Reverse-Engineering Robby Roto: A 1980s Embedded System Masquerading as an Arcade Game

Stephen A. Edwards
Columbia University

January, 2005

Abstract

Robby Roto was an arcade game produced by Bally/Midway in 1981. Although not especially popular at the time, it does have the distinction of being one of the few commercial arcade games whose code is now in the public domain (the rights reverted to the author Jay [now Jamie] Fenton[1], who released it in 1999). Although primitive by today's standards, it is representative of many early arcade games and illustrates a realistic, commercial embedded system.

The description presented here has been synthesized from many sources, including disassembled ROM images; source code from MAME[2], the multiple arcade machine emulator by Nicola Salmoria and many others; service manuals; and documentation on the Bally Astrocade home video game system[3], which contains many of the same custom chips.

Like many arcade games, Robby was one of a family built with similar hardware. Specifically, it shares its general design with Seawolf II (1978), Space Zap (1980), Extra Bases (1980), Wizard of Wor (1980), Gorf (1981), Professor Pac-Man (1983), and most interestingly, the Bally Astrocade home video game system (1978). All use the same microprocessor, a Zilog Z80, and the same custom graphics and sound chips, which were designed in part by Dave Nutting. For more history of these games and many others, see Herman [2] and Kent [3].

[1] www.fentonia.com
[2] www.mame.net
[3] See www.ballyalley.com.

1 System Architecture

Like most videogame systems[4], Robby is a bus-based microprocessor system with support for video, sound, and some simple input devices. Built around a Z80 running at 1.7897725 MHz (derived from a 14.31818 MHz crystal, which is $4\times$ the NTSC colorburst frequency, an artifact of its legacy as a home videogame. Robby drives an NTSC-speed RGB monitor directly; composite video is never generated), it contains the usual RAMs (both static and dynamic), ROMs, and memory-mapped I/O devices, including a video controller with bit-mapped graphics and a pair of sound synthesizers.

Figure 1 shows the overall system block diagram, which is built around two busses. The Z80 communicates through a sixteen-bit address bus and separate bidirectional eight-bit data bus, which are connected directly to the ROM and SRAM. A simple bus bridge multiplexes data and address onto a single bus, which connects to the custom address, data, and sound I/O chips. This was done to reduce pin count on these custom chips.

Physically, the system is built on backplane containing six boards: the CPU and custom address and data chips, a memory board holding the ROMs (10 4K $\times$ 8) and SRAMs (four 2K $\times$ 8 8416s), a "game" board holding the two sound I/O chips and other digital I/O, a "pattern board" containing a blit controller, and two boards containing the DRAM chips (16 4K $\times$ 1 TMS4027s per board). One of the SRAM chips is powered by a NiCad

[4] Early games were built completely from discrete components and did not incorporate a microprocessor. Sega's Monaco GP (1980) was one of the last.

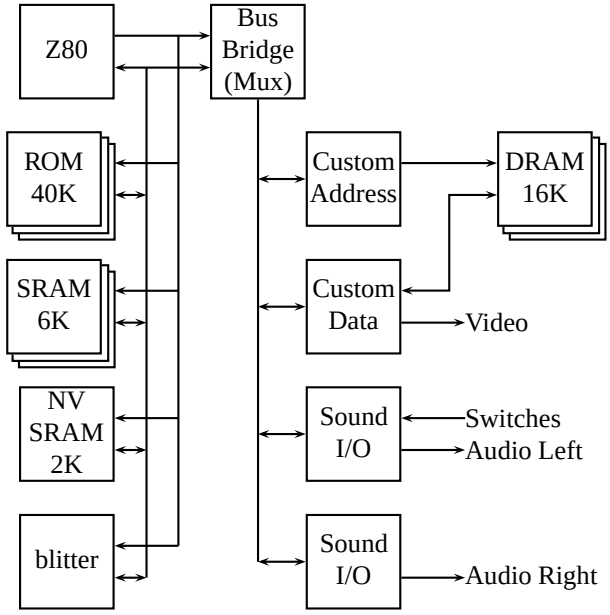

Figure 1: System Block Diagram

rechargable battery so its contents (mainly high scores) are retained when the game is powered off.

The custom address and data chips are responsible for the video display. 16K of DRAM is used to hold a frame buffer. As is typical in all processor-driven video controllers, both the processor and video circuitry need access to video memory. The address and data chips arbitrate such accesses, giving priority to the video circuity, which never makes two back-to-back accesses, so it is only ever necessary to stall the Z80 for a single memory cycle.

2 The Video System

Somewhat unusual for video games of the time because of its large memory requirements, Robby (and those in its family) used a frame buffer. More typical games of the time, such as Pac-Man, combined a character-based background with sprite foregrounds. See Collins's survey [1] of similar systems.

