

# LCD背景：

## 基于FPGA逻辑的图像生成

版本 2.3 2025 年 9 月 13 日

课程教材  
逻辑系统与处理器

理查德·苏斯塔



控制工程系  
布拉格捷克技术大学费伊分校



本文档主页及LCD源代码：<https://dcenet.fel.cvut.cz/edu/fpga/guides.aspx>

GHDL安装手册：[https://dcenet.fel.cvut.cz/edu/fpga/install\\_en.aspx](https://dcenet.fel.cvut.cz/edu/fpga/install_en.aspx)

本文提及的FPGA-LCD工具：[https://github.com/cvut/FPGA-LCD\\_Utils](https://github.com/cvut/FPGA-LCD_Utils)

版权所有 (c) 2024, 2025, Richard Susta。

根据自由软件基金会发布的 GNU 自由文档许可证第 1.3 版或任何后续版本的条款，允许复制和分发本文

根据自由软件基金会发布的 [GNU 自由文档许可证](#) 第 1.3 版或任何后续版本的条款进  
行复制和分发；

无不变章节，无封面文本，无封底文本。

作者： 理查德·苏斯塔，[richard@susta.cz](mailto:richard@susta.cz)，<https://susta.cz/> 插图：

理查德·苏斯塔

出版机构： 布拉格捷克技术大学控制工程系CTU-FEE，地址：Technicka 2,  
166 00 Prague 6, 网址：<https://control.fel.cvut.cz/en>

发行日期： 2025年9月

## 目录

引言 .....	3
LCD电路版本 2 .....	4
LCDpackV2.vhd— 定义库 .....	4
VeekMT2_LCDgenV2— 同步发生器 .....	5
VeekMT2_LCDregV2— 向LCD发送颜色 .....	5
LCDlogic0— 图像绘制电路 .....	6
文件 testbenchV2_LCDlogic.vhd .....	6
代码原型 .....	7
runlcd.bat 文件 .....	8
运行GHDL仿真 .....	9
颜色表 .....	10
直线模板 .....	11
椭圆模板 .....	15
问题：为什么我们没有使用条件赋值，例如： - else? .....	17
基于幂次除法的图案生成器 2 .....	18
计数器生成的重复形状 .....	21
从FPGA ROM存储器插入图像 .....	23
位图转换 .....	24
如何从存储器读取图像 .....	27
从存储器插入图像的VHDL代码 .....	28

## 简介

可配置逻辑单元作为FPGA电路的核心组件，在生成某些图像时比加载BMP、JPEG或PNG文件更高效。另一方面，这些文件格式更擅长存储复杂场景。但若要绘制LCD控制面板的背景，其设计完全取决于我们自身。因此我们可以利用逻辑单元擅长的几何图形来构建背景。

下图展示了一个800x480像素的背景示例，该图像在80%质量下存储于33,817字节的JPEG文件中，或以7,384字节的无损压缩PNG（可移植网络图形）文件形式存储——后者对具有重复图案的图形处理更为精细。

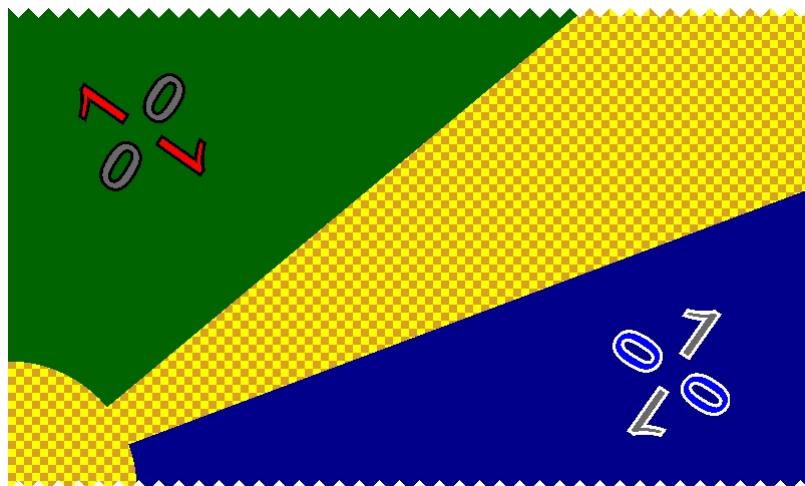


图1 - 逻辑构建的LCD背景示例

当背景采用逻辑实现时，仅需339个逻辑单元（LE）。每个逻辑单元存储2字节，因此占用约680字节空间。此外还需额外4096字节ROM存储器来生成0和1位符号。逻辑实现的背景总计消耗约**4800**字节空间，相当于PNG文件大小的2/3。

