香港中文大學
The Chinese University of Hong Kong

CENG3430 Rapid Prototyping of Digital Systems

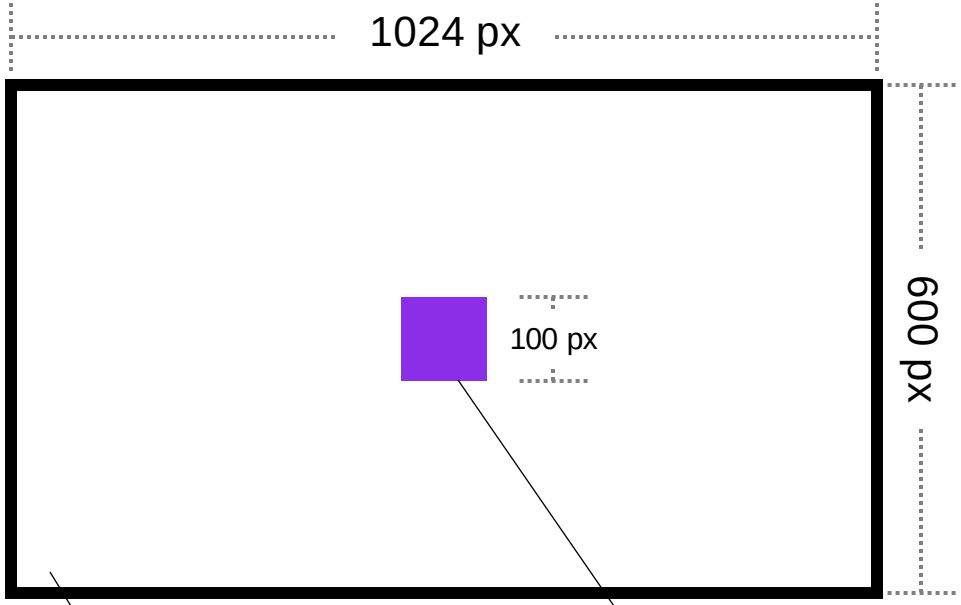
Lab05 Driving VGA Display with ZedBoard

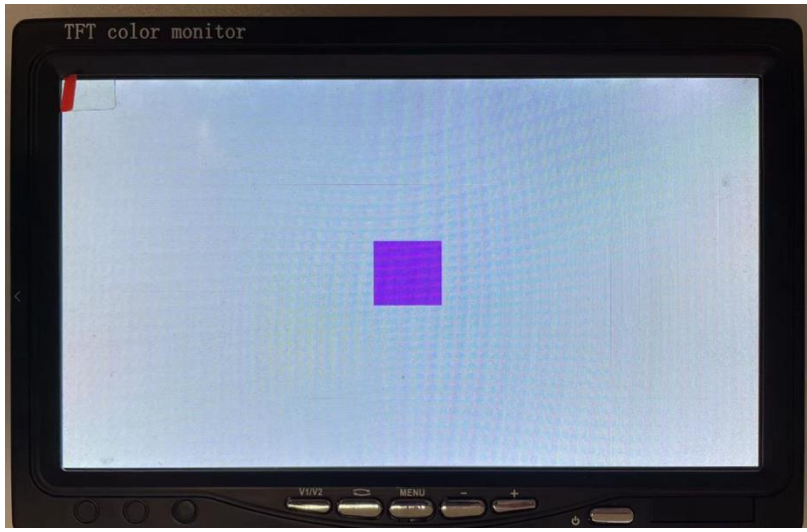
Example

(The task is the extension of this example, please try the example first)

Centered Square

- **Behavior**
  - A purple square is in the center of the white screen.
  - The square side length is 100 pixels (px).



