Reconstructing Pong on an FPGA

Stephen A. Edwards
Department of Computer Science, Columbia University

CUCS–023–12, December 2012

Abstract

I describe in detail the circuitry of the original 1972 *Pong* video arcade game and how I reconstructed it on an FPGA—a modern-day programmable logic device. In the original circuit, I discover some sloppy timing and a previously unidentified bug that subtly affected gameplay. I emulate the quasi-synchronous behavior of the original circuit by running a synchronous "simulation" circuit with a 2× clock and replacing each flip-flop with a circuit that effectively simulates one. The result is an accurate reproduction that exhibits many idiosyncracies of the original.

Introduction

This work started with a desire to play *Pong*, Atari's 1972 video arcade game that effectively launched the industry [11]. While I could have sought out one of the few remaining machines, I chose instead to reconstruct it on an FPGA, much as I had done for the Apple II computer [7] and others have done for various other classic video arcade games [8].

Pong and other early games were implemented largely with discrete TTL chips, hence my choice of an FPGA. By contrast, most later games were processor-based and have been successfully emulated in software using an instruction-set simulator interacting with a high-level *ad hoc* simulator for the video hardware [9]. While modern processors are vastly faster than the roughly 7 MHz clock frequency of *Pong* and many have written *Pong*-like programs in software, my goal was precise (cycle-accurate) emulation. I ruled out doing so in software because I expect it would be difficult to implement a software circuit simulator able to consistently run this fast.

However, while the *Pong* circuit is ostensibly synchronous, it is actually littered with ripple counters, RS latches built from discrete NAND gates, and flip-flops clocked from combinational logic, all of which are anathema to robust FPGA designs.

Below, I describe the circuit of the original *Pong* with a focus on its timing and analog components (§ 1), then describe my technique for reconstructing it on a modern-day, fully synchronous FPGA (§ 2).

1 Pong Circuit Description

In addition to reading the schematics for *Pong* that can be found online (they appear to have been scanned from service manuals), Dan Boris [4] presents an extensive description of the circuit; much of what I write here is derived from his work, especially his division of the circuit into sections. Arkush [15] also describes part of the circuitry in *Pong*, focusing closely on how counters are used to control the position of the ball.

1.1 The Main Clock

Figure 1 shows the main clock generator: a 14.318 MHz crystal oscillator[1] driving a NAND gate driving a JK flip-flip that halves the frequency and generates a 50% duty-cycle square wave: the 7.159 MHz master clock. 14.318 MHz is a common crystal frequency in video circuits because it is four times the NTSC colorburst frequency of 315/88 MHz = 3.579545 MHz. *Pong*, however, is black-and-white so another frequency could have been used.

1.2 The Horizontal Counter

The horizontal counter (Figure 2), built from two 7493s (F8 and F9) and a 74107 (F6), keeps track of the horizontal position of the video beam. The 7493 is a four-bit ripple counter built from four negative-edge-triggered T flip-flops.

[1] The frequency is not labeled on the schematic, but multiple sources, including schematics of Pong clones, confirm this value.

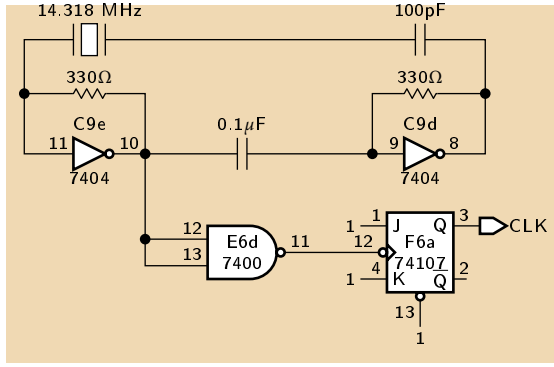


Figure 1: The main clock oscillator. This generates a 7.159 MHz square wave.

Abstractly, the eight-input NAND F7 detects the count 256 + 128 + 64 + 4 + 2 = 454 and causes the counter to reset, but the behavior is slightly more subtle, as shown in Figure 4. Because the ripple counters are triggered on the negative edge of the clock but the output of F7 is buffered by the positive-edge-triggered D flip-flip E7b, the count 454 is only seen for half a clock period while the count 0 is seen for one-and-a-half clock periods because the HRESET signal only rises after the next rising edge of the clock, effectively surpressing the count on the next falling edge of the clock.

The period of HRESET is thus 7.159 MHz/455 = 15.734 kHz, exactly the NTSC horizontal frequency.

The horizontal counter is one of the most active parts of the circuit, yet Alcorn used slower, more problematic ripple counters instead of synchronous counters. Why he did this is not clear; one hypothesis is that it was a cost-saving measure: 7493s were nearly half the price of 74161s in 1975 [10].

The use of ripple counters in the horizontal timing circuitry of these games appears to be characteristic of Alcorn. Bushnell's earlier, more complex, and far less successful *Computer Space* (Nutting Associates, 1973) did use 74161s [12]. Alcorn designed [6] Atari's 1973 successor to *Pong*, *Space Race* [11], and again used 7493s [3]. Atari's 1974 *Pin Pong*, not designed by Alcorn, used synchronous counters (9316s) [2].

The presence of ripple counters makes the timing of this circuit worth discussing. In Figure 2, I drew some of the internal structure of the counters to help explain the behavior of this circuit. The outputs 1H...256H do not change simultaneously: they ripple, which can be problematic for decoding particular columns. To decode 454 as needed, fortunately, the delay is effectively modest because it is triggered by 2H going high, which occurs only two flip-flop delays after the falling edge of the 7.159 MHz clock—all the other signals went high in a previous cycle and stay stable just before reaching a count of 454.

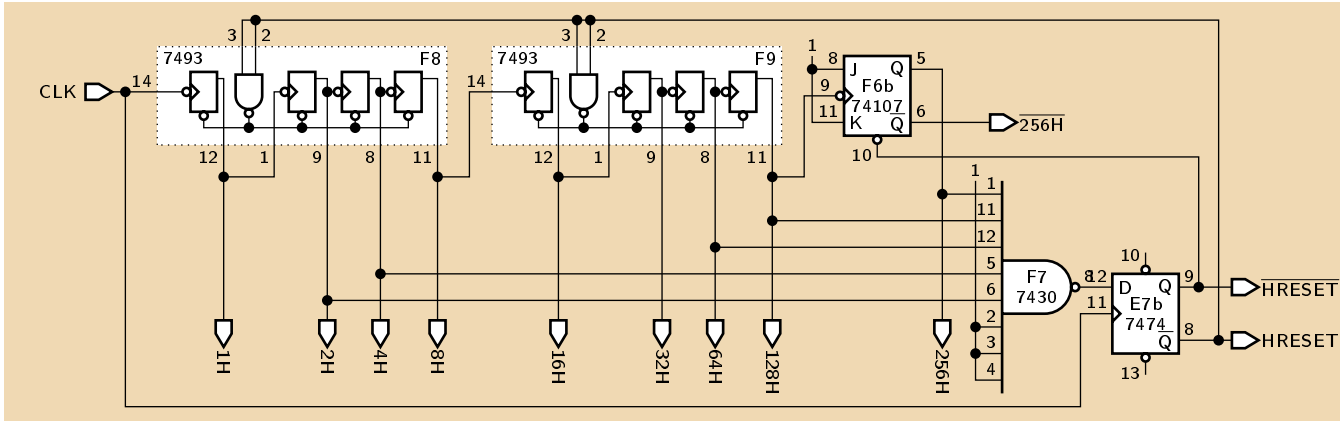


Figure 2: The horizontal counter: This counts 0, 1, …, 454, generating a 15.734 kHz horizontal frequency.

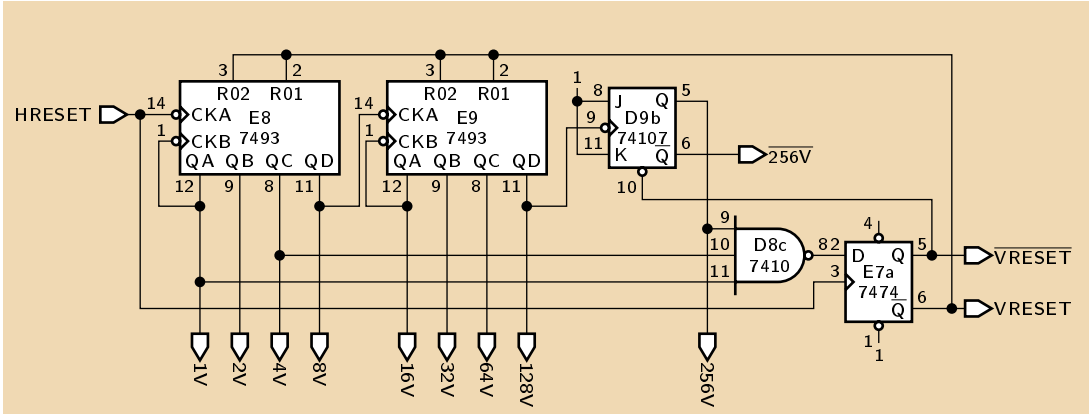


Figure 3: The vertical counter: This counts 0, 1, …, 261

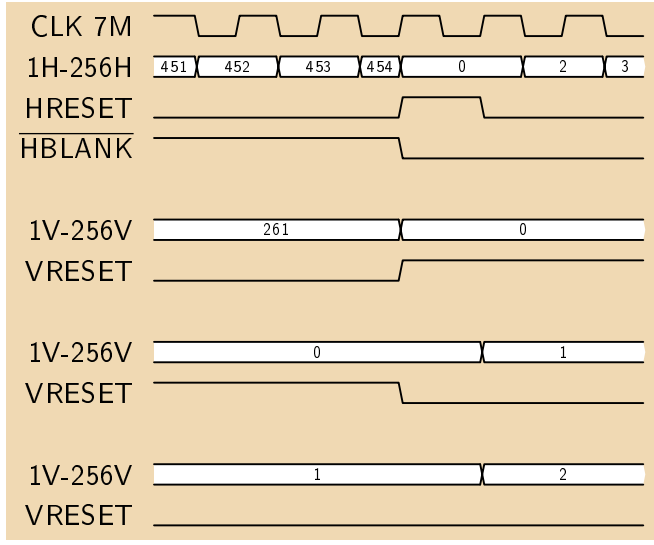


Figure 4: Behavior of the horizontal and vertical counters at the end of a horizontal line

1.3 *The Vertical Counter*

The vertical counter (Figure 3) is similar to the horizontal counter, but is clocked once per field by the HRESET signal generated by the horizontal counter and resets on a count of 1 + 4 + 256 = 261. This gives a vertical refresh frequency of 15.734 kHz/262 = 60.05 Hz, which is certainly close enough to 60 Hz for most monitors. A strictly compliant (interlaced) NTSC signal actually has 525/2 = 262.5 horizontal line periods per field.