然而节省三分之一并非关键因素。PNG和JPEG图像并非连续像素数组编码，其解压过程涉及多个步骤——位图各部分需填充重写，因此必须将完整位图存储在内存中。即使采用经济高效的四位调色板索引编码方案，上图测试背景的解包过程仍需额外占用240千字节FPGA内存。其总存储需求是逻辑部分的51倍。而图像解压过程将拖慢处理器运行速度——该处理器需执行复杂的算法。

此外，该逻辑方案以比特流形式传输像素——这恰恰是LCD面板的工作原理。我们可直接将数据传输至面板。

为完整起见，我们必须提及RLE（[连续序列编码](#)），这是JPEG压缩的一个子步骤，可轻松在硬件中实现，并能输出LCD所需的比特流。对上图进行最优RLE压缩后，文件大小接近36千字节。当然，若FPGA内存空间充足，我们可仅对图像局部应用RLE方法——而[本次任务允许这样做！](#)未来版本的FPGA实用工具V3.0将提供图像转RLE的选项。

然而RLE压缩缺乏可变性。RLE解码器仅能显示图像。若采用逻辑电路创建图案，则可根据输入数据动态修改图案。

我们已创建图形图案模板及对应的VHDL代码作为参考，展示部分逻辑实现方案。所有方案均在[Terasic的](#)Veek-MT2开发板上测试通过，但也可适配其他FPGA及其开发板。

## LCD电路版本2

图像生成在Quartus Lite开发环境中进行了测试。下图架构中的中间实体LCDlogic0相当于绘图模块，它从生成器获取同步信号和像素坐标，为每个坐标分配颜色，并将结果发送至连接LCD面板的寄存器。

