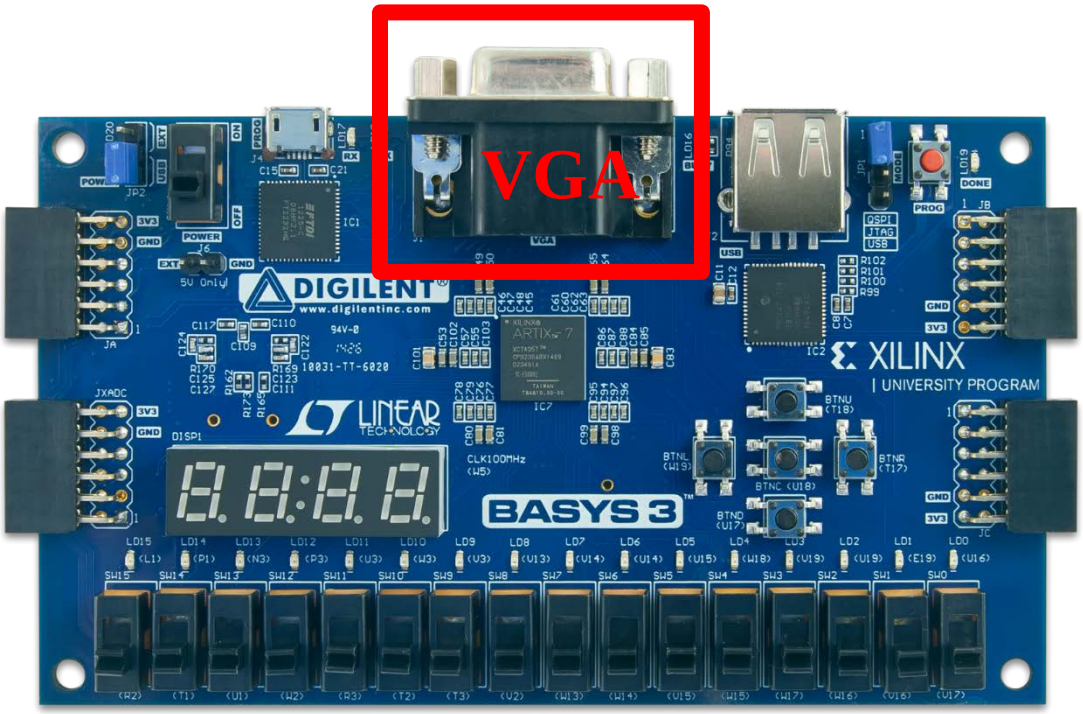
VGA

黃元豪
Yuan-Hao Huang

國立清華大學電機工程學系
Department of Electrical Engineering
National Tsing-Hua University

VGA Port


VGA
DIGILENT
www.digilentinc.com
XILINX
UNIVERSITY PROGRAM
LINEAR TECHNOLOGY
BASYS 3
CLK100MHz
(W5)

VGA

- VGA = Video Graphics Array
- Introduced by IBM in 1987, and still used today
- Transmitting analog video signal

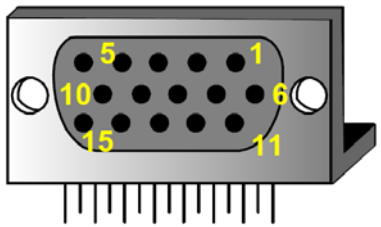
Cathode-Ray Tube Monitor

Video Graphics Array

LED Monitor

VGA Video Signal

- A VGA video signal contains 5 active signals:
  - horizontal sync (HS): used for video synchronization in the horizontal direction
  - vertical sync (VS): used for video synchronization in the vertical direction
  - red (R): used to control the red color, 0v (fully off) ~ 0.7v (fully on)
  - green (G): used to control the green color, 0v (fully off) ~ 0.7v (fully on)
  - blue (B): used to control the blue color, 0v (fully off) ~ 0.7v (fully on)

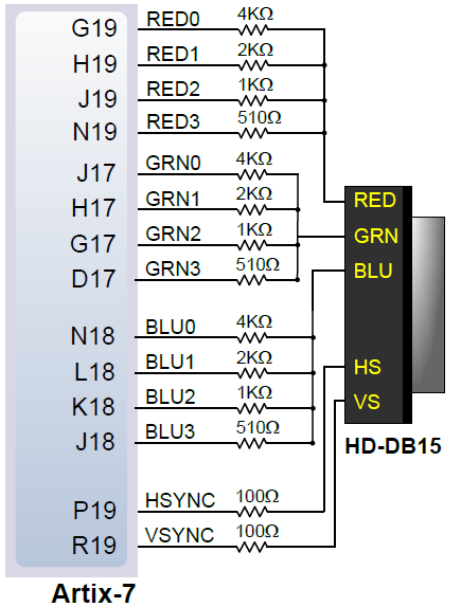


Pin 1: Red
Pin 2: Grn
Pin 3: Blue
Pin 13: HS
Pin 14: VS

Pin Assignments

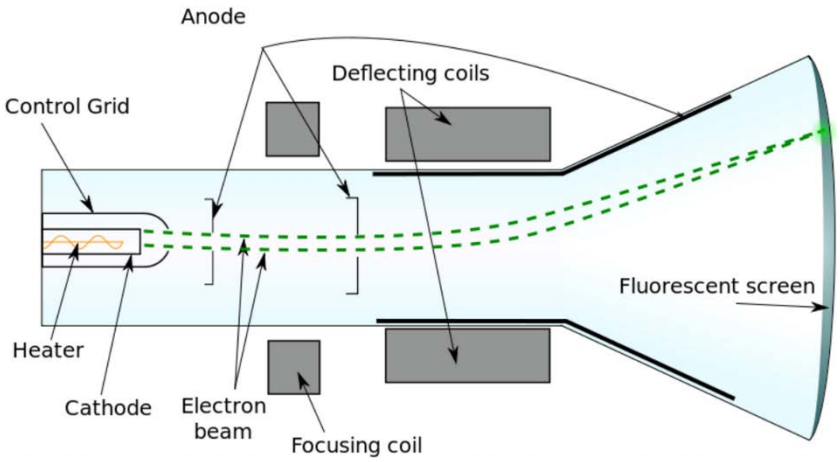
- Corresponding pins in a constraint file (.xdc)

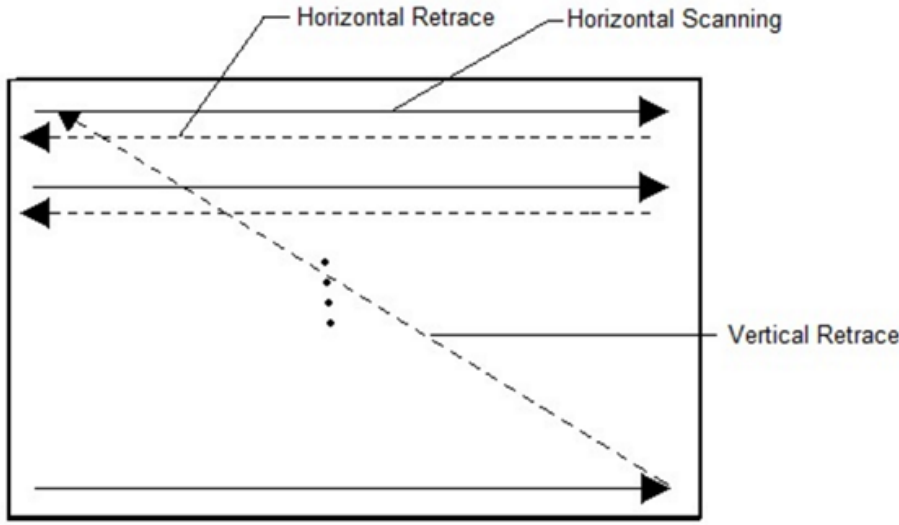
```
##VGA Connector
set_property PACKAGE_PIN G19 [get_ports {vgaRed[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {vgaRed[0]}]
set_property PACKAGE_PIN H19 [get_ports {vgaRed[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {vgaRed[1]}]
set_property PACKAGE_PIN J19 [get_ports {vgaRed[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {vgaRed[2]}]
set_property PACKAGE_PIN N19 [get_ports {vgaRed[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {vgaRed[3]}]
set_property PACKAGE_PIN N18 [get_ports {vgaBlue[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {vgaBlue[0]}]
set_property PACKAGE_PIN L18 [get_ports {vgaBlue[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {vgaBlue[1]}]
set_property PACKAGE_PIN K18 [get_ports {vgaBlue[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {vgaBlue[2]}]
set_property PACKAGE_PIN J18 [get_ports {vgaBlue[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {vgaBlue[3]}]
set_property PACKAGE_PIN J17 [get_ports {vgaGreen[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {vgaGreen[0]}]
set_property PACKAGE_PIN H17 [get_ports {vgaGreen[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {vgaGreen[1]}]
set_property PACKAGE_PIN G17 [get_ports {vgaGreen[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {vgaGreen[2]}]
set_property PACKAGE_PIN D17 [get_ports {vgaGreen[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {vgaGreen[3]}]
set_property PACKAGE_PIN P19 [get_ports hsync]
set_property IOSTANDARD LVCMOS33 [get_ports hsync]
set_property PACKAGE_PIN R19 [get_ports vsync]
set_property IOSTANDARD LVCMOS33 [get_ports vsync]
```



