ECE 316: Digital Logic Design

Tokyo Yamanote Line Simulator - Planning Phase

20.08.2026

Rachit Kakkar

Summer 2026

Contents

1. Project Proposal (Part 2)

Our project is a Yamonate Line Simulator, and consists of the following *four modes:*

OFF (Mode 0): We begin at the default stage with the 7 segment display holding the output "OFF". The 16 lights at the bottom of the FPGA board are all turned off. You can press BTNU to begin the game and advance to Mode 1.

Station Selection (Mode 1): A starting station, direction, and ending station on the Yamanote Line are chosen. The text "RIDE TO" plus the name of the randomly chosen ending station is looped on the display. Once the user has read it, they can press BTNU to begin playing. Since this is the same button that is used to go from Mode 0 to Mode 1, a rising edge detector is used to ensure that the user cannot just hold down the button and skip this crucial information stage.

The Ride (Mode 2): The display cycles through each train station based on the direction. Each station name pans to the right once, followed by the next station name. This is the main animation for this project, although there are several others in the other modes. For the station name, we will display the names of stops on the Yamanote line in either the clockwise or counter-clockwise direction (depending on the value of the randomly chosen direction bit). The corresponding Kanji for the station names may be printed on the instruction sheet for Japanese students. The max length of a station name is 16 characters, and most are longer than 5 characters, allowing us to demonstrate non-trivial panning behavior. The 16 lights at the bottom are also set to the station number in binary, simulating riding a train and arriving at progressively higher station numbers. The panning behavior and the transition to each station is done using a slow clock, meeting that part of requirement 8 as well.

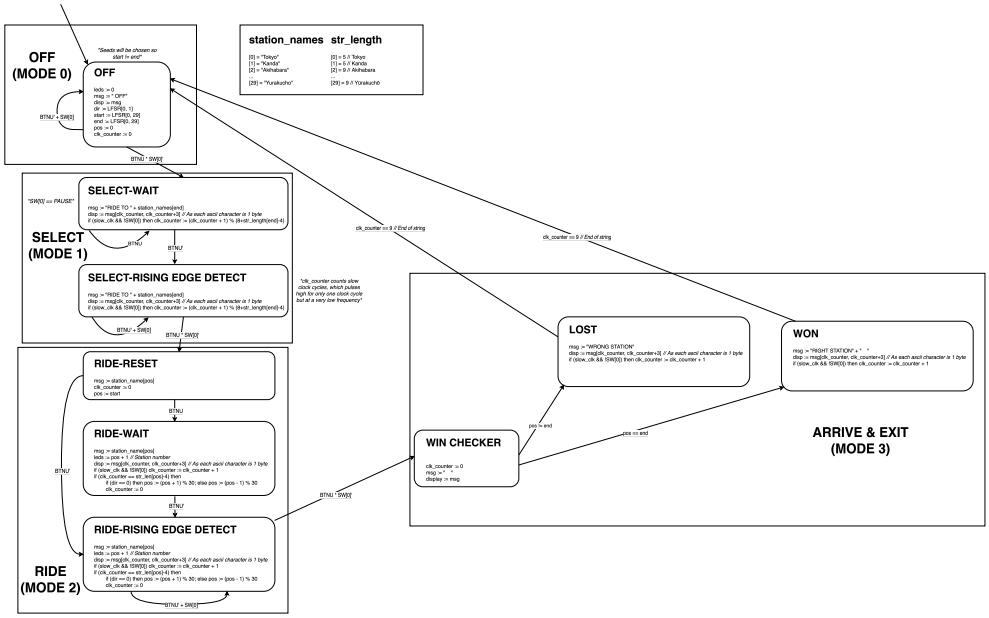
Arrival & Exit (Mode 3): To simulate the IC scan off the train, press BTNU one last time. Again, a rising edge detector is used to ensure the user doesn't just scan instantly by pressing too long when going from Mode 1 to Mode 2.

