Sega Genesis I/O Chip and Peripherals

By Charles MacDonald
http://cgfm2.emuviews.com

Contents

Overview

The I/O chip manages three 7-bit I/O ports. It also provides an way for the CPU to read the state of several jumpers in the system. Later versions of the Genesis hardware have the I/O chip integrated into some of the other custom hardware, but they all function identically.

Connector details

Ports A and B are male DB-9 connectors, while port C is female. In the Genesis 2 and 3, there is no physical connector for port C, but it can still be programmed and will respond like any other port. (as if no gamepad was connected) Here are the pin assignments:

```
Pin 1 : D0
Pin 2 : D1
Pin 3 : D2
Pin 4 : D3
Pin 5 : +5V
Pin 6 : TL
Pin 7 : TH
Pin 8 : Ground
Pin 9 : TR
```

CPU interface

The I/O chip is connected to the 68000 and Z80. When in Mark-III compatibility mode, the I/O chip has a different set of registers which mimic those of the SMS, which will not be discussed here.

The I/O chip is mapped to $A10000-$A1001F in the 68000/VDP/Z80 banked address space. It is normally read and written in byte units at odd addresses. The chip gets /LWR only, so writing to even addresses has no effect. Reading even addresses returns the same data you'd get from an odd address. If a word-sized write is done, the LSB is sent to the I/O chip and the MSB is ignored. Here are some examples to illustrate these points:

```
; Does nothing, byte writes to even addresses are ignored
move.b  #$40, $A10002

; Returns contents of version register, reading even addresses is the same as reading odd ones
move.b $A10000, d0

; Same as writing #$40 to $A10007, the MSB is ignored
```

```
move.w #$C040, $A10006
```

Register list

Address	Description
$A10001	Version
$A10003	Port A data
$A10005	Port B data
$A10007	Port C data
$A10009	Port A control
$A1000B	Port B control
$A1000D	Port C control
$A1000F	Port A TxData
$A10011	Port A RxData
$A10013	Port A serial control
$A10015	Port B TxData
$A10017	Port B RxData
$A10019	Port B serial control
$A1001B	Port C TxData
$A1001D	Port C RxData
$A1001F	Port C serial control

Version register

The version register returns several types of information, such as a hard-coded version number, settings of the domestic/export and PAL/NTSC jumpers, and the state of a sense pin which the Sega CD uses.

```
D7 : Console is 1= Export (USA, Europe, etc.) 0= Domestic (Japan)
D6 : Video type is 1= PAL, 0= NTSC
D5 : Sega CD unit is 1= not present, 0= connected.
D4 : Unused (always returns zero)
D3 : Bit 3 of version number
D2 : Bit 2 of version number
D1 : Bit 1 of version number
D0 : Bit 0 of version number
```

Bit 5 is used by the Sega CD, returning '0' when it is attached and '1' when it is not.

The version number is zero for the original model Genesis and MegaDrive. All later versions of the hardware are version 1, and have additional security hardware.

Bits 7,6 are used in country protection checks. Some early games used them to display English or Japanese text, so the same game program could be used in both regions. Here's a list of settings:

Bit 7	Bit 6	TV type	Region	Comments
0	0	NTSC	Japan, Korea, Taiwan	262 lines, 60 FPS
0	1	PAL	Japan	No hardware actually uses this setting
1	0	NTSC	USA, Canada	262 lines, 60 FPS
1	0	PAL-M	Brazil	262 lines, 60 FPS
1	1	PAL	Europe, Hong Kong	313 lines, 50 FPS

Early games stored a region compatibility code in their header at offset $0001F1. This value could be the ASCII text J, U, or E for Japan, USA or Europe. During game initialization, bits 7,6 could be sampled and compared to the region type stored in the header. If there is a mismatch, many games will fill the screen with a single color lock up intentionally, or sometimes display an error message.

Later games have a slightly more complex code. Here's the table Sega uses to determine valid codes to use:

$A10001			Main Sales Territories	Hardware Enable Code (numbers from 0 to F below)														
Bit 7	Bit 6	Hardware Type		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

Common uses of the new code type are ASCII values '4' for N.America, 'A' for UK, and 'B' for UK and Japan.

