Description:

In this lab we created a game called Slug Cross using a VGA monitor and the BASYS3 board. The game is played in a rectangular region (640 x 480 pixels) with blue borders. A green square moves horizontally and vertically within the 'walls' when the push buttons (btnU, btnD, btnL, btnR) are pressed. There are also 7 vertical and 7 horizontal obstacles that must be avoided. There is a gap in each obstacle that moves up and down or left and right. The player ("slug") begins in the top left corner and must reach the bottom right without running into an obstacle in order to win. There is also a minutes and seconds timer on the 7-segment displays that begins when the player starts the game, stops if the player hits an obstacle, and stops and flashes if the player makes it to the bottom right corner. The game difficulty can be changed by adjusting the size of the gap.

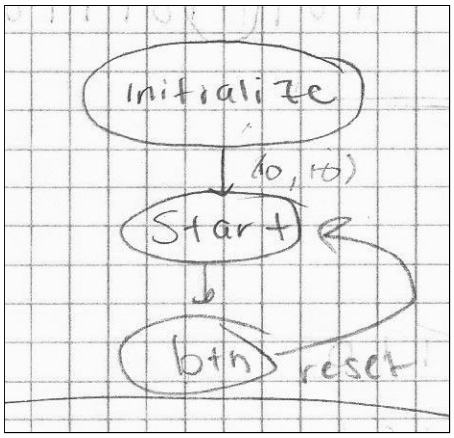
VGAController:

In order to create my VGA Controller I created two counters, one for the current column and one for the current row. I set it up such that the row counter would increment only after the column counter reached the last pixel (799). The output of my horizontal column counter is called HQ and the vertical row counter called VQ. Once I had the counters working I output Hsync, Vsync, active region (AR), and frame (high for one pixel per frame) as follows:

```
Hsync = (HQ <= 10'd654) | (HQ >= 10'd751)
Vsync = (VQ <= 10'd488) | (VQ >= 10'd491)
AR = (HQ <= 10'd639) & (VQ <= 10'd479)
frame = (HQ == 10'd660) & (VQ == 10'd490)
```

Slug:

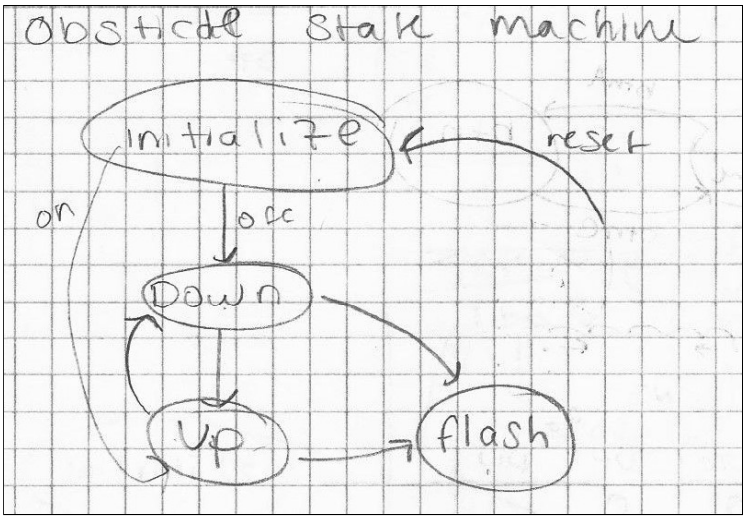
I created a state machine in order to reset my slug to the coordinates (10,10) at the start of every play. My machine only had two states, Initialize and play. When first running the program or anytime btnC is pressed, my state machine remains in initialize where the coordinates (10,10) are loaded into my slug position counters which are also in my state machine module. After any movement button is pressed the counters stop loading in the initial position and the state machine remains in play until btnC is pressed again.



Obstacles:

In order to create my obstacles I decided to create a state machine to each obstacle individually. For simplicity I also placed my gap counter, which kept track of the top left reference pixel of

where the gap should begin, in the same module. The state machine had 4 states: init, up, down, and flash. My state machine was given HQ, and VQ, and output a single bit signal that was high when the current pixel should be the color of an obstacle. If the machine is in the init state the entire obstacle is displayed without a gap until it reaches the up or down state after any direction button is pressed and determined by a direction variable which decides if the first state it should go into after init is up or down (or left or right for horizontal obstacles). If in the flash state, the gap stops moving and begins to flash.



Flash Clocks:

In order to make the slug and obstacles flash I created a module that would count a certain number of frames and then reset. Every time my counter reached that number reset would be high for one clock cycle. I also used a flip flop that alternated between high and low whenever CE was on, I wired the counters reset to CE and this way I have a clock that is high for a given number of frames and then low for the same number of frames and it continues repeating. I used this signal to make things flash.

Time Counter:

In order to create my time counter I created a counter that counts up to 60 frames (1 second) and then resets. When it resets a seconds counter increments, once that seconds timer reaches 9 seconds the next seconds counter increments until the two counters reach 60 seconds, then the first minutes counter is incremented. Once the first minutes counter reaches 9 minutes the next minutes counter increments. If the time module receives a flash signal the 7-segment displays all begin to flash.

Debugging:

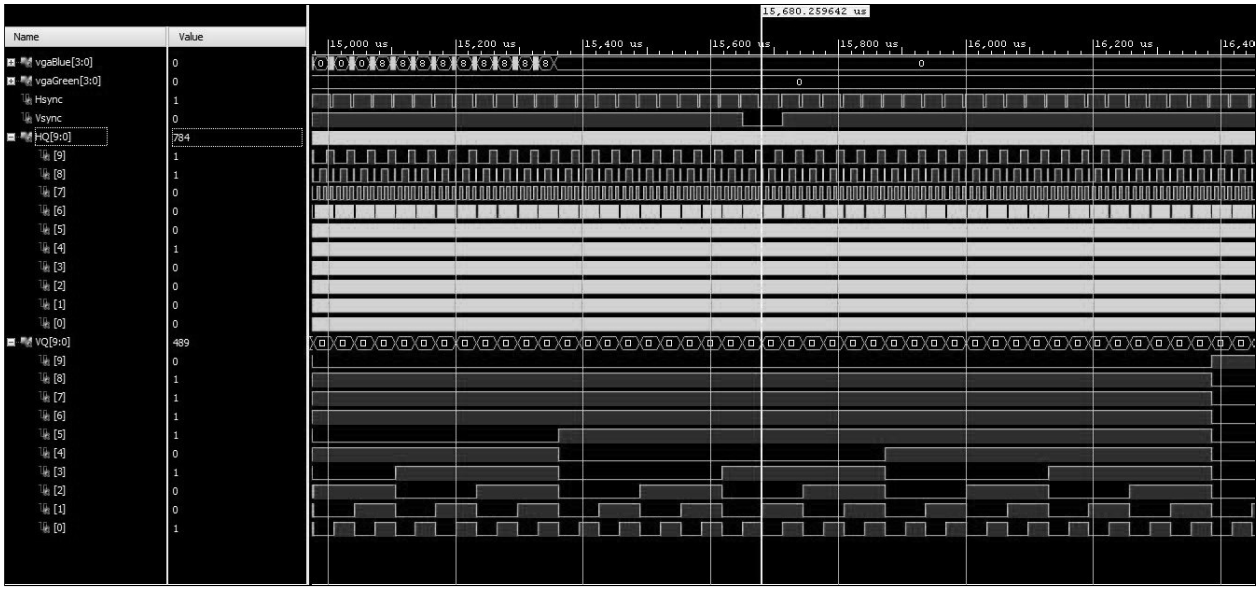
I used the simulator to help me debug my VGA controller. I would run it such that my Hsync and Vsync counters were counting every clock cycle and I made sure that the output signals were changing at the right time. Once I was beginning to create the game, debugging was not so easy on the simulator because you would need to run it for a very long time to get an idea of one single frame. So I would generate a bitstream and figure out the errors in my code by running the game and experimenting different cases to check my bounds and collision logic.

