch1_r2_z8030
*       v1.0 29-jul-87 rb  comments in English!
*
*       Local EPROM memory    : 2 x 27C512 = 128 Kbytes
*       ********************************************
*
bas_loc_eprom   equ     $00000000
top_loc_eprom   equ     $0001FFFF
len_eprom       equ     top_loc_eprom-bas_loc_eprom+1
*
*
*       Local RAM memory     : 4 x 62256 = 128 Kbytes
*       ********************************************
*
bas_loc_ram     equ     $08000000
top_loc_ram     equ     $0801FFFF
len_ram         equ     top_loc_ram-bas_loc_ram+1
*
*
        page
*
*       On-board peripherals
*       ********************
*
*       - Z8030         : RS232 controller
*       - Z8036         : Timer, Parallel I/O, FIFO Interrupt controller
*       - DP8570        : RTC, with small battery back-up CMOS RAM
*       - SCB68154      : VME Interrupter
*       - SBC68155      : Interrupt controller
*
*       Z8030 : 2 RS232 channels
*       ------------------------
*       (Refer to Zilog documents)
*
bas_z8030       equ     $10000000
*
ch1_r0_z8030    equ     $10000021       ; register #00 channel #1
ch1_r1_z8030    equ     $10000023       ; register #01 channel #1
ch1_r2_z8030    equ     $10000025       ; register #02 channel #1
ch1_r3_z8030    equ     $10000027       ; register #03 channel #1
ch1_r4_z8030    equ     $10000029       ; register #04 channel #1
ch1_r5_z8030    equ     $1000002B       ; register #05 channel #1
ch1_r6_z8030    equ     $1000002D       ; register #06 channel #1
ch1_r7_z8030    equ     $1000002F       ; register #07 channel #1
ch1_r8_z8030    equ     $10000031       ; register #08 channel #1
ch1_r9_z8030    equ     $10000033       ; register #09 channel #1
ch1_rA_z8030    equ     $10000035       ; register #10 channel #1
ch1_rB_z8030    equ     $10000037       ; register #11 channel #1
ch1_rC_z8030    equ     $10000039       ; register #12 channel #1
ch1_rD_z8030    equ     $1000003B       ; register #13 channel #1
ch1_rE_z8030    equ     $1000003D       ; register #14 channel #1
ch1_rF_z8030    equ     $1000003F       ; register #15 channel #1
```

```
ch2_r0_z8030    equ     $10000000       ; register #00 channel #2
ch2_r1_z8030    equ     $10000002       ; register #01 channel #2
ch2_r2_z8030    equ     $10000004       ; register #02 channel #2
ch2_r3_z8030    equ     $10000006       ; register #03 channel #2
ch2_r4_z8030    equ     $10000008       ; register #04 channel #2
ch2_r5_z8030    equ     $1000000A       ; register #05 channel #2
ch2_r6_z8030    equ     $1000000C       ; register #06 channel #2
ch2_r7_z8030    equ     $1000000E       ; register #07 channel #2
ch2_r8_z8030    equ     $10000010       ; register #08 channel #2
ch2_r9_z8030    equ     $10000012       ; register #09 channel #2
ch2_rA_z8030    equ     $10000014       ; register #10 channel #2
ch2_rB_z8030    equ     $10000016       ; register #11 channel #2
ch2_rC_z8030    equ     $10000018       ; register #12 channel #2
ch2_rD_z8030    equ     $1000001A       ; register #13 channel #2
ch2_rE_z8030    equ     $1000001C       ; register #14 channel #2
ch2_rF_z8030    equ     $1000001E       ; register #15 channel #2
        page
*       Z8036 : Parallel I/O & Timers
*
*       Refer (again!) to Zilog documents
*       Adresses referances avec RJA=1 (voir Master Interrupt Control)
*
bas_z8036       equ     $11000000
*
*               Main Control Registers
mir_z8036       equ     $11000000       ; Master Interrupt Control
mcc_z8036       equ     $11000001       ; Master Configuration Control
iva_z8036       equ     $11000002       ; Port A Interrupt Vector
ivb_z8036       equ     $11000003       ; Port B Interrupt Vector
ctir_z8036      equ     $11000004       ; Counter/Timers Interrupt Vector
dppc_z8036      equ     $11000005       ; Port C Data Path Polarity
ddc_z8036       equ     $11000006       ; Port C Data Direction
siocc_z8036     equ     $11000007       ; Port C Special I/O Control
*
*               Most frequently accessed Registers
csa_z8036       equ     $11000008       ; Port A Command & Status
csb_z8036       equ     $11000009       ; Port B Command & Status
ct1_z8036       equ     $1100000A       ; Command & Status,Counter/Timer 1
ct2_z8036       equ     $1100000B       ; Command & Status,Counter/Timer 2
ct3_z8036       equ     $1100000C       ; Command & Status,Counter/Timer 3
da_z8036        equ     $1100000D       ; Port A Data
db_z8036        equ     $1100000E       ; Port B Data
dc_z8036        equ     $1100000F       ; Port C Data
*
*               Counter/Timer Related registers
ct1cc_z8036     equ     $11000010       ; Timer/Counter 1 Current Count (16 bit word)
ct2cc_z8036     equ     $11000012       ; Timer/Counter 2 Current Count (16 bit word)
ct3cc_z8036     equ     $11000014       ; Timer/Counter 3 Current Count (16 bit word)
ct1tc_z8036     equ     $11000016       ; Timer/Counter 1 Time Constant (16 bit word)
ct2tc_z8036     equ     $11000018       ; Timer/Counter 2 Time Constant (16 bit word)
ct3tc_z8036     equ     $1100001A       ; Timer/Counter 3 Time Constant (16 bit word)
ct1ms_z8036     equ     $1100001C       ; Timer/Counter 1 Mode Specification
ct2ms_z8036     equ     $1100001D       ; Timer/Counter 2 Mode Specification
ct3ms_z8036     equ     $1100001E       ; Timer/Counter 3 Mode Specification
ctcv_z8036      equ     $1100001F       ; Timer/Counter Current Vector

*               Port A Specification Registers
msa_z8036       equ     $11000020       ; Port A Mode Specification
hsa_z8036       equ     $11000021       ; Port A Handshake Specification
dppa_z8036      equ     $11000022       ; Port A Data Path Polarity
dda_z8036       equ     $11000023       ; Port A Data Direction
sioa_z8036      equ     $11000024       ; Port A Special I/O Control
ppa_z8036       equ     $11000025       ; Port A Pattern Polarity
pta_z8036       equ     $11000026       ; Port A Pattern Transition
pma_z8036       equ     $11000027       ; Port A Pattern Mask
```

```
*               Port B Specification Registers
msb_z8036       equ     $11000028       ; Port B Mode Specification
hsb_z8036       equ     $11000029       ; Port B Handshake Specification
dppb_z8036      equ     $1100002A       ; Port B Data Path Polarity
ddb_z8036       equ     $1100002B       ; Port B Data Direction
siob_z8036      equ     $1100002C       ; Port B Special I/O Control
ppb_z8036       equ     $1100002D       ; Port B Pattern Polarity
ptb_z8036       equ     $1100002E       ; Port BPattern Transition
pmb_z8036       equ     $1100002F       ; Port B Pattern Mask
        page
*       DP8570 : Real Time Clock
*       ------------------------
*       Refer to National Semiconductor documents.
*
bas_dp8570      equ     $15000000
*
r00_dp8570      equ     $15000000       ; Main Status register
r01_dp8570      equ     $15000001       ; Real Time Mode Reg/Timer 1 CTL reg
r02_dp8570      equ     $15000002       ; ... etc
r03_dp8570      equ     $15000003       ;
r04_dp8570      equ     $15000004       ;
r05_dp8570      equ     $15000005       ;
r06_dp8570      equ     $15000006       ;
r07_dp8570      equ     $15000007       ;
r08_dp8570      equ     $15000008       ;
r09_dp8570      equ     $15000009       ;
r0A_dp8570      equ     $1500000A       ;
r0B_dp8570      equ     $1500000B       ;
r0C_dp8570      equ     $1500000C       ;
r0D_dp8570      equ     $1500000D       ;
r0E_dp8570      equ     $1500000E       ;
r0F_dp8570      equ     $1500000F       ;
r10_dp8570      equ     $15000010       ;
r11_dp8570      equ     $15000011       ;
r12_dp8570      equ     $15000012       ;
r13_dp8570      equ     $15000013       ;
r14_dp8570      equ     $15000014       ;
r15_dp8570      equ     $15000015       ;
r16_dp8570      equ     $15000016       ;
r17_dp8570      equ     $15000017       ;
r18_dp8570      equ     $15000018       ;
r19_dp8570      equ     $15000019       ;
r1A_dp8570      equ     $1500001A       ;
r1B_dp8570      equ     $1500001B       ;
r1C_dp8570      equ     $1500001C       ;
r1D_dp8570      equ     $1500001D       ;
r1E_dp8570      equ     $1500001E       ;
r1F_dp8570      equ     $1500001F       ;
        page
*       SCB68155 : Interrupt controller
*       -------------------------------
*       Refer to Signetics data sheet (for a change)
*
bas_68155       equ     $16000000
*
r0_68155        equ     $16000000       ; Pointer register (W)
r1_68155        equ     $16000002       ; Control register (R/W)
r2_68155        equ     $16000004       ; LRQ vector register (R/W)
r3_68155        equ     $16000006       ; LRQ Mask register (R/W)
r4_68155        equ     $16000008       ; LRQ Source register (R)
r5_68155        equ     $1600000A       ; IRQ Mask (R/W)
r6_68155        equ     $1600000C       ; IRQ Status register (R)
r7_68155        equ     $1600000E       ; Last Interrupt acknowledged (R)
*
```