Again the 7493s are negative-edge triggered, so the vertical count changes when HRESET falls and is reset when HRESET rises—see Figure 4.

1.4 *Horizontal and Vertical Sync*

Figure 5 shows the circuits for generating horizontal and vertical blanking and synchronization signals. These are two RS latches built from discrete gates with some extra gating logic.

Part of the timing of the horizontal blanking latch is problematic. The $\overline{\text{HRESET}}$ signal is not a problem because it comes directly from a flip-flop triggered by the clock (E7b, see Figure 2, so $\overline{\text{HBLANK}}$ falls quickly after the rising edge of the clock. However, for $\overline{\text{HBLANK}}$ to rise, 64H must be high and NAND G5b looks for the rising edge of 16H (a count of 64 + 16 = 80). This is initiated by a falling edge on the 7 MHz clock and occurs after passing through five flip-flop stages (all of the F8 ripple counter and the first stage of F9). Below is an accounting of delays based on numbers from TI's 1988 data book [14].

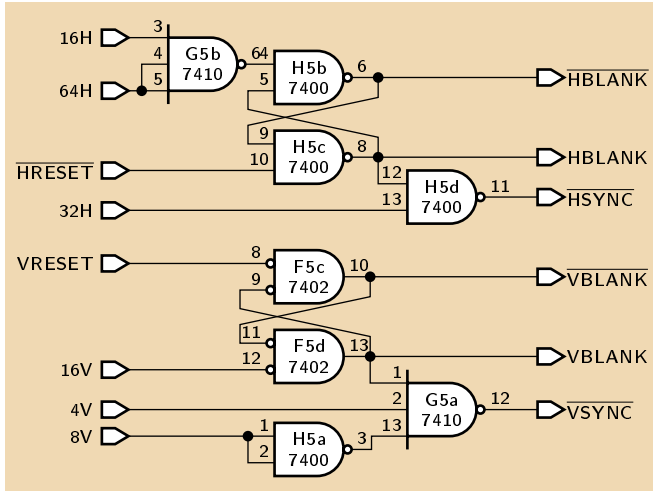


Figure 5: Horizontal and Vertical Blanking and Sync

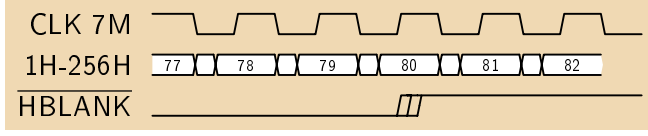


Figure 6: Behavior of $\overline{\text{HBLANK}}$ near the start of the line. Because of the ripple counters in the horizontal counter, $\overline{\text{HBLANK}}$ rises after the next rising edge of the clock.

Path	Typ.	Max.
CLK falling to 8H falling	46 ns	70
8H falling to 16H rising	10	16
16H rising to G5b-6 falling	7	15
G5b-6 falling to $\overline{\text{HBLANK}}$ rising	11	22
CLK falling to $\overline{\text{HBLANK}}$ rising	74	123

However, the 7 MHz clock has a period of about 140 ns, meaning $\overline{\text{HBLANK}}$ rises *after* the next rising edge of the clock. Figure 6 illustrates this.

$\overline{\text{HSYNC}}$ only goes low when $\overline{\text{HBLANK}}$ is low (horizontal counts 0–79) and 32H is high, i.e., from horizontal counts 32 to 63 inclusive.

Using similar reasoning, $\overline{\text{VBLANK}}$ goes low when VRESET goes high: during line 0, and high again at the start of line 16. $\overline{\text{VSYNC}}$ is only low when $\overline{\text{VBLANK}}$ is low, 4V is high, and 8V is low: lines 4 through 7 inclusive.

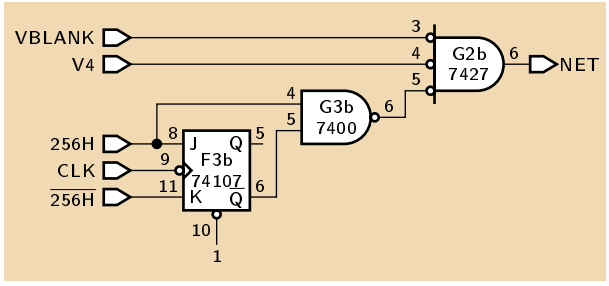


Figure 7: Circuit that generates the net

1.5 *The Net*

The net is a simple graphic object—a dashed line going down the center of the screen—that does not affect gameplay. Figure 7 shows the circuit, which displays a vertical line in column 256 that alternates between four pixels on and four pixels off. Before the horizontal counter reaches 256, 256H is low and $\overline{256H}$ is high, so F3b's $\overline{Q}$ is high. When the horizontal counter reaches 256, 256H goes high, sending the output of G3b low, enabling G2b to pass V4 as the net signal when out of vertical blanking. In the next cycle, since 256H was high and $\overline{256H}$ was low, F3b's $\overline{Q}$ falls, setting NET low.

1.6 *The Paddles*

Figure 8 shows the circuit generating the signals that indicate when (and hence where) each player's paddle is displayed. There are two identical circuits, one for each player. Each 555 is wired as a one-shot triggered near the bottom of the screen; the paddle sets the delay time, controlling on which line the one-shot expires.

Player 1's paddle is a 5K potentiometer[2] that forms a voltage divider feeding the control voltage input of 555 timer B9. Internally, this pin is connected to a ladder of three 5K that sets its nominal voltage to 2/3 V_{cc}; the paddle input shifts the control voltage away from this value. The 0.1μF capacitor on CTRL input bypasses the reference voltage to help keep it constant.

At rest, OUT is low and the diode clamps the 0.1μF capacitor to ground through the DIS pin. Pulling $\overline{TRG}$ low turns on OUT and floats DIS, allowing the 56K fixed resistor and 50K trim pot to charge the 0.1μF capacitor. When the voltage on THR reaches that on CTRL, OUT goes low and the capacitor is again discharged through DIS for another cycle. Figure 8 shows a rough diagram of these waveforms.

While the exact period of the one-shot is a complex, nonlinear function, it is $\approx RC$ = 8.1ms for R = 81K and C = 0.1μF. The vertical refresh period is approximately 16ms, so with a suitable setting of the 50K trim pot, and adjusting the paddle voltage divided, the one-shot period should vary from the top to the bottom of the screen.

[2] This value is not marked on the original schematics; manuals and schematics of clones [1] confirm this value.