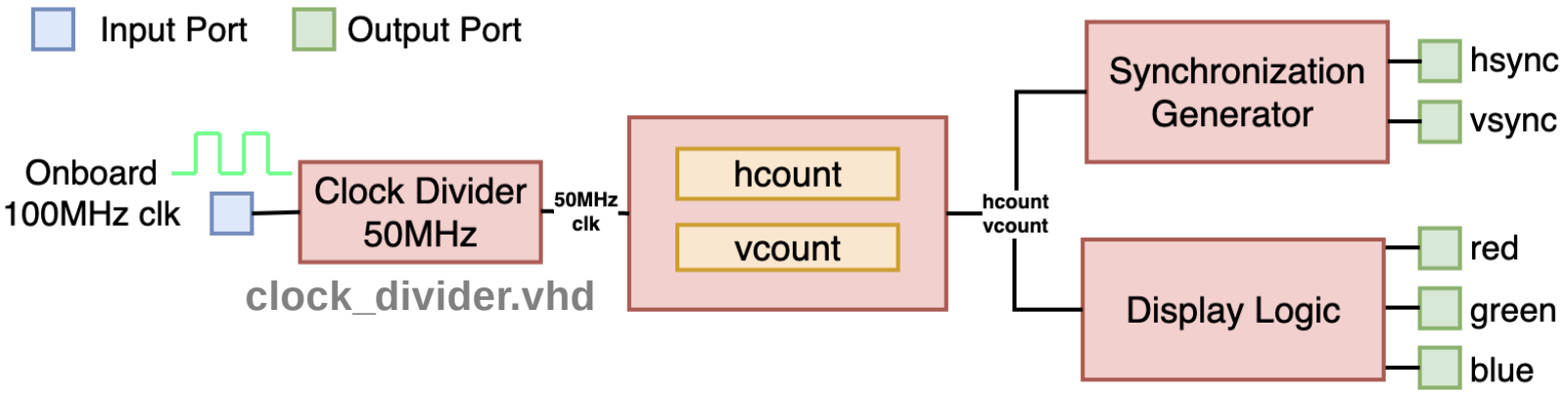



```
-- White
red   <= "1111";
green <= "1111";
blue  <= "1111";
```

```
-- Purple
red   <= "1111";
green <= "0000";
blue  <= "1111";
```

Centered Square

- **Logic Schematic**



lab05.vhd

Step by Step Implementation

lab05.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;

entity vga_driver is
    port (
        clk : in std_logic;
        hsync, vsync : out std_logic;
        red, green, blue : out std_logic_vector(3 downto 0)
    );
end vga_driver;
architecture vga_driver_arch of vga_driver is
    signal clk50MHz : std_logic;
    signal hcount, vcount : integer := 0;
    -- ① row and column constants
begin
    -- ② generate 50MHz clock
    -- ③ horizontal counter
    -- ④ vertical counter
    -- ⑤ generate hsync
    -- ⑥ generate vsync
    -- ⑦ generate RGB signals for 1024x600 display area
end vga_driver_arch;
```

Fill in the input and output port according to the logic schematic

Step by Step Implementation

lab05.vhd

```
entity vga_driver is
    ...

architecture vga_driver_arch of vga_driver is
    ...

-- ① row and column constants
-- row constants
constant H_TOTAL:integer:=1344-1;
constant H_SYNC:integer:=48-1;
constant H_BACK:integer:=240-1;
constant H_START:integer:=48+240-1;
constant H_ACTIVE:integer:=1024-1;
constant H_END:integer:=1344-32-1;
constant H_FRONT:integer:=32-1;
    ...

begin
    ...
end vga_driver_arch;
```

Fill in the row constants

Step by Step Implementation

lab05.vhd

```
entity vga_driver is
    ...

architecture vga_driver_arch of vga_driver is
    ...

-- ① row and column constants
    ...
-- column constants
constant V_TOTAL:integer:=625-1;
constant V_SYNC:integer:=3-1;
constant V_BACK:integer:=12-1;
constant V_START:integer:=3+12-1;
constant V_ACTIVE:integer:=600-1;
constant V_END:integer:=625-10-1;
constant V_FRONT:integer:=10-1;
    ...
begin
    ...
end vga_driver_arch;
```

Fill in the column constants

Step by Step Implementation

clock_divider.vhd

Implement the generic clock divider

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.Numeric_Std.all;
entity clock_divider is
  generic (N : integer);
  port( clk : in std_logic;
        clk_out : out std_logic );
end clock_divider;
architecture arch_clock_divider of clock_divider is
  signal pulse : std_logic := '0';
  signal count : integer := 0;
begin
  process (clk)
  begin
    if rising_edge(clk) then
      if (count = (N - 1)) then
        pulse <= not pulse;
        count <= 0; -- reset count
      else
        count <= count + 1;
      end if;
    end if;
  end process;
  clk_out <= pulse;
end arch_clock_divider;
```