Conclusion:

Overall I enjoyed this lab the most out of all, the liberty of designing our Slug Cross game however we see fit was challenging yet intuitive. I was able to take many of the tools that I have learned in class, in my homework, and through working other labs in order to design something that works the way I intend it to and I am happy with. I began by designing my VGAController to get Hsync and Vsync working, I was able to display a solid color on the screen. After that I programmed the blue walls and continued to create a working slug with my state machine. Once my slug was functioning I created one single working vertical obstacle then one single working horizontal obstacle.

Appendix:

Simulator



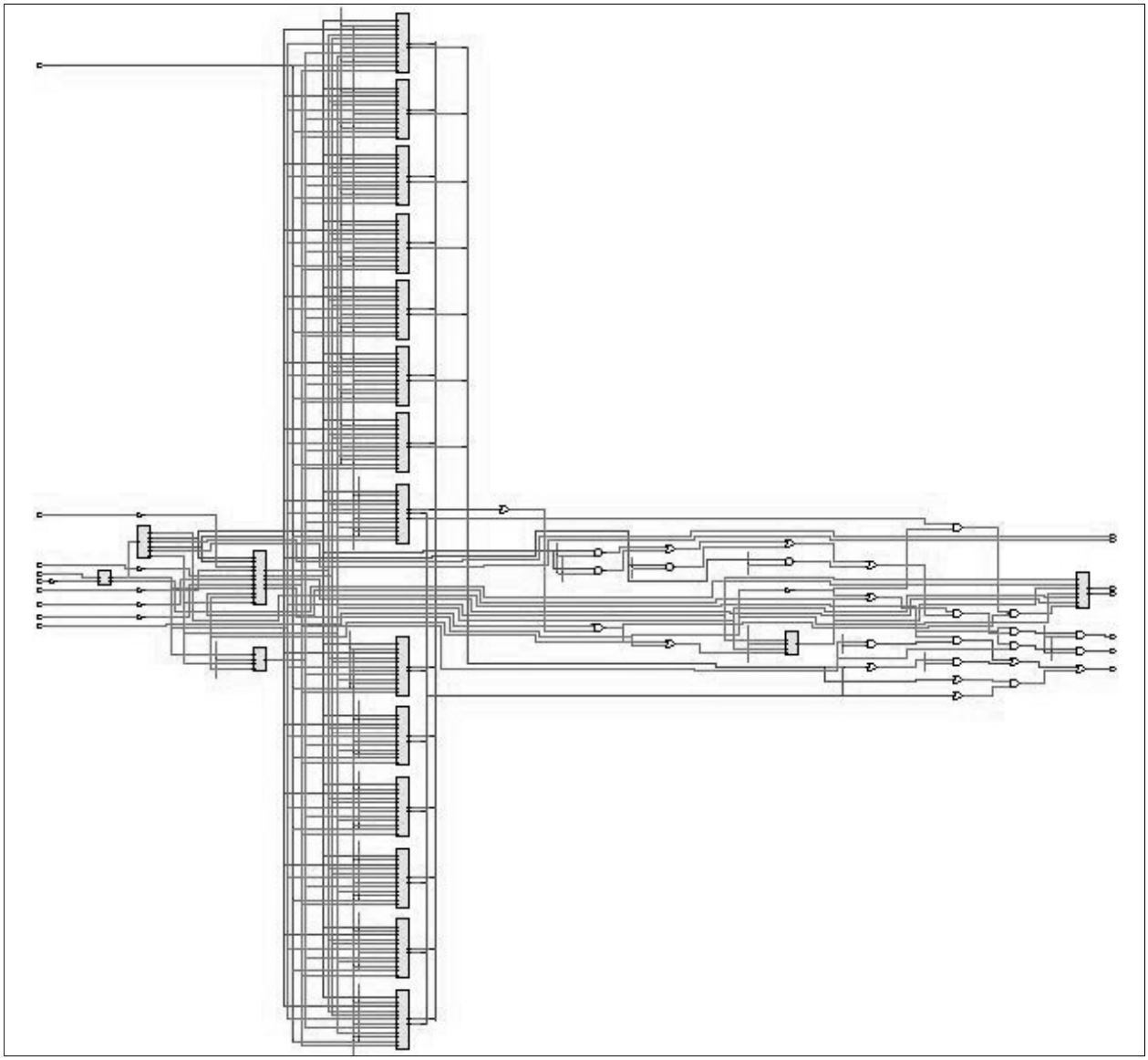
Timing Diagram

Design Timing Summary

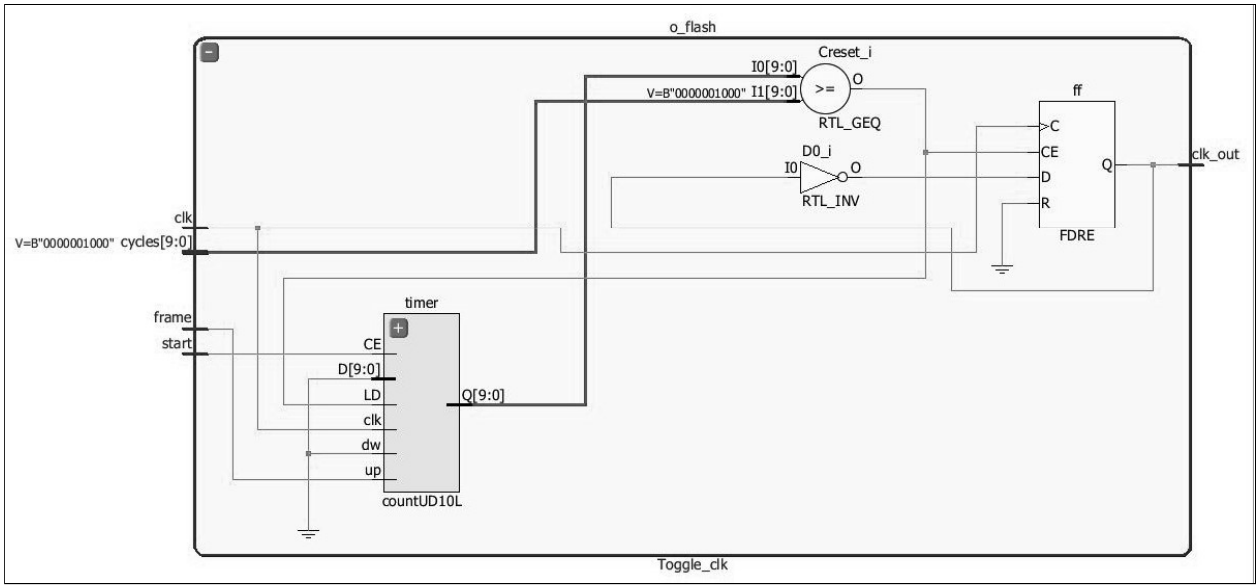
Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	30.489 ns	Worst Hold Slack (WHS):	0.174 ns	Worst Pulse Width Slack (WPWS):	3.000 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	519	Total Number of Endpoints:	519	Total Number of Endpoints:	311