```
*       SCB68154 : VME Interrupt Generator
*       ----------------------------------
*       Refer to Signetics data sheet (for a change)
*
bas_68154       equ     $17000000
*
r0_68154        equ     $17000000       ; Interrupt vector register
r1_68154        equ     $17000002       ; Interrupt request register
        page
*       DMA Controlleur
*       ***************
*       Refer to AT&T documents
*       All accesses are LWORD.
*
*       Access to the Peripheral Bus
pb1_bas_dmac    equ     $18000000  ; Base Address of peripheral channel #1
pb2_bas_dmac    equ     $18000080  ; Base Address of peripheral channel #2
pb3_bas_dmac    equ     $18000100  ; Base Address of peripheral channel #3
pb4_bas_dmac    equ     $18000180  ; Base Address of peripheral channel #4
*
*       DMAC Internal registers - channel #1
sar1_dmac       equ     $18000200  ; Source Address Register channel #1
dar1_dmac       equ     $18000204  ; Destination Address Register channel #1
bar1_dmac       equ     $1800020C  ; Base Address Register channel #1
tcr1_dmac       equ     $18000210  ; Transfer Count Register channel #1
ivr1_dmac       equ     $18000218  ; Interrupt Vector Register channel #1
csr1_dmac       equ     $18000220  ; Status & Control Register channel #1
md1_dmac        equ     $18000224  ; Mode Register channel #1
dc1_dmac        equ     $18000228  ; Device Control Register channel #1
*
*                               - channel #2
sar2_dmac       equ     $18000280  ; Source Address Register channel #2
dar2_dmac       equ     $18000284  ; Destination Address Register channel #2
bar2_dmac       equ     $1800028C  ; Base Address Register channel #2
tcr2_dmac       equ     $18000290  ; Transfer Count Register channel #2
ivr2_dmac       equ     $18000298  ; Interrupt Vector Register channel #2
csr2_dmac       equ     $180002A0  ; Status & Control Register channel #2
md2_dmac        equ     $180002A4  ; Mode Register channel #2
dc2_dmac        equ     $180002A8  ; Device Control Register channel #2
*
*                               - channel #3
sar3_dmac       equ     $18000300   ; Source Address Register channel #3
dar3_dmac       equ     $18000304   ; Destination Address Register channel #3
bar3_dmac       equ     $1800030C   ; Base Address Register channel #3
tcr3_dmac       equ     $18000310   ; Transfer Count Register channel #3
ivr3_dmac       equ     $18000318   ; Interrupt Vector Register channel #3
csr3_dmac       equ     $18000320   ; Status & Control Register channel #3
md3_dmac        equ     $18000324   ; Mode Register channel #3
dc3_dmac        equ     $18000328   ; Device Control Register channel #3
*
*                               - channel #4
sar4_dmac       equ     $18000380   ; Source Address Register channel #4
dar4_dmac       equ     $18000384   ; Destination Address Register channel #4
bar4_dmac       equ     $1800038C   ; Base Address Register channel #4
tcr4_dmac       equ     $18000390   ; Transfer Count Register channel #4
ivr4_dmac       equ     $18000398   ; Interrupt Vector Register channel #4
csr4_dmac       equ     $180003A0   ; Status & Control Register channel #4
md4_dmac        equ     $180003A4   ; Mode Register channel #4
dc4_dmac        equ     $180003A8   ; Device Control Register channel #4
*
*       DMAC Internal registers ( Global )
mask_dmac       equ     $18000410   ; Mask Register Global
*
*       DMAC Internal Data Buffers
ibuf1_dmac      equ     $18000600    ; Base internal buffer channel #1
ibuf2_dmac      equ     $18000680    ; Base internal buffer channel #1
ibuf3_dmac      equ     $18000700    ; Base internal buffer channel #1
ibuf4_dmac      equ     $18000780    ; Base internal buffer channel #1
        page
```

```
*       Global Memory
*       *************
*
bas0_glob       equ     $20000000    ; Base of dual-port region
bas1_glob       equ     $20002000    ; Base of dual-port memory
*end1_glob       equ     $2007FFFF   ; End of 512K (static) RAM
end2_glob       equ     $200FFFFF    ; End of 1M (static) RAM
end3_glob       equ     $201FFFFF    ; End of 2M (dynamic) RAM
*
*       Control & Status Registers
*       --------------------------
*       Refer to the FIC8230 User Guide
*       All accesses are LWORD
*
ctlcpu          equ     $20000000              ; CPU register
ctlvsb          equ     $20000004              ; VSB register
ctlvme1         equ     $20000008              ; VME! register
ctlvme2         equ     $2000000C              ; VME2 register
*
stacpu          equ     $20000404              ; CPU status register
stavsb          equ     $20000400              ; VSB status register
stavme          equ     $20000408              ; VME status register
stafif          equ     $2000040C              ; FIF status register
        page
*       FIFO Registers (R/W)
*
*       Refer to the FIC 8230 User's Guide
*       Use addresses  00-4f
*
bas_fifo        equ     $20000800

*       FIFO #0
fif0_rpt        equ     $20000810              ; Acces Read Pointer FIFO #0
fif0_wpt        equ     $20000820              ; Acces Write Pointer FIFO #0
fif0_wct        equ     $20000830              ; Acces Word Counter FIFO #0
*
*       FIFO #1
fif1_rpt        equ     $20000814              ; Acces Read Pointer FIFO #1
fif1_wpt        equ     $20000824              ; Acces Write Pointer FIFO #1
fif1_wct        equ     $20000834              ; Acces Word Counter FIFO #1
*
*       FIFO #2
fif2_rpt        equ     $20000818              ; Acces Read Pointer FIFO #2
fif2_wpt        equ     $20000828              ; Acces Write Pointer FIFO #2
fif2_wct        equ     $20000838              ; Acces Word Counter FIFO #2
*
*       FIFO #3
fif3_rpt        equ     $2000081C              ; Acces Read Pointer FIFO #3
fif3_wpt        equ     $2000082C              ; Acces Write Pointer FIFO #3
fif3_wct        equ     $2000083C              ; Acces Word Counter FIFO #3
*
*       Control (Set/Reset) for FIFO FULL & EMPTY flags
*       -Write only access in LWORD
*               - bit #0 : FULL flag
*               - bit #1 : EMPTY flag
fif0_flag       equ     $20000840   ; Acces Set/Reset flags FIFO #0
fif1_flag       equ     $20000844   ; Acces Set/Reset flags FIFO #1
fif2_flag       equ     $20000848   ; Acces Set/Reset flags FIFO #2
fif3_flag       equ     $2000084C   ; Acces Set/Reset flags FIFO #3
*
*       FIFO Access
fifo0           equ     $20000C00   ; Acces I/O FIFO #0
fifo1           equ     $20000C04   ; Acces I/O FIFO #1
fifo2           equ     $20000C08   ; Acces I/O FIFO #2
fifo3           equ     $20000C0C   ; Acces I/O FIFO #3
*       LIFO Access
lifo0           equ     $20000C40   ; Acces O LIFO #0
lifo1           equ     $20000C44   ; Acces O LIFO #1
lifo2           equ     $20000C48   ; Acces O LIFO #2
lifo3           equ     $20000C4C   ; Acces O LIFO #3
```

```
*       FIFO memory storage
fif0_bas        equ     $20001000   ; Base of FIFO#0 memory
fif1_bas        equ     $20001400   ; Base of FIFO#1 memory
fif2_bas        equ     $20001800   ; Base of FIFO#2 memory
fif3_bas        equ     $20001C00   ; Base of FIFO#3 memory
        page
*       VSB Access
*       **********
*
bas_vsbmem      equ     $40000000   ; Base for VSB memory access
bas_vsbio       equ     $60000000   ; Base for VSB I/O access
*
*
*
*       VME Access
*       **********
*
*       VME access mode A32:D32
*       -----------------------
*               - supervisor data: AM = 0D
*               - user data      : AM = 09
*               - supervisor prog: AM = 0E
*               - user prog      : AM = 0A
*
vmeext      equ     $80000000               ; VME access mode A32:D32
*
*
*       VME access mode A24:D16/D32
*       ---------------------------
*               - supervisor data: AM = 3D
*               - user data      : AM = 39
*               - supervisor prog: AM = 3E
*               - user prog      : AM = 3A
*
vmest_d16   equ     $F0000000               ; VME standard access A24:D16
vmest_d32   equ     $D0000000               ; VME standard access A24:D32
*
*
*       VME access mode A16:D16/D32
*       ---------------------------
*               - supervisor data: AM = 2D
*               - user data      : AM = 29
*               - supervisor prog: AM = 2E  ; BEWARE, not supported
*               - user prog      : AM = 2A  : BEWARE, not supported
*
vmeio_d16   equ     $FFFF0000               ; VME short i/o A16:D16
vmeio_d32   equ     $DFFF0000               ; VME short i/o A16:D32
        page
*       VME special access, mode A24:D16/D32
*       ------------------------------------
*
*                                   Special Access Number
*                          ------------------------------------------------
*                          *  0  *  1  *  2  *  3  *  4  *  5  *  6  *  7  *
*    ------------------------------------------------------------------------
*    -supervisor data: AM = 00  *  01 *  02 *  03 *  10 *  11 *  12 *  13 *
*    -user data      : AM = 04  *  05 *  06 *  07 *  14 *  15 *  16 *  17 *
*    -supervisor prog: AM = 18  *  19 *  1A *  1B *  20 *  21 *  22 *  23 *
*    -user prog      : AM = 1C  *  1D *  1E *  1F *  24 *  25 *  26 *  27 *
*    ------------------------------------------------------------------------
```

