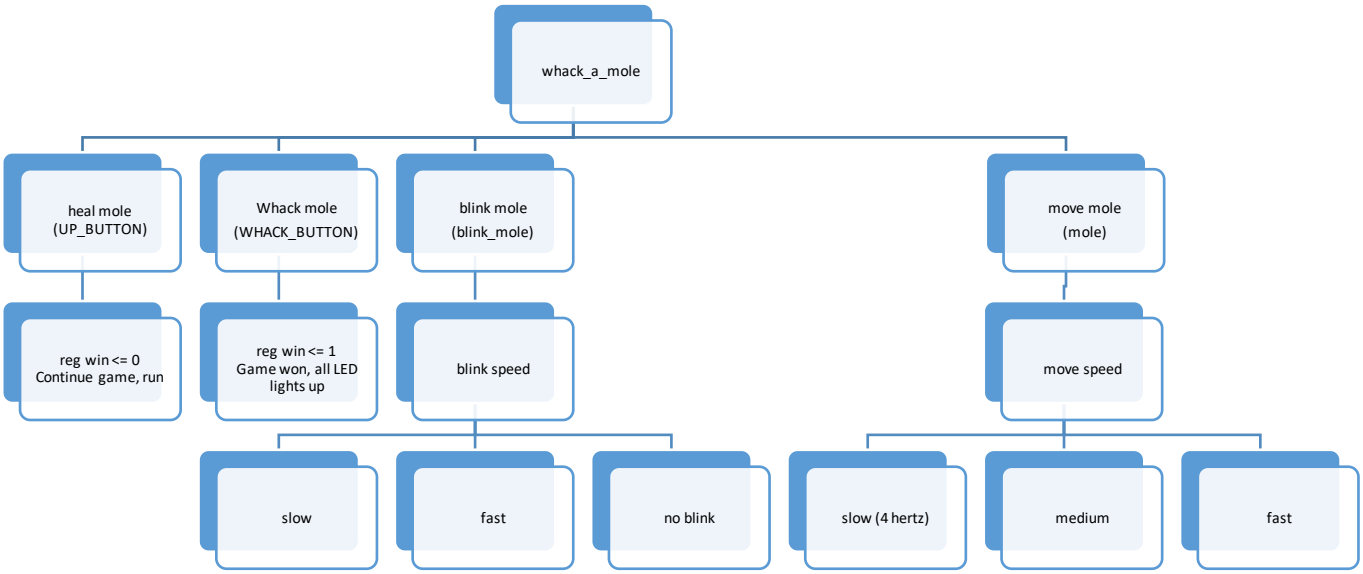
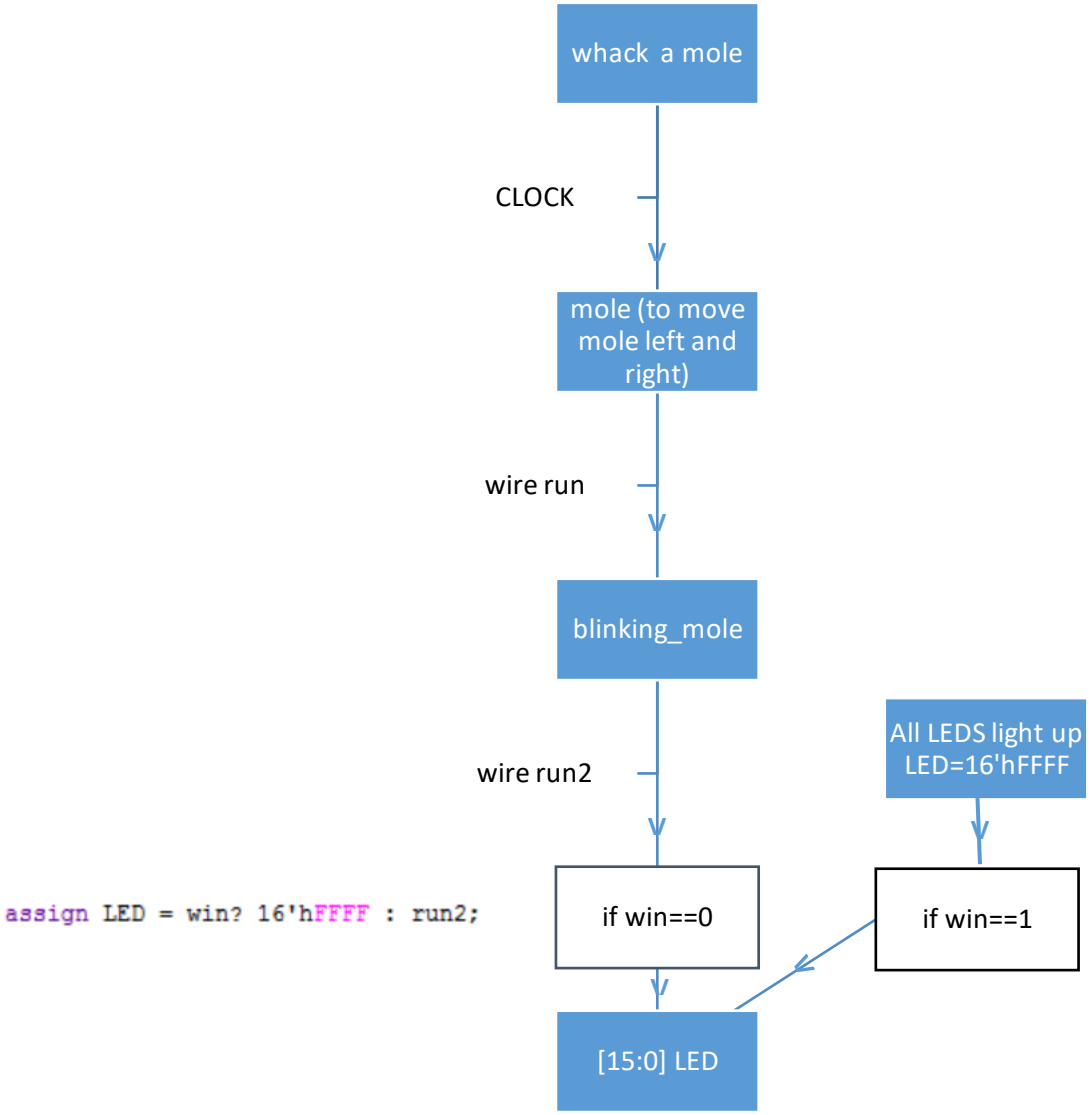





```
assign LED = win? 16'hFFFF : run2;
```

