A MULTI-PROCESSOR SLAVE PERIPHERAL CONTROLLER

by

J.F. Harvey

Approved for public release.

Distribution Statement
Approved for public release
Distribution Unlimited

(C) COMMONWEALTH OF AUSTRALIA 1986

NOVEMBER 1986
A MULTI-PROCESSOR SLAVE PERIPHERAL CONTROLLER

by

J.F. Harvey

SUMMARY

A system providing multiple slave co-processors operating independently, yet in parallel upon the same VME bus, all under the control of a master processor is described.

Data are passed unidirectionally from master to slave enabling rapid processing of complex algorithms for the control of external peripherals.
CONTENTS

1. INTRODUCTION .............................. 1
2. DESIGN PHILOSOPHY ......................... 2
3. HARDWARE ................................. 2
   3.1. General .............................. 2
   3.2. Bus Communication Master to Slave ... 4
   3.3. SPC Processor Circuitry ............... 5
      3.3.1. Processor ...................... 5
      3.3.2. Watch Dog Timer ............... 6
      3.3.3. Memory ........................ 6
      3.3.4. PIT ............................ 7
      3.3.5. SPC Interrupts ................. 8
      3.3.6. External I/O Buffering ......... 8
      3.3.7. SPC System Reset ............... 9
      3.3.8. SPC Clock ...................... 9
4. SOFTWARE ................................... 9
   4.1. General .............................. 9
   4.2. Master Utility Programs ............... 10
   4.3. Master Memory to Slave Memory Transfer 10
   4.4. Slave Memory to Master Memory Transfer 11
   4.5. Execution of Program with Slave Memory 11
5. CONCLUSION ................................ 12

ACKNOWLEDGEMENTS

APPENDIX 1

APPENDIX 2

TABLE

FIGURES

DOCUMENT CONTROL DATA
1. INTRODUCTION

An intelligent peripheral controller is described. The controller operates as a slave processor in a multiple processor environment.

The system utilizes Motorola MC68000 microprocessors operating on VME bus (Reference 1) and Eurocard construction. Each slave processor is totally self contained with individual bus buffering, memory and input/output.

Multiple slave processors connect to a common VME bus operating under the control of a master microprocessor. The master may operate under its own resident program or under the direct control of an external host computer. Passing of programs (algorithms) and exchange of data to the slave are controlled by the master processor. However, once the slaves are set up they operate independently of each other and the master until again interrupted. Each slave, although capable of independent operation, may operate as a synchronous or asynchronous system under hand shake control with the master or other slaves.

The system was originally designed to control high speed (10,000 step/second) stepper motors to position a six degrees of freedom, hot wire anemometer probe. The application for the probe was in flow measurement around a model in either the low speed or transonic wind tunnels at the Aeronautical Research Laboratories (ARL).

Both hardware and software for the system is described. The memorandum concludes with suggestions concerning alternative uses for the system.

Reference 1 VMEBUS Specification Manual Motorola MVMEBS/D1 1982
2. DESIGN PHILOSOPHY

Before describing the operation of the system in detail, it is useful to set out the general design philosophy.

(i) The system is controllable from a host computer via a RS232C serial interface.

(ii) Each slave processor operates independently upon complex positioning algorithms contained within their memory.

(iii) Synchronous and asynchronous hand-shake control between each slave is provided.

(iv) Speed of operation is limited by the driven devices and not the software execution time.

(v) Developmental software is down-line loadable from the host to the master and transferrable from master to a selected slave for performance evaluation.

3. HARDWARE

3.1. General

The Motorola MC68000 microprocessor is a high performance 16 bit data bus processor with 16 registers of 32 bit width, capable of directly addressing 16 Mega bytes of memory. This processor, with VME bus and Eurocard hardware, has been selected by ARL as a preferred system for in-house electronic instrumentation.
A variety of cards have been designed and constructed at ARL to provide building blocks for data acquisition and control systems.

The master processor section of the system requires a minimum of three cards. These are a MC68000L8 processor, a combined RAM/ROM memory and a dual serial RS232C interface (Figure 1). The slave processors were required to fit the same chassis hardware, so are VME bus compatible Eurocards. The amount of circuitry required for each slave was too great to fit onto a single Eurocard. Therefore, a piggy-back arrangement of two cards (Fig. 2) was selected and is known as a Slave Peripheral Controller (SPC). The main card contains a MC 68000 ZB10 microprocessor, the VME bus interface, the watch-dog timer, the independent slave system clock, memory and input/output address decoding (Fig. 3). The piggy-backed card contains 8 kbytes of RAM, 8kbytes of ROM, the Parallel Interface Timer (PIT MC68230-10) and the output buffering circuit.