```
*       With a 16 bit data port
*
vmesp0_d16  equ     $E0000000  ; VME special access #0  A24:D16
vmesp1_d16  equ     $E2000000  ; VME special access #1  A24:D16
vmesp2_d16  equ     $E4000000  ; VME special access #2  A24:D16
vmesp3_d16  equ     $E6000000  ; VME special access #3  A24:D16
vmesp4_d16  equ     $E8000000  ; VME special access #4  A24:D16
vmesp5_d16  equ     $EA000000  ; VME special access #5  A24:D16
vmesp6_d16  equ     $EC000000  ; VME special access #6  A24:D16
vmesp7_d16  equ     $EE000000  ; VME special access #7  A24:D16
*
*       With a 32 bit data port
*
vmesp0_d32  equ     $C0000000  ; VME special access #0 A24:D32
vmesp1_d32  equ     $C2000000  ; VME special access #1 A24:D32
vmesp2_d32  equ     $C4000000  ; VME special access #2 A24:D32
vmesp3_d32  equ     $C6000000  ; VME special access #3 A24:D32
vmesp4_d32  equ     $C8000000  ; VME special access #4 A24:D32
vmesp5_d32  equ     $CA000000  ; VME special access #5 A24:D32
vmesp6_d32  equ     $CC000000  ; VME special access #6 A24:D32
vmesp7_d32  equ     $CE000000  ; VME special access #7 A24:D32
```

A.2. FIC 8230 environment setup code for pROBE-SDT

```
Quelo  ...A68K D6.2Ja Dec 23 1986   ...Run on Dec 19, 1988  15:43:53
8230YNN.LTX , 8230YNN.PRN , 8230YNN.SYM = 8230YNN.A68

1.          ttl    FIC8230 environment setup code for pROBE-SDT
2.  ynn8230 idnt   3,02
3.          opt     p=68020/68881
4.          opt     mex,cex
5.  ************************************************************************
6.  *                                                                      *
7.  *                       Environment setup for pROBE-SDT                *
8.  *                        on CES FIC8230                                *
9.  *                                                                      *
10. ************************************************************************
11. *                                                                      *
12. *                      MODULE:  8230YNN.M68                            *
13. *                      VERSION: 3.02                                   *
14. *                      AUTHOR:  R. Bell/ J. Bovier (CES SA)            *
15. *                      DATE:    30/03/88                               *
16. *                                                                      *
17. *        Copyright 1987, CES Creative Electronic Systems SA            *
18. *        70 route du Pont-Butin, CH-1213 Petit-Lancy, Geneva           *
19. *                      All Rights Reserved                             *
20. *                                                                      *
21. ************************************************************************
22.
23.  ************************************************************************
24.  * Configuration section
25.  * - pROBE will be present if either pSOS or pRISM is present; then the
26.  *   system config and probe tables will be in RAM.
27.  * - prodt will be assumed if both pSOS and pRISM are excluded; then
28.  *   the sys config and probe tables will be in EPROM.
29.  ************************************************************************
     # 00000000            30.  psos_inc  equ 0         ;pSOS included =1  excluded = 0
     # 00000000            31.  prism_inc equ 0         ;pRISM included =1  excluded = 0
     # 00000000            32.  vers3     equ 0         ;temporary elimination of prism startup


34.  ************************************************************************
35.  *                                                                      *
36.  *   This  environment package for the CES FIC8230 will contain the     *
37.  *   following modules:                                                 *
38.  *                                                                      *
39.  *           Module                              ROM offset             *
40.  *      --------------------                     ----------             *
41.  *                                                                      *
42.  *      8230booz       (v. 3.02)                     0                 *
43.  *                                                                      *
44.f         ifne    prism_inc!psos_inc
45.n         endc
46.t         ifeq    prism_inc!psos_inc
47.  *      pRODT(emasculated pROBE)                 6700 - 10040 **       *
48.n         endc
49.  *                                                                      *
50.f         ifne    prism_inc
51.n         endc
52.f         ifne    psos_inc
53.n         endc
54.  *                                                                      *
55.  ************************************************************************
56.  *
```

```
57.          nolist
72.          list
73.  ************************************************************************
74.  *                                                                      *
75.  *    The FIC8230 initialisation module consists of:                    *
76.  *                                                                      *
77.  *      * BOOT CODE - starts the system upon Power-on or RESET.         *
78.  *       -performs these functions:                                     *
79.  *           * Initialise control registers                             *
80.  *           * Test local ram                                           *
81.  *           * Initialise the vector pointer and vector page            *
82.  *           * Start pROBE as procedure                                 *
83.  *           * Initialise various on-board controllers                  *
84.  *                                                                      *
85.  *             -then according to the configuration selected,           *
86.  *           * Construct the system configuration table in RAM          *
87.  *           * Copy pRISM and pSOS into RAM                             *
88.  *           * Initialise pRISM appropriately if it is present          *
89.  *           * Write to a control register to generate a 'strobe'       *
90.  *           * Pass control to the start-up entry point in pROBE        *
91.  *             or pRISM                                                 *
92.  *                                                                      *
93.  *      * system configuration table                                    *
94.  *                                                                      *
95.  *      * pROBE configuration table                                     *
96.  *                                                                      *
97.  *      * pSOS, pRISM configuration & initialisation tables             *
98.  *                                                                      *
99.  *      * pROBE serial drivers - for console and host channels          *
100. *                                                                      *
101. *      * Console output driver for demo package                        *
102. *                                                                      *
103. ************************************************************************


104.  ************************************************************************
105.  *    This code and pROBE code is in EPROM, but pSOS, pRISM and the     *
106.  *    system control tables are copied into ram as part of the          *
107.  *    initialisation process.                                           *
108.  ************************************************************************
109.
110.          nolist
```

A.3. pROBE - FIC 8230 parameter definition file

```
524.          list

526.f         ifne    psos_inc        ; if pSOS to be included
527.n         endc
528.
529.
530.  ************************************************************************
531.  * location of major segments of private address space
532.  ************************************************************************
     # 0801FC00    533.  v_base      equ      top_loc_ram-$3ff    ;base of 1k vector page
     # 08000000    534.  ram_base    equ      top_loc_ram-$1ffff  ;start of evaluation RAM
     # 00000000    535.  rom_base    equ      bas_loc_eprom       ;start of ROM
                   536.
                   537.  ************************************************************************
                   538.  * important addresses in vector page
                   539.  ************************************************************************
     # 0801FC08    540.  v_buserr    equ      v_base+$008         ;(pROBE) bus error
     # 0801FC0C    541.  v_addrerr   equ      v_base+$00C         ;(pROBE) address error
     # 0801FC10    542.  v_ils       equ      v_base+$010         ;(pROBE) illegal instruction
     # 0801FC24    543.  v_trace     equ      v_base+$024         ;(pROBE) trace vector
     # 0801FC80    544.  v_trap0     equ      v_base+$080         ;TRAP 0  - pROBE breakpoints
     # 0801FC74    545.  v_manbrk    equ      v_base+$074         ;(pROBE) manual break switch
                   546.
     # 0801FCAC    547.  v_trap11    equ      v_base+$0AC         ;TRAP 11 - pSOS kernel call
     # 0801FCB0    548.  v_trap12    equ      v_base+$0B0         ;TRAP 12 - pSOS I/O call
     # 0801FCB4    549.  v_trap13    equ      v_base+$0B4         ;TRAP 13 - pSOS ret_i call
                   550.
     # 0801FF00    551.  v_ipris     equ      v_base+$300         ;pRISM startup entry point
     # 0801FCB8    552.  v_trap14    equ      v_base+$0B8         ;TRAP 14 - pRISM service entry
                   553.
     # 0801FC78    554.  v_clock     equ      v_base+$78          ;clock vector (LIRQ 6)
                   555.
                   556.

