CS4341: Bec-Man System Design

The MetaStable Flip-Flops

Carson Page Husna Chaudhary

Fall 2019

Contents

List of Tables

List of Figures

1 Parts List

The parts list given here are derived from Vivado 2019.1, during the RTL expansion phase of synthesis. If built from discrete logic, these listed components would approximate the real world components necessary to build the system. Following it is an implementation report for a Xilinx Artix-7 FPGA.

Table 1: **RTL Parts List**

Part Name	Count
2 Input, 12 Bit Adder	1
2 Input, 11 Bit Adder	1
2 Input, 10 Bit Adder	8
2 Input, 4 Bit Adder	2
2 Input, 2 Bit Adder	1
12 Bit Register	1
11 Bit Register	1
10 Bit Register	4
8 Bit Register	3
6 Bit Register	2
5 Bit Register	2
4 Bit Register	3
3 Bit Register	2
2 Bit Register	4
1 Bit Register	13
1K Single Port RAM	2
2 Input, 50 Bit, Mux	2
NOTE: This can be replaced with async ROM chip:	
65 Input, 16 Bit, Mux	1
2 Input, 12 Bit, Mux	1
2 Input, 11 Bit, Mux	1
2 Input, 10 Bit, Mux	12
5 Input, 10 Bit, Mux	2
2 Input, 8 Bit, Mux	7
6 Input, 6 Bit, Mux	1
2 Input, 6 Bit, Mux	1
2 Input, 4 Bit, Mux	4
3 Input, 4 Bit, Mux	4
5 Input, 3 Bit, Mux	1
5 Input, 1 Bit, Mux	4
2 Input, 1 Bit, Mux	11
3 Input, 1 Bit, Mux	1

Table 2: **Artix-7 Implementation**

Part Name	Count	Description
BUFG	1	Global clock buffer. Used to insert a signal onto the FPGA's global clock tree.
CARRY4	5	Carry logic between LUTs in different slices. Used to implement high-speed ripple-carry adders in fabric.
LUT1	4	1-input, 1-output lookup table, generic logic function.
LUT2	18	2-input, 1-output lookup table, generic logic function.
LUT3	40	3-input, 1-output lookup table, generic logic function.
LUT4	24	4-input, 1-output lookup table, generic logic function.
LUT5	35	5-input, 1-output lookup table, generic logic function.
LUT6	102	6-input, 1-output lookup table, generic logic function.
MUXF7	14	General-Purpose Multiplexer.
MUXF8	3	General-Purpose Multiplexer.
RAM32X1S	100	Fabric distributed ROM. Used to store data on-masse near computation without Block RAM overhead.
FDRE	121	Synchronous Edge-Triggered D-Flip Flop with Reset Signal.
FDSE	11	Synchronous Edge-Triggered D-Flip Flop with Set Signal.
IBUF	6	Input Buffer. Used to bring signals into the FPGA.
OBUF	26	Output Buffer. Used to bring signals into the FPGA.

2 Module Descriptions and Listing

Each used module is documented here with a high-level description and its function. Detailed port listings follow in the next section.

2.1 Game State Engine (game_state.sv)

The game state is where the "game" proper lives in logic. It is a combination of a sequencing state machine, along with a decision path, that produces the outputs required for the video layer.

The logic is keyed into different states annotated by a *state* enum. This state advanced unconditionally except when initially leaving the "idle" state. The condition for leaving the idle state is a positve edge on the state engine enable. This causes the game to update once per frame, allowing for consistant timing on based on the screen refresh rate.

2.2 Video Timing Generator (vtg.sv)

The Video Timing Generator is a critical component of timing and sequencing both the game state and video engine. Its principal role is for creating the necessary signals to output to a VGA display for video output. It also outputs screen coordinates internally to the rest of the design to allow the video systems to have an easy reference to the currently drawn pixels.

The VTG is also used for sequencing the handover between the game state and the video subsystem. The vertical blanking signal is routed to the enable of the game state, and its inverse to the sprite engines. This allows game state to remain constant while the screen is drawing preventing visual artifacts, and provides for timing based on the screen refresh rate. The horizontal blanking signal is also used to properly sequence the sprite engines in addition to the vblank signal.

2.3 Sprite Generators (becman_sprite.sv and map_sprite.sv)

The sprite generators form the core of the video engine. They are a purely feedforward system, taking information from the VTG and the game state to blit pixels to the screen. Both the Bec-Man and Map sprite engine use a series of ROMs or RAMs to hold the graphics data.

When a sprite engine decides to output a pixel, it emits a valid signal and the color information. This is fed into the video arbiter as a final processing step.

2.4 Map RAM (map_ram.sv)

This hold the current map state. In our current implementation, the write signal is always held low treating it like a ROM, but it supports writes to enable a dynamic update of the map for future implementation of bonuses or dots.

This was broken out as a seperate module to be easily reused in both the game state engine and the map sprite subsystems.

2.5 Video Arbiter (video_arbiter.sv)

The video arbiter is the last component before sending the signal to the display. It decides for each pixel which sprite takes priority and has its color drawn to screen. This is done with a fixed priority based on input port.

This is required as the video subsystem contains no framebuffer in memory, requiring it to generate video in realtime without a compositing step. This requires we select the appropriate signal in realtime as well. Since the valid signals are able to be toggled on a pixel by pixel basis, this allows a crude form of transparancy by allowing a lower priority pixel to come through at that time if a higher priortity pixel is not being shown. If no sprite engine is attempting to write data at that time, the arbiter outputs black to set a global background color.

2.6 TMDS Encoder (tmds_encoder.sv)

NOTE: This module is unused in the design as implemented, but noted here for reference. It does not appear later in the document as such.