An ARL designed 19 slot VME backplane accepts eight SPC's and three cards for the main processor (Fig. 1). Each SPC occupies two card slots of the VME back plane. If additional master processor cards are required, such as extra memory or an arithmetic processor the number of SPC's to a 19 slot VME back plane must be reduced.

For control of the six degrees of freedom hot wire anemometer probe the number of SPC's is six, thus permitting a maximum of seven master processor cards to be connected to the back plane. Smaller or larger systems may be assembled depending upon the task requirements, electrical loading and mechanical constraints of the VME bus system.

A listing of all ARL drawing numbers associated with the SPC board is provided in Table 1.
3.2. Bus Communication Master to Slave

There is no direct interconnection between the VME bus and the internal SPC bus. All communication between the two bus's is achieved by data transfers via back-to-back latch's (Fig. 3).

The two processor systems (master & slave) employ interrupts to initiate an exchange of status communication codes via a back-to-back status latch. The status latch is four bits wide and the status codes are therefore limited to sixteen (Fig. 4). To achieve control for the exchange of data from master memory to slave memory and the reverse, as well as to initiate the execution of a program residing in the slave memory, requires seven status codes.

Level 3 (IRQ3) user interrupt vector is called by the SPC onto the VME bus. All SPC's are hardwired to this one interrupt level. To identify which SPC in a multiple system requires service, a different interrupt vector number is returned by each SPC. This 8 bit hexadecimal number is preset by links on the SPC cards. Hence up to 256 individual SPC's may connect to the master processor, if the electrical loading specifications of the VME bus are met and interconnections are practical.

The master processor, on being interrupted on level 3 by the slave processor, completes execution of its current instruction then accesses the interrupt by asserting interrupt acknowledge and reading back from the slave the interrupt vector number. This vector number is automatically multiplied by 4 by the master to obtain the address of the interrupt service routine from the MC68000 Exception Vector Assignment (Fig. 5). A slave initiates an interrupt by writing bit 5 via its SPC data bus into the interrupt latch.
The master may also interrupt the slave by writing bit 5 via the VME data bus to the other half of the back-to-back interrupt latch. This calls a SPC level 4 interrupt autovector. In calling an autovectored interrupt the processor does not access the bus for an interrupt vector number, but accepts the Exception Vector Assignment at vector number 28 as the address of the interrupt service routine.

The interrupt latch is an extension of the status latch and is addressed differently from the back-to-back data transfer latch (Fig. 6). The status latch is at an odd address because it is only one byte wide (8 bits) and is decoded by the appropriate address and the Lower Data Strobe (LDS). Whereas, the data latch is at an even address and is one word wide (16 bits). This is decoded by the appropriate address and both the Lower (LDS) and Upper Data Strobes (UDS).

The address decoders for both the data latch and the status latch on the VME bus are programmed into Programmable Array Logic (PAL) circuits. Likewise the address decoders for the data and status latch's on the SPC bus are programmed in PAL's. The address decoding on the VME bus side is different for each SPC (Fig. 6), whereas, the address decoding on each individual SPC bus is identical (Fig. 7). This has the advantage that the software data handling routines are common within each SPC.

3.3. SPC Processor Circuitry

3.3.1. Processor

A Motorola MC68000ZB10 microprocessor was selected as the processor for the SPC. This is a 68000 packaged in a JEDEC type B leadless chip carrier, running at a clock frequency of 10 MHz.
The leadless chip carrier uses less board space, being physically one third the size of the dual-in-line 64 pin package used as the master processor. The higher clock speed of 10 MHz (compared with 8 MHz for the master) ensures that the processor instruction cycle is minimal, enabling complex positioning algorithms to be executed within the high speed step rate of 10,000 steps/second of the stepper motors.

3.3.2. Watch Dog Timer

A 6 bit shift register, enabled each time an SPC access address is asserted, provides a 6 microsecond delay before initiating a bus error. This ensures that the system will not hang-up should the handshake signal DTACK not be returned.

The memory is relatively slow with an inherent DTACK delay of 300 nanoseconds, whereas, the Parallel Interface and Timer (PIT) returns DTACK almost immediately. Likewise, the response of DTACK from the data and status latch is immediate.

Should a fault occur and bus error is asserted, the slave processor forces an exception and enters the bus error routine (Vector number 2 of the Exception Vector Assignment) from which recovery can be obtained and an error flag set for the master.

3.3.3. Memory

Four by twenty eight pin sockets are provided for the memory integrated circuits. This is configured so that two ROM's either 2732A (4K bytes), 2764 (8Kbytes) or 27128 (16Kbytes) of erasable read only memory and two RAM's either 6116 (2Kbytes) or 6264 (8 Kbytes) of read/write memory may be accommodated on
the piggy-backed board. The maximum memory capability is therefore 48 Kbytes, made up of program 32Kbytes (2 x 27128) and read/write memory 16 Kbytes (2 x 6264).