CRT

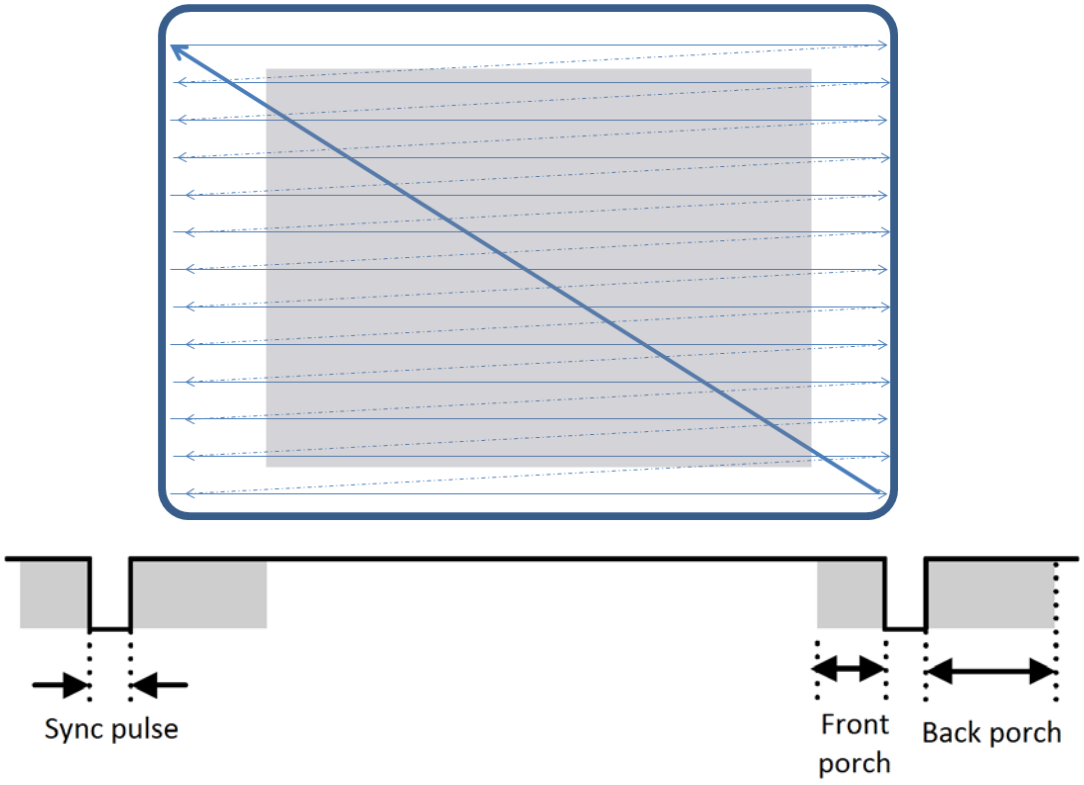
- Cathode Ray Tube (CRT) is a vacuum tube containing one or more electron guns, and a phosphorescent screen is used to view images.



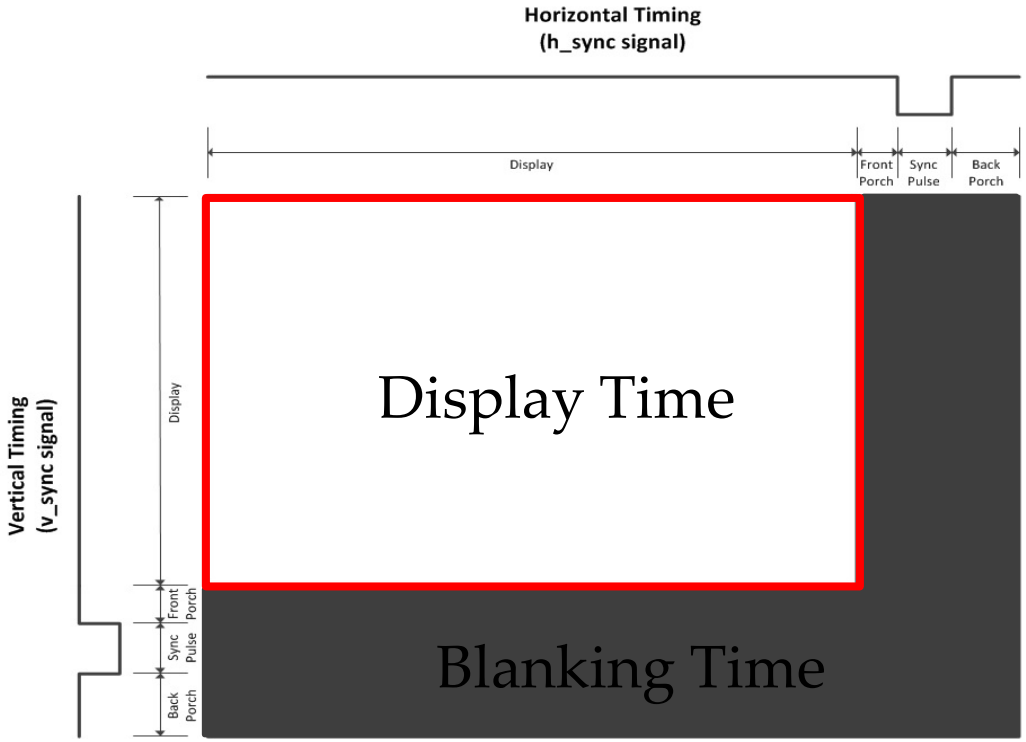


VGA System Timing (1/3)

- Front porch : blank while still moving right.
- Sync pulse : blank while rapidly moving left.
- Back porch : blank while moving right again.



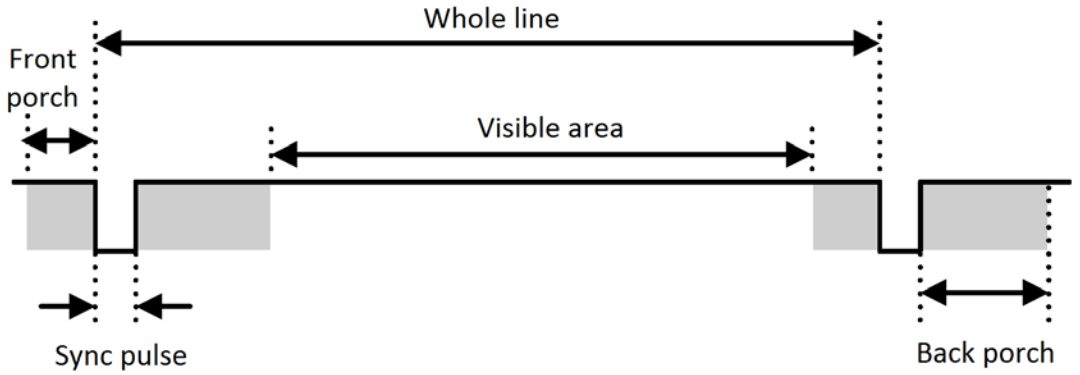
VGA System Timing (2/3)



VGA System Timing (3/3)

- Signal timing for a 640-pixel by 480 rows display using a 25MHz pixel clock

Parameter	Ver. Sync		Hor. Sync	
	Lines	Time(ms)	Pixels	Time(μs)
Visible area	480	15.3	640	25
Front porch	10	0.3	16	0.64
Sync pulse	2	0.064	96	3.8
Back porch	33	1.05	48	1.9
Whole line	525	16.7	800	32



Pixel Clock

- The pixel clock defines the time available to display one pixel of information.
- Example: Suppose we want to display an image with 480 rows and 640 columns, and its refresh rate is 60Hz. The pixel clock which will be delivered to VGA screen must be

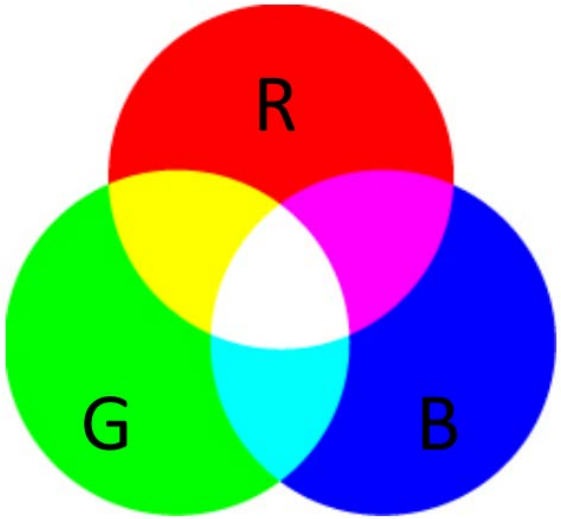
800*525*60(frame/sec) = 25M (pixel/sec)

RGB Bitmap

- A digital color image is composed by a lot of pixels.
- Each pixel contains three R, G, B values to represent the intensity of these three primary colors.