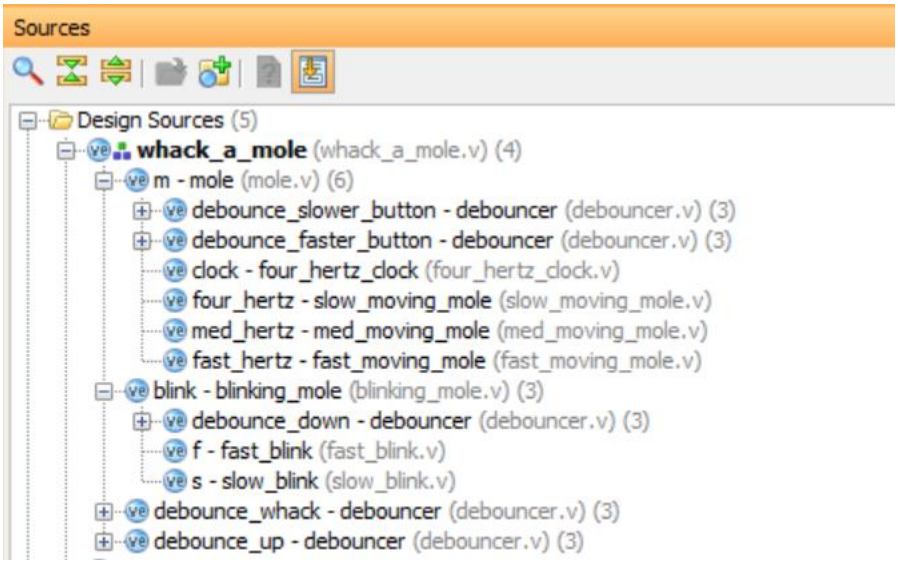
```
always @ (CLOCK) begin
if(whack==1 && ((LED[0]==1 && whackable_hole[0]==1)||
(LED[1]==1 && whackable_hole[1]==1)||
(LED[2]==1 && whackable_hole[2]==1)||
(LED[3]==1 && whackable_hole[3]==1)||
(LED[4]==1 && whackable_hole[4]==1)||
(LED[5]==1 && whackable_hole[5]==1)||
(LED[6]==1 && whackable_hole[6]==1)||
(LED[7]==1 && whackable_hole[7]==1)||
(LED[8]==1 && whackable_hole[8]==1)||
(LED[9]==1 && whackable_hole[9]==1)||
(LED[10]==1 && whackable_hole[10]==1)||
(LED[11]==1 && whackable_hole[11]==1)||
(LED[12]==1 && whackable_hole[12]==1)||
(LED[13]==1 && whackable_hole[13]==1)||
(LED[14]==1 && whackable_hole[14]==1)||
(LED[15]==1 && whackable_hole[15]==1)))
    begin
    win <= 1;
    end
else if(heal==1)
    begin
    win<=0;
    end
end
```