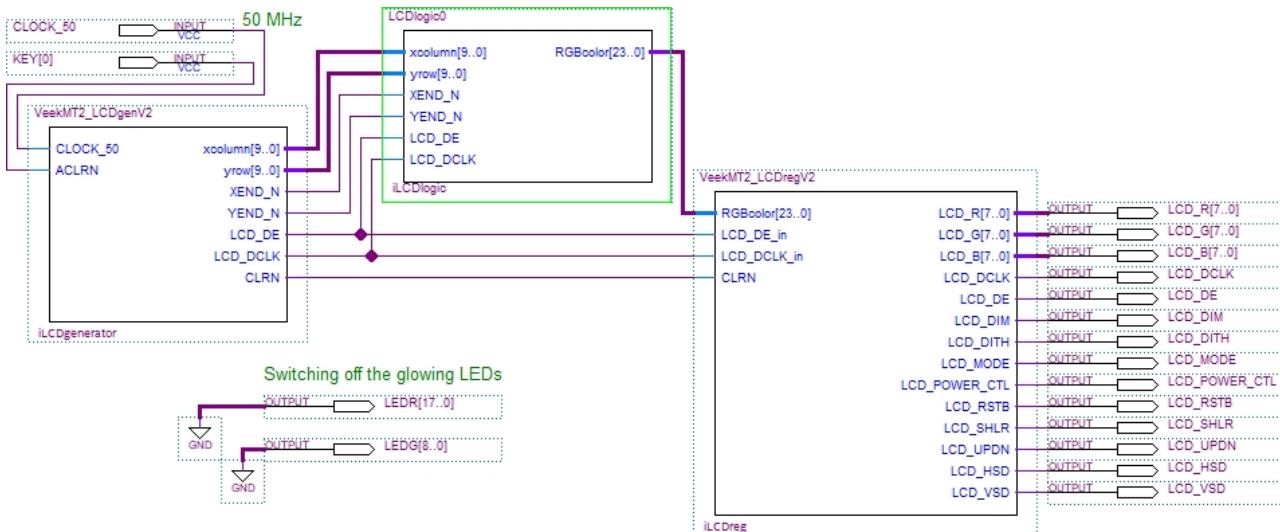


图2 - 图像生成基本电路

**注1：**本文档附带包含全部三个VHDL电路的示例代码附件，即

VeekMT2\_LCDgenV2、VeekMT2\_LCDregV2 以及默认的 LCDlogic0。

**注2：**该架构的工作原理类似于处理器流水线，各时钟负责控制不同阶段的执行。同步生成器发送当前像素的x和y坐标。在下一阶段，该像素在LCDlogic0中被赋予颜色值。颜色数据被加载到寄存器并发送至LCD面板。与此同时，前两个阶段已为后续像素完成处理。

时钟	1: VeekMT2_LCDgenV2	2: LCDlogic0	3: VeekMT2_LCDregV2
上电	-	-	-
时钟1	像素坐标 [x,y]=[0,0]	-	-
时钟2	像素坐标 [x,y]=[1,0]	为[0,0]分配颜色	-
时钟3	像素坐标 [x,y]=[2,0]	赋予颜色 [1,0]	颜色 [0,0] → LCD
时钟4	像素坐标 [x,y]=[3,0]	赋予颜色 [2,0]	颜色 [1,0] → n LCD

## LCDpackV2.vhd – 定义库

LCDpackV2.vhd 是 VHDL 包文件。本文档假定**其版本为 2.1 或更高版本（含附加定义）**。版本号位于文件头部，且 v2.1 与 v2.0 向上兼容。该包定义了用于 LCD 面板几何处理和颜色转换的常量与函数，后续所有 VHDL 代码均引用此包。

主要定义如下：

```

常量 LCD_WIDTH : 整型 := 800;          -- LCD屏幕可见区域的x列轴常量
                                         LCD_HEIGHT : integer := 
                                         480;           -- LCD屏幕可见区域的y行轴常量
                                         XCOLUMN_MAX : 
                                         integer := 1023;      -- 最大x列值位于不可见区域
                                         YROW_MAX       : 整型 := 524;      -- 最大y 行位于不可见区域
                                         子类型 xy_t 为 unsigned(9 downto 0) 类型;   --由LCDgenV2发送的x列和y行数据
                                         常量 XY_ZERO : xy_t := (others => '0');
                                         子类型 RGB_t 为 std_logic_vector(23 downto 0); -- R G B 颜色, R:23..16, G:15..8, B:7..0
                                         
```

```

函数 ToRGB(r, g, b:natural) 返回 RGB_t;
-- + 16种网页颜色常量，亦即（亦称）16种Windows颜色：
- AQUA, BLACK, BLUE, GRAY, GREEN, LIME, OLIVE, MAROON,
- 海军蓝、紫色、红色、银色、蓝绿色、紫罗兰色、白色、黄色

```

注：封装说明详见《[基于VHDL的数据流与结构化电路设计](#)》V10版PDF文件第7章第58至62页。

## VeekMT2\_LCDgenV2 — 同步发生器

VeekMT2\_LCDgenV2 生成器包含一对计数器及其值的比较器。其连接方式简单，主要遵循LCD面板制造商目录中硬件规格的时序要求。

我们可用两个循环在C语言中实现该发生器的类比：

```
unsigned short int xcolumn, yrow;
unsigned int color; bool LCD_DE, XEND_N, YEND_N; for (yrow = 0;
yrow < 525; yrow++)
{ for (xcolumn = 0; xcolumn < 1024; xcolumn++)
{ LCD_DE = xcolumn>=800 || yrow>=480 ? 0 : 1;
XEND_N = xcolumn==1023 ? 0 : 1; YEND_N = yrow==524 ? 0 : 1;
color = LCDlogic0( xcolumn, yrow, XEND_N, YEND_N, LCD_DE);
}
} // 在硬件中, LCDlogic0函数由逻辑电路实现
}
```

