SEGA OF AMERICA, INC.
Consumer Products Division

# GENESIS TECHNICAL BULLETIN #1

**TO:** Developers and Third Parties

**FROM:** Technical Support

**DATE:** March 13, 1991

**SUBJECT:** **ERRORS WITHIN THE MICROTEC EXAMPLES**

---

There are three errors in examples that are provided. Please make the changes as noted.

In TESTC68K.bat:

The asm68k commands end with a semicolon, remove it and the file will assemble correctly.

The link command is incorrect. It should be:
LNK68K -c sieve.cmd -o sieve sieve

The C compiler documentation refers to a command line option of 'STRINGSINTEXT' for allocating string in code segment rather than data segment. The correct spelling for this option is 'STRINTEXT'.

125 Shoreway Road, San Carlos, CA 94070 • (415) 802-4400 • FAX (415) 802-4458

SEGA OF AMERICA, INC.
Consumer Products Division

# GENESIS TECHNICAL BULLETIN #2

**TO:** Developers and Third Parties

**FROM:** Technical Support

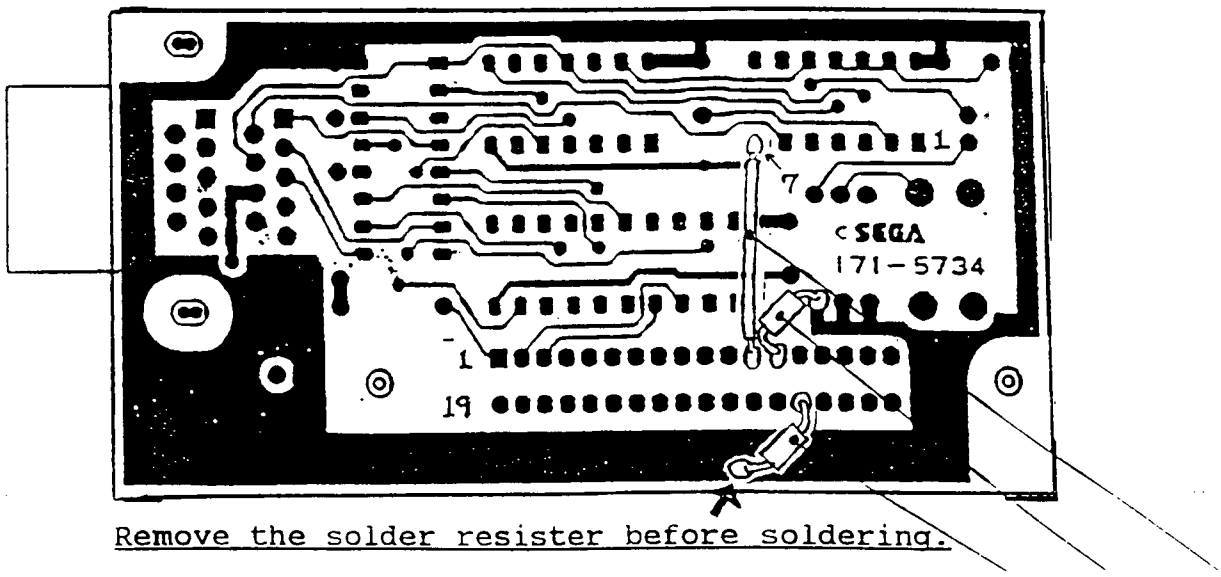
**DATE:** June 19, 1990

**SUBJECT:** **GENESIS LOADER BOARD**

---

In order to prevent loading errors, please have the following modification on the Genesis Loader Board. You need to have:

2 Resister 4.7kΩ 1/8W
1 Jumper Wire




Remove the solder resister before soldering.

1. Place 4.7k resister between Centronics Connector Pin #32 and +5V.

2. Place 4.7k resister between Centronics Connector Pin #13 and +5v.

3. Place jumper wire between Centronics Connector Pin #12 and ICI 74HC74 Pin #7 (GND).

125 Shoreway Road, San Carlos, CA 94070 • (415) 802-4400 • FAX (415) 802-4458

SEGA OF AMERICA, INC.
Consumer Products Division

# GENESIS TECHNICAL BULLETIN #3

**TO:** Developers and Third Parties

**FROM:** Technical Support

**DATE:** August 2, 1990

**SUBJECT:** **PRECAUTION WHEN ACCESSING THE Z80 BUS FROM A 68000 MAIN (NON-INTERRUPT) ROUTINE**

---

If the following routine was executed and an interrupt occurred between Step 2 and 3, a data error would occur IF the interrupt also access the Z80 bus.

1. Send Z80 bus request
2. Check acknowledge
3. Writing data to Z80 bus
4. Release Z80 bus

Essentially, the main routine sends a bus request, but before it can conduct its data transfer, the interrupt sends its own bus request. When the interrupt completes, it releases the Z80 bus, as it should. Unfortunately, the main routine expects the bus to be available; thus, an error occurs. When this error occurs, the Z80 RAM is corrupted. One symptom of this error is a sound is SOMETIMES interrupted.

The solution is really quite simple, before Step 1 above, disable interrupts and re-enable when done.

125 Shoreway Road, San Carlos, CA 94070 • (415) 802-4400 • FAX (415) 802-4458

SEGA OF AMERICA, INC.
Consumer Products Division

# GENESIS TECHNICAL BULLETIN #4

**TO:** Developers and Third Parties

**FROM:** Technical Support

**DATE:** September 7, 1990

**SUBJECT:** **READING THE CONTROLLER PADS**

---

When reading a location within the range of $A10000-$A100FF, which includes the controller pads, the Z80 must have a bus request. If not, the wait state will change from 250ns to 110ns. The shorter time will cause the Z80 to misread ALL further data. Once the ports have been read, the Z80 may be released.

Included with this bulletin is a reprint of the routine that reads the control pads in the Logo Demo program. As you can see, this routine sends a bus request, reads the pads, and then, releases the Z80.

125 Shoreway Road, San Carlos, CA 94070 • (415) 802-4400 • FAX (415) 802-4458

```
;           :           :           ( TC )
;           :           ( TA )
;           ( ST )
;sw_data1+1 is the address for edge data of port_1

read_sw:

                    z80_diw                         ; set busreq
                    bsr.b       rs_sub
                    z80_ei
                    rts

          rs_sub

                    lea         sw_data1,a5         ;switch data store address
                    lea         port_1,a6           ;port for p1
                    bsr         ??rs_0
                    addq.w      #2,a6               ;a6 = port_2

;
??rs_0:
                    move.b      #$0,(a6)            ;set TH = 0
                    nop                             ;wait
                    nop
                    move.b      (a6),d7             ;input data when TH = 0
                    asl.b       #2,d7               ;chane data when TH = 0 & wait
                    move.b      #$40,(a6)           ;set TH = 1
                    andi.w      #11000000b,d7       ;chane data when TH = 0 & wait
                    move.b      (a6),d6             ;input data when TH = 1
                    andi.w      #00111111b,d6
                    or.b        d6,d7
                    not.b       d7                  ;push -> 1
                    move.b      (a5),d6             ;copy
                    eor.b       d7,d6
                    move.b      d7,(a5)+            ; switch data
                    and.b       d7,d6
                    move.b      d6,(a5)+            ;switch edge data
                    rts

;*******< port initial  >******;
sw_init:                                            ;port initial
                    moveq       #$40,d7             ;set TH out
                    move.b      d7,cont_1
                    move.b      d7,cont_2
                    move.b      d7,cont_3
                    rst
;*****************************************************;
;                                                     ;
;-----------------------------------------------------;
;     input       a6.1 = controller port address      ;
;     output      d7.b == $00       Modem             ;
;                         $0d       Mega Drive Joy Stick;
;                         $0e       Ram Disk          ;
;                         $0f       ETC. or None      ;
;     use         d0,a0   |                           ;
;*****************************************************;
cont                set     $0006
```

```
c:  _port:

                movem.l d1-d2/a5,-(sp)          ;register push
                dis
                z80_diw
                lea     ??and_data(pc),a5       ;data table address set
                move.b  (a5),cont(a6)           ;Th bit set output
                moveq.l #0,d7                   ;rerturn register initial
                moveq.l #$08,d1                 ;counter set
```

SEGA OF AMERICA, INC.
Consumer Products Division

# GENESIS TECHNICAL BULLETIN #5

**TO:** Developers and Third Parties

**FROM:** Technical Support

**DATE:** November 26, 1990

**SUBJECT:** **CARTRIDGE IDENTIFICATION**

---

Every product must have, at location 100h, an IDTABLE. The table is slightly different for each type of developer. The difference is small, affecting the product code and the copyright, both important for product approval. Please replace the following pages in your Sega Software Manual.

125 Shoreway Road, San Carlos, CA 94070 • (415) 802-4400 • FAX (415) 802-4458

```
;************************************************
;*     SEGA GENESIS CARTRIDGE ID TABLE          *
;*                                              *
;*     STANDARD FORMAT FOR SEGA OF AMERICA      *
;*                                              *
;*     NOV 26 1990 SEGA OF AMERICA              *
;*                                              *
;************************************************

check_sum equ  $0000
     ORG $0100
;-------------------------------------------------------------------------
     dc.b 'SEGA GENESIS    '         ;100
     dc.b '(C)SEGA 199X.XXX'         ;110 release year.month
     dc.b 'your game tile  '         ;120 Japan title
     dc.b '                '         ;130
     dc.b '                '         ;140
     dc.b 'your game title '         ;150 US title
     dc.b '                '         ;160
     dc.b '                '         ;170
     dc.b 'GM MK-XXXX -XX'           ;180 product #, version
     dc.w check_sum                  ;18E check sum
     dc.b 'J               '         ;190 controller
     dc.l $00000000,$0007ffff,$00ff0000,$00ffffff     ;1a0
     dc.b '                '         ;1B0
     dc.b '                '         ;1C0
     dc.b '                '         ;1D0
     dc.b '                '         ;1E0
     dc.b 'U               '         ;1F0
```

```
;*************************************************
;*     SEGA GENESIS CARTRIDGE ID TABLE           *
;*                                               *
;*     STANDARD FORMAT FOR THIRD PARTIES         *
;*                                               *
;*     NOV 26 1990 SEGA OF AMERICA               *
;*                                               *
;*************************************************

check_sum equ  $0000
     ORG $0100
;-------------------------------------------------------------------------
     dc.b 'SEGA GENESIS    '         ;100
     dc.b '(C)T-XX 199X.XXX'         ;110 release year.month
     dc.b 'your game tile  '         ;120 title
     dc.b '                '         ;130
     dc.b '                '         ;140
     dc.b 'your game title '         ;150 title
     dc.b '                '         ;160
     dc.b '                '         ;170
     dc.b 'GM T-XXXXXX XX'           ;180 product #, version
     dc.w check_sum                  ;18E check sum
     dc.b 'J               '         ;190 controller
     dc.l $00000000,$0007ffff,$00ff0000,$00ffffff     ;1a0
     dc.b '                '         ;1B0
     dc.b '                '         ;1C0
     dc.b '                '         ;1D0
     dc.b '                '         ;1E0
     dc.b 'U               '         ;1F0
```

SEGA®

SEGA OF AMERICA, INC.
Consumer Products Division

# GENESIS TECHNICAL BULLETIN #6

To: Developers and Third Parties

From: Mac Senour, Technical Support

Date: 4/2/91

Re: Impossible values read from the controller pads

It is possible for a well used controller pad to yield incorrect values. Specifically: up AND down, left AND right. This is due to the small piece of rubber wearing out inside the pad.

