Game Controller FSM - Manual

Circuit 7 (Simon Says Design Project) - FSM, Round Counter, Random Start Address

How the FSM maps to gameplay

The Game Controller is the part of the Simon Says machine that knows where the game is in its lifecycle. It does not store the color sequence or compare presses; those jobs belong to the Pattern Display (SP) and Pattern Matcher (PM) subsystems. The FSM watches a small set of event signals from those subsystems plus the START button, and it tells the rest of the machine what to do next: load a counter, set a latch, increment the round, hold steady, or restart. Alongside the FSM, a 4-bit round counter tracks how many rounds the player has completed in the current game, and a random start-address generator picks a different EPROM address to start each game.

From the player's point of view, the game cycles through four moments. The machine sits in IDLE with no LEDs blinking and the round counter at 0. The player presses START. The machine transitions to PLAY and shows the round's color sequence on the LEDs - starting from a freshly captured random address, so each game's sequence is different. The machine then transitions to USER and waits for the player's button presses. Each correct press is logged. If the player completes the round correctly the round counter ticks up by one and the FSM returns to PLAY for the next, longer sequence (still starting at the same random address; the sequence builds outward from there). If the player presses wrong or runs out of time the FSM drops to IDLE and lights the LOSE indicator. If the player completes round 15 the FSM drops to IDLE and lights the WIN indicator. Pressing START again clears the indicators and the round counter, captures a new random start address, and begins a new game.

The tables below describe the FSM's state codes, event codes, transitions, outputs, the round counter, and the random start-address subsystem. The Test section after that walks through what to set and what to look for to confirm the circuit behaves correctly.

State encoding

The state register holds two flip-flops, Q0 (MSB) and Q1 (LSB). Three of the four possible bit patterns are named gameplay states; the fourth (BLNK = 10) is reserved as an unreachable recovery code.

State	Q0	Q1	Meaning
IDLE	0	0	Game idle. The machine waits for the START button. Used for initial power-up and the rest period after a game ends.
PLAY	0	1	The machine is showing the player the next color sequence. Pattern Display drives the color LEDs.
USER	1	1	The player is pressing color buttons. The Pattern Matcher compares each press against the stored pattern.
BLNK	1	0	Unreachable in normal operation. Recovery is built into the next-state logic so any glitch into BLNK returns to IDLE on the next clock.

Event codes

The FSM consumes a 3-bit event code (E0 E1 E2, MSB-first) from a priority encoder upstream. Five of the eight codes are decoded into named event rails inside the FSM; the other three either hold the FSM steady or are unused.

Code (E0 E1 E2)	Event Name	Meaning
0 0 0	START	Player pressed the START button.
0 0 1	PD_FIN	Pattern Display finished playing the round's sequence; player's turn now.
0 1 0	LOSE	Either the response timer expired or the player pressed the wrong color.
0 1 1	(no rail)	Correct mid-round press. The Pattern Matcher self-increments; the FSM holds USER.
1 0 0	ADV	Player completed a round successfully and more rounds remain.
1 0 1	WIN	Player completed the final round (round 15) successfully.
1 1 0	(unused)	Encoder code not assigned. No effect on the FSM.
1 1 1	(no event)	No event signal active. The FSM holds its current state.

Event encoder (CircuitVerse-only subcircuit)

The CircuitVerse sim has an Event Encoder subcircuit between the event toggle switches and the rest of the FSM. It maps each asserted event input to the corresponding 3-bit event code the FSM expects, in a single combinational layer. On the integrated breadboard, this subcircuit's role is filled by a 74LS148 priority encoder with the same code mapping. The CircuitVerse version is simpler because Simon Says events are mutually exclusive by design — only one event can fire at a time given the current game state — so a priority chain isn't needed.

Internal logic uses two 2-input OR gates plus a direct wire: E0 (LSB) = PD_FIN OR WIN; E1 = LOSE; E2 (MSB) = ADV OR WIN. The START input is intentionally not wired into any output gate. START's encoded value is 000, which is also the encoder's default output when no input is asserted, so START needs no logic — the encoder produces its code automatically. The toggle still exists on the schematic as a UI element so the tester has somewhere to register the START event, but its signal goes nowhere downstream. This mirrors a known limitation of the real 74LS148: a separate GS (group select) signal would normally distinguish 'START asserted' from 'no input,' but the simple sim encoder doesn't model GS — pressing CLOCK with no event toggled would be interpreted as START. The test procedure works around this by only clocking when a specific event is intended.