### 输入

- **CLOCK\_50** – 开发板同名引脚提供的50 MHz频率输入。根据其内置的PLL（锁相环）要求，该振荡器必须直接连接至此引脚且不得插入任何逻辑电路，该锁相环将频率从50 Hz转换为33 MHz。所有高频电子设备通常都包含若干PLL，通过调整其参数可实现处理器或显卡的超频。
- **ACLRN** – 上电后初始化。在VEEK-MT2开发板上，该引脚连接至KEY[0]。

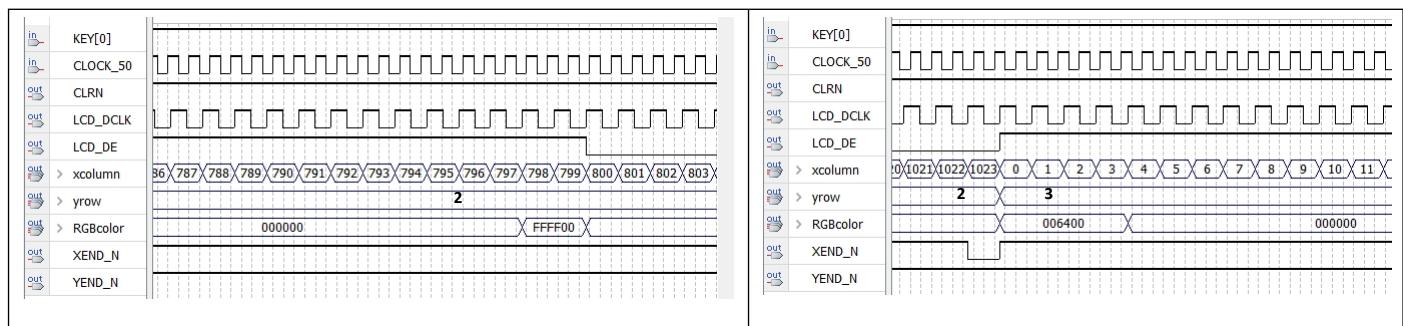


图3 - LCDgenV2输出模拟

## VeekMT2\_LCDregV2 — 向LCD发送颜色信息

在LCD\_DCLK的上升沿，该寄存器存储由LCDlogic0电路分配的颜色。其输出连接至Veek-MT2主板的大型后置LCD屏。该寄存器同时对图像进行裁剪，确保当LCD\_DE='0'时，LCD\_R、LCD\_G和LCD\_B输出均为0（黑色），以满足LCD面板的显示要求。