The various types of memory chips are link selectable on the piggy-backed board. This memory is arranged to be word wide (16 bits), hence the need for two chips of each type, allowing faster access to both data and program than would byte access.

3.3.4. PIT

The Parallel Interface and Timer (PIT) is a programmable input/output interface, designed to be compatible with 68000 microprocessors. The parallel interface operates in unidirectional or bi-directional modes either 8 or 16 bits wide. Also included within the PIT is a 5 bit prescaler and a 24 bit wide counter timer. This timer can generate periodic interrupts, a square wave output or a single interrupt after a programmed time period.

Four hand-shake lines are available on the PIT, which enable external devices to interrupt the SPC or provide synchronous or asynchronous operation of two or more SPC's within a system.

The PIT is interconnected for possible use of all the facilities available within the device. The multi-function lines of Port C may be utilized, via a series of hardwired links, for Port C to perform input/output functions or act as a timer and interrupt handling device.
3.3.5. SPC Interrupts

Various interrupts may be employed on the SPC processor. As discussed in Section 3.2, the VME bus master calls a level 4 interrupt autovector. When this level is called VPA is asserted informing the processor that this is an autovector and not a user vector interrupt (Reference 2). User vector interrupts are also employed on the SPC. Level 1 (IRQ1) may be called by the PIT timer TOUT pin active. Level 2 (IRQ2) may be called by the PIT programmed to the handshake lines H1 to H4 (Reference 3). The PIT provides five separate interrupt vectors, the vector number to be returned to the processor is pre-loaded into special interrupt vector registers.

3.3.6. External I/O Buffering

The PIT outputs (Port A to C) are unable to sink current of any consequence and board space for electrical buffering is provided. All Port A to C lines and the 4 hand-shake lines are connected to stakes. Also, the 40 pin ribbon connector header (to external devices) is connected to stakes. Buffering circuits may be wire-wrapped between the stakes or the PIT connected directly to the 40 pin ribbon header.

The DC electrical characteristics of the PIT outputs are:

<table>
<thead>
<tr>
<th>Output Voltage</th>
<th>Load Current</th>
</tr>
</thead>
<tbody>
<tr>
<td>VOH = 2.4V</td>
<td>ILH = -100 A</td>
</tr>
<tr>
<td>VOL = 0.5V</td>
<td>ILO = 2.4 MA</td>
</tr>
</tbody>
</table>

Reference 2  Motorola ADI-814-R5 MC68000 March 1985
Reference 3  Motorola ADI-860-R2 MC68230 Dec 1983
3.3.7. SPC System Reset

The SPC is provided with a power-on-reset line which is held low for several milliseconds at switch "on" of the +5 volt VCC.

To regain control, in the advent of a fault condition, the main VME bus reset is common to all SPC resets. Therefore, an initialization switch may be utilized for manual overall system reset.

3.3.8. SPC Clock

Both the MC68000ZB10 and the MC682300-10 (PIT) are 10 MHz dynamic refresh devices and a crystal controlled clock module of 10 MHz is provided. This clock may also be connected to the Timer IN pin (TIN) for use with the 5 bit prescaler and the 24 bit counter timer function.

4. SOFTWARE

4.1. General

The memory map for the master processor is different from that allocated to the slaves. The master map has a much larger memory capacity (ROM capacity of the master is larger than the combined RAM/ROM capacity of a slave). Therefore, memory transfers from master to slave and vice versa requires an offset address. The offset chosen for the hot wire anemometer probe application is hex 10000. This means that data or program residing in the master memory at hex 12000, when transferred to the slave, resides at hex 2000. The offset also applies for the reverse transfer of data from the slave at hex 2000 to the master at hex 12000. The offset is
built into the software as a constant and may be changed depending upon the application.

In a minimum system, three routines are required to control master and slave.

(i) Transfer of data from master memory to slave memory.
(ii) Transfer of data from slave memory to master memory.
(iii) Master to initiate the execution of application program residing in slave memory.

4.2. Master Utility Programs

The master processor is equipped with software for control of all main system functions, such as, serial RS232C Communication between master and external Visual Display Terminal (VDU) and between master and an external host computer.

The system debugging utilities enable break-points to be inserted, registers of the master to be displayed and memory content to be examined and altered. The portion of the master software that controls data transfer between master and slave is listed in appendix 1.

4.3. Master Memory to Slave Memory Transfer

This transfer has been designated status code 2 (Fig. 4). Prior to any action, both status latches (master and slave) contain status code 0, which is the null code, meaning that both processors are free to operate independently of each other.