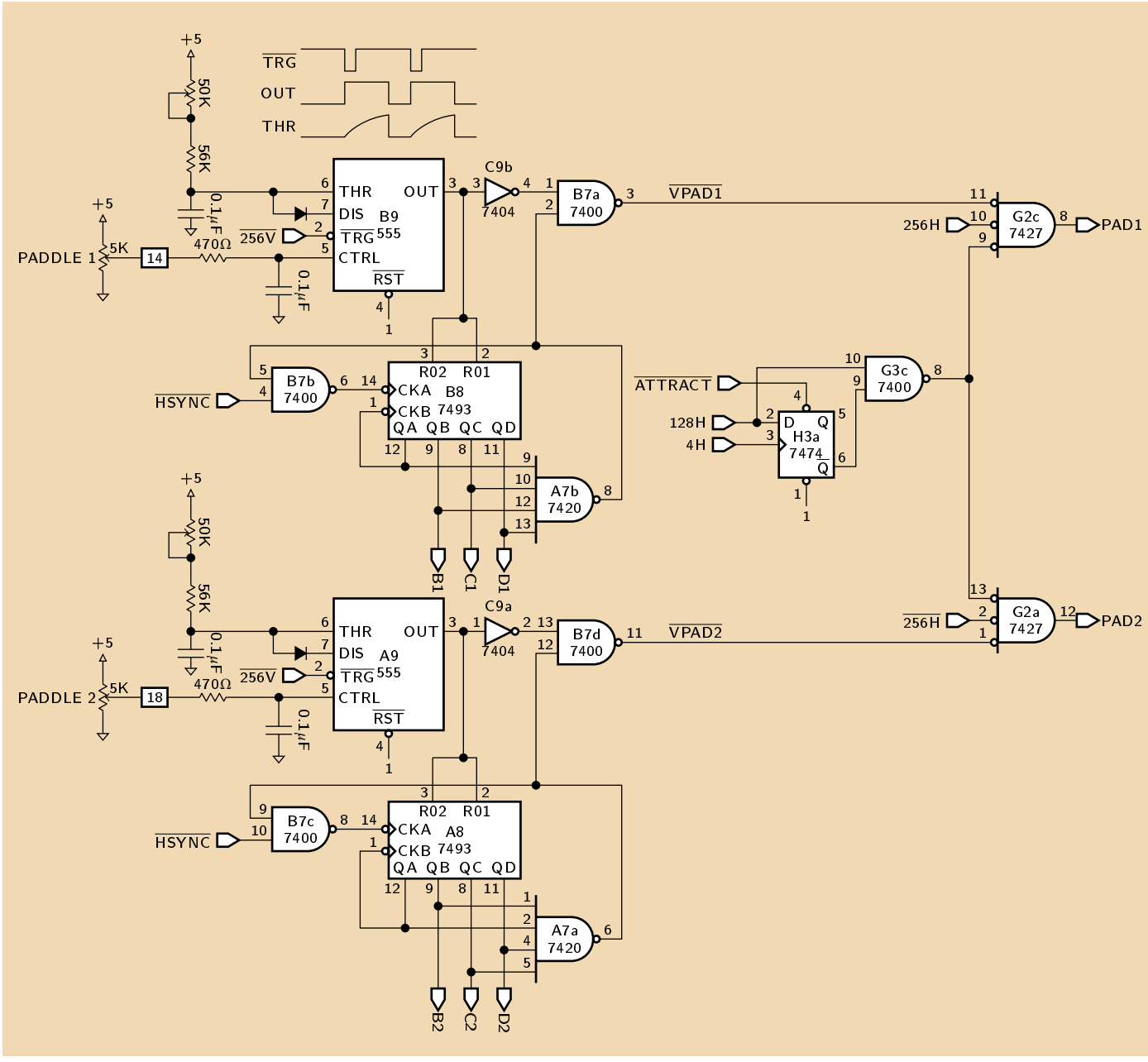


Figure 8: The paddle circuit. 555 timers A9 and B9 operate as one-shots triggered by $\overline{256V}$ that select the top line of each player's paddle. Counters A8 and B8 and flip-flop H3a make each paddle 15 lines high and 4 pixels wide.

However, Alcorn explains,

> The other—not hack—but, one of my lessons learned, is that if you can't fix it, call it a feature. The paddles on the original Pong didn't go all the way to the top. There was a defect in the [circuit]—I used a very simple circuit, I had to, to make the paddles, but they didn't go to the top. I could have fixed it, but it turned out to be important, because if you get two good players they could just volley and play the game forever. And the game has to end in about three or four minutes otherwise it's a failure as a game. So that gap at the top, again—a feature. So that was sort of a happy accident. [13]

Going higher on the screen corresponds to a shorter timeout, which in this circuit is limited in part by how low the voltage can go on CTRL, which is affected by the internal 5K resistor ladder.

While the 555's OUT is high, the paddle is not displayed because the the output of B7a stays high, PAD1 stays low, and the four-bit ripple counter B8 is reset.

The paddle starts to be displayed on the line where OUT first goes low and continues to be displayed for the next fourteen lines. Here, both the output of inverter C9b and the output of A7b are high, so $\overline{\text{VPAD1}}$ is low. The output of A7b stays high until the counter reaches fifteen. B7b clocks it on the rising edge of $\overline{\text{HSYNC}}$. On the count of fifteen, the output of A7b falls sending B8's clock A high. This is a glitch: the output of B7b falls, causing QA to rise, causing the output of A7b to fall, which finally causes the output of B7b to rise again. However, the 7493 only requires a 15 ns pulse on CLKA for proper operation and the delay from CLKA to QA is typically 10 ns and the delay through a 7420 is typically 8 ns, so it should work.

The three higher-order output bits of each paddle's vertical counter are used to determine the angle at which the ball bounces off the paddle; see § 1.9.

Together, H3a and G3c determine the horizontal width and position of the paddle: the output of G3c goes low between the rising edge of 128H (horizontal counts 128 and 384) and the next rising edge of 4H (horizontal counts 132 and 388) to give a four-pixel-wide paddle. This occurs twice per line, once when 256H is low, displaying the left paddle through G2c, and once when 256H is high, displaying the right through G2a.

1.7 The Score

Pong displays two two-digit scores in "seven-segment" style on the upper part of the screen. Each player's score is held in a decade ripple counter for the first digit augmented with a JK flip-flop to represent the tens digit. A switch controls whether the game ends when either player reaches a score of 11 or 15.

At the core of the circuit is a 7448 BCD-to-seven-segment decoder, which was originally designed to drive seven-segment LED displays. In *Pong*, a pair of four-to-one multiplexers steer one of the four score digits to the 7448, which feeds the decoded number into a bank of seven NAND gates that are each activated during a "segment" region on the screen. Finally,

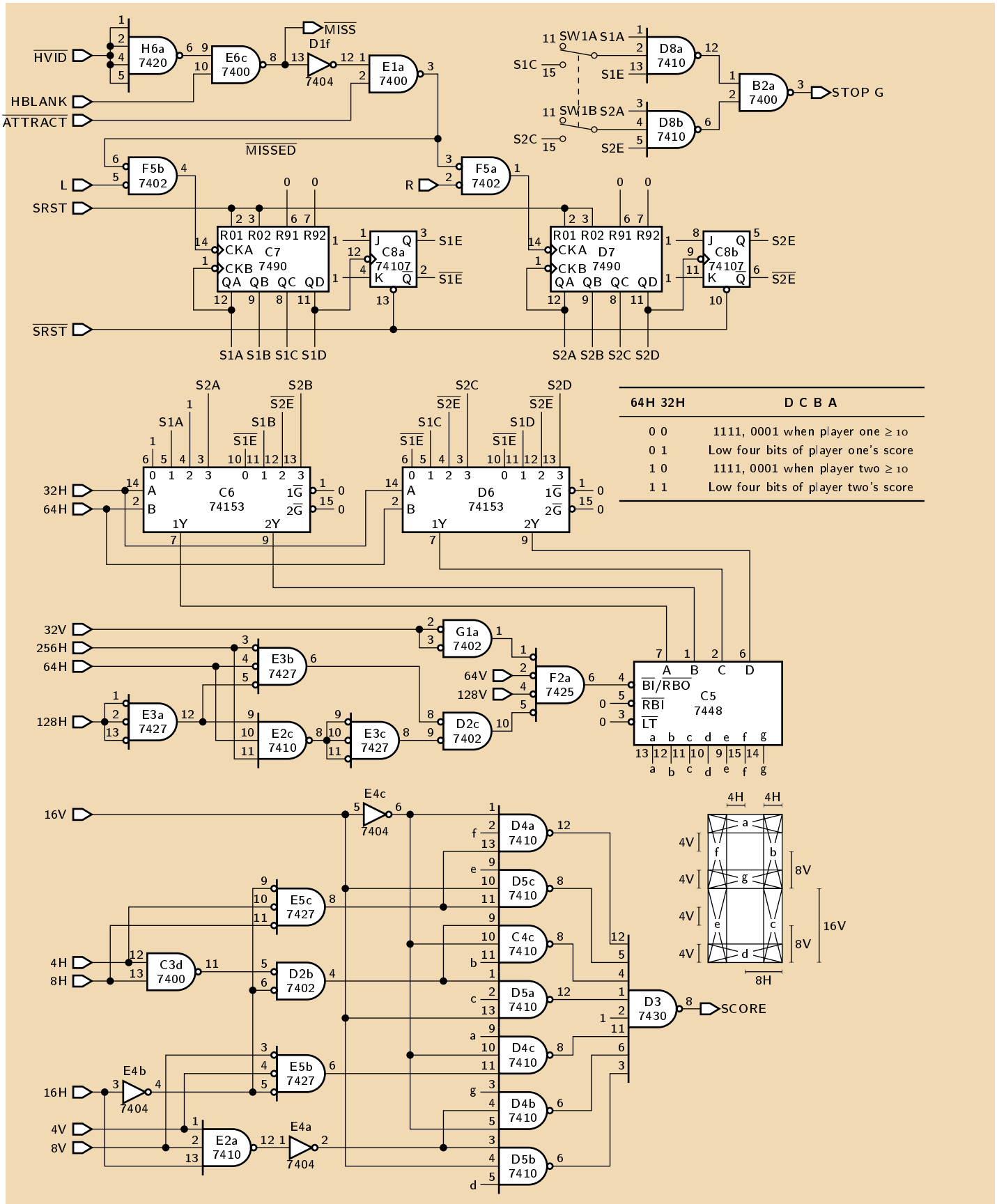


Figure 9: Score circuitry: the outputs of F5b and F5a pulse when a player has scored; C7, C8a, D7, and C8b maintain the score; the remainder displays the score as two two-digit seven-segment decimal numbers.

the output of these gates are fed into an eight-input NAND gate that effectively sums them to produce the final score display signal, which is mixed to produce a slightly dimmer display than the signal for the ball, paddles, and net.

Starting from the top left of Figure 9, $\overline{\text{HVID}}$ is low where the ball is present (see § 1.8), so $\overline{\text{MISS}}$ goes low when the ball reaches the horizontal blanking region (i.e., off the left or right of the screen). When the game is not in attract mode, this causes the $\overline{\text{MISSED}}$ signal to go low, sending the clock of either C7 or D7 high. If the ball was moving left when this happens (i.e., past player one), R will be low, pulsing the clock of D7 high. When it falls, player two's score counter (D7) will increment.