The TMDS encoders function is to translate VGA signals into one that can be output using an FPGA's DDR I/O ports to a HDMI display. This allows the internal design to work with simple signals, and only apply the relatively complex TMDS encoding at a final processing stage.

Since the design was not sucessfully put onto an FPGA by project completion, this module was not used in the final test bench.

2.7 Primary Testbench (tb/drawing_test.sv)

This is the primary testbench file that provides for the top level design. This is fed into Verilator to make a working cycle-accurate of the verilog as if it was on a hardware device like an FPGA or ASIC. It additionally breaks out the signals to enable the VGASim component of the Verilator to read the display information.

3 System Listing

Each of the modules Inputs, Outputs, and Notable Registers are listed here. This also satisfies the interface listing requirements as each module describes how it may interconnect with other modules.

3.1 Defined Types

We defined multiple new types to assist in development. These are replicated commonly in the next sections.

Table 3: **Defined Types**

Type Name	Width	Description
s_width_t	clogb2(SCREEN_WIDTH)	Generated type that fits the visible screen width.
s_height_t	clogb2(SCREEN_HEIGHT)	Generated type that fits the visible screen height.
coord_t	s_width_t + s_height_t	XY Coordinate Pair.
rgb_t	[23:0]	Packed struct representing 8:8:8 RGB color.

3.2 Input Listing

Each module has its input(s) name, type, and description listed in a table below. Inputs that are prevelent and share the same semantic meaning are in Table 4.

Table 4: **Common Inputs**

Name	Type	Description
i_clk	logic	Global system clock tree. All logic is driven off of this or a derived clock from a PLL / MMCM module.
i_rst	logic	Active high reset. Module must reset registers and/or outputs to known base state on activation.
i_en	logic	Active high clock enable. Registers part of the data path should only update when activated. Control registers or edge detect registers *may* not be disabled on a low if necessary for proper function.

Table 5: **Video Timing Generator Inputs (vtg)**

Name	Type	Description
ACTIVE_WIDTH	parameter	The width of the visible area of the display.
ACTIVE_HEIGHT	parameter	The height of the visible area of the display.
V_FRONT_PORCH	parameter	Vertical front porch of the timing spec in lines.
V_BACK_PORCH	parameter	Vertical back porch of the timing spec in lines.
V_PULSE	parameter	Length of vertical sync pulse in lines.
V_POL	parameter	Defines if the module outputs a active high/low vertical sync pulse.
H_FRONT_PORCH	parameter	Horizontal front porch of the timing spec in pixels.
H_BACK_PORCH	parameter	Horizontal back porch of the timing spec in pixels.
H_PULSE	parameter	Length of horizontal sync pulse in pixels.
H_POL	parameter	Defines if the module outputs a active high/low horizontal sync pulse.

Table 6: **Game State Engine Inputs (game_state)**

Name	Type	Description
i_joystick	logic [3:0]	Joystick Input, lowest bit represents the left direction. Successive bits represent the next cardinal direction in a counter-clockwise manner. The zero vector represents no input.

Table 7: **Bec-Man Sprite Generator Inputs (becman_sprite)**

Name	Type	Description
i_rotate	logic [2:0]	Dictates the rotation of the becman sprite. This value must be held constant while the enable is active.
i_becman	coord_t	The screen top-left referenced screen coordnates of the becman sprite. This value must be held constant while the enable is active.
i_screen	coord_t	The pixel currently being blited to the screen. This is used to determine if the sprite contains an active pixel at the translated coordinates.

Table 8: **Map Sprite Generator Inputs (map_sprite)**

Name	Type	Description
i_screen	coord_t	The pixel currently being blited to the screen. This is used to determine if the sprite contains an active map element.

Table 9: **Map RAM Inputs (map_ram)**

Name	Type	Description
i_write	logic	Sets if the RAM is in read/write mode.
i_tile_x	logic [5:0]	The X address of the map tile.
i_tile_y	logic [4:0]	The Y address of the map tile.

Table 10: **Video Arbiter Inputs (video_arbiter)**

Name	Type	Description
i_vsync	logic	Passthrough input for vsync to keep it in sync with the muxed video signal.
i_hsync	logic	Passthrough input for hsync to keep it in sync with the muxed video signal.
i_req	logic [1:0]	Arbiter request input, MSB takes priority. A zero vector indicates to draw the default color.
i_vport	rgb_t [1:0]	Array of input video ports. These are muxed depedent on the decided input from the arbiter.

3.3 Output Listing

Each module has its output(s) name, type, and description listed in a table below.

Table 11: **Video Timing Generator Outputs (vtg)**

Name	Type	Description
o_hsync	logic	Horizontal sync pulse output.
o_vsync	logic	Vertical sync pulse output.
o_hblank	logic	Active high when the display is in the horizontal blanking period.
o_vblank	logic	Active high when the display is in the vertical blanking period.
o_blank	logic	*o_hblank* || *o_vblank*
o_screen	coord_t	The current XY pixel being sent to the display, in relation to the sync pulses.

Table 12: **Game State Engine Outputs (game_state)**

Name	Type	Description
o_becman	coord_t	Where the Bec-Man sprite should be drawn on the display, as derived from the internal state.
o_becman_dir	logic [2:0]	The direction of the Bec-Man sprite should be rotated to, as derived from the internal state.

Table 13: **Bec-Man Sprite Generator Outputs (becman_sprite)**

Name	Type	Description
o_valid	logic	Active high when the video system should blit pixels from this sprite.
o_color	rgb_t	The color to display when *o_valid* is high.

Table 14: **Map Sprite Generator Outputs (map_sprite)**

Name	Type	Description
o_valid	logic	Active high when the video system should blit pixels from this sprite.
o_color	rgb_t	The color to display when *o_valid* is high.

