00001

ECE 350: Final Project

Cathy Wang and Samantha Whitt

Minimum Viable Product

Dinosaur Jump

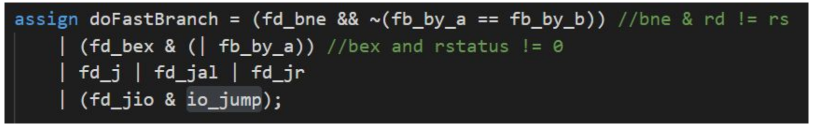
Overview

- Fixed x position, variable y position
- Jump in response to button push
- Processor integration
    - $r1: x position
    - $r2: y position
    - New instruction (11111): branch if button pressed
    - New instruction (11100): stall till screen end

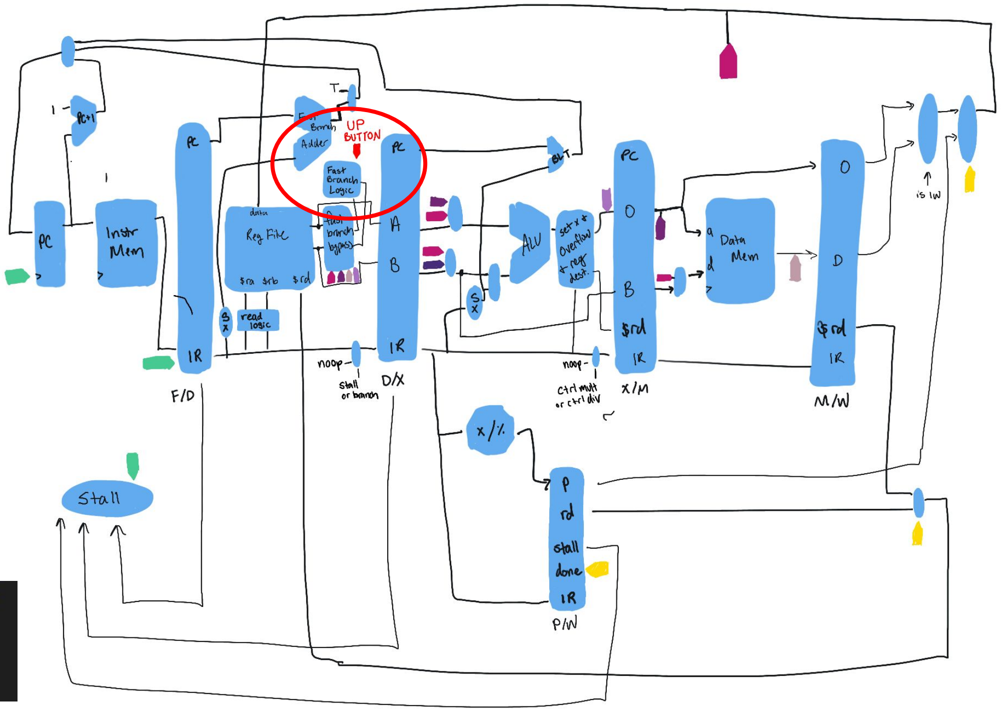
Dinosaur Jump: Processor Changes

Branch if Button Pressed

- Instruction: bio T
- Opcode: 11111
- Type: JI
- if (IO = 1) PC = T