Step by Step Implementation

lab05.vhd

```
entity vga_driver is
    ...

architecture vga_driver_arch of vga_driver is
    ...

    component clock_divider is
    generic (N : integer);
    port (
        clk : in std_logic;
        clk_out : out std_logic
    );
    end component;

begin
-- ② generate 50MHz clock
comp_clk50MHz : clock_divider generic map(N => 1) port map(clk, clk50MHz);
    ...
end vga_driver_arch;
```

Declare the clock divider

Instantiate the module as a 50MHz clock

Step by Step Implementation

lab05.vhd

```
entity vga_driver is
    ...

architecture vga_driver_arch of vga_driver is
    ...
begin
    ...

-- ③ horizontal counter
hcount_proc: process(clk50MHz)
begin
  if( rising_edge(clk50MHz) )
  then
    if(hcount = H_TOTAL) then
      hcount <= 0;
    else
      hcount <= hcount + 1;
    end if;
  end if;
end process hcount_proc;

end vga_driver_arch;
```

Increase hcount at 50MHz

Step by Step Implementation

lab05.vhd

```
entity vga_driver is
    ...

architecture vga_driver_arch of vga_driver is
    ...
begin
    ...

-- ④ vertical counter
vcount_proc: process(clk50MHz)
begin
  if( rising_edge(clk50MHz) ) then
    if(hcount = H_TOTAL) then
      if(vcount = V_TOTAL) then
        vcount <= 0;
      else
        vcount <= vcount + 1;
      end if;
    end if;
  end if;
end process vcount_proc;

end vga_driver_arch;
```

Increase vcount when hcount reaches the end of line.

Step by Step Implementation

lab05.vhd

```
entity vga_driver is
    ...

architecture vga_driver_arch of vga_driver is
    ...
begin
    ...

-- ⑤ generate hsync
hsync_gen_proc: process(hcount) begin
  if(hcount < H_SYNC) then
    hsync <= '0';
  else
    hsync <= '1';
  end if;
end process hsync_gen_proc;

end vga_driver_arch;
```

Step by Step Implementation

lab05.vhd

```
entity vga_driver is
    ...

architecture vga_driver_arch of vga_driver is
    ...
begin
    ...

-- ⑥ generate vsync
vsync_gen_proc: process(vcount)
begin
  if(vcount < V_SYNC) then
    vsync <= '0';
  else
    vsync <= '1';
  end if;
end process vsync_gen_proc;

end vga_driver_arch;
```

Step by Step Implementation

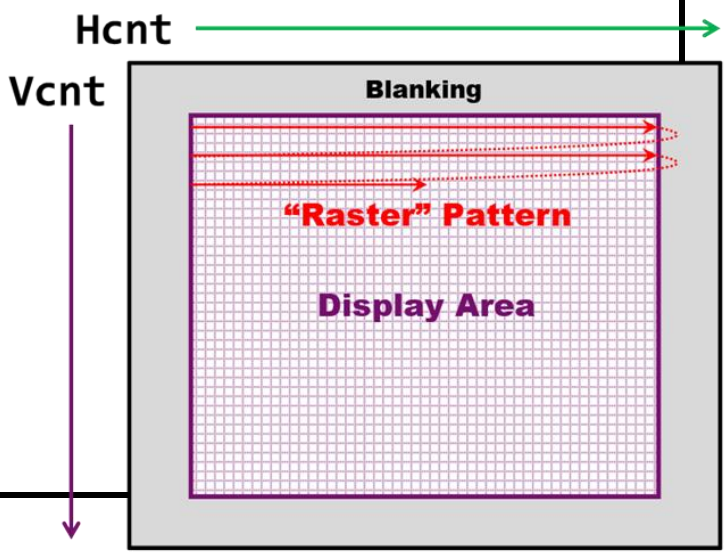
lab05.vhd

```
entity vga_driver is
    ...

architecture vga_driver_arch of vga_driver is
    ...
begin
    ...
-- ⑦ generate RGB signals for 1024x600 display area
data_output_proc: process(hcount, vcount)
begin
  if( (hcount >= H_START and hcount < H_END) and
      (vcount >= V_START and vcount < V_END) ) then
    -- Display Area (draw the square here)
    ... (on the next page)
  else
    -- Blanking Area
    red   <= "0000";
    green <= "0000";
    blue  <= "0000";
  end if;
end process data_output_proc;

end vga_driver_arch;
```

Draw image in Display Area.
Output black in Blanking Area.