Table 15: **Map RAM Outputs (map_ram)**

Name	Type	Description
o_tile_value	logic	Active high when a tile is present at the input coordinates.

Table 16: **Video Arbiter Outputs (video_arbiter)**

Name	Type	Description
o_vport	rgb_t	The selected video signal based on the input request and priority level.
o_hsync	logic	Passthrough of the input hsync signal for timing purposes.
o_vsync	logic	Passthrough of the input vsync signal for timing purposes.

3.4 Register Listing

Each module has its registers wires(s) name, type, and description listed in a table below, where applicable.

Table 17: **Video Timing Generator Interfacing (vtg)**

Name	Type	Description
r_pix_cnt	logic [clogb2(H_TOTAL-1):0]	Master horizontal count state.
r_line_cnt	logic [clogb2(V_TOTAL-1):0]	Master vertical count state.

Table 18: **Game State Engine Interfacing (game_state)**

Name	Type	Description
r_en_edge	logic [1:0]	Register used to detect the enable edge on *vblank* to advance states.
r_joystick	logic [1:0]	Latches input during state computation.
will_collide	logic	Stores weather becman will collide this frame.
r_next_pos	coord_t	Stores becman's position to be drawn next frame during computation.
r_next_dir	logic [2:0]	Stores the computed rotation for the becman sprite during computation.
state	state_t	Keeps track of the current state of the micro-sequencer FSM.

Table 19: **Bec-Man Sprite Generator Interfacing (becman_sprite)**

Name	Type	Description
r_pix_count	logic [3:0]	Sprite horizontal count state.
r_line_count	logic [3:0]	Sprite vertical count state.
x/yvalid	logic	Determines weather sprite should enable output.

3.5 Mode Listing

The system contains several modal systems listed here.

Top Level

The primary system has two modes, state update and rendering. These are controlled by the vtg *o_vblank*, *o_hblank*, and *o_blank* signals in a variety of ways. In general, it works as follows:

Table 20: **Top Level Drawing Modes**

vblank	Description
Logic High	System is in state update mode. The game state engine advances through a single cycle of the microsequencer. Conversely, video output is disabled as we are in the blanking period.
Logic Low	System is in drawing mode. The state engine is halted, and the enables come alive for the sprite engines. These are further gated by the *o_hblank* signals to maintain horizontal drawing accuracy.

Map RAM

The Map RAM can in theory be in two modes, read or write as controlled by *i_write*. However, this is currently hard-wired to read in the current implementation.

4 Circuit Diagrams

The various circuit and state diagrams are included here on seperate pages.
NOTE: Vivado breaks up packed structs when generating diagrams, so a wire of type rgb_t will be broken into three wires for the diagram, labled each with an R, G, and B component. In addition, Vivado will postfix registers with "_reg".

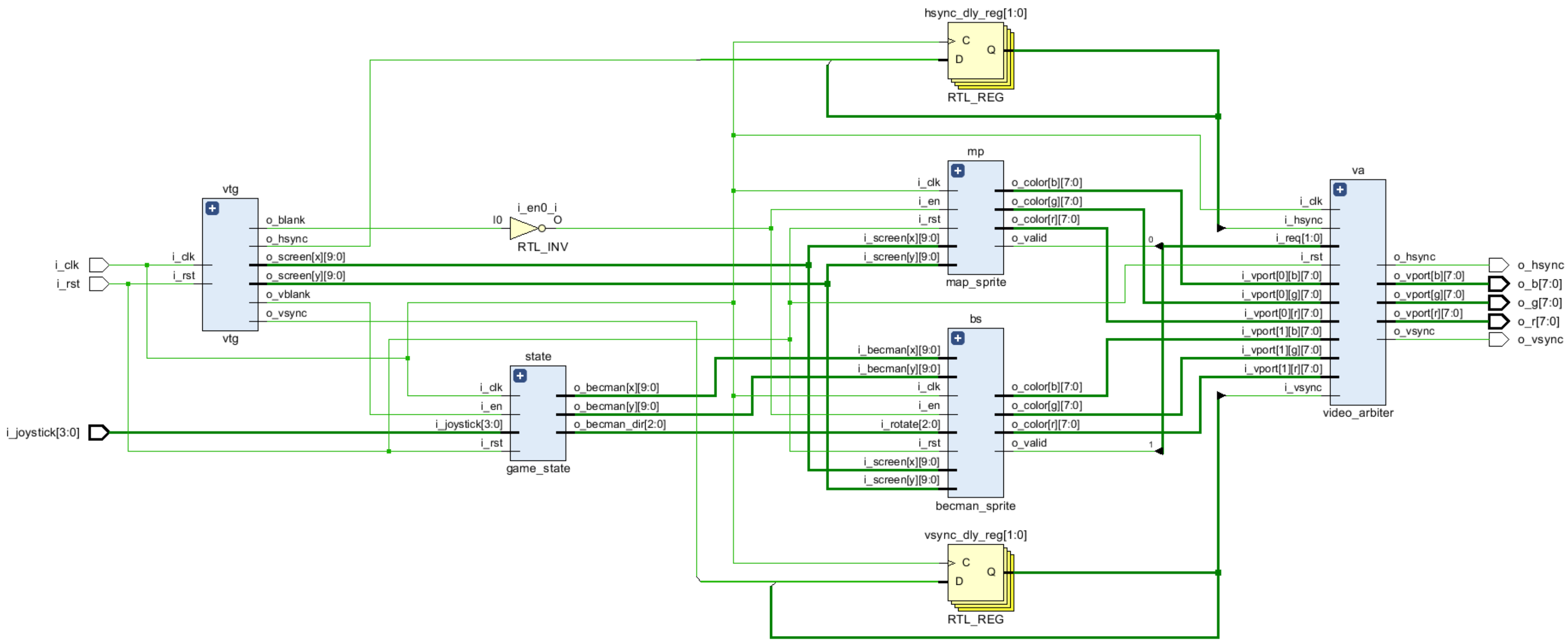

Figure 1: Top Level Block Diagram of the Bec-Man System.