Your routine that handles the values read should be written so that if the number is NOT one of the expected values, it should re-read the pad to obtain a correct value.

As always, please call if you have any questions or comments.



573 Forbes Blvd., P.O. Box 2167, So. San Francisco, CA 94080 • (415) 742-9300 • FAX (415) 742-0516

SEGA OF AMERICA, INC.
Consumer Products Division

# GENESIS TECHNICAL BULLETIN #11

**TO:** Developers and Third Parties

**FROM:** Technical Support

**DATE:** September 9, 1991

**SUBJECT:** **PROBLEMS DURING SOUND ACCESS**

---

Sound output stops during a game.

Problem:

The busy flag was read in the FM sound generator YM2616 like this (address 4001h)

(CS,RD,WR,A1,A0) = (0,0,1,0,1)

However, in the case of the YM2612, it's output is not regulated according to the conditions set above. This results in the device being read as "not busy," and as a consequence, ends up outputting sound.

Fix:

When the FM sound generator's busy flag is read, do not access anything else other than:

(A1,A0) = (0,0) (address 4000h)

125 Shoreway Road, San Carlos, CA 94070 • (415) 802-4400 • FAX (415) 802-4458

SEGA OF AMERICA, INC.
Consumer Products Division

# GENESIS TECHNICAL BULLETIN #12

**TO:** Developers and Third Parties

**FROM:** Technical Support

**DATE:** September 9, 1991

**SUBJECT:** **PROBLEMS WITH REPEATED RESETS**

---

The software seems to go out of control when resets are repeated.

Problem:

When the reset occurs, the CPU is reset; however, the VDP is not. When the reset occurs during a DMA, the VDP continues the DMA. The VDP is accessed right after the reset. If this is done while the VDP is executing a DMA, then this access is ineffective.

Fix:

Before accessing the VDP after the initialization program (ICD_BLK4), check the DMA BUSY status register. If a DMA is being executed, do not attempt to access the VDP.

If the problem persists, ensure that you are not executing a DMA right after a reset.

125 Shoreway Road, San Carlos, CA 94070 • (415) 802-4400 • FAX (415) 802-4458

SEGA OF AMERICA, INC.
Consumer Products Division

# GENESIS TECHNICAL BULLETIN #13

**TO:** Developers and Third Parties

**FROM:** Technical Support

**DATE:** September 9, 1991

**SUBJECT:** **CORRECTIONS TO THE GENESIS SOFTWARE MANUAL**

---

When discussing VRAM, CRAM, and VSRAM access, the manual states in Pages 22-27 that byte access is possible. This is incorrect. Access is limited to word or long word.

On Page 77, it implies that the 68000 may set the bank switches. The bank switches MUST be set by the Z80.

These changes affect Version 1.0 of the manual, later versions will reflect this correction.

125 Shoreway Road, San Carlos, CA 94070 • (415) 802-4400 • FAX (415) 802-4458

SEGA OF AMERICA, INC.
Consumer Products Division

# GENESIS TECHNICAL BULLETIN #14

**TO:** Developers and Third Parties

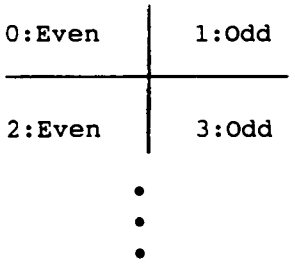
**FROM:** Technical Support

**DATE:** September 5, 1991

**SUBJECT:** **ROM SPLITTING**

---

As we all know, Genesis products must be split into 128k odd and even pieces. Sega expects the ROM images to be in the following format:




For larger products, continue the above pattern. We would appreciate it if it all products would conform to this method of splitting. Please request the utility to split ROMs, M2B, or Split4, if your current tool can't output files in this manner.

125 Shoreway Road, San Carlos, CA 94070 • (415) 802-4400 • FAX (415) 802-4458

SEGA OF AMERICA, INC.
Consumer Products Division

# GENESIS TECHNICAL BULLETIN #15

To: Sega Developers

From: Dave Marshall, Technical Support

Date: April 5, 1993

Re: Genesis Technical Information

==========

The attached document contains information on waiting during VRAM reading. The Super Target information applies to a CD development system.

125 Shoreway Road, San Carlos, CA 94070 • (415) 802-4400 • FAX (415) 802-4458

o WAIT during Mega Drive VRAM READ.

In the Mega Drive, during VRAM READ, when the following conditions are not met, data may not be read correctly.

<u>During display period and H-blank</u>

VRAM Address Set

VRAM READ

WAIT exceeding 116 CPU clock cycles

Next VRAM Address Set

<u>During V-blank</u>

VRAM Address Set

VRAM READ

Wait exceeding 12 CPU clock cycles

Next VRAM Address Set

As seen above, after the last VRAM READ is completed, WAIT is required before the next VRAM address is set. With respect to VRAM READ, it can be read continuously.

o SUPER TARGET *IC 19*

When 68000 CPU accesses an illegal area, "DATA ACKNOWLEDGE" signal may not be sent back - causing "hang up" in the production unit. Depending on some ROM emulators or ICE (+SUPER TARGET), even the emulator may hang up. To avoid such hang ups, measures have been taken so that SUPER TARGET would return ACKNOWLEDGE signal to the CPU without system hang ups. This is done via IC 19.
As a result, due to this IC function, for SUPER TARGET to function normally, the hang up occurs in the production unit.

Please take the following steps to correct the problem:

(1) Before checking the software through SUPER TARGET + CPU, remove IC 19 from the socket.

(2) When using ICE (ROM emulator), IC19 may be removed. Reinstall the IC should ICE (ROM emulator) act abnormally.

(3) The above measures are common with the MEGA DRIVE and the MEGA CD.

SEGA OF AMERICA, INC.
Consumer Products Division

# GENESIS TECHNICAL BULLETIN #16

To: Sega Developers

From: Dave Marshall, Technical Support

Date: April 5, 1993

Re: Genesis Technical Information

==========

The attached document contains information on new peripherals and their protocol. The source files that this note references are available on the BBS in the Genesis conference. The files apply mainly to the 4P adapter. There is already driver code available for all other peripherals in the Genesis conference. This driver code is in a file called UNIVERS.ZIP

**MEGA DRIVE**
(January 7, 1993)

1. Regarding Mega Drive register #11 IE2 bit

This bit is set to allow or inhibit the external interrupt. Set this bit to "0" when not using external interrupt (GUN, etc.). When this bit is "1," error occurs on the Sega-made address checker.

2. Regarding Mega Drive Control Pad access

When accessing control pad, please strictly observe the following items:
(If not observed, the peripherals to be sold in the future may not operate properly.)

(1) Access Pad from within V-INT (routine)
Generally, pad should be accessed from within V-INT.
The access period is approximately 16 ms for NTSC and 20 ms for PAL.
The access interval should not exceed the above periods.

(2) Pad access through V-INT routine must be limited to one time only. Reading pad data repeatedly via V-INT could cause malfunction.

(3) Check the peripherals
Currently, the sale of 6-button pad micro trackball 4P adapter (all tentative names) is being planned. Therefore, even for the Pad-exclusive software, peripherals other than the Pad may be used or those devices may be switched. So please diligently check peripheral devices currently used and make sure to input data that meet those devices.
We will supply a sample program that checks the device ID at each V_INT and then executes controller input. Please use this program as your reference.

(4) Modify sample program only after understanding it completely.
Our sample program has been modified incorrectly into some software product that is for sale and suffer from erratic operation. Please modify the sample program after full understanding of its structure and operation.

(Translated by: J. Tabrizi, April 5, 1993)

## How to accommodate peripherals

The 6-button pad, Mouse, and 4P TAP will soon go on sale. Whether or not each game will accommodate such peripherals, they must be made switchable once the game is in progress. 4P TAP has the selector function, therefore, it is very possible to be switched in mid-game. To this end, a sample program has been made to check the device ID at each V-INT and then execute the controller input. Please study this program. The file is <I0.ASM>. We also prepared a set of I/O check program.

The work structure may vary depending on the accommodation arrangement (the sample is described here). The unwanted areas may be deleted.