- If you get off at the correct station: The game ends. After this, the display will print a win message and return to the off state (Stage 0).
- If you get off at the wrong station: The game ends. After this, the display will print a loss message and return to the off state (Stage 0).

System Controls: You can stop at any time by holding the pause switch (SW[0]) high. Once the switch falls low again, the system will resume, meeting the requirement for a level-sensitive switch. The HLSM will remember its current state upon resuming. Together, these controls meet requirement 4 and part of requirement 8.

2. HLSM Design (Part 3)

2.1. HLSM diagram



2.2. Panning position explanation

All panning behavior is controlled by a sliding 4 character window starting from the value of the `clk_counter` up-counter, which counts `slow_clk` cycles that come from the output of a clock divider. However, this count is gated by `!SW[0]`, ensuring that the `clk_counter`, and therefore the *panning position*, only counts up when the pause switch (`SW[0]`) is not held high. Otherwise, the system is paused and the panning animation remains suspended (as `clk_counter` retains its previous value instead of counting up). Moreover, before displaying the next station name or transitioning to a new mode, `clk_counter` is first cleared and set to 0, allowing the next animation to start panning from the beginning of the string. Transitions between modes also depend on `SW[0]`, and the full boolean expression `BTNU * SW[0]'` is used to ensure that these mode transitions occur only on both the rising egde of BTNU and while the pause switch is not asserted.

2.3. Truth table for each output per state

The table below lists the user-visible outputs in each of the four primary modes.

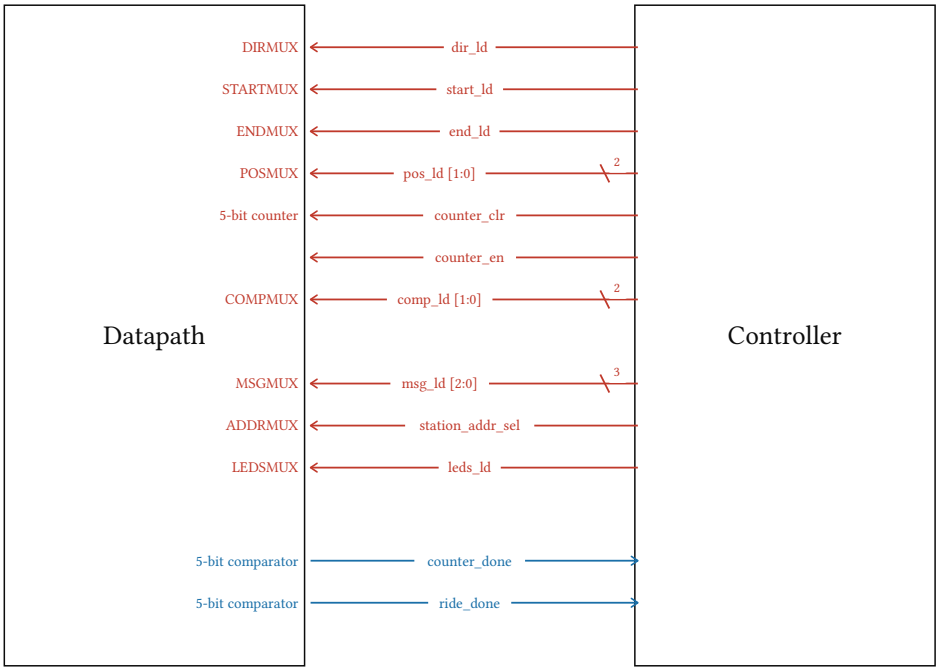
Mode	`leds[4:0]`	7-segment display (`disp`)
Mode 0 — OFF	0	Static: `"OFF "` (no scrolling)
Mode 1 — SELECT	0	Scrolling: `"RIDE TO " + station_names[end]`