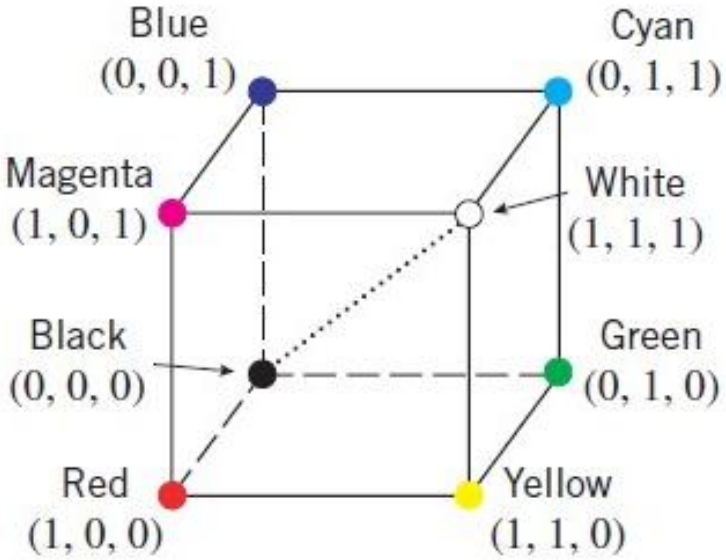
RGB Color Mode

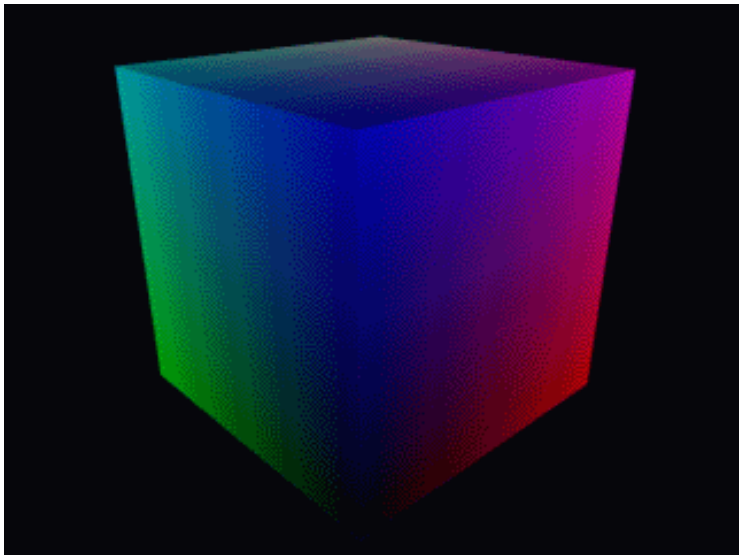
- Three primary colors
  - **Red**
  - **Green**
  - **Blue**
- No one of them can be created as a combination of the other two.
- Any other color is a combination of them.



RGB Color Cube

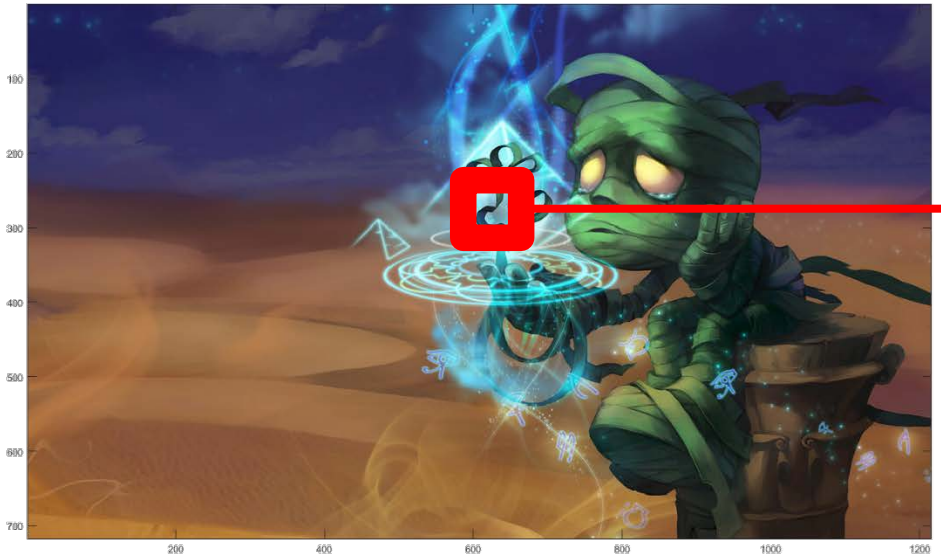
- R, G, and B correspond to three axes in 3D space.
- Normalize the relative amounts of R, G and B so that each value varies between 0 and 1.



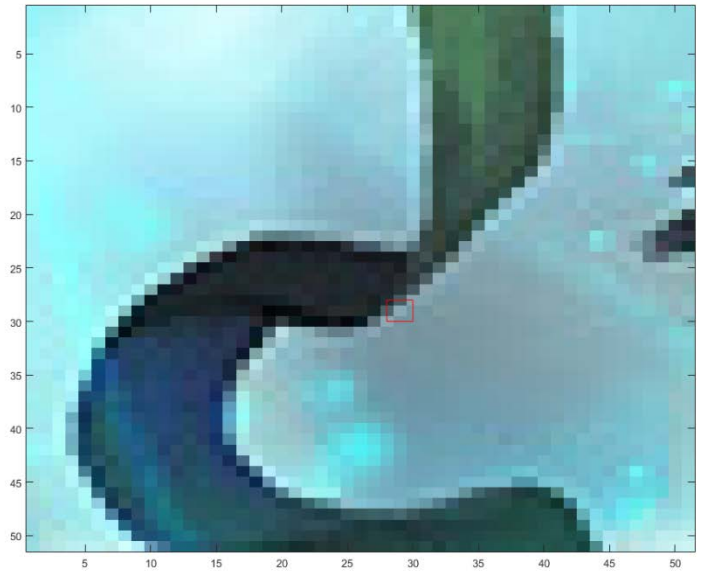


RGB Bitmap Example (1/2)

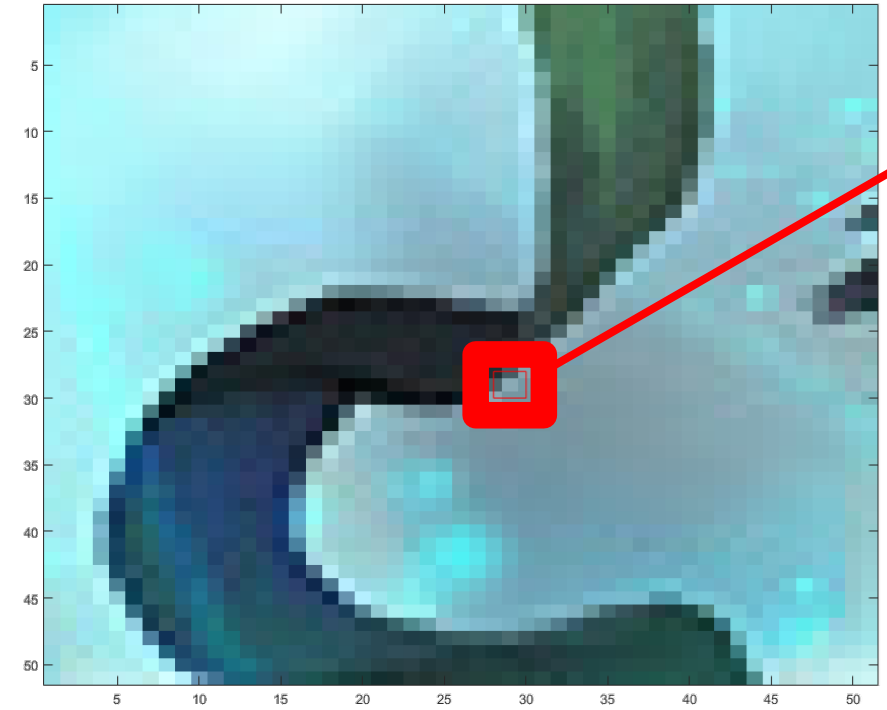
Size: 1215*717

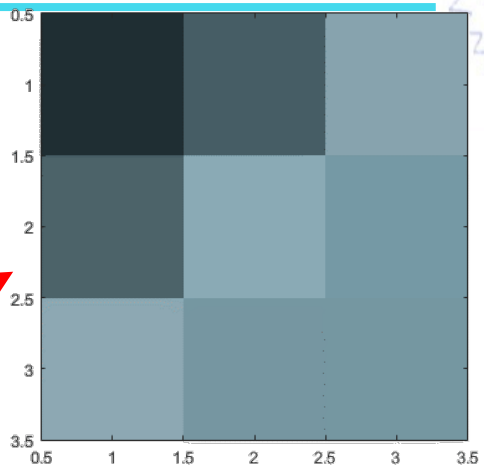


Size: 51*51



RGB Bitmap Example (2/2)