                   558.  ************************************************************************
                   559.  * Memory map of private ram data area.
                   560.  * Installed immediately below the vector page, which is defined above.
                   561.  ************************************************************************
     # 0801FC00    562.  p_ram_bas   set      v_base
                   563.
                   564.f         ifne    psos_inc        ; if pSOS to be included
                   565.n         endc
                   566.
                   567.f         ifne    prism_inc       ; if pRISM to be included
                   568.n         endc
                   569.
                   570.f         ifne    psos_inc        ; if pSOS to be included
                   571.n         endc
                   572.
     # 0801F400    573.  p_ram_bas   set     p_ram_bas-$800      ;pROBE stack space
     # 0801EC00    574.  p_ram_bas   set     p_ram_bas-$0800     ;pROBE data space
                   575.
                   576.
                   577.f         ifne    prism_inc!psos_inc      ; if pRISM or pSOS included
                   578.n         endc
     # 0801EBF0    579.  p_ram_bas   set      p_ram_bas-$10      ;system data area
     # 0801EBF0    580.  l_p_ram_bas set      (p_ram_bas/4)*4    ;-and on a lword boundary
                   581.
0801EBF0           582.          org     p_ram_bas
                   583.
                   584.f         ifne    psos_inc        ; if pSOS to be included
                   585.n         endc
                   586.
                   587.f         ifne    prism_inc       ; if pRISM to be included
                   588.n         endc
                   589.
                   590.f         ifne    psos_inc        ; if pSOS to be included
                   591.n         endc
                   592.
0801EBF0  <800>    593.              ds.b     $800               ;pROBE stack, lo address
0801F3F0  <0>      594.  ram_prostk  ds.b     $0                 ;pROBE stack, from here down
0801F3F0  <800>    595.  ram_prodata ds.b     $0800              ;pROBE data, from here up
                   596.
                   597.f         ifne    prism_inc!psos_inc      ; if pRISM or pSOS included
```

```
                598.n         endc
0801FBF0  <10>  599.  sys_data    ds.b     $10                ;system data area
                600.
     @ 0801FC00 601.  ram_end     equ      *                  ;end of ram
     @ 00001010 602.  ram_len     equ      ram_end-p_ram_bas  ;length of ram area
                603.
                604.f         ifne    prism_inc       ; if pRISM to be included
                605.n         endc
                606.
                607.f         ifne    prism_inc!psos_inc       ; if pRISM or pSOS included
                608.n         endc
                609.
                610.  ********************************************************************
                611.  * memory map of rom - 64k required.  begins at "rom_base" which is
                612.  * defined above.
                613.  ********************************************************************
     # 00000400 614.  boot_code   equ      rom_base+$400       ;environment boot code offset
     # 00001800 615.  prism_rom   equ      rom_base+$1800      ;pRISM code offset
     # 00005000 616.  psos_rom    equ      rom_base+$5000      ;pSOS code offset
     # 00006700 617.  probe_rom   equ      rom_base+$6700      ;pROBE code offset
     # 00006700 618.  probe_code  equ      probe_rom           ;pROBE code execution address
     # 00009940 619.  probe_size  equ      $9940               ;and size version 3.14
     # 0001003F 620.  p_code_end  equ      probe_code+probe_size-1 ;end of known code
                621.
                622.  ********************************************************************
                623.  * pROBE entry points relative to start of probe
                624.  ********************************************************************
     # 0000670C 625.  re_start    equ      probe_code+$0c     ;startup
     # 00006720 626.  re_brk      equ      probe_code+$20     ;instruction breakpoint
     # 00006724 627.  re_trace    equ      probe_code+$24     ;trace vector
     # 00006728 628.  re_man      equ      probe_code+$28     ;manual break
     # 0000672C 629.  re_exce     equ      probe_code+$2c     ;exception
     # 00006730 630.  re_ba       equ      probe_code+$30     ;bus/address error
                631.
                632.f         ifne    psos_inc        ; if pSOS to be included
                633.n         endc
                635.f         ifne    prism_inc       ; if pRISM to be included
                636.n         endc
                637.
                638.  ********************************************************************
                639.  * Local definitions
                640.  ********************************************************************
                641.  ******************************************************************
                642.  * Boot code definitions.
                643.  ******************************************************************
                644.
     # 00000001 645.  delay_len   equ      1          ;delay on startup to let things settle
                646.                                  ;(loops for this no. of 65534 cycles)
     # 00000004 647.  bn_sta_to   equ      4          ;bit no. of 'time-out' in stacpu reg
     # 00000004 648.  bn_ctl_to   equ      4          ;bit no. of 'time-out' in ctlcpu reg
                649.
     # 000186A0 650.  prism_to    equ      100000          ;time out for pRISM initial.
                651.
                652.  *-----------------------------------------------------------------
                653.  * Codes d'erreur stckes dans d6 pendant la procedure d'init
                654.  *-----------------------------------------------------------------
                655.
     # 0000F1F1 656.  su_be       equ      $f1f1      ;bus error
     # 0000F1F2 657.  su_ae       equ      $f1f2      ;address error
     # 0000F1F3 658.  su_ui       equ      $f1f3      ;interruption/trap non-identifiey
     # 0000F1F4 659.  su_id       equ      $f1f4      ;plusieurs FIC avec meme id
     # 0000F1F5 660.  su_np       equ      $f1f5      ;pROBE entry missing from system conf
     # 0000F1F0 661.  chk_loc_ram equ      $f1f0      ;error while checking local ram
                662.
                663.  * Error codes placed in 'ces_inif+4' during initialisation procedure
     # 0000F102 664.  er_nopr     equ      $f102      ;NOT a light node, but no pROBE present
     # 0000F101 665.  er_toin     equ      $f101      ;timeout pRISM not setup by init node
     # 0000F103 666.  er_noin     equ      $f103      ;ad err accessing init node memory
```

```
                    668.
                    669.  ************************************************************************
                    670.  *                                                                      *
                    671.  *              Initial vector table                                    *
                    672.  *    The FIC8230 has its rom from zero upwards; therefore no           *
                    673.  *    problems with restart vectors 0 & 4                               *
                    674.  *                                                                      *
                    675.  ************************************************************************
00000000            676.              ORG      0
                    677.f             ifne     psos_inc        ; if pSOS to be included
                    678.n             endc
00000004            679.              ORG      4
00000004  00000400 680.               DC.L     BOOT_CODE             ; INITIAL PC ON START-UP
                    681.
00000008  00000690 682.               dc.l     be_st                   ; bus err on startup
0000000C  0000069A 683.               dc.l     ae_st                   ; address error
                    684.
00000010  000006A4 685.               dc.l     ui_st,ui_st,ui_st,ui_st,ui_st,ui_st,ui_st,ui_st
          000006A4
          000006A4
          000006A4
          000006A4
          000006A4
          000006A4
          000006A4
00000030  000006A4 686.               dc.l     ui_st,ui_st,ui_st,ui_st
          000006A4
          000006A4
          000006A4
                    687.
                    688.  ************************************************************************
                    689.  *                                                                      *
                    690.  *            S Y S T E M   A N C H O R     (at org $40)                *
                    691.  *                                                                      *
00000040            692.  sys_anch:
                    693.f          ifne     prism_inc!psos_inc ; if pRISM or pSOS included
                    694.n          endc
                    695.t          ifeq     prism_inc!psos_inc ; not pRISM nor pSOS included
00000040  000006CC 696.           dc.l     sys_con_tab        ;system anchor -> rom version
                    697.n          endc
                    698.  ************************************************************************
                    699.
00000044  000006A4  700.         dc.l     ui_st,ui_st,ui_st,ui_st,ui_st,ui_st,ui_st,ui_st ;$44
          000006A4
          000006A4
          000006A4
          000006A4
          000006A4
          000006A4
          000006A4
00000064  000006A4  701.         dc.l     ui_st,ui_st,ui_st,ui_st,ui_st,ui_st,ui_st,ui_st ;$64
          000006A4
          000006A4
          000006A4
          000006A4
          000006A4
```

```
          000006A4
          000006A4
00000084  000006A4  702.        dc.l    ui_st,ui_st,ui_st,ui_st,ui_st,ui_st,ui_st,ui_st ;$84
          000006A4
          000006A4
          000006A4
          000006A4
          000006A4
          000006A4
          000006A4
          000006A4
000000A4  000006A4  703.        dc.l    ui_st,ui_st,ui_st,ui_st,ui_st,ui_st,ui_st,ui_st ;$a4
          000006A4
          000006A4
          000006A4
          000006A4
          000006A4
          000006A4
          000006A4
          000006A4
000000C4  000006A4  704.        dc.l    ui_st,ui_st,ui_st,ui_st,ui_st,ui_st,ui_st,ui_st ;$c4
          000006A4
          000006A4
          000006A4
          000006A4
          000006A4
          000006A4
          000006A4
          000006A4
000000E4  000006A4  705.        dc.l    ui_st,ui_st                                     ;$e4
          000006A4
                    706.
                    707.
                    708.f         ifne    prism_inc       ; if pRISM to be included
                    709.n         endc
                    710.
                    711.
                    712.
                    713.  ************************************************************************
                    714.  *                                                                      *
                    715.  *           B A S I C   B O O T   C O D E                              *
                    716.  *                                                                      *
                    717.  ************************************************************************
00000400            718.            org     $400
00000400            719.  boot_8230:
00000400  7201      720.            move.l    #delay_len,d1       ; outer loop multiplies inner
00000402            721.  st_loop:
00000402  243C 0000FFFF
                    722.            move.l    #$ffff,d2           ;inner loop of 75535
                    723.
00000408            724.  st_loop1:
00000408  4E71      725.          nop                           ;wait for things to get stable
0000040A  4E71      726.          nop
0000040C  51CA FFFA 727.          dbra      d2,st_loop1
                    728.
00000410  51C9 FFF0 729.          dbra      d1,st_loop
                    730.
                    731.  *-----------------------------------------------------------------------
                    732.  * Initialise the RAM shadowing the control registers
                    733.  *-----------------------------------------------------------------------