The master initiates the transfer by writing hex 22 into the master status latch. This is code 2 and dataline d5 of the VME bus calls SPC interrupt level 4.
All memory to memory transfers require a destination address to precede the data. Hence the master passes to the slave two consecutive words which represent the 32 bit destination address of the data to be transferred. The slave automatically subtracts the offset from the address received (section 4.1) then incrementally stores the following data, starting at that address and continuing until the EOT, code 10 is received. The slave, on receipt of an EOT responds with code 0 (null), exits from the interrupt service routine and proceeds with the program it was executing prior to being interrupted by the master.

Two handshake codes (code 14 and code 15) are used to signify that data have been received and the processor is waiting for further response.

An outline of the mechanics of the master memory to slave memory transfer and the use of the status codes is shown in Fig. 8.

4.4. Slave Memory to Master Memory Transfer

This transfer has been designated Status Code 1 (Fig. 4). Again the action is initiated by the master, writing hex 21 into the master status latch. The process that then follows is similar to that described in Section 4.3 and detailed in Fig. 9.

4.5. Execution of Program with Slave Memory

Execution of application software residing in slave memory is designated status code 9 (Fig. 4). The start address for the execution of the application software is passed from the master to the slave and loaded into the slave program counter.

The slave commences execution of the previously stored software starting at that address.
No offset is involved with the execution address passed from the master to the slave. Fig. 10 details the process involved.

Appendix 2 provides a listing of the data transfer routines residing within the slave software. Complete listings are stored on the ARL Philips PMDS Microcomputer Development System under /arl-rout/cards/SPC.

5. CONCLUSION

The memorandum has presented a general description of both hardware and software utilized in this intelligent slave peripheral controller. For the application of positioning a six degree of freedom hot wire anemometer probe the controller has been successfully demonstrated using two high speed stepper motors. (The other four motors are not yet available).

No difficulties were experienced in the control of two slaves by a single master. The full capabilities of the system have yet to be fully tested but indications are that the system is capable of control of stepper motors, using complex positioning algorithms, at speeds in excess of 10,000 steps/second. This controller could have application in any system which involves real time complex calculations for control, such as in high speed robotics.

The use of slave parallel 68000 microprocessors operating independently under the control of a master provides very powerful processing capabilities for high speed data acquisition and real time predictive control applications.
ACKNOWLEDGEMENTS

Of the many people who have made significant contributions to the development of this system, the author would particularly wish to record the contribution of W. Dekker and Dr J. Watmuff.
APPENDIX 1  DATA TRANSFER ROUTINES - MASTER SOFTWARE

!*********************************************************
! code 0 :: null each processor to do own thing
! code 1 :: spc memory to vme memory transfer
! code 2 :: vme memory to spc memory transfer
! code 9 :: begin execution in spc memory : address from vme
! code 10 :: end end of transfer
! code 14 and 15 :: ecode and fcode are data handshake codes
! bit 5 of vme status calls interrupt level 4 spc
! data address ffa60 for chnl1, ffb60 for chnl2
! status address ffa51 for chnl1, ffb51 for chnl2
!
!*********************************************************

!************************ get start and finish address from keyboard *************
!
acadd: move.l #ipiopm,a0
    jsr  txout.l
jsr  space.l
    jsr  fascii.l
move.l save.l,iopsad.l
    cmp.b #0x1,d3
    bne  xhot
move.b #0x2c,d0
    jsr  put.l
    jsr  fascii.l
xhot: move.l save.l,iopead.l
    jsr  crlf.l
chnagn: jsr  crlf.l
    move.l #meschn,a0
    jsr  txout.l
    jsr  get.l
    jsr  put.l
    cmp.b #'1',d0
    beq  isone
    cmp.b #'2',d0
    bne  chnagn
move.l #stats2,a5
move.l #datrg2,a6
    jsr  crlf.l
rts
isone: move.l #stats1,a5
move.l #datrg1,a6
    jsr  crlf.l
rts

************ get start and finish address from keyboard *************
!
!start address in iopsad
!finish address in iopead
******************** output address to spc **********************

opsfad: swap d1
move.w d1,(a6)
move.b #0x0e,(a5)
jsr ecode
swap d1
move.w d1,(a6)
move.b #0x0f,(a5)
jsr fcode
rts

Dec 4 10:41 1986 ---- report ---- Page 2

Code 0: send and receive null status

Code 0: move.b #0,(a5)
await: move.b (a5),d0
and. #0x0f,d0
beq acon
bra await
acon: rts