Accommodating 4P_TAP

I/O data

| offset | use | | label in MEM_MAP. ASS |
|---|---|---|---|
| +0 | 1byte | Connector-1 device ID | port_id1 |
| +1 | 1byte | Connector-2 device ID | port_id2 |
| +2 | 4b | 4p tap connector info for Connector-1 | mlt_info1 |
| +2 | | CN1 of 4P-TAP from Connector-1 | |
| +3 | | CN2 of 4P-TAP from Connector-1 | |
| +4 | | CN3 of 4P-TAP from Connector-1 | |
| +5 | | CN4 of 4P-TAP from Connector-1 | |
| +6 | 4b | 4p tap connect info for Connector-1 | mlt_info2 |
| +6 | | CN1 of 4P-TAP from Connector-1 | |
| +7 | | CN2 of 4P-TAP from Connector-1 | |
| +8 | | CN3 of 4P-TAP from Connector-1 | |
| +9 | | CN4 of 4P-TAP from Connector-1 | |

10Bytes of data must be secured for controller data WORK. An amount equal to 8 units is secured. Whether or not Connector-1 is 4P_TAP, Connector-2 data is stored in "iodata_5."

| | | |
|---|---|---|
| +10 | 1P data or 4P_TAP CN1 data | iodata_1 |
| +20 | CN2 data at 4P_TAP | iodata_2 |
| +30 | CN3 data at 4P_TAP | iodata_3 |
| +40 | CN4 data at 4P_TAP | iodata_4 |
| +50 | 2P data or 4P_TAP CN1 data | iodata_5 |
| +60 | CN2 data at 4P_TAP | iodata_6 |
| +70 | CN3 data at 4P_TAP | iodata_7 |
| +80 | CN4 data at 4P_TAP | iodata_8 |

iodata_x data varies by controller.

Note:
ST=start button
A=A button
B=B button
C=C button
X=X button
Y=Y button
Z=Z button
MD=Mode button
[...] describes each bit.

;3-button JOYPAD

| | bit7 | bit6 | bit5 | bit4 | bit3 | bit2 | bit1 | bit0 |
|---|---|---|---|---|---|---|---|---|
| +0 | 0...JOYPAD 3-button Controller ID | | | | | | | |
| +2 | [ST | A | | C | B | Right | Left | Down Up] |
| +3 | +2 leading edge data | | | | | | | |

(Translated by: J. Tabrizi, April 5, 1993)

+4 Not used

;6-button JOYPAD
+0 1...JOYPAD 3-button Controller ID
+2 [ST A C B Right Left Down Up]
+3 +2 leading edge data
+4 [0 0 0 0 MD X Y Z]
+5 +4 leading edge data
+6 Not used

;Mouse
+0 2...Mouse Controller ID
+2 [Center Middle Right Left Yover Xover Ysign Zsign]
+3 +2 leading edge data
+4 X data
+5 Y data
+6 X data sign
+8 Y data sign

The files have the following composition:

```
I-- IO_READ.ME        This file
I-- MEM_MAP.ASS       Memory map assign
I-- MACRO.MAC         Macro
I-- MAIN.ASM          Processing from Vector, ID, power supply ON
I-- ICD_BLK4.PRG      Included in Hard Initial Program <MAIN.ASM>
I-- INT.ASM           Interrupt program
I-- IO.ASM            This file is the center of I/O program
I-- SUB.ASM           Subroutine
I-- VDP.PRG           Subroutine for VDP related items
I-- SPCNT.ASM         Sprite control
I-- IO_CHK.ASM        Checking and displaying I/O data. Not to be minded
I-- EQU_EV.ASM          For reflecting mem_map.ass into EVT file
I-- ASCII.ASM         ASCII CG data
I-- IOSAMP.MK           Make definition file
I-- IOSAMP.LNK        Link definition file
I-- I.CMD               ICE initial batch
I-- MK.BAT              Batch file for MAKE
I-- XREF.1              file for Xref inclusion

I              I--EQU_EV.OBJ
I              I--MAIN.OBJ
I              I--INT.OBJ
I              I--IO.OBJ
I              I--SUB.OBJ
I--OBJ--    I--SPCNT.OBJ
               I--ASCII.OBJ
               I--IO_CHK.OBJ
               I--IOSAMP.SYM
               I--IOSAMP.EVT
               I--IOSAMP.S28
```

Reference:
This is for those who say "ID does the check, but only JOYPAD can do the control."

<3-button accommodation>
Essential WORK is 1 Byte only (2 Bytes when making edge data)

(Translated by: J. Tabrizi, April 5, 1993)

For program see <IO_JOY3.PRG>. Make sure to call only once from V_INT routine. Used in both cases of making edge and otherwise.

```
MAKE_EDGE       set   1       ;if 1 then make edge, 0...only read
If so: Making edge
MAKE_EDGE       set   0       ;if 1 then make edge, 0...only read
If so: Not making edge
```

<Distinguishing between 3-button and 6-button>
Please see "get_joy" routine within <IO.ASM>

(Translated by: J. Tabrizi, April 5, 1993)

# GENESIS TECHNICAL BULLETIN #17

To: Sega Developers

From: Bert Mauricio, Technical Support

Date: May 25, 1993

Re: Genesis Technical Information

==========

The attached document contains information on Eproms for use in Sega development boards.

Listed are manufacturers and model numbers of EPROM chips that are recommended to use with the Genesis/Game Gear boards. Please note chips are non-JEDEC.

| | | | |
|---|---|---|---|
| 1 megabit: | NEC | D27C1000A | 120ns or 150ns |
| | AMD | AM27C100 | 150ns |
| | Fujitsu | MBM27C1000 | 150ns |
| | Intel | D27C100 | 150ns |
| 2 megabit: | NEC | D27C2001D | 150ns |
| | AMD | AM27C020 | 200ns |
| 4 megabit | NEC | D27C4001D | 150ns |
| | Toshiba | TC574000D | 150ns |

*note We have found the Intel 27C020 2 megabit chips to be unreliable.

There has also been problems when using 2 megabit chips on the standard 8M/16M EPROM board w/64k backup. (171-5663 M5 Test MB)

Dip switch settings:

| | |
|---|---|
| 1 megabit chips | 1-on |
| | 2-on |
| | 3-on |
| | 4-off |
| 2-megabit chips | 1-on |
| | 2-on |
| | 3-off |
| | 4-on |

If there are recurring problems, it may be more reliable to use 2 megabit chips on the new SRAM/EE backup board which accepts 1,2 and 4 megabit chips. (GN 837-8093)

Eprom source: America II Electronics (Florida) (800) 767-2637

# GENESIS TECHNICAL BULLETIN #18

To: Sega Developers

From: Dave Marshall, Technical Support

Date: May 28, 1993

Re: Genesis Technical Information

==========

This is just a reminder that all Genesis carts must contain "lockout code". This code will prevent cartridges from being run on machines not listed in the Country Code field of the ID Block.

This lockout code is available on the Sega Technical Support BBS in a file called LOCK.ZIP in the Genesis and 3rdParty conferences. The zip file contains information on how and where to use the lockout code.

The test department does look for this feature and will bug any game that does not contain the lockout capability.

SEGA OF AMERICA, INC.
Consumer Products Division

# GENESIS TECHNICAL BULLETIN #19

To: Sega Developers

From: Dave Marshall, Technical Support

Date: July 6, 1993

Re: Mega CD Technical Information

==========

This document contains requirements that must be followed when producing games for the non-U.S. market.

1. All software available in Japan should not contain a "TM" or "R" with the Sega Logo. All software available for the U.S. and European markets must have a "TM" on the right side of the Sega Logo.

2. Any SEGA games made available for the European market must be PAL compatible. The PAL version must be of the same graphic and sound quality as the NTSC version.

3. Any SEGA games made available in Japan must be compatible with both NTSC and PAL game systems.

130 Shoreline Drive, Redwood City, CA 94065 • (415) 508-2800 • FAX (415) 802-1448

SEGA OF AMERICA, INC.
Consumer Products Division

# GENESIS TECHNICAL BULLETIN #20

To: Sega Developers

From: Dave Marshall, Technical Support

Date: July 8, 1993

Re: Genesis Technical Information

==========

The following tech note contains information for developing Genesis games that are larger than 16mbits.

130 Shoreline Drive, Redwood City, CA 94065 • (415) 508-2800 • FAX (415) 802-1448

# MEGA DRIVE TECHNICAL INFORMATION

## 1. Regarding the VDP Access

In cases that the transmission to VRAM within the main routine is interrupted while in progress and the VDP is being accessed during the interruption, the writing to VRAM that was taking place in the main routine cannot function properly, thereby becoming the cause of a bug. Accordingly, when accessing towards VDP in the main routine, it is necessary to deal with it by using either 1 or 2.

1 Allow the main routine and the interrupt routine to run at the same time.

2 Prohibit interrupt while transmitting in the main routine.

Regarding 1:

When transmitting to VDP in the main routine, establish "a flag that says it is transmitting" and that flag must be confirmed in the interrupt routine. Access VDP only when VDP is not being accessed in the main.

Regarding 2:

When transmitting to VDP in the main routine, prohibit interrupt before performing the address set for the transmission, and permit interrupt after the data transmission has ended.

## 2. Regarding the Operation CHECK While Using EP-ROM

The enlarging of game capacity is anticipated hereafter as a trend, but in order to be certain that the accompanying EP-ROM operation is stabilized, be cautious of the following items.

1 Decrease the number of ROM that loads onto the ROM baseboard as much as possible by using large capacity devices.

Because the electric power consumption of ROM increases when the number of loads on the ROM increases, it is burdened and can be the cause of an operation malfunction.

The maximum number of possible loads varies according to:

- the ROM being used
- the type of base board being used
- the type of device being used

and cannot be specified but if kept about 4 will remain stabilized.



(Translated Bv: DDP. June 28. 1993)

2 Because the electric power consumption is great, EP-ROM's whose maximum use should be avoided are

•AMD Company made products
•INTEL Company made products

Specifically, the new product "Mega Drive 2" has a severe design on the electric power consumption. In the case that the worst operation malfunction occurs, check to see if 1 or 2 is relevant and can be applied. Furthermore, do all that can be done in the normal operation times to establish the operation circumstances referred to in the previous section.

## 3. Regarding the Production of Games over 16 Mbits

By using the present cartridge software corresponding to MEGA DRIVE, the capacity where present development is possible is as follows:

•16Mbit+256Kbit(max)
000000H~1FFFFFH(ROM) + 200000H~20FFFFH(BUP)

The methods corresponding to capacities larger than this are written on the last page. Development is in progress (release date pending).

If there is no backup, it is possible to develop up to the following capacity without using bank switch.

•Up to 32Mbit
000000H~3FFFFFH(ROM)

The S-RAM board is presently under development (release scheduled for summer of '93).