Whack mole
Check if whack was successful
win = 1

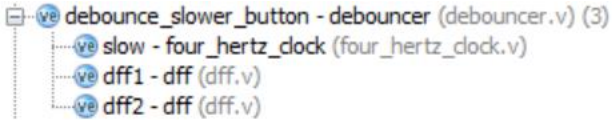
Heal mole

Win = 0

Overall hierarchy of sources



debouncer sources



Whack_a_mole code

```
module whack_a_mole(
  input CLOCK,
  input LEFT_BUTTON,
  input RIGHT_BUTTON,
```

```
input UP_BUTTON,
input DOWN_BUTTON,
input WHACK_BUTTON,
input [15:0] whackable_hole,
output [15:0]LED
);
wire whack, heal;
wire [15:0]run;
wire [15:0]run2;
reg win;

initial
begin
win=1'b0;
end
mole m(CLOCK, LEFT_BUTTON, RIGHT_BUTTON, run); //moving mole
blinking_mole blink(DOWN_BUTTON, run, CLOCK, run2);

debouncer debounce_whack(WHACK_BUTTON, CLOCK, whack);
always @ (CLOCK) begin
if(whack==1 && ((LED[0]==1 && whackable_hole[0]==1)||
(LED[1]==1 && whackable_hole[1]==1)||
(LED[2]==1 && whackable_hole[2]==1)||
(LED[3]==1 && whackable_hole[3]==1)||
(LED[4]==1 && whackable_hole[4]==1)||
(LED[5]==1 && whackable_hole[5]==1)||
(LED[6]==1 && whackable_hole[6]==1)||
(LED[7]==1 && whackable_hole[7]==1)||
(LED[8]==1 && whackable_hole[8]==1)||
(LED[9]==1 && whackable_hole[9]==1)||
(LED[10]==1 && whackable_hole[10]==1)||
(LED[11]==1 && whackable_hole[11]==1)||
(LED[12]==1 && whackable_hole[12]==1)||
(LED[13]==1 && whackable_hole[13]==1)||
(LED[14]==1 && whackable_hole[14]==1)||
(LED[15]==1 && whackable_hole[15]==1)))
```

```
    begin
    win <= 1;
    end
  else if(heal==1)
    begin
    win<=0;
    end
  end

  assign LED = win? 16'hFFFF : run2;
  debouncer debounce_up(UP_BUTTON, CLOCK, heal);
endmodule
```

mole (to control the mole moving left and right) code

```
module mole(
  input CLOCK,
  input slower_button,
  input faster_button,
  output [15:0]LED
  );
  wire slow, med, fast, slower, faster, move;
  reg [1:0] speed = 2'b01;
  reg [15:0] LED = 16'h01; //indicates mole position
  reg [4:0]counter = 0;

  wire slowclock;



  debouncer debounce_slower_button(slower_button, CLOCK, slower);
  debouncer debounce_faster_button(faster_button, CLOCK, faster);

 //to slow down clock, so that the period in which slower = 1 will only be read by one slowclock //posedge signal.
Prevent double counting of slower/faster signals
four_hertz_clock clock(CLOCK, slowclock);

  always @ (posedge slowclock) begin
```

```
    if((speed==2||speed==3)&& slower==1)
        begin
        speed <= speed - 1;
        end
    else if ((speed==1||speed==2)&&faster==1)
        begin
        speed <= speed + 1;
        end
   end

  assign move = (speed == 2'b01)?
        slow : (speed == 2'b10)?
        med : fast;

  slow_moving_mole four_hertz(CLOCK, slow);
  med_moving_mole med_hertz(CLOCK, med);
  fast_moving_mole fast_hertz(CLOCK, fast);

   always @ (posedge move) begin
     counter <= counter + 1;
     if(counter < 15)
        LED <= LED << 1;
     else if(counter <30)
        LED <= LED >> 1;
     if(counter == 29)
        counter<=0;
   end
endmodule
```