Player one's score counter consists of the four-bit ripple decade counter C6 and JK flip-flop C8a. After C7 reaches a count of nine, it will reset itself to zero on the next cycle, dropping QD and causing C8a to toggle, indicating a score of ten. Player two's score counter (D7 and C8b) is wired similarly.

NAND gates D8a and D8b detect the end-of-game condition, dictated by SW1. In its "11" position, the inputs to D8a and D8b float high, meaning STOP G will rise when either player reaches a score of 11. Similarly, when SW1 is in the "15" position, the first, third, and fifth bits of the score must be one: a score of 15.

Depending on horizontal counter values 32H and 64H, four-input muxes D6 and C6 steer either the low digit of each player's score, 1 when the high-order bit of a player's score is 1, or 15 when it is zero, to the seven-segment decoder C5. The 7448 turns off all digits when fed an input of 15, so this logic disables a leading zero.

Gates E3a, E3b, E2c, E3c, G1a, D2a, and F2a enable the outputs of C5 when the beam is over the score area at the top of the screen.

Gates E4b, C3d, E5c, D2b, E5b, E4a, and E4c compute functions for displaying the segments as shown on the map on the left of Figure 9. Consider the output of D4b. It drives the output of D3 high when g is high, 4V and 8V are high, 16V is low, and 16H is high. This produces the horizontal bar that appears above the midpoint as shown in the diagram. It turns out 16H must be high for any segment to be visible; the constraints on 4V, 8V, 16V, 4H, and 8H vary depending on the segment.

1.8 Horizontal Ball Control

To control the position of the ball, *Pong* uses a clever technique described in Bushnell's 1974 patent [5] that involves running "position" counters at almost the same frequency as the horizontal and vertical counters. In such a situation, the *phase* of these "position" counters relative to the main horizontal counters determine where on the screen the ball will appear; slight perturbations to the period of the position counters cause the ball to move.

Figure 10 shows the circuit responsible for the horizontal position of the ball. This is built around a nine-bit counter built from G7, H7, and G6b, which reloads itself when it reaches a count of 511.

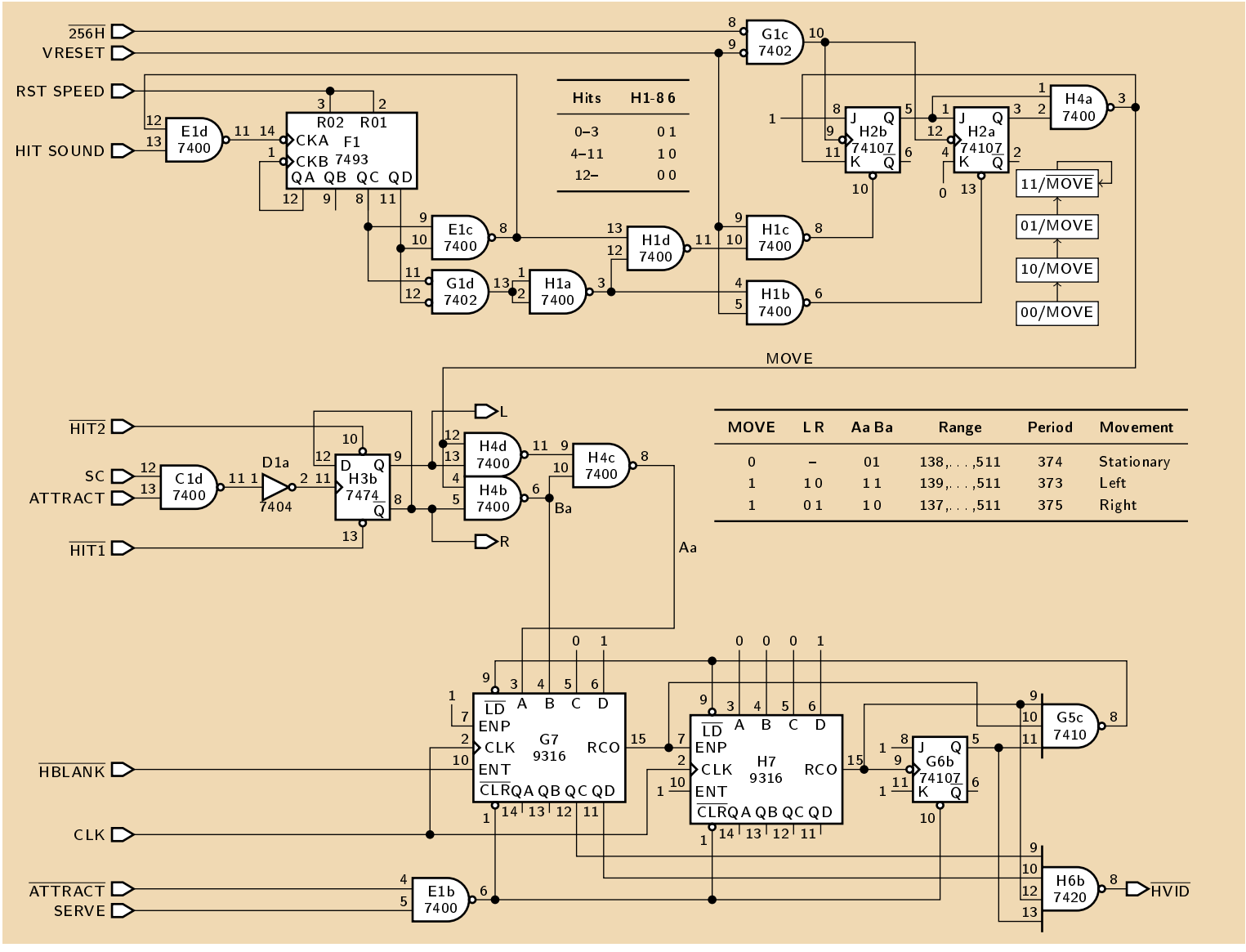


Figure 10: Ball horizontal circuit. The output, $\overline{\text{HVID}}$, is asserted when the ball is visible in the current column. Four-bit counter F1 counts hits and speeds up the ball in response. The duration of the MOVE signal controls how many pixels to move the ball left or right. Finally, the phase of the nine-bit counter formed by G7, H7, and G6b determines the horizontal position of the ball.

When the MOVE signal is low, the horizontal ball counter stays in phase with the master horizontal counters, in effect keeping the ball in the same horizontal position. When MOVE is low. H4b, H4c, and H4d set Aa and Bb, inputs to the synchronous four-bit counter G7 (a 9316, pin compatible with the more familiar 74161 chip), to 0 and 1 respectively. Thus, when the horizontal ball counter reaches 511, the output of G5c goes low and the counter is loaded with the value 010001010 = 128 + 8 + 2 = 138 on the next rising edge of the clock (H7's RCO falls when this value is loaded, toggling G6b, which becomes a 0 because it was a 1 to activate G5c). In this mode, therefore, it counts 138, 139, …, 511: a period of 374.

The horizontal ball counter only advances on rising edges of the clock when $\overline{\text{HBLANK}}$ is high since $\overline{\text{HBLANK}}$ drives G7's ENT input. From the discussion in § 1.4, this occurs during horizontal counts 81, 82, …, 454: 374 times per line. Thus, when MOVE is low, the horizontal ball counter stays in sync with the horizontal counters.

The $\overline{\text{HVID}}$ signal indicates the position of the ball. It is asserted when the horizontal ball counter has values 111111100–111111111, i.e., 508, 509, …, 511, making the ball four pixels wide.

When MOVE is high, the horizontal ball counter resets to a slightly different value, effectively making the ball move left or right. H3b, a D flip-flop, controls the direction. When L is 1, R is 0, Ba is 1, and Aa is 1, making the counter reset to 128 + 8 + 2 + 1 = 139. This makes the period one less then the duration of $\overline{\text{HBLANK}}$, moving the ball to the left.