(Translated Bv: DDP. June 28. 1993)

# Regarding MEGA DRIVE Bank Switching

Divide all 68,000 (ROM) space, every 4 Mbit.
(Bank 0 - bank 63; MAX 256 Mbit)

Divide the MD cartridge also into 8 areas of 4 Mbit and fix only area 0 that has vectors and it is possible to assign option banks to the remaining 7 areas.

The bank should be designated by the bank establishing register (odd number place of MD's $A13OF1~$A13OFF).

Register 0's 0th bit switches ROM/RAM after the $200000th place, and the number 1 bit designates the RAM light protect.

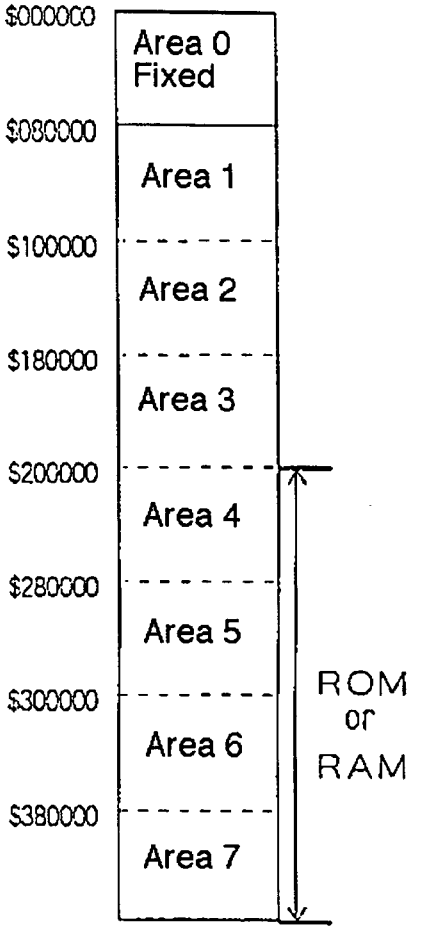
Register 1 thru register 7 corresponds to area 1 thru area 7.

The initial that depends on the hardware at the time of reset turns the light protect OFF (write in possible), as the cartridge area becomes 32 Mbit ROM space.



(Translated Bv: DDP. June 28. 1993)

MD Cartridge Area




Bank Setting Register

| | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|
| REG. 0 $A130F1 | 0 | 0 | 0 | 0 | 0 | 0 | 0:W 1:P | 0:ROM 1:RAM |
| REG. 1 $A130F3 | 0 | 0 | BN5 | BN4 | BN3 | BN2 | BN1 | BN0 |
| REG. 2 $A130F5 | 0 | 0 | BN5 | BN4 | BN3 | BN2 | BN1 | BN0 |
| REG. 3 $A130F7 | 0 | 0 | BN5 | BN4 | BN3 | BN2 | BN1 | BN0 |
| REG. 4 $A130F9 | 0 | 0 | BN5 | BN4 | BN3 | BN2 | BN1 | BN0 |
| REG. 5 $A130FB | 0 | 0 | BN5 | BN4 | BN3 | BN2 | BN1 | BN0 |
| REG. 6 $A130FD | 0 | 0 | BN5 | BN4 | BN3 | BN2 | BN1 | BN0 |
| REG. 7 $A130FF | 0 | 0 | BN5 | BN4 | BN3 | BN2 | BN1 | BN0 |

BN0～BN5 : Bank Number

REG.0 , D1 • • 0 : WRITE
1 : PROTECT



(Translated Bv: DDP. June 28. 1993)

SEGA OF AMERICA, INC.
Consumer Products Division

# GENESIS TECHNICAL BULLETIN #27

To: SEGA & Third Party Developers

From: Technical Support Group

Date: January 24, 1994

Re: Genesis Technical Information

==========

The following tech note contains information on Genesis peripheral devices.

130 Shoreline Drive, Redwood City, CA 94065 • (415) 508-2800 • FAX (415) 802-1448

1. **Warning regarding control pad data reads**

   For both the 3 and 6 button pads, the pad data is determined 2µsec after `TH` is modified. Therefore, as shown in the sample below, read data from the pad 2µsec (4 nop's) after `TH` is modified.

## Joypad Reads

Pad data from the 3 and 6 button controllers are read 2µsec after `TH` is modified. The wait is necessary because the data in the chip needs time to stabilize after `TH` is modified. If data is read without this wait, there is no guarantee that the data will be correct.

Moreover, the 2µsec time is equivalent to 4 `nop`, including the 68000's `prefetch`.

```
Z80BusReq   equ   $A11100
Z80Reset    equ   $A11200
;*************************************************************************
;
; _JSID         joystick id get.
;               This routine essentially performs connection check of the Genesis I/O
;               ports.  Figures out if some kind of controller is attached to I/O port and
;               distinguishes between different classes (regular vs handshaking)
;               controllers.  This identification scheme is based on Addendum #1 in the
;               Genesis Software Manual
; in:           d1.w  :       port number (0...2)
; out:          d0.l== ID
;                       $0d     MD FP6B(six button) or MD 3button
;                       $03     SOA/SOJ mouse
;                       $07     SEGA TEAM PLAYER
;                       $ff     input error(passed incorrect port number)
;                       $0f     unknown or nothing connected
;*************************************************************************
;
_JSID:
        cmp.b       #3,d1
        blt         _JSID_10
        moveq       #-1,d0
        rts
_JSID_10:
        movem.l     d2-d3/a0,-(sp)
        move.w      #$100,Z80BusReq     ;Z80 bus request
        move.w      #$100,Z80Reset      ;Z80 reset line high
        btst.b      #0,Z80BusReq        ;Z80 bus grant acknowledge
        bne.s       *-8                 ;wait until bus granted
        move.l      #$00a10003,a0
```

```
moveq       #0,d0
move.b      d1,d0
add.l       d0,d0
adda.l      d0,a0
moveq       #0,d0
move.l      d0,d1
move.l      6(a0),d3            ;save original status of control port
swap        d3
move.b      (a0),d3             ;original status of I/O port
move.b      #%01000000,6(a0)    ;I/O control : TH (PD6) output
nop
nop
nop
nop
move.b      #%01000000,(a0)             ;set TH=1(high)
nop
nop
nop
nop
move.b      (a0),d0
move.b      #%00000000,(a0)             ;set TH=0(low)
nop
nop
nop
nop
nop
move.b      (a0),d1
move.b      #%01000000,(a0)             ;set TH=1(high)
move.w      d0,d2
lsr.b       #1,d0
ror.w       #1,d0
lsr.b       #1,d0
rol.w       #1,d0
ror.w       #1,d2
lsr.b       #1,d2
rol.w       #1,d2
or.b        d2,d0               ;ID2=(PD6=1) & (PD1 OR PD0)
and.w       #$03,d0
lsl.b       #2,d0
move.w      d1,d2
lsr.b       #1,d1
ror.w       #1,d1
lsr.b       #1,d1
rol.w       #1,d1
ror.w       #1,d2
lsr.b       #1,d2
```

```
rol.w       #1,d2
or.b        d2,d1           ;ID0=(PD6=0) & (PD1 OR PD0)
and.w       #$03,d1
or.b        d1,d0           ;return id value
move.b      d3,(a0)         ;restore original status of I/O port
swap        d3
move.b      d3,6(a0)        ;restore original status of I/O control port
move.w      #0,Z80BusReq    ;Z80 bus release
movem.l     (sp)+,d2-d3/a0
rts
```