All user specified timing constraints are met.

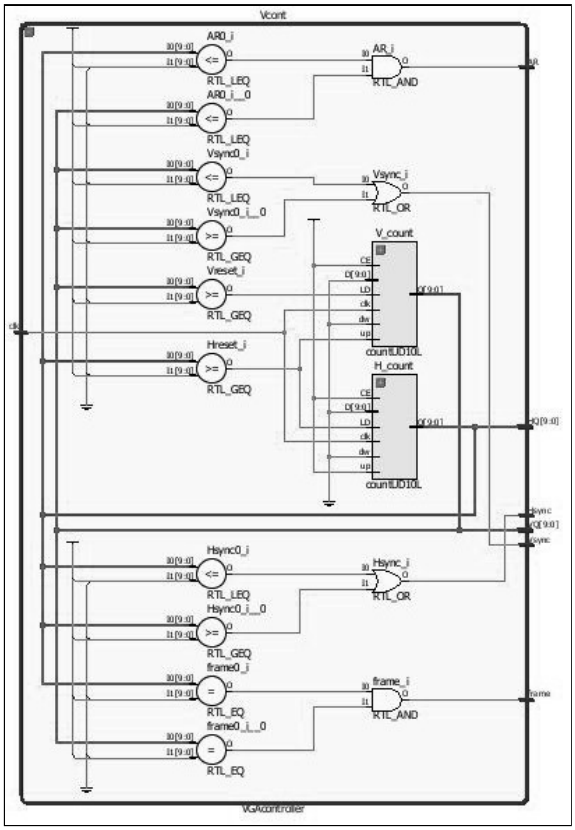
Top Module Schematic



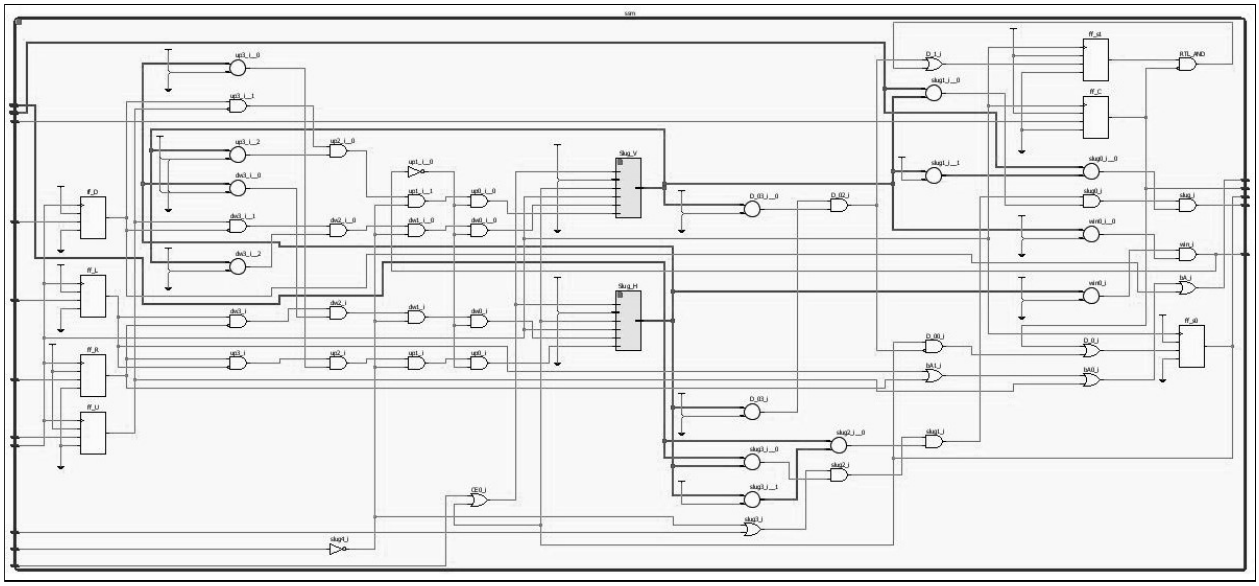
Toggle Clk Schematic



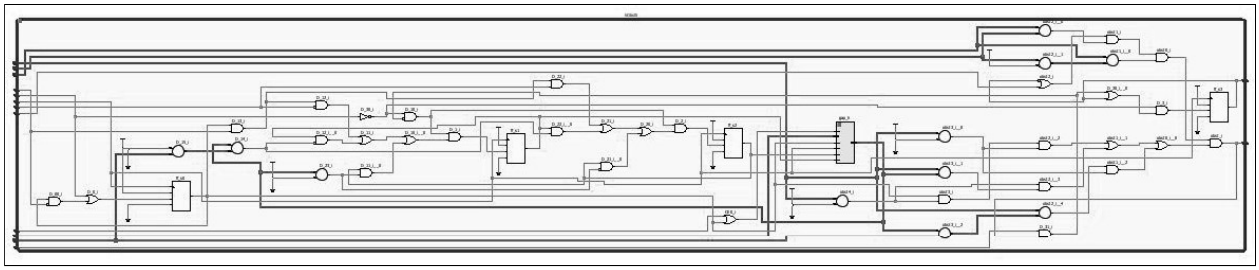
VGA Controller Schematic



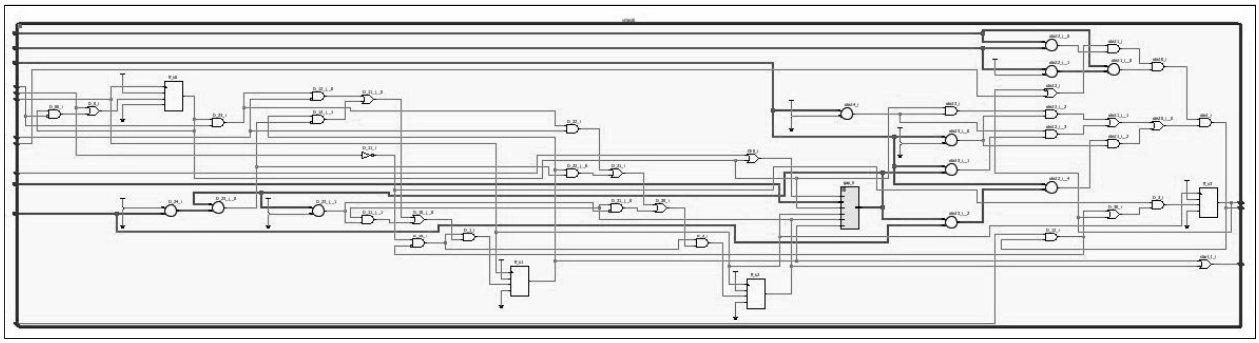
Slugsm Schematic



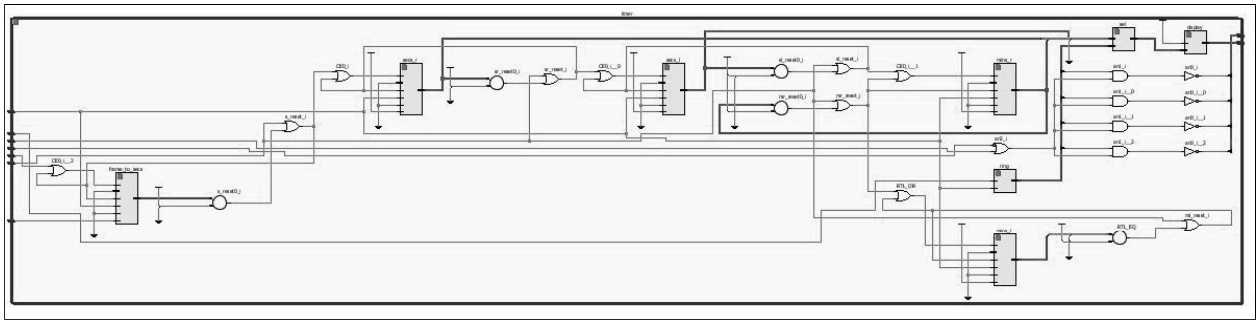
Hobst Schematic



Vobst Schematic



Timer Schematic



Top Module

```
module Top_Lab7(
    input clkin,
    input btnC, btnR, btnU, btnD, btnL,
    input reset_sw, cheat_sw,
    output [6:0] seg,
    output [3:0] an,
    input [6:4] sw,
    output [3:0] vgaRed, vgaBlue, vgaGreen,
    output Hsync, Vsync
    );
    //wires
    wire [13:0] obs, f;
    wire [9:0] HQ, VQ, size;
    wire clk, digsel, o_clk, s_clk, frame, start_flash, obsticle, walls;
    //Given clocks
    lab7_clks not_so_slow (.clkin(clkin), .greset(reset_sw), .clk(clk), .digsel(digsel));
    //Flash clocks
    Toggle_clk o_flash(.clk(clk), .start(start_flash), .frame(frame), .cycles(10'd8), .clk_out(o_clk));
    Toggle_clk s_flash(.clk(clk), .start(start_flash | win), .frame(frame), .cycles(10'd16), .clk_out(s_clk));
    Mins_Secs timer(.clk(clk), .start(start_t & ~win & ~start_flash), .flash(win), .digsel(digsel), .frame(frame),
                    .f_clk(s_clk), .reset(reset), .an(an), .seg(seg));
    VGAcontroller Vcont(.clk(clk), .AR(AR), .Hsync(Hsync), .Vsync(Vsync), .HQ(HQ), .VQ(VQ), .frame(frame));
    //Slug
    Slug_SM ssm(.clk(clk), .btnL(btnL), .btnR(btnR), .btnU(btnU), .btnD(btnD), .btnC(btnC),
                .flash(start_flash), .frame(frame), .f_clk(s_clk),
                .HQ(HQ), .VQ(VQ), .bC(bC), .bA(bA), .slug(slug), .win(win), .reset(reset));
    //Vertical obsticles
    vObst_SM vObst0(.clk(clk), .bC(bC), .bA(bA), .drctn(1'b0), .frame(frame), .slug(slug), .f_clk(o_clk),
            .HQ(HQ), .VQ(VQ), .size(size), .TLC(10'd158), .gap_in(10'd78), .start_t(start_t), .obst(obs[0]), .flash(f[0]));
    vObst_SM vObst1(.clk(clk), .bC(bC), .bA(bA), .drctn(1'b1), .frame(frame), .slug(slug), .f_clk(o_clk),
            .HQ(HQ), .VQ(VQ), .size(size), .TLC(10'd208), .gap_in(10'd128), .obst(obs[1]), .flash(f[1]));
    vObst_SM vObst2(.clk(clk), .bC(bC), .bA(bA), .drctn(1'b0), .frame(frame), .slug(slug), .f_clk(o_clk),
            .HQ(HQ), .VQ(VQ), .size(size), .TLC(10'd258), .gap_in(10'd178), .obst(obs[2]), .flash(f[2]));
    vObst_SM vObst3(.clk(clk), .bC(bC), .bA(bA), .drctn(1'b1), .frame(frame), .slug(slug), .f_clk(o_clk),
            .HQ(HQ), .VQ(VQ), .size(size), .TLC(10'd308), .gap_in(10'd228), .obst(obs[3]), .flash(f[3]));