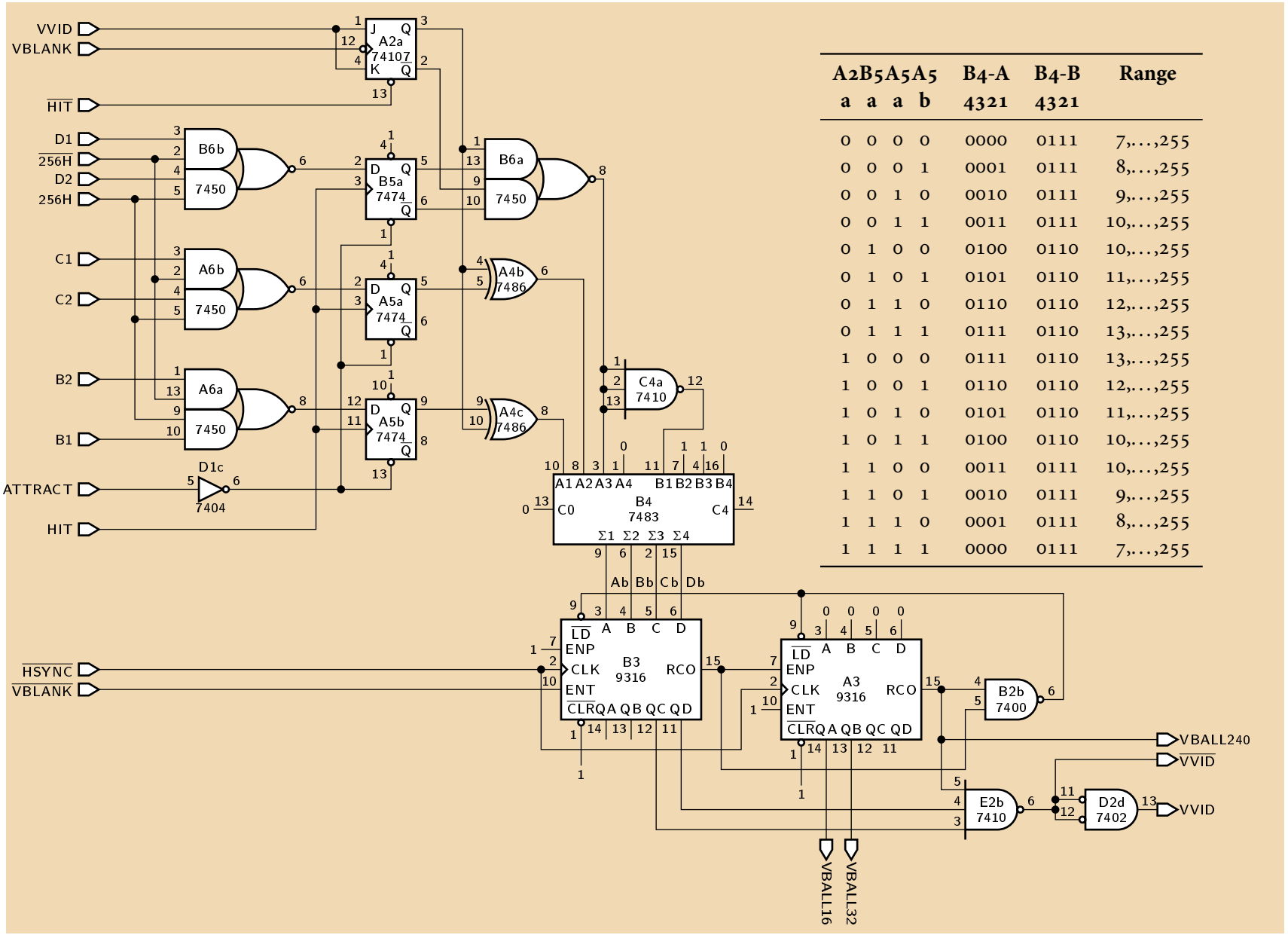
Conversely, when MOVE is high and L is 0, R is 1, Ba is 0 and Aa is 1, making the counter reset to 128 + 8 + 1 = 137 and moving the ball to the right.

H3b makes the ball bounce off the players' paddles. When the left paddle (player one) touches the ball, $\overline{\text{HIT1}}$ goes low, setting H3B into a state where L=0 and R=1, thus sending the ball right. Similarly, colliding with player two's paddle sends the ball left. Note that because these signals set the direction of the ball, rather than change it, there is no danger of the ball becoming trapped inside a paddle that suddenly appears from being moved very quickly.

H3b also makes the ball bounce off the left and right edges of the screen in attract mode. The SC signal pulses high for a few clock cycles when the ball goes off the screen horizontally (i.e., causing a score if the game were being played). In attract mode, C1d passes this pulse through to H3b's clock, causing it to toggle once when the ball hits the edge of the screen.

Ripple counter F1 counts the number of hits, which is used to increase the (horizontal) speed of the ball. RST SPEED goes high briefly during game play when one player scores a point, resetting counter F1. After reset, QC and QD are low, setting E1C's output high. Thereafter, when HIT SOUND pulses high (when the ball hits either paddle), it clocks F1. Once F1 reaches a count of 8 + 4 = 12, E1C's output goes low, inhibiting further counts until the next point.

The output of counter F1 is decoded to control the clear inputs of JK flip-flops H2a and H2b, which form a state machine that controls whether MOVE is asserted on one, two, or three lines per field, controlling the horizontal speed of the ball. Figure 10 shows the state transition diagram for this machine, which is clocked on the falling edge of 256H except on the first line of each field (when VRESET is high), when NAND gates H1b and H1C set the state of this machine.



A2a	B5a	A5a	A5b	B4-A 4321	B4-B 4321	Range
0	0	0	0	0000	0111	7,...,255
0	0	0	1	0001	0111	8,...,255
0	0	1	0	0010	0111	9,...,255
0	0	1	1	0011	0111	10,...,255
0	1	0	0	0100	0110	10,...,255
0	1	0	1	0101	0110	11,...,255
0	1	1	0	0110	0110	12,...,255
0	1	1	1	0111	0110	13,...,255
1	0	0	0	0111	0110	13,...,255
1	0	0	1	0110	0110	12,...,255
1	0	1	0	0101	0110	11,...,255
1	0	1	1	0100	0110	10,...,255
1	1	0	0	0011	0111	10,...,255
1	1	0	1	0010	0111	9,...,255
1	1	1	0	0001	0111	8,...,255
1	1	1	1	0000	0111	7,...,255

Figure 11: Ball vertical circuit. The output, VVID, is asserted when the ball is visible on the current line. Flip-flops B5a, A5b, and A5b remember the position at which the ball last hit a paddle; A2a toggles to rebound the ball off the top or bottom. Four-bit adder B4 translates the ball motion signals into a reset value for the eight-bit counter formed by A3 and B3, which determines the vertical position of the ball.

Figure 11 shows the circuit responsible for the vertical position of the ball. Like the horizontal circuit, it consists of a counter (only eight bits for the vertical counter: A3 and B3) whose period, and hence vertical ball speed, is set by a state machine affected by a ball hitting a paddle or the top or bottom of the screen.

When the game is in attract mode, the paddles are hidden and the ball travels at one speed diagonally, bouncing off the sides of the screen. In this state, inverter D1C drives the clear inputs of D flip-flops B5a, A5a, and A5b, setting their Q's to all 0.

JK flip-flop A2a, together with exclusive-OR gates A4b, A4c, and B6a (an AND-OR-invert gate wired to compute exclusive-OR) are responsible for bouncing the ball off the top and bottom of the screen. When A2a's Q is low, the exclusive-OR gates pass the outputs of the three D flip-flops unchanged and invert them when the Q output is high. The state of A2a toggles when the ball hits the screen top or bottom: when VVID is high (i.e., when the ball is visible on the current line) at the end of vertical blanking (when VBLANK falls).

As in the horizontal circuit, the phase of the vertical counter (formed by synchronous four-bit counters A3 and B3) affects the vertical position of the ball and the period of the counter affects the ball's vertical velocity. Here, however, the counter is clocked by the rising edge of $\overline{\text{HSYNC}}$ and gated by $\overline{\text{VBLANK}}$. As discussed in § 1.4, there are 262 lines per field and $\overline{\text{VBLANK}}$ is active for 16 of them, giving the ball vertical counter 262 − 16 = 246 counts per field. Thus loading the vertical ball counter with 10 will hold the ball in the same position vertically; smaller numbers increase the period, causing the ball to move down on the screen; larger numbers will cause it to move up. In attract mode, the counter is loaded with 7 or 13 depending on A2a, so it will move at the maximum speed vertically.

During gameplay, the ball rebounds off the paddle at an angle determined by where the ball hit the paddle. D flip-flops B5a, A5a, and A5b remember this collision location; binary adder B4 transforms it into a value with which to load the ball vertical counter.

When the ball hits the paddle, HIT pulses high, resetting A2a and loading D flip-flops B5a, A5a, and A5b with the one's complement of the top three bits of one of the paddle counters (see § 1.6). And-or-invert gates A6a, A6b, and B6b are wired as two-input multiplexers that pass the top three bits of player one's paddle position (B1, C1, and D1) when 256H is low (i.e., the ball is to the left of the net) and player two's paddle position when 256H is high.

In the table in Figure 11, note that the range 10,...,255 appears four times instead of two. This makes it easier to rebound the ball purely horizontally—there is a large area in the center of the paddle where this will happen.

It appears the inputs to A6a are wired incorrectly. While B6b and A6b select data from player one when 256H is low as we would expect, A6a selects data from player two. The schematic in Figure 11 is consistent with the original schematics (Figure 12) as well as the board.[3] Even the schematic for an (unauthorized) clone of Pong has this error [1].

[3] I traced the signals on a high-resolution photo of the front and back of the two-layer board.