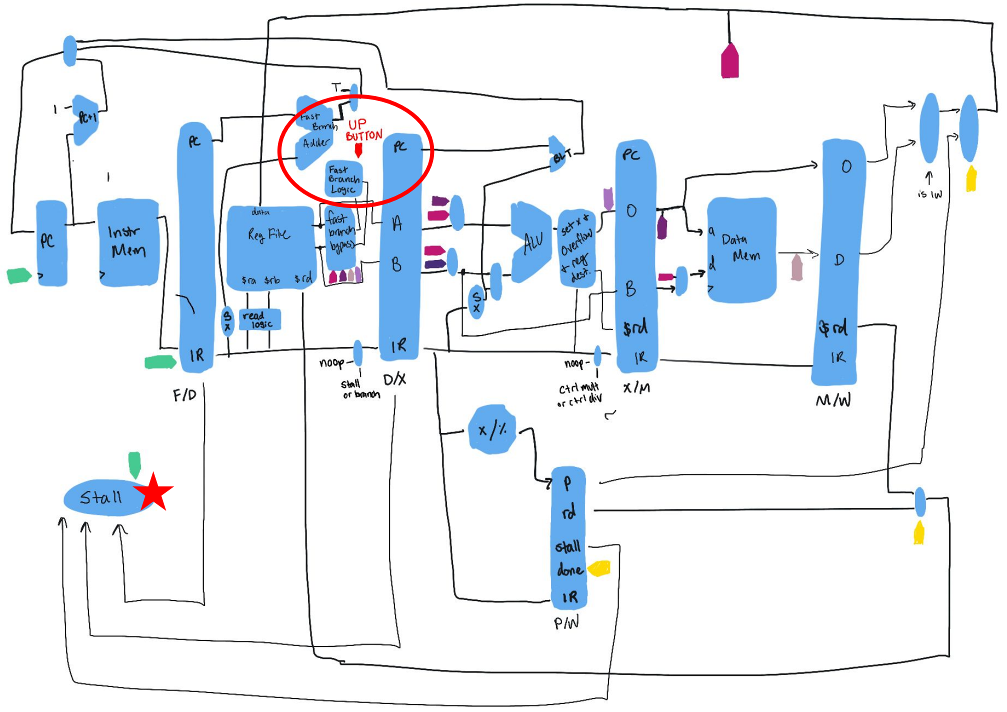
```
assign doFastBranch = (fd_bne && ~(fb_by_a == fb_by_b)) //bne & rd != rs
    | (fd_bex & (| fb_by_a)) //bex and rstatus != 0
    | fd_j | fd_jal | fd_jr
    | (fd_jio & io_jump);
```



Dinosaur Jump: Processor Changes

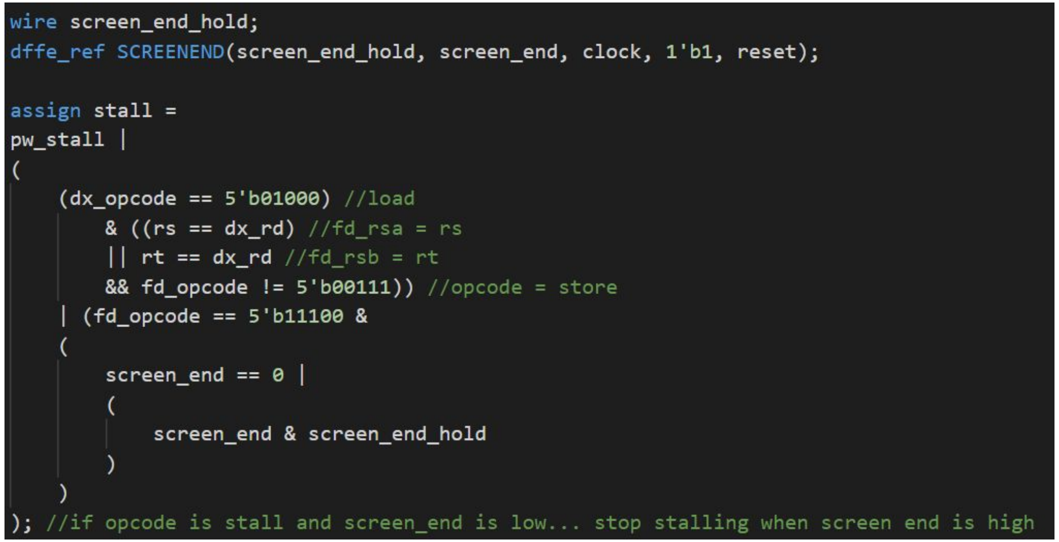
Stall until Screen End

- Instruction: wait
- Opcode: 11100
- Type: Z
- if (screenEnd = 0) stall



Dinosaur Jump: Processor Changes

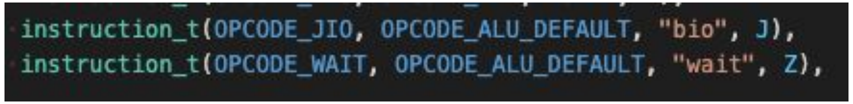
- Stall until posedge screen end
  - Lets previous instructions though
- dffe to detect posedge screenend