Code A: all data sent end transfer

Code A: move.b #0x0a,(a5)
bwait: move.b (a5),d0
and. #0x0f,d0
bne bwait
rts

Codes E & F: handshake control codes

Code E: move.b (a5),d0
and. #0x0f,d0
cmp #0x0e,d0
bne ecode
rts

Code F: move.b (a5),d0
and. #0x0f,d0
cmp #0x0f,d0
bne fcode
rts

--- report ---
code 1: spc to vme : memory to memory transfer

```assembly
! test if null are on status
! set start and finish address
! start address to a4
! offset on iopsad
! end address to a3
! offset on iopsad
! interrupt on iopsad
! wait for return of code 1

code1: jsr code0,1
        jsr accadd,1
        move.l iopsad,1,a4
        add.l #0x10000,a4
        move.l iopsad,1,a3
        add.l #0x10000,a3
        move.b #0x21,(a5)
cd1:    move.b (a5),d0
        and.l #0x0f,d0
        cmp #0x01,d0
        bne cd1
        move.l iopsad,1,d1
        jsr opsfad,1

dloop1: move.w (a6),(a4)
        cmp.l a4,a3
        bie recend
        add #2,a4
        move.b #0x0e,(a5)
        jsr ccode
        move.w (a6),(a4)
        cmp.l a4,a3
        bie recend
        add #2,a4
        move.b #0x0f,(a5)
        jsr fcode
        bra dloop1

! test to spc
! set null status
! get start address
! add address offset
! last byte of last word
! get finish address

recend: jsr acode,1
        move.b #0,(a5)
        move.l iopsad,1,a1
        add.l #0x10000,a1
        move.l a1,alt,1
        add.l #1,a4
        move.l a4,save,1
        cir.l offlag,1
        cir.l spnum,1
        bra hot

! set to vdu contents of iop memory
```
Dec 4 10:41 1986

--- report ---
Page 4

code 2: vme to spc: memory to memory transfer

```
code2:       jsr    accadd.l       !get start and end address
            jsr    trmiop
            bra    red

trmiop:      jsr    code0.l        !test null status
            move.l iopsad,a4
            sub.l  #0x10000,a4
            move.l iopead,a3
            sub.l  #0x10000,a3
            move.b #0x22,(a5)

cd2:         move.b (a5),d0       !loop until code 2 sent by spc
            and    #0x3f,d0
            cmp    #0x02,d0
            bne    cd2
            jsr    opsfad.l       !o/p start address to spc
            move.w (a4),(a6)     !o/p data
            move.b #0x0e,(a5)    !change handshake
            jsr    code
            cmp.l  a4,a3
            ble    endat
            add    #2,a4
            move.w (a4),(a6)     !o/p word
            move.b #0x0f,(a5)    !change handshake
            jsr    fcode
            cmp.l  a4,a3
            ble    endat
            add    #2,a4
            bra    datlop

endat:       jsr    acode.l        !loop to iop
            move.b #0,(a5)      !set null status
            rts
```
code 9: execute program in spc memory

| code 9: | move.l #ipstm,a0  |
|        | jsr txout.l       |
|        | jsr fascii.l      |
|        | jsr chnagn        |
|        | jsr cdex          |
|        | bra red           |

| cdex: | jsr code0.l       |

| cd9:  | move.b #0x29,(a5) |
|      | and.l #0x0f,d0   |
|      | cmp #0x09,d0     |
|      | bne cd9          |
|      | move.l save.l,d4 |
|      | swap d4          |
|      | move.w d4,(a6)   |
|      | move.b #0:0e,(a5)|
|      | jsr ecde.l       |
|      | swap d4          |
|      | move.w d4,(a6)   |
|      | jsr acode.l      |
|      | move.b #0,(a5)   |
|      | rts              |

message

get spc start address in d4
which spc
return to monitor

test null status
interrupt and code 9 status
for code 9

wait for code 9 reply

get high address first
io/p high address
change handshake to 0e
wait for responding 0e
get low address
io/p low address
set bot and wait for spc null
set vme status null
APPENDIX 2 DATA TRANSFER ROUTINES - SLAVE SOFTWARE

*****************************************************************************
!* Activate interrupts and wait in loop for incoming command
*****************************************************************************

iopst: move.w #0x2300, sr
loop: move.l d0,d0
       bra loop

*****************************************************************************
!* interrupt level 4 auto vector (070) called **************
*****************************************************************************

int4: movem.l d0+d7/a0-a6,-(a7)
       move.b status.l,d0
       and.b #0x0f,d0
       cmp.b #0,d0
       beq null
       cmp.b #1,d0
       beq code1
       cmp.b #2,d0
       beq code2
       cmp.b #3,d0
       beq code3
       cmp.b #4,d0
       beq code4
       cmp.b #5,d0
       beq code5
       cmp.b #6,d0
       beq code6
       cmp.b #7,d0
       beq code7
       cmp.b #8,d0
       beq code8
       cmp.b #9,d0
       beq code9
       cmp.b #0x0f,d0
       bne waits
       beq eot
       bra return

       return: movem.l (a7)+,a0-a6/d0-d7
       bra iopst

       comp: move.b #0,status.l
       waits: move.b status.l,d0
              and #0x0f,d0
              bne waits
              bra return