slow_moving_mole (1 of 3 mole run speeds) code

```
module slow_moving_mole(
  input CLOCK,
  output move
  );

  reg [22:0] COUNT= 23'b00000000000000000000000;
```

```
   reg move = 0;


      always @(posedge CLOCK) begin
         COUNT<=COUNT+1;
         move<=(COUNT==23'b00000000000000000000000)?~move:move;
       end
endmodule
```

med_moving mole (1 of 3 mole run speeds) code

```
module med_moving_mole(
   input CLOCK,
   output move
   );

   reg [21:0] COUNT= 22'b0000000000000000000000;
   reg move = 0;

      always @(posedge CLOCK) begin
         COUNT<=COUNT+1;
         move<=(COUNT==22'b0000000000000000000000)?~move:move;
       end
endmodule
```

fast_moving_mole (1 of 3 mole run speeds) code

```
module fast_moving_mole(
   input CLOCK,
   output move
   );

   reg [20:0] COUNT= 21'b000000000000000000000;
   reg move = 0;

      always @(posedge CLOCK) begin
         COUNT<=COUNT+1;
         move<=(COUNT==21'b000000000000000000000)?~move:move;
       end
```

```
endmodule
```

blinking_mole code

```
module blinking_mole(
    input down_button,
    input [15:0]run,
    input CLOCK,
    output [15:0]blink
    );
    debouncer debounce_down(down_button, CLOCK, change_blink);

    reg [1:0]speed = 0;
    wire [15:0] fast;
    wire [15:0]slow;

    always @(posedge change_blink) begin
        speed<=speed+1;
        if(speed==3)
        begin
        speed<=0;
        end
    end

    fast_blink f(run, CLOCK, fast);
    slow_blink s(run, CLOCK, slow);

    assign blink = (speed==0)? run : (speed == 1)? slow : fast;

endmodule
```

fast_blink code

```
module fast_blink(
    input [15:0] run,
    input CLOCK,
    output [15:0] LED
    );


     reg [26:0] COUNT= 27'b000000000000000000000000000;
     parameter [27:0]maxCount=27'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF/2;
     //bits of maxCount will be truncated


       always @(posedge CLOCK) begin
           COUNT<=COUNT+1;
        end

        assign LED = (COUNT<=(maxCount/2))?0:run;

endmodule
```

slow_blink code

```
module slow_blink(
    input [15:0] run,
    input CLOCK,
    output[15:0] LED
    );

    parameter [30:0]maxCount=31'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF/2;
     reg [30:0] COUNT= 31'b0000000000000000000000000;


       always @(posedge CLOCK) begin
           COUNT<=COUNT+1;
        end

        assign LED = (COUNT<=maxCount/2)?0:run;
endmodule
```

misc modules

debouncer

```
module debouncer(
    input BUTTON,
    input CLOCK,
    output pulse
    );

    wire Q1, Q2, pulse, SLOWCLOCK;

        four_hertz_clock slow(CLOCK, SLOWCLOCK);

        dff dff1(SLOWCLOCK, BUTTON, Q1);
        dff dff2(SLOWCLOCK, Q1, Q2);

        assign pulse= Q1 & ~Q2;

endmodule
```

four_hertz_clock

```
module four_hertz_clock(
    input CLOCK,
    output LED
    );

    reg [22:0] COUNT= 23'b00000000000000000000000;
    reg LED = 0;

    always @(posedge CLOCK) begin
        COUNT<=COUNT+1;
        LED<=(COUNT==23'b00000000000000000000000)?~LED:LED;
     end

endmodule
```

dff

```
module dff(
    input DFF_CLOCK,
    input D,
    output reg Q
    );

    always @ (posedge DFF_CLOCK) begin
        Q<=D;
    end

endmodule
```

Feedback

I enjoyed this project a lot, however if more verilog examples are given it would make figuring out errors and debugging a lot easier. (for example `always @(posedge slower or posedge faster)` does not work). Had to use slower and faster signal from buttons as variable conditions in `always @(posedge clock)`. This project though fun and addictive is very time consuming. It also happens during midterm week. Thus it can be quite stressful, to meet deadlines. The report writing wasn't fun as I didn't really know what is the ideal way to present my report. Report examples would be appreciated.