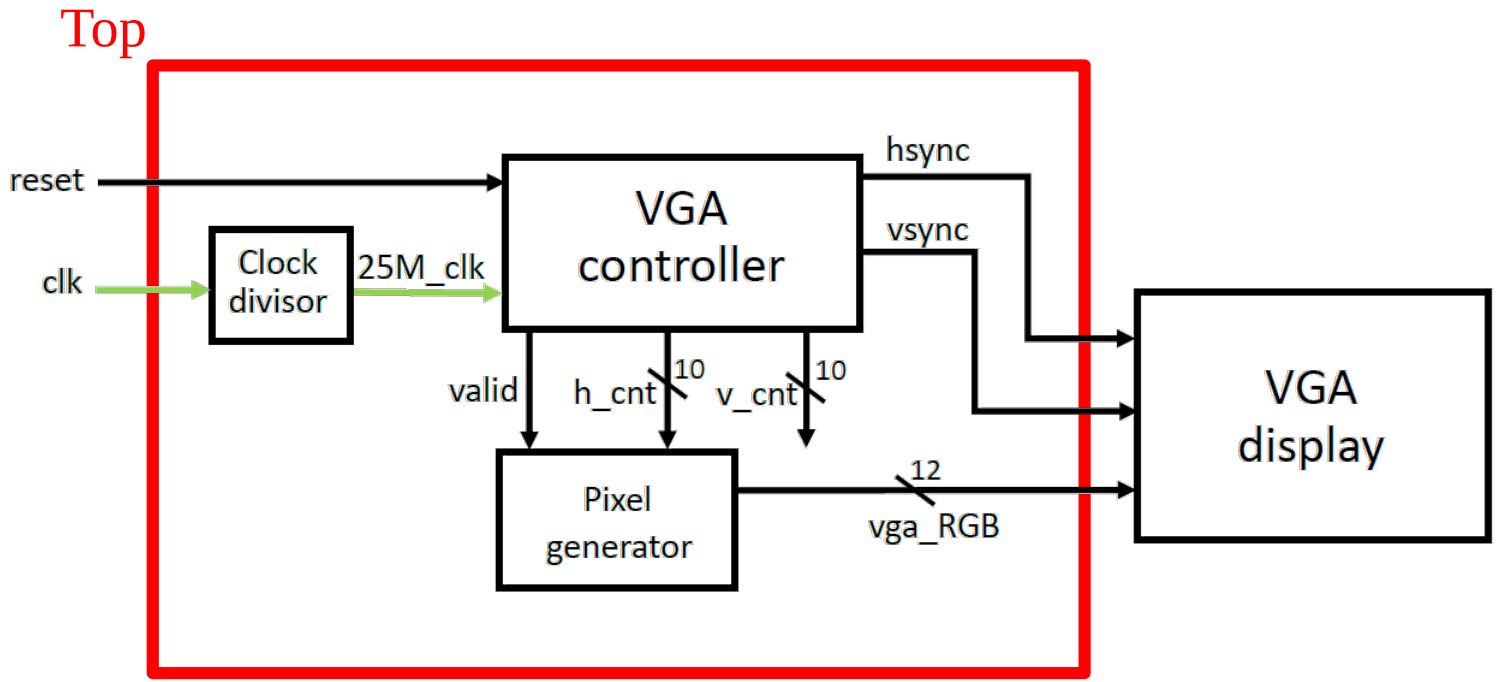




(R,G,B) : range from 0 to 255

(31,46,51)	(70,93,101)	(135,163,174)
(76,99,105)	(138,170,181)	(117,153,165)
(141,168,179)	(118,150,161)	(117,151,161)

VGA Demo 1: Block Diagram



VGA Demo 1: Top Module

```
module top(
    input clk,
    input rst,
    output [3:0] vgaRed,
    output [3:0] vgaGreen,
    output [3:0] vgaBlue,
    output hsync,
    output vsync
     );
     wire clk_25MHz;
     wire valid;
     wire [9:0] h_cnt; //640
     wire [9:0] v_cnt;  //480
```

```
      clock_divisor clk_wiz_0_inst(
        .clk(clk),
        .clk1(clk_25MHz)
      );

    pixel_gen pixel_gen_inst(
        .h_cnt(h_cnt),
        .valid(valid),
        .vgaRed(vgaRed),
        .vgaGreen(vgaGreen),
        .vgaBlue(vgaBlue)
      );

     vga_controller   vga_inst(
        .pclk(clk_25MHz),
        .reset(rst),
        .hsync(hsync),
        .vsync(vsync),
        .valid(valid),
        .h_cnt(h_cnt),
        .v_cnt(v_cnt)
      );

endmodule
```

VGA Demo 1: Clock Divisor

```
module clock_divisor(clk1, clk);
input clk;
output clk1;
reg [1:0] num;
wire [1:0] next_num;

always @(posedge clk) begin
  num <= next_num;
end

assign next_num = num + 1'b1;
assign clk1 = num[1];
endmodule
```

VGA Demo 1: Pixel generator

```
module pixel_gen(
    input [9:0] h_cnt,
    input valid,
    output reg [3:0] vgaRed,
    output reg [3:0] vgaGreen,
    output reg [3:0] vgaBlue
    );

        always @(*) begin
        if(!valid)
                {vgaRed, vgaGreen, vgaBlue} = 12'h0;
         else if(h_cnt < 128)
                {vgaRed, vgaGreen, vgaBlue} = 12'h000;
         else if(h_cnt < 256)
                {vgaRed, vgaGreen, vgaBlue} = 12'h00f;
         else if(h_cnt < 384)
                {vgaRed, vgaGreen, vgaBlue} = 12'hf00;
         else if(h_cnt < 512)
                {vgaRed, vgaGreen, vgaBlue} = 12'h0f0;
         else if(h_cnt < 640)
                {vgaRed, vgaGreen, vgaBlue} = 12'hfff;
         else
                {vgaRed, vgaGreen, vgaBlue} = 12'h0;
    end
endmodule
```

VGA Controller (1/3)

```
//Module Name: vga
  module vga_controller (
   input wire pclk, reset,
   output wire hsync,vsync,valid,
   output wire [9:0] h_cnt,
   output wire [9:0] v_cnt
   );
   reg [9:0] pixel_cnt;
   reg [9:0] line_cnt;
   reg hsync_i, vsync_i;
   wire hsync_default, vsync_default;
   wire [9:0] HD, HF, HS, HB, HT;
   wire [9:0] VD, VF, VS, VB, VT;
```

```
   assign HD = 640;
   assign HF = 16;
   assign HS = 96;
   assign HB = 48;
   assign HT = 800;
   assign VD = 480;
   assign VF = 10;
   assign VS = 2;
   assign VB = 33;
   assign VT = 525;
   assign hsync_default = 1'b1;
   assign vsync_default = 1'b1;
```

VGA Controller (2/3)

```
always@(posedge pclk)
    if(reset)
        pixel_cnt <= 0;
    else if(pixel_cnt < (HT - 1))
        pixel_cnt <= pixel_cnt + 1;
    else
        pixel_cnt <= 0;

always@(posedge pclk)
    if(reset)
        hsync_i <= hsync_default;
    else if( (pixel_cnt >= (HD + HF - 1))
    && (pixel_cnt < (HD + HF + HS - 1)) )
        hsync_i <= ~hsync_default;
    else
        hsync_i <= hsync_default;
```

```
always@(posedge pclk)
    if(reset)
        line_cnt <= 0;
    else if(pixel_cnt == (HT -1))
        if(line_cnt < (VT - 1))
            line_cnt <= line_cnt + 1;
        else
            line_cnt <= 0;

always@(posedge pclk)
    if(reset)
        vsync_i <= vsync_default;
    else if( (line_cnt >= (VD + VF - 1))
    &&(line_cnt < (VD + VF + VS - 1)) )
        vsync_i <= ~vsync_default;
    else
        vsync_i <= vsync_default;
```

VGA Controller (3/3)

```
    assign hsync = hsync_i;
    assign vsync = vsync_i;
    assign valid = ( (pixel_cnt < HD) && (line_cnt < VD) );
    assign h_cnt = (pixel_cnt < HD) ? pixel_cnt : 10'd0;
    assign v_cnt = (line_cnt < VD) ? line_cnt : 10'd0;

endmodule
```

RAM

- FPGA chip uses RAM to map and buffer the image to the VGA display
- RAM memory generation
  - In IP generator, choose 'Single Port RAM'
- Timing
  - Write control, address, data should be at the same clock cycle.
  - Data read out from RAM is one clock cycle late than the address control