Mode 2 — RIDE	`pos + 1` (station number in binary)	Scrolling: `station_name[pos]`, advances to next station when `clk_counter` reaches `str_length[pos] - 4`
Mode 3 — ARRIVE & EXIT (correct stop)	0	Scrolling: `"RIGHT STATION "` + padding
Mode 3 — ARRIVE & EXIT (wrong stop)	0	Scrolling: `"WRONG STATION "` + padding

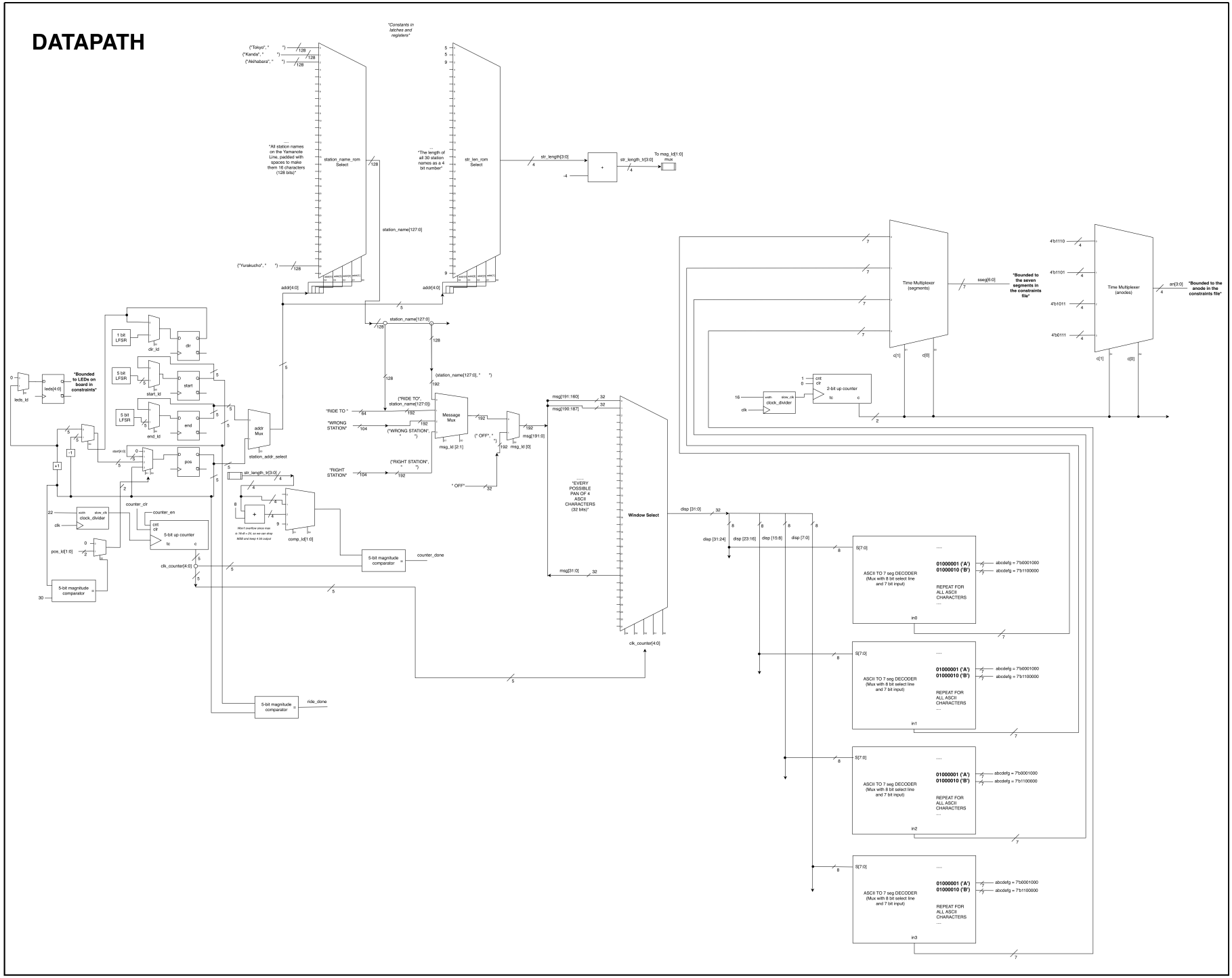
As mentioned, the 7-segment display content scrolls via a 4-character window driven by `clk_counter`; the value shown is the full message string. In all modes, `SW[0]` held high freezes `clk_counter`, pausing the scroll animation in place. Releasing `SW[0]` resumes from exactly where it stopped. `leds[4:0]` in RIDE reflects the station number in binary (`pos + 1`).

