香港中文大學
The Chinese University of Hong Kong

*CENG3430 Rapid Prototyping of Digital Systems*

# Lecture 05: Driving VGA Display with ZedBoard

**Ming-Chang YANG**

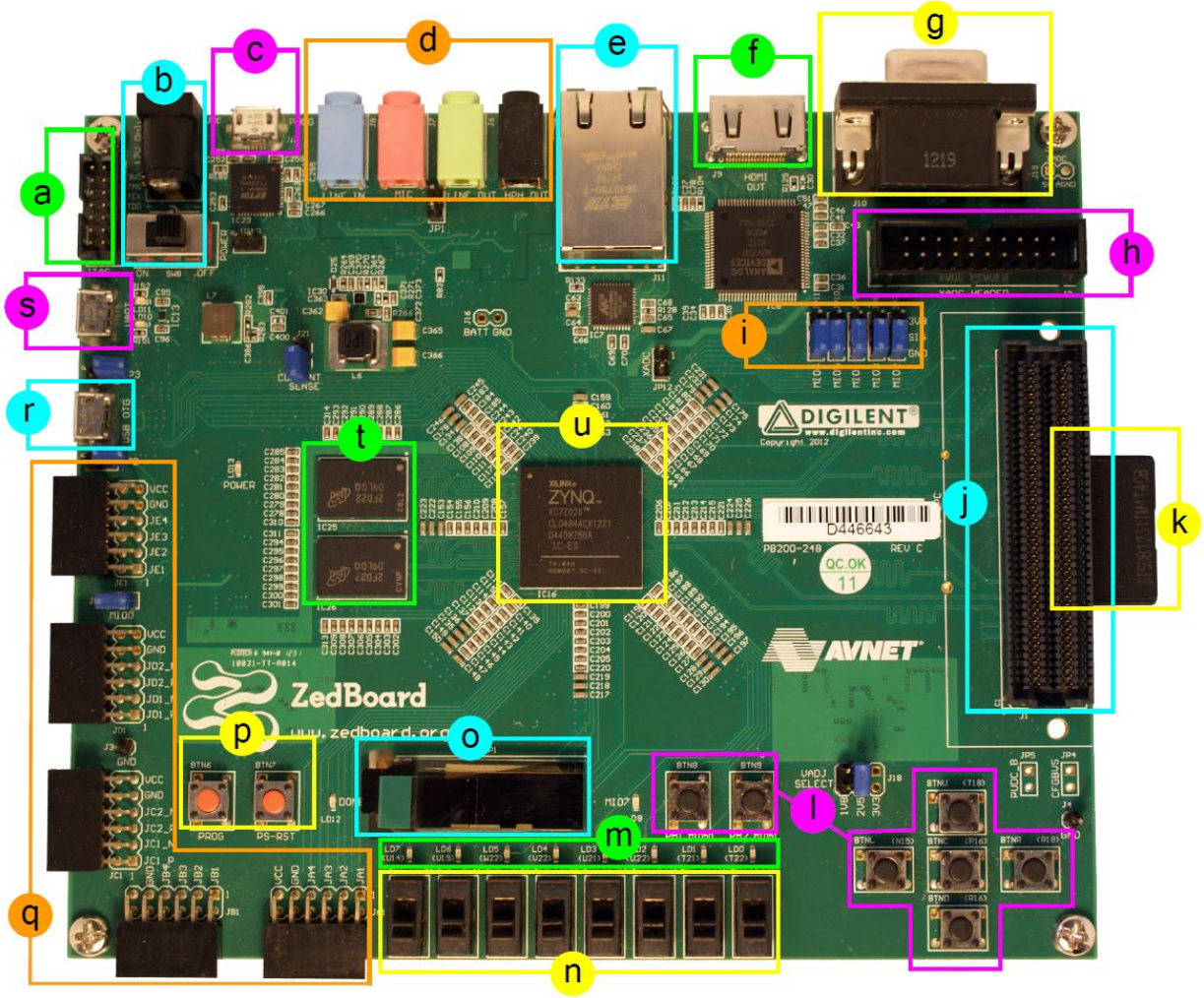
mcyang@cse.cuhk.edu.hk

# Outline

- VGA Basics
  - VGA Connector
  - Pixel-by-Pixel Raster Pattern
  - Timing Specification
- Clock Sources on ZedBoard
  - Clock Divider
- Case Study: 1024x600@60Hz
  - vga_test.vhd
  - vga_test.xdc

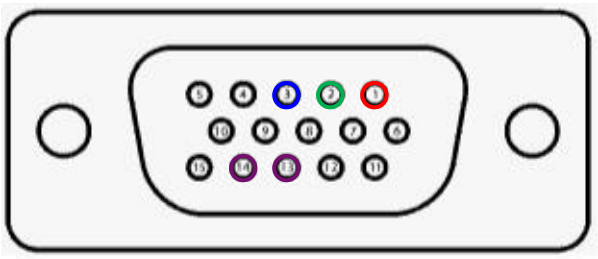
# VGA Connector (1/2)

- ZedBoard allows 12-bit color video output through a **VGA connector** (g), TE 4-1734682-2.



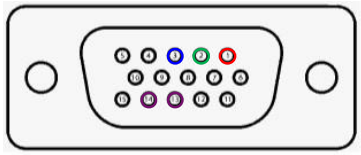
# VGA Connector (2/2)

- The VGA connector comprises *15* pins:
  - **VGA Pin 1~3**: Carry **RED** (4 bits), **GREEN** (4 bits), and **BLUE** (4 bits) signals to control the pixel color;
  - **VGA Pin 13 & 14**: Carry **HSync** (1 bit) and **VSync** (1 bit) signals to control the monitor refresh cycles.