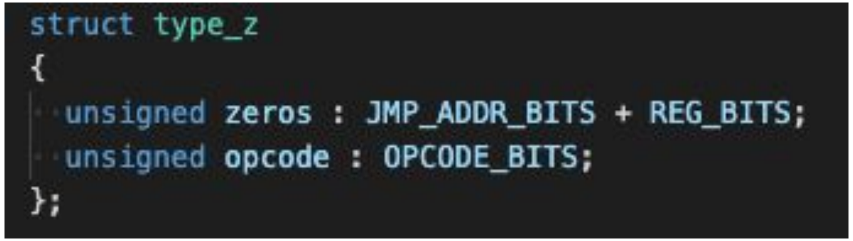
```
wire screen_end_hold;
dffe_ref SCREENEND(screen_end_hold, screen_end, clock, 1'b1, reset);

assign stall =
pw_stall |
(
    (dx_opcode == 5'b01000) //load
        & ((rs == dx_rd) //fd_rsa = rs
        || rt == dx_rd //fd_rsb = rt
        && fd_opcode != 5'b00111)) //opcode = store
    | (fd_opcode == 5'b11100 &
    (
        screen_end == 0 |
        (
            screen_end & screen_end_hold
        )
    )
); //if opcode is stall and screen_end is low... stop stalling when screen end is high
```

Dinosaur Jump: Processor Changes

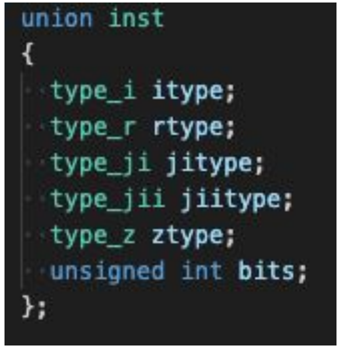
Adding to the assembly compiler



```
instruction_t(OPCODE_JIO, OPCODE_ALU_DEFAULT, "bio", J),
instruction_t(OPCODE_WAIT, OPCODE_ALU_DEFAULT, "wait", Z),
```



```
struct type_z
{
    unsigned zeros : JMP_ADDR_BITS + REG_BITS;
    unsigned opcode : OPCODE_BITS;
};
```



```
union inst
{
    type_i itype;
    type_r rtype;
    type_ji jitype;
    type_jii jiitype;
    type_z ztype;
    unsigned int bits;
};
```

Game Instructions

```
nop (x3)
addi $r1, $r0, 60
addi $r2, $r0, 275

io_loop:
  bio     buttonPress
  j       io_loop
```

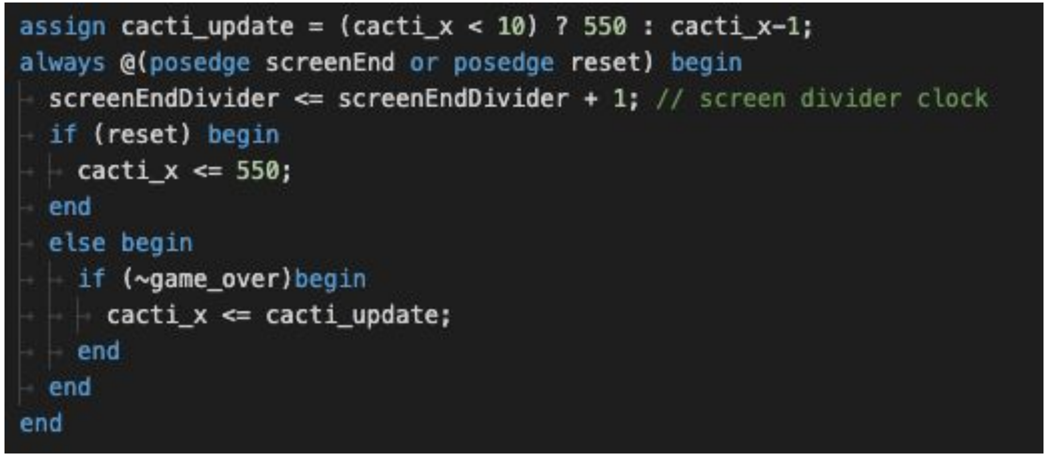
```
buttonPress:
wait (x5)
addi $r2, $r2, -60
wait (x5)
addi $r2, $r2, -40
wait (x5)
addi $r2, $r2, -30
wait (x5)
addi $r2, $r2, -20
wait (x5)
addi $r2, $r2, -10
wait (x5)
addi $r2, $r2, -5
wait (x5)
addi $r2, $r2, 5
```

```
wait (x5)
addi $r2, $r2, 0
wait (x5)
addi $r2, $r2, 10
wait (x5)
addi $r2, $r2, 20
wait (x5)
addi $r2, $r2, 30
wait (x5)
addi $r2, $r2, 40
wait (x5)
addi $r2, $r2, 60
wait (x2)
j io_loop
```