注：VeekMT2\_LCDregister的输出引脚是在\*.bdf原理图中通过其符号的上下文菜单选择“为符号端口生成引脚”自动生成的。

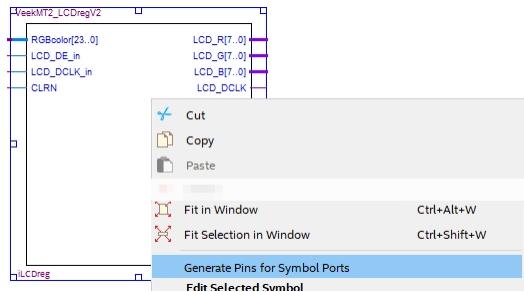


图4 - 为符号端口生成引脚

## LCDlogic0 — 图像绘制电路

LCDlogic从LCD同步发生器接收无符号的x列和y行坐标，其x轴和y轴方向与Windows图形系统一致。

该电路包含组合逻辑，用于为当前x,y像素分配RGBcolor变量。其原型LCDlogic0与发生器及寄存器共同收录于ZIP文件中。



图5 - LCDlogic组合逻辑电路

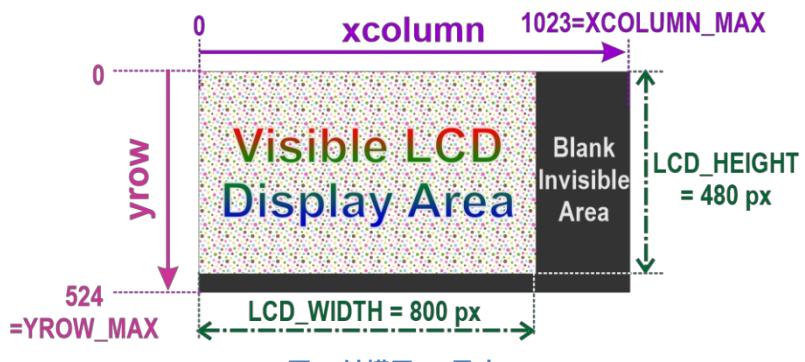


图6 - 触摸屏LCD尺寸

### LCDlogic输入端

**xcolumn, yrow** - 10位像素坐标信号采用无符号类型 `xy_t`，该类型在

`LCDpackV2.vhd` 套件中引入的无符号类型 `xy_t`，该套件同时定义了下列其他常量。

**xcolumn** 列范围为 0 至 `1023=XCOLUMN_MAX`，但可见图像仅位于 0 至 `799=LCD_WIDTH-1` 区间。

**yrow** 行号范围为 0 至 `524=YROW_MAX`，但可见部分仅限于 0 至 `479=LCD_HEIGHT-1` 的区间。

当 `xcolumn=1023` 时，`XEND_N` 为逻辑 '0'；否则为 '1'。该标志表示最后一个列。

其频率为  $32.2 \text{ kHz} = 33 \text{ MHz}/1024 = 33 \text{ MHz}/(\text{XCOLUMN\_MAX}+1)$

当 `yrow=524` 时，`YEND_N` 为逻辑 '0'，否则为 '1'。该信号标记帧的末行。

其频率为  $61.4 \text{ Hz} = 33 \text{ MHz}/(1024*525) = 33 \text{ MHz}/(\text{XCOLUMN\_MAX}+1*\text{YROW\_MAX}+1)$

**LCD\_DE** 是液晶屏数据使能同步信号。当 `LCD_DE= '1'` 时，发送属于可见区域的像素。在第 800 至 1023 列及第 480 至 524 行区域（即可见区域之外），该信号保持 `LCD_DE='0'`。LCD 需要这些不可见区域来写入图像行并准备接收后续行或帧。制造商手册规定了 `LCD_DE` 必须保持为 '0' 且颜色为黑色的时间间隔。VeekMT2\_LCDregister 执行此裁剪操作。

**LCD\_DCLK** – LCD 数据时钟频率精确为 33 MHz，占空比为 50%。

### 输出

**RGBcolor** - 24位std\_logic\_vector，包含8位RGB颜色值。红色值位于高位八位，蓝色值位于低位八位。注意：RGB 不包含带透明度信息的Alpha通道。LCD 通常不支持透明效果，透明度仅在图形处理过程中使用。其输出结果不含Alpha通道。

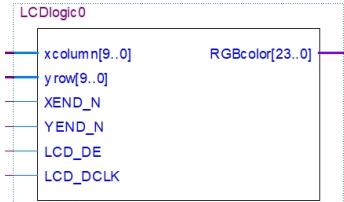
### 文件 testbenchV2\_LCDlogic.vhd

该模块模拟绘制过程，并将像素颜色以压缩形式保存至文本文件。该文件可通过[FPGA-LCD实用工具中的](#)测试台查看器加载，以显示生成的LCD图像。

测试台包含其自身的同步发生器，该发生器经过优化以适应仿真需求，并配有专属寄存器。仅向其中插入LCDlogic\*模块。

## 代码原型

我们将首先使用LCDpackV2中的类型定义输入和输出。假设使用2.1及以上版本，该版本包含assignIf函数。



我们将使用LCDpackV2包中的以下定义：**子类型 xy\_t 为 unsigned(9 downto 0);** -- 用于

数据列 xcolumn 和行 yrow 常量 XY\_ZERO : xy\_t := (others => '0');

子类型 RGB\_t 为 std\_logic\_vector(23 downto 0); -- RGB 颜色, R:23..16, G:15..8, B:7..0