Memory in FPGA

- Xilinx FPGA defines memory coefficient file (COE) that is stored in embedded memory in the FPGA chip.
- Two parameter:
  - **memory_initialization_radix**
    - Radix of the values in the *memory_initialization_vector*
    - Ex: 2, 10, or 16
  - **memory_initialization_vector:**
    - Memory content
    - Memory words are separated by whitespace
    - You can use comma (,) to help identify the boundary
    - Vector (entire memory) ended by semicolon

COE Example

out.coe

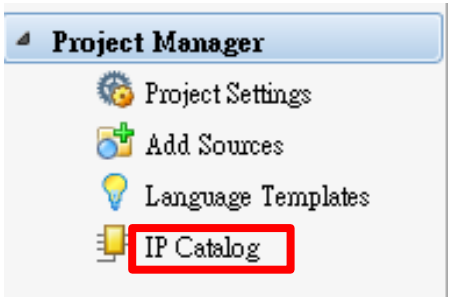
```
memory_initialization_radix=16;
memory_initialization_vector=
115,
115,
115,
115,
115,
115,
115,
115,
115,
115,
```

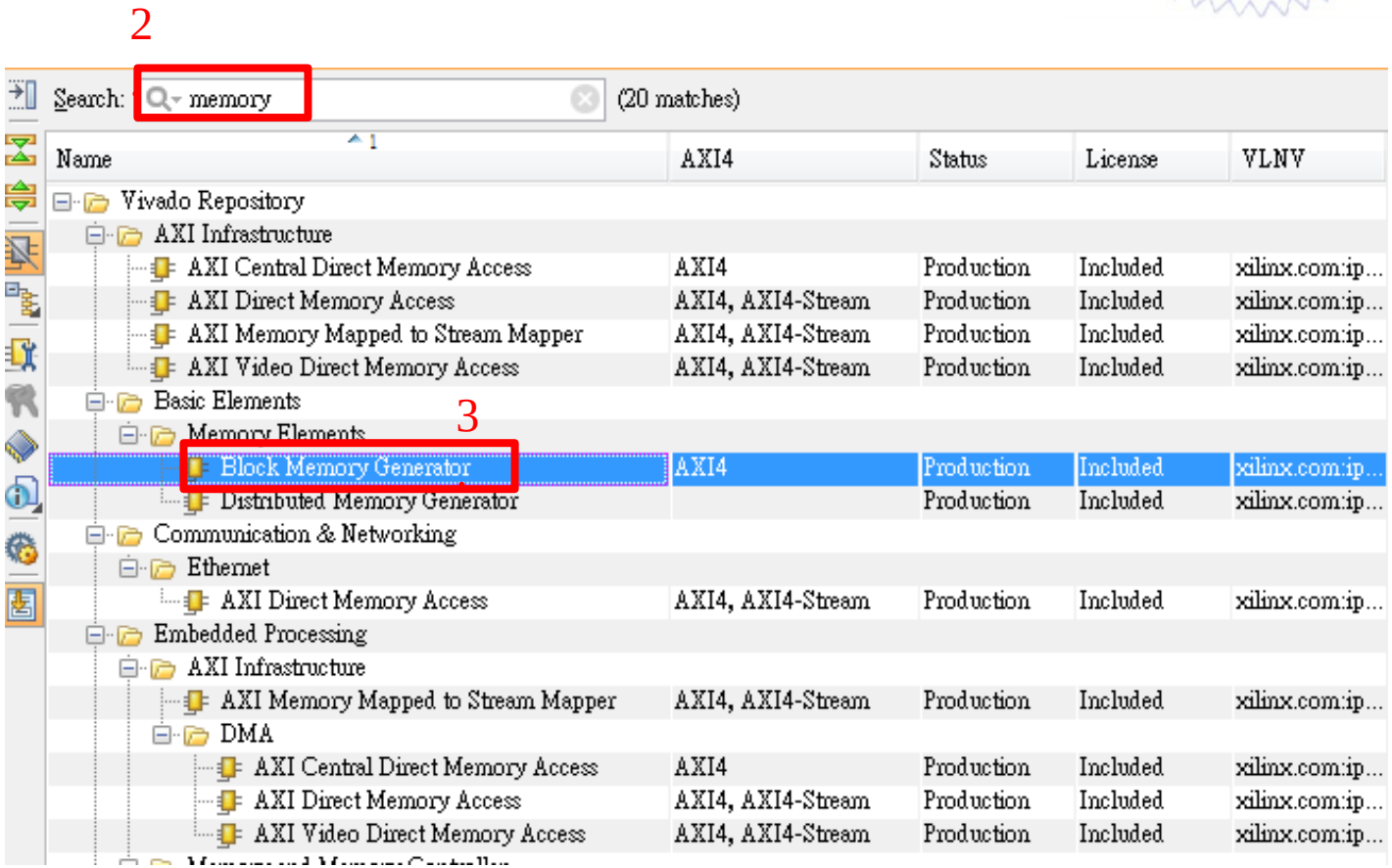
```
743,
743,
743,
743,
743,
743;
```

PicTrans.exe

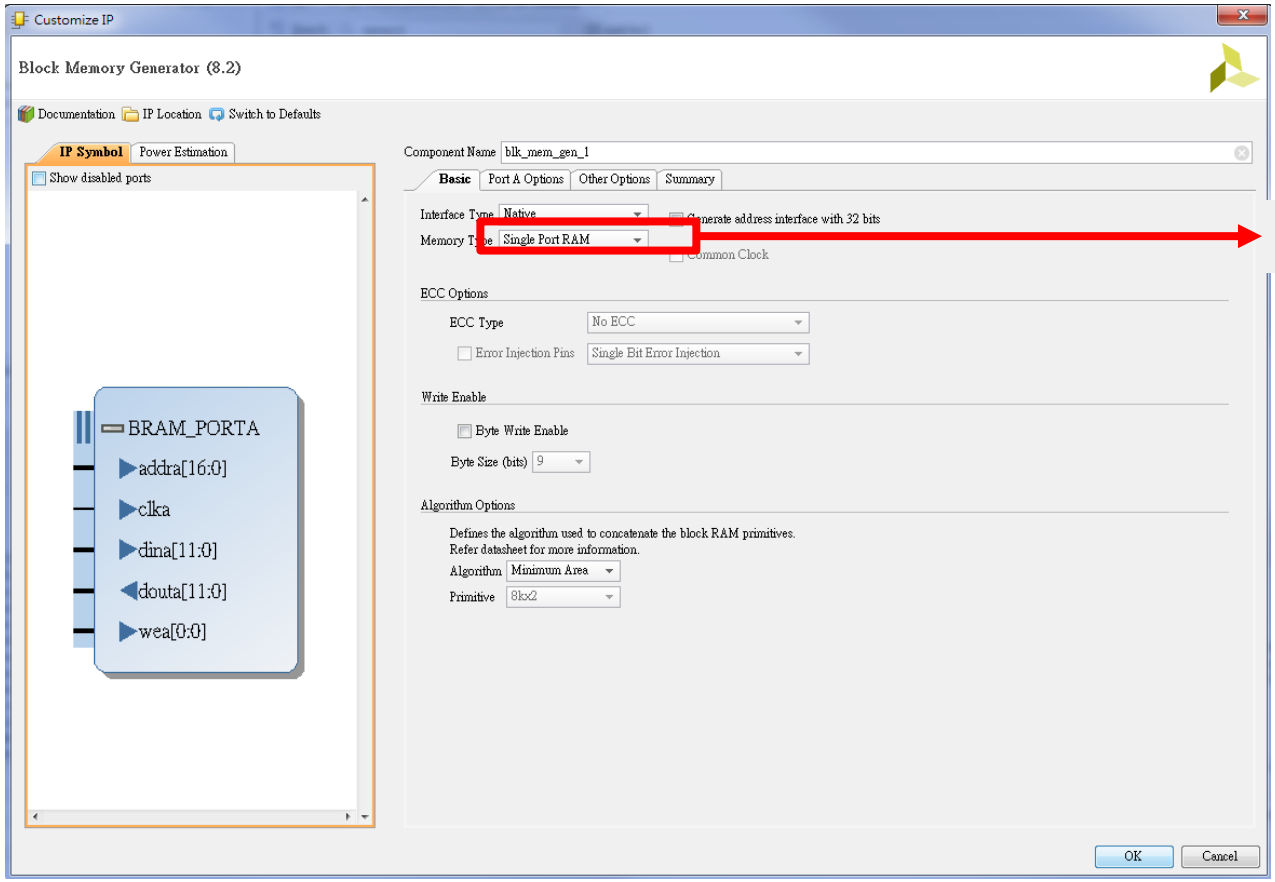
- Memory mapping to the image
  - You can use ASCII art generator to generate the pictures or use drawing tool to export the image figure.