## 2. Warning regarding Sega Mouse data requests

After the 10th data read, always issue an `END DATA` command. Set both TH & TR bit high to issue an `END DATA` command(see example below). Abnormal operation may occur if data beyond the 11th state is read.

```
*************************************************************************
* MOUSE_GET       mouse/tablet driver
* THIS DRIVER READS THE SEGA MOUSE
* Access this routine from within the V_INT.
* The Z80 Enable/Disable code is provided as a safety check, to
 * prevent unfavorable bus access.
*
* in     d1.w : port number ( 0..2 )
* out:  d0.l & d2.l
*        d0.l contains connection status and reserved bits
*        d2.l contains MOUSE bits
*
*        d0.l = xxxx|xxxx|xxxx|xxxx|t1 m1 1 1|1 1 1 1|1 1 1 1
*        If mouse is connected, value returned in d0.l is as follows:
*        d0.l = xxxx|xxxx|xxxx|xxxx|1 0 1 1|1 1 1 1|1 1 1 1
*        if mouse connected, bit t1 =1 m1=0
*        if tablet connected, bit t1=0 m1 =1
*        note bits t1 and m1 cannot simultaneously be zero.
*
*        In case of error(nothing connected,other device, or incorrect port value)
*        the driver returns 0 in register d0.l.  If mouse/tablet is actually
*        connected, the value in d0.l cannot be zero.
*
*        In case of timeout, the driver sets bit 31 in d2.l, and returns 0 in d0.l
*        MOUSE DATA is returned in register d2.l as follows:
* d2.l = 0xxx|xxxx|YoXoYsXs|CMRL|X7X6X5X4|X3X2X1X0|Y7Y6Y5Y4|Y3Y2Y1Y0
```

```
*
*       x denotes don't care bits
*       Yo (Yover) 1 when Y data exceeds 255(ffh)
*       Xo (Xover) 1 when X data exceeds 255(ffh)
*       Ys (Ysign) 0 when Y data is positive, 1 when negative
*       Xs (Xsign) 0 when X data is positive, 1 when negative
*       C Center button (START/PAUSE)
*       M Middle button (TRG-B)
*       R Right button (TRG-C to CANCEL)
*       L Left button (TRG-A to SELECT)
*       X7-X0 absolute values of movement in X direction
*       Y7-Y0 absolute values of movement in Y direction
*       REGISTERS: register d1 is exclusively used for input to the routine.
*                  register d0 & d2 is used for output
*************************************************************************
MOUSE_GET:
        movem.l     d1/d3/d4/d7/a0,-(sp)
        move.w      #$100,Z80BusReq
        move.w      #$100,Z80Reset
        btst.b      #0,Z80BusReq
        bne         *-8
        moveq       #0,d0                   * error flag
        cmp.w       #$0002,d1
        bhi         MOUSE_EXIT
        add.w       d1,d1
        move.l      #$00a10003,a0

MOUSE_CONNECT:
        move.b      #$60,6(a0,d1.w)
        nop
        nop
        move.b      #$60,(a0,d1.w)          * TH,TR=11 (END  DATA command)
        moveq       #0,d2
        moveq       #0,d3
mouse_rdy_lp:
        btst.b      #4,(a0,d1.w)
        beq.b       mouse_rdy_lp
        move.b      (a0,d1.w),d4            * d4.b=? 1 1 1|0 0 0 0
        and.b       #$0f,d4
        tst.b       d4
        bne         MOUSE_EXIT              * nothing connected/differnt controller
        move.b      #$20,(a0,d1.w)          * select t1 m1 1 1
        move.w      #$fe,d7                 * timeout parameter
mouse_lp1:
        btst.b      #4,(a0,d1.w)
```

```
        bne.b       MOUSE_10
        dbra        d7,mouse_lp1
        bra         MOUSE_ERR

MOUSE_10:
        move.b      (a0,d1.w),d0        * d0 = xxxx|xxxx|xxxx|t1 m1 1 1
        lsl.w       #8,d0               *d0 = xxxx|t1 m1 1 1|0000|0000
        move.b      #$00,(a0,d1.w)      * select 1(rsv1) 1(rsv2) 1(rsv3) 1(rsv4)
        nop
mouse_lp2:
        btst.b      #4,(a0,d1.w)
        beq.b       MOUSE_20
*       bne.b       MOUSE_20
        dbra        d7,mouse_lp2
        bra         MOUSE_ERR

MOUSE_20:
        move.b      (a0,d1.w),d3        * d3 = xxxx|xxxx|xxxx|1 1 1 1
        move.b      #$20,(a0,d1.w)      * select 1(rsv5) 1(rsv6) 1(rsv7) 1(rsv8)
        lsl.w       #8,d3               * d3 = xxxx|1 1 1 1|0000|0000
mouse_lp3:
        btst.b      #4,(a0,d1.w)
        bne.b       MOUSE_30
        dbra        d7,mouse_lp3
        bra         MOUSE_ERR

MOUSE_30:
        move.b      (a0,d1.w),d3        * d3 = xxxx|1 1 1 1|xxxx|1 1 1 1
        lsl.b       #4,d3
        lsr.w       #4,d3
        move.b      #$00,(a0,d1.w)      * select Yo Xo Ysgn  Xsgn
        or.w        d3,d0               * d0 = xxxx|t1 m1 1 1|1 1 1 1|1 1 1 1
        moveq       #0,d3
mouse_lp4:
        btst.b      #4,(a0,d1.w)
        beq.b       MOUSE_40
        dbra        d7,mouse_lp4
        bra         MOUSE_ERR

MOUSE_40:
        move.b      (a0,d1.w),d2        * d2 = xxxx|xxxx|xxxx|Yo Xo Ys Xs
        move.b      #$20,(a0,d1.w)      * select Center Middle Right Left
        lsl.w       #8,d2               *d2 = xxxx|Yo Xo Ys Xs|xxxx|xxxx
mouse_lp5:
        btst.b      #4,(a0,d1.w)
```

```
        bne.b       MOUSE_50
        dbra        d7,mouse_lp5
        bra         MOUSE_ERR

MOUSE_50:
        move.b      (a0,d1.w),d2        * d2 = xxxx|Yo Xo Ys Xs|xxxx|C M R L
        move.b      #$00,(a0,d1.w)      * select X high X7 X6 X5 X4
        lsl.b       #4,d2
        lsl.w       #4,d2               * d2 = Yo Xo Ys Xs|C M R L|xxxx|xxxx
mouse_lp6:
        btst.b      #4,(a0,d1.w)
        beq.b       MOUSE_60
        dbra        d7,mouse_lp6
        bra         MOUSE_ERR

MOUSE_60:
        move.b      (a0,d1.w),d2      * d2 = Yo Xo Ys Xs|C M R L|xxxx|X7 X6 X5 X4
        move.b      #$20,(a0,d1.w)        * select X low X3 X2 X1 X0
        lsl.b       #4,d2
        lsl.l       #4,d2       *d2 = xxxx|Yo Xo Ys|C M R L|X7 X6 X5 X4|xxxx|xxxx
mouse_lp7:
        btst.b      #4,(a0,d1.w)
        bne.b       MOUSE_70
        dbra        d7,mouse_lp7
        bra         MOUSE_ERR

MOUSE_70:
        move.b      (a0,d1.w),d2  * d2 = YoXoYsXs|CMRL|X7X6X5X4|xxxx|X3X2X1X0
        move.b      #$00,(a0,d1.w)        * select Y High Y7 Y6 Y5 Y4
        lsl.b       #4,d2
        lsl.l       #4,d2
        *d2=YoXoYsXs|CMRL|X7X6X5X4|X3X2X1X0|xxxx|xxxx
mouse_lp8:
        btst.b      #4,(a0,d1.w)
        beq.b       MOUSE_80
        dbra        d7,mouse_lp8
        bra         MOUSE_ERR

MOUSE_80:
        move.b      (a0,d1.w),d2
        * d2 = YoXoYsXs|CMRL|X7X6X5X4|X3X2X1X0|xxxx|Y7Y6Y5Y4
        move.b      #$20,(a0,d1.w)        * select Y Low Y3 Y2 Y1 Y0
        lsl.b       #4,d2
        lsl.l       #4,d2
        *d2 = YoXoYsXs|CMRL|X7X6X5X4|X3X2X1X0|Y7Y6Y5Y4|xxxx|xxxx
```

```
mouse_lp9:
        btst.b          #4,(a0,d1.w)
        beq.b           MOUSE_90
        dbra            d7,mouse_lp9
        bra             MOUSE_ERR

MOUSE_90:
        move.b          (a0,d1.w),d2
        *d2=YoXoYsXs|CMRL|X7X6X5X4|X3X2X1X0|Y7Y6Y5Y4|xxxx|Y3Y2Y1Y0
        move.b          #$60,(a0,d1.w)          * TH TR =11(END  DATA command) and exit
        lsl.b           #4,d2
        lsr.l           #4,d2
        *d2= xxxx|xxxx|YoXoYsXs|CMRL|X7X6X5X4|X3X2X1X0|Y7Y6Y5Y4|Y3Y2Y1Y0
mouse_lp10:
        btst.b          #4,(a0,d1.w)
        beq.b           mouse_lp10
        or.l            #$00000000,d2
MOUSE_EXIT:
        move.w          #0,Z80BusReq
        movem.l         (sp)+,d1/d3/d4/d7/a0
        rts

MOUSE_ERR:
        move.b          #$60,(a0,d1.w)
        nop
        nop
mouse_erlp:
        btst.b          #4,(a0,d1.w)
        beq.b           mouse_erlp
        or.l            #$80000000,d2
        moveq           #0,d0
        move.w          #0,Z80BusReq
        movem.l         (sp)+,d1/d3/d4/d7/a0
         rts
```

## 3. Warning regarding the Sega infrared 6 button cordless pad

The following software issues were discovered prior to the release of the Mega Drive infrared 6 button cordless pad. Detailed technical specifications regarding the infrared pad will follow at a later date.

- **Change in the method to switch between the Fighting Pad 6B and the 3B Pad.**
The mode switch is made by the operation described below. Do not implement the following button combinations in game controls.

1. After installing the infrared 6 button pad receiver on the Mega Drive, turn on power to the Mega Drive.
2. When the receiver has power, push the **MODE**, **X** and **C** buttons on the cordless 6 button pad simultaneously.
3. The controller is now in 3 button mode.

## 4. Warning regarding Fighting Pad 6B (hereafter FP6B) data

The shaded data in the table below is used for expansion uses. The data is not currently used by the FP6B. The data there is invalid, therefore, please do not use them.

| | TH(bit6) | TR(bit5) | TL(bit4) | R(bit3) | L(bit2) | D(bit1) | U(bit0) | Comments |
|---|---|---|---|---|---|---|---|---|
| 1 | Hi | TRG-C | TRG-B | RIGHT | LEFT | DOWN | UP | |
| 2 | Low | START | TRG-A | 0 | 0 | DOWN | UP | |
| 3 | Hi | TRG-C | TRG-B | RIGHT | LEFT | DOWN | UP | |
| 4 | Low | START | TRG-A | 0 | 0 | DOWN | UP | |
| 5 | Hi | TRG-C | TRG-B | RIGHT | LEFT | DOWN | UP | |
| 6 | Low | START | TRG-A | 0 | 0 | 0 | 0 | Recognition |
| 7 | Hi | TRG-C | TRG-B | MODE | TRG-X | TRG-Y | TRG-Z | Expansion |
| 8 | Low | START | TRG-A | 1 | 1 | 1 | 1 | " " |
| 9 | Hi | TRG-C | TRG-B | ---- | ---- | ---- | ---- | |

Fighting Pad 6B Data Table

## 5. Cartridge Information (Corrections and Additions to Genesis Technical Bulletin #26, Sec. 3.)

The data "-yy" entered in cartridge information ID 18Bh~18Dh is the *version number* of the game. In some games however, the data suffix "-zz" as shown in numbers 2) and 5) below denotes the *sales territory* for the cartridge. Please make sure not to confuse the two.

### Explanation

There are currently 5 product number types.

1) T-xxxxx ; third party brand <Japan>
2) T-xxxxx-zz ; third party brand <international>
3) G-†††† ; Sega brand <Japan>
4) MK-OOOO ; Sega brand <US>
5) MK-OOOO-zz ; Sega brand <international other than US>

For European third party products, the product number is "T-xxxxx-50" (zz=50). Only "T-xxxxx" is necessary when the information is entered in the cartridge (-50 is unnecessary).

### Example

If the product number is "T-12345-50" (European version), the cartridge information is as follows:

**Incorrect**

```
180h:"GM T-12345 -50xx"  *Disk kind, good number, ver no.
checksum
```

**Correct**

```
180h:"GM T-12345 -00xx"  *Disk kind, good number, ver no.
checksum
```

**Note:** The information contained here is not included in any previously released documentation.

SEGA OF AMERICA, INC.
Consumer Products Division

# GENESIS TECHNICAL BULLETIN #28

To: SEGA & Third Party Developers

From: Technical Support Group

Date: January 31, 1994

Re: Genesis Technical Information

==========

1. In accordance with an increased number of Mega Drive peripheral devices, I/O port usage data is added at 190H of the cartridge ID.
   For more information on I/O port usage data, please see Page 1 of the "GENESIS SOFTWARE DEVELOPMENT MANUAL COMPLEMENT."