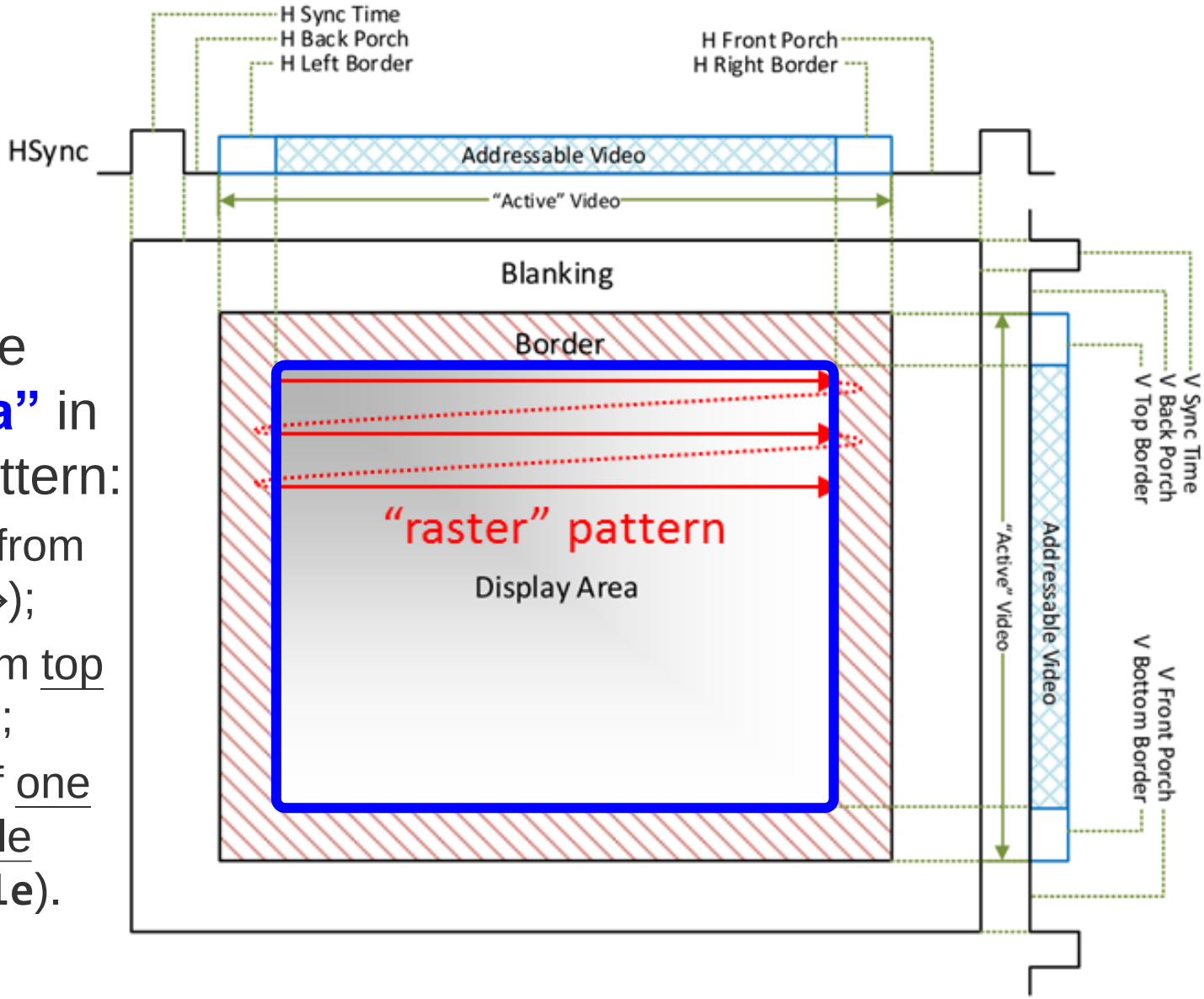


| VGA Pin | Signal | Description | Zynq Pin |
|---|---|---|---|
| 1 | RED | Red video | V20, U20, V19, V18 |
| 2 | GREEN | Green video | AB22, AA22, AB21, AA21 |
| 3 | BLUE | Blue video | Y21, Y20, AB20, AB19 |
| 4 | ID2/RES | formerly Monitor ID bit 2 | NC |
| 5 | GND | Ground (HSync) | NC |
| 6 | RED_RTN | Red return | NC |
| 7 | GREEN_RTN | Green return | NC |
| 8 | BLUE_RTN | Blue return | NC |
| 9 | KEY/PWR | formerly key | NC |
| 10 | GND | Ground (VSync) | NC |
| 11 | ID0/RES | formerly Monitor ID bit 0 | NC |
| 12 | ID1/SDA | formerly Monitor ID bit 1 | NC |
| 13 | HSync | Horizontal sync | AA19 |
| 14 | VSync | Vertical sync | Y19 |
| 15 | ID3/SCL | formerly Monitor ID bit 3 | NC |

# Pixel-by-Pixel Raster Pattern



- Information is showed on the **“display area”** in a **“raster”** pattern:
  - Horizontally from left to right (→);
  - Vertically from top to bottom (↓);
  - At the rate of one pixel per cycle (**`pixel/cycle`**).



# VGA Timing Specification

- VGA displays can accommodate different resolutions:
  - VGA controller circuit needs to dictate the resolution by producing "right" timing signals to control the raster patterns.

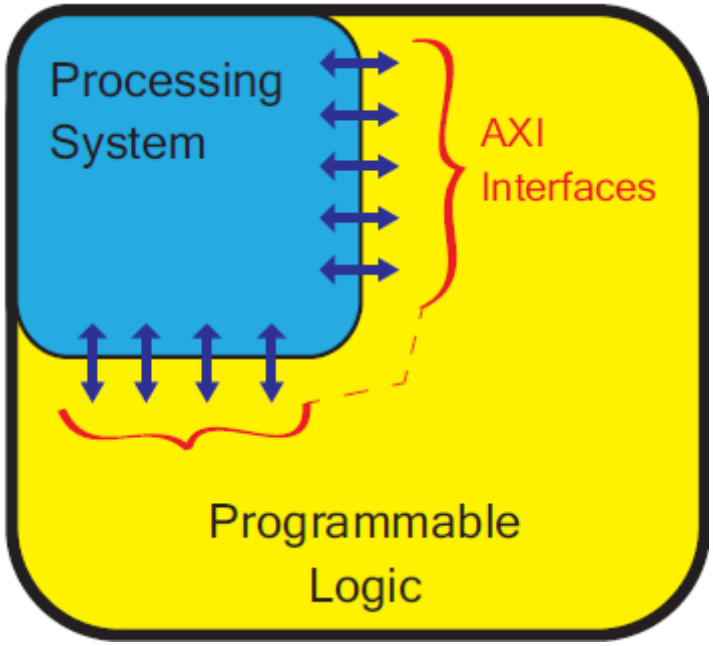
| Format | Pixel Clock (MHz) | Horizontal (in Pixels) | | | | Vertical (in Lines) | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Active Video | Front Porch | Sync Pulse | Back Porch | Active Video | Front Porch | Sync Pulse | Back Porch |
| 640x480, 60Hz | 25.175 | 640 | 16 | 96 | 48 | 480 | 11 | 2 | 31 |
| 640x480, 72Hz | 31.500 | 640 | 24 | 40 | 128 | 480 | 9 | 3 | 28 |
| 640x480, 75Hz | 31.500 | 640 | 16 | 96 | 48 | 480 | 11 | 2 | 32 |
| 640x480, 85Hz | 36.000 | 640 | 32 | 48 | 112 | 480 | 1 | 3 | 25 |
| 800x600, 56Hz | 38.100 | 800 | 32 | 128 | 128 | 600 | 1 | 4 | 14 |
| 800x600, 60Hz | 40.000 | 800 | 40 | 128 | 88 | 600 | 1 | 4 | 23 |
| 800x600, 72Hz | 50.000 | 800 | 56 | 120 | 64 | 600 | 37 | 6 | 23 |
| 800x600, 75Hz | 49.500 | 800 | 16 | 80 | 160 | 600 | 1 | 2 | 21 |
| 800x600, 85Hz | 56.250 | 800 | 32 | 64 | 152 | 600 | 1 | 3 | 27 |
| 1024x768, 60Hz | 65.000 | 1024 | 24 | 136 | 160 | 768 | 3 | 6 | 29 |
| 1024x768, 70Hz | 75.000 | 1024 | 24 | 136 | 144 | 768 | 3 | 6 | 29 |
| 1024x768, 75Hz | 78.750 | 1024 | 16 | 96 | 176 | 768 | 1 | 3 | 28 |
| 1024x768, 85Hz | 94.500 | 1024 | 48 | 96 | 208 | 768 | 1 | 3 | 36 |

# Outline

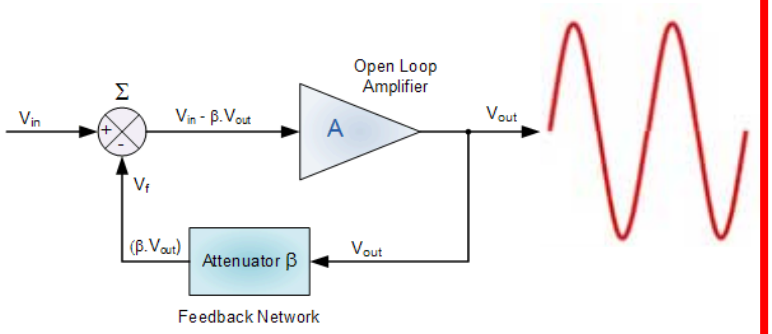
- VGA Basics
  - VGA Connector
  - Pixel-by-Pixel Raster Pattern
  - Timing Specification
- Clock Sources on ZedBoard
  - Clock Divider
- Case Study: 1024x600@60Hz
  - vga_test.vhd
  - vga_test.xdc

# Recall: Clock Sources on ZedBoard (1/2)

- **Processing System**
  - PS subsystem uses a dedicated 33.3333 MHz clock source with series termination.
    - IC18, Fox 767-33.333333-12
  - PS subsystem can generate up to four phase-locked loop (PLL) based clocks for the PL system.