State transitions

Each row shows the state the FSM is in (From), the event it sees, the next state it transitions to on the clock edge, and any one-cycle Mealy pulses that fire on that transition. RESET is asynchronous and overrides everything; it is not part of the clocked transition table.

From	Event	To	Pulses Fired
IDLE	START	PLAY	LDPD, GMRST
IDLE	any other	IDLE	(no pulses)
PLAY	PD_FIN	USER	LDPM
PLAY	any other	PLAY	(no pulses)
USER	ADV	PLAY	LDPD, RINC
USER	WIN	IDLE	SWIN
USER	LOSE	IDLE	SLOSE
USER	any other	USER	(no pulses)
BLNK	any	IDLE	(no pulses)
any	RESET (async)	IDLE	(no pulses; latches clear)

Outputs

The FSM produces eight outputs. Two are Moore (state-only, level signals) and six are Mealy (state and event, single-cycle pulses on transitions). The Mealy pulses drive the rest of the Simon Says machine - counters load, latches set, the round increments, the random start is captured - at exactly the right clock edge.

Signal	Type	Active when	Purpose / Drives
PLAYD	Moore	State = PLAY	Drives the color LED bank and the Pattern Display's enable. Lit any time the machine is showing a sequence.
USERD	Moore	State = USER	Drives the Pattern Matcher's enable and the EPROM address mux select. Lit any time the player is pressing buttons.
LDPD	Mealy	START-IDLE or ADV-USER	Pulse that loads the Pattern Display's step counter to the round's start address. Fires at game start and at the start of every new round.
LDPM	Mealy	PD_FIN-PLAY	Pulse that loads the Pattern Matcher's user-turn counter at the start of the player's turn.
RINC	Mealy	ADV-USER	Pulse that increments the round counter on a successful round.
SLOSE	Mealy	LOSE-USER	Pulse that sets the LOSE latch when the player loses.
SWIN	Mealy	WIN-USER	Pulse that sets the WIN latch when the player wins the final round.

Signal	Type	Active when	Purpose / Drives
GMRST	Mealy	START-IDLE	Pulse fired at game start. Clears the LOSE and WIN latches, clears the round counter, AND captures the random start address (see below).

Round counter

A 4-bit ripple counter sits next to the FSM in the Game Controller block. It tracks how many rounds the player has completed in the current game, and signals when the player is on the final round so the encoder upstream can fire WIN instead of ADV on the next correct press.

Counter signal	Description
CLK input	Tied to the FSM's RINC Mealy pulse. The counter increments by one on each rising edge of RINC.
CLR input	Tied to (GMRST OR RESET). Clears the counter to 0 at every game start AND on a global reset press.
Count output (4 bits)	Current round number, 0 through 15. Displayed on a hex digit (0 through F) for visibility.
MAX_RND output	Goes high when the count equals 15 (1111). Tells the encoder upstream that the player is on the final round, so the next correct press fires WIN instead of ADV.

In CircuitVerse, the count output is wired to a hex display so the round number is readable as a single digit. In the integrated breadboard build, the counter is a 74LS191 with the same pin shape; MAX_RND comes off the chip's RCO pin.

Random start address

To make each game's color sequence different, an 8-bit free-running counter (2× 74LS191 cascaded) spins continuously inside the Game Controller block. On the rising edge of GMRST (the game-start pulse), an 8-bit D register (74374) captures the random counter's instantaneous value into start_address. That captured value holds for the entire game and feeds the PD step counter's parallel-load on every LDPD. Each new START-IDLE fires a fresh capture, so consecutive games land on different starting addresses without needing a manual RESET between them.

Random subsystem signal	Description
Random counter CLK	Tied to a fast free-running clock (CV: a Clock component running at 5-10 Hz). The counter spins continuously, much faster than the player can react.
Random counter CLR	Global RESET only. The counter starts spinning again from 0 after a reset; otherwise it just keeps cycling.
STRT_ADR output (8 bits)	Live counter value, exposed to the top level. Changes every fast-clock edge.
start_address	Tied to GMRST. On each game-start pulse, the D register captures

Random subsystem signal	Description
D register CLK	whatever STRT_ADR is at that instant.
start_address D register CLR	Global RESET. Clears the held start address back to 0 on RESET.
start_address output (8 bits)	Held value of the captured random address. Feeds the PD step counter's parallel-load. Holds for the entire game until the next game-start pulse overwrites it.

The randomness comes from the player's reaction time being unpredictable relative to the random counter's spin rate. The counter is moving much faster than the player can press, so whatever value it happens to be on at the moment of the press is effectively unpredictable. Sequential games naturally pick different starting addresses because the counter has continued to spin between games.