Cacti Scroll

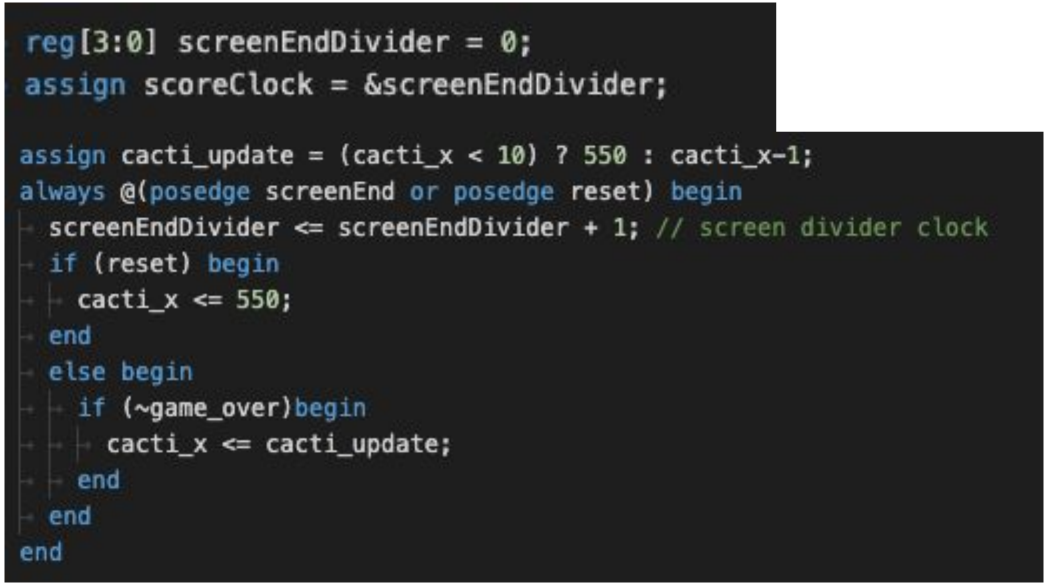
Position

- Variable x
  - -1 per posedge screenEnd
  - Reset to starting position on right
    - If almost off screen (<10) OR new game
- Fixed y



```
assign cacti_update = (cacti_x < 10) ? 550 : cacti_x-1;
always @(posedge screenEnd or posedge reset) begin
  screenEndDivider <= screenEndDivider + 1; // screen divider clock
  if (reset) begin
    cacti_x <= 550;
  end
  else begin
    if (~game_over)begin
      cacti_x <= cacti_update;
    end
  end
end
```

Scoring



```
reg[3:0] screenEndDivider = 0;
assign scoreClock = &screenEndDivider;
```

```
assign cacti_update = (cacti_x < 10) ? 550 : cacti_x-1;
always @(posedge screenEnd or posedge reset) begin
  screenEndDivider <= screenEndDivider + 1; // screen divider clock
  if (reset) begin
    cacti_x <= 550;
  end
  else begin
    if (~game_over)begin
      cacti_x <= cacti_update;
    end
  end
end
```

Clocking

- Score register updates on a screenEnd clock divider
- Counts for 16 screen ends (1111)