       !set sr to allow interrupt
       !do nothing until ie 4 auto vector
       !save all registers
       !get code number
       !test if null
       !test if code 1
       !test if code 2
       !test if code 3
       !test if code 4
       !test if code 5
       !test if code 6
       !test if code 7
       !test if code 8
       !test if eot
       !return code not recognised
       !restore all registers
       !return from interrupt
       !null status on spc
       !waits for null on vme
*************** handshake control routines ***************

ecode:  move.b status,l,d0  
and.b  #0x00f,d0  
cmp.b  #0x0a,d0  
beq comp  
cmp.b  #0x0e,d0  
bne  ecode  
  rts  

fcode:  move.b status,l,d0  
and.b  #0x00f,d0  
cmp.b  #0x0a,d0  
beq comp  
cmp.b  #0x0f,d0  
bne  fcode  
  rts  

*****************************************************************************

code 1 - iop memory to vme memory transfer  *
*****************************************************************************

codel:  move.b #1,status.1  
  jsr ecode.1  
  move.w datreg.1,d0  
  swap d0  
  and.l  #0xfffffffff00000,d0  
  move.l d0,a4  
  move.b #0x00, status.1  
  jsr fcode.1  
  move.w datreg.1,d0  
  and.l  #0xfffffffff00000,d0  
  add.l d0,a4  
  move.w (a4)+,datreg.1  
  move.b #0x0f,status.1  
  jsr ecode.1  
  move.w (a4)+,datreg.1  
  move.b #0x0e,status.1  
  jsr fcode  
  bra nxdat  

nxdat:  move.w (a4)+,datreg.1  
move.b #0x0f,status.1  
  jsr ecode.1  
  move.w (a4)+,datreg.1  
move.b #0x0e,status.1  
  jsr fcode  
  bra nxdat  

  !get status  
  !test eot code  
  !if eot exit  
  !test if vme status == 0e  
  !not 0e, try again  
  !is ecode can return  

  !get status  
  !test eot code  
  !if eot exit  
  !test if vme status == 0f  
  !not 0f, try again  
  !is fcode can return  

  !respond with code 1  
  !wait for ecode  
  !read high word start address  
  !put high word into top of d0  
  !high address into a4  
  !ecode onto status  
  !wait for responding fcode  
  !read low word start address  
  !complete address in a4  
  !o/p data from memory to vme  
  !o/p data to vme  
  !loop until data transferred  

**code 2 - VME memory to SPC memory transfer**

```assembly
   code2:  move.b #2,status.l
            jsr  ecode.l
            move.w datreg.l,d0
            swap  d0
            and.l #0xffff0000,d0
            move.l d0,a4
            move.b #0xe,status.l
            jsr  fcode.l
            move.w datreg.l,d0
            and.l #0x0000ffff,d0
            add.l d0,a4
            move.b #0xf,status.l
            jsr  ecode.l
            move.w datreg.l,(a4)+
            jsr  fcode.l
            move.w datreg.l,(a4)+
            bra  datlp

   datlp:  jsr  ecode.l
            move.w datreg.l,d0
            swap  d0
            and.l #0xffff0000,d0
            move.l d0,a0
            move.b #0xe,status.l
            jsr  fcode.l
            move.w datreg.l,d0
            and.l #0x0000ffff,d0
            add.l d0,a0
            move.b #0xf,status.l
            jmp  (a0)
```

**code 9 - execute a program in SPC memory**

```assembly
   code9:  move.b #9,status.l
            jsr  ecode.l
            move.w datreg.l,d0
            swap  d0
            and.l #0xffff0000,d0
            move.l d0,a0
            move.b #0xe,status.l
            jsr  fcode.l
            move.b status.l,d0
            cmp.b #0xa,d0
            bne  cd9
            move.w datreg.l,d0
            and.l #0x0000ffff,d0
            add.l d0,a0
            move.b #9,status.l
            jmp  (a0)
```

**cd9:**

```assembly
   cd9:  move.b status.l,d0
            and.b #0x0f,d0
            cmp.b #0x0a,d0
            bne  cd9
            move.w datreg.l,d0
            and.l #0x0000ffff,d0
            add.l d0,a0
            move.b #0,status.l
            jmp  (a0)
```
Table 1. ARL Drawing Numbers for SPC Hardware