- **Programmable Logic**
  - An on-board 100 MHz oscillator supplies the PL subsystem clock input on bank 13, pin Y9.
    - IC17, Fox 767-100-136




http://zedboard.org/sites/default/files/documentations/ZedBoard_HW_UG_v2_2.pdf
https://www.electronics-tutorials.ws/oscillator/oscillators.html

# Recall: Clock Sources on ZedBoard (2/2)

- To use the on-board 100 MHz clock input on bank 13, pin Y9, you need to include the following in your XDC constraint file:

```
set_property IOSTANDARD LVCMOS33 [get_ports clk]
set_property PACKAGE_PIN Y9 [get_ports clk]
create_clock -period 10 [get_ports clk]
```

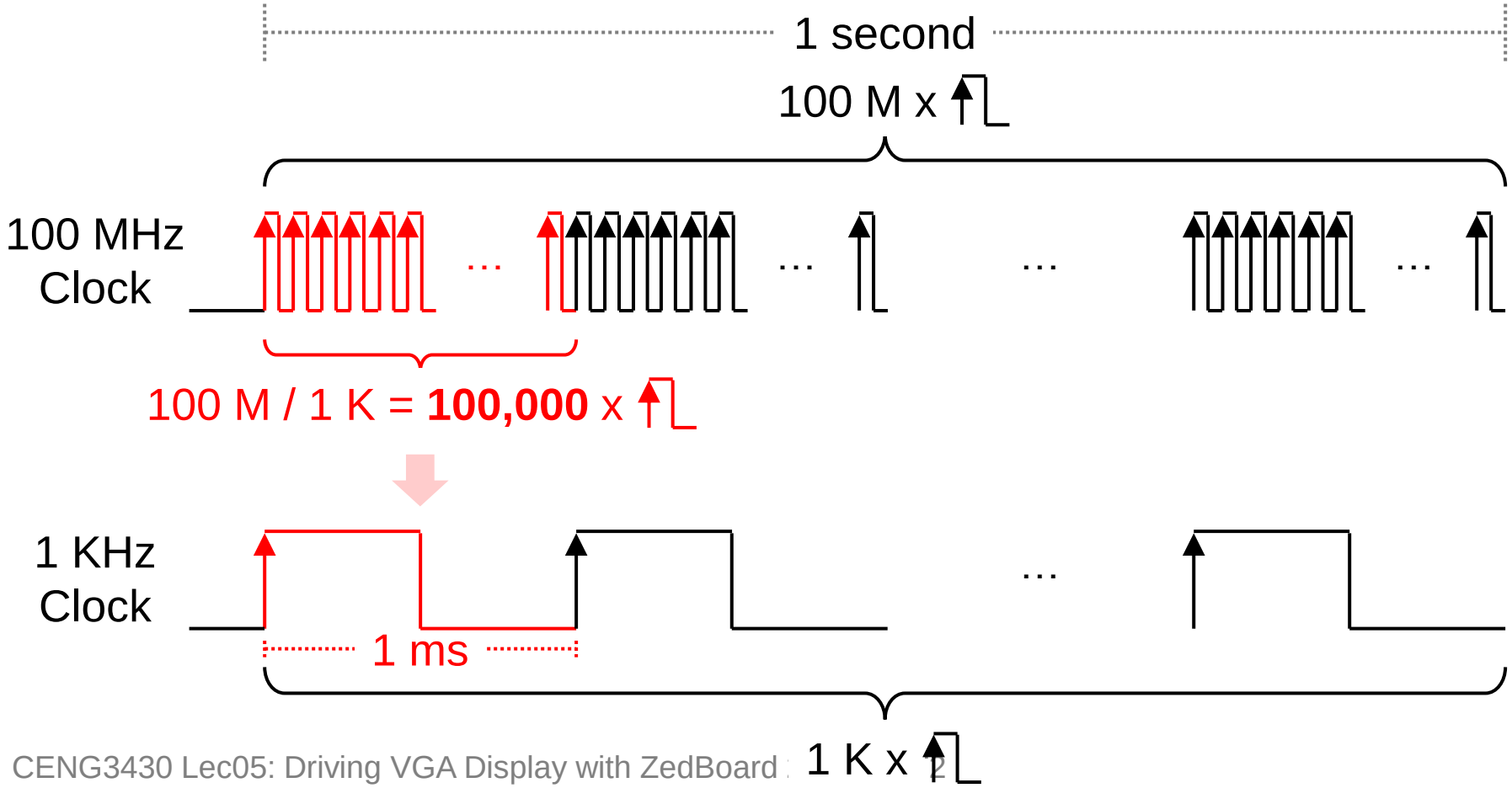
***Note:***

- *The constraint **`-period 10`** is only used to inform the tool that clock period is 10 ns (i.e., 100 MHz).*
- *The constraint **`-period 10`** is **NOT** used specify or generate a different clock period from a given clock source.*

http://zedboard.org/content/changing-frequency-clock-using-createclock

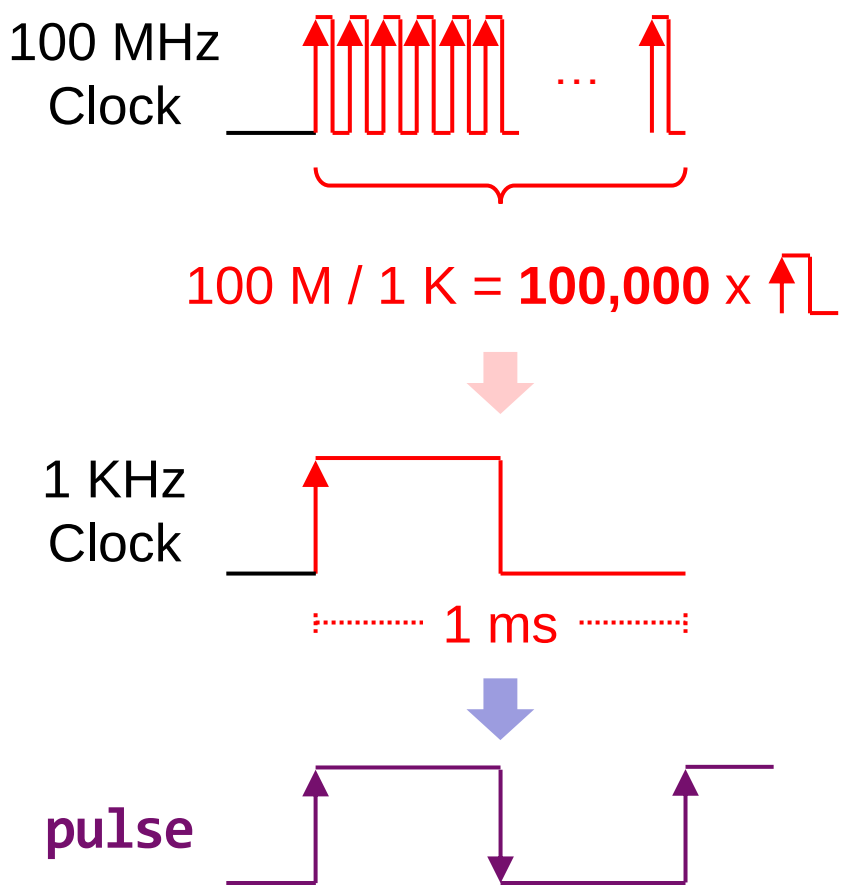
# Clock Divider (1/2)

- In practice, we often need clocks of different rates.
- Example: How to create a 1 KHz clock from the on-board 100 MHz oscillator (**`clk`**)?



# Clock Divider (2/2)

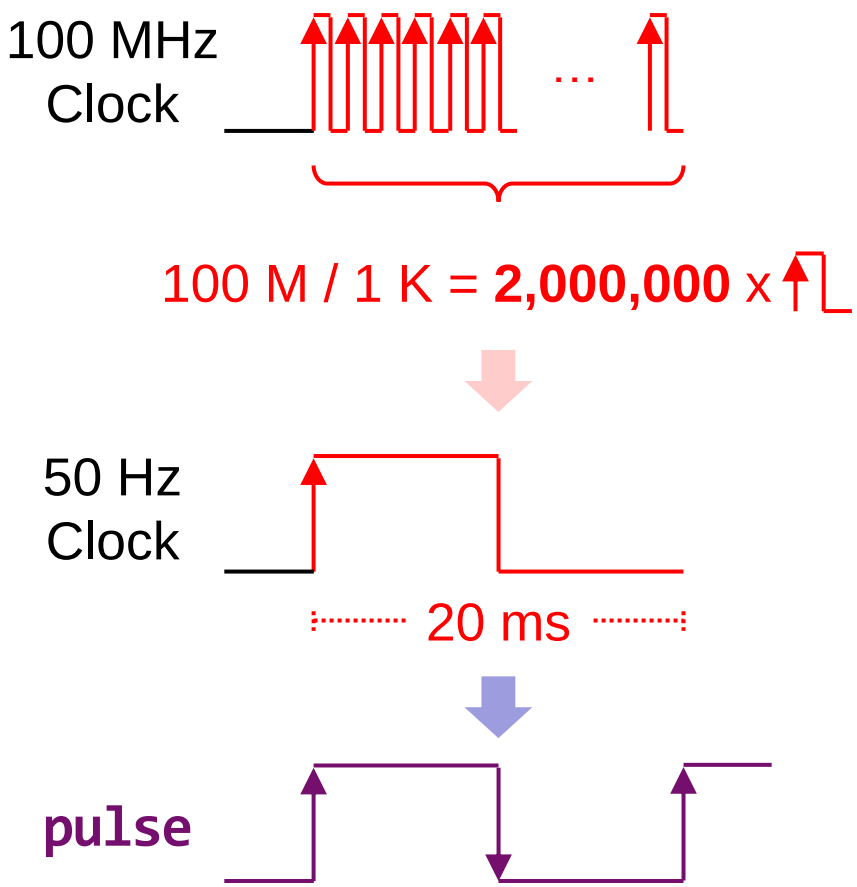
- **Trick**: We can make a counter (**count**) that counts *n* cycles to generate a “slower” clock with a cycle *n*.




```
signal pulse: STD_LOGIC:='0';
signal count: integer:=0;
process(clk)
begin
  if rising_edge(clk) then
    if (count = (50000-1)) then
      pulse <= not pulse;
      count <= 0; -- reset count
    else
      count <= count + 1;
    end if;
  end if;
end process;
```