00000414  23FC 00000000   734.          move.l    #0,ctlcpu
          20000000
0000041E  23FC 00000000   735.          move.l    #0,ctlvsb
          20000004
00000428  23FC 00000000   736.          move.l    #0,ctlvme1
          20000008
00000432  23FC 00000000   737.          move.l    #0,ctlvme2
          2000000C
```

```
                                738.
                                739.  *-------------------------------------------------------
                                740.  * Establish local RAM coherence for stack and system areas
                                741.  *-------------------------------------------------------
0000043C  4E71                  742.          nop
0000043E  2C3C 0000F1F0         743.          move.l  #chk_loc_ram,d6
00000444  207C 0801EBF0         744.          move.l  #l_p_ram_bas,a0         ; a0 = adresse initial
                                745.  *       Generation de l'array
0000044A  2248                  746.          move.l  a0,a1                   ; a1 = pt d'adresse
0000044C  203C 1F2E3D4C         747.          move.l  #$1f2e3d4c,d0           ; d0 = data
00000452  343C 0503             748.          move.w  #(top_loc_ram-l_p_ram_bas)/4,d2 ; d2 = cnt de passe
                                749.
     @ 00000456                 750.  loop    set     *
00000456  22C0                  751.          move.l  d0,(a1)+                ; load data
00000458  0680 31313131         752.          addi.l  #$31313131,d0           ; constante
0000045E  51CA FFF6             753.          dbf     d2,loop
                                754.
                                755.  *       Relecture et test de l'array
00000462  2248                  756.          move.l  a0,a1                   ; a1 = pt d'adresse
00000464  203C 1F2E3D4C         757.          move.l  #$1f2e3d4c,d0           ; d0 = data
0000046A  343C 0503             758.          move.w  #(top_loc_ram-l_p_ram_bas)/4,d2 ; d2 = cnt de passe
     @ 0000046E                 759.  loop    set     *
0000046E  2219                  760.          move.l  (a1)+,d1                ; recover data
00000470  B280                  761.          cmp.l   d0,d1                   ; test data lu
00000472  6600 024E             762.          bne     init_err                ;...problems
00000476  0680 31313131         763.          addi.l  #$31313131,d0           ; constante
0000047C  51CA FFF0             764.          dbf     d2,loop
00000480                        765.  en_ramt:
                                766.  *-------------------------------------------------------------
                                767.  * Local RAM ok
                                768.  * -construire stack et table des vecteurs; d'abord tous les vecteurs
                                769.  *  sont achemines vers pROBE
                                770.  *-------------------------------------------------------------
00000480  2E7C 0801EBF0         771.          movea.l  #l_p_ram_bas,sp   ; set up our initial stack (isp)
00000486  207C 0000672C         772.          move.l   #re_exce,a0         ;A0 -> pROBE excptn handler
0000048C  227C 0801FC00         773.          move.l   #v_base,a1          ;A1-> vector page
00000492  303C 00FF             774.          move.w   #256-1,d0           ;D0: # vectors to init
00000496                        775.  vec_loop:
00000496  22C8                  776.          move.l   A0,(A1)+            ;patch each vector
00000498  51C8 FFFC             777.          dbra     D0,vec_loop         ; until done
                                778.
                                779.  *-------------------------------------------------------------
                                780.  * -ensuite les vecteurs particuliers pour pROBE
                                781.  *-------------------------------------------------------------
0000049C  23FC 00006730         782.          move.l   #re_ba,v_buserr          ;bus error vector
          0801FC08
000004A6  23FC 00006730         783.          move.l   #re_ba,v_addrerr         ;address error vector
          0801FC0C
000004B0  23FC 00006724         784.          move.l   #re_trace,v_trace        ;trace vector
          0801FC24
000004BA  23FC 00006728         785.          move.l   #re_man,v_manbrk         ;set manual break vector
          0801FC74
000004C4  23FC 00006720         786.          move.l   #re_brk,v_trap0          ;set pROBE brkpt vector
          0801FC80
                                787.
                                788.
                                789.
                                790.f         ifne    psos_inc        ; if pSOS to be included
                                791.n         endc
                                792.
                                793.f         ifne    prism_inc!psos_inc  ; if pRISM or pSOS included
                                794.n         endc
                                795.
                                796.f         ifne    psos_inc        ; if pSOS to be included
                                797.n         endc
                                798.
                                799.
                                800.f         ifne    prism_inc       ; if pRISM to be included
                                801.n         endc
                                802.
```

```
                        803.  *-------------------------------------------------------------
                        804.  *  Switch the vector base reg to ram
                        805.  *-------------------------------------------------------------
000004CE  203C 0801FC00 806.          move.l  #v_base,d0
000004D4  4E7B 0801     807.          movec   d0,vbr
                        808.
                        809.  *-------------------------------------------------------------
                        810.  * Initialise and clear the FIFOs
                        811.  *-------------------------------------------------------------
000004D8  207C 20000800 812.          move.l  #bas_fifo,a0
000004DE  7000          813.          move.l  #0,d0
000004E0  323C 0013     814.          move.w  #20-1,d1
000004E4                815.  clr_fifo:
000004E4  20C0          816.          move.l  d0,(a0)+
000004E6  51C9 FFFC     817.          dbra    d1,clr_fifo
                        818.
                        819.  *-------------------------------------------------------------
                        820.  * Initialise the 8570
                        821.  *-------------------------------------------------------------
                        822.
000004EA  13FC 0000     823.          move.b  #$00,r00_dp8570   ; Page sel. 0, Register sel 0
          15000000
000004F2  0839 0006     824.          btst.b  #$6,r03_dp8570 ; Test if power-up or re-start, to avoid
          15000003
000004FA  6700 0036     825.          beq     en_in_8570      ; re-initialising chip unnecessarily
                        826.
                        827.                                  ; Initial power-up or oscillator failed
000004FE  13FC 0000     828.          move.b  #$0,r03_dp8570 ; Set test mode 0, battery standby mode
          15000003
                        829.
00000506  41F9 000005BA 830.          lea     init_rtc_val,a0   ; Move initial clock values into
0000050C  227C 15000005 831.          move.l  #r05_dp8570,a1    ; chip registers
00000512  7019          832.          move.l  #$19,d0
    @ 00000514          833.  loop:   set     *
00000514  12D8          834.          move.b   (a0)+,(a1)+
00000516  51C8 FFFC     835.          dbra    d0,loop
                        836.
                        837.
                        838.  *       Real Time mode reg
                        839.
0000051A  13FC 0040     840.          move.b  #$40,r00_dp8570   ; Page sel 0, Reg sel 1
          15000000
00000522  13FC 0008     841.          move.b  #$08,r01_dp8570   ; Leap year current year, 24 hr mode
          15000001
                        842.                                  ; alarm,periodic & PF disabledin s/by ,
                        843.                                  ; timers disabled in s/by, 32.768Khz
                        844.
0000052A  08F9 0003     845.          bset    #$3,r01_dp8570    ; then start the clock
          15000001
                        846.
00000532                847.  en_in_8570:
                        848.  *      Here we could read the time of preceding power fail
                        849.  *      from the time-save ram, befor re-enabling the time-save.
                        850.
                        851.  *      Interrupt routing reg
00000532  13FC 0000     852.          move.b  #$00,r00_dp8570   ; Page sel. 0, Register sel 0
          15000000
0000053A  13FC 00A0     853.          move.b  #$a0,r04_dp8570   ; All ints routed to INT pin,
          15000004
                        854.  ;                               ; enable 480us delay, enable time save
                        855.
                        856.
                        857.  *      Output mode register
                        858.
00000542  13FC 0040     859.          move.b  #$40,r00_dp8570   ; Page sel 0, Reg sel 1
          15000000
0000054A  13FC 008A     860.          move.b  #$8A,r02_dp8570   ; T1 push-pull, active lo
          15000002
                        861.                                  ; INTpush-pull, active lo
                        862.                                  ; MFO    drain, active lo
                        863.                                  ; MFO as oscillator
                        864.
                        865.  *      Interrupt ctl reg 0
00000552  13FC 0000     866.          move.b  #$00,r03_dp8570   ; periodic ints all disabled
          15000003
                        867.                                  ; timer 0 ints disabled
                        868.
```