<table>
<thead>
<tr>
<th>ARL Drawing Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>60278-A2</td>
<td>Parts List - Hardware</td>
</tr>
<tr>
<td>60279-A2</td>
<td>General Assembly - Hardware</td>
</tr>
<tr>
<td>60280-F3</td>
<td>Art Work - Front Panel</td>
</tr>
<tr>
<td>60281-A2</td>
<td>Parts List - Main Card (CPU)</td>
</tr>
<tr>
<td>60282-A1</td>
<td>Circuit Diagram - Main Card (CPU)</td>
</tr>
<tr>
<td>60283-A2</td>
<td>Assembly - Main Card (CPU)</td>
</tr>
<tr>
<td>60284-F2</td>
<td>Art Work 1 - 2:1 Lay up Main Card (CPU)</td>
</tr>
<tr>
<td>60285-F2</td>
<td>Art Work 2 - 2:1 Lay Up Main Card (CPU)</td>
</tr>
<tr>
<td>60286-A2</td>
<td>Parts List - Piggy-back Card (Memory)</td>
</tr>
<tr>
<td>60287-A1</td>
<td>Circuit Diagram - Piggy-back Card (Memory)</td>
</tr>
<tr>
<td>60288-A2</td>
<td>Assembly - Piggy-back card (Memory)</td>
</tr>
<tr>
<td>60289-F2</td>
<td>Art Work 1 - 2:1 Lay up Piggy-back Card (Memory)</td>
</tr>
<tr>
<td>60290-F2</td>
<td>Art Work 2 - 2:1 Lay up Piggy-back Card (Memory)</td>
</tr>
<tr>
<td>60330-A3</td>
<td>Detail - Front Panel Cut-Out</td>
</tr>
</tbody>
</table>
FIG. 1  MASTER – SLAVE SYSTEM CONFIGURATION
FIG. 2(a)  PIGGY-BACKED SPC

FIG. 2(b)  SEPARATED SPC
<table>
<thead>
<tr>
<th>CODE</th>
<th>HEX</th>
<th>OPERATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
<td>Null.</td>
</tr>
<tr>
<td>1</td>
<td>01</td>
<td>SPC Memory to VME Memory</td>
</tr>
<tr>
<td>2</td>
<td>02</td>
<td>VME Memory to SPC Memory</td>
</tr>
<tr>
<td>3</td>
<td>03</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>04</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>05</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>06</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>07</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>08</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>09</td>
<td>Execute program in SPC Memory</td>
</tr>
<tr>
<td>10</td>
<td>0A</td>
<td>End of transmission EOT</td>
</tr>
<tr>
<td>11</td>
<td>0B</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>0C</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>0D</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>0E</td>
<td>Handshake e code</td>
</tr>
<tr>
<td>15</td>
<td>0F</td>
<td>Handshake f code</td>
</tr>
</tbody>
</table>

FIG. 4  SPC STATUS CODES
<table>
<thead>
<tr>
<th>Vector Number(s)</th>
<th>Address</th>
<th>Assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 000</td>
<td>SP</td>
</tr>
<tr>
<td></td>
<td>0 004</td>
<td>SP</td>
</tr>
<tr>
<td>2</td>
<td>0 008</td>
<td>SD</td>
</tr>
<tr>
<td>3</td>
<td>0 00C</td>
<td>SD</td>
</tr>
<tr>
<td>4</td>
<td>0 010</td>
<td>SD</td>
</tr>
<tr>
<td>5</td>
<td>0 014</td>
<td>SD</td>
</tr>
<tr>
<td>6</td>
<td>0 018</td>
<td>SD</td>
</tr>
<tr>
<td>7</td>
<td>0 01C</td>
<td>SD</td>
</tr>
<tr>
<td>8</td>
<td>0 020</td>
<td>SD</td>
</tr>
<tr>
<td>9</td>
<td>0 024</td>
<td>SD</td>
</tr>
<tr>
<td>10</td>
<td>0 028</td>
<td>SD</td>
</tr>
<tr>
<td>11</td>
<td>0 02C</td>
<td>SD</td>
</tr>
<tr>
<td>12</td>
<td>0 030</td>
<td>SD</td>
</tr>
<tr>
<td>13</td>
<td>0 034</td>
<td>SD</td>
</tr>
<tr>
<td>14</td>
<td>0 038</td>
<td>SD</td>
</tr>
<tr>
<td>15</td>
<td>0 03C</td>
<td>SD</td>
</tr>
<tr>
<td>16-23</td>
<td>0 040</td>
<td>SD</td>
</tr>
<tr>
<td>24</td>
<td>0 060</td>
<td>SD</td>
</tr>
<tr>
<td>25</td>
<td>0 064</td>
<td>SD</td>
</tr>
<tr>
<td>26</td>
<td>0 068</td>
<td>SD</td>
</tr>
<tr>
<td>27</td>
<td>0 06C</td>
<td>SD</td>
</tr>
<tr>
<td>28</td>
<td>0 070</td>
<td>SD</td>
</tr>
<tr>
<td>29</td>
<td>0 074</td>
<td>SD</td>
</tr>
<tr>
<td>30</td>
<td>0 078</td>
<td>SD</td>
</tr>
<tr>
<td>31</td>
<td>0 07C</td>
<td>SD</td>
</tr>
<tr>
<td>32-47</td>
<td>0 080</td>
<td>SD</td>
</tr>
<tr>
<td>48-63*</td>
<td>0 0CO</td>
<td>SD</td>
</tr>
<tr>
<td>255</td>
<td>0 FF</td>
<td>SD</td>
</tr>
<tr>
<td>64-255</td>
<td>100</td>
<td>SD</td>
</tr>
<tr>
<td>1023</td>
<td>3 FF</td>
<td>SD</td>
</tr>
</tbody>
</table>