# Class Exercise 5.1

- Complete the code that creates a 50 Hz clock from the on-board 100 MHz oscillator (**clk**):




```
signal pulse: STD_LOGIC:='0';
signal count: integer:=0;
process(clk)
begin
  if rising_edge(clk) then
    if (count = (_____-1)) then
      pulse <= not pulse;
      count <= 0; -- reset count
    else
      count <= count + 1;
    end if;
  end if;
end process;
```

# Generating Multi-Clocks: Method 1

- **Method 1**: Create entity/process for each of clocks

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.Numeric_Std.all;
entity clk_1hz is
  port( clk : in std_logic;
        clk_out : out std_logic );
end clk_1hz;
architecture arch_clk_1hz of clk_1hz is
  signal pulse : std_logic := '0';
  signal count : integer := 0;
begin
  process (clk)
  begin
    if rising_edge(clk) then
      if (count = (50000000 - 1)) then
        pulse <= not pulse;
        count <= 0; -- reset count
      else
        count <= count + 1;
      end if;
    end if;
  end process;
  clk_out <= pulse;
end arch_clk_1hz;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.Numeric_Std.all;
entity clk_4hz is
  port( clk : in std_logic;
        clk_out : out std_logic );
end clk_4hz;
architecture arch_clk_4hz of clk_4hz is
  signal pulse : std_logic := '0';
  signal count : integer := 0;
begin
  process (clk)
  begin
    if rising_edge(clk) then
      if (count = (12500000 - 1)) then
        pulse <= not pulse;
        count <= 0; -- reset count
      else
        count <= count + 1;
      end if;
    end if;
  end process;
  clk_out <= pulse;
end arch_clk_4hz;
```

**Drawback: Most of the codes are redundant!**

# generic: Key to Parameterized Entity

- In VHDL, you can create a "*parameterized entity*" by including a **generic** clause that lists all supported parameters (i.e., generics) in the entity declaration.

```
generic ( PARA_NAME: <type> [:= <value>] );
```

  - *Note: Default values are optional for generics and can be given in the entity declaration or the component declaration.*
- You can then instantiate a parameterized entity with a **component** instantiation statement in a similar way as instantiating an unparameterized entity.
  - Generics can be set (via **generic map**) in the instantiation.

```
generic map ( PARA_NAME => <value> )
```

# Generating Multi-Clocks: Method 2

- **Method 2**: Use **generic**

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.Numeric_Std.all;
entity generic_ex is
  port( clk : in std_logic );
end generic_ex;
architecture arch_generic_ex of generic_ex is
  signal clk_1, clk_4 : std_logic;
  component clock_divider is
    generic (N : integer);
    port( clk : in std_logic;
          clk_out : out std_logic );
  end component;
begin
  clk_1hz: clock_divider
           generic map (N => 50000000)
           port map(clk, clk_1);
           -- instantiation
  clk_4hz: clock_divider
           generic map(N => 12500000)
           port map(clk, clk_4);
           -- instantiation
end arch_generic_ex;
```

```
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;
```

# Outline

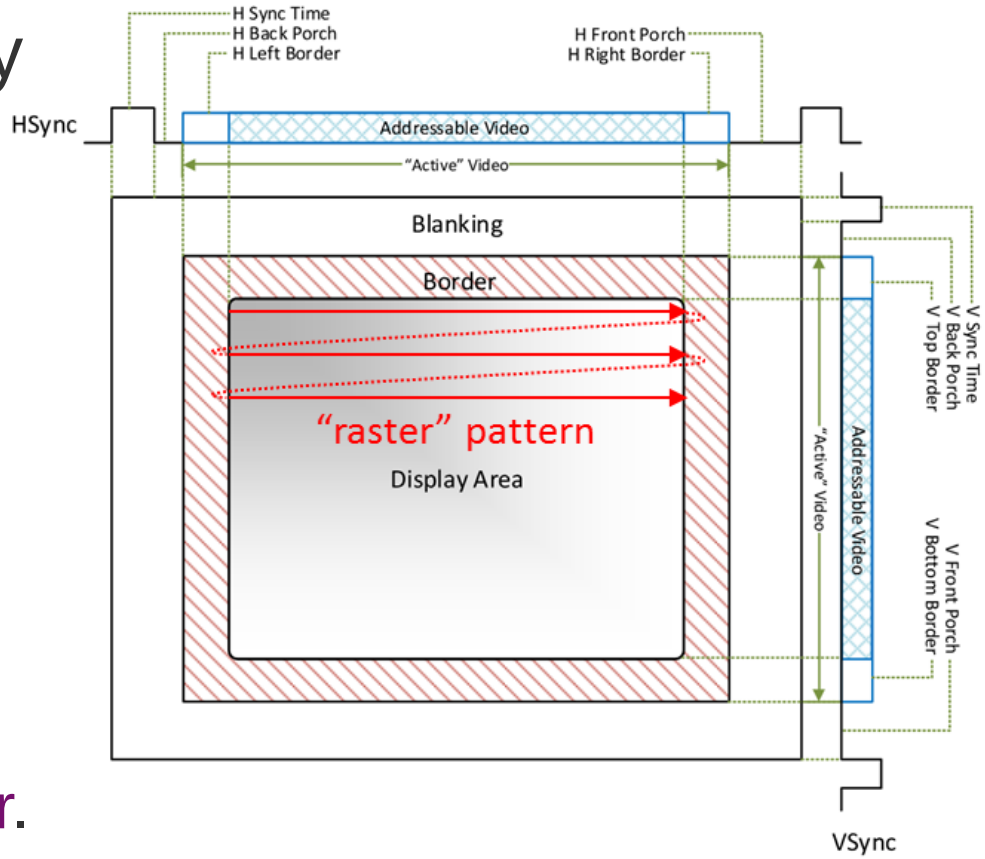
- VGA Basics
  - VGA Connector
  - Pixel-by-Pixel Raster Pattern
  - Timing Specification
- Clock Sources on ZedBoard
  - Clock Divider
- Case Study: 1024x600@60Hz
  - vga_test.vhd
  - vga_test.xdc

# Our VGA: 1024x600@60Hz (1/4)


- **Resolution** (i.e., *displayable area size*): 1024x600 pixels
- **Refreshing Rate** (i.e., *the number of times per second that the image refreshes on the screen*): 60 Hz