Step by Step Implementation

lab05.vhd

```
architecture vga_driver_arch of vga_driver is
-- Constants of the square
constant LENGTH : integer := 100;
constant H_TOP_LEFT : integer := (H_START + H_END)/2 - LENGTH/2;
constant V_TOP_LEFT : integer := (V_START + V_END)/2 - LENGTH/2;
begin
  ...
  if( (hcount >= H_START and hcount < H_END) and
      (vcount >= V_START and vcount < V_END) ) then
    -- Display Area (draw the square here)
    if ((hcount >= H_TOP_LEFT and hcount < H_TOP_LEFT + LENGTH) and
        (vcount >= V_TOP_LEFT and vcount < V_TOP_LEFT + LENGTH)) then
       red <= "1111";
       green <= "0000";
       blue <= "1111";
    else
       red <= "1111";
       green <= "1111";
       blue <= "1111";
    end if;
  else
    ...
end vga_driver_arch;
```

Define the square's side length and position.

Output purple in the square.
Output white out of the square in the display area.

Implementation (Complete Code)

clock_divider.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.Numeric_Std.all;
entity clock_divider is
  generic (N : integer);
  port( clk : in std_logic;
        clk_out : out std_logic );
end clock_divider;
architecture arch_clock_divider of clock_divider is
  signal pulse : std_logic := '0';
  signal count : integer := 0;
begin
  process (clk)
  begin
    if rising_edge(clk) then
      if (count = (N - 1)) then
        pulse <= not pulse;
        count <= 0; -- reset count
      else
        count <= count + 1;
      end if;
    end if;
  end process;
  clk_out <= pulse;
end arch_clock_divider;
```

lab05.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;

entity vga_driver is
  port (
    clk : in std_logic;
    hsync, vsync : out std_logic;
    red, green, blue : out std_logic_vector(3
downto 0)
  );
end vga_driver;

architecture vga_driver_arch of vga_driver is

signal clk50MHz : std_logic;
signal hcount, vcount : integer := 0;
-- ① row and column constants
-- row constants
constant H_TOTAL : integer := 1344 - 1;
constant H_SYNC : integer := 48 - 1;
constant H_BACK : integer := 240 - 1;
constant H_START : integer := 48 + 240 - 1;
constant H_ACTIVE : integer := 1024 - 1;
constant H_END : integer := 1344 - 32 - 1;
constant H_FRONT : integer := 32 - 1;
```

Continue…

Implementation (Complete Code)

```
-- column constants
constant V_TOTAL : integer := 625 - 1;
constant V_SYNC : integer := 3 - 1;
constant V_BACK : integer := 12 - 1;
constant V_START : integer := 3 + 12 - 1;
constant V_ACTIVE : integer := 600 - 1;
constant V_END : integer := 625 - 10 - 1;
constant V_FRONT : integer := 10 - 1;

-- ⑦ constants of the square
constant LENGTH : integer := 100;
constant H_TOP_LEFT:integer:=(H_START+H_END)/2-LENGTH/2;
constant V_TOP_LEFT:integer:=(V_START+V_END)/2-LENGTH/2;


-- ② component declaration
component clock_divider is
  generic (N : integer);
  port (
    clk : in std_logic;
    clk_out : out std_logic
  );
end component;

begin
-- ② generate 50MHz clock
comp_clk50MHz : clock_divider
generic map(N => 1) port map(clk, clk50MHz);
```

Continue…

```
-- ③ horizontal counter


hcount_proc : process (clk50MHz)
begin
  if (rising_edge(clk50MHz))
  then
    if (hcount = H_TOTAL) then
      hcount <= 0;
    else
      hcount <= hcount + 1;
    end if;
  end if;
end process hcount_proc;


-- ④ vertical counter


vcount_proc : process (clk50MHz)
begin
  if (rising_edge(clk50MHz)) then
    if (hcount = H_TOTAL) then
      if (vcount = V_TOTAL) then
        vcount <= 0;
      else
        vcount <= vcount + 1;
      end if;
    end if;
  end if;
end process vcount_proc;
```

Continue…

Implementation (Complete Code)

```
-- ⑤ generate hsync

hsync_gen_proc : process (hcount)
begin
  if (hcount < H_SYNC) then
    hsync <= '0';
  else
    hsync <= '1';
  end if;
end process hsync_gen_proc;


-- ⑥ generate vsync


vsync_gen_proc : process (vcount)
begin
  if (vcount < V_SYNC) then
    vsync <= '0';
  else
    vsync <= '1';
  end if;
end process vsync_gen_proc;



-- ⑦ generate RGB signals for 1024x600


data_output_proc : process (hcount, vcount)
begin
```