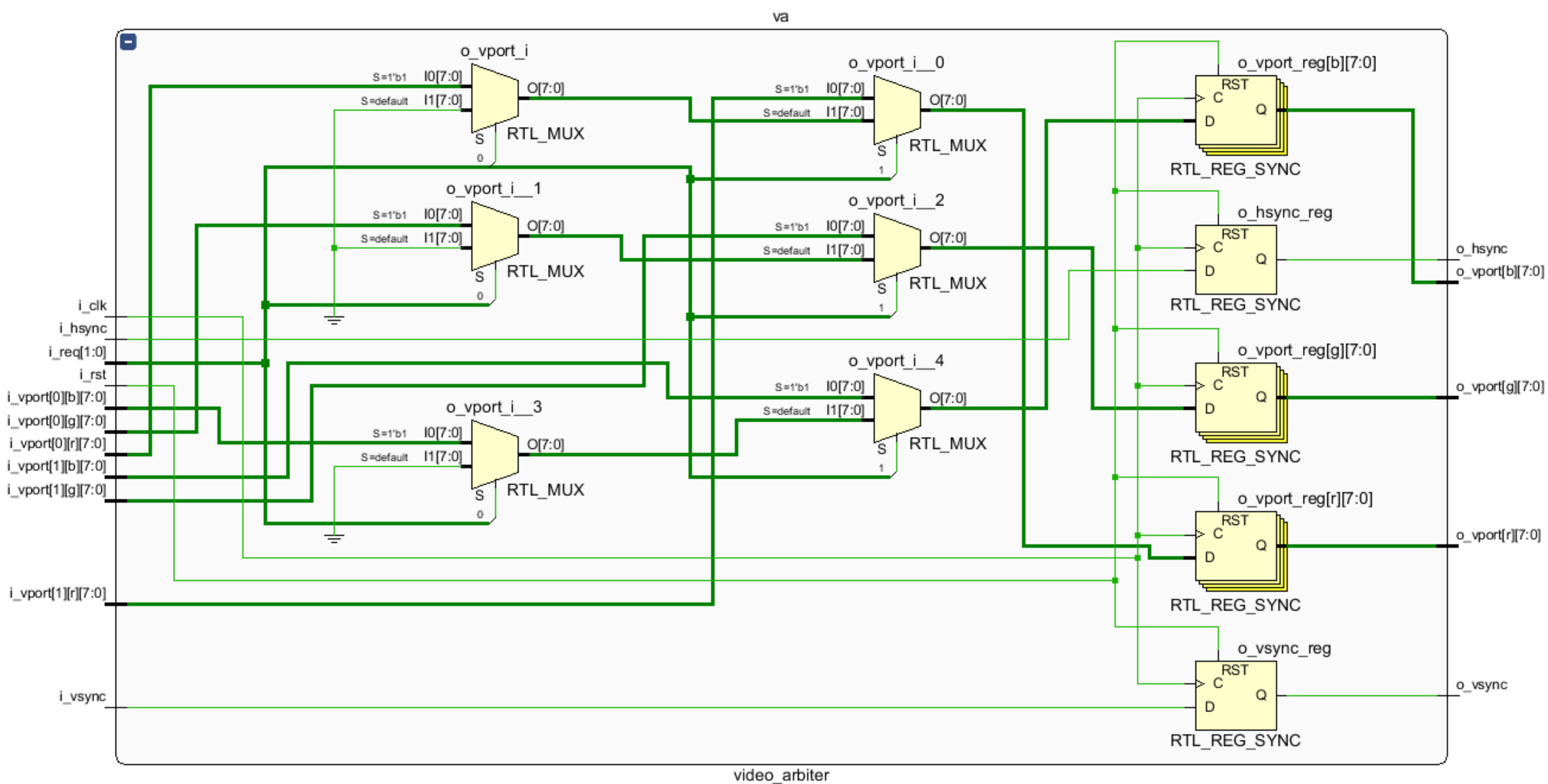

Figure 2: Video Arbiter

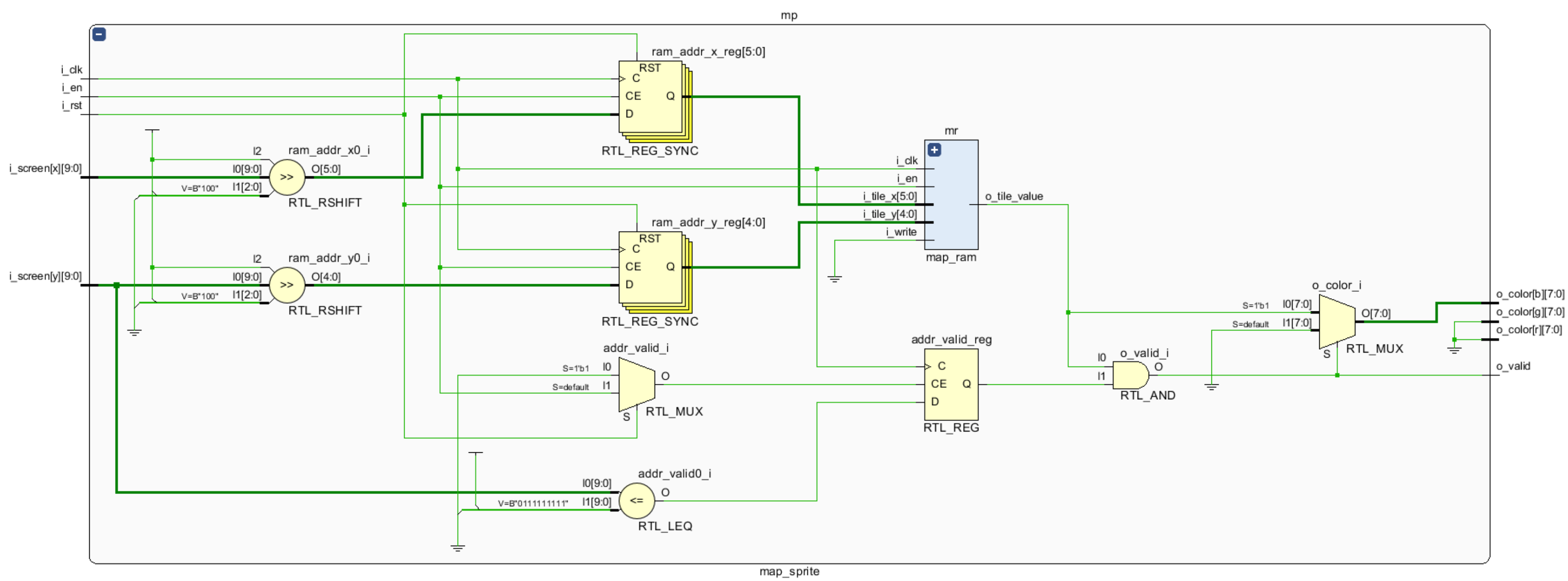

Figure 3: Map Sprite Engine