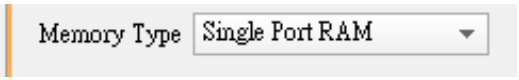
Memory IP (1/5)

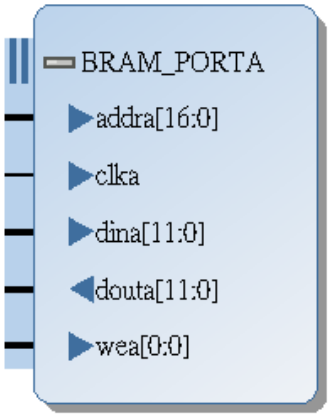

Project Manager
Project Settings
Add Sources
Language Templates
IP Catalog
1


2
Search: memory
(20 matches)
Name
AXI4
Status
License
VLNV
Vivado Repository
AXI Infrastructure
AXI Central Direct Memory Access
AXI Direct Memory Access
AXI Memory Mapped to Stream Mapper
AXI Video Direct Memory Access
Basic Elements
Memory Elements
3
Block Memory Generator
Distributed Memory Generator
Communication & Networking
Ethernet
AXI Direct Memory Access
Embedded Processing
AXI Infrastructure
AXI Memory Mapped to Stream Mapper
DMA
AXI Central Direct Memory Access
AXI Direct Memory Access
AXI Video Direct Memory Access
Production
Included
xilinx.com:ip...

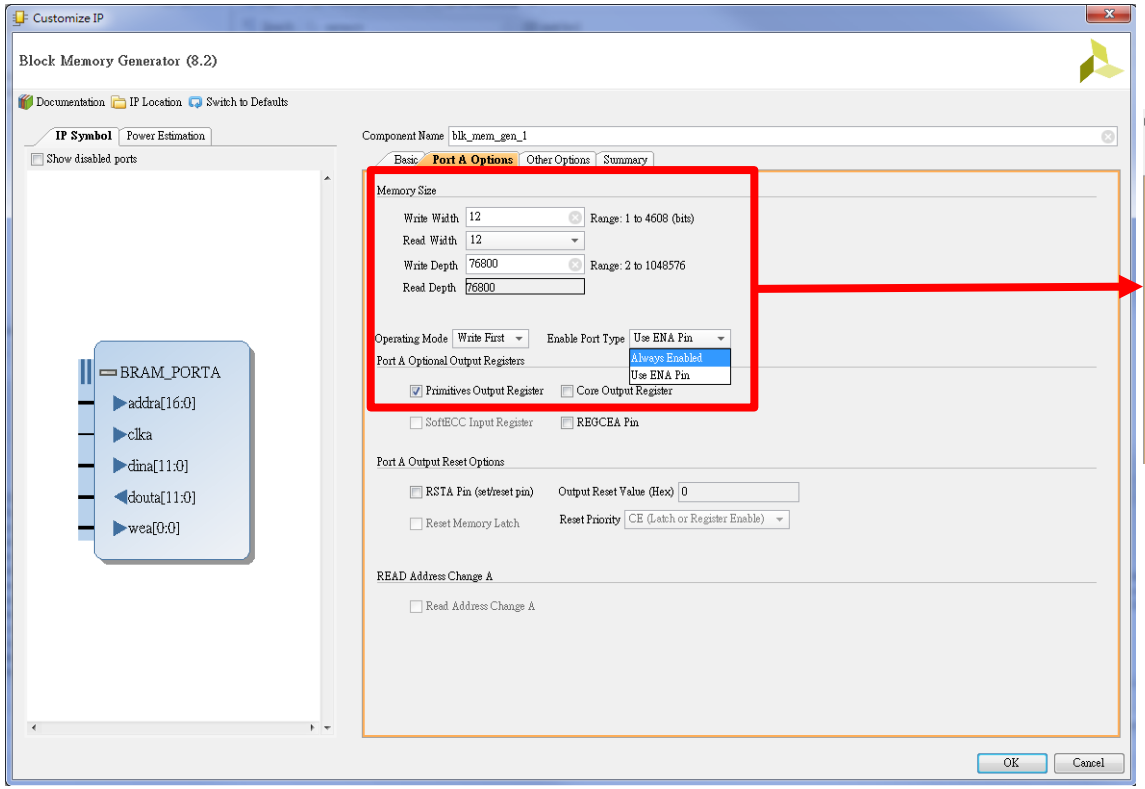
Memory IP (2/5)

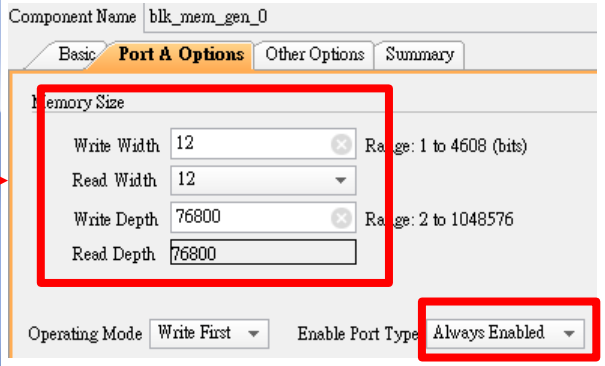

Customize IP
Block Memory Generator (8.2)
Documentation
IP Location
Switch to Defaults
IP Symbol
Power Estimation
Show disabled ports
BRAM_PORTA
addra[16:0]
clka
dina[11:0]
douta[11:0]
wea[0:0]
Component Name
blk_mem_gen_1
Basic
Port A Options
Other Options
Summary
Interface Type
Native
Memory Type
Single Port RAM
Common Clock
ECC Options
ECC Type
No ECC
Error Injection Pins
Single Bit Error Injection
Write Enable
Byte Write Enable
Byte Size (bits)
9
Algorithm Options
Defines the algorithm used to concatenate the block RAM primitives.
Refer datasheet for more information.
Algorithm
Minimum Area
Primitive
8kx2
OK
Cancel


Memory Type
Single Port RAM


BRAM_PORTA
addra[16:0]
clka
dina[11:0]
douta[11:0]
wea[0:0]

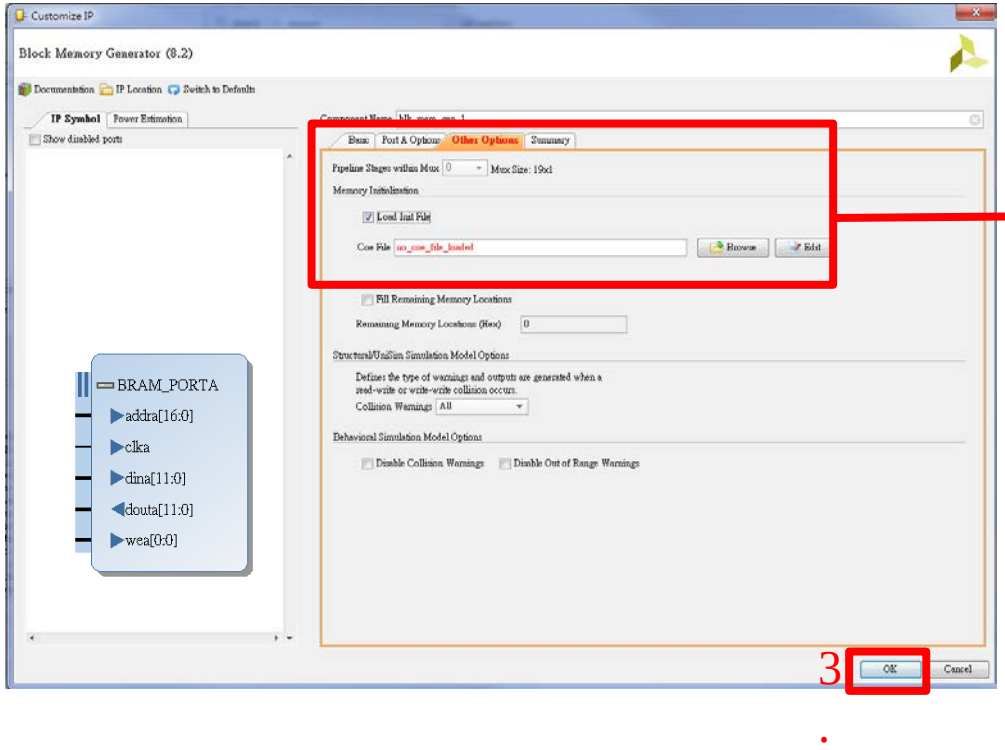
Memory IP (3/5)