Continue…

```
  if ((hcount >= H_START and
       hcount < H_END) and
      (vcount >= V_START and
       vcount < V_END))
  then
    -- Display Area
    if ((hcount >= H_TOP_LEFT and
         hcount < H_TOP_LEFT + LENGTH) and
        (vcount >= V_TOP_LEFT and
         vcount < V_TOP_LEFT + LENGTH))
    then
      red <= "1111";
      green <= "0000";
      blue <= "1111";
    else
      red <= "1111";
      green <= "1111";
      blue <= "1111";
    end if;
  else
    -- Blanking Area
    red <= "0000";
    green <= "0000";
    blue <= "0000";
  end if;
end process data_output_proc;

end vga_driver_arch;
```

Constraints for VGA display

- Write XDC for VGA:

Manual:

1	RED	Red video	V20, U20, V19, V18
2	GREEN	Green video	AB22, AA22, AB21, AA21
3	BLUE	Blue video	Y21, Y20, AB20, AB19
13	HSync	Horizontal sync	AA19
14	VSync	Vertical sync	Y19

Codes:

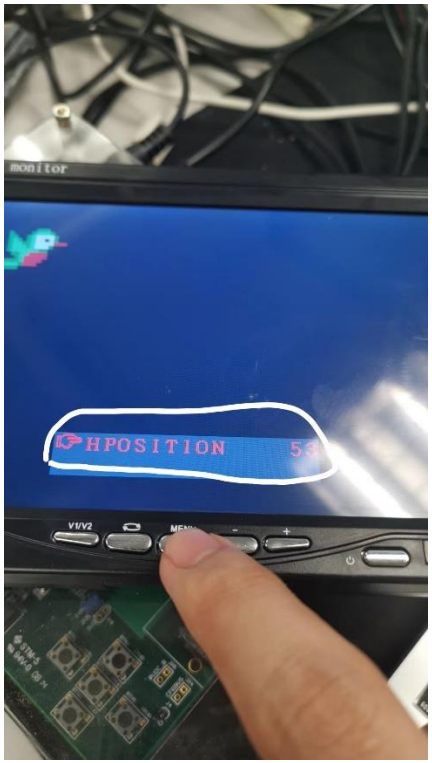
```
set_property PACKAGE_PIN Y21  [get_ports {blue[0]}];  # "VGA-B0"
set_property PACKAGE_PIN Y20  [get_ports {blue[1]}];  # "VGA-B1"
set_property PACKAGE_PIN AB20 [get_ports {blue[2]}];  # "VGA-B2"
set_property PACKAGE_PIN AB19 [get_ports {blue[3]}];  # "VGA-B3"
set_property PACKAGE_PIN AB22 [get_ports {green[0]}];  # "VGA-G0"
set_property PACKAGE_PIN AA22 [get_ports {green[1]}];  # "VGA-G1"
set_property PACKAGE_PIN AB21 [get_ports {green[2]}];  # "VGA-G2"
set_property PACKAGE_PIN AA21 [get_ports {green[3]}];  # "VGA-G3"
set_property PACKAGE_PIN V20  [get_ports {red[0]}];  # "VGA-R0"
set_property PACKAGE_PIN U20  [get_ports {red[1]}];  # "VGA-R1"
set_property PACKAGE_PIN V19  [get_ports {red[2]}];  # "VGA-R2"
set_property PACKAGE_PIN V18  [get_ports {red[3]}];  # "VGA-R3"
set_property PACKAGE_PIN AA19 [get_ports {hsync}];  # "VGA-HS"
set_property PACKAGE_PIN Y19  [get_ports {vsync}];  # "VGA-VS"

# All VGA pins are connected by bank 33, so specified 3.3V together.
set_property IOSTANDARD LVCMOS33 [get_ports -of_objects [get_iobanks 33]];

set_property PACKAGE_PIN Y9 [get_ports {clk}];  # "clk"
set_property IOSTANDARD LVCMOS33 [get_ports -of_objects [get_iobanks 13]];
```

Click here to access the full constraints for Zedboard.