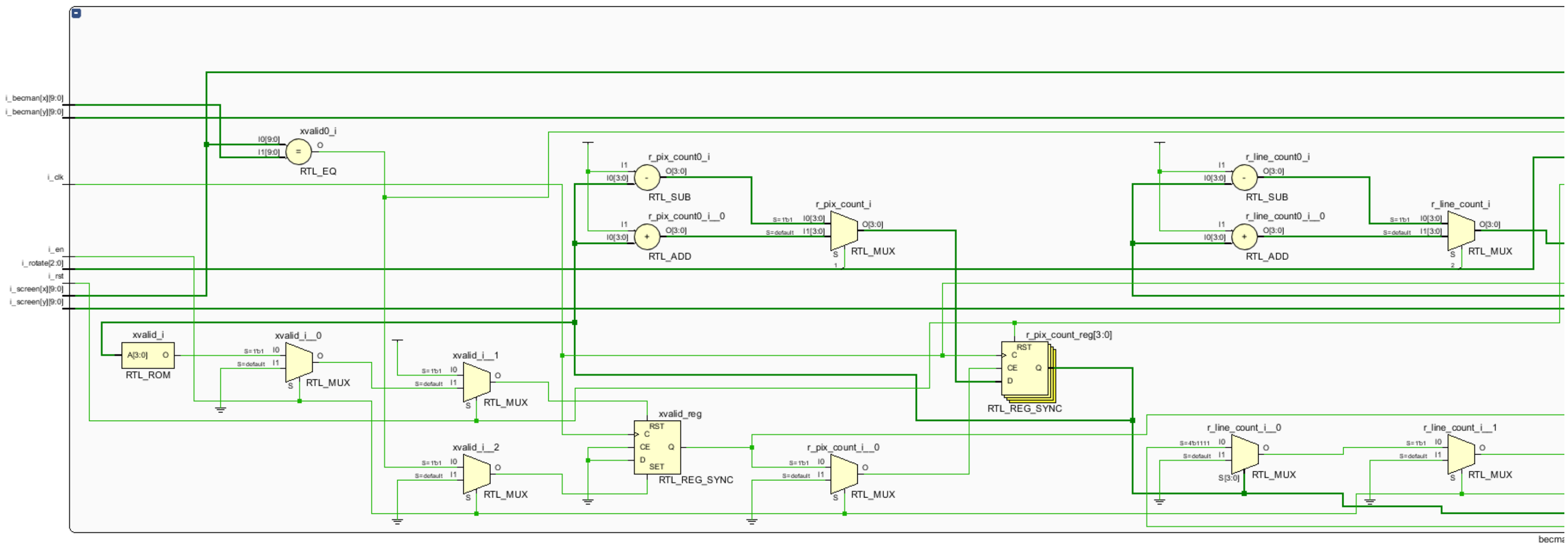

Figure 4: Becman Sprite Engine

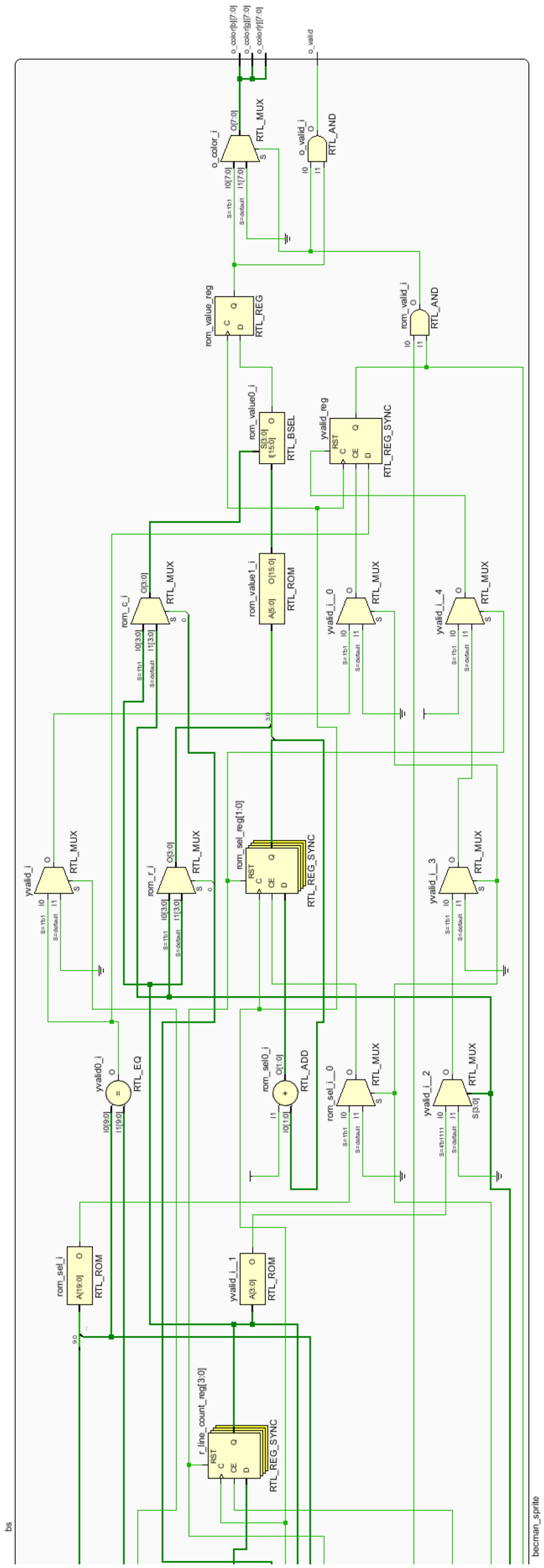

Figure 5: Becman Sprite Engine Cont.