FIG. 5  MOTOROLA MC 68000 EXCEPTION VECTOR ASSIGNMENT
<table>
<thead>
<tr>
<th>Address</th>
<th>Device</th>
<th>Type</th>
<th>Latch</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFFF60</td>
<td>SPC 6</td>
<td>DATA</td>
<td>LATCH</td>
</tr>
<tr>
<td>FFFFS1</td>
<td>SPC 6</td>
<td>STATUS</td>
<td>LATCH</td>
</tr>
<tr>
<td>FFEF60</td>
<td>SPC 5</td>
<td>DATA</td>
<td>LATCH</td>
</tr>
<tr>
<td>FFEF51</td>
<td>SPC 5</td>
<td>STATUS</td>
<td>LATCH</td>
</tr>
<tr>
<td>FFDF60</td>
<td>SPC 4</td>
<td>DATA</td>
<td>LATCH</td>
</tr>
<tr>
<td>FFDF51</td>
<td>SPC 4</td>
<td>STATUS</td>
<td>LATCH</td>
</tr>
<tr>
<td>FFCF60</td>
<td>SPC 3</td>
<td>DATA</td>
<td>LATCH</td>
</tr>
<tr>
<td>FFCF51</td>
<td>SPC 3</td>
<td>STATUS</td>
<td>LATCH</td>
</tr>
<tr>
<td>FFBF60</td>
<td>SPC 2</td>
<td>DATA</td>
<td>LATCH</td>
</tr>
<tr>
<td>FFBF51</td>
<td>SPC 2</td>
<td>STATUS</td>
<td>LATCH</td>
</tr>
<tr>
<td>FFAF60</td>
<td>SPC 1</td>
<td>DATA</td>
<td>LATCH</td>
</tr>
<tr>
<td>FFAF51</td>
<td>SPC 1</td>
<td>STATUS</td>
<td>LATCH</td>
</tr>
</tbody>
</table>
FIG. 8 MASTER MEMORY TO SLAVE MEMORY TRANSFER
FIG. 9  SLAVE MEMORY TO MASTER MEMORY TRANSFER
FIG. 10 EXECUTION OF PROGRAM IN SPC MEMORY

Sets Program Counter equal to Execution Address when execution of an application Program

Status Register Data Register Comment
DISTRIBUTION

AUSTRALIA

Department of Defence

Defence Central
Chief Defence Scientist
Deputy Chief Defence Scientist (shared copy)
Superintendent, Science and Program Administration (shared copy)
Controller, External Relations, Projects and
Analytical Studies (shared copy)
Counsellor, Defence Science (London) (Doc Data Sheet Only)
Counsellor, Defence Science (Washington) (Doc Data Sheet Only)
Defence Science Representative (Bangkok)
Defence Central Library
Document Exchange Centre, DISB (18 copies)
Joint Intelligence Organisation
Librarian H Block, Victoria Barracks, Melbourne
Director General - Army Development (NSO) (4 copies)

Aeronautical Research Laboratories
Director
Library
Superintendent - Aerodynamics
Divisional File - Aerodynamics
Group File (IFEG) - Aerodynamics (2 copies)
N. Pollock
B. Fairlie
Author: J.F. Harvey

Materials Research Laboratories
Director/Library

Defence Research Centre
Library

SPARES (3 copies)
TOTAL (41 copies)
A system providing multiple slave co-processors operating independently, yet in parallel upon the same VME bus, all under the control of a master processor is described.

Data are passed unidirectionally from master to slave enabling rapid processing of complex algorithms for the control of external peripherals.
This paper is to be used to record information which is required by the Establishment for its own use but which will not be added to the DISTIS data base unless specifically requested.

16. Abstract (contd)

17. Imprint

Aeronautical Research Laboratories, Melbourne

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Aerodynamics Technical Memorandum 385</td>
<td>55 6055</td>
<td></td>
</tr>
</tbody>
</table>

21. Computer Programs Used

22. Establishment File Ref(s)