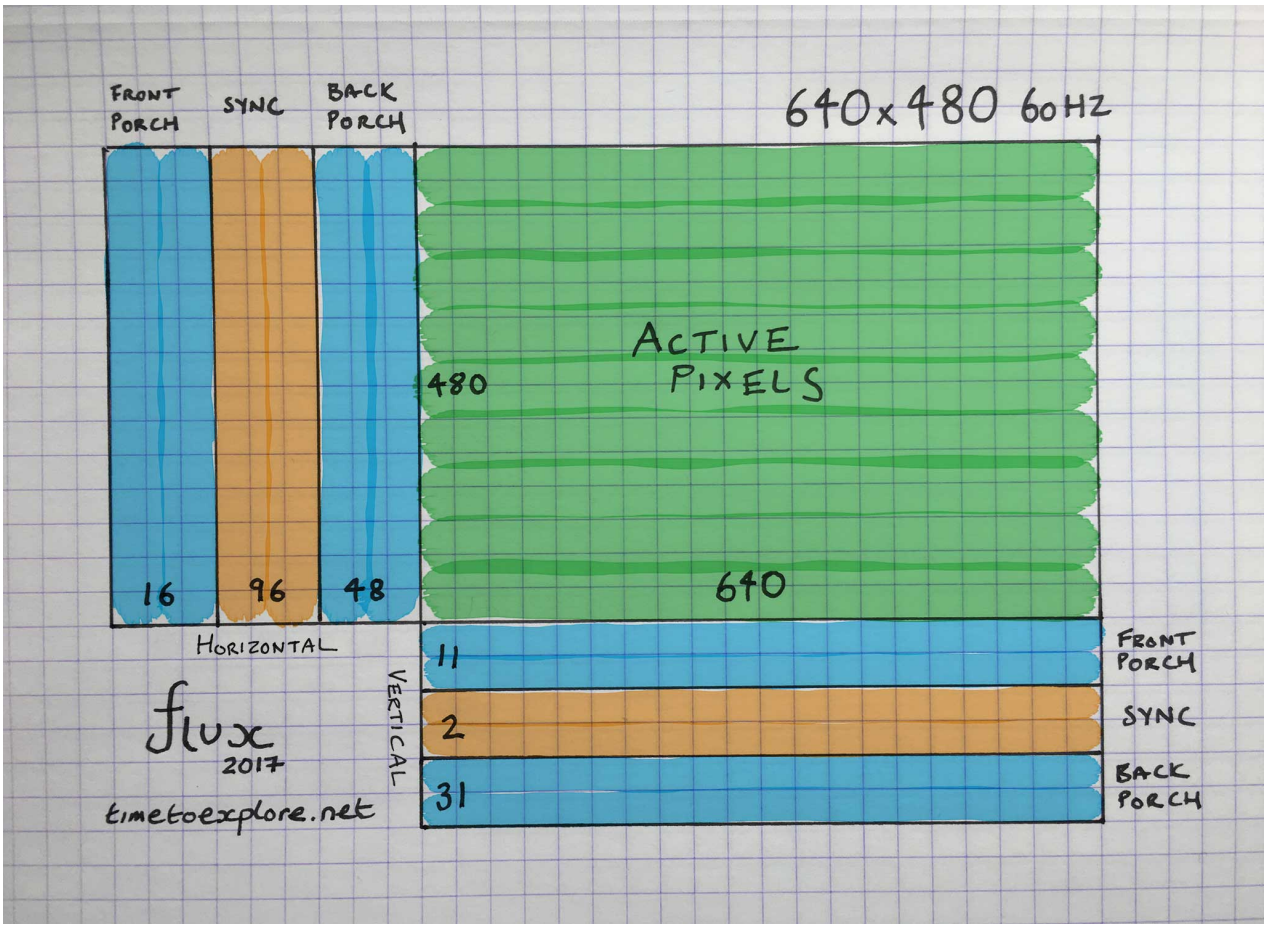
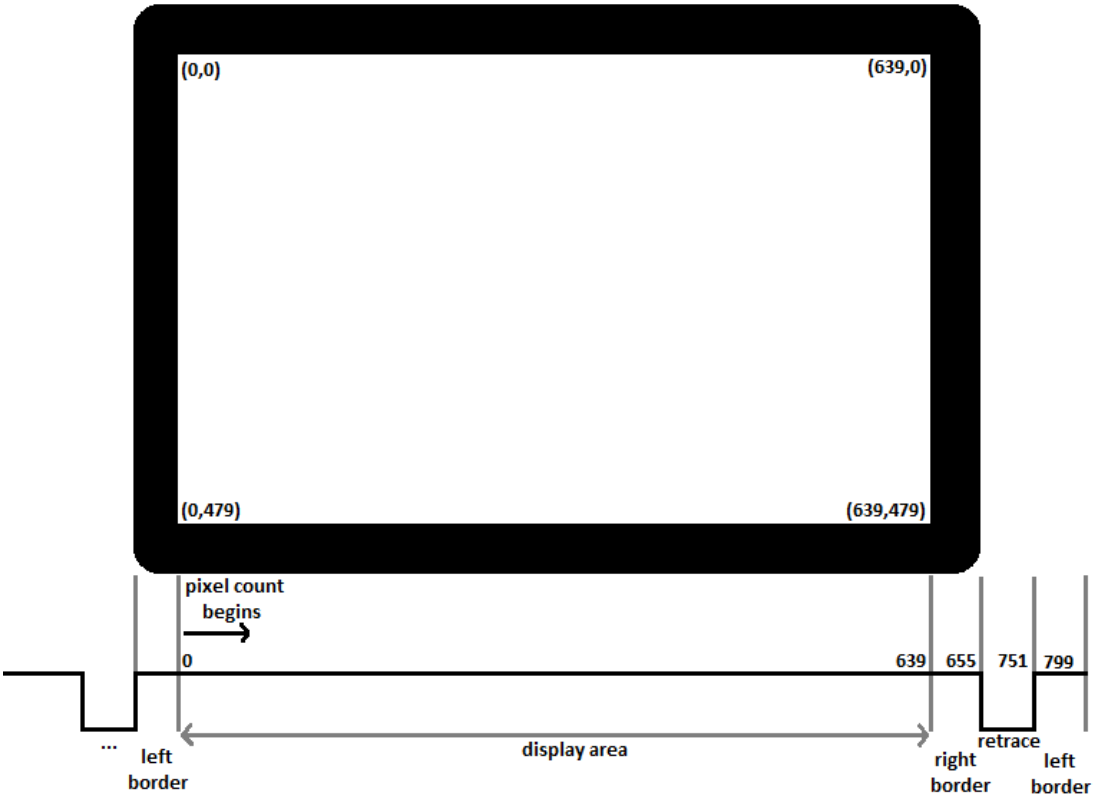
FPGA 보드상에서의 VGA 출력