```
                          869.  *       Interrupt ctl reg 1
0000055A  13FC 0080       870.          move.b  #$80,r04_dp8570   ; power-fail detect enabled
          15000004
                          871.                                    ; compare interrupt disabled
                          872.                                    ; alarm interrupt disabled
                          873.
                          874.
                          875.  *       Test for battery low in 8570
00000562  13FC 0000       876.          move.b  #$00,r00_dp8570   ; Page sel. 0, Register sel 0
          15000000
0000056A  207C 000005D4   877.          move.l  #msg_nok_batt,a0  ;assume battery not ok, then
00000570  0839 0006       878.          btst.b  #$6,r04_dp8570    ;check battery low status
          15000004
00000578  6600 0008       879.          bne     batt_nok          ;indeed
                          880.
0000057C  207C 000005F5   881.          move.l  #msg_ok_batt,a0
00000582                  882.  batt_nok:
                          883.
                          884.  *       Set timer 1
00000582  13FC 0000       885.          move.b  #$00,r12_dp8570   ; Timer 1 used for FIC timeout; 04 gives
          15000012
0000058A  13FC 0004       886.          move.b  #$04,r11_dp8570   ; 16us timeout for 100nano clock.
          15000011
                          887.
                          888.  *       Timer 1 control reg.
00000592  13FC 0004       889.          move.b  #$04,r02_dp8570   ; square wave, extern srce
          15000002
0000059A  08F9 0000       890.          bset.b  #0,r02_dp8570     ;..and start timer
          15000002
                          891.
                          892.  *       Timer 0 control reg.
000005A2  13FC 0000       893.          move.b  #$00,r01_dp8570   ; timer stopped, mode 0, external srce
          15000001
                          894.
                          895.
                          896.
                          897.  *       Display battery state message
                          898.  *
000005AA                  899.  dis_bat_mes:
000005AA  6100 01B0       900.          bsr     con_init          ; Initialise the z8030
000005AE                  901.  dis_bat_nxc:
000005AE  1E18            902.          move.b  (a0)+,d7
000005B0  6700 005A       903.          beq     en_batt_tes       ; String ends with null
000005B4  6100 02A8       904.          bsr     con_out
000005B8  60 F4           905.          bra     dis_bat_nxc
                          906.
                          907.  *       Values moved into page 0 registers to initialise time/date
000005BA                  908.  init_rtc_val:
000005BA  00 00 00 00     909.          dc.b    0,0,0,0                 ;0:0:0:0 (1/100s:ss:mm:hh
000005BE  01 04 88 92 00  910.          dc.b    $01,$4,$88,$92,$0,$5    ;1-apr-88, day 92,friday
          05
000005C4  00 00 00 00     911.          dc.b    0,0,0,0                 ;timer 0,1 zeroes
000005C8  00 00 00 00 00  912.          dc.b    0,0,0,0,0,0             ;compare ram zeroed
          00
000005CE  00 00 00 00 00  913.          dc.b    0,0,0,0,0               ;'save' ram zeroed
000005D3  00              914.          dc.b    0                       ;ram extra word
                          915.
000005D4                  916.  msg_nok_batt:
000005D4  0D 0A 42 61 74  917.          dc.b    $d,$a,'Battery voltage low--BEWARE!',$d,$a,0
          74 65 72 79 20
          76 6F 6C 74 61
          67 65 20 6C 6F
          77 2D 2D 42 45
          57 41 52 45 21
          0D 0A 00
000005F5                  918.  msg_ok_batt:
000005F5  0D 0A 42 61 74  919.          dc.b    $d,$a,'Battery voltage OK',$d,$a,0
          74 65 72 79 20
          76 6F 6C 74 61
```

```
          67 65 20 4F 4B
          0D 0A 00
                        920.
                        921.  *       Wait until last character hasbeen sent from serial port,
                        922.  *       otherwise it will be corrupted when we init pROBE
0000060C                923.  en_batt_tes:
0000060C  1C39 10000021 924.          move.b  ch1_r0_z8030,d6         ;get chA status,
00000612  0806 0002     925.          btst  #2,d6                     ;and test for tx buff empty
00000616  67 F4         926.          beq.s   en_batt_tes             ;no, wait until it is
                        927.

                        929.
00000618  13FC 0001     930.          move.b  #$1,mir_z8036     ; Reset Z8036
          11000000
00000620  6100 006A     931.          bsr     wait_zilog
00000624  13FC 0000     932.          move.b  #$0,mir_z8036     ; End reset Z8036
          11000000
0000062C  6100 005E     933.          bsr     wait_zilog
00000630  13FC 0002     934.          move.b  #$02,mir_z8036    ; Set Right justify (A0 from AD0)
          11000000
00000638  6100 0052     935.          bsr     wait_zilog
                        936.                                    ; ... for byte alignement
                        937.
                        938.  *---------------------------------------------------------------------
                        939.  * Initialise SCB68155 Interrupt controller
                        940.  *   for Front Panel abort (LIRQ5-level 5) and
                        941.  *   (power fail from DP8570 (LIRQ6-level 6) not used here)
                        942.  *---------------------------------------------------------------------
                        943.
0000063C  13FC 0006     944.          move.b  #$6,r0_68155            ; select sub-register 6 (LIRQ6)
          16000000
00000644  13FC 0006     945.          move.b  #$6,r1_68155            ; set edge trig, act lo,local vector
          16000002
0000064C  13FC 0005     946.          move.b  #$5,r0_68155            ; select sub-register 5 (LIRQ5)
          16000000
00000654  13FC 0007      947.           move.b  #$7,r1_68155             ; set edge trig, act hi, local
vector
          16000002
0000065C  13FC 0018     948.          move.b  #$18,r2_68155           ; base vector = $18
          16000004
                        949.                                    ; so abort vectors to 1d or $74
00000664  13FC 00A1     950.          move.b  #$A1,r3_68155           ; enable NMI & level 5
          16000006
                        951.
                        952.  *---------------------------------------------------------------------
                        953.  * Enable cache register
                        954.  *---------------------------------------------------------------------
0000066C  7001          955.          move.l  #$1,d0
0000066E  4E7B 0002     956.          movec   d0,cacr
                        957.
                        958.
                        959.  ********************************************************************
                        960.  *---------------------------------------------------------------------
                        961.  *  Fin de la phase d'initialisation generale
                        962.  *---------------------------------------------------------------------
                        963.  ********************************************************************
                        964.

00000672  2A78 0040     966.          movea.l sys_anch,a5         ;Set up the system anchor address
                        967.f         ifne    vers3
                        968.n         endc            ;ends 'prism or vers3'
                        969.
```

```
                        971.  ************************************************************************
                        972.  *  If pROBE present, pass control to pROBE.
                        973.  ************************************************************************
                        974.
                        975.
00000676                976.  ret_p:
00000676  4AAD 0018      977.          tst.l   $18(a5)                 ; probe code addr in sys_con_tab
0000067A  6700 003C      978.          beq     no_probe
0000067E  23FC 00000000  979.          move.l  #0,ctlcpu               ; trigger for joel's analyser
          20000000
00000688  4EF8 670C      980.          JMP      re_start
                        981.

                        983.  ********************************************************************
                        984.  * Zilog 8036 chip becomes VERY upset if successive accesses are
                        985.  * too close together....so calling this subroutine between accesses
                        986.  * keeps it happy.
                        987.  ********************************************************************
                        988.
0000068C                989.  wait_zilog:
0000068C  4E71          990.          nop
0000068E  4E75          991.          rts
                        992.
                        993.  ********************************_***********************************
                        994.  *  Recuperation (!) des erreurs du materiel a l'initialisation.      *
                        995.  *                                                                    *
                        996.  *  Al'initialisation la table des vecteurs est situee a l'adresse 0  *
                        997.  *  dans EPROM.  Les vecteurs pointent vers les routines suivants.    *
                        998.  *  Avant de tester si la RAM locale est utilisable et d'etablir le   *
                        999.  *  stach et la table des vects en RAM, les erreurs  eventuels sont   *
                       1000.  *  traites comme catastrophiques - alors l'uc reste dans un etat     *
                       1001.  *  qui sera detectionne par le J11 comme non-operationnel.           *
                       1002.  *  Le code dans d6 servira eventuellement a identifier l'erreur.     *
                       1003.  ************************************************************************
                       1004.
00000690  2C3C 0000F1F1 1005.  be_st:  move.l  #su_be,d6               ; bus error
00000696  6000 002A     1006.          bra     init_err
                       1007.
0000069A  2C3C 0000F1F2 1008.  ae_st:  move.l  #su_ae,d6               ; address error
000006A0  6000 0020     1009.          bra     init_err
                       1010.
000006A4  2C3C 0000F1F3 1011.  ui_st:  move.l  #su_ui,d6               ;unidentified interrupt
000006AA  6000 0016     1012.          bra     init_err
                       1013.
000006AE  2C3C 0000F1F4 1014.  dup_id: move.l  #su_id,d6                  ;duplicate cpu id - 2 or more
000006B4  6000 000C     1015.          bra     init_err                ; FICs with same id selected
                       1016.
000006B8                1017.  no_probe:
000006B8  2C3C 0000F1F5 1018.          move.l  #su_np,d6               ;no enrty for pROBE in config tab.
000006BE  6000 0002     1019.          bra     init_err
                       1020.
                       1021.  *-----------------------------------------------------------------
                       1022.  * Copy error code to first word of fifo 0 buffer, so we can read
                       1023.  * it from any other processor on VME
                       1024.  * -then loop forever.
                       1025.  *-----------------------------------------------------------------
000006C2                1026.  init_err:
000006C2  23C6 20001000 1027.          move.l  d6,$20001000
000006C8                1028.  init_err1:
000006C8  4E71          1029.          nop
000006CA  60 FC         1030.          bra     init_err1               ; boucle pour l'eternite
```

