



**Oregon State**  
**University**

OREGON STATE UNIVERSITY

ECE 271

## **Final Design Project**

*Benjamin Geyer*  
*Christien Hotchkiss*  
*Phi Luu*

November 30<sup>th</sup>, 2018

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Project Goals .....	3
<b>2</b>	<b>High-Level Description</b>	<b>3</b>
2.1	Top-Level Hardware Diagram .....	3
2.2	Top-Level Hardware Diagram .....	3
<b>3</b>	<b>Controllers</b>	<b>3</b>
3.1	PS/2 Keyboard .....	3
3.2	RC5 IR.....	3
3.3	NES Controller.....	3
<b>4</b>	<b>HDL Modules</b>	<b>3</b>
4.1	PS/2 Modules .....	3
4.1.1	PS/2 Decoder .....	3
4.2	RC5 IR Modules .....	3
4.2.1	RC5 IR Decoder .....	3
4.2.2	RC5 IR Convert Pulse.....	3
4.2.3	RC5 IR Read Word .....	3
4.3	NES Modules .....	3
4.3.1	NES Decoder .....	3
4.4	General Modules Used in Each Design .....	3
4.4.1	Clock Frequency Module .....	3
4.4.2	Box Controller Module .....	3
4.4.3	XY Counter Module .....	3
4.4.4	Drawer Module .....	3
<b>5</b>	<b>Putting It All Together</b>	<b>3</b>
<b>6</b>	<b>Above and Beyond</b>	<b>3</b>
<b>7</b>	<b>Appendix</b>	<b>3</b>
7.1	SystemVerilog Source Code.....	3
7.1.1	Top Level.....	3
7.1.2	PS/2 Top Level.....	5
7.1.3	PS/2 Controller .....	6
7.1.4	RC5 IR Top Level .....	8
7.1.5	RC5 IR Controller .....	10
7.1.6	NES Top Level .....	15
7.1.7	NES Controller .....	17
7.1.8	Box Controller .....	20
7.1.9	Half Clock.....	21
7.1.10	XY Counter .....	22
7.1.11	Drawer .....	23



# 1 Introduction

## 1.1 Project Goals

# 2 High-Level Description

## 2.1 Top-Level Hardware Diagram

## 2.2 Top-Level Hardware Diagram

# 3 Controllers

## 3.1 PS/2 Keyboard

## 3.2 RC5 IR

## 3.3 NES Controller

# 4 HDL Modules

## 4.1 PS/2 Modules

### 4.1.1 PS/2 Decoder

## 4.2 RC5 IR Modules

### 4.2.1 RC5 IR Decoder

### 4.2.2 RC5 IR Convert Pulse

### 4.2.3 RC5 IR Read Word

## 4.3 NES Modules

### 4.3.1 NES Decoder

## 4.4 General Modules Used in Each Design

### 4.4.1 Clock Frequency Module

### 4.4.2 Box Controller Module

### 4.4.3 XY Counter Module

### 4.4.4 Drawer Module

# 5 Putting It All Together

# 6 Above and Beyond

# 7 Appendix

## 7.1 SystemVerilog Source Code

### 7.1.1 Top Level

```

3     input logic[1:0] dir_x, dir_y,
4     output logic vs, hs,
5     output logic[3:0] r, g, b
6 );
7
8 logic[9:0] xpos, ypos;
9 logic slowclk;
10
11 logic[3:0] box_r, box_g, box_b;
12 logic[9:0] box_x, box_y, box_size;
13
14 half_clock hc (
15     .oldclk(clk),
16     .newclk(slowclk)
17 );
18
19 XYCounter c (
20     .clk(slowclk),
21     .vs(vs),
22     .hs(hs),
23     .x(xpos),
24     .y(ypos)
25 );
26
27 box_controller box (
28     .vs(vs),
29     .dir_x(dir_x),
30     .dir_y(dir_y),
31     .x(xpos),
32     .y(ypos),
33     .box_r(box_r),
34     .box_g(box_g),
35     .box_b(box_b),
36     .box_x(box_x),
37     .box_y(box_y),
38     .box_size(box_size)
39 );
40
41 Drawer d (
42     .clk(slowclk),
43     .box_r(box_r),
44     .box_g(box_g),
45     .box_b(box_b),
46     .x(xpos),
47     .y(ypos),
48     .box_x(box_x),
49     .box_y(box_y),