3. Datapath and Controller (Part 4)

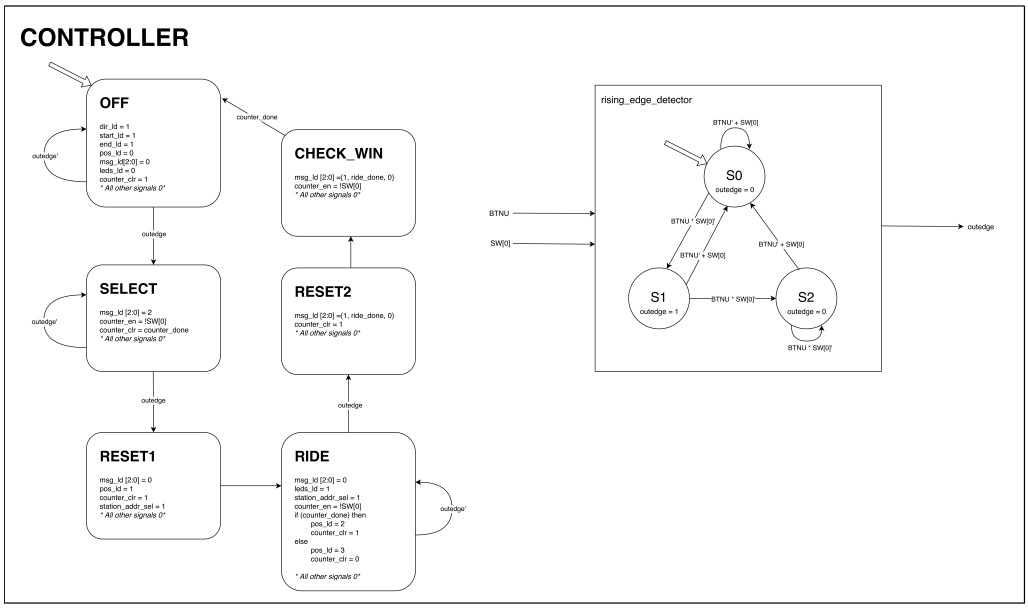
3.1. Top-level block diagram showing the datapath/controller split



3.2. Detailed datapath diagram (entire page)



3.3. Detailed controller FSM diagram



3.4. Signal table

3.4.1. Control Signals (Controller → Datapath)

Signal	Width	Purpose
`dir_ld`	1	Load LFSR output into `dir` register
`start_ld`	1	Load LFSR output into `start` register
`end_ld`	1	Load LFSR output into `end` register
`pos_ld[1:0]`	2	00 = clear · 01 = load `start` · 10 = increment or decrement based on `dir` · 11 = mantain value
`leds_ld`	1	Load `leds[4:0]` with station number (`pos + 1`)
`msg_ld[2:0]`	3	Selects message input (OFF / SELECT / RIDE / WIN / LOSE)
`counter_clr`	1	Synchronously reset `clk_counter` to 0
`counter_en`	1	Enable 5-bit `clk_counter`
`comp_load`	1	Select the value to compare `clk_counter` with for panning calculations
`station_addr_sel`	1	Choose between grabbing the station name of the ending station or position for the `station_name_rom` address