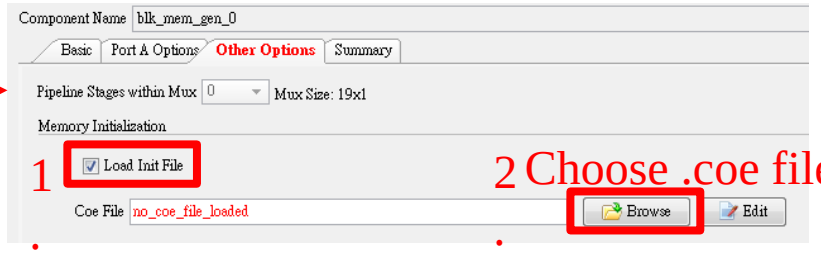




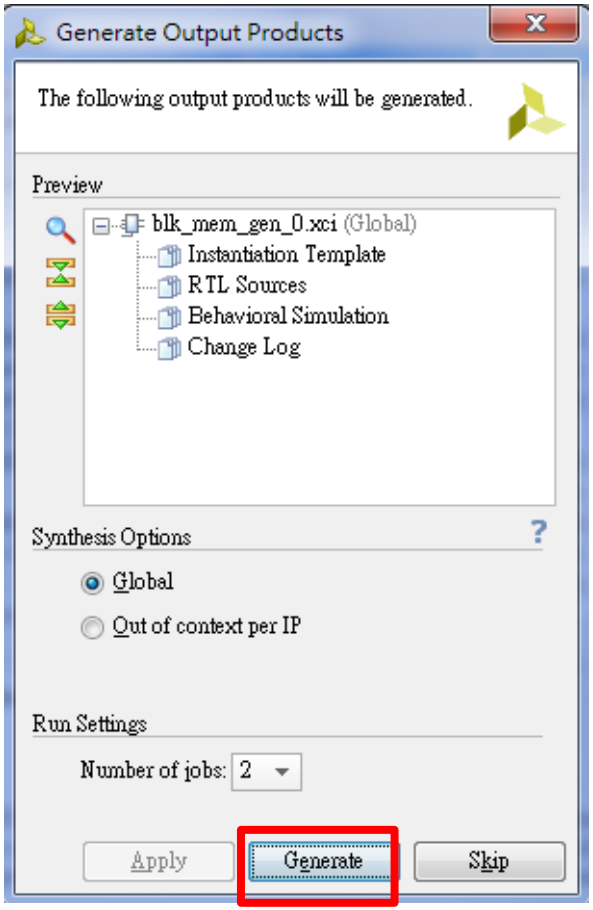
RGB : 12 bits
320x240 : 76800

Memory IP (4/5)


Customize IP
Block Memory Generator (8.2)
Documentation
IP Location
Switch to Defaults
IP Symbol
Power Estimation
Show disabled ports
BRAM_PORTA
addra[16:0]
clka
dina[11:0]
douta[11:0]
wea[0:0]
Basic
Port A Options
Other Options
Summary
Pipeline Stages within Mux
Mux Size: 19x1
Memory Initialization
Load Init File
Coe File
no_coe_file_loaded
Browse
Edit
Fill Remaining Memory Locations
Structural/UniSim Simulation Model Options
Collision Warnings
All
Behavioral Simulation Model Options
Disable Collision Warnings
Disable Out of Range Warnings
3
OK
Cancel


Component Name
blk_mem_gen_0
Basic
Port A Options
Other Options
Summary
Pipeline Stages within Mux
Mux Size: 19x1
Memory Initialization
1
Load Init File
2 Choose .coe file
Coe File
no_coe_file_loaded
Browse
Edit

Memory IP (5/5)


Generate Output Products
The following output products will be generated.
Preview
blk_mem_gen_0.xci (Global)
Instantiation Template
RTL Sources
Behavioral Simulation
Change Log
Synthesis Options
Global
Out of context per IP
Run Settings
Number of jobs: 2
Apply
Generate
Skip

Picture Format Translation (1/2)

amumu.jpg

PicTrans.exe

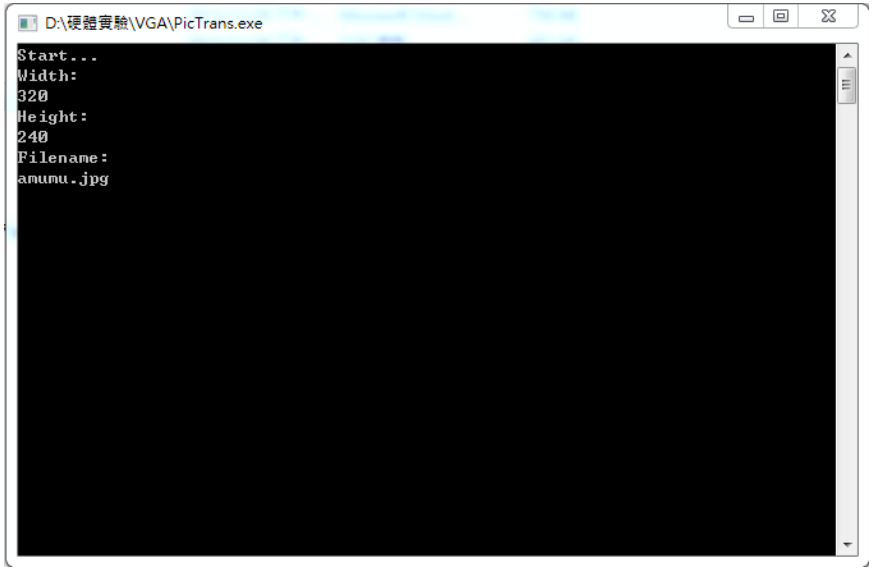
out.coe

out.coe

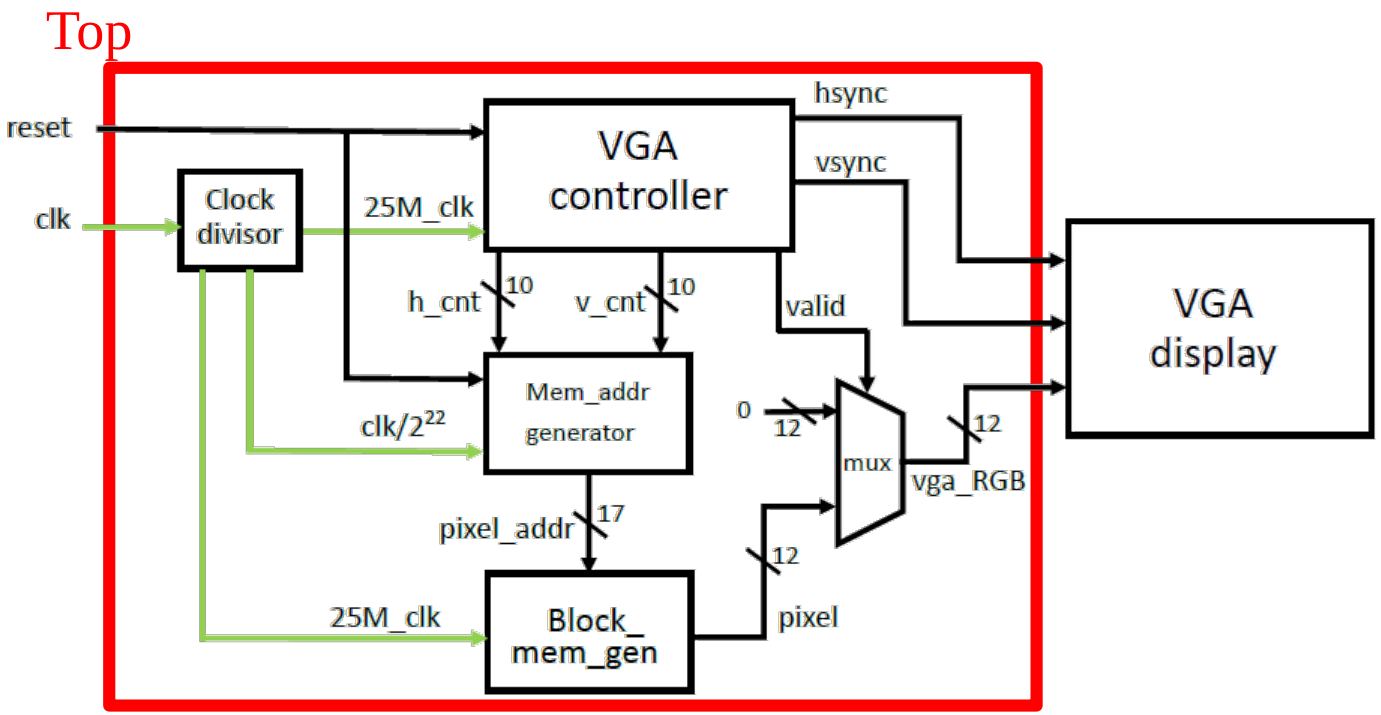
```
memory_initialization_radix=16;
memory_initialization_vector=
115,
115,
115,
115,
115,
115,
115,
115,
115,
115,
115,
  :   :
743,
743,
743,
743,
743,
743;
```

Picture Format Translation (2/2)

- PicTrans.exe:
  - Convert a *.jpg file to a bit map file
- Input:
  - image (*.jpg)
  - the width of the output file
  - the height of the output file
- Output:
  - out.coe