```

```

50     .box_size(box_size),
51     .r(r),
52     .g(g),
53     .b(b)
54 );
55
56 endmodule

```

---

## 7.1.2 PS/2 Top Level

---

```

1  module ps2_top_level(
2      input logic clk, psClk, data,
3      output logic vs, hs,
4      output logic[3:0] r, g, b
5  );
6
7  // Track x and y and slow clock
8  logic[9:0] xpos, ypos;
9  logic slowclk;
10
11 // Get direction input
12 logic[1:0] dir_x, dir_y;
13
14 // Track box info
15 logic[3:0] box_r, box_g, box_b;
16 logic[9:0] box_x, box_y, box_size;
17
18 // Slow clock
19 half_clock hc (
20     .oldclk(clk),
21     .newclk(slowclk)
22 );
23
24 // Generate position and vs/hs signals
25 XYCounter c (
26     .clk(slowclk),
27     .vs(vs),
28     .hs(hs),
29     .x(xpos),
30     .y(ypos)
31 );
32
33 // Get keyboard input for direction
34 ps2 P (

```

```

35     .clk(psClk),
36     .data(data),
37     .dir_x(dir_x),
38     .dir_y(dir_y)
39 );
40
41 // Move box
42 box_controller box (
43     .vs(vs),
44     .dir_x(dir_x),
45     .dir_y(dir_y),
46     .box_r(box_r),
47     .box_g(box_g),
48     .box_b(box_b),
49     .box_x(box_x),
50     .box_y(box_y),
51     .box_size(box_size)
52 );
53
54 // Draw box
55 Drawer d (
56     .clk(slowclk),
57     .box_r(box_r),
58     .box_g(box_g),
59     .box_b(box_b),
60     .x(xpos),
61     .y(ypos),
62     .box_x(box_x),
63     .box_y(box_y),
64     .box_size(box_size),
65     .r(r),
66     .g(g),
67     .b(b)
68 );
69
70 endmodule

```

---

### 7.1.3 PS/2 Controller

---

```

1 module ps2 #(parameter
2     UP = 8'h1D, DOWN = 8'h1B, LEFT = 8'h1C, RIGHT = 8'h23 // PS2 key code of W,
   ↪ S, A, and D, repsectively
3 ) (
4     input logic clk, data,

```

```

5     output logic[1:0] dir_x, dir_y
6 );
7
8 // Track data input
9 logic[7:0] bits = 8'b0;
10
11 // Track count of bits coming in and number of ones
12 logic[3:0] count = 0;
13 logic[3:0] even_odd = 0;
14
15 // Hold output values
16 // 0 = up/left
17 // 1 = down/right
18 // 2 = stop
19 logic[1:0] x = 2;
20 logic[1:0] y = 2;
21
22 // Track valid input and whether past input was F0 (indicating key release)
23 logic valid = 0;
24 logic fo = 0;
25
26 always_ff @(negedge clk)
27     begin
28         if (count == 10) // Ending bit
29             begin
30                 if (valid)
31                     // Translate directions to movement
32                     begin
33                         // not released && is not moving && to move up -> move up
34                         if (~fo && y == 2 && bits == UP) y <= 0;
35                         // released && is moving up && to move up -> stop
36                         else if (fo && y == 0 && bits == UP) y <= 2;
37                         // not released && is not moving && to move down -> move
38                         ↪ down
39                         else if (~fo && y == 2 && bits == DOWN) y <= 1;
40                         // released && is moving down && to move down -> stop
41                         else if (fo && y == 1 && bits == DOWN) y <= 2;
42
43                         // not released && is not moving && to move left -> move
44                         ↪ left
45                         if (~fo && x == 2 && bits == LEFT) x <= 0;
46                         // released && is moving left && to move left -> stop
47                         else if (fo && x == 0 && bits == LEFT) x <= 2;
48                         // not released && is not moving && to move right -> move
49                         ↪ right
50                         else if (~fo && x == 2 && bits == RIGHT) x <= 1;
51                         // released && is moving right && to move right -> stop