    vObst_SM vObst4(.clk(clk), .bC(bC), .bA(bA), .drctn(1'b0), .frame(frame), .slug(slug), .f_clk(o_clk),
            .HQ(HQ), .VQ(VQ), .size(size), .TLC(10'd358), .gap_in(10'd278), .obst(obs[4]), .flash(f[4]));
    vObst_SM vObst5(.clk(clk), .bC(bC), .bA(bA), .drctn(1'b1), .frame(frame), .slug(slug), .f_clk(o_clk),
            .HQ(HQ), .VQ(VQ), .size(size), .TLC(10'd408), .gap_in(10'd328), .obst(obs[5]), .flash(f[5]));
    vObst_SM vObst6(.clk(clk), .bC(bC), .bA(bA), .drctn(1'b0), .frame(frame), .slug(slug), .f_clk(o_clk),
            .HQ(HQ), .VQ(VQ), .size(size), .TLC(10'd458), .gap_in(10'd378), .obst(obs[6]), .flash(f[6]));
```

```
//Horizontal obsticles
hObst_SM hObst0(.clk(clk), .bC(bC), .bA(bA), .drctn(1'b1), .frame(frame), .slug(slug), .f_clk(o_clk),
        .HQ(HQ), .VQ(VQ), .size(size), .TLC(10'd78), .gap_in(10'd158), .obst(obs[7]), .flash(f[7]));
hObst_SM hObst1(.clk(clk), .bC(bC), .bA(bA), .drctn(1'b0), .frame(frame), .slug(slug), .f_clk(o_clk),
        .HQ(HQ), .VQ(VQ), .size(size), .TLC(10'd128), .gap_in(10'd208), .obst(obs[8]), .flash(f[8]));
hObst_SM hObst2(.clk(clk), .bC(bC), .bA(bA), .drctn(1'b1), .frame(frame), .slug(slug), .f_clk(o_clk),
        .HQ(HQ), .VQ(VQ), .size(size), .TLC(10'd178), .gap_in(10'd258), .obst(obs[9]), .flash(f[9]));
hObst_SM hObst3(.clk(clk), .bC(bC), .bA(bA), .drctn(1'b0), .frame(frame), .slug(slug), .f_clk(o_clk),
        .HQ(HQ), .VQ(VQ), .size(size), .TLC(10'd228), .gap_in(10'd308), .obst(obs[10]), .flash(f[10]));
hObst_SM hObst4(.clk(clk), .bC(bC), .bA(bA), .drctn(1'b1), .frame(frame), .slug(slug), .f_clk(o_clk),
        .HQ(HQ), .VQ(VQ), .size(size), .TLC(10'd278), .gap_in(10'd358), .obst(obs[11]), .flash(f[11]));
hObst_SM hObst5(.clk(clk), .bC(bC), .bA(bA), .drctn(1'b0), .frame(frame), .slug(slug), .f_clk(o_clk),
        .HQ(HQ), .VQ(VQ), .size(size), .TLC(10'd328), .gap_in(10'd408), .obst(obs[12]), .flash(f[12]));
hObst_SM hObst6(.clk(clk), .bC(bC), .bA(bA), .drctn(1'b1), .frame(frame), .slug(slug), .f_clk(o_clk),
        .HQ(HQ), .VQ(VQ), .size(size), .TLC(10'd378), .gap_in(10'd458), .obst(obs[13]), .flash(f[13]));
assign size = {2'b00, sw[6:4], 5'b10000};
assign start_flash = ~cheat_sw & (| f);
assign obsticle = (| obs);
assign brighten = (| obs[13:7]) & (| obs[6:0]);
assign walls = (~win | s_clk) & ((HQ <= 10'd7) | (HQ >= 10'd631) | (VQ <= 10'd7) | (VQ >= 10'd471));
assign vgaRed = {4{AR}} & ({4{slug}} & {4'b1000}) | ({4{obsticle}} & {4'b1000}) | ({4{brighten}} & {4'b1111});
assign vgaBlue = {4{AR}} & {4{walls}} & {4'b1000};
assign vgaGreen = {4{AR}} & {4{slug}} & {4{4'b1000}};
```