Data register

Writing to the data register sets the logic level of pins configured as outputs (0= 0V, 1= +5V), and does nothing to pins configured as inputs. Reading from the data register returns the state of each pin. Here's a list of which bits in the data register correspond to pins on the I/O port.

```
D7 : Unused. This bit will return any value written to it
D6 : TH pin
D5 : TR pin
D4 : TL pin
D3 : D3 pin
D2 : D2 pin
D1 : D1 pin
D0 : D0 pin
```

If a pin is configured as an input, reading it returns the true logic state of whatever the pin is connected to (e.g. 0 for ground, 1 for +5V). If nothing is connected to it, the pin returns '1', maybe due to internal pull-up resistors.

If a pin is configured as an output, reading it returns the last value written to this register.

Control register

The control register determines which pins are inputs and outputs.

```
D7 : /HL output control (see description)
D6 : TH pin is 1=output, 0=input
D5 : TR pin is 1=output, 0=input
D4 : TL pin is 1=output, 0=input
D3 : D3 pin is 1=output, 0=input
D2 : D2 pin is 1=output, 0=input
D1 : D1 pin is 1=output, 0=input
D0 : D0 pin is 1=output, 0=input
```

If bit 7 of this register is set, and TH is configured as an input, then whenever external hardware makes high to low transition on TH the I/O chip will strobe it's /HL output pin low. This pin connects to the VDP, which can be set up to latch the HV counter and/or cause a level 2 interrupt upon /HL going low.

Serial control register

The serial control register defines how a port is used for serial communications and provides status flags to indicate the current state of sending or receiving data.

```
D7 : Serial baud rate bit 1
D6 : Serial baud rate bit 0
D5 : TR pin functions as 1= serial input pin, 0= normal
D4 : TL pin functions as 1= serial output pin, 0= normal
D3 : 1= Make I/O chip strobe /HL low when a byte has been received, 0= Do nothing
D2 : 1= Error receiving current byte, 0= No error
D1 : 1= Rxd buffer is ready to read, 0= Rxd buffer isn't ready
D0 : 1= Txd buffer is full, 0= Can write to Txd buffer
```

The available baud rates are:

D7	D6	Baud rate
0	0	4800 bps
0	1	2400 bps

1	0	1200 bps
1	1	300 bps

When doing serial communication, TL and/or TR can be used for sending and receiving data. During this time the data and control registers have no effect for whichever pin(s) are in serial mode. The rest of the pins in the same port function normally.

The intended use of bit 3 is to also have the VDP set up to enable level 2 interrupts when /HL goes low. This way, when the I/O chip receives a byte through the TR pin, if this bit is set then it will strobe /HL low and cause an interrupt. So receiving data serially can either be accomplished through manual polling or be interrupt driven.

Bits 2-0 are read-only status flags, the rest of the bits in this register can be read and written.

TxData register

Writing a byte to this register will make the I/O chip output the data serially through the TL pin, providing it was configured for serial mode. You should poll bit 0 of the serial control register until the bit returns 0 to ensure the previously written byte has been sent and the TxData buffer is empty.

RxData register

Reading from this register returns the last byte received serially through the TR pin. You should check bits 3,2 of the serial control register to ensure that the RxData input buffer is full and that there were no errors in receiving the byte (which would then be invalid).

Using serial communication

When in serial mode, the I/O ports output TTL-level signals. You need something like a MAX232 line driver to convert these to RS-232 compatible signals for interfacing with (for example) a PC.

If you just want to do experiments on the serial ports themselves, you can use a null modem cable with the TL/TR wires switched around to connect port A to B. I used this for testing the serial capabilities while writing this document. Be sure to disconnect the +5V line as well.