```



```

49         else if (fo && x == 1 && bits == RIGHT) x <= 2;
50
51         // Get code for key release
52         if (bits == 8'hF0) fo <= 1;
53         else fo <= 0;
54     end
55
56     even_odd <= 0;
57     count <= 0;
58     bits <= 8'd0;
59     valid <= 0;
60 end
61 else if (count == 9) // Check parity bit
62 begin
63     count <= count + 1;
64
65     // Check validity based on number of ones and parity bit (odd
66     ⇨ total)
67     valid <= even_odd[0] == ~data;
68 end
69 else if (count == 0) // Start bit -- do nothing but advance count
70 count <= count + 1;
71 else
72 begin
73     // Track count and number of odd
74     even_odd <= even_odd + data;
75     count <= count + 1;
76
77     // Load data
78     bits <= (bits >> 1) + (data * 128);
79 end
80 end
81 // Set output
82 assign dir_x = x;
83 assign dir_y = y;
84
85 endmodule

```

---

#### 7.1.4 RC5 IR Top Level

---

```

1 module ir_top_level(
2     input logic clk, data,
3     output logic vs, hs,

```

```

4      output logic[3:0] r, g, b
5  );
6
7  // Track x and y and slow clock
8  logic[9:0] xpos, ypos;
9  logic slowclk;
10
11 // Get direction input
12 logic[1:0] dir_x, dir_y;
13
14 // Track box info
15 logic[3:0] box_r, box_g, box_b;
16 logic[9:0] box_x, box_y, box_size;
17
18 // Slow clock
19 half_clock hc (
20     .oldclk(clk),
21     .newclk(slowclk)
22 );
23
24 // Generate position and vs/hs signals
25 XYCounter c (
26     .clk(slowclk),
27     .vs(vs),
28     .hs(hs),
29     .x(xpos),
30     .y(ypos)
31 );
32
33 // Get IR input for direction
34 ir I (
35     .clk(clk),
36     .data(data),
37     .dir_x(dir_x),
38     .dir_y(dir_y)
39 );
40
41 // Move box
42 box_controller box (
43     .vs(vs),
44     .dir_x(dir_x),
45     .dir_y(dir_y),
46     .box_r(box_r),
47     .box_g(box_g),
48     .box_b(box_b),
49     .box_x(box_x),
50     .box_y(box_y),

```

```

51     .box_size(box_size)
52 );
53
54 // Draw box
55 Drawer d (
56     .clk(slowclk),
57     .box_r(box_r),
58     .box_g(box_g),
59     .box_b(box_b),
60     .x(xpos),
61     .y(ypos),
62     .box_x(box_x),
63     .box_y(box_y),
64     .box_size(box_size),
65     .r(r),
66     .g(g),
67     .b(b)
68 );
69
70 endmodule

```

---

### 7.1.5 RC5 IR Controller

---

```

1  // IR module for RC5 code
2  module ir #(parameter
3      UP = 2, DOWN = 8, LEFT = 4, RIGHT = 6
4  )(
5      input logic data, clk,
6      output logic[1:0] dir_x, dir_y
7  );
8
9  // Count for slowing clock and new clock
10 logic[10:0] clkCount = 0;
11 logic newclk = 0;
12
13 // Complete data word and its update status
14 logic[5:0] dataWord = 0;
15 logic newDataWord = 0;
16
17 // Internal x/y movement
18 logic[1:0] x = 2;
19 logic[1:0] y = 2;
20
21 // Used for handling pulses