3.4.2. Status Signals (Datapath → Controller)

Signal	Width	Purpose
`counter_done`	1	`clk_counter == value` loaded by `comp_load`; triggers next station, message loop, or mode exit
`ride_done`	1	`pos == end`; used in CHECK_WIN to select win / lose message

4. Module Hierarchy and Port Lists (Part 5)

4.1. Module Hierarchy

`ad_sign_top` is the root and contains exactly two children: `controller` and `datapath`. All behavioral logic lives inside those two modules; the top level only wires them together and exposes board I/O.

ad_sign_top
- controller
  - rising_edge_detector
- datapath
  - clkdiv (×2, width parameterized)
  - lfsr (×3 for dir, start, end, width parameterized)
  - pos_counter
  - clk_counter
  - mag_cmp (×2 for counter_done, ride_done)
  - station_name_rom
  - str_len_rom
  - message_mux
  - window_select
  - ascii_to_7seg (×4, one per each 7 segment)
  - time_mux_sm

4.2. Per-Module Port and Submodule Lists

4.2.1. `ad_sign_top`

Top-level wrapper. Instantiates `controller` and `datapath`, wires named control/status signals between them, while also driving board I/O.

Port	Dir / Width	Description
`clk`	in 1	100 MHz board clock
`rst`	in 1	Synchronous reset (active high)
`BTNU`	in 1	Mode-advance button
`SW[0]`	in 1	Pause switch (level-sensitive)
`seg[6:0]`	out 7	7-segment pattern
`an[3:0]`	out 4	7-segment anode select
`leds[4:0]`	out 5	Station progress indicator LEDs

Submodules instantiated: `controller u_ctrl`, `datapath u_dp`

4.2.2. `controller`

Controller FSM. Reads `outedge` (edge-detected button) and datapath status signals; drives all control signals into the datapath and the one-hot `led` output directly. States: OFF, SELECT, RESET1, RIDE, RESET2, CHECK_WIN.

Port	Dir / Width	Description
`clk`	in 1	System clock
`rst`	in 1	Synchronous reset
`outedge`	in 1	Rising-edge pulse from `rising_edge_detector`
`counter_done`	in 1	If `clk_counter` reached value specified by `comp_load` in datapath
`ride_done`	in 1	If pos == end
`dir_ld`	out 1	Load LFSR into dir register
`start_ld`	out 1	Load LFSR into start register
`end_ld`	out 1	Load LFSR into end register
`pos_ld[1:0]`	out 2	00 = clear · 01 = load start · 10 = increment or decrement · 11 = maintain value
`leds_ld`	out 1	Enable LED register update
`msg_ld[2:0]`	out 3	Message mux select (5 inputs)
`counter_clr`	out 1	Reset `clk_counter` to 0
`counter_en`	out 1	Enable `clk_counter`
`comp_load`	out 1	Select value to compare `clk_counter` with
`station_addr_sel`	out 1	ROM address mux: 0 = end, 1 = pos

Submodules instantiated: `rising_edge_detector u_red`

4.2.3. `rising_edge_detector`

Three-state FSM (S0, S1, S2). Detects a rising edge on BTNU while SW[0] is low. Outputs a single-cycle pulse `outedge`.

Port	Dir / Width	Description
`clk`	in 1	System clock
`BTNU`	in 1	Raw button input
`SW[0]`	in 1	Pause switch
`outedge`	out 1	Single-cycle high pulse on valid rising edge

Submodules: none (leaf module)

4.2.4. `datapath`

Wrapper for all datapath logic. Accepts control signals from the controller; produces display outputs and status signals.