   Items added

   - Sega mouse "M"
   - Fighting pad 6B "6"
   - Multi-tap "4"

   Setting Conditions

   - Fighting Pad 6B
     Only when the XYZ buttons are effective. For the rest, enter "J" in the usual manner.

   - Multi-tap
     When multi-tap multi-switch is on.
     (When data can be read with multi-switch turned on even if it is not geared for the multi-play.)

130 Shoreline Drive, Redwood City, CA 94065 • (415) 508-2800 • FAX (415) 802-1448

Multi-Tap Supplement

- Multi-tap switching setup has the following functions:

A~D : Terminal selectors (any one terminal can be selected)
MULTI: Used with games designed for more than 3 players

2. For the country information to be added at 1FOH of the Mega Drive cartridge ID, please input such data in accordance with A10001H of the I/O port.

| | | | |
|---|---|---|---|
| BIT 7 | 0: | Domestic } | Input as "J." |
| BIT 6 | 0: | NTSC } | Japan/Korea/Taiwan sales hardware |
| BIT 7 | 1: | Overseas } | Input as "U." |
| BIT 6 | 0: | NTSC } | US/Canada sales hardware |
| BIT 7 | 1: | Overseas } | Input as "E." |
| BIT 6 | 1: | PAL } | Europe/Hong Kong sales hardware |

Note: For the above I/O port, see Page 72 of the "GENESIS SOFTWARE MANUAL."

SEGA OF AMERICA, INC. 275 Shoreline Drive, Redwood City, CA 94065 • (415) 508-2800

# GENESIS TECHNICAL BULLETIN #29

To: SEGA & Third Party Developers

From: Technical Support Group

Date: March 10, 1994

Re: Genesis Technical Information

==========

**The information on the bottom of page 2 of the Genesis Sound Software Manual is incorrect. It should be changed to:**

**CONFIRMATION OF BUS STATUS:**

This infomation is in $A11100, bit 8. (not bit 0 as stated in the manual)

0 = Z80 is inactive, 68k can access
1 = Z80 is active

(Please note that this is reversed from what is in the manual.)

SEGA™
SEGA OF AMERICA, INC.
Consumer Products Division

# GENESIS TECHNICAL BULLETIN #29A

To: SEGA & Third Party Developers

From: Technical Support Group

Date: April 11, 1994

Re: Genesis Technical Information

==========

**This is a clarification on Genesis Technical Bulletin #29.**

The information on the bottom of page 2 of the Genesis Sound Software Manual is incorrect. It should be changed to:

**CONFIRMATION OF BUS STATUS:**

This infomation is in $A11100, bit 8 when accessed by WORD and bit 0 by BYTE.

0 = Z80 is inactive, 68k can access
1 = Z80 is active

(Please note that this is reversed from what is in the manual.)

SEGA OF AMERICA, INC. Product Development • 150 Shoreline Drive, Redwood City, CA 94065

# GENESIS TECHNICAL BULLETIN #31

**To:** **Sega and Third Party Developers**

**From:** **Developer Technical Support**

**Date:** **November 22, 1994**

**Re:** **Genesis ID table update**

---

1. Changes to cartridge data (ID) specifications
   Please note that the specifications below also apply to the Genesis 32X.

The following changes have been made to the code entered in 1FOh.

- Old: Country code data
- New: 1 byte ASCII hardware enable code in 1FOh.
  1F1h to 1FFh are filled with the "space" character code (20h).
- Explanation
  The hardware enable code is used when the application reads the hardware data information in $A10001 in order to perform territory lockout. The code is used by SEGA to check hardware and software compatibility.

| $A10001 | | | Main Sales | Hardware Enable Code (numbers from 0 to F below) | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Bit 7 | Bit 6 | Hardware Type | Territories | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| 0 | 0 | Japan, NTSC | Japan, S. Korea, Taiwan | X | O | X | O | X | O | X | O | X | O | X | O | X | O | X | O |
| 0 | 1 | Japan, PAL | | X | X | O | O | X | X | O | O | X | X | O | O | X | X | O | O |
| 1 | 0 | Overseas, NTSC | N. America, Brazil | X | X | X | X | O | O | O | O | X | X | X | X | O | O | O | O |
| 1 | 1 | Overseas, PAL | Europe, Hong Kong | X | X | X | X | X | X | X | X | O | O | O | O | O | O | O | O |

↑ Example 2 Example 1 ↑

O: Enable X: Disable

Ex. 1) The hardware enable code "F" is entered in applications that do not limit hardware compatibility (i.e., a universal compatibility ROM). As can be seen in the table above, the hardware enable code "F" is compatible with all hardware types.

Ex. 2) For applications that are compatible only with the US version of the Genesis, the hardware enable code "4" is used. The application is compatible only with hardware that shows Bit 7 = 1 and Bit 6 = 0 at $A10001.

| Address | 1F0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Hardware Enable Code | 4 | | | | | | | | | | | | | | | |
| ASCII Code | 34 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 |

**NOTE:** Software containing errors in the code above will not be acceptable for final master ROM release purposes.

SEGA OF AMERICA, INC. Product Development • 150 Shoreline Drive, Redwood City, CA 94065

# GENESIS TECHNICAL BULLETIN #32

**To:** **Sega and Third Party Developers**

**From:** **Developer Technical Support and Catapult Entertainment**

**Date:** **30-Jan-95**

**Re:** **Getting Games Ready For XBAND™**

NOTE: This document is provided to Sega Certified Developers under the confidential terms of your Sega Developer Agreement.

---

## 1. Introduction

This document provides an introduction to the XBAND Video Game Modem and Network, and a preliminary overview of game design guidelines that will make it easier for your game to be adapted for network play. If you are developing a new multi-player game that you would like to be compatible with the XBAND Network, or if you would like the XBAND Network to be compatible with an existing multi-player game, this is the first document that you should read.

NOTE: Many of the technologies described in this document are proprietary to Catapult Entertainment, Inc. and are either patented or patent-pending.

## 2. How to reach XBAND

Catapult Entertainment, Inc.
c/o Developer Programs
20823 Stevens Creek Blvd.
Cupertino, CA 95014

Voice: 408.366.1735
Fax: 408.366.1729
Internet: developers@catapent.com

## 3. What is XBAND?

XBAND is made up of two major components: The XBAND Video Game Modem and the XBAND Network. The XBAND Video Game Modem is a device that

plugs in-between a Sega Genesis cartridge slot and a video game cartridge. The XBAND Network is an on-line system built into the XBAND Video Game Modem that automatically matches players to play video games with each other, keeps track of game statistics, provides e-mail and electronic newspapers, and much more.

As a video game developer you will be mostly concerned with the operation and programming of the XBAND Video Game Modem during the execution of your video game. The information you will need regarding this starts in Section 4.

### 3.1. The XBAND Video Game Modem

The XBAND Video Game Modem is more than just a modem. Physically, it plugs into a Sega Genesis cartridge slot and accepts a video game cartridge on a connector on the top. The phone line plugs into an RJ-11 jack on the side. There is also a slot to accept an ISO 7816-compliant Smart Card, whose use will be explained later, and a "pass-through" switch which allows you to play the attached cartridge in single-player mode.

The XBAND Video Game Modem has 512KBytes (4 MBits) of ROM that holds the XBAND OS (XOS) code, XOS graphics and sound data, and the XOS GameTalk routines (which can be directly called by game cartridges). The XBAND Video Game Modem also contains 64KBytes (512KBits) of battery-backed SRAM that holds the player's personal information (X-Mail, Player List, Player ID), XOS system patches, cartridge-specific game patches, game results from the last game played, and the XBAND on-line newspapers (BANDWIDTH and XBAND News).

The XBAND Video Game Modem contains a proprietary custom chip, which provides all of the logic functions for the operation of the XBAND Video Game Modem and also the means for controlling the execution of plugged-in cartridges.

Finally, the XBAND Video Game Modem contains a special modem chip that provides low-latency, noise-tolerant communications as well as special detection mechanisms for Call Waiting and other noise events. This allows the modem to run at only 2400 bits per second, transmitting a maximum of 4 bytes per frame.

### 3.2. The XBAND Network

The XBAND Network is different than traditional on-line systems. If a player of a traditional on-line service wishes to play a game with another person, the player must be connected to the service's Server for the duration of the game. With XBAND, players are only connected to the XBAND Server for a small percentage of their connect time, resulting in a lower cost and lower latency connection (typically less than 50 msec end-to-end).

When an XBAND player wishes to play against another person, the player's XBAND Video Game Modem connects to the XBAND Server. During the brief connection to the XBAND Server (typically between 10 seconds and a minute), the XBAND Video Game Modem uploads the player's phone number, any outgoing mail the player might have, and the game results from the last game played. Simultaneously, the XBAND Video Game Modem downloads the phone number of an opponent (or a number of minutes to wait for a call), any incoming mail, the two XBAND on-line newspapers, any new game patch that is needed (to be explained below), and any new system updates for the XOS.

Once the transaction is complete, the XBAND Video Game Modem hangs up the phone, and if it downloaded an opponent's phone number, it immediately dials the opponent. If it received instructions to wait for an opponent, it notifies the player of how long it expects to wait, and then waits for the telephone to ring.

Either by dialing out or by receiving an incoming call, the XBAND Video Game Modem connects with another XBAND Video Game Modem. There is a handshake where they exchange player information, then the video game cartridge is started up, the two games synchronize to each other, and the video game is played. Once the game is completed, the XBAND Video Game Modems disconnect, storing the results of the game in battery-backed up RAM, to be stored until one of the XBAND Video Game Modems connects to the XBAND Server again and uploads the data.