编写实体和架构：

```
library ieee, work; use ieee.std_logic_1164.all; use ieee.numeric_std.all; -- 用于整型和无符号类型
use work.LCDpackV2.all;
实体 LCDlogic0 定义为
port(xcolumn, yrow : in xy_t := XY_ZERO; -- 像素的x,y坐标 (列,行索引)
      XEND_N   : 在标准逻辑中 := '0'; -- 仅当xcolumn=XCOLUMN_MAX时为'0', 否则为'1'; 频率
                  -- 32.2 kHz = LCD_DCKL/1024 = LCD_DCKL/(XCOLUMN_MAX+1)
      YEND_N   : in std_logic := '0'; -- 仅当yrow=YROW_MAX 时为'0', 否则为 '1'; 频率
                  -- 61.4 Hz = LCD_DCKL/(1024*525) = LCD_DCKL/(XCOLUMN_MAX+1)*(YROW_MAX+1)
      LCD_DE : in std_logic := '0'; -- DataEnable指示LCD可见区域
      LCD_DCLK : in std_logic := '0'; -- 33 MHz精确值; LCD数据时钟
      RGBcolor : out RGB_t); -- 定义于LCDpackV2; RGB_t = std_logic_vector(23 downto 0)
end entity;
```

LCDlogic0的行为架构为

常量 DARKBLUE: RGB\_t := ToRGB(0, 0, 139); -- 相当于X"00008B", 添加了LCDpackV2 未定义的颜色  
开始 -- 架构

```
LSPImage: process(xcolumn, yrow, LCD_DE) variable RGB
:RGB_t := BLACK; -- 像素颜色 variable x : integer range 0 to
XCOLUMN_MAX:=0; variable y : integer range 0 to
YROW_MAX:=0; begin -- 进程
x := to_integer(xcolumn); y := to_integer(yrow); -- 将无符号输入转换为整数
----- 我们的图像 -----
RGB := 深蓝色;
```

```
RGBcolor <= RGB; -- 赋值输出信号
end process; end
architecture;
```

**重要提示：**

1. 请保持命名一致性，否则代码将无法编译。文件LCDlogic0.vhd包含**实体LCDlogic0及其行为架构**。行为标识符为局部名称，仅在实体内部有效，可在其他实体中重复使用。
2. 该实体还包含未引用的输入端口以增强通用性。编译器在优化过程中会自动忽略所有未使用资源。但若未来需要新增输入端口，这些预留端口可直接调用，无需重新添加至实体或再生成原理图符号。
3. 在定义中初始化信号和变量值主要用于仿真。综合仅针对常量定义或函数与过程中的局部变量进行。  
**该过程需在其主代码中通过赋值语句进行初始化。**
4. **关键字process**开启VHDL顺序域。LSPImage为可选标志，综合阶段不可引用，但在仿真中将作为引用出现。
5. **关键字process后**括号内的列表并非参数，而是“敏感性列表”，用于列出其输出可能因信号变化而改变的信号。该列表对仿真必不可少！

## runled.bat文件

runlcd.bat批处理文件包含脚本化Shell语言命令，其结构类似于DCENET手册《[GHDL语言安装与使用指南](#)》第7页所述的 runmorse.bat文件。但该文件不为GtkWave生成输出，因此"ghdl.exe -r"命令中省略了--vcd参数。最终结果将通过FPGA-LCD实用工具中的测试台查看器进行展示。

```
@ECHO OFF  
rem SETLOCAL — 批处理结束后以下定义将自动取消。重要提示：切勿省略！  
SETLOCAL  
注：测试台文件名必须不带扩展名，因为其名称也会用于其他组件  
set TBNAME=testbenchV2_LCDlogic  
rem 文件包含扩展名及相对于父目录的相对路径。请按正确编译顺序列出！  
set FILES=../LCDpackV2.vhd ..//LCDlogic0.vhd  
rem 模拟运行时采用其自身时间单位。  
set SIMTIME=20ms  
rem 将 mingw64 移至 PATH 路径顶部（仅临时生效，因使用 SETLOCAL）
```

```
rem GHDL 编译基于 VHDL-2008 规范  
set GHDL_FLAGS=-fsynopsys --std=08 @ECHO ON  
ghdl.exe -a %GHDL_FLAGS% %FILES% ./%TBNAME%.vhd@IF ERRORLEVEL 1  
GOTO BAT-END  
ghdl.exe -e %GHDL_FLAGS% %TBNAME% @IF  
ERRORLEVEL 1 GOTO BAT-END  
ghdl.exe -r %GHDL_FLAGS% %TBNAME% --stop-time=%SIMTIME%  
:BAT-END
```