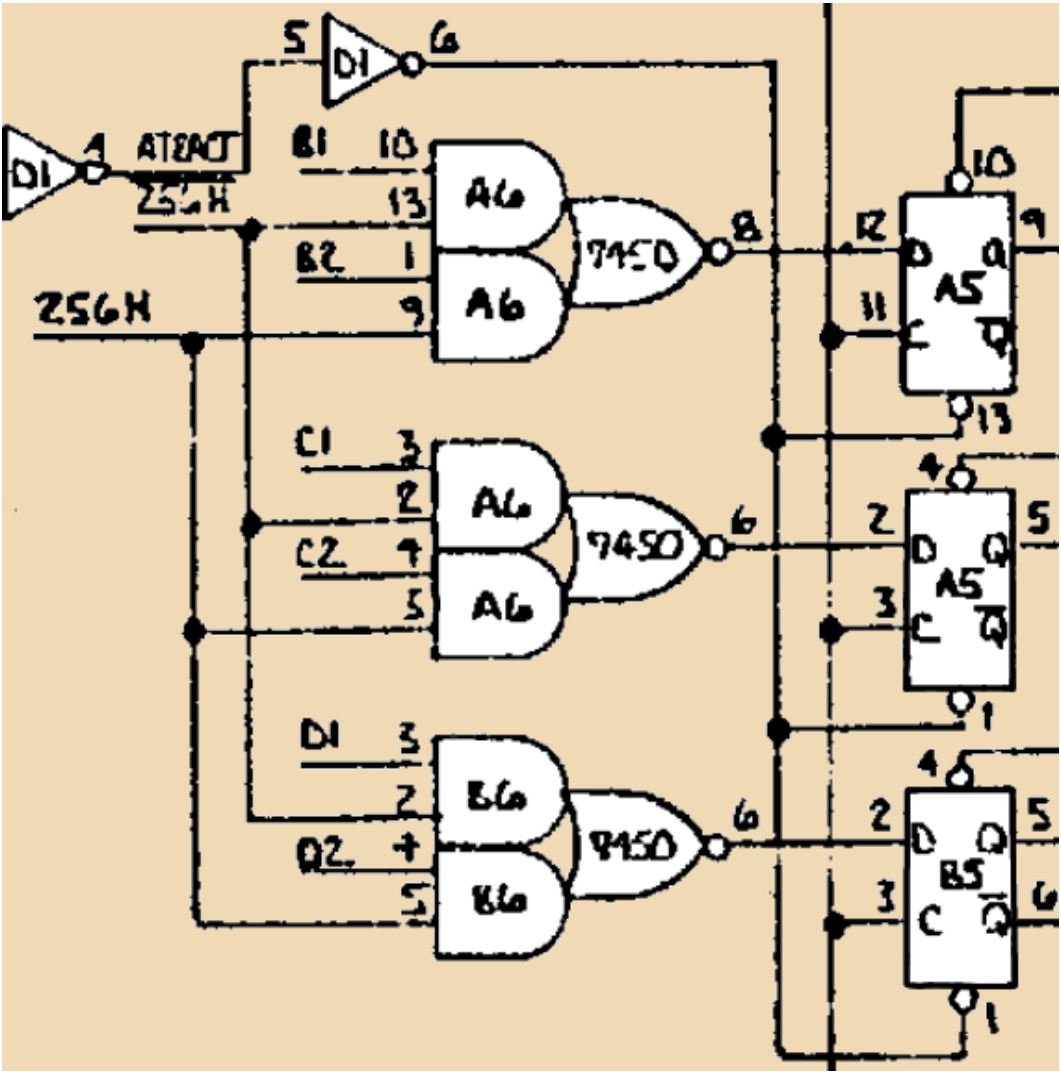


Figure 12: Part of *Pong*'s original schematic: the vertical ball counter. The intention of the top AND-OR-invert gate (A6) is clear: pass B1 under the same conditions as C1 and D1, but pins 1 and 10 are swapped: pins 1 and 13 are one group; 9 and 10 are the other.

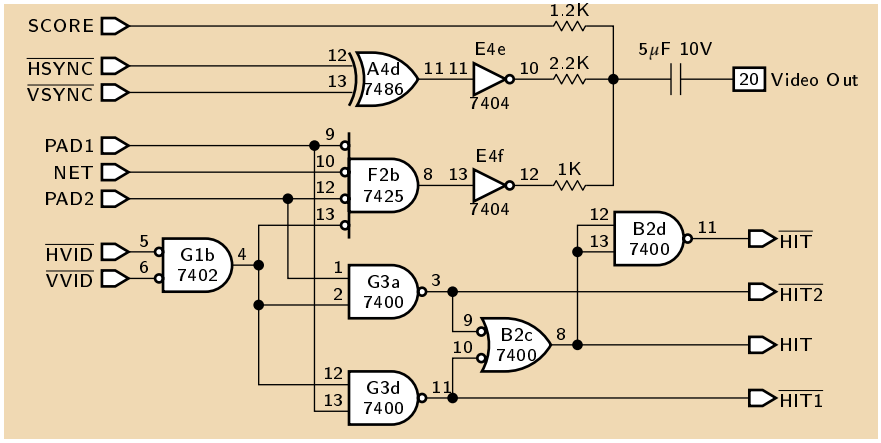


Figure 13: Video generation circuitry. F2b combines the signal from the two paddles, the net, and the ball. The score display is mixed with a 1.2K resistor, making it slightly dimmer.

1.10 Video Generation

Figure 13 shows the circuit that combines the horizontal and vertical synchronization signals (§ 1.4), the net (§ 1.5), paddles (§ 1.6), score (§ 1.7), and the ball (§§ 1.8 and 1.9) to produce the final composite (black-and-white) video signal. A resistor summing network combines the score, sync, and other video elements. Note a higher-value resistor is used for the score (1.2K); this makes it slightly dimmer than the paddles, ball, and net. A 5μF coupling capacitor removes any DC bias.

Also in Figure 13 is the logic generating HIT, $\overline{\text{HIT1}}$, and $\overline{\text{HIT2}}$, used, e.g., by the ball vertical circuit.

1.11 Sound

The sound in *Pong* is legendarily simple, perfect, and almost serendipitous, as Alcorn relates:

> Now the issue of sound …People have talked about the sound and I've seen articles written about how intelligently the sound was done and how appropriate the sound was. The truth is, I was running out of parts on the board. Nolan wanted the roar of a crowd of thousands—the approving roar of cheering people when you made a point. Ted Dabney told me to make a boo and a hiss when you lost a point, because for every winner there's a loser.
>
> I said, "Screw it, I don't know how to make any one of those sounds. I don't have enough parts anyhow." Since I had the wire wrapped on the scope, I poked around the sync generator to find an appropriate frequency or a tone. So those sounds were don in half a day. They were the sounds that were already in the machine. [11]

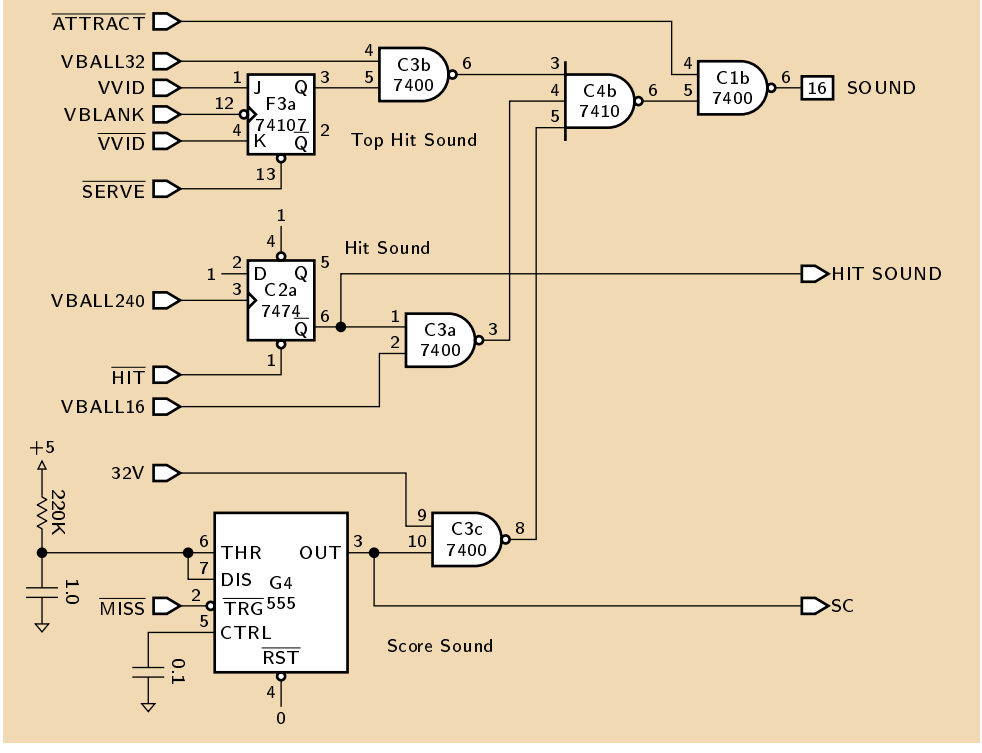


Figure 14: Sound Circuitry

Figure 14 bears this out. *Pong* generates three sounds: a "ping" sound when the ball hits a paddle, a "pong" sound when the ball reflects off the top or bottom of the screen, and a similar sound when either player scores a point.

JK flip-flop F3a is active when the ball is bouncing off the top or bottom, much like A2a does for the vertical ball control (Figure 11). F3a takes a new value at the top of each screen (i.e., when VBLANK falls before the first line of the screen). If the ball was visible at the end of vertical blanking (i.e., went off the top or bottom of the screen), F3a's Q output goes high for a field, enabling C3b, which passes VBALL32 (the sixth bit of the vertical ball counter). This is 15.734 kHz / 16 = 980 Hz, near B_5 on the piano.

D flip-flop C2a turns on briefly after the ball is hit. Normally, C2a's $\overline{Q}$ is low, disabling C3a, but when the ball is hit, the C2a's reset goes low, sending $\overline{Q}$ high and gating VBALL16 (the fifth bit from the vertical ball counter). This is 15.734 kHz / 32 = 490 Hz, near B_4 on the piano.

Timer G4 activates the score sound for a period of time after $\overline{\text{MISS}}$ pulses low, which occurs when the ball goes off the screen horizontally—see Figure 9. The 220K resistor and 1.0μF capacitor set the width of the pulse, which is roughly $1.1 \cdot 220\text{K} \cdot 1.0\mu\text{F} = 240$ ms. This sound is set by 32V, the sixth bit of the vertical counter, which also counts at about 980 Hz.

C4b "sums" the three sound sources, which are silenced during attract mode by C1b. The sound output is sent, unamplified, to the TV monitor.

1.12 Game Control

Figure 15 shows the game control circuitry. It consists of a debouncing circuit for the coin switch (inverters C9c and C9f), a latch/power-on reset circuit built from discrete transistors (Q1, Q2, and associated resistors and capacitors), and the serve timer (F4).