VGA Demo 2: Block Diagram



VGA Demo 2: Top Module

```
module top(
    input clk,
    input rst,
    output [3:0] vgaRed,
    output [3:0] vgaGreen,
    output [3:0] vgaBlue,
    output hsync,
    output vsync
     );

     wire [11:0] data;
     wire clk_25MHz;
     wire clk_22;
     wire [16:0] pixel_addr;
     wire [11:0] pixel;
     wire valid;
     wire [9:0] h_cnt; //640
     wire [9:0] v_cnt;  //480

assign {vgaRed, vgaGreen, vgaBlue}
        = (valid==1'b1) ? pixel:12'h0;
```

```
 clock_divisor clk_wiz_0_inst(
   .clk(clk),
   .clk1(clk_25MHz),
   .clk22(clk_22)
);

mem_addr_gen mem_addr_gen_inst(
.clk(clk_22),
.rst(rst),
.h_cnt(h_cnt),
.v_cnt(v_cnt),
.pixel_addr(pixel_addr)
);


blk_mem_gen_0 blk_mem_gen_0_inst(
   .clka(clk_25MHz),
   .wea(0),
   .addra(pixel_addr),
   .dina(data[11:0]),
   .douta(pixel)
);
```

```
    vga_controller   vga_inst(
       .pclk(clk_25MHz),
       .reset(rst),
       .hsync(hsync),
       .vsync(vsync),
       .valid(valid),
       .h_cnt(h_cnt),
       .v_cnt(v_cnt)
    );

endmodule
```

Demo 2: Clock Divisor

```
module clock_divisor(clk1, clk, clk22);
input clk;
output clk1;
output clk22;
reg [21:0] num;
wire [21:0] next_num;

always @(posedge clk) begin
  num <= next_num;
end

assign next_num = num + 1'b1;
assign clk1 = num[1];
assign clk22 = num[21];
endmodule
```

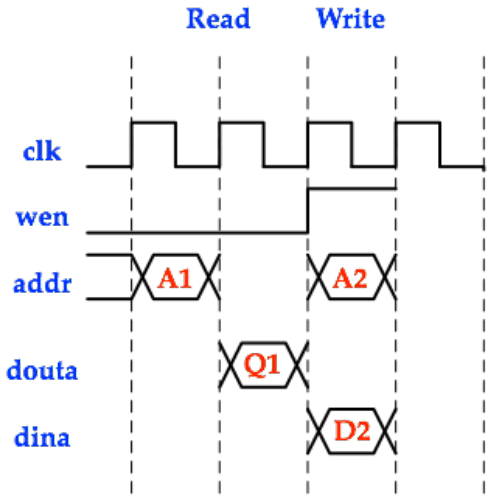
VGA Demo 2: Memory address generator

```
module mem_addr_gen(
    input clk,
    input rst,
    input [9:0] h_cnt,
    input [9:0] v_cnt,
    output [16:0] pixel_addr
    );

    reg [7:0] position;

    assign pixel_addr = ((h_cnt>>1)+320*(v_cnt>>1)+ position*320 )% 76800;  //640*480 --> 320*240

    always @ (posedge clk or posedge rst) begin
        if(rst)
            position <= 0;
        else if(position < 239)
            position <= position + 1;
        else
            position <= 0;
    end

endmodule
```

How to Use RAM Module

- Memory Read and Write is controlled by
  - "wen" pin : wen=0 is "read", wen=1 is "write".
  - Separate read-address generator and write-address generator.
- To refresh the image, you can
  - Always read out image from memory (wen=0) and display image in VGA synchronized with hsync and vsync. Write in image changes (wen=1) when necessary.

```
wire clk;
wire wen;
wire [11:0] data_in;
wire [11:0] data_out;
wire [16:0] addr;

RAM R1(
  .clka(clk),
  .wea(wen),
  .addra(addr),
  .dina(data_in),
  .douta(data_out)
);
```



Lab 11: VGA Display

Action Item (1/2)

Modify the Verilog code introduced in class to design a circuit for controlling the VGA display.

- input ports:
```
input  clk;
input  reset;
input  en;
```

- output ports:
```
output  [3:0] vgaRed;
output  [3:0] vgaGreen;
output  [3:0] vgaBlue;
output  hsync;
output  vsync;
```

Action Item (2/2)

- At the beginning or when pressing the **reset** button, the VGA display will show the image (e.g., amumu.jpg) at the origin position. It will stay still until the **en** button is pressed.
- The image will start/resume scrolling down row by row under the frequency of clk divided by 2^{22} (i.e., clk/2^{22}), or pause, depending on whether the number of the **en** button pressed is odd or even.

Example (1/6)

at the beginning or pressing **reset**

(0 row scrolled down)

Example (2/6)

press **en** ➔ start to scroll down

(100 rows scrolled down)

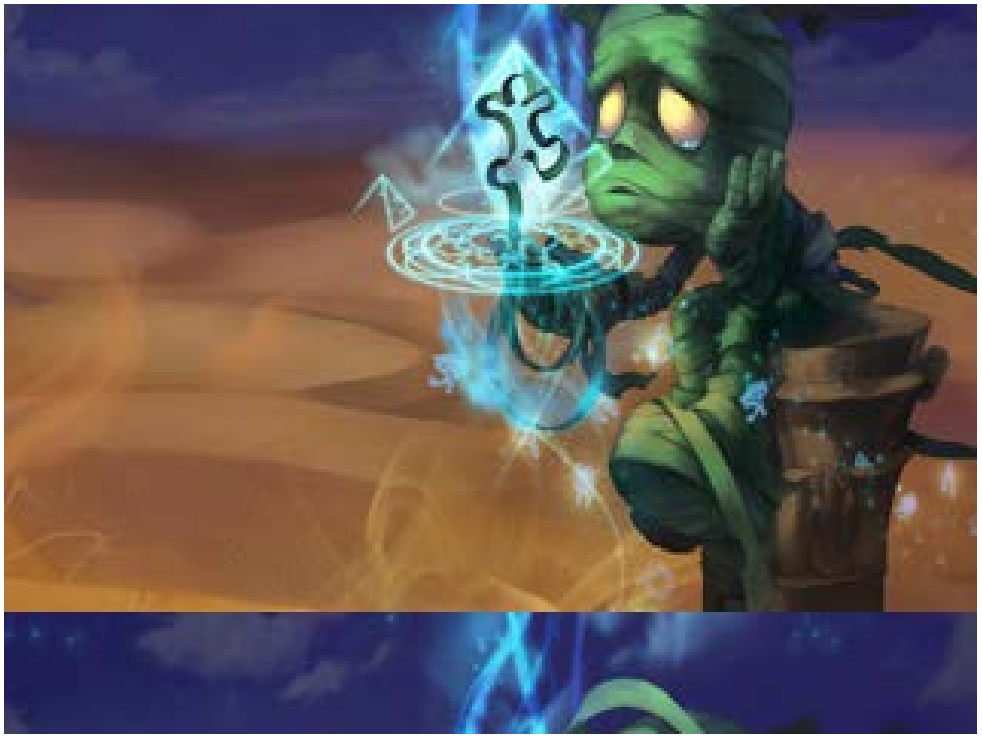
Example (3/6)

(200 rows scrolled down)

Example (4/6)

(300 rows scrolled down)

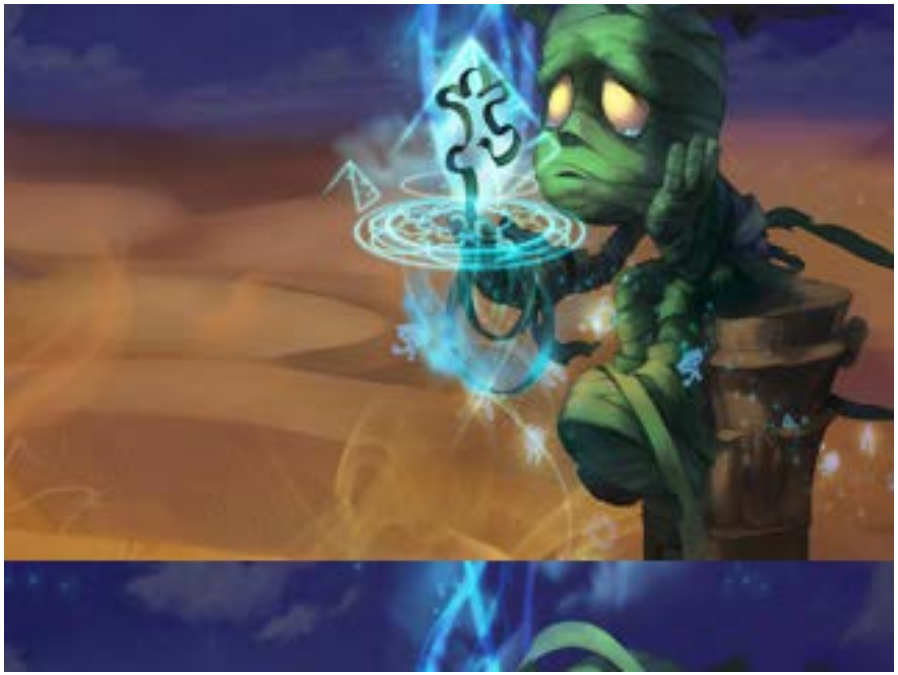
Example (5/6)



(400 rows scrolled down)

Example (6/6)

press **en** ➔ pause



(400 rows scrolled down)