其命令说明如下：

ECHO OFF – 处理此\*.bat文件时不显示已执行命令。

启用时，将显示非@符号开头的命令。

SETLOCAL - 执行环境变量局部化设置。更改将持续有效直至批处理文件结束或遇到对应的ENDLOCAL命令。若不使用SETLOCAL，  
**这些更改将永久生效！**

rem - 以下文本为注释，直至行尾。

set TBNAME – 测试台实体，**仅指其名称**，不含扩展名或路径。

该参数集以 %its\_name% 形式引用，例如 %TBNAME%。

set FILES= – 用于组装电路的文件。文件名以空格分隔，必须按编译顺序列出！请勿添加同步生成器和寄存器——测试台已  
包含这些组件。

设置模拟时间（SIMTIME）——模拟应运行16.6毫秒（其内部时间）后自动停止。若未停止，则表明存在错误，系统将在  
20毫秒后强制终止。

set PATH – 仅临时将 mingw64 路径移至首位（因先前 SETLOCAL 设置） set GHDL\_FLAGS – 启用 VHDL 2008 支持。GHDL 可处理其  
中绝大部分特性。

ghdl.exe -a – 分析VHDL文件。其命令行中，FILEs必须位于TBNAME之前。

IF ERRORLEVEL 1 GOTO BAT-END - 若前一条命令执行失败则跳转至文件末尾。

ghdl.exe -e – 在 TBNAME.exe 文件中创建电路仿真

ghdl.exe -r – 运行TBNAME.exe

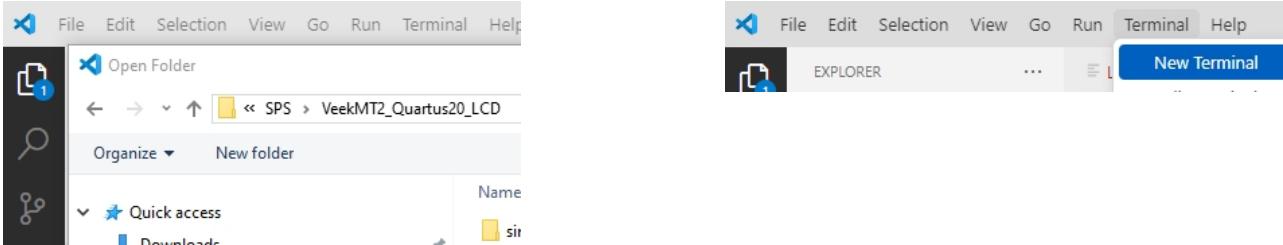
**注意：**若需模拟其他VHDL文件，请修改命令

**设置** FILES=，若使用不同测试台**则需设置** TBNAME

## 运行GHDL仿真

GHDL可实现更快速的调试。示例项目包含`runlcd.bat`文件，该文件被刻意放置在模拟子目录，确保GHDL创建的所有临时文件都保存在该目录中。

首先，在免费应用程序VSC（即Visual Studio Code）中打开项目文件夹。随后创建一个新终端。



在终端中输入两条 Windows PowerShell 命令：

```
PS C:\SPS\VeeekMT2_Quartus20_LCD> cd .\simulation  
PS C:\SPS\VeeekMT2_Quartus20_LCD\simulation> ./runlcd.bat
```

输入 `cd` 后按Tab键，VSC将自动补全命令。同理，输入`./r`后按Tab键可补全后续内容。仿真运行时将列出执行命令，并以以下提示成功结束：

:-) 完成单帧仿真。

```
.\testbenchv2_lcdlogic.exe: 错误: 在进程.testbenchv2_lcdlogic(testbench).stimulus 中断言失败  
.testbenchv2_lcdlogic.exe: 错误: 仿真失败  
PS C:\SPS\VeeekMT2_Quartus20_LCD\simulation>
```