Peripheral devices

2-button Mark-III gamepad

This controller has a 4-way direction pad, and two buttons (2 and 1). Here's a description of how the controller interfaces with an I/O port:

```
D6 : (TH) Not used
D5 : (TR) Button 2
D4 : (TL) Button 1
D3 : (D3) D-pad Right
D2 : (D2) D-pad Left
D1 : (D1) D-pad Down
D0 : (D0) D-pad Up
```

All buttons are active-low logic, so they return '0' when pressed and '1' when released.

3-button standard gamepad

This controller has a 4-way direction pad, and four buttons (Start, A, B, C). It uses a multiplexer that selects 1 of 2 inputs using TH, and routes buttons Start or C to TR, and B or A to TL.

Here's a description of how the controller interfaces with an I/O port:

```
             When TH=0          When TH=1
    D6 : (TH)   0                  1
    D5 : (TR)   Start button       Button C
    D4 : (TL)   Button A           Button B
    D3 : (D3)   0                  D-pad Right
    D2 : (D2)   0                  D-pad Left
    D1 : (D1)   D-pad Down         D-pad Down
    D0 : (D0)   D-pad Up           D-pad Up
```

All buttons are active-low logic, so they return '0' when pressed and '1' when released.

You need a small delay between changing TH and reading the button state back, as the multiplexer takes some time in switching inputs. I've found about four NOPs provides a reasonable delay.

Here's some example code to read a 3-button gamepad:

```
; Returns D0 with the button states: 'SACBRLDU'
_read_joypad:
                        move.l  d1, -(a7)
                        move.b  #$40, $A10003
                        nop
                        nop
                        move.b  $A10003, d0
                        andi.b  #$3F, d0
                        move.b  #$00, $A10003
                        nop
                        nop
                        move.b  $A10003, d1
                        lsl.b   #2, d1
                        andi.b  #$C0, d1
                        or.b    d1, d0
                        eori.b  #$FF, d0
                        move.l  (a7)+, d1
                        rts
```

Fighting Pad 6B

Information coming soon.

Lightguns (Sega Menacer, Konami Justifier)

Information coming soon.

EA 4-Way Play

The EA 4-Way Play plugs into ports A and B, and allows up to four standard Genesis controllers to be connected at once.

I've determined how the multitap works by examining the joystick reading code from NHL '95 and trying the same routine on a new-style Sega Team Player in 'EXTRA' mode. That said, the actual EA 4-Way Play could have some differences that Sega's compatibility mode doesn't take care of.

To detect the multitap, set CTRL1 to 0x40, CTRL2 to 0x7F and DATA2 to $7C. Reading the lower 2 bits of DATA1 will return zero if the multitap is present. Usually these bits return 0x03 if there is no tap, no controller, or the Sega Team Player isn't in EXTRA mode, but this could be a behavior specific to the Team Player.

To read the pads, write 0x0C, 0x1C, 0x2C, or 0x3C to DATA2 to select the controller in port A, B, C, or D. You can then use DATA1/CTRL1 to read the pad state just like polling a regular 3-button pad in port A.

Here's some example code to use the EA 4-Way Play:

```
; Returns D0 == 0 if the multitap is present, or D0 != 0 if it isn't
_detect_4wayplay:
                        move.b  #$40, $A10009
                        nop
                        move.b  #$7F, $A1000B
                        nop
                        move.b  #$7C, $A10005
                        nop
                        moveq   #0, d0
                        move.b  $A10003, d0
                        andi.b  #$03, d0
                        rts

; Returns the state of all four 3-button gamepads in D0
_read_4wayplay:
```

```
                    move.l  d1-d3/a0, -(a7)
                    lea     $A10000, a0
                    move.l  #$0C1C2C3C, d2
                    moveq   #$03, d3
      poll:         move.b  d2, $05(a0)
                    nop
                    nop
                    move.b  #$40, $03(a0)
                    nop
                    nop
                    move.b  $A10003, d0
                    andi.b  #$3F, d0
                    move.b  #$00, $03(a0)
                    nop
                    nop
                    move.b  $03(a0), d1
                    lsl.b   #2, d1
                    andi.b  #$C0, d1
                    or.b    d1, d0
                    lsl.l   #8, d0
                    lsr.l   #8, d2
                    dbra    d3, poll
                    eori.l  #-1, d0
                    move.l  (a7)+, d2-d3/a0
                    rts
```