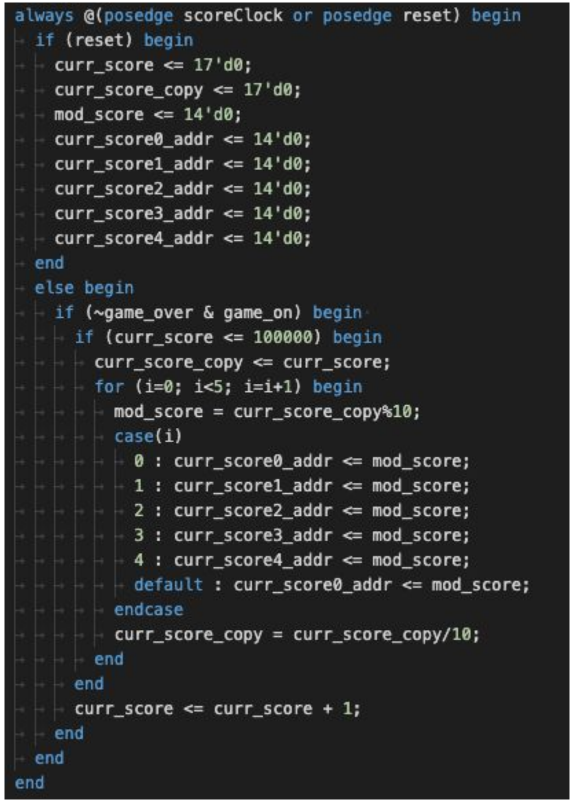
Scoring

Score Register

- 17 bits: up to 99,999
- +1 per posedge of scoreClock

Score by digit [4][3][2][1][0]

- Need to choose score frame from number RAM by digit (curr_score#_addr)
- Loop through with NON-blocking to copy, mod, and then divide
  - Ex. 13%10 = 3 → 13/10 = 1



```
always @(posedge scoreClock or posedge reset) begin
  if (reset) begin
    curr_score <= 17'd0;
    curr_score_copy <= 17'd0;
    mod_score <= 14'd0;
    curr_score0_addr <= 14'd0;
    curr_score1_addr <= 14'd0;
    curr_score2_addr <= 14'd0;
    curr_score3_addr <= 14'd0;
    curr_score4_addr <= 14'd0;
  end
  else begin
    if (~game_over & game_on) begin
      if (curr_score <= 100000) begin
        curr_score_copy <= curr_score;
        for (i=0; i<5; i=i+1) begin
          mod_score = curr_score_copy%10;
          case(i)
            0 : curr_score0_addr <= mod_score;
            1 : curr_score1_addr <= mod_score;
            2 : curr_score2_addr <= mod_score;
            3 : curr_score3_addr <= mod_score;
            4 : curr_score4_addr <= mod_score;
            default : curr_score0_addr <= mod_score;
          endcase
          curr_score_copy = curr_score_copy/10;
        end
      end
      curr_score <= curr_score + 1;
    end
  end
end
```

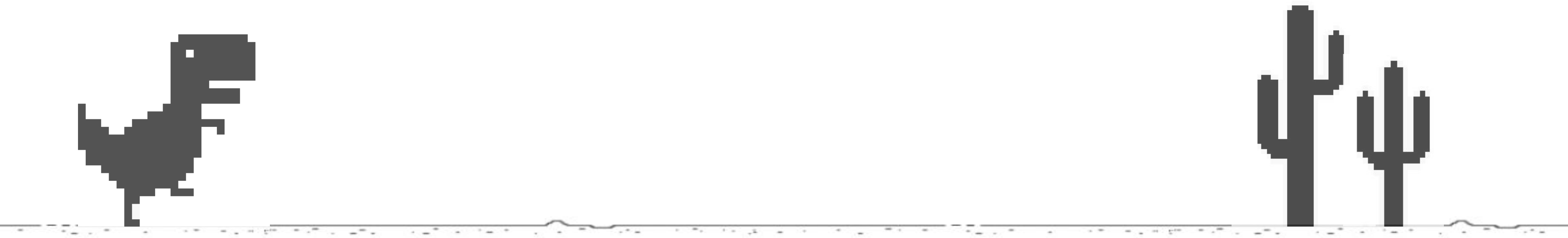
Game Over

- Game ends when dino and cactus collide
    - Collision = pixel overlap
- Write Enable = 0 for Regfile
    - Freeze Dino Position
- Freeze Current Score & Cactus

```
dffe_ref COLLISION(
      game_over,
      ((dinoSquare & dino_data)
            & (cactiSquare & cacti_data)),
      clk,
      ~game_over,
      reset
);
```