VGA Controller Module

```
module VGAcontroller(
    input clk,
    output Hsync, Vsync, AR, frame,
    output [9:0] HQ, VQ
    );

    //Column Counter
    assign Hreset = (HQ >= 10'd799);
    countUD10L H_count(.clk(clk), .up(1'b1), .dw(1'b0), .CE(1'b1), .LD(Hreset), .D(10'd0), .Q(HQ));
    assign Hsync = (HQ <= 10'd654) | (HQ >= 10'd751);

    //Row Counter
    assign Vreset = (VQ >= 10'd524);
    countUD10L V_count(.clk(clk), .up(Hreset), .dw(1'b0), .CE(1'b1), .LD(Vreset), .D(10'd0), .Q(VQ));
    assign Vsync = (VQ <= 10'd488) | (VQ >= 10'd491);

    //Active Region
    assign AR = (HQ <= 10'd639) & (VQ <= 10'd479);

    //High once per frame
    assign frame = (HQ == 10'd660) & (VQ == 10'd490);

endmodule
```

Toggle Clk Module

```
module Toggle_clk(
    input clk, start, frame,
    input [9:0] cycles,
    output clk_out
    );

    wire [9:0] CQ;
    countUD10L timer(.clk(clk), .up(frame), .dw(1'b0), .CE(start), .LD(Creset), .D(10'd0), .Q(CQ));
    assign Creset = (CQ >= cycles);

    FDRE #(.INIT(1'b0)) ff(.C(clk), .R(1'b0), .CE(Creset), .D(~clk_out), .Q(clk_out));

endmodule
```

Slugsm Module

```
module Slug_SM(
    input clk, btnL, btnR, btnU, btnD, btnC,
    input flash, frame, f_clk,
    input [9:0] HQ, VQ,
    output bC, bA, slug, win, reset
    );

    wire [2:0] D, Q;
    wire [9:0] slug_h, slug_v;

    //syncing inputs to clock
    FDRE #(.INIT(1'b0)) ff_L (.C(clk), .R(1'b0), .CE(1'b1), .D(btnL), .Q(bL));
    FDRE #(.INIT(1'b0)) ff_R (.C(clk), .R(1'b0), .CE(1'b1), .D(btnR), .Q(bR));
    FDRE #(.INIT(1'b0)) ff_U (.C(clk), .R(1'b0), .CE(1'b1), .D(btnU), .Q(bU));
    FDRE #(.INIT(1'b0)) ff_D (.C(clk), .R(1'b0), .CE(1'b1), .D(btnD), .Q(bD));
    FDRE #(.INIT(1'b0)) ff_C (.C(clk), .R(1'b0), .CE(1'b1), .D(btnC), .Q(bC));
    assign bA = bL | bR | bU | bD;

    assign D[0] = bC | Q[0] & ~(slug_h == 10'd10 & slug_v == 10'd10);
    assign D[1] = (slug_h == 10'd10 & slug_v == 10'd10) | Q[1] & ~bC;
    FDRE #(.INIT(1'b1)) ff_s0 (.C(clk), .R(1'b0), .CE(1'b1), .D(D[0]), .Q(Q[0]));
    FDRE #(.INIT(1'b0)) ff_s1 (.C(clk), .R(1'b0), .CE(1'b1), .D(D[1]), .Q(Q[1]));

    //Slug Position
    assign reset = Q[0];
    countUD10L Slug_H(.clk(clk), .up(bR & ~bL & (slug_h < 10'd615) & ~flash & ~win), .dw(bL & ~bR & (slug_h > 10'd8) & ~flash & ~win),
                      .CE(frame | reset), .LD(reset), .D(10'd10), .Q(slug_h));
    countUD10L Slug_V(.clk(clk), .up(bD & ~bU & (slug_v < 10'd455) & ~flash & ~win), .dw(bU & ~bD & (slug_v > 10'd8) & ~flash & ~win),
                      .CE(frame | reset), .LD(reset), .D(10'd10), .Q(slug_v));

    assign slug = (~flash | f_clk) & (HQ >= slug_h) & (HQ <= (slug_h + 10'd15)) & (VQ >= slug_v) & (VQ <= (slug_v + 10'd15));
    assign win = (slug_h >= 10'd609) & (slug_v >= 10'd449);
```