Since the phone lines used for XBAND are often used for voice calls or fax machines as well, it's important to gracefully deal with Call Waiting and line noise events. Such disruptions typically cause modems to lose carrier and hang up. The XBAND Video Game Modem, however, will pause the video game once disrupted, and then make every effort to wait out the noise event, restore carrier, and resume the game. Even if the modem has no choice but to hang up, it will redial the opponent and attempt to restore carrier and resume the game. This level of tenaciousness is essential when dealing with shared voice lines. We have found that about one in every three games experiences a line noise event that would have resulted in a lost connection with a conventional modem.

## 4. Getting Network Games to Run In Sync

The essential problem of real-time point-to-point modem game play is maintaining synchronization between the two video game machines. Although the video game hardware and cartridge software are identical on both ends of the phone line, slight variances between the systems as well as the limited timing resolution of the modem data stream creates a situation where it is quite easy for the two machines to drift out of synchronization. Once the two video games are out of sync even slightly, they will drift even further apart as game play continues, and in the end the game will likely have a different outcome.

There is no single technique that will magically maintain synchronization for all games. Catapult has found that there are about as many techniques for maintaining synchronization as there are techniques for designing main game loops. And, we have also come very quickly to the realization that it would be ludicrous to try to impose a single mechanism upon all video games developers. You need to look at the particular architecture of your game, see which issues apply in your case and which issues don't, and make use of the tools and techniques that Catapult has developed. And if that doesn't do the trick, we're here to help you. Many times we've been told that because of this and that a game will never run synchronized over a modem. We have yet to find one that we could not get to work with XBAND.

## 4.1. Why Synchronization Errors Occur

One would think that two identical video game machines running the same software and fed the same controller inputs would have identical execution. Unfortunately, this is not at all the case. There are several reasons why two identical video game machines running identical software will have varying execution behavior. This list is by no means complete, but it covers the major issues that we have encountered.

To begin with, any two free-running systems will run at different speeds. That is to say, the crystal oscillator that governs the clock speed of the video game processor and the timing of the video scanning is only accurate to a certain precision. In practical terms, we have found that the worst case speed differential between two video game systems will cause one machine to get a full frame ahead of the other in about two minutes. So, if your game uses Vertical Blanking Interrupt (VBI) as a timing reference, you will find that given two free-running video game systems, you can only hope to keep both systems on the same frame for about two minutes.

[Fortunately, since VBI synchronization is essential for many games, Catapult has developed a technology (called Sync-O-Tron™) which keeps VBIs locked together for the duration of the game, despite the drift between the two crystal oscillators. This technology usually helps solve game synchronization problems, but it is not always a complete solution. It is discussed later in Section 5.6.]

Another common source of synchronization problems occurs in the critical relationship between VBI and the main game loop. One would think that given identical hardware, identical software, and identical state (e.g. the same controller input), the VBI interrupt for a given frame will occur after the exact same instruction in the main game loop in both machines. If it were the case that there were no other asynchronous events occurring in the video game machine, this indeed would be true. Unfortunately, events such as DRAM refresh and sound DMA are unpredictable and asynchronous to the CPU execution causing the two CPUs to have the same average execution speed, although not the same cycle-by-cycle execution profile. So, while it is the case that VBI will occur at

approximately the same instruction in each main game loop, on a given VBI it might be one or two instructions off. If there is a game state decision that occurs in one of these instructions that is affected by a variable changed during VBI, the game execution will be different on both machines.

This same slight variation in CPU timing can also affect the number of VBI frames per game loop. For example, if the main game loop takes almost exactly three frames to execute, it might finish executing just before VBI on one machine and just after VBI on the other machine. If the number of frames per game loop affects game logic (as it often does), one game will count three frames and the other will count four, causing the two games will exhibit different behavior.

The slight variation in CPU timing can also affect reads from hardware registers that affect game behavior. For example, some games read the current HV counter at various points in the game loop for use as a random number. Clearly, given the execution variances just mentioned, the number fetched could have widely differing values on the two machines.

DMA timing can also introduce variations in execution for many of the same reasons. On the Sega Genesis, when graphics DMA occurs, it shuts down the CPU completely. Normally, DMA is only used during vertical blanking, allowing the CPU to run during active video, but sometimes when games are doing major screen updates (e.g. between segments of the game), they activate DMA continuously until the update is complete. If the DMA continues for more than a frame time, an entire VBI may be missed. And if the DMA completes at approximately the same moment as VBI, it may complete before VBI on one machine, but after VBI on the other machine. This can create a situation where the two machines will count a different number of VBIs resulting in differing game execution.

The game cartridge itself can introduce variances. Many games today have battery-backed cartridge RAM that they use for storing results from previous games. If these results affect game play (as is often the case with sports games), there will be varying execution between the two cartridges. It isn't a reasonable solution to just erase the RAM or update both with the same values. Very often the information stored in the RAM is of great importance to the game player (e.g. hard-earned stats, etc.). So, the game must have a mode of execution where the RAM status is disregarded.

In some situations there is more than one release of the same game cartridge. For example, NBA® JAM™ for Sega Genesis had two releases, and there were significant differences in the code (e.g. one of the basketball players was changed, and some of the timings were changed) between the two releases. A separate Game Patch had to be developed for each version to reconcile the execution differences so that a Rev. 1 could play against a Rev. 2 and stay in sync.

Finally, there is occasionally code that samples player reaction time so precisely that the exact same timing cannot be reproduced on both machines. For example,

a game might enter a spin loop to count the length of time before a player presses start for the purpose of getting a random number seed. The timing resolution of a CPU spin loop is far more precise than the time resolution of a modem, so the two game machines will certainly measure the player reaction time differently, resulting in a different random number seed.

### 4.2. How XBAND Synchronizes Existing Game Cartridges

Multi-player Sega Genesis video games played across XBAND must be carefully designed to 1) accept controller input from the remote Sega Genesis system, 2) maintain synchronized execution between the Sega Genesis systems, and 3) have appropriate options for networked play. Clearly, games which are designed from the start with the knowledge of the XBAND Network and Video Game Modem can design in these capabilities. However, cartridges which are already out on the market require these capabilities to be added in the field. The XBAND Network and Video Game Modem work together to make these in-field extensions possible.

The process goes as follows: When someone hits the "Challenge" button to register for a match, the XOS running on the XBAND Video Game Modem dials into the XBAND Server and uploads the checksum of the cartridge that the player wants to play. The XBAND Server checks to see whether the cartridge is a supported XBAND game, and if it is, it downloads to the XBAND Video Game Modem a "Game Patch" which contains the code necessary to augment the video game cartridge to make it function properly over the modem. When the XBAND Video Game Modem connects to an opponent and the handshake is complete, control is passed to the video game cartridge. The video game cartridge executes as it would normally, except for the changes to the cartridge provided by the Game Patch. These changes route the controller reads through the modem to the remote Sega Genesis machine, resolve synchronization issues between the two systems, and record results of the game execution for later reporting to the XBAND Server. When game completes, the XBAND Video Game Modem hangs up the phone, and control returns to the XOS.

The custom chip in the XBAND Video Game Modem provides a very flexible set of capabilities for augmenting the execution of a game cartridge. The chip permits you to make any change to a cartridge ROM that is needed to make it work on XBAND.

There are a number of ways that a Game Patch is developed for an existing cartridge. Ideally, it is developed by the programmer who wrote the original game cartridge, but often this person is not available. In fact, sometimes the original source code for the game is not even available or is commented in a foreign language, rendering it unreadable. Catapult has found that given the relatively small changes required to augment a game for use on XBAND, it is quite feasible to get the game to work from the object code using reverse-

engineering techniques, and several of the Game Patches on XBAND were developed using these methods.

The following sections cover the basics of game synchronization, and give several examples. If you are designing a new game, we hope that this will cover enough of the "gotchas" to keep it "sync clean". If you are developing a Game Patch for a game that is already in the field, this should give you a starting point in understanding key places where the game must be patched.

### 4.3. Synchronization Approaches

There are two main communication approaches that are used between video games to maintain synchronization: Game State Communication and Controller Communication.

With the Game State approach, game state information (such as player position, number of enemies, number of lives, etc.) is transmitted between the game machines. This approach is commonly used in point-of-view games. The advantage of this approach is that the games truly free run on each machine and data is only sent when there is relevant data to send. For example, if the two players are in different rooms, only information that might affect the other player is transmitted. The disadvantage of this approach is that a great deal of information must sometimes be transmitted in real-time (e.g. if both players are shooting at each other). If you are using this approach for your game, then most of the discussion that follows does not apply to you since you are just trying to achieve identical game state (not identical execution) on both machines. Synchronization issues for Game State approach games are quite game-specific and are not addressed in this document. Please contact Catapult for more information.

With the Controller Communication approach, player game controller input is transmitted between the game machines. This approach is commonly used by games where all players are visible on the screen at the same time such as sports and fighting games. The advantage of this approach is that there is a relatively small, fixed amount of data that must be transmitted, allowing the data to be transmitted every frame time, thereby allowing real-time response. The disadvantage of this approach is that it requires identical real-time game logic execution on both game machines. The following discussions review the issues surrounding the synchronization of Controller Communication approach games.

# 5. Game Design Guidelines

This section provides a number of guidelines for game design that either avoid synchronization problems or suggest architectures in which the synchronization problems can be controlled.

## 5.1. Random Number Generators

DO:

• Implement a single random number generator routine in your code that starts from a settable seed and follows a reproducible sequence (mathematical algorithms work best). The random number generator should only be called by a single thread of execution (i.e. either the main game loop or the VBI routine, but not both). One of the game machines can determine the initial seed by any means you wish, and then it will transmit to the other game machine.

DON'T:

• Don't scatter random number functions throughout your code. Call to a central routine!

• Don't implement an irreproducible random number sequence. For example, reading the HV Counter or using whatever register D0's value happens to be upon entry to the VBI routine are both no-no's.

• Don't call the random number generator from more than one thread of execution (e.g. from both the main game loop and the VBI routine). VBI will occur at an unpredictable point in the main game loop. If the random number generator is being called (or the seed is being updated) in both threads of execution, the sequence in which it is being called will be unpredictable.

## 5.2. Battery-Backed Cartridge RAM Usage

DO:

• Provide a mode whereby the current contents of the battery-backed cartridge RAM on your cart are not utilized by the game logic. Careful! Make sure that in this mode none of the game logic considers the value of the RAM.