HI 00012 00013

Additional Features



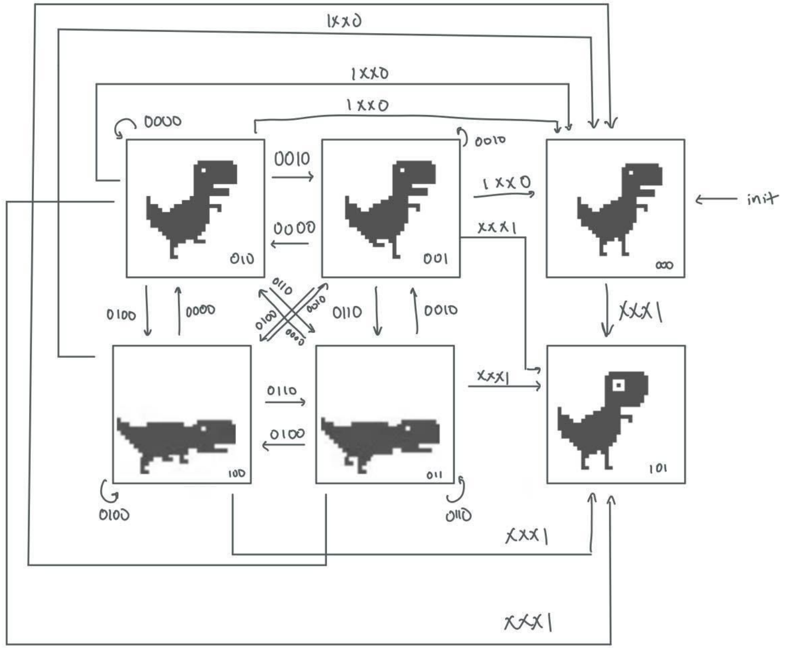
HI 00012 00014

Dinosaur Animation

Concept FSM (Moore)

- Input:

  [dino y != ground] [down] [score's LSB] [game over]
  - Down from bottom button press
- States reflect registered input → Chose to do Mealy Machine



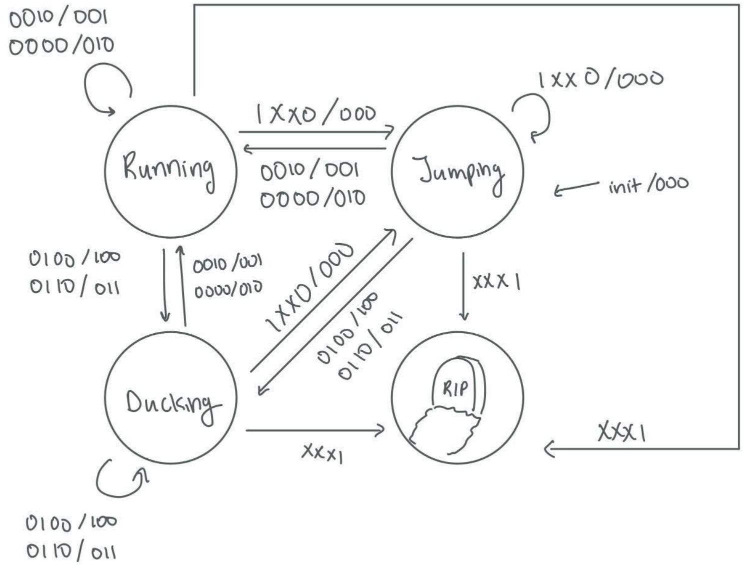
HI 00012 00015

Dinosaur Animation

Actual Frames FSM (Mealy)

- Input:
  [dino y != ground] [down] [score LSB] [game over]

- Inputs are already DFFEs
- Output is dinosaur frame #
  - combinational logic



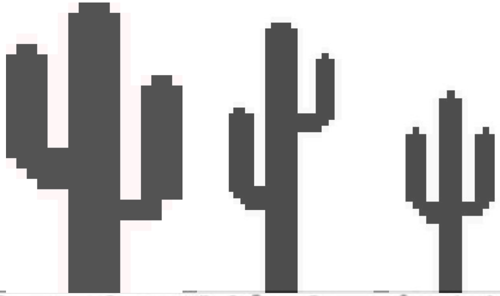
```
// update dinosaur position
 assign dino_frame[0] = (dino_y != 275 | curr_score == 0) ? 0 : curr_score[0];
 assign dino_frame[1] = (dino_y != 275 | curr_score == 0) ? 0 : ((down & curr_score[0]) | (~down & ~curr_score[0]));
 assign dino_frame[2] = (dino_y != 275 | curr_score == 0) ? 0 : (down & ~curr_score[0]);

 assign dino_frame_addr = game_over ?3'd5 : dino_frame;
```