Adjust the Monitor

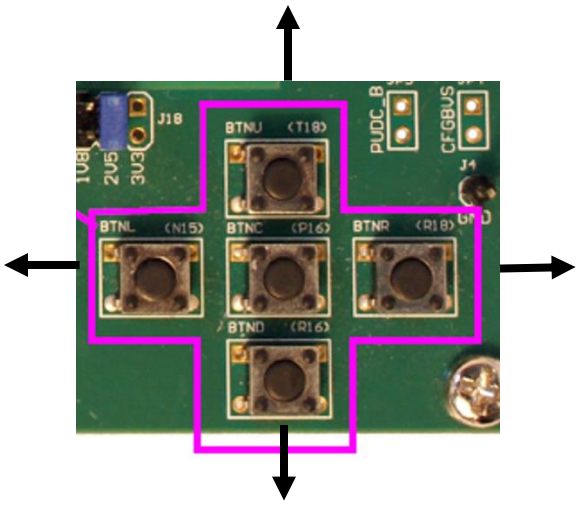


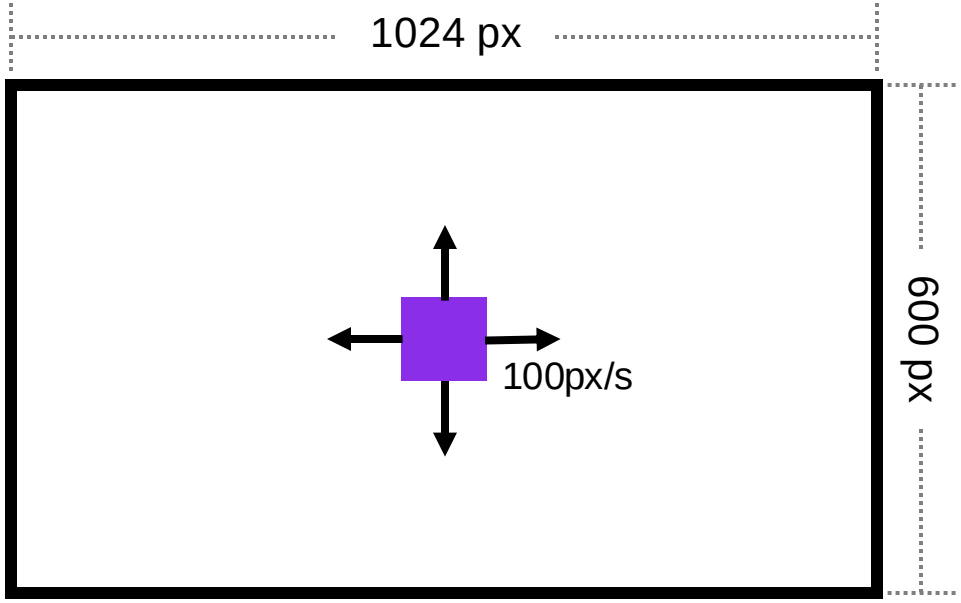
- If the content of your monitor is not fully displayed, then you can press the MENU button to select HPOSITION or VPOSITION, and then use the -/+ buttons to adjust the position of the screen.

Task

Lab05: Moving Square

- **Behavior**
  - Initially, a purple square is in the center of the white screen.
  - The square can move up, down, left, and right, controlled by keeping pressing the corresponding buttons.