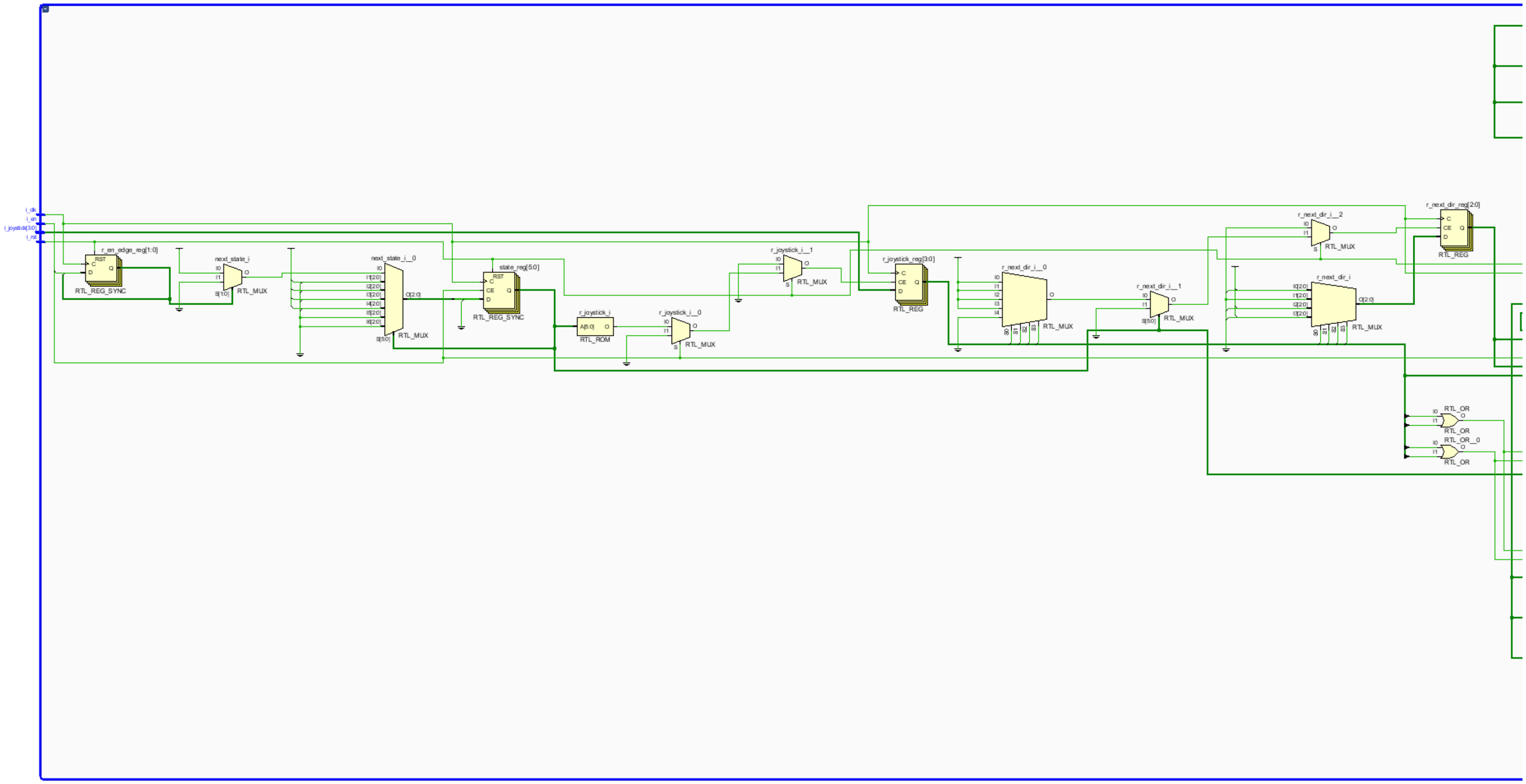

Figure 6: Game State Engine

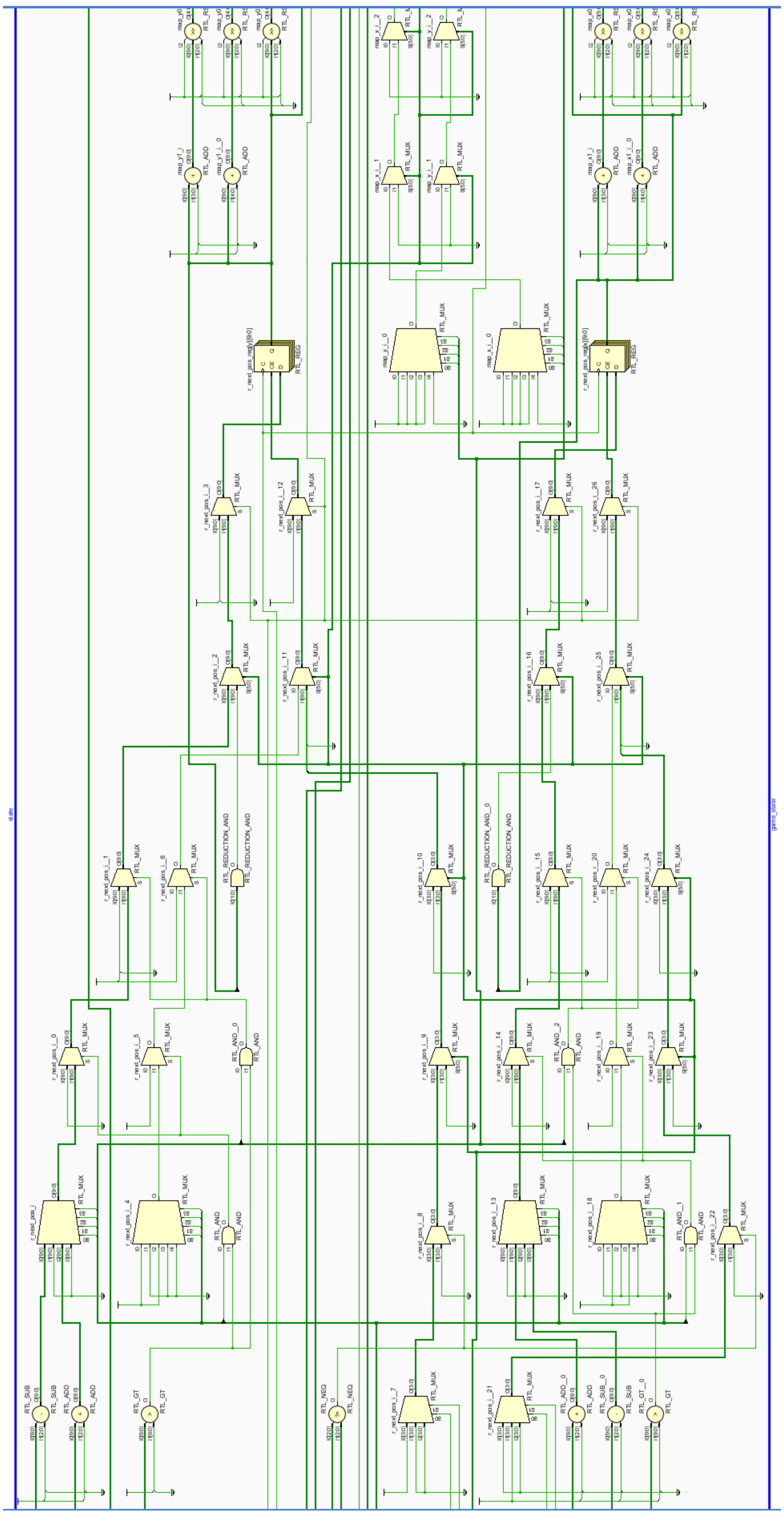

Figure 7: Game State Engine Cont.