```

```

22  logic pusleVal = 0;
23  logic newPulseVal = 0;
24  logic waitForRepeat = 0;
25
26  // Slow clock from 50MHz to 36KHz as is used for RC5
27  always_ff @(posedge clk)
28      if (clkCount < 1388)
29          clkCount <= clkCount + 1;
30      else
31          begin
32              clkCount <= 0;
33              newclk = ~newclk;
34          end
35
36  // Converts from encoded pulse to binary
37  irConvertPulse cp(
38      .data(data),
39      .clk(newclk),
40      .val(pusleVal),
41      .newval(newPulseVal),
42      .waitForRepeat(waitForRepeat)
43  );
44
45  // Converts from encoded binary to data word
46  irReadWord rw(
47      .val(pulseVal),
48      .clk(newPulseVal),
49      .waitForRepeat(waitForRepeat),
50      .data(dataWord),
51      .finished(newDataWord)
52  );
53
54  // Translate from data word to movement value
55  always_ff @(posedge newDataWord)
56      begin
57          if (dataWord == UP) y <= 0;
58          else if (dataWord == DOWN) y <= 1;
59          else y <= 2;
60
61          if (dataWord == LEFT) x <= 0;
62          else if (dataWord == RIGHT) x <= 1;
63          else x <= 2;
64      end
65
66  assign dir_x = x;
67  assign dir_y = y;
68

```

```

69  endmodule
70
71
72  module irReadWord(
73      input logic val, clk, waitForRepeat,
74      output logic[5:0] data,
75      output logic finished
76  );
77
78      // Stores output and counts bits
79      logic[5:0] out = 0;
80      logic[4:0] count = 0;
81
82      // Tracks previous toggle -- not used in this implementation
83      // but necessary for certain controls so it will be tracked
84      // anyway
85      logic prevToggle = 1;
86
87      always_ff @(posedge clk, negedge waitForRepeat)
88          begin
89              if (~waitForRepeat) // Reset if time limit for repeating signal is up
90                  begin
91                      out <= 0;
92                      count <= 0;
93                      finished <= 1;
94                  end
95              else if (count == 13) // Reset count, store final value, and signal
96                  ↪ finished transmission
97                  begin
98                      out <= (out << 1) + val;
99                      finished <= 1;
100                     count <= 0;
101                 end
102             else if (count == 2 && val != prevToggle) // Set toggle and reset value
103                 ↪ if different button press
104                 begin
105                     prevToggle <= val;
106                     out <= 0;
107                     count <= count + 1;
108                     finished <= 0;
109                 end
110             else if (count > 7) // If in data word, add to output
111                 begin
112                     out <= (out << 1) + val;
113                     count <= count + 1;
114                     finished <= 0;
115                 end
116             end
117

```

```

114         else
115             begin
116                 count <= count + 1;
117                 finished <= 0;
118             end
119         end
120
121     assign data = out;
122
123 endmodule
124
125
126 module irConvertPulse(
127     input logic data, clk,
128     output logic val, newval, waitForRepeat
129 );
130
131 // Tracks two sections with potential to have pulses
132 logic[5:0] dataCounterOne = 0;
133 logic[5:0] dataCounterTwo = 0;
134
135 // Counts time
136 logic[5:0] timeCounter = 0;
137
138 // Counts time from
139 logic[6:0] waitCounter = 78;
140 logic idle = 1;
141 logic out = 0;
142
143 logic tcReset = 0;
144 logic dcReset = 0;
145 logic dcResetSuccess = 0;
146
147 always_ff @(posedge data)
148     begin
149         if (idle) // Reset time counter if start of a transmission
150             begin
151                 dataCounterOne <= 0;
152                 dataCounterTwo <= 1;
153                 tcReset <= 1;
154             end
155         else if (dcReset)
156             begin
157                 if (timeCounter > 31 && dataCounterTwo < 32)
158                     begin
159                         dataCounterOne <= 0;
160                         dataCounterTwo <= 1;