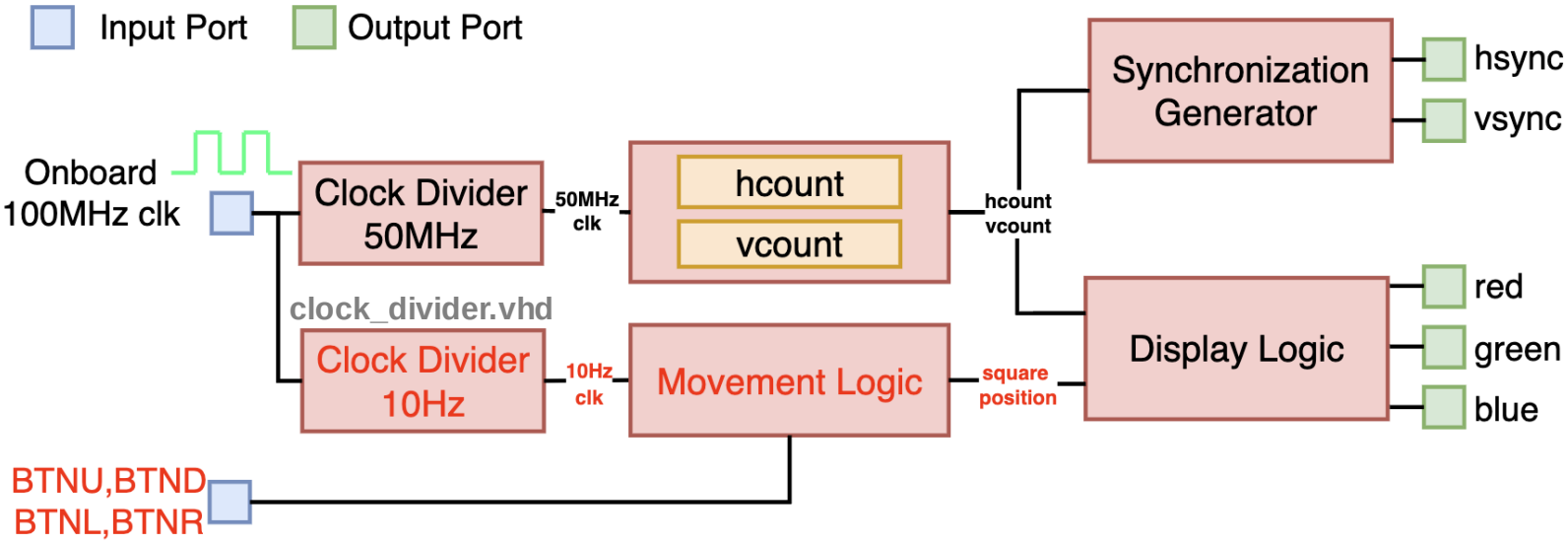


Lab05: Moving Square

- **Behavior in details**
  - The side length of the square is 100 pixels (px).
  - When BTNU, BTND, BTNL or BTNR is pressed, the square should move up, down, left, or right, respectively.
  - The movement speed is set at 100 pixels per second, guaranteeing a seamless and fluid transition as the object moves at a consistent pace of 10 pixels every 100 milliseconds (triggered by 10Hz clock).
  - The square should always stay within the display area, meaning that it should stop moving when it attempts to move out of it.

Lab05: Moving Square

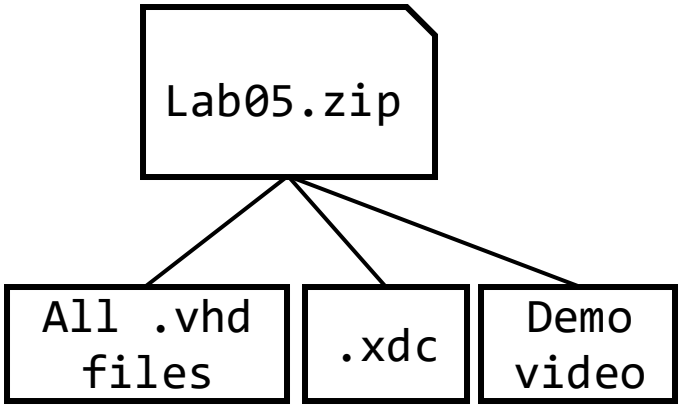
- **Logic Schematic**



lab05.vhd

Submission Guide

- Demo video instruction
  1. Keep pressing BTND
  2. Wait until the square stops
  3. Release the BTND
  4. Keep pressing BTNR
  5. Wait until the square stops
  6. Release the BTNR
  7. Move the square back to the center of the screen by pressing BTNU and then BTNL



- Submit the zip file according to the figure
  - The video should clearly demonstrate Screen and BTNs at the same time (A demo video is provided in the blackboard)
  - Deadline: **12:30 on 26 Feb. 2025**
    - Late submission is NOT acceptable.

Lab05: Hints

1. Change the position of square (`H_TOP_LEFT`, `V_TOP_LEFT`) from constants to signals.
2. Maintain the position signals under 10Hz clock and button pressing signals. This is to say,
   - When the button signals are detected @10Hz clock, increase/decrease position signals.
   - To keep the square in the display area, update the position signals when the new position is valid: all sides are in the display area.
3. When the position signals are changed, the square on the screen changes automatically (read the example code ⑦ **`generate RGB signals for 1024x600 display area`**).

香港中文大學
The Chinese University of Hong Kong

Thank You!