# Our VGA: 1024x600@60Hz (2/4)

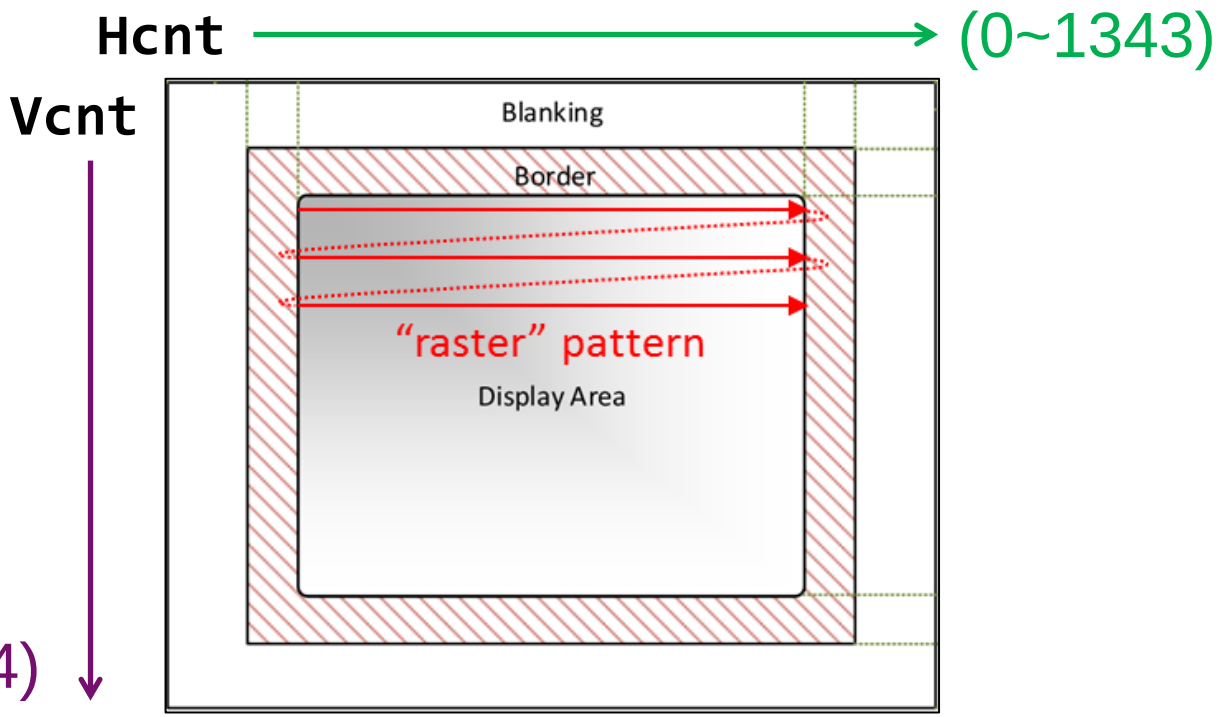
- The pixel clock frequency for 1024x600@60Hz can be 50.00 MHz.
  - Frequency with about ±0.5% accuracy can be typically acceptable.
  - 50.00 MHz can be easily generated using the 100 MHz clock source on ZedBoard via clock divider.




| Format | Pixel Clock (MHz) | Horizontal (in Pixels) | | | | Vertical (in Pixels) | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Active Video | Front Porch | Sync Pulse | Back Porch | Active Video | Front Porch | Sync Pulse | Back Porch |
| 1024x600@60Hz | 50.000 | 1024 | 32 | 48 | 240 | 600 | 10 | 3 | 12 |

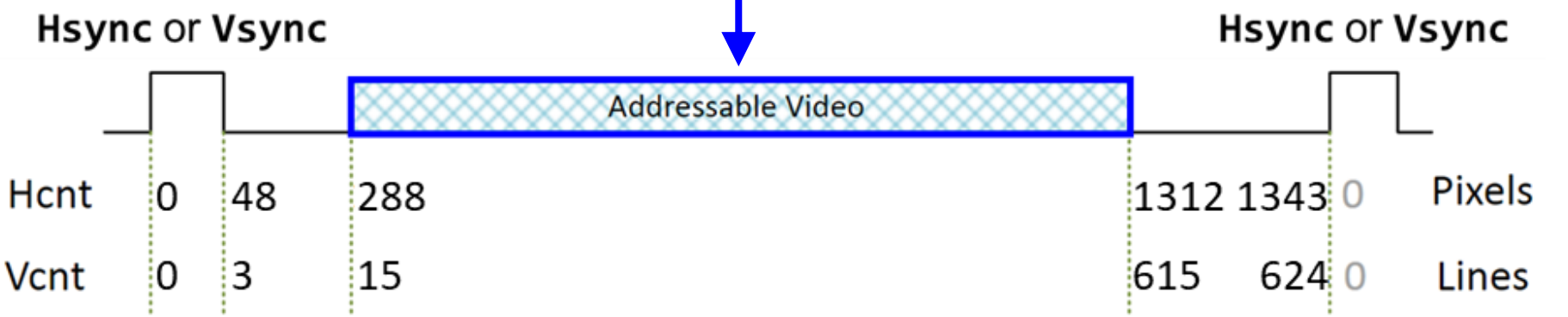
# Our VGA: 1024x600@60Hz (3/4)

- VGA controller circuit usually maintains two counters (**`Hcnt`** and **`Vcnt`**) to control pixel-wise raster patterns:
  - **`Hcnt`** resets itself (when it reaches the end of each line) and increment **`Vcnt`** by 1 to begin a new line.
  - **`Vcnt`** resets itself when it reaches the end of a frame.



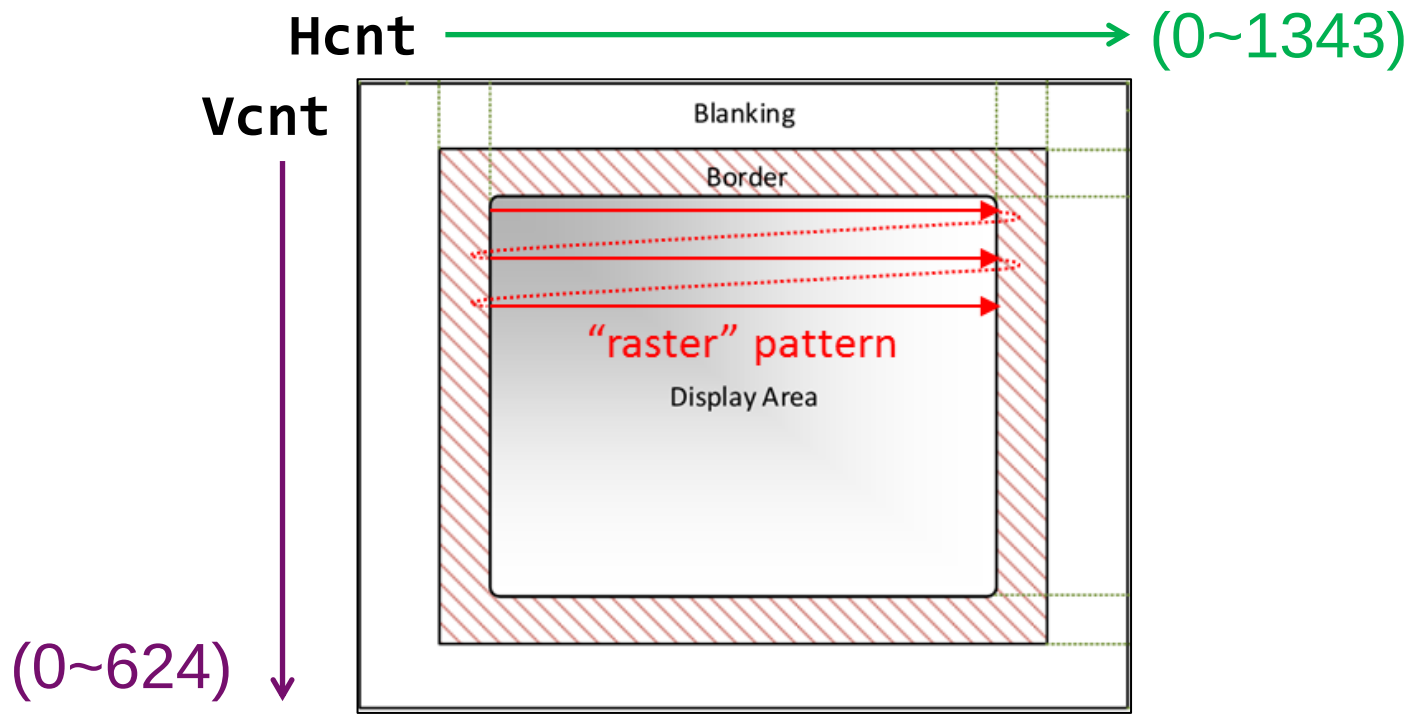
# Our VGA: 1024x600@60Hz (4/4)

- VGA controller circuit must generate the five signals (**RED**, **GREEN**, **BLUE**, **Hsync**, and **Vsync**) as follows:
  - Activate the **HSync** signal when $0 \leq Hcnt < 48$
  - Activate the **VSync** signal when $0 \leq Vcnt < 3$
  - Drive **RED**, **GREEN**, and **BLUE** signals to display the desired pixel colors inside the 1024x600 addressable video area (when $288 \leq Hcnt < 1312\ and\ 15 \leq Vcnt < 615$)
  - Set **RED**, **GREEN**, and **BLUE** signals to **GND** (all zeros) outside the addressable video area



# Class Exercise 5.2

- What is the total number of pixels we need to refresh for the 1024x600@60Hz VGA “in one second”?