DON'T:

• Don't leave stray references to the battery-backed RAM in your code for modes that supposedly don't consider the RAM (e.g. Exhibition mode in sports games). Such stray references result in subtle synchronization bugs that only come up when one player has a particular value in the RAM and the other player does not. They are a hassle to track down.

### 5.3. Sega Genesis DMA Usage

DO:

• Utilize DMA as you normally would in a game, but make sure the DMA is complete and you have returned to the main game loop when the next VBI occurs.

DON'T:

• Don't have giant DMA operations that extend through one or more VBIs. This may result in a different number of VBIs occurring in each game machine.

• If you decide how many DMA operations you do during a VBI by checking whether vertical blanking is over or not, don't use the number of completed DMA operations in your game logic. For example, you are in the VBI routine, you are doing a number of DMA operations, and before each DMA operation you check to see whether you've run out of vertical blanking time. Because of variances in execution speed, one game machine may report that vertical blanking is over, and the other one may report the opposite. If the main game loop makes game logic decisions based on the number of completed DMA operations, the games will get out of sync.

### 5.4. Controller Reads and the Main Game Loop

DO:

• Ideally, read the controllers only once per game loop, before the start of the game loop. It doesn't matter whether you read the controllers during the VBI routine or in the main game loop (although Sega encourages you to read the six-button controllers and the multi-play adapter during VBI).

• Often, a game loop is so long (e.g. 3 frame times or more) that the controllers must be read more frequently than once per game loop to provide adequate responsiveness. There are two approaches to handling this situation:

(a) sample the controllers each VBI, but create a data structure that reflects the controller behavior through the duration of the game loop (e.g. Up on 2nd frame, A and Right on 3rd frame), then feed this data structure, all at once, into the next game loop. This data structure, rather than raw controller data, will be transmitted to the other game machine, once per game loop. This is the preferred method.

(b) sample the controllers each VBI, but only feed in the updated controller information at predictable points in the main game loop. For example, if you have 16 objects on the screen that are updated in sequence in the main game loop, and it typically takes 8 frame times to update them all, feed in each VBIs new controller value after every two objects are updated. Take

care to design your code such that the game loop is not usually held up waiting for VBI. There are a number of other subtleties that must be considered since we can't be certain that both game machines will have the same number of VBIs per game loop. Overall, this is a doable, but tricky, solution to the problem.

DON'T:

• Don't sample the controllers asynchronously to the main game loop and feed in their values at unpredictable points. It is critical that both machines utilize the same controller data for the same game logic. Since VBI will occur at unpredictable points in the game loop, controller values that are sampled at VBI and immediately fed into the game loop may result in differing game logic between the two game machines.

### 5.5. Game Loop Duration and Game Logic

Game loops typically are longer than one VBI in duration. Unfortunately, it is often very difficult to predict how many VBIs of duration a particular game loop will take, and in fact, for the execution reasons mentioned above, it is quite possible that the two game machines may take a different number of VBIs to execute the same game loop code.

Also, there are situations when the game logic execution is the same on both machines, but the display code execution is vastly different. Point-of-view games, such as racing games, that are split screen when played on a single video game machine, can be full-screen on each game machine when played on XBAND. The game logic is the same, since the state of both players is updated on both machines, but the display code for each player's point-of-view could be vastly different. Consequently, the execution time per game loop on each machine can be different.

If you can predict the number of VBIs that will elapse in your game loop easily (i.e. both machines can determine in advance the worst case number of frames per a given game loop), that simplifies the synchronization problem significantly. But if you can't (don't feel bad, most games can't), and you need to consider the number of elapsed VBIs in your game logic (e.g. for animation speed regulation), Catapult has developed a synchronization technology that will maintain sync between the two machines, provided you stay within the DOs and DON'Ts guidelines.

DO:

• Ideally, create a game loop with a predictable number of VBIs. For example, if you have a list of routines you need to go through in your game loop, and you have a worst case execution time for each routine, you might be able to determine how many VBIs will pass in the game loop. If you happen to complete

the game loop sooner than worst-case, you would wait it out until the predicted number of VBIs has elapsed.

• If you don't have a predictable number of VBIs per game loop, and the number of elapsed VBIs is used as part of the game logic (e.g. for animation speed regulation), then make sure that you use a single variable that contains the elapsed VBIs of the *previous* game loop. This variable will be controlled by a synchronization mechanism available from Catapult to maintain synchronization between the two game machines.

DON'T:

• Don't use the number of VBIs elapsed in the *current* game loop in your game logic. If you need to use elapsed VBIs for speed regulation, use the elapsed VBI count from the *previous* game loop, and make sure you use a single variable to store the value.

### 5.6. Using Sync-O-Tron™

Sync-O-Tron is a technology developed by Catapult that maintains synchronization between the VBIs of two video game systems. The technology is available for your use by your game while it is running on the XBAND Network. It provides two key advantages.

First, Sync-O-Tron often simplifies the development of synchronized game loops. Without Sync-O-Tron two free-running game machines can drift apart by an entire frame within 2 minutes. With Sync-O-Tron you are guaranteed that both game machines will always be frame synchronous to each other.

Second, Sync-O-Tron reduces communications latency. Without Sync-O-Tron there might be as much as a full frame of skew between the two game machines. To allow for this worst-case scenario it is necessary to add an additional frame of latency to the communications. With Sync-O-Tron there is always a fixed relationship between the VBIs of the two systems and minimum latency can be achieved.

Sync-O-Tron requires a very small routine that runs in VBI once per game loop.

# 6. User Interface Considerations

When a multi-player game is played over a telephone line by people who don't know each other (and can't talk to each other), there are a number of user interface issues that need to be considered. Although it is possible to get existing multi-player games to run "as is" on XBAND, we have found certain user interface changes are essential to avoid disputes between players and dissatisfaction with game play. These issues are discussed below:

## 6.1. Fair Access to Critical Controls

• Have a means for either player to gain access to critical controls without the other player interfering. Don't create a situation where one player who is faster at pushing buttons can prevent another player from accessing an important control screen.

• Have a reasonable time-out for every screen so that players don't tie up the game indefinitely. Don't create a situation where both players are required to push a button for the game to continue. One player may deliberately try to delay the game just to be obnoxious. Also, sometimes players are out of the room when the game starts up (which means they forfeit). It is fair that one player should be able to play the game from beginning to end if the other player is not there.

## 6.2. Tournament Mode

• Provide a tournament mode which has locked game play options (e.g. fixed game duration, secret power-up codes disabled, battery-backed cartridge RAM disabled, etc.). Don't leave it up to the players to amicably agree on settings for modem play. Decide on what is an appropriate tournament configuration, and lock it down.

• Make sure any options that should be available during a tournament should be settable by team. Don't provide options that affect both teams in tournament mode. The players will literally go back-and-forth, fighting over the option screen. If options can't be settable by team, then lock it down to a fixed setting.

## 6.3. Non-gameplay Features during Gameplay

• Provide reasonable limits to non-gameplay features that may delay the game (or annoy the other player). For example, if you provide an instant replay feature, you might want to limit each player to one instant replay per quarter. Don't expect the players to discipline themselves in the use non-gameplay features. When they are losing, some players will do anything to annoy the winning players.

### 6.4. Call Waiting and Line Noise Dialog Boxes

• Provide a means to display text during gameplay to alert the player about Call Waiting or Line Noise. The text will only be displayed when the game is frozen, so if characters need to be moved off the screen during the display of text, they can be moved back after the text has been removed from the screen. Don't rely on players to guess what is going on when the screen freezes during a Line Noise event. Also, if you cannot display text during Call Waiting, then the game must be terminated when the Call Waiting event occurs. When the player is done with the phone call, the game must start again from the beginning.

### 6.5. 4-Player Controller Allocation

• Make sure that during tournament play the controllers are allocated as controllers 1 and 2 against 3 and 4. Although it is interesting to split players in different configurations, it is very confusing, and usually results in players not knowing what is going on. Don't assume that the players can talk to each other to decide on a controller configuration.

• Make sure that all 4 players have a chance to join in the beginning, but there should be no additions after the game begins. We have found that players get very upset when a person joins in later in the match. Don't let players join in at any arbitrary time. Limit it to a single screen and make it very clear how many players are playing.

• Ideally, you should have a tournament mode for 1-on-1 and 2-on-(1 or 2). XBAND will eventually provide special matching for people who only want to play 1-on-1.

# 7. Special XBAND Features for New Games

The following features can significantly add to the game play value of your games, utilizing the special capabilities of the XBAND systems. These are just a few ideas. If you have special needs or a really cool idea, let us know, and we'll see if we can code it into XBAND.

## 7.1. Full-screen for Point-of-View 2-Player Games

If you are currently splitting the screen in your 2-player point-of-view game, you might consider a full-screen mode when playing through XBAND.

## 7.2. Updated Stats Throughout the Season

If you are writing a sports game with real players, you can have the player roster updated with the latest trades, injuries, etc. whenever the player logs into the XBAND Server. Catapult is working on content relationships to provide you with this data for your sports games.

## 7.3. New Levels, Mazes

The XBAND Server can download new levels and mazes for RPG or DOOM-like games, within available RAM constraints.

## 7.4. Eight-player Games

XBAND can host 4-against-4 games for teams of players using multi-play adapter in each game machine.

## 7.5. Multi-Way Gaming

In the future XBAND will be able to support multi-way gaming for 10s or even 100s of simultaneous players, competing against each other in the same game.

Contact Catapult for more details if you have a game that would take advantage of this functionality.

### 7.6. Handicapping

Your game patch can provide information for the XBAND Network to use to handicap players. For example, you could start out a better player in a fighting game with a pre-existing "damage" level, or assign a higher point level to blows he/she receives.

### 7.7. Bug Fixes

Your Game Patch can include bug fixes to your game. Whether the bugs are crashers or just imperfections that should be fixed, it is trivial to use the Game Patching mechanism to fix the bugs when the cartridge is played on XBAND. Indeed, most of the games running on the XBAND network today have been patched with bug fixes.

### 7.8. Your Idea Here

If you've got a unique idea for information to store on the XBAND Server and integrate with your XBAND savvy game, we're interested in hearing it. Contact Developer Relations at (408) 366-1735.