Hobstsm Module

```
module hObst_SM(
    input clk, bC, bA, drctn, frame, slug, f_clk,
    input [9:0] HQ, VQ, size, TLC, gap_in,
    output obst, flash
    );

    wire [3:0] D, Q;
    wire [9:0] gap;

    assign D[0] = bC | Q[0] & ~bA;
    assign D[1] = ~bC & ~(slug & obst) & ( Q[0] & bA & ~drctn | Q[1] & ~(gap >= (10'd631 - size)) | Q[2] & (gap == 10'd8));
    assign D[2] = ~bC & ~(slug & obst) & (Q[0] & bA & drctn | Q[1] & (gap >= (10'd631 - size)) | Q[2] & ~(gap == 10'd8));
    assign D[3] = ~bC & (slug & obst | Q[3]);
    FDRE #(.INIT(1'b1)) ff_s0 (.C(clk), .R(1'b0), .CE(1'b1), .D(D[0]), .Q(Q[0]));
    FDRE #(.INIT(1'b0)) ff_s1 (.C(clk), .R(1'b0), .CE(1'b1), .D(D[1]), .Q(Q[1]));
    FDRE #(.INIT(1'b0)) ff_s2 (.C(clk), .R(1'b0), .CE(1'b1), .D(D[2]), .Q(Q[2]));
    FDRE #(.INIT(1'b0)) ff_s3 (.C(clk), .R(1'b0), .CE(1'b1), .D(D[3]), .Q(Q[3]));
    assign flash = Q[3];
    assign reset = Q[0];

    countUD10L gap_h(.clk(clk), .up(Q[1]), .dw(Q[2]), .CE(frame | reset), .LD(reset), .D(gap_in), .Q(gap));

    assign obst = (~flash | f_clk) & (VQ >= TLC) & (VQ <= (TLC + 10'd7)) &
                      ((Q[0] & (HQ >= 10'd8) & (HQ <= 10'd630)) |
                      ((HQ >= 10'd8) & (HQ < gap)) | ((HQ > (gap + size)) & (HQ <= 10'd630)));
```

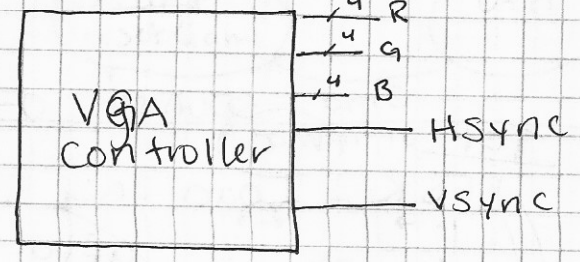
Vobstsm Module

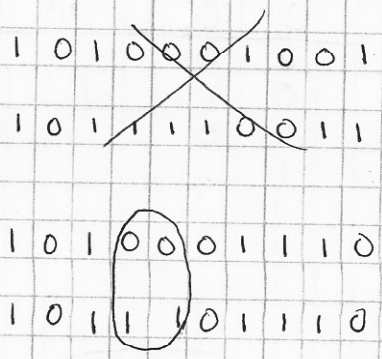
```
module vObst_SM(
    input clk, bC, bA, drctn, frame, slug, f_clk,
    input [9:0] HQ, VQ, size, TLC, gap_in,
    output start_t, obst, flash
    );

    wire [3:0] D, Q;
    wire [9:0] gap;

    assign D[0] = bC | Q[0] & ~bA;
    assign D[1] = ~bC & ~(slug & obst) & (Q[0] & bA & ~drctn | Q[1] & ~(gap >= (10'd471 - size)) | Q[2] & (gap == 10'd7));
    assign D[2] = ~bC & ~(slug & obst) & (Q[0] & bA & drctn | Q[1] & (gap >= (10'd471 - size)) | Q[2] & ~(gap == 10'd7));
    assign D[3] = ~bC & (slug & obst | Q[3]);
    FDRE #(.INIT(1'b1)) ff_s0 (.C(clk), .R(1'b0), .CE(1'b1), .D(D[0]), .Q(Q[0]));
    FDRE #(.INIT(1'b0)) ff_s1 (.C(clk), .R(1'b0), .CE(1'b1), .D(D[1]), .Q(Q[1]));
    FDRE #(.INIT(1'b0)) ff_s2 (.C(clk), .R(1'b0), .CE(1'b1), .D(D[2]), .Q(Q[2]));
    FDRE #(.INIT(1'b0)) ff_s3 (.C(clk), .R(1'b0), .CE(1'b1), .D(D[3]), .Q(Q[3]));
    assign flash = Q[3];
    assign reset = Q[0];
    assign start_t = Q[1] | Q[2];

    countUD10L gap_h(.clk(clk), .up(Q[1]), .dw(Q[2]), .CE(frame | reset), .LD(reset), .D(gap_in), .Q(gap));

    assign obst = (~flash | f_clk) & (HQ >= TLC) & (HQ <= (TLC + 10'd7)) &
                      ((Q[0] & (VQ >= 10'd8) & (VQ <= 10'd470)) |
                      ((VQ >= 10'd8) & (VQ < gap)) | ((VQ > (gap + size)) & (VQ <= 10'd470)));
```

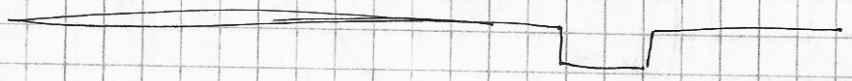
Timer Module

```
module Mins_Secs(
    input clk, start, flash, digsel, frame, f_clk, reset,
    output [3:0] an,
    output [6:0] seg
    );

    wire [11:0] disp;
    wire [9:0] s_out;
    wire [3:0] mr_out, ml_out, sr_out, sl_out;

    //s_reset is high once per second
    assign s_reset = reset | (s_out == 10'd60);
    countUD10L frame_to_secs(.clk(clk), .up(start), .dw(1'b0), .CE(frame | s_reset), .LD(s_reset), .D(10'd0), .Q(s_out));

    //seconds
    assign sr_reset = reset | (sr_out == 5'd10);
    countUD5L secs_r(.clk(clk), .up(1'b1), .dw(1'b0), .CE(s_reset | sr_reset), .LD(sr_reset), .D(5'd0), .Q(sr_out));
    assign sl_reset = reset | (sl_out == 5'd6);
    countUD5L secs_l(.clk(clk), .up(1'b1), .dw(1'b0), .CE(sr_reset | sl_reset), .LD(sl_reset), .D(5'd0), .Q(sl_out));

    //minutes
    assign mr_reset = reset | (mr_out == 5'd10);
    countUD5L mins_r(.clk(clk), .up(1'b1), .dw(1'b0), .CE(sl_reset | mr_reset), .LD(mr_reset), .D(5'd0), .Q(mr_out));
    assign ml_reset = reset | (ml_out == 5'd6);
    countUD5L mins_l(.clk(clk), .up(1'b1), .dw(1'b0), .CE(mr_reset | ml_reset), .LD(ml_reset), .D(5'd0), .Q(ml_out));

    assign disp = {ml_out[3:0], mr_out[3:0], sl_out[3:0], sr_out[3:0]};

    wire [3:0] toSel, toHex;
    ringCounter ring(.advance(digsel), .clk(clk), .out(toSel));
    selector sel(.in(disp), .sel(toSel), .H(toHex));
    hex7seg display(.n(toHex), .e(1'b1), .seg(seg));

    assign an[0] = ~(toSel[0] & (~flash | f_clk));
    assign an[1] = ~(toSel[1] & (~flash | f_clk));
    assign an[2] = ~(toSel[2] & (~flash | f_clk));
    assign an[3] = ~(toSel[3] & (~flash | f_clk));

endmodule
```