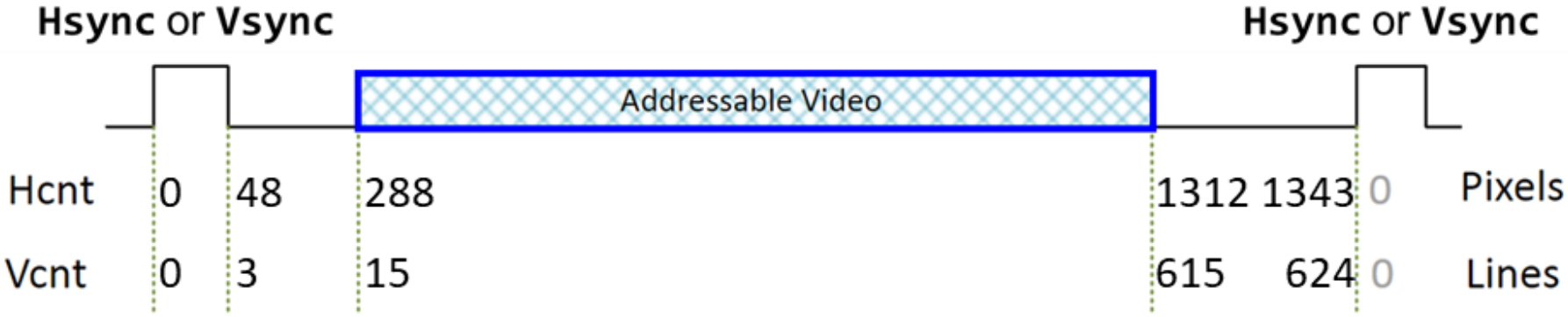


# Example Code: vga_test.vhd

```
entity vga_test is
    port(
        clk100MHz      : in  std_logic;
        hsync,vsync    : out std_logic;
        red,green,blue : out std_logic_vector(3 downto 0)
    );
end vga_test;
architecture vga_test_arch of vga_test is
signal clk50MHz: std_logic;
signal hcount, vcount: std_logic_vector(11 downto 0);
-- ① row and column constants
begin
-- ② generate 50MHz clock
-- ③ horizontal counter
-- ④ generate hsync
-- ⑤ vertical counter
-- ⑥ generate vsync
-- ⑦ generate RGB signals for 1024x600 display area
end vga_test_arch;
```

# ① row and column constants

```
-- ① row and column constants
-- row constants                          -- column constants
constant H_TOTAL:integer:=1344-1;  constant V_TOTAL:integer:=625-1;
constant H_SYNC:integer:=48-1;     constant V_SYNC:integer:=3-1;
constant H_BACK:integer:=240-1;    constant V_BACK:integer:=12-1;
constant H_START:integer:=48+240-1;constant V_START:integer:=3+12-1;
constant H_ACTIVE:integer:=1024-1; constant V_ACTIVE:integer:=600-1;
constant H_END:integer:=1344-32-1; constant V_END:integer:=625-10-1;
constant H_FRONT:integer:=32-1;    constant V_FRONT:integer:=10-1;
```




| Format | Pixel Clock (MHz) | Horizontal (in Pixels) | | | | Vertical (in Pixels) | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Active Video | Front Porch | Sync Pulse | Back Porch | Active Video | Front Porch | Sync Pulse | Back Porch |
| 1024x600@60Hz | 50.000 | 1024 | 32 | 48 | 240 | 600 | 10 | 3 | 12 |

# Recall: Constant Object (Lec01)

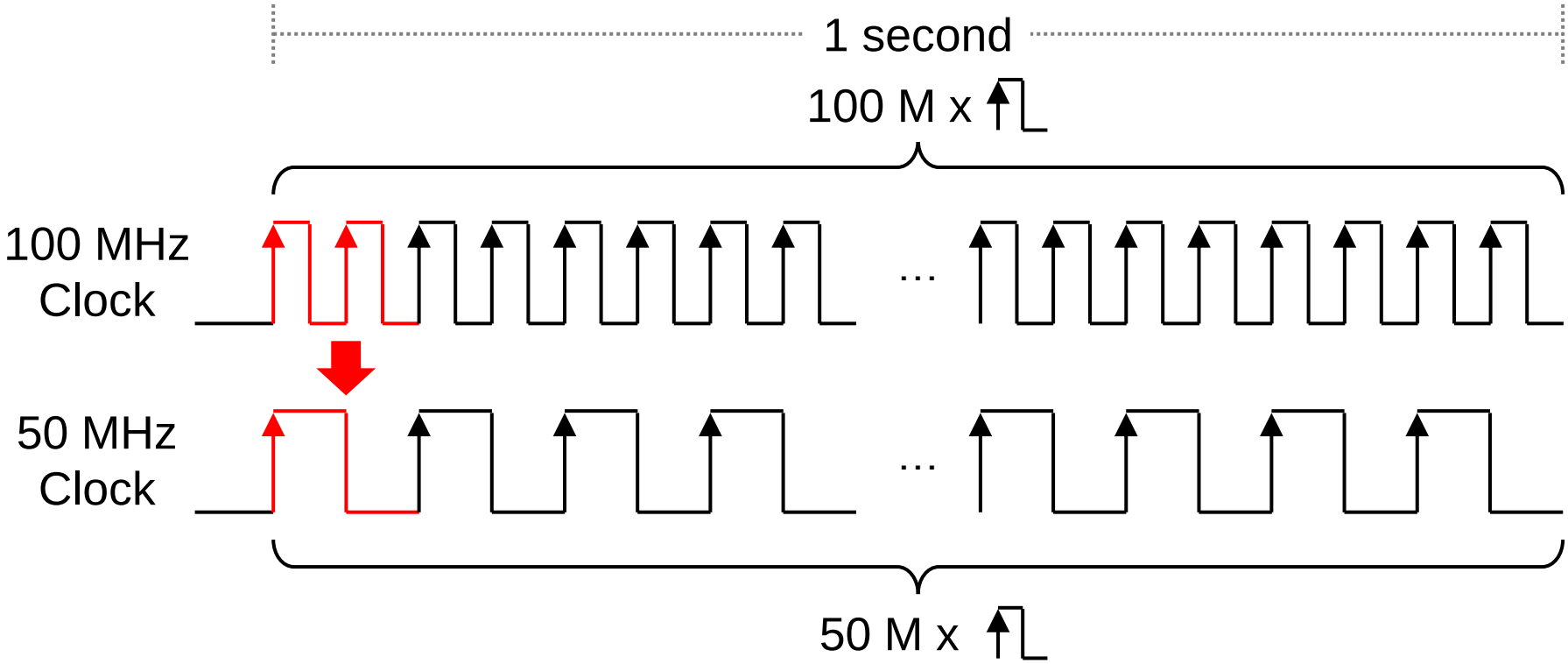
```
constant CONST_NAME: <type> := <value>;
```

Note: Constants must be declared with initialized values.

- Examples:
  - `constant CONST_NAME: STD_LOGIC := 'Z';`
  - `constant CONST_NAME: STD_LOGIC_VECTOR (3 downto 0) := "0-0-";`
    - `'-'` means “don’t care”
- Constants can be declared in
  - Anywhere allowed for declaration.