```

```

161         end
162     else if (dataCounterOne < 32)
163     begin
164         dataCounterOne <= 1;
165         dataCounterTwo <= 0;
166     end
167
168     dcResetSuccess <= 1;
169 end
170 else // Increment data
171 begin
172     if (timeCounter > 31 && dataCounterTwo < 32) dataCounterTwo <=
        ↪ dataCounterTwo + 1;
173     else if (dataCounterOne < 32) dataCounterOne <= dataCounterOne +
        ↪ 1;
174
175     tcReset <= 0;
176     dcResetSuccess <= 0;
177 end
178 end
179
180 always_ff @(posedge clk)
181 begin
182     if (tcReset)
183     begin
184         timeCounter <= 33;
185         waitCounter <= 0;
186         idle <= 0;
187     end
188     else if (timeCounter == 63) // Set output and reset
189     begin
190         // Increment counter if still in window for additional
191         if (idle && dataCounterOne == 0 && dataCounterTwo == 0 &&
            ↪ waitCounter < 78)
192             waitCounter <= waitCounter + 1;
193
194         out <= dataCounterOne == 0 && dataCounterTwo == 32 && ~dcReset;
195         newval <= (dataCounterOne == 32 || dataCounterTwo == 32) &&
            ↪ ~dcReset;
196         idle <= (dataCounterOne == 0 && dataCounterTwo == 0) || ~dcReset;
197         dcReset <= 1;
198         timeCounter <= 0;
199     end
200     else // Increment data
201     begin
202         timeCounter <= timeCounter + 1;
203         newval <= 0;

```

```

204
205         if (dcResetSuccess)
206             dcReset <= 0;
207         end
208     end
209
210 assign val = out;
211
212 // Do not wait if outside of continued transmission window
213 assign waitForRepeat = waitCounter < 78;
214
215 endmodule

```

---

### 7.1.6 NES Top Level

---

```

1 module nes_top_level(
2     input logic clk, data,
3     output logic vs, hs, latch_out, clk_out,
4     output logic[3:0] r, g, b
5 );
6
7 // Track x and y and slow clock
8 logic[9:0] xpos, ypos;
9 logic slowclk;
10
11 // Get direction input
12 logic[1:0] dir_x, dir_y;
13
14 // Track box info
15 logic[3:0] box_r, box_g, box_b;
16 logic[9:0] box_x, box_y, box_size;
17
18 // Slow clock
19 half_clock hc (
20     .oldclk(clk),
21     .newclk(slowclk)
22 );
23
24 // Generate position and vs/hs signals
25 XYCounter c (
26     .clk(slowclk),
27     .vs(vs),
28     .hs(hs),
29     .x(xpos),

```



```

30     .y(ypos)
31 );
32
33 // Get nes input for direction
34 nes N (
35     .clk(clk),
36     .data(data),
37     .dir_x(dir_x),
38     .dir_y(dir_y),
39     .latch_out(latch_out),
40     .clk_out(clk_out)
41 );
42
43 // Move box
44 box_controller box (
45     .vs(vs),
46     .dir_x(dir_x),
47     .dir_y(dir_y),
48     .box_r(box_r),
49     .box_g(box_g),
50     .box_b(box_b),
51     .box_x(box_x),
52     .box_y(box_y),
53     .box_size(box_size)
54 );
55
56 // Draw box
57 Drawer d (
58     .clk(slowclk),
59     .box_r(box_r),
60     .box_g(box_g),
61     .box_b(box_b),
62     .x(xpos),
63     .y(ypos),
64     .box_x(box_x),
65     .box_y(box_y),
66     .box_size(box_size),
67     .r(r),
68     .g(g),
69     .b(b)
70 );
71
72 endmodule