```
                    1032.f          ifne    prism_inc!psos_inc      ; if pRISM or pSOS included
                    1033.n          endc
                    1034.
                    1035.  ********************************************************************
                    1036.  *  SYSTEM CONFIGURATION TABLE
                    1037.  *  a normal node has entries for probe, pSOS and prism
                    1038.  *                              or pROBE and psos
                    1039.  *                              or pROBE and prism
                    1040.  *                              or pROBE only
                    1041.  *  a remote node has entries for all of those configs too
                    1042.  *  BUT a lighht node has no probe
                    1043.  ********************************************************************
000006CC            1044.  sys_con_tab:
000006CC  0801FBF0  1045.              dc.l    sys_data                ;sys data area
000006D0  00000002  1046.              dc.l    2                       ;cpu type (68020)
000006D4  00000001  1047.              dc.l    1                       ;default-pROBE starts first
000006D8  00000000  1048.              dc.l    0                       ;reserved
                    1049.
                    1050.f          ifne    prism_inc       ; if pRISM to be included
                    1051.n          endc
                    1052.t          ifeq    prism_inc       ; if pRISM not to be included
000006DC  00000000  1053.              dc.l    0
000006E0  00000000  1054.              dc.l    0
                    1055.n          endc
                    1056.
                    1057.f          ifne    prism_inc!psos_inc      ; if pRISM or pSOS included
                    1058.n          endc
                    1059.t          ifeq    prism_inc!psos_inc      ; not pRISM nor pSOS included
000006E4  0000070C  1060.              dc.l    probe_con_tab           ;pROBE conf tablein eprom
000006E8  00006700  1061.              dc.l    probe_code              ;start of pROBE code
                    1062.n          endc
                    1063.
                    1064.f          ifne    psos_inc                ; if pSOS to be included
                    1065.n          endc
                    1066.t          ifeq    psos_inc                ; if pSOS not to be included
000006EC  00000000  1067.              dc.l    0
000006F0  00000000  1068.              dc.l    0
                    1069.n          endc
000006F4  00000000  1070.              dc.l    0,0,0,0,0,0
          00000000
          00000000
          00000000
          00000000
          00000000
                    1071.
                    1072.  ********************************************************************
                    1073.  * pROBE configuration table
                    1074.  ********************************************************************
0000070C            1075.  probe_con_tab:
0000070C  00006700  1076.              dc.l    probe_code              ;pROBE code
00000710  0801F3F0  1077.              dc.l    ram_prodata             ;pROBE stack & static data
00000714  0000075C  1078.              dc.l    con_init                ;serial port initialisation
00000718  00000876  1079.              dc.l    con_sts                 ;console status
0000071C  000008EC  1080.              dc.l    con_in                  ;console input
00000720  0000085E  1081.              dc.l    con_out                 :console output
00000724  0000090C  1082.              dc.l    hst_sts                 ;host status
00000728  00000982  1083.              dc.l    hst_in                  ;host input
0000072C  000008F4  1084.              dc.l    hst_out                 ;host output
00000730  00000000  1085.              dc.l    0                       ;unused
00000734  4E40      1086.              dc.w    $4E40                   ;breakpoint opcode (trap 0)
00000736  0004      1087.              dc.w    4                       ;pROBE interrupt level
00000738  0001      1088.              dc.w    1                       ;trace vector refresh (on)
0000073A  0000      1089.              dc.w    0                       ;unused
0000073C  00000000  1090.              dc.l    0                       ;sym to val procedure
00000740  00000000  1091.              dc.l    0                       ;val to sym procedure
00000744  00000000  1092.              dc.l    0                       ;unrecog. command procedure
00000748  0000      1093.              dc.w    0                       ;start mode = normal
0000074A  0000      1094.              dc.w    0                       ;reserved
0000074C  00000000  1095.              dc.l    0,0,0,0                 ;reserved
          00000000
          00000000
          00000000
                    1096.
    # 00000090      1097.  copy_len    equ     *-sys_con_tab
                    1098.
                    1099.
                    1100.f          ifne    psos_inc        ; if pSOS to be included
```

```
                        1101.n          endc
                        1102.
                        1103.
                        1104.f          ifne    prism_inc       ; if pRISM to be included
                        1105.n          endc
                        1107.  ************************************************************************
                        1108.  *                                                                      *
                        1109.  *     p R O B E   S E R I A L   D R I V E R S   F O R   8 0 3 0        *
                        1110.  *                                                                      *
                        1111.  ************************************************************************
                        1112.
     # 00000000         1113.  rbfull  equ     0               ;recv buffer full  (reg 0)
     # 00000005         1114.  overrun equ     5               ;character overrun (reg 1)
     # 00000006         1115.  framerr equ     6               ;framing error     (reg 1)
     # 00000007         1116.  break   equ     7               ;break/abort detect(reg 0)
                        1117.
                        1118.  ************************************************************************
                        1119.  * Serial port initialisation routine.
                        1120.  ************************************************************************
0000075C                1121.  con_init:
0000075C                1122.  hst_init:
                        1123.
                        1124.  ************************************************************************
                        1125.  * z8030 channel a
                        1126.  ************************************************************************
                        1127.
0000075C  13FC 00C0     1128.          move.b #$c0,ch1_r9_z8030        ;reset chip
          10000033
00000764  13FC 00C0     1129.          move.b #$c0,ch1_r9_z8030        ;reset chip for the second time!
          10000033
0000076C  13FC 0044     1130.          move.b #$44,ch1_r4_z8030        ;16x clock,1 stop,no par
          10000029
00000774  13FC 0068     1131.          move.b #$68,ch1_r5_z8030        ;8 bits, Tx enable
          1000002B
0000077C  13FC 00C1     1132.          move.b #$C1,ch1_r3_z8030        ;8 bits, Rx enable
          10000027
00000784  13FC 0000     1133.          move.b #$00,ch1_r9_z8030        ;
          10000033
0000078C  13FC 0000     1134.          move.b #$00,ch1_r1_z8030        ;no interrupts
          10000023
00000794  13FC 0000     1135.          move.b #$00,ch1_r7_z8030        ;synch
          1000002F
0000079C  13FC 0000     1136.          move.b #$00,ch1_rA_z8030        ;
          10000035
000007A4  13FC 0056     1137.          move.b #$56,ch1_rB_z8030        ;clock to Tx Rx TrC
          10000037
000007AC  13FC 000E     1138.          move.b #$0E,ch1_rC_z8030        ;time const lo
          10000039
000007B4  13FC 0000     1139.          move.b #$00,ch1_rD_z8030        ;time const hi
          1000003B
000007BC  13FC 0003     1140.          move.b #$03,ch1_rE_z8030        ;enable baud rate
          1000003D
000007C4  13FC 0000     1141.          move.b #$00,ch1_rF_z8030        ;clear and
          1000003F
000007CC  13FC 0080     1142.          move.b #$80,ch1_rF_z8030        ;enable break/abort
          1000003F
                        1143.
                        1144.
                        1145.  ************************************************************************
                        1146.  * z8030 channel b
                        1147.  ************************************************************************
000007D4  13FC 0044     1148.          move.b #$44,ch2_r4_z8030        ;16x clock,1 stop,no par
          10000008
000007DC  13FC 0068     1149.          move.b #$68,ch2_r5_z8030        ;8 bits, Tx enable
          1000000A
000007E4  13FC 00C1     1150.          move.b #$C1,ch2_r3_z8030        ;8 bits, Rx enable
          10000006
000007EC  13FC 0000     1151.          move.b #$00,ch2_r9_z8030        ;
          10000012
000007F4  13FC 0000     1152.          move.b #$00,ch2_r1_z8030        ;no interrupts
          10000002
000007FC  13FC 0000     1153.          move.b #$00,ch2_r7_z8030        ;synch
          1000000E
00000804  13FC 0000     1154.          move.b #$00,ch2_rA_z8030        ;
          10000014
0000080C  13FC 0056     1155.          move.b #$56,ch2_rB_z8030        ;clock to Tx Rx TrC
          10000016
```