The hardware uses two bits per pixel (i.e., four pixels per byte), each of which selects four colors from a palette of 256. It supports two resolutions: 160×102, for home machines (which reduces the framebuffer memory requirements to a modest 4K), and 320×204, used by Robby. Oddly, pixel colors are set by eight separate eight-bit color registers: four for pixels on the left half of the screen, the other four for the right. The choice of column dividing left and right is under software control. Finally, the color in the overscan area (i.e., outside the 320×204 raster but before any blanking) is controlled by another two bits.

Video memory appears twice in the Z80's memory space: once as normal memory that may be read and written as usual, and once as write-only "magic RAM" that invokes simple pixel processing operations according to a global mode register. Writing a byte in this area of memory can automatically be expanded from one to two bits-per-pixel, shifted between zero and three pixels to the right, mirrored left-to-right, rotated to modify four pixels vertically instead of horizontally, and made to perform logical OR or XOR operations instead of a simple overwrite. Expansion is done first, then either rotation or shifting (they are mutually exclusive), flopping, and finally OR or XOR (also mutually exclusive). The system also includes a collision detection mechanism that tracks whether any non-zero pixel is overwritten during these operations. This "magic RAM" mechanism offloads many awkward bit manipulation operations from the processor to increase performance, an effective substitute for the sprite graphics used by other systems of the time.

The video display is the only source of interrupts in the system. It can generate two types: a light pen interrupt that goes unused in the Robby game (another artifact of its home arcade system origins), and a scan-line interrupt that can be triggered at any scan line under program control.

Robby, as well as other commercial games in its family, includes a "pattern board" that implements a blit controller in discrete logic. Controlled through seven write ports, it performs fast memory-to-memory copies suited for the framebuffer (i.e., it can copy a contiguous block of memory to a rectangular region in the framebuffer).

3 The Sound System

Robby contains two identical custom sound chips, one per channel for stereo sound, that also provide support for scanning a key matrix. Robby only uses one chip to de-

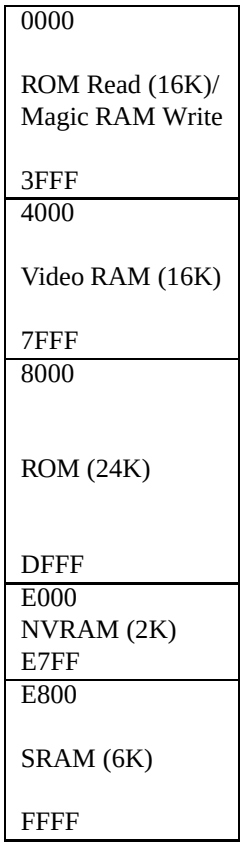

Figure 2: Memory map

code nineteen one-bit inputs (joysticks, coin sensors, etc.) and eight dipswitches.

The synthesizer in each sound chip contains a collection of numerical oscillators capable of producing three independent tones plus a noise source. A "master oscillator" square wave is generated from either a programmable noise source or a programmble low-frequency oscillator that produces a vibrato effect. This signal is then fed to three tone generators (programmable dividers) with volume controls that drive drive three four-bit DACs. The output of these three DACs are summed along with the output of the noise generator to produce the final analog audio signal that is passed through an amplifier to a speaker.

00	COL0R	Color 0 right
00	COL0R	Color 0 right
01	COL1R	Color 1 right
02	COL2R	Color 2 right
03	COL3R	Color 3 right
04	COL0L	Color 0 left
05	COL1L	Color 1 left
06	COL2L	Color 2 left
07	COL3L	Color 3 left

7–3	Hue
2–0	Intensity (luminance) value

08 CONCM Resolution

7–1	Unused
0	1 = 320×204, 0 = 160×102

09 HORCB Left/right boundary

7–6	Background (overscan) color
5–0	First byte (low-res) or word (high-res) that uses right palette colors. 20 decimal is center of the screen.

0A VERBL Vertical blank

7–0	Line number at which vertical blanking should begin. 203 or less for high-res.

0B COLBX Color block output port

7–0	Write-through to one of the eight color registers in sequence. Intended for the Z80 OTIR instruction, writes go to registers 7–6–5–4–3–2–1–0–7–6· · ·

0C MAGIC Magic control

7	Unused
6	Flop: Mirror the pixels by interpreting the the most-significant bits of the byte as being the rightmost pixel instead of the leftmost.
5	XOR transfer: Perform the logical XOR of each pixel as it is written.
4	OR transfer: Perform the logical OR each each pixel as it is written.
3	Expand each written bit into a pixel. Colors are selected by write port 19.
2	Rotate: Write the given pixels vertically instead of horizontally.
1–0	Shift amount: Number of pixels to shift the written pixels to the right on screen.

Table 1: Video I/O write ports (1/2)

0D INFBK Interrupt vector

7–0	Least significant eight bits of interrupt vector. Together with the processor's Interrupt page register (I), this controls the address where control is sent when the video system generates an interrupt. The lowest four bits of this register are treated as being zero for a light pen interrupt, allowing it to invoke a different interrupt handler.