Gap Length: SW[6:4] * 32 + 16

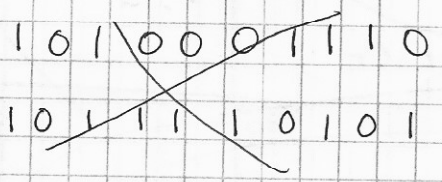


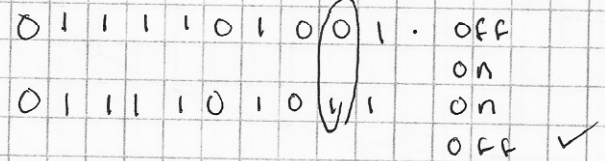


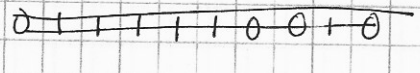


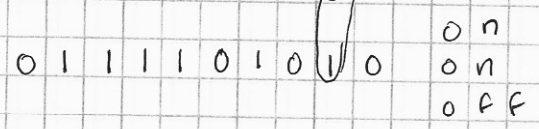
799

655



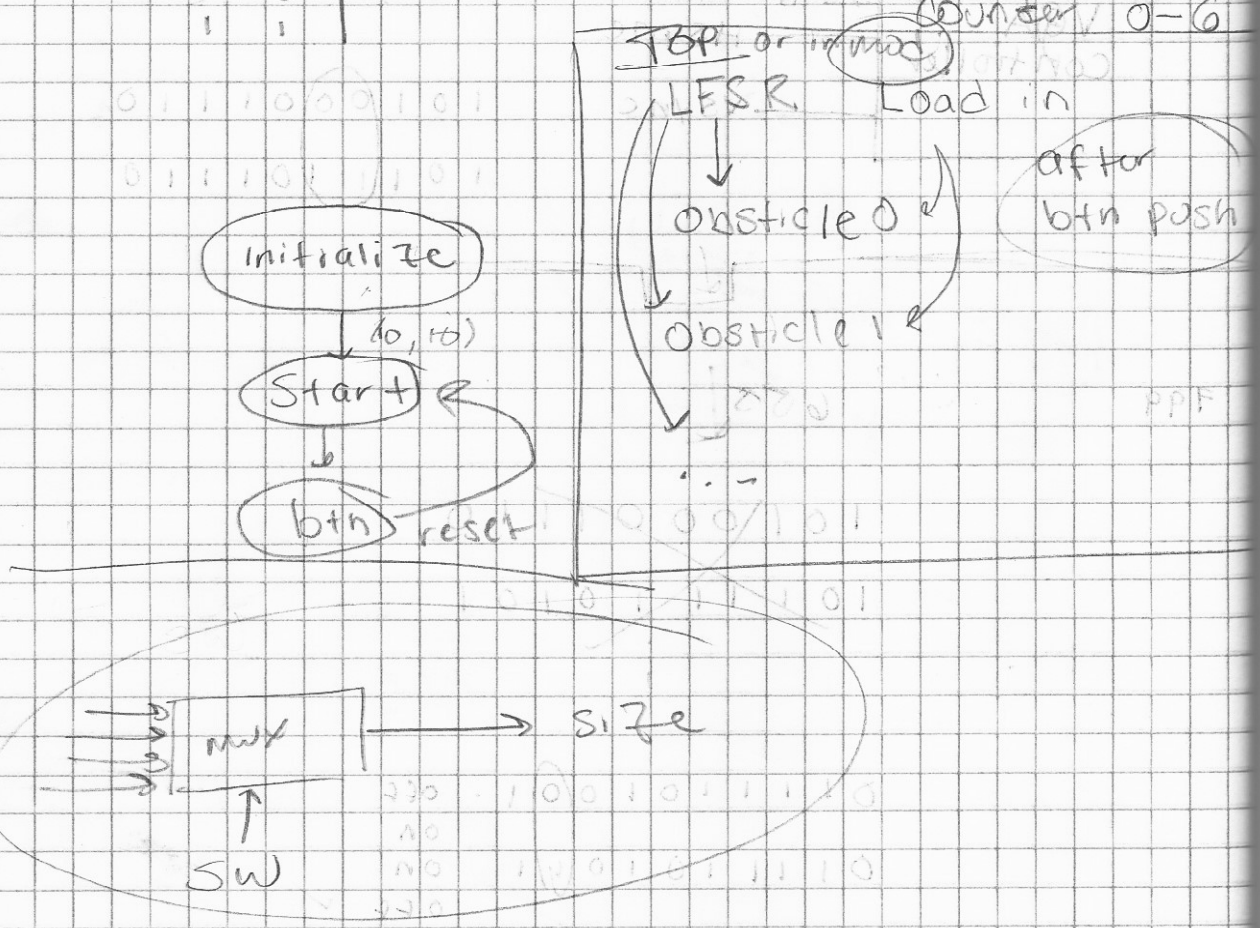


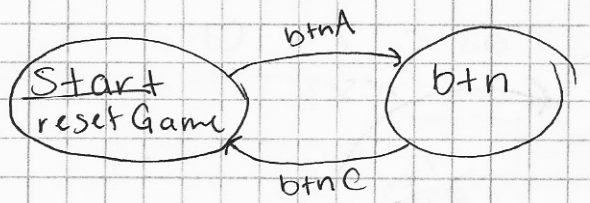


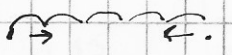


PS

Q_1	Q_0	UTC UP	DTC DW
0	0		
0	1		
1	0		
1	1		



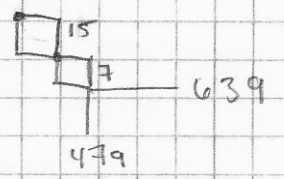


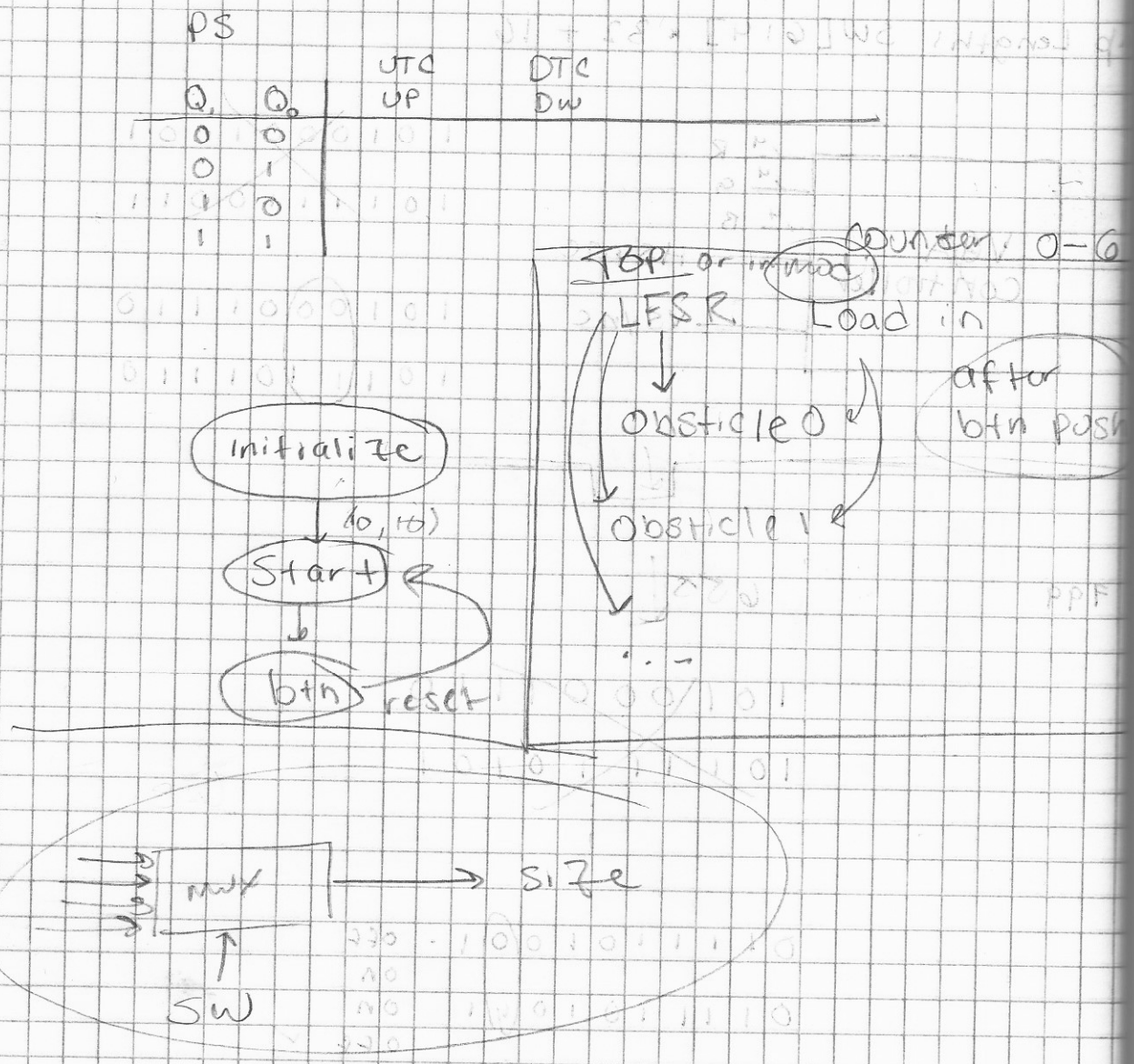


NS Q_0

PS

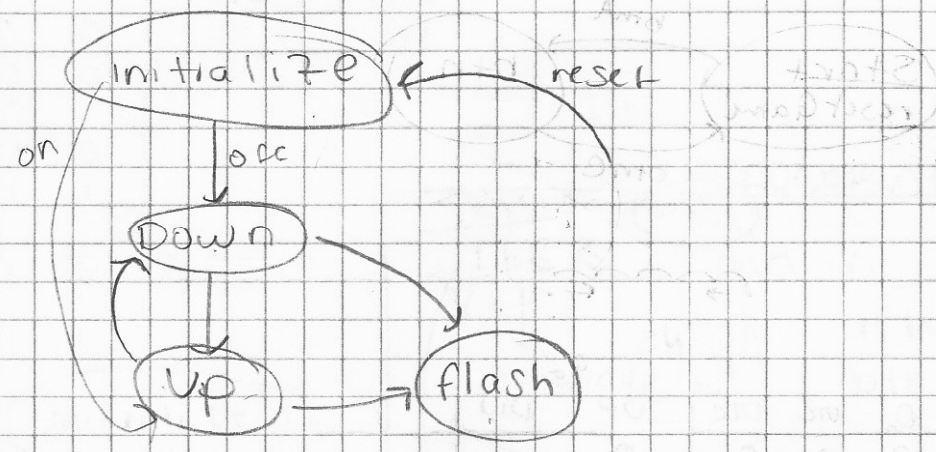
Q_1	Q_0	UTC	DTC	UP	DW