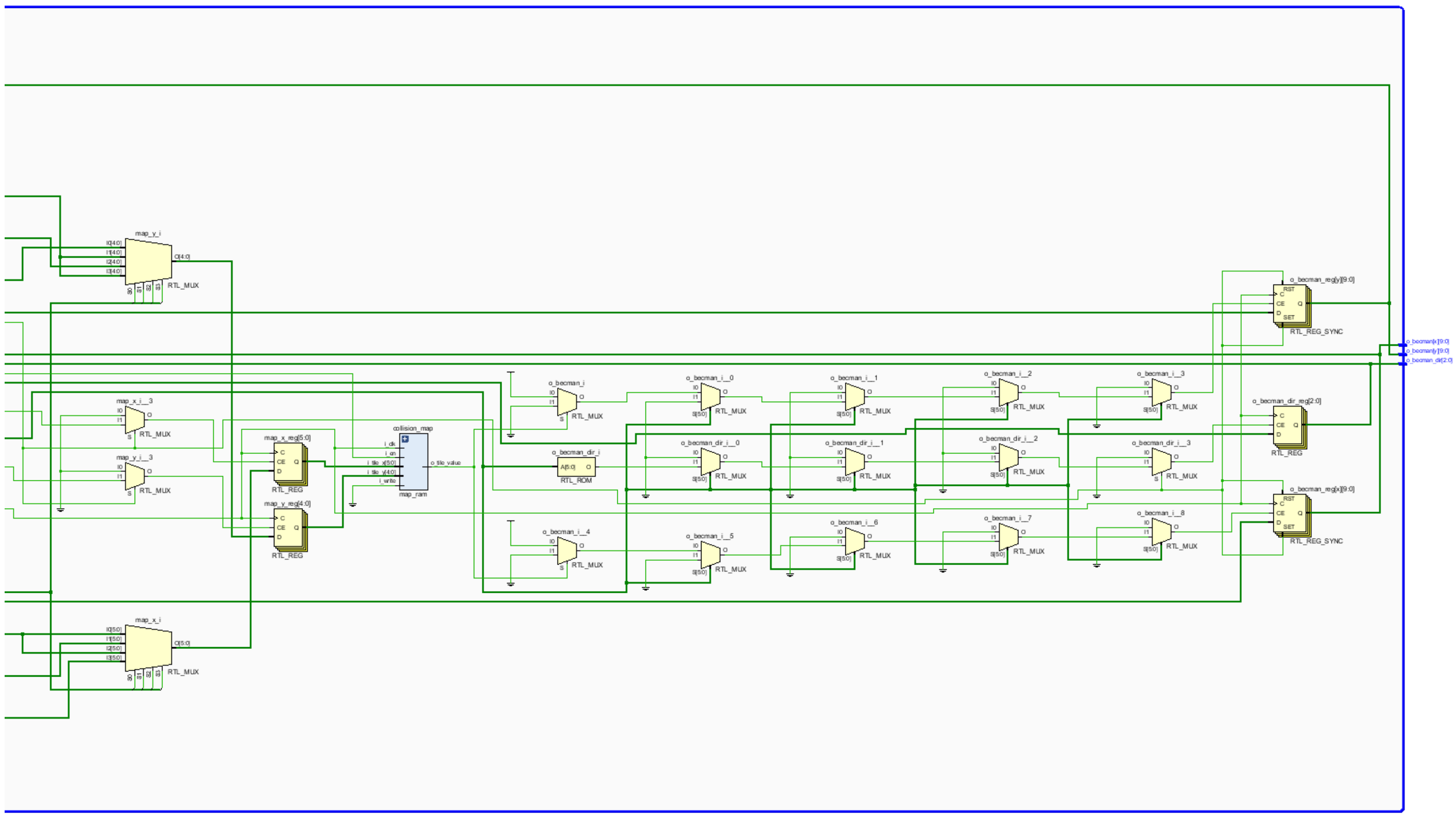

Figure 8: Game State Engine Cont.