0E INMOD Interrupt mode

7–4	Unused
3	Scanline interrupt enable
2	Scanline interrupt mode: 0=interrupt until acknowledged, 1=interrupt will be dropped after one instruction.
1	Light pen interrupt enable
0	Light pen interrupt mode: same as for scanline interrupt mode

0F INLIN Interrupt line

7–0	Scanline at whose end to generate an interrupt. This number should be twice the desired scanline in low-resolution mode, i.e., so that the value for the bottom of the screen is the same in either mode.

19 XPAND Expand mode colors

3–2	Color to use for "1" bits when writing to "magic" RAM with expansion enabled.
1–0	Color to use for "0" bits when writing to "magic" RAM with expansion enabled.

Table 2: Video I/O write ports (2/2)

10 TONMO Master oscillator frequency

7–0	Master oscillator frequency: $1789/(x+1)$kHz

11 TONEA Tone A Frequency
12 TONEB Tone B Frequency
13 TONEC Tone C Frequency

7–0	Tone generator frequency: $F_m/2(x+1)$, where F_m is the master oscillator frequency.

14 VIBRA Vibrato

7–6	Vibrato speed: 00=fastest, 11=slowest
5–0	Vibrato depth

15 VOLC Tone C Volume

7–6	Unused
5	Noise generator enable
4	Module master oscillator with noise (0=modulate with vibrato)
3–0	Tone C volume

16 VOLAB Tone A/B Volume

7–4	Tone B volume
0–3	Tone A volume

17 VOLN Noise Volume

7–4	Noise volume/noise mask: These control both the volume of the noise generator output and the mask used to affect the modulation of the master oscillator.
3–0	Noise mask. These bits are used only to mask the modulation of the master oscillator.

18 SNDBX Sound block output port

7–0	Write-through to one of the eight sound registers in sequence. Intended to be used with the Z80 OTIR instruction, the first write goes to the noise volume register, the next to the tone a/b volume register and so forth down to the master oscillator frequency register and then the pattern repeats.

50 Second sound controller
⋮
58 repeats 10–18

Table 3: Sound I/O write ports

5B FIXME

78 SRCLO Source address LSB

7–0	Least significant byte of source address

79 SRCHI Source address MSB

7–0	Most significant byte of source address

7A MODE Copy mode

7–6	Unused
5	Flop
4	Flip
3	Flush
2	Constant
1	Expand mode
0	Direction

7B SKIPLO Bytes to skip/destination LSB

7–0	LSB of destination and number of bytes to skip after each row is copied.

7C DESTHI Destination MSB

7–0	Most significant byte of destination address

7D LENGTH Length of each row

7–0	Length (in bytes) of each row to copy

7E LOOPS Row count and trigger

7–0	Number of rows to copy minus one. Writing this initiates the transfer.

Table 4: I/O Write ports for the pattern board

08 INTST Intercept status

7	Intercept in pixel 0 during last OR or XOR write
6	Intercept in pixel 1 during last OR or XOR write
5	Intercept in pixel 2 during last OR or XOR write
4	Intercept in pixel 3 during last OR or XOR write
3	Intercept in pixel 0 in an OR or XOR since last reset
2	Intercept in pixel 1 in an OR or XOR since last reset
1	Intercept in pixel 2 in an OR or XOR since last reset
0	Intercept in pixel 3 in an OR or XOR since last reset

0E VERAF Vertical address feedback

7–0	Line number at which a light pen interrupt occurred. LSB is always zero in low-resolution mode.

0F HORAF Horizontal address feedback

7–0	Horizontal position of light pen interrupt. Pixel number is $x-8$ in low-res mode, $2(x-8)$ in high-res mode.

Table 5: I/O Read Ports for Video

10 SW0 Switch bank 0 (Active-low)

7	Unused
6	Start 2 player
5	Start 1 player
4	Tilt
3	Service
2	Coin 3 (unused)
1	Coin 2
0	Coin 1

11 SW1 Switch bank 1 (Active-low)

7–6	Unused
5	Magic (2nd player)
4	Unused
3	Right (2nd player)
2	Left (2nd player)
1	Down (2nd player)
0	Up (2nd player)

12 SW2 Switch bank 2

7–6	Unused
5	Magic
4	Unused
3	Right
2	Left
1	Down
0	Up

13 SW3 Dipswitches

7	Demo sounds
6	Unused
5	Unused
4	Unused
3	1=Upright, 0=Cocktail
2	Free play
1	Reset game options to factory settings
0	Full reset

15 LEDs and Coin Counters? (FIXME)

Table 6: I/O Read Ports for Buttons, etc.

References

[1] Steven Collins. Computer graphics during the 8-bit computer game era. Technical Report TCD-CS-1998-15, Trinity College, Dublin, Ireland, September 1998.

[2] Leonard Herman. *Phoenix: The Fall & Rise of Videogames*. Rolenta Press, Union, New Jersey, second edition, 1997.

[3] Steven L. Kent. *The Ultimate History of Video Games*. Prima Publishing, 2001.