# ② generate 50MHz clock

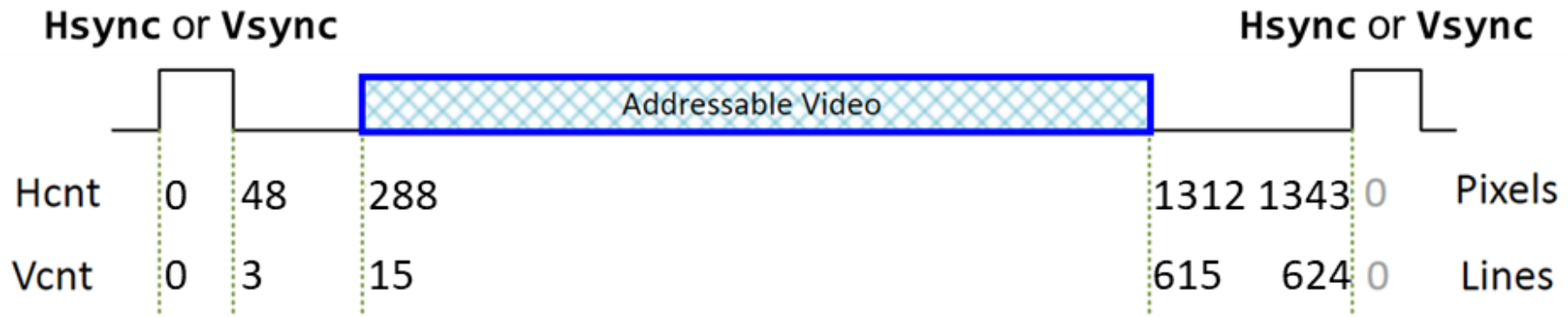
```
-- ② generate 50MHz clock
vga_clk_gen_proc1: process(clk100MHz)
begin
    if( rising_edge(clk100MHz) ) then
        clk50MHz <= not clk50MHz;
    end if;
end process vga_clk_gen_proc1;
```



# ③ horizontal counter & ④ gen. hsync

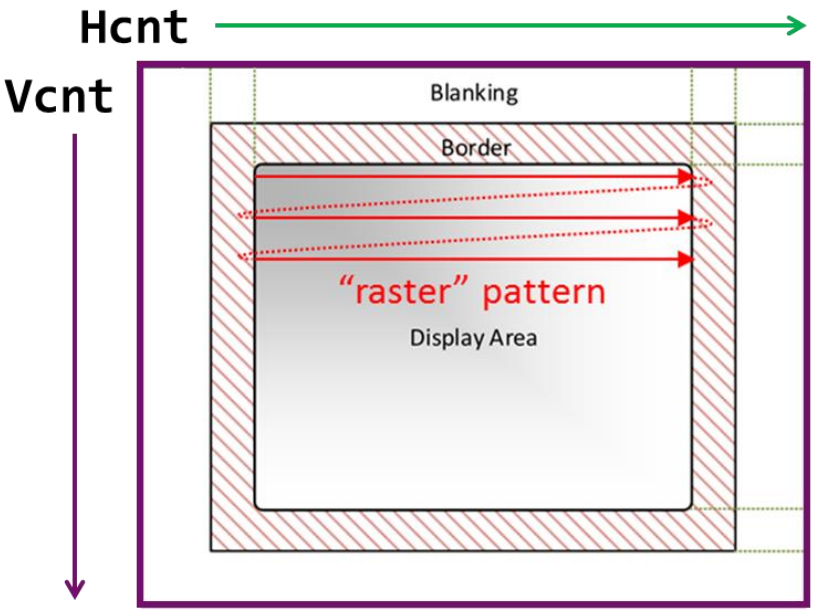
```
-- ③ horizontal counter
pixel_count_proc:
process(clk50MHz)
begin
  if( rising_edge(clk50MHz) )
  then
    if(hcount = H_TOTAL) then
      hcount <= (others => '0');
    else
      hcount <= hcount + 1;
    end if;
  end if;
end process pixel_count_proc;
```

```
-- ④ generate hsync
hsync_gen_proc: process(hcount)
begin
  if(hcount <= H_SYNC) then
    hsync <= '1';
  else
    hsync <= '0';
  end if;
end process hsync_gen_proc;
```



# ⑤ vertical counter & ⑥ generate vsync

```
-- ⑤ vertical counter
line_count_proc: process(clk50MHz)
begin
  if( rising_edge(clk50MHz) ) then
    if(hcount = H_TOTAL) then
      if(vcount = V_TOTAL) then
        vcount <= (others => '0');
      else
        vcount <= vcount + 1;
      end if;
    end if;
  end if;
end process line_count_proc;
```

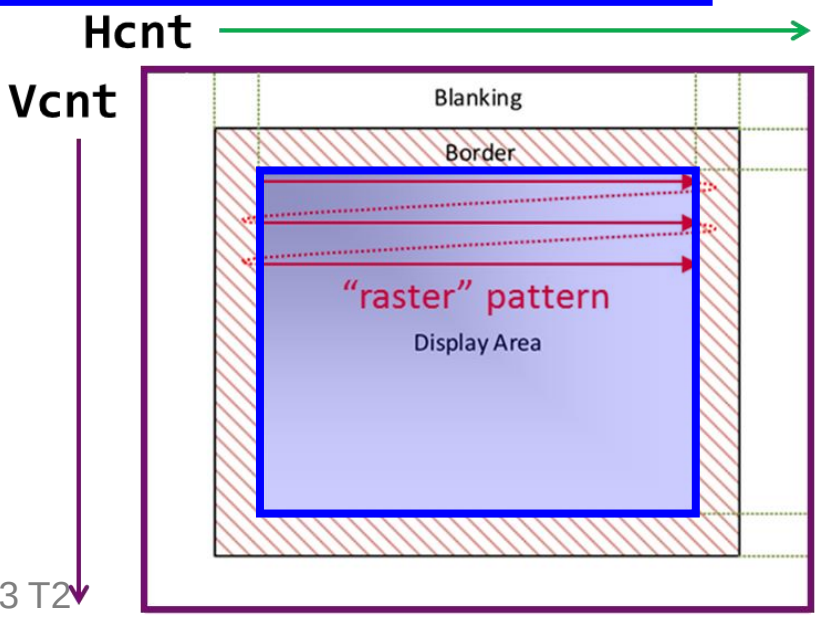



```
-- ⑥ generate vsync
vsync_gen_proc: process(hcount)
begin
  if(vcount <= V_SYNC) then
    vsync <= '1';
  else
    vsync <= '0';
  end if;
end process vsync_gen_proc;
```

# ⑦ generate RGB signals for display area

```
-- 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

        red   <= “0000";
        green <= "0000";
        blue  <= “1111";

    else

        red   <= "0000";
        green <= "0000";
        blue  <= "0000";

    end if;
end process data_output_proc;
```

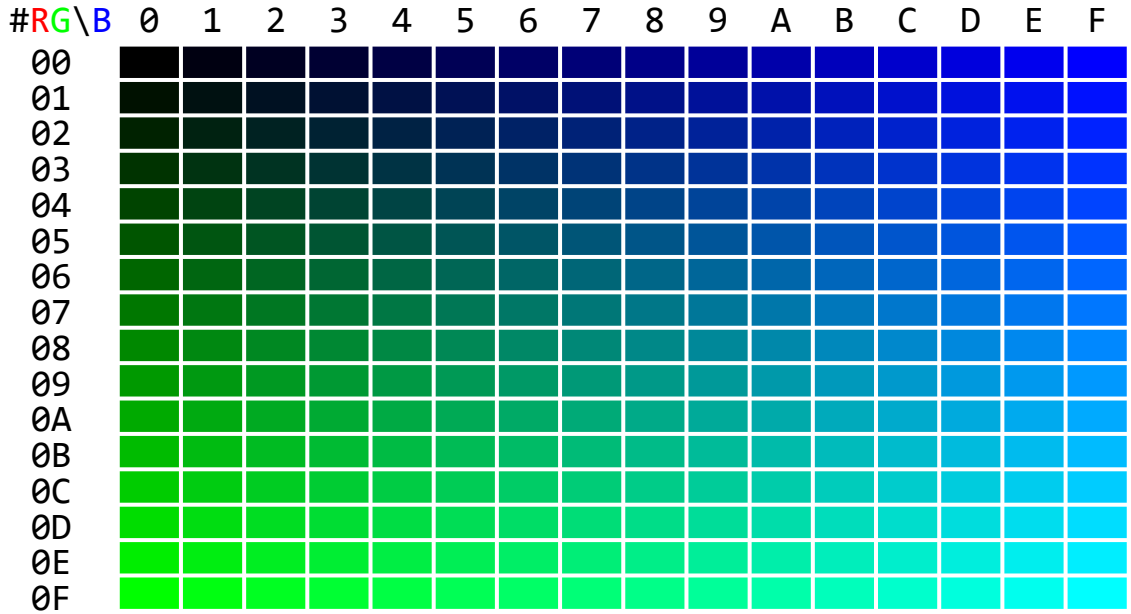
**You can display and draw anything pixel-by-pixel in this 1024x600 display area!**



# RGB Color Model

- **RGB color model** is an additive color model.
  - The red, green, and blue primary colors of light are added together to reproduce a broad array of colors.
- 12-bit RGB color model uses 4 bits for each of the red, green, and blue color components.
  - This results in a palette of $(2^4)^3 = 16^3 = 4096$ colors.
  - Each color can be represented by a **hexadecimal triple color code** (#RGB).
    - from #000 to #3FF
    - from #400 to #7FF
    - from #800 to #BFF
    - from #C00 to #FFF