20190766 윤병준



VGA 스크린은 다음과 같은 구조를 가진다, 전체 크기는 640 * 480 이고, Horizontal, Vertical로 sync 신호를 만들고, 실제 디스플레이 옆으로 `Front porch, sync pulse, back porch` 의 범위를 가진다. 각각 좀 더 쉽게 설명하자면, 스크린의 왼쪽 보더와 오른쪽 보더리고, Sync pulse는 CRT 모니터 시절의 유산이라고 한다.



실제 화면에 대입해보자면 위와 같다. 일반적인 모니터의 주사율인 60Hz 로 모니터를 갱신하려면, 25MHz 의 클락을 사용한다. 이러면 각 픽셀이 스캔되는데 각 1/25MHz= 40ns를 가지고, 한줄에 800 (보더와 싱크를 모두 더한 값이다) * 525 픽셀을 업데이트해야하므로 약 1/60 초의 화면을 갱신할 수 있다. 따라서 프로젝트에서는 25MHz의 클락을 사용 할 것이다. (두번째 그림에 조금 오류가 있는데 top boarder의 범위는 34가 아닌 31이다.)

VGA_Sync 모듈을 작성해보자.

```
`timescale 1 ns / 1 ns
module vga_sync
  (
```

```
    input wire clk, reset,//100 MHz clock generated by DCM clock provided by
Basys3 board
    //hsync, vsync는 현재 업데이트 중인 픽셀이 hsync, vsync 범위일때, active-low하게 한다.
    //video_on은 실제 보여지는 픽셀일때, 변경한다.
    output wire hsync, vsync, video_on,
    output wire [9:0] x, y
  );
  //https://web.mit.edu/6.111/www/s2004/NEWKIT/vga.shtml
  localparam H_ACTIVE       = 640; // horizontal Active video area
  localparam H_BACK_PORCH      =  48; // horizontal back porch
  localparam H_FRONT_PORCH      =  16; // horizontal front porch
  localparam H_SYNC       =  96; // horizontal sync pulse
  localparam H_MAX            = H_ACTIVE + H_BACK_PORCH + H_FRONT_PORCH +
H_SYNC - 1;
  localparam START_H_SYNC = H_ACTIVE + H_FRONT_PORCH;
  localparam END_H_SYNC   = H_ACTIVE + H_FRONT_PORCH + H_SYNC;

  localparam V_ACTIVE       = 480; // vertical display area
  localparam V_FRONT_PORCH      =  11; // vertical front porch
  localparam V_BACK_PORCH      =  31; // vertical back porch
  localparam V_SYNC       =   2; // vertical sync pulse
  localparam V_MAX            = V_ACTIVE + V_FRONT_PORCH + V_B_BORDER +
V_RETRACE - 1;
  localparam START_V_SYNC = V_ACTIVE + V_FRONT_PORCH;
  localparam END_V_SYNC   = V_ACTIVE + V_FRONT_PORCH + V_SYNC;

  reg [9:0] h_count_reg, h_count_next, v_count_reg, v_count_next;
  reg vsync_reg, hsync_reg;
  wire vsync_next, hsync_next;
  reg [2:0] pixel_reg;
  wire pixel_next;
  wire pixel_tick;

  //mod-4 counter possibily replaced by gate level implementation
  //use to divide 100MHz clk to 25 clk
  always @(posedge clk, posedge reset)
    if(reset)
      pixel_reg <= 0;
    else
      pixel_reg <= pixel_next;

  assign pixel_next = pixel_reg + 1; // increase by one

  assign pixel_tick = (pixel_reg == 0)


  always @(posedge clk, posedge reset)
    if(reset)
      begin
```

```
        v_count_reg <= 0;
        h_count_reg <= 0;
        vsync_reg   <= 0;
        hsync_reg   <= 0;
      end
    else
      begin
        v_count_reg <= v_count_next;
        h_count_reg <= h_count_next;
        vsync_reg   <= vsync_next;
        hsync_reg   <= hsync_next;
      end
  always @*
    begin
      //active low signal
      h_count_next = clk? (h_count_reg == H_MAX ? 0 : h_count_reg+1
):h_count_reg;
      v_count_next = clk? (v_count_reg == V_MAX ? 0 : v_count_reg+1
):v_count_reg;
    end

  assign hsync_next = START_H_SYNC <= h_count_reg && h_count_reg < END_H_SYNC;
  assign vsync_next = START_V_SYNC <= v_count_reg && v_count_reg < END_V_SYNC;

  assign x = h_count_reg;
  assign y = v_count_reg;
  assign hsync = hsync_reg;
  assign vsync = vsync_reg;
```