```

---

### 7.1.7 NES Controller

---

```
1 module nes #(parameter
2     UP = 5, DOWN = 6, LEFT = 7, RIGHT = 8
3 )
4     input logic data, clk,
5     output logic latch_out, clk_out,
6     output logic[1:0] dir_x, dir_y
7 );
8
9 // Track status of x and y direction and if a directional button was pressed
10 logic[1:0] x = 2;
11 logic[1:0] y = 2;
12 logic x_btn_press = 0;
13 logic y_btn_press = 0;
14
15 // Track when to poll controller for button presses
16 logic[19:0] pollClkCount = 0;
17 logic pollClk = 0;
18
19 // Track when to pulse clock
20 logic[9:0] pulseClkCount = 0;
21 logic pulseClk = 0;
22
23 // Track info about latch and clock pulse
24 logic latch_inner = 0;
25 logic new_pulse = 0;
26 logic start_pulse_out = 0;
27 logic[3:0] pulseCounter = 0;
28
29 // Convert clk from 50MHz to 60Hz for polling controller
30 // and to 83kHz to produce clock pulses
31 always_ff @(posedge clk)
32     begin
33         if (pollClkCount == 416666)
34             begin
35                 // Start polling controller by resetting pulse data, starting
36                 // ↪ latch,
37                 // and signaling start of new pulse
38                 if (~pollClk)
39                     begin
40                         latch_inner <= 1;
41                         new_pulse <= 1;
42                         pulseClkCount <= 0;
43                         pulseClk <= 0;
44                         pulseCounter <= 0;
```

```

44         end
45
46         pollClkCount <= 0;
47         pollClk <= ~pollClk;
48     end
49 else
50     pollClkCount <= pollClkCount + 1;
51
52     if (pulseClkCount == 301)
53         begin
54             if (new_pulse && ~pulseClk) // Only update if controller poll
55                 ↪ has started
56                 begin
57                     // Track beginning of clock pulse to controller (after
58                     ↪ latch)
59                     if (pulseCounter > 0 && latch_inner == 0)
60                         start_pulse_out <= 1;
61
62                     // Count pulse
63                     pulseCounter <= pulseCounter + 1;
64                 end
65
66             if (new_pulse && pulseClk) // Only update if controller poll has
67                 ↪ started
68                 begin
69                     // Reset latch once its 12us have passed
70                     if (pulseCounter > 0 && latch_inner)
71                         latch_inner <= 0;
72
73                     // Reset pulse data if all buttons have been accounted
74                     if (pulseCounter == 9)
75                         begin
76                             new_pulse <= 0;
77                             start_pulse_out <= 0;
78                             y_btn_press <= 0;
79                             x_btn_press <= 0;
80                         end
81                     end
82                 else
83                     begin
84                         // Check for UP or DOWN buttons being pressed and
85                         ↪ track that
86                         // a button in y direction has been pressed
87                         if (pulseCounter == UP && data == 0)
88                             begin
89                                 y <= 0;
90                                 y_btn_press <= 1;
91                             end
92                         end
93                     end
94                 end
95             end
96         end
97     end
98 end

```

```

87         else if (pulseCounter == DOWN && data == 0)
88             begin
89                 y <= 1;
90                 y_btn_press <= 1;
91             end
92         else if (pulseCounter >= UP && pulseCounter >=
93             ↪ DOWN && y_btn_press == 0) // Reset if no y
94             ↪ button press
95                 y <= 2;
96
97         // Check for RIGHT or LEFT buttons being pressed
98         ↪ and track that
99         // a button in x direction has been pressed
100         if (pulseCounter == LEFT && data == 0)
101             begin
102                 x <= 0;
103                 x_btn_press <= 1;
104             end
105         else if (pulseCounter == RIGHT && data == 0)
106             begin
107                 x <= 1;
108                 x_btn_press <= 1;
109             end
110         else if (pulseCounter >= LEFT && pulseCounter >=
111             ↪ RIGHT && y_btn_press == 0) // Reset if no x
112             ↪ button press
113                 x <= 2;
114         end
115     end
116
117     pulseClkCount <= 0;
118     pulseClk <= ~pulseClk;
119 end
120 else
121     pulseClkCount <= pulseClkCount + 1;
122 end
123
124 // Assign outputs, and only set clock if part of pulse output
125 assign latch_out = latch_inner;
126 assign clk_out = start_pulse_out && pulseClk;
127 assign dir_x = x;
128 assign dir_y = y;
129
130 endmodule