When the game powers up, the coin switch pulls SRST low, $\overline{\text{SRST}}$ high, and transistors Q1 and Q2 remain off, allowing the 100Ω and 300Ω resistors to pull $\overline{\text{RUN}}$ high, so $\overline{\text{ATTRACT}}$ falls and the game enters attract mode (paddles are hidden, ball bounces off the sides of the screen instead of causing a score).

Inserting a coin flips the state of the Q1-Q2 latch to indicate the game is being played: $\overline{\text{SRST}}$ briefly pulses low, resetting the score and pulling Q1's collector low through the 1N914 diode. This drops the voltage on Q2's base and turns it on, raising the voltage on Q1's base and turning it on, pulling $\overline{\text{RUN}}$ down, which keeps Q2 turned on after $\overline{\text{SRST}}$ rises.

At the same time, when $\overline{\text{SRST}}$ pulses low, RST SPEED pulses high to reset the horizontal ball speed and triggers 555 F4, the serve timer, which is wired as a one-shot, so F4's OUT pulses high for roughly $1.1 \cdot 330\text{K} \cdot 4.7\mu\text{F} = 1.7$s. This puts the output of E5a low, which asserts SERVE. Once OUT falls, the output of E5a rises and SERVE falls on the next rising edge of PAD1.

When either player's score reaches 11 or 15, the game ends: the score counters assert STOP G, which pulls the base of Q1 low, turning off Q2 as well, raising $\overline{\text{RUN}}$ and asserting ATTRACT.

The "Antenna" input is intended to prevent free games initiated by a static shock to, say, the coin logic. If the voltage on the base of Q3 rises, it pulls the base of Q1 low, turning it off and letting $\overline{\text{RUN}}$ rise.

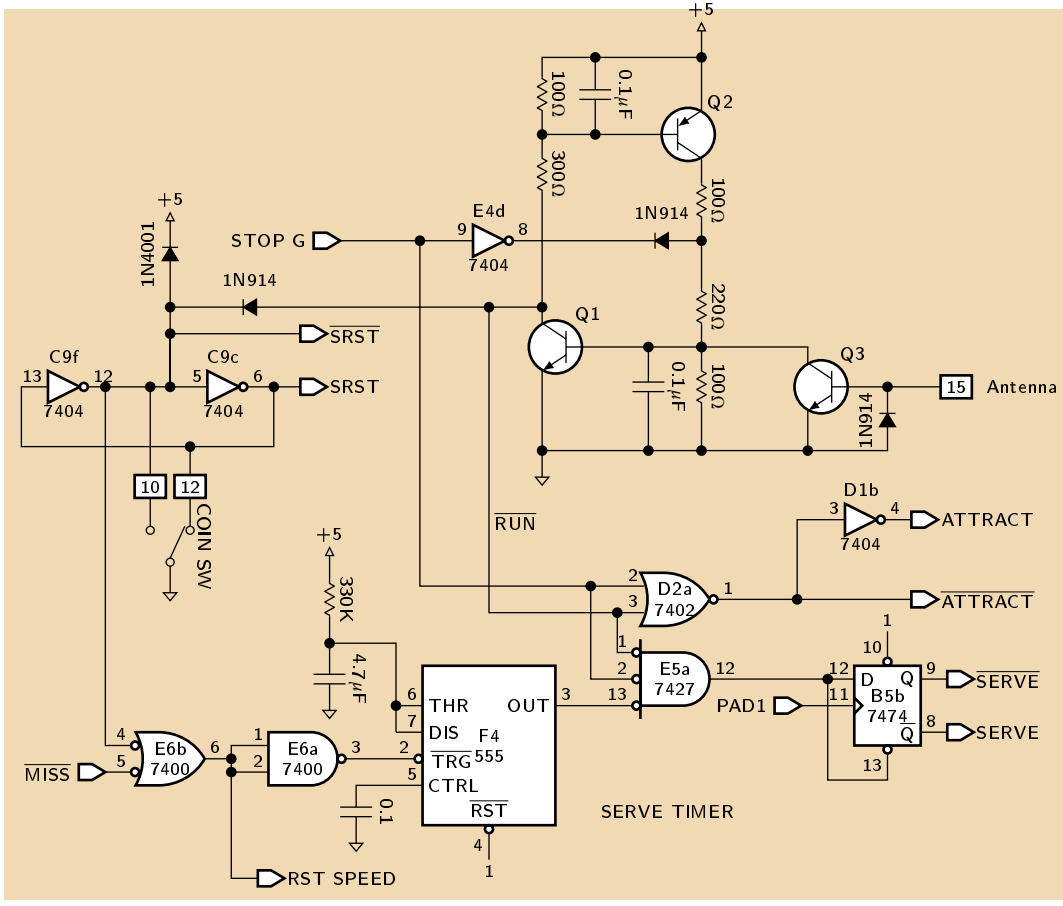


Figure 15: Game Control Circuitry

2 Reconstructing Pong on an FPGA

Virtually all of *Pong* is digital so implementing it on a modern FPGA is feasible, but not straightforward. First, there are a handful of analog sections: the master oscillator; the one-shot timers for the paddles, score, and serve; and the final video output circuitry. FPGAs typically use external clock oscillators anyway, so this part is easy. The timers can be emulated with counters. The video circuitry is a primitive D/A converter with four levels: sync, black, gray, and white. Gray is used for displaying the score; the ball, paddles, and net are all white. As described earlier, *Pong*'s sound is actually digital: a 1-bit output produces square waves at three frequencies.

2.1 Handling Quasi-Synchronous Circuits

Despite utilizing a 7.159 MHz master oscillator, *Pong* is not classically synchronous. Deviations from this ideal (i.e., acyclic combinational logic between flip-flips driven by a global clock) include the ripple counters in the horizontal and vertical circuits, which cause the unexpected timing of $\overline{\text{HBLANK}}$ that I described in § 1.4; R-S latches built from discrete gates that generate horizontal and vertical sync (Figure 5); and numerous flip-flops driven by clocks driven by derived signals and employing asynchronous set and reset signals. Most of these were reasonable shortcuts to take in 1972, but are anathema in modern FPGA designs.

My solution for reconstructing *Pong* in a purely synchronous style was to construct synchronous circuits that essemtially simulate the asynchronous aspects of the original *Pong* circuit. My goal was to make the behavior of the reconstructed circuit match the original on the edges of the 7.159 MHz clock (since parts of *Pong* are sensitive to both rising and falling edges) and to assume the absence of glitches on the generated clocks in the original circuit, i.e., that any switch more quickly than the global clock.

Figure 16 shows the circuit I use to emulate the 7474: a positive-edge-triggered dual D flip-flop with asynchronous set and reset inputs. I replace each original D flip-flop with three flip-flops: "Q" to hold the current state, "D" to capture the (possibly irrelevant) next state, and "C" to remember the previous value of the (asynchronous) clock.

This circuit assumes it is clocked fast enough so that neither the clock nor the "D" input transitions more than once per cycle. The AND gate marked "RISE" and the "C" flip-flop detect whether CLK has risen since it was last sampled. If CLK rose, the mux supplies the value of the D input sampled before the rising edge by the "D" flip-flop. Otherwise, CLK was stable and the mux delivers the previous value of Q, held by the "Q" flip-flop. Combinational inputs $\overline{\text{CLR}}$ and $\overline{\text{PRE}}$ set and clear the output regardless of the behavior of CLK; the feedback loop ensures their effect is felt in the next cycle even if CLK did not rise.

This circuit can simulate the original flip-flop provided it is run at least twice as fast. In the exactly twice-as-fast case, the CLK input is high in alternate cycles and makes the mux alternate between the "D" and "Q" flip-flops, which latches the D input in cycles when CLK is low.

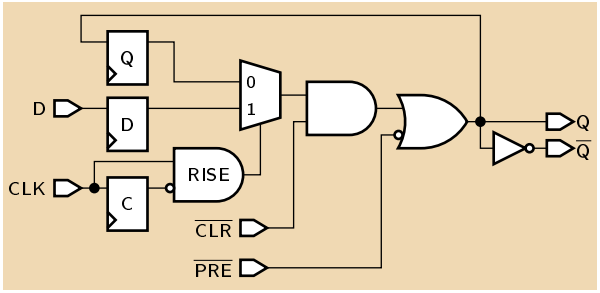


Figure 16: My circuit for emulating a 7474 positive-edge-triggered D flip-flop. The AND gate "RISE" detects a rising edge on the CLK input and either delivers the old Q output or the newly latched value of D.

The remaining sequential elements in *Pong* are 74107 negative-edge-triggered JK flip-flops, 7493 four-bit ripple counters, 7490 four-bit ripple decade counters, and 9316 four-bit synchronous counters. My circuits for each closely parallel that for the 7474 I describe above. In particular, I assemble the ripple counters from a cascade of JK flip-flops.

For the R-S latches for HSYNC and VSYNC, I simply inserted a one-cycle delay (a D flip-flop) to break the combinational feedback loop.

2.2 *A Minimal Hardware Description Language*

VHDL and Verilog are the hardware description languages accepted by most synthesis tools, including the ones I used. I certainly could have written the *Pong* netlist in one of these formats, but I chose to develop a simple language of my own to keep the length of the description to a minimum. I chose its syntax to be simple enough to "parse" with an AWK script.

The basic challenge is to specify the type and connectivity of each component. To do this, my basic syntax resembles that of SPICE: a typical line describes an instance of a component and consists of a component type, a designator, and a list of connected nets ordered by the pins defined for the component. Figure 17 shows my code for the horizontal counter. I cut a corner here and used a five-input NAND gate even though the actual circuit uses a 7430 eight-input NAND with three inputs tied high.