이제 여기서 생성된 x,y 좌표와 원하는 화면의 위치값과 비교하여, 그 비교 값이 참일때, rgb 신호를 만들어, FPGA 보드의 RGB 핀과 연결해주면된다.

Top-down design

Ball position

```
input
  - XU
  - XD
  - YU
  - YD
  - clk
  - rst
output
  - X[9:0]
  - Y[9:0]
```

Collision

```
input
  - ball_x
  - ball_y
  - L_paddle
  - R_paddle
output
  - Cx
  - Cy
  - Score
```

Paddle

```
input
  - U
  - D
  - clk
  - rst
output
  - Y[9:0]
```

Render

```
input
  - ball_x
  - ball_y
  - screen_x
  - screen_y
  - L_paddle
  - L_paddle_rgb
  - R_paddle
  - R_paddle_rgb
  - [11:0] color_paddle, color_ball
  - game_state[1:0]
  - video_on
```

```
  - clk
  - rst
output
  - rgb [11:0]
```

전체적인 구조는 퐁게임 내부의 볼의 이동 방향을 저장하는 T FF와 Ball pos 모듈을 연결합니다. Ball pos와 Paddle pos 모듈의 출력 결과를 이용해 충돌을 연산해주고, Cx, Cy(충돌여부) 는 다시 T FF로 보내줍니다. 페달과 공 모듈의 아웃풋은 따로 빼내어 랜더링 모듈로 보내줍니다. 랜더링 모듈상에서 출력해야하는 객체의 위치를 받고, `vga_sync` 에서 현재 x,y 좌표에 따라서 원하는 x,y 좌표에 위치해 있다면, 원하는 색상 신호를 보내주면 된다.

1. Overall Circuit Structure

게임의 출력 크기는 640 x 480으로 만들 예정이고, 640 픽셀이므로 패들과 공의 위치를 표현하기 위해 10bit를 사용할 예정이다. (ex: y[9:0]) 패들의 세로 길이는 80, 공의 크기는 10이다.

A. Game Controller Module: 게임의 전반적인 진행(게임 플레이, 게임 오버 등)을 담당

B. Paddle Module: 사용자 입력을 받아 패들을 움직인다

Game Controller Module과 Paddle Module의 경우 아래 FSM 부분에서 Input / Output, 그리고 자세한 설명이 있다.

C. Collision Module: 패들과 공의 충돌 여부를 감지한다.

공의 위치와 패들의 위치를 입력 받고 충돌이 일어났는지 감지하는 모듈이다. 패들에 충돌했으면 C_x가, 벽에 충돌했으면 C_y가 1이 되고, 만약 플레이어가 공을 받지 못하면 set_over에 1을 전달한다.

D. Ball Direction Module

iii. Collision 모듈의 Output C_x, C_y를 입력으로 받아 공의 진행 방향 변경을 결정한다. 1이면 Toggle이 일어나고, 0이면 일어나지 않아 진행방향이 유지된다. x y축의 진행 방향이 음수이면 0, 양수이면 1로 설정한다.

E. Ball Position Module

Ball Direction을 입력에 따라 Adder를 이용해 Ball Position에 해당하는 값을 더해 다시 Ball Position을 출력한다.

F. Data-to-screen Module: 주어진 Data들을 VGA를 통해 화면에 출력한다.

ii. Output: Screen에 위의 정보를 VGA 케이블을 통해 출력한다. 이 부분에서는 Behavioral Modeling을 쓸 계획이다.

집중보강전 구현 계획

현재 게임 컨트롤 모듈과 hvsync_generator를 구현 완료 했다.

이후 구현해야할 부분은,

- Ball position
- Collision
- Paddle
- Render
- Top

이렇다. 집중보강 시작하기 전,

박성현이 Ball position, 윤병준이 Render, Paddle을 구현할 계획이다.