```
00000814  13FC 000E      1156.          move.b #$0E,ch2_rC_z8030        ;time const lo
          10000018
0000081C  13FC 0000      1157.          move.b #$00,ch2_rD_z8030        ;time const hi
          1000001A
00000824  13FC 0003      1158.          move.b #$03,ch2_rE_z8030        ;enable baud rate
          1000001C
0000082C  13FC 0000      1159.          move.b #$00,ch2_rF_z8030        ;clear and
          1000001E
00000834  13FC 0080      1160.          move.b #$80,ch2_rF_z8030        ;enable break/abort
          1000001E
                         1161.
0000083C  13FC 00C0      1162.          move.b  #$c0,ch1_r0_z8030       ;reset Tx underrun/eom
          10000021
00000844  13FC 00C0      1163.          move.b  #$c0,ch2_r0_z8030       ;reset Tx underrun/eom
          10000000
0000084C  13FC 0010      1164.          move.b  #$10,ch1_r0_z8030       ;reset Error condition
          10000021
00000854  13FC 0010      1165.          move.b  #$10,ch2_r0_z8030       ;reset Error condition
          10000000
                         1166.
0000085C  4E75           1167.          rts                             ;THAT'S ALL THERE IS!
                         1168.
                         1169.  ********************************************************************
                         1170.  * Serial port output routine. Note that we poll status BEFORE sending,
                         1171.  * rather than after, in order to save time.
                         1172.  *       Input: d7 = character to display
                         1173.  *       Output: -
                         1174.  ********************************************************************
0000085E                 1175.  con_out:
0000085E  2F00           1176.          move.l  d0,-(sp)
                         1177.
00000860                 1178.  con_out_w:
00000860  1039 10000021  1179.          move.b  ch1_r0_z8030,d0         ;get chA status,
00000866  0800 0002      1180.          btst  #2,d0                     ;and test for tx buff
0000086A  67 F4          1181.          beq.s   con_out_w               ;no, wait until it is
0000086C  13C7 10000031  1182.          move.b  d7,ch1_r8_z8030         ;send the char
                         1183.
00000872  201F           1184.          move.l  (sp)+,d0
00000874  4E75           1185.          rts
                         1186.
                         1187.  ********************************************************************
                         1188.  * Serial port status routine.
                         1189.  *       Input:  -
                         1190.  *       Output: 0 for no character,
                         1191.  *               1 if character available,
                         1192.  *               2 if break character received
                         1193.  ********************************************************************
00000876                 1194.  con_sts:
00000876  48E7 80C0      1195.          movem.l d0/a0/a1,-(sp)
0000087A                 1196.  con_sts_1:
0000087A  1039 10000021  1197.          move.b  ch1_r0_z8030,d0
00000880  4287           1198.          clr.l   d7
00000882  0800 0000      1199.          btst  #rbfull,d0                ;character recvd?
00000886  67 5E          1200.          beq.s   con_sts_4               ;nope, return
                         1201.
00000888  1039 10000023  1202.          move.b  ch1_r1_z8030,d0         ;read status reg 1
0000088E  0800 0005      1203.          btst  #overrun,d0               ;test buffer over-run,
00000892  67 04          1204.          beq.s   con_sts_2               ;char ok
00000894  61 56          1205.          bsr.s   con_in                  ;nok  forget char, and get another
00000896  60 E2          1206.          bra.s   con_sts_1
                         1207.
00000898                 1208.  con_sts_2:
00000898  7E01           1209.          moveq.l #1,d7                   ;return value '1=char recvd'
0000089A  1039 10000021  1210.          move.b  ch1_r0_z8030,d0         ;get the status
000008A0  0800 0007      1211.          btst  #break,d0                 ;and check for break
000008A4  67 40          1212.          beq.s   con_sts_4               ;not break
000008A6                 1213.  con_sts_3:
000008A6  13FC 0000      1214.          move.b #$00,ch1_rF_z8030        ;clear and
          1000003F
000008AE  13FC 0080      1215.          move.b #$80,ch1_rF_z8030        ;enable break/abort
          1000003F
000008B6  13FC 0030      1216.          move.b  #$30,ch1_r0_z8030       ;error reset
          10000021
000008BE  13FC 0010      1217.          move.b  #$10,ch1_r0_z8030       ;reset  status interrupts
          10000021
000008C6  1039 10000023  1218.          move.b  ch1_r1_z8030,d0         ;clear break status with 3 reads
000008CC  1039 10000023  1219.          move.b  ch1_r1_z8030,d0
```

```
000008D2  1039 10000023  1220.          move.b  ch1_r1_z8030,d0
000008D8  1039 10000021  1221.          move.b  ch1_r0_z8030,d0          ;check status again
000008DE  0800 0007      1222.          btst  #break,d0
000008E2  66 C2          1223.          bne     con_sts_3                ;still there, keep trying to clear
000008E4  7E02           1224.          moveq.l #2,d7                    ;else set status return
000008E6                 1225.  con_sts_4:
000008E6  4CDF 0301      1226.          movem.l (sp)+,d0/a0/a1
000008EA  4E75           1227.          rts
                         1228.
                         1229.  ************************************************************************
                         1230.  * Serial port input routine. This should be called only AFTER
                         1231.  * CON_STS() has returned a value of 1.
                         1232.  *       input:  -
                         1233.  *       output: d7 = character read
                         1234.  ************************************************************************
000008EC                 1235.  con_in:
000008EC  1E39 10000031  1236.            move.b   ch1_r8_z8030,D7
000008F2  4E75           1237.            rts
                         1238.
                         1239.  ************************************************************************
                         1240.  * Host port output routine. Note that we poll status BEFORE sending,
                         1241.  * rather than after, in order to save time.
                         1242.  *       Input: d7 = character to send
                         1243.  *       Output: -
                         1244.  ************************************************************************
000008F4                 1245.  hst_out:
000008F4  2F00           1246.          move.l  d0,-(sp)
                         1247.
000008F6                 1248.  hst_out_w:
000008F6  1039 10000000  1249.          move.b  ch2_r0_z8030,d0          ;get chB status,
000008FC  0800 0002      1250.          btst  #2,d0                      ;and test for tx buff
00000900  67 F4          1251.          beq.s   hst_out_w                ;no, wait until it is
00000902  13C7 10000010  1252.          move.b  d7,ch2_r8_z8030          ;send the char
                         1253.
00000908  201F           1254.          move.l  (sp)+,d0
0000090A  4E75           1255.          rts
                         1256.
                         1257.  ************************************************************************
                         1258.  * Host port status routine.
                         1259.  *       Input:  -
                         1260.  *       Output: 0 for no character,
                         1261.  *               1 if character available,
                         1262.  *               2 if break character received
                         1263.  ************************************************************************
0000090C                 1264.  hst_sts:
0000090C  48E7 80C0      1265.          movem.l d0/a0/a1,-(sp)
00000910                 1266.  hst_sts_1:
00000910  1039 10000000  1267.          move.b  ch2_r0_z8030,d0
00000916  4287           1268.          clr.l   d7
00000918  0800 0000      1269.          btst  #rbfull,d0                 ;character recvd?
0000091C  67 5E          1270.          beq.s   hst_sts_4                ;nope, return
                         1271.
0000091E  1039 10000002  1272.          move.b  ch2_r1_z8030,d0          ;read status reg 1
00000924  0800 0005      1273.          btst  #overrun,d0                ;test buffer over-run,
00000928  67 04          1274.          beq.s   hst_sts_2                ;char ok
0000092A  61 56          1275.          bsr.s   hst_in                   ;nok; forget char, and get another
0000092C  60 E2          1276.          bra.s   hst_sts_1
                         1277.
0000092E                 1278.  hst_sts_2:
0000092E  7E01           1279.          moveq.l #1,d7                    ;return value '1=char recvd'
00000930  1039 10000000  1280.          move.b  ch2_r0_z8030,d0          ;get the status
00000936  0800 0007      1281.          btst  #break,d0                  ;and check for break
0000093A  67 40          1282.          beq.s   hst_sts_4                ;not break
0000093C                 1283.  hst_sts_3:
0000093C  13FC 0000      1284.          move.b #$00,ch2_rF_z8030         ;clear and
          1000001E
00000944  13FC 0080      1285.          move.b #$80,ch2_rF_z8030         ;enable break/abort
          1000001E
0000094C  13FC 0030      1286.          move.b  #$30,ch2_r0_z8030        ;error reset
          10000000
00000954  13FC 0010      1287.          move.b  #$10,ch2_r0_z8030        ;reset  status interrupts
          10000000
0000095C  1039 10000002  1288.          move.b  ch2_r1_z8030,d0          ;clear break status with 3 reads
00000962  1039 10000002  1289.          move.b  ch2_r1_z8030,d0
00000968  1039 10000002  1290.          move.b  ch2_r1_z8030,d0
```

```
0000096E  1039 10000000  1291.          move.b  ch2_r0_z8030,d0        ;check status again
00000974  0800 0007      1292.          btst  #break,d0
00000978  66 C2          1293.          bne     hst_sts_3              ;still there, keep trying to clear
0000097A  7E02           1294.          moveq.l #2,d7                  ;else set status return
0000097C                 1295.  hst_sts_4:
0000097C  4CDF 0301      1296.          movem.l (sp)+,d0/a0/a1
00000980  4E75           1297.          rts
                         1298.
                         1299.  ************************************************************************
                         1300.  * Host port input routine. This should be called only AFTER
                         1301.  * CON_STS() has returned a value of 1.
                         1302.  *       input:  -
                         1303.  *       output: d7 = character read
                         1304.  ************************************************************************
00000982                 1305.  hst_in:
00000982  1E39 10000010  1306.            move.b   ch2_r8_z8030,d7
00000988  4E75           1307.            rts
                         1308.

                         1310.f          ifne    psos_inc        ; if pSOS to be included
                         1311.n          endc
                         1312.
                         1313.
                         1314.~ ...M68K D2.1M  Oct 3, 1986   ...Run on Dec 19, 1988  15:43:14
0000098A                 1315.              end
    0 Errors
```

Artisan Technology Group is an independent supplier of quality pre-owned equipment

Gold-standard solutions

Extend the life of your critical industrial, commercial, and military systems with our superior service and support.

We buy equipment

Planning to upgrade your current equipment? Have surplus equipment taking up shelf space? We'll give it a new home.

Learn more!

Visit us at **artisantg.com** for more info on price quotes, drivers, technical specifications, manuals, and documentation.

Artisan Scientific Corporation dba Artisan Technology Group is not an affiliate, representative, or authorized distributor for any manufacturer listed herein.

We're here to make your life easier. How can we help you today?

(217) 352-9330 | sales@artisantg.com | artisantg.com

ARTISAN® TECHNOLOGY GROUP