A net name is any sequence of printable characters (not space, newline, or control characters). This allows the use of existing, natural names such as "64H" (horizontal counter bit representing sixty-four), notation such as /HRESET to represent the complement of the HRESET signal, and "0" and "1" to represent logical false and true.

Nets are never explicitly defined; their first reference in an instance line brings them into scope. Each is assumed to carry a single bit.

One special rule: the "underscore" net (_) means "unconnected;" it is intended for representing unconnected outputs since I do not provide a connect-by-name syntax. I adopted this from the ML-derived functional languages.

```
# Horizontal counter
SN7493 F8 CLK7M 1H HRESET HRESET 1H 2H 4H 8H
SN7493 F9 8H 16H HRESET HRESET 16H 32H 64H 128H
SN74107 F6B 1 1 128H /HRESET 256H /256H
TTLNAND5 F7 256H 4H 2H 64H 128H n13
SN7474 E7B n13 1 CLK7M 1 /HRESET HRESET
```

Figure 17: Code in my HDL for defining the horizontal counter circuit; cf. Figure 2. The # line is a comment. Each other line lists a component name, a component designator, and the names of the nets to which its pins should connect.

```
:TTLINV A !Y
  Y <= not A;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity TTLINV is
  port (
    clk : in std_logic;
    A : in std_logic;
    Y : out std_logic
  );
end TTLINV;

architecture ttl of TTLINV is
begin
  Y <= not A;
end ttl;
```

Figure 18: An inverter component defined in my HDL and the VHDL code I generate for it.

A component definition line starts consists of a colon, the name of the component, and a space-separated list of pins. Pin names prefixed with an exclamation mark are outputs; all others are inputs. Lines after the initial definition are treated as VHDL that defines the body of the component. The body definition continues until the next non-empty line that does not start with a space (e.g., a comment, another component definition, or an instance). The lines containing the VHDL keyword "signal" after the initial space are understood to define signals and are placed before the "begin" keyword for the body of the instance.

Figure 18 compares the definition of an inverter component in my HDL to one my translator generates for one in VHDL. Note that the `clk` input is added automatically to each component to support sequential elements; it goes unused on this entity.

Figure 19 show some simple component definitions followed by a more complicated one. Note the use of the VHDL \…\ extended identifier notation.

```
:XOR2 A B !Y
  Y <= A xor B;
:TTLNAND2 A B !Y
  Y <= A nand B;
:TTLNAND5 A B C D E !Y
  Y <= not (A and B and C and D and E);
# Delta delay to break feedback loops (e.g., in RS latches)
:DELTA D !Q
  process (clk)
  begin
    if rising_edge(clk) then
      Q <= D;
    end if;
  end process;

# (Dual) D-type positive-edge-triggered flip-flops with preset and clear
:SN7474 D /PRE CLOCK /CLR !Q !/Q
  signal next_d, last_d, next_q1, next_q, last_q, last_clock : std_logic := '0';

  process (clk)
  begin
    if rising_edge(clk) then
      last_d <= next_d;
      last_q <= next_q;
      last_clock <= CLOCK;
    end if;
  end process;

  next_q1 <= last_d when clock = '1' and last_clock = '0' else
             last_q;
  next_q <= (next_q1 and \_CLR\) or not \_PRE\;
  next_d <= d;

  Q <= next_q;
  \_Q\ <= not next_q;
```

Figure 19: Definition of some combinational gates, a “delta” delay component, and the definition for an “asynchronously clocked” D flip-flop; cf. Figure 16.

I targeted a Terasic DE2 board as the emulation platform for *Pong*. *Pong* has limited I/O: a coin input, two paddle inputs, and audio and video out.[4] Each of these needs to be emulated on the board.

The coin input was the simplest: the DE2 has a number of keyswitches that provide one-bit inputs. I coded a small synchronous module that emulates the discrete components of the game control circuitry, basically the top half of Figure 15, that generates $\overline{\text{SRST}}$ and $\overline{\text{RUN}}$ from the coin input and STOP G.

The video output was probably second hardest. The DE2 has a video DAC driving a standard 15-pin VGA jack and I had plenty of VGA-capable LCD panels around. One problem is that *Pong* generates NTSC-speed video (15.734 kHz horizontal refresh), while VGA expects exactly twice that (31.4686 kHz). I implemented a line doubler that uses a double-ported 1024 × 2bit RAM. The line doubler fills half of the buffer with video data from *Pong* (two bits per pixel: one representing the brightness for the score; the other representing all others) while displaying the other half. Every other NTSC line, the roles of the two halves of the buffer are swapped.

The line doubler expects exactly the horizontal timing produced by *Pong*, generates a VGA-speed horizontal sync signal, and passes the vertical sync signal directly to the monitor.

The audio output, ironically, was even more difficult. The DE2 has a Wolfson WM8731 24-bit stereo audio codec connected to 3.5mm headphone jacks, which is certainly overkill for *Pong*'s 1-bit digital audio output, but I chose to use it anyway because I did not want to build additional hardware.

The WM8731 must be configured through an I²C bus interface, so I took a circuit from other projects that generates the appropriate I²C signals to configure the codec along with the logic necessary to generate the timing signals it desires (clocks, framing, left/right). After all of this, I simply feed the single bit from the *Pong* audio output into the most significant bit of both channels of the codec. It beeps.

The two paddle inputs presented a conundrum: while perhaps the most authentic route would have been to build the 555-based timing circuitry and communicate appropriate timing pulses to/from the *Pong* core, I wanted to avoid adding anything but off-the-shelf peripherals. I settled on connecting a mechanical PS/2 mouse to the DE2 because it had the proper connector and is fairly easy to interface, but manually spinning the two optical encoders in the mouse hardly makes for convenient game play.

I instantiated an open-source PS/2 controller block and created a fairly complicated state machine that reset the mouse and sent an "enable data reporting" byte to it so it would start sending back movement information that my FSM could interpret and convert to coordinates.

Finally, the raw coordinates are converted into counts for a 555 timer emulator circuit that waits for a "start" pulse to start a counter that times out based on the coordinates from the mouse controller.

[4] It also has an antenna input and a coin counter output, which I ignore.

Alcorn's comments about the upper range of the paddle timers suggest that it would be worthwhile to have a more careful understanding of exactly the low and high ranges of the paddle timers; in my current design, they are sufficient to make the game playable, but may provide a wider range of motion than the original game.

There are two more 555 timers in *Pong*: one that sets the duration of the "score" sound, and one that controls the serve delay. Each are configured as digitally triggered one-shots whose period is set by RC networks. To emulate these, I created digital counters whose delays roughly match those of the 555's. I did not attempt to match the delays precisely because of *Pong*'s use of low-tolerance components (an RC network with an electrolytic capacitor) and because they do not drastically affect game play.

3 Conclusions

It was great fun going through and understanding the circuitry of the original *Pong* in such detail, although it took far longer to redraw the schematics and write about them than I expected. The FPGA reconstruction was easier since someone else had already digitized the schematic, but it was still puzzling at times, largely because of timing anomalies arising from imperfectly synchronous behavior.

The use of a PS/2 mouse as a controller is unsatisfactory, although it could be reasonable if there was a convenient way to remove and separate the two optical interrupters.

Acknowledgements

I must thank Andrew Welburn [16], who subjected his own Syzyzy *Pong* board to a logic analyzer to verify my hypothesis about the horizontal timing, the delays of the various 555s on the board. He also pointed out value of the 5K pots, supplied photos, confirmed my observation about the odd wiring around A6, and confirmed my reconstruction exhibited many of the same odd behaviors as the original.

References

[1] Allied Leisure Industries, Inc., 245 West 74th Place, Hialeah, Floria 33010. *Allied's Paddle Battle Parts & Wiring Catalog*, 1973.

[2] Atari, Inc., 14600 Winchester Boulevard, Los Gatos, California. *Pin Pong Operation and Maintenance Manual*, 1974. Part Number TM-007.

[3] Atari, Inc. *Space Race Service Manual*, 1976. Part number TM-008. Schematics dated 8/5/74.

[4] Dan Boris. Dan boris' tech blog. Online, 2007. http://www.atariage.com/forums/blog/52-danboris-tech-blog/.

[5] Nolan K. Bushnell. Video image positioning control system for amusement device. US Patent 3,794,383, February 1974.

[6] Brian Deuel. Interview with Al Alcorn. Online, 2000? http://atari.vg-network.com/aainterview.html.

[7] Stephen A. Edwards. Retrocomputing on an FPGA. *Circuit Cellar*, 233:24–35, December 2009.

[8] Mike Johnson et al. FPGA arcade. Online. http://fpgaarcade.com.

[9] Nicola Salmoria et al. MAME: The multiple arcade machine emulator. Online, 1997–. http://mamedev.org.

[10] James Electronics. Advertisement. *Byte Magazine*, 1(1):83, 1975. An SN7493N was $0.82; an SN74161N was $1.45.

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

[12] Nutting Associates. *Two-Player Computer Space Trouble-Shooting Guide*, April 1973. Nolan K. Bushnell, Chief Engineer.

[13] Cam Shea. Al alcorn interview. Online, March 2008. http://retro.ign.com/articles/858/858351p1.html.

[14] Texas Instruments. *The TTL Logic Data Book*, 1988.

[15] William Arkush (uncredited). *The Textbook of Video Game Logic*, volume I. Kush N' Stuff Amusement Electronics, 60 Dillon Avenue, Campbell, California, 1976.

[16] Andrew Welburn. Andys-arcade. Online. http://www.andysarcade.net.