Example: Hex Triple Color Codes (#000-#0FF)

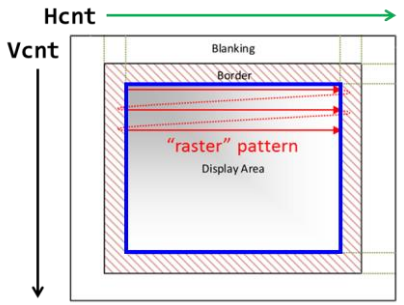


# Example Code: vga_test.xdc

```
set_property IOSTANDARD LVCMOS33 [get_ports {clk100MHz}]
set_property PACKAGE_PIN Y9 [get_ports {clk100MHz}]
create_clock -period 10 [get_ports clk100MHz]
# ------------------------------------------------------------------
# VGA Output - Bank 33
# ------------------------------------------------------------------
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“
set_property IOSTANDARD LVCMOS33 [get_ports -of_objects [get_iobanks 33]];
```

# Class Exercise 5.3

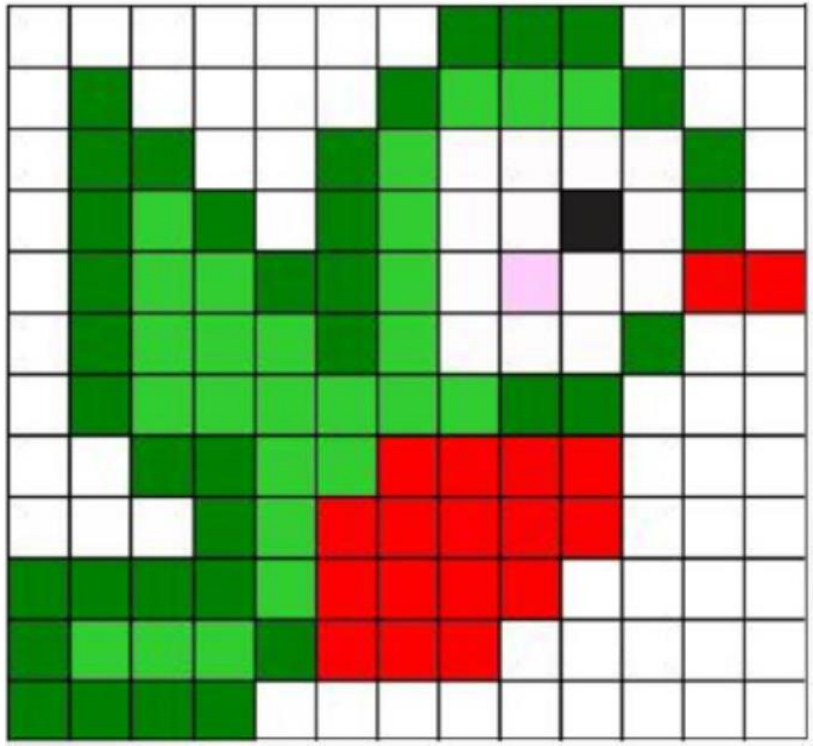
- Draw a red square (300x300) in the middle of the display area (1024x600):




```
-- 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
        if( _____________________________________
            _____________________________________
            _____________________________________
            _____________________________________ )
        then
            red <= "1111"; green <= “0000"; blue <= “0000";
        else
            red <= "0000"; green <= "0000"; blue <= "0000";
    else
        red   <= "0000"; green <= "0000"; blue  <= "0000";
    end if;
end process data_output_proc;
```

# Draw a Pixel Art

- How to draw the below bird pixel-by-pixel?



  - Use the **“enumeration type”** to define the required colors;
  - Use the **“array type”** to construct the pixel art “pixel-by-pixel” with the defined colors!

# Recall: Enumeration Type (Lec04)

- An enumeration type introduces abstraction into circuits by allowing users defining a list of values.
  - Example:

```
type colors is (RED, GREEN, BLUE);
signal my_color: colors;
```

- An enumerated type is ordered.
  - The order in which the values are listed in the type declaration defines their relation:

*Each values is greater than the one to the left, and less than the one to the right.*

  - Example: a comparison can be:

```
my_color > RED and my_color < BLUE
```

# Array Type (1/2)

- An array type consists of elements of the same type.

```
type TYPE_NAME is array (range) of element_type;
```

- Many common and useful array types are predefined in the IEEE 1164 standard:

```
type STD_LOGIC_VECTOR is array (NATURAL range <>) of std_logic;
```

  - *Note: Their ranges are **unconstrained** (i.e., `range <>`), and only bounded by `NATURAL` (positive integers).*

```
signal s: std_logic_vector (3 downto 0); -- or 0 to 3
```

  - *Note: An **unconstrained** array type must have its index type range defined when being declared.*

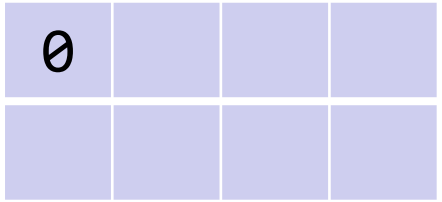
# Array Type (2/2)

- Array can be **unconstrained** or **constrained**:

```
type INT_ARRAY is array (NATURAL range <>) of integer;
signal s: INT_ARRAY (0 to 3);

type INT_ARRAY is array (0 to 3) of integer range 0 to 255;
-- the range of element type can also be designated
signal s: INT_ARRAY;
```


- Two (or more) dimensional arrays can be declared.

```
type INT_2D is array (0 to 1, 0 to 3) of integer;
signal s: INT_2D;
s(0, 0) <= 1; -- access the array element
```

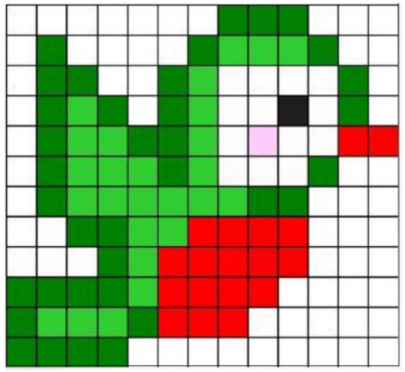



- Arrays of arrays can also be declared.

```
type INT_2D is array (0 to 1) of INT_ARRAY;
```

# Revisit: Draw a Pixel Art

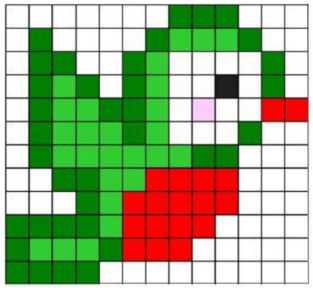
- How to draw the below bird pixel-by-pixel?



```
architecture DrawBird_Arch of DrawBird is
  -- colors
  type colors is (C_Black, C_DarkGreen,
  C_LightGreen, C_Red, C_White, C_Pink);

  -- 2d array (size: 12 rows x 13 columns)
  type T_2D is array (0 to 11, 0 to 12) of colors;
  constant bird : T_2D := (
  -- first row
  (C_Black, C_Black, C_Black, C_Black, C_Black, C_Black,
   C_Black, C_DarkGreen, C_DarkGreen, C_DarkGreen, C_Black,
   C_Black, C_Black),
  -- second row
  (C_Black, C_DarkGreen, C_Black, C_Black, C_Black, C_Black,
   C_DarkGreen, C_LightGreen, C_LightGreen, C_LightGreen,
   C_DarkGreen, C_Black, C_Black),
  ... );
```

# Class Exercise 5.4

- Draw the bird in the middle of the display area:



```
-- 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
    if( vcount >= (V_START+V_END)/2 and
        vcount < (V_START+V_END)/2 + 12 and
        hcount >= (H_START+H_END)/2 and
        hcount < (H_START+H_END)/2 + 13 ) then
      _____________________________________________________
      _____________________________________________________
      _____________________________________________________
      _____________________________________________________
  else
    red   <= "0000"; green <= "0000"; blue  <= "0000";
  end if;
end process data_output_proc;
```

# Summary

- VGA Basics
  - VGA Connector
  - Pixel-by-Pixel Raster Pattern
  - Timing Specification
- Clock Sources on ZedBoard
  - Clock Divider
- Case Study: 1024x600@60Hz
  - vga_test.vhd
  - vga_test.xdc