```

### 7.1.8 Box Controller

---

```
1  module box_controller #(parameter
2      SIZE = 40
3  )(
4      input logic vs,
5      input logic[1:0] dir_x, dir_y,
6      output logic[3:0] box_r, box_g, box_b,
7      output logic[9:0] box_x, box_y, box_size
8  );
9
10 // Track x and y
11 logic[9:0] xpos = 300;
12 logic[9:0] ypos = 220;
13
14 // Track box color
15 logic[3:0] r = 4'b1111;
16 logic[3:0] g = 4'b0;
17 logic[3:0] b = 4'b0;
18
19 always_ff @(posedge vs)
20     begin
21         if(~(dir_x == 0 && xpos == 0) && ~(dir_x == 1 && xpos == 640 - SIZE)) //
22             ↳ Check that box will remain in screen bounds
23             begin
24                 if((xpos - 1 == 0 && dir_x == 0) || (xpos + 1 == 640 - SIZE &&
25                     ↳ dir_x == 1)) // Check for collision
26                     begin
27                         // Swap colors if collision occurs
28                         r <= g;
29                         g <= b;
30                         b <= r;
31                     end
32
33                 // Increment position
34                 case(dir_x)
35                     0: xpos <= xpos - 1;
36                     1: xpos <= xpos + 1;
37                 endcase
38             end
39
40         if(~(dir_y == 0 && ypos == 0) && ~(dir_y == 1 && ypos == 480 - SIZE)) //
41             ↳ Check that box will remain in screen bounds
42             begin
43                 if((ypos - 1 == 0 && dir_y == 0) || (ypos + 1 == 480 - SIZE &&
44                     ↳ dir_y == 1)) // Check for collision
```

```

41         begin
42             // Swap colors if collision occurs
43             r <= g;
44             g <= b;
45             b <= r;
46         end
47
48         // Increment position
49         case(dir_y)
50             0: ypos <= ypos - 1;
51             1: ypos <= ypos + 1;
52         endcase
53     end
54 end
55
56 assign box_x = xpos;
57 assign box_y = ypos;
58 assign box_size = SIZE;
59 assign box_r = r;
60 assign box_g = g;
61 assign box_b = b;
62
63 endmodule

```

---

### 7.1.9 Half Clock

---

```

1  module half_clock(
2      input logic oldclk,
3      output logic newclk
4  );
5
6  logic count = 0;
7
8  always_ff @(posedge oldclk)
9      if (count == 1)
10         begin
11             newclk <= 1;
12             count <= 0;
13         end
14     else
15         begin
16             newclk <= 0;
17             count <= count + 1;
18         end

```

```
19
20 endmodule
```

---

### 7.1.10 XY Counter

---

```
1  module XYCounter(
2      input logic clk,
3      output logic vs, hs,
4      output logic[9:0] x, y
5  );
6
7  // Track current x and y
8  logic[9:0] xpos = 0;
9  logic[9:0] ypos = 0;
10
11  always_ff @(posedge clk)
12      begin
13      // Generate HSync and VSync signals when in sync section
14      hs <= ~((xpos >= 640 + 16) && (xpos < 640 + 16 + 96));
15      vs <= ~((ypos >= 480 + 10) && (ypos < 480 + 10 + 2));
16
17      if(xpos == 800 && ypos == 525) // Reset when at end of screen
18          begin
19              ypos <= 0;
20              xpos <= 0;
21          end
22      else if(xpos == 800) // Increment y and reset x at end of line
23          begin
24              ypos <= ypos + 1;
25              xpos <= 0;
26          end
27      else // Otherwise increment x
28          xpos <= xpos + 1;
29      end
30
31  assign x = xpos;
32  assign y = ypos;
33
34  endmodule
```

---

### 7.1.11 Drawer

---

```
1 module Drawer(  
2     input logic clk,  
3     input logic[3:0] box_r, box_g, box_b,  
4     input logic[9:0] x, y, box_x, box_y, box_size,  
5     output logic[3:0] r, g, b  
6 );  
7  
8 always_ff @(posedge clk)  
9     // Draw black if not in box pixels  
10    if (x > 640 || y > 480 || x < box_x || y < box_y || x > box_x + box_size || y  
11        ↪ > box_y + box_size)  
12        begin  
13            r <= 4'b0;  
14            g <= 4'b0;  
15            b <= 4'b0;  
16        end  
17    else // Draw box  
18        begin  
19            r <= box_r;  
20            g <= box_g;  
21            b <= box_b;  
22        end  
23 endmodule
```

---