How to test it

Open the CircuitVerse file. The tester sees an Event Encoder subcircuit with five toggle inputs (START, PD_FIN, LOSE, ADV, WIN), a CLOCK button (one click is one full pulse — both edges advance the FSM), and a RESET button. The encoder converts whichever event input is asserted into the 3-bit event code the FSM consumes. Mealy outputs go hot before the click; the state register, the latches, the round counter, and the random start-address register all update on the click.

Three short walks below cover every transition the FSM can make. The tester should match the LED pattern after each click against the Expected column. Toggle the event input on before clicking the CLOCK; toggle it back off after the click before setting the next event.

Win path (game start through final-round win)

Step	Event input	Click CLOCK?	Expected result after click
0	(any)	Press RESET	State = IDLE. Q0 dark, Q1 dark, PLAYD dark, Q_LOSE dark, Q_WIN dark, hex count = 0, start_address = 0.
1	Toggle START on	yes	State = PLAY. Q1 lights, PLAYD lights. LDPD and GMRST were lit before click. Hex count stays 0; start_address captures a random value from the spinning counter.
2	Toggle PD_FIN on	yes	State = USER. Q0 also lights, PLAYD drops. LDPM was lit before click. start_address holds.
3	Toggle ADV on	yes	State = PLAY. Q0 drops, PLAYD lights. LDPD and RINC were lit before click. Hex count ticks up by 1. start_address holds (same value, LDPD only re-loads SP, doesn't re-capture random).
4	Toggle PD_FIN on	yes	State = USER. Both Q LEDs lit, PLAYD dark.

Step	Event input	Click CLOCK?	Expected result after click
5	Toggle WIN on	yes	State = IDLE. Both Q LEDs dark, PLAYD dark, Q_WIN lights and stays lit. SWIN was lit before click. Hex count holds. start_address holds.
6	Toggle START on	yes	State = PLAY. Q1 lights, PLAYD lights, Q_WIN clears (GMRST cleared the latch), hex count clears to 0, start_address captures a NEW random value (different from game 1's value because the random counter has kept spinning).

Lose path (game start through wrong-color or timeout loss)

Step	Event input	Click CLOCK?	Expected result after click
0	(any)	Press RESET	State = IDLE. All LEDs dark. Hex count = 0. start_address = 0.
1	Toggle START on	yes	State = PLAY. Q1 and PLAYD light. start_address captures a random value.
2	Toggle PD_FIN on	yes	State = USER. Both Q LEDs lit, PLAYD dark.
3	Toggle LOSE on	yes	State = IDLE. Both Q LEDs dark, Q_LOSE lights and stays lit. SLOSE was lit before click. Hex count holds; start_address holds.
4	Toggle START on	yes	State = PLAY. Q_LOSE clears via GMRST. Hex count clears via GMRST. start_address captures a fresh random value.

Hold paths (state should not advance)

From	Event input	Click CLOCK?	Expected result after click
PLAY	All toggles off	yes (try 2-3x)	State stays at PLAY. Q1 and PLAYD remain lit. No Mealy pulses fire. Hex count and start_address stay.
PLAY	Toggle START on	yes	State stays at PLAY. START is ignored mid-game. start_address holds (no GMRST pulse).
USER	Mid-round correct press (handled by Pattern Matcher; all toggles off, no FSM event	yes	State stays at USER. The Pattern Matcher self-increments externally. Hex count and start_address hold.

From	Event input	Click CLOCK?	Expected result after click
	fires)		
USER	All toggles off	yes (try 2-3x)	State stays at USER.

Reset

From any state, pressing RESET asynchronously returns the state register to IDLE, clears Q_LOSE and Q_WIN, clears the round counter to 0, and clears the start_address register to 0. The FSM does not need a clock pulse for this; RESET overrides everything immediately.

Recovery from BLNK (verified by equation analysis)

BLNK (Q0 = 1, Q1 = 0) is unreachable in normal clocked operation from any of IDLE, PLAY, or USER. If a glitch ever drove the state register to BLNK, the K(Q0) equation includes a Q1' term that fires K(Q0) = 1 whenever Q1 is low. On the next clock edge, Q0 resets to 0 and the FSM lands in IDLE. This recovery is guaranteed for every event code, including no-event.

AI Disclosure

This document was drafted with the help of Claude (Anthropic). Claude assisted with synthesizing the design notes into manual form, formatting tables, and prose drafting. All design decisions, the LogicAid Petrick minimization, the encoding choice, and the verification of the FSM behavior were done by me.