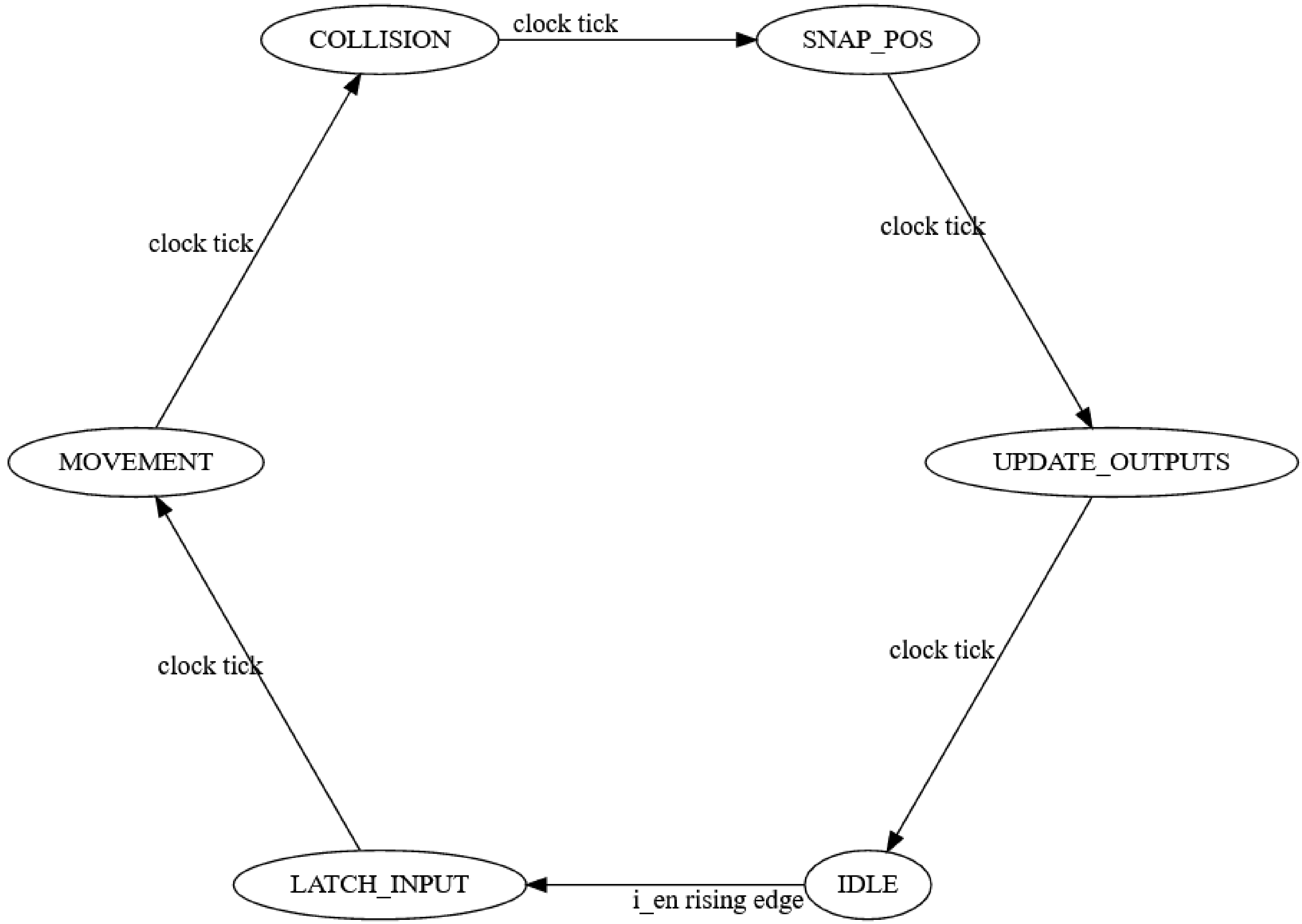

Figure 9: Sequencer State Machine