Port	Dir / Width	Description
`clk`	in 1	System clock
`rst`	in 1	Synchronous reset
`dir_ld`	in 1	Load LFSR into dir register
`start_ld`	in 1	Load LFSR into start register
`end_ld`	in 1	Load LFSR into end register
`pos_ld[1:0]`	in 2	00 = clear · 01 = load start · 10 = increment or decrement · 11 = maintain value
`leds_ld`	in 1	LED register load enable
`msg_ld[2:0]`	in 3	Message mux select
`counter_clr`	in 1	Reset `clk_counter`
`counter_en`	in 1	Enable `clk_counter`
`comp_load`	in 1	Latch comparator target value
`station_addr_sel`	in 1	Station ROM address select
`seg[6:0]`	out 7	7-segment cathode (from `time_mux_sm`)
`an[3:0]`	out 4	7-segment anode select (from `time_mux_sm`)
`leds[4:0]`	out 5	Station number `(pos + 1)` in binary
`counter_done`	out 1	`clk_counter` == comparator target

ride_done	out 1	pos == end

Submodules instantiated: clkdiv u_slow, clkdiv u_disp, lfsr u_dir, lfsr u_start, lfsr u_end, pos_counter u_pos, clk_counter u_ctr, mag_cmp u_cmp_ctr, mag_cmp u_cmp_pos, station_name_rom u_name_rom, str_len_rom u_len_rom, message_mux u_msg, window_select u_win, ascii_to_7seg u_seg0, ascii_to_7seg u_seg1, ascii_to_7seg u_seg2, ascii_to_7seg u_seg3, time_mux_sm u_tmux

4.2.5. Leaf Module Port Lists

Module	Port	Dir / W	Description
clkdiv	clk	in 1	System clock
	slow_clk	out 1	Divided clock output
	WIDTH	param	Counter bit width; top bit toggles at $\frac{f_{clk}}{2^{\{WIDTH\}}}$
lfsr	clk	in 1	System clock
	en	in 1	Shift enable
	WIDTH	param	Output bit width
	val[WIDTH-1:0]	out WIDTH	Current LFSR value
pos_counter	clk	in 1	System clock
	slow_clk	in 1	Advance clock
	pos_ld[1:0]	in 2	00 clr · 01 load · 10 inc/dec · 11 hold
	start[4:0]	in 5	Value to load on pos_ld = 01
	dir	in 1	0 = increment, 1 = decrement
	pos[4:0]	out 5	Current station index (mod 30)
clk_counter	clk	in 1	System clock
	slow_clk	in 1	Advance clock
	counter_en	in 1	Count enable
	counter_clr	in 1	Synchronous clear
	clk_ctr[4:0]	out 5	Current count
mag_cmp	A[4:0]	in 5	First operand
	B[4:0]	in 5	Second operand

	`eq`	out 1	A == B
	`lt`	out 1	A < B
	`gt`	out 1	A > B
`station_name_rom`	`addr[4:0]`	in 5	Station index (0–29)
	`name[127:0]`	out 128	16-char ASCII name, space-padded
`str_len_rom`	`addr[4:0]`	in 5	Station index (0–29)
	`len[3:0]`	out 4	Actual character count for this station name
`message_mux`	`msg_ld[2:0]`	in 3	Input select
	`*_bus`	in —	Five string input buses (OFF / SEL / RIDE / WIN / LOSE)
	`msg[191:0]`	out 192	Selected 24-char message bus
`window_select`	`msg[191:0]`	in 192	Full message bus
	`ctr[4:0]`	in 5	Window start index (character offset)
	`disp[31:0]`	out 32	4-char window
`ascii_to_7seg`	`char[7:0]`	in 8	ASCII character code
	`sel[1:0]`	in 2	Digit position (for active-low anode pairing)
	`seg[6:0]`	out 7	Segment pattern (active low)
`time_mux_sm`	`disp_clk`	in 1	Fast refresh clock
	`seg_0..3[6:0]`	in 7×4	Per-digit segment patterns
	`seg[6:0]`	out 7	Muxed segment output to board
	`an[3:0]`	out 4	Active-low anode select