Sega Team Player

The Team Player is a multitap device that allows four peripherals to be connected to the Genesis at one time. It has a mode select switch with several settings:

```
EXTRA - All four ports are enabled. This is the setting used by
        EA 4-Way Play compatible games.
    A - Routes the peripheral in port A to port A of the Genesis.
    B - Routes the peripheral in port B to port A of the Genesis.
    C - Routes the peripheral in port C to port A of the Genesis.
    D - Routes the peripheral in port D to port A of the Genesis.
MULTI - All four ports are enabled. This is the setting used by
        Team Player compatible games.
```

Sega made two versions of the Team Player. The first one was not compatible with the EA 4-Way Play and only had one cable to connect it to port A. The second one added the "EXTRA" setting for EA 4-Way Play compatibility and provides a second cable to connect it to port B as well. Both seem to function identically with the exception of EXTRA mode.

I don't have any programming information for this device.

Sega Mega Mouse

This is a three button mouse (left, middle, right) with an extra button (Start) located near the thumb position. It uses a PIC16C54 microcontroller which manages the buttons and position tracking.

The Sega Mega Mouse that is distributed in North America is incompatible with the European version of Populous 2. The mouse functions normally but has Y axis inverted so up is down and vice-versa. It may have been designed for the Japanese Mega Drive mouse, which is a **different** product with 2 buttons and unique physical appearance.

The protocol works as follows: TH and TR are outputs which tell the microcontroller to stop or start a data transfer, and to acknowledge received data. TL is an input which returns a busy flag for the microcontroller. D3-D0 are inputs that return the data. Here's a table showing the communication process:

Write	TH	TR	TL	D3	D2	D1	D0	Description
$60	1	1	1	0	0	0	0	Request data
$20	0	1	1	0	0	0	0	ID #0 ($0)
$00	0	0	1	1	0	1	1	ID #1 ($B)
$20	0	1	0	1	1	1	1	ID #2 ($F)
$00	0	0	1	1	1	1	1	ID #3 ($F)
$20	0	1	0	Y Over	X Over	Y Sign	X Sign	Axis sign and overflow
$00	0	0	1	Start	Middle	Right	Left	Button state
$20	0	1	0	X7	X6	X5	X4	X axis MSN

$00	0	0	1	X3	X2	X1	X0	X axis LSN
$20	0	1	0	Y7	Y6	Y5	Y4	Y axis MSN
$00	0	0	1	Y3	Y2	Y1	Y0	Y axis LSN

Write #$60 when you are done polling to stop the data transfer. If you continue to poll the mouse after collecting all the data by writing $20 / $00, the Y axis LSN will always be returned. Sega advises polling beyond this point will make the mouse behave abnormally. It's possible that the PIC code in some versions of the Mega Mouse may have problems if the poll sequence is too long.

The X/Y overflow flags are supposed to be set if the mouse is moved over a distance greater than can be measured. I can't get them to become set, maybe the mouse supports a fairly wide range of movement.

You need a considerable delay between writing new values to TH/TR. I think the intention is to poll TL until the microcontroller returns 'not busy', but I can't get this to work reliably. Sega's mouse reading code also has a timeout when checking TL which would indicate it may get stuck in a certain state.

All buttons (start, left, middle, right) are active-high logic, so they return '1' when pressed and '0' when released.

Miscellaneous

After power-up, the default state of the I/O chip are as follows:

```
$A10001 = $80 (Bits 7,6,5 depend on the domestic/export, PAL/NTSC jumpers and having a Sega CD or not)
$A10003 = $7F
$A10005 = $7F
$A10007 = $7F
$A10009 = $00
$A1000B = $00
$A1000D = $00
$A1000F = $FF
$A10011 = $00
$A10013 = $00
$A10015 = $FF
$A10017 = $00
$A10019 = $00
$A1001B = $FB
$A1001D = $00
$A1001F = $00
```

Disclaimer