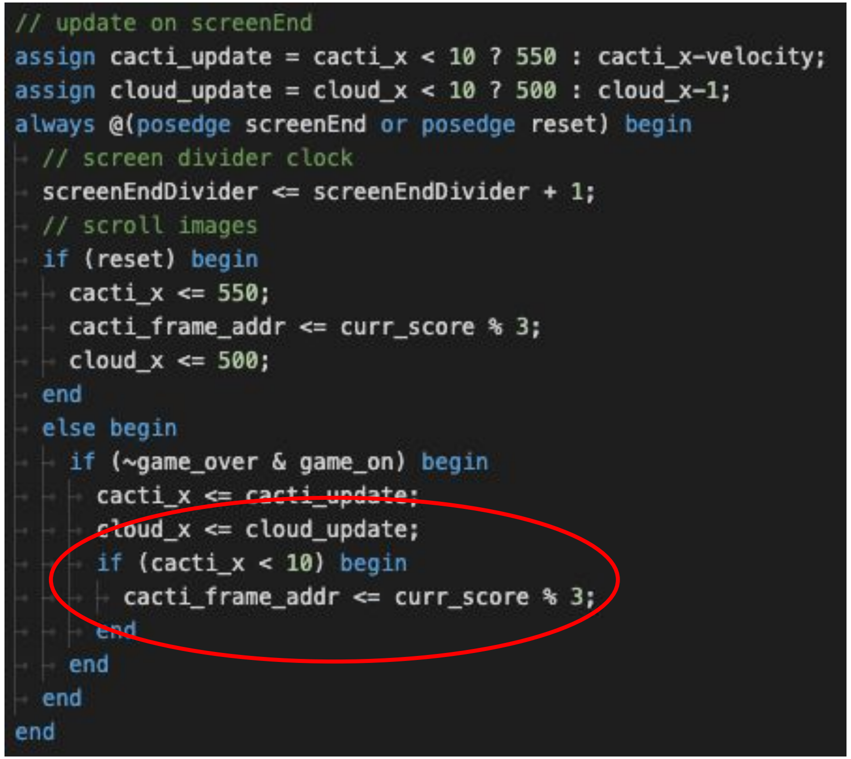
HI 00012 00016

Cacti Heights

“Random” Scrolling Cacti

- Current score dictated which of the three to choose from in RAM (cacti_frame_addr)
    - Only when current cacti finishes scrolling





```
// update on screenEnd
assign cacti_update = cacti_x < 10 ? 550 : cacti_x-velocity;
assign cloud_update = cloud_x < 10 ? 500 : cloud_x-1;
always @(posedge screenEnd or posedge reset) begin
  // screen divider clock
  screenEndDivider <= screenEndDivider + 1;
  // scroll images
  if (reset) begin
    cacti_x <= 550;
    cacti_frame_addr <= curr_score % 3;
    cloud_x <= 500;
  end
  else begin
    if (~game_over & game_on) begin
      cacti_x <= cacti_update;
      cloud_x <= cloud_update;
      if (cacti_x < 10) begin
        cacti_frame_addr <= curr_score % 3;
      end
    end
  end
end
```

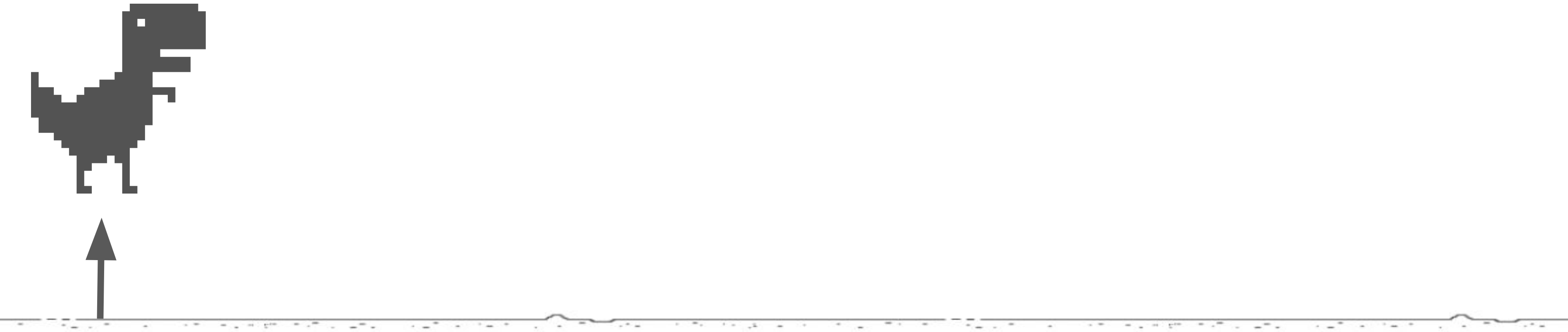
HI 00012 00017

Game Start

- Similar to real game
- Start on the first jump

```
dffe_ref STARTGAME(game_on, up, clk, ~game_on, reset);
```

```
if (~game_over & game_on) begin
      // update score & cactus values for gameplay
end
```