0	0	0	0	0	
0	0	0	1		
0	0	1	0		
0	1	0	0		
0	1	0	1		
0	1	1	0		
1	0	0	0		
1	0	0	1		
1	0	1	0		
1	1	0	0		
1	1	0	1		
1	1	1	0		



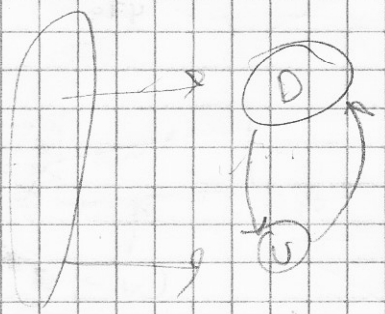

PS
Q_1 Q_0
UTC
UP
DTC
DW
TOP or initial
counter: 0-6
LFSR
Load in
after
btn push
obstacle0
obstacle1
initialize
(0, 10)
start
btn
reset
mux
size
SW

9

Obstickle Stak machine



If any Obstical is flashing the slug will also need to flash

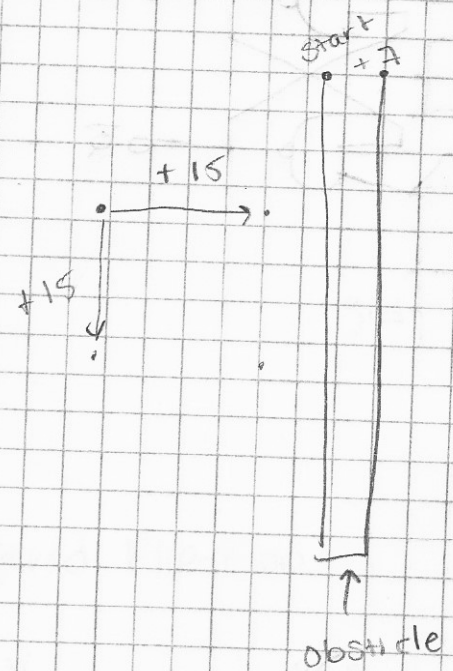


counter (reset 60 o

counter (up 9 rest 0)

counter (up #Reset)

$10 > x > 20$



2640

11.29.18

2:06 pm

~~HARLEY~~