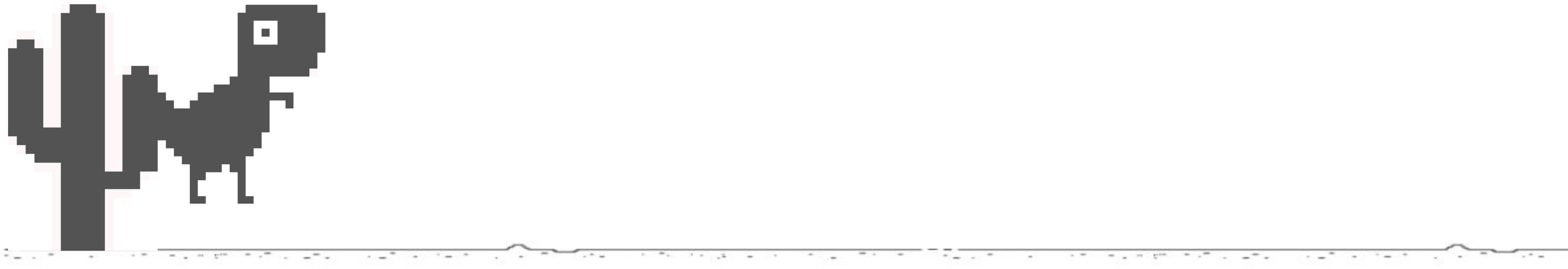


HI 00012 00018

More on Game Over

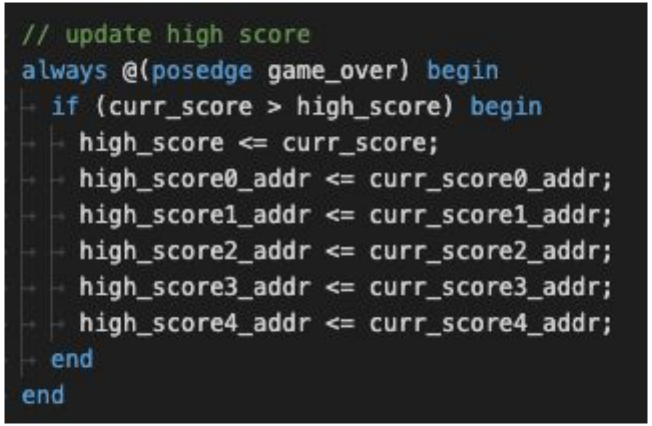
- Still Game Over on Collision
- Write Enable = 0 for
  - Score
  - Dino/Clouds/Cacti Position
- Change dino frame to reflect defeat

```
dffe_ref COLLISION(
      game_over,
      ((dinoSquare & dino_data)
            & (cactiSquare & cacti_data)),
      clk,
      ~game_over,
      reset
);
```



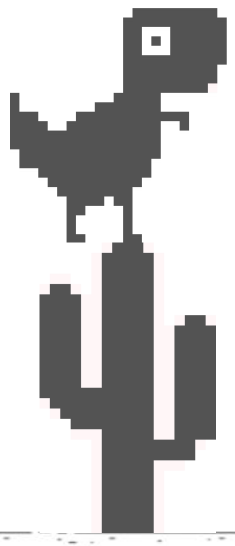
Additionalx2 Features

- Updating high score on game over
  - Floating register that never resets
  - Take bit by bit from curr_score#_addr to choose the correct number from RAM
- Adding light gray clouds
  - Scroll like the cacti but never speeds
- Increasing Cactus Velocity
  - On every level (100, 200, 300 ...)
  - assign velocity = curr_score / 100 + 2;
- Textured background



```
// update high score
always @(posedge game_over) begin
  if (curr_score > high_score) begin
    high_score <= curr_score;
    high_score0_addr <= curr_score0_addr;
    high_score1_addr <= curr_score1_addr;
    high_score2_addr <= curr_score2_addr;
    high_score3_addr <= curr_score3_addr;
    high_score4_addr <= curr_score4_addr;
  end
end
```

HI 00020 00020



GAME OVER