忽略“错误”消息后 **:)** 结束。矛盾的是，报告致命错误却显示一切正常 **:)**，正是VHDL中终止模拟的机制。

测试平台结果可通过FPGA LCD工具包的测试平台查看器进行查看。

在原型代码（第7页）中，我们为可见区域和不可见区域的所有像素都赋予了深蓝色，这在将测试台查看器切换为全屏模式时可能造成混淆，如下图所示。为便于辨识，**可使用if命令添加裁剪功能。**

该寄存器已实现裁剪功能，因此无需在LCDlogic中重复添加，但为便于我们自身定位仍可添加。后续代码将不再包含此功能。

----- 我们的图像 -----  
----- 我们的图像 -----  
RGB color<= DARKBLUE;  
if LCD\_DE = '0' then RGBcolor <= BLACK; end if;

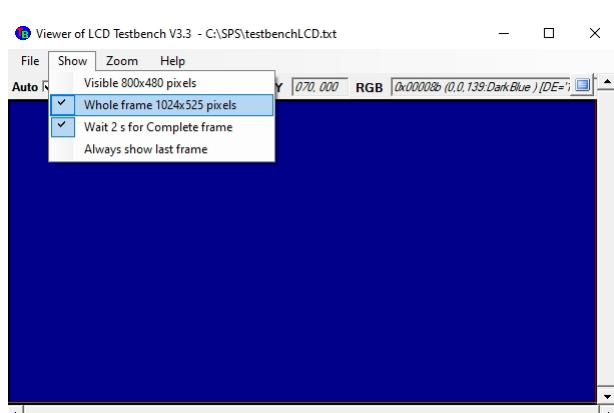


图7- 所有像素均为深蓝色



图8-裁剪  
+ 突出显示位置 XEND\_N 和 YEND\_N

## 颜色表

[Reddit上有个按色调整理的实用颜色表。网页十六进制代码在VHDL中用X表示。例如栗色#800000在VHDL中写为X"800000"，或使用转换函数ToRGB\(128, 0, 0\)。](#)

注：十六进制格式多样，详见维基概述：[与十进制区分](#)

## HEXADECIMAL COLOR CODES

Color	Hex Code #RRGGBB	Color	Hex Code #RRGGBB	Color	Hex Code #RRGGBB
maroon	#800000	aqua	#00FFFF	beige	#F5F5DC
dark red	#8B0000	cyan	#00FFFFFF	bisque	#FFE4C4
brown	#A52A2A	light cyan	#E0FFFF	blanched almond	#FFEBBC
firebrick	#B22222	dark turquoise	#00CED1	wheat	#F5DEB3
crimson	#DC143C	turquoise	#40E0D0	corn silk	#FFF8DC
red	#FF0000	medium turquoise	#48D1CC	lemon chiffon	#FFFACD
tomato	#FF6347	pale turquoise	#AFEEEE	light golden rod yellow	#FAFAD2
coral	#FF7F50	aqua marine	#7FFFBD	light yellow	#FFFFE0
indian red	#CD5C5C	powder blue	#B0E0E6	saddle brown	#8B4513
light coral	#F08080	cadet blue	#5F9EA0	sienna	#A0522D
dark salmon	#E9967A	steel blue	#4682B4	chocolate	#D2691E
salmon	#FA8072	corn flower blue	#6495ED	peru	#CD853F
light salmon	#FFA07A	deep sky blue	#00BFFF	sandy brown	#F4A460
orange red	#FF4500	dodger blue	#1E90FF	burly wood	#DEB887
dark orange	#FF8C00	light blue	#ADD8E6	tan	#D2B48C
orange	#FFA500	sky blue	#87CEEB	rosy brown	#BC8F8F
gold	#FFD700	light sky blue	#87CEFA	moccasin	#FFE4B5
dark golden rod	#B8860B	midnight blue	#191970	navajo white	#FFDEAD
golden rod	#DAA520	navy	#000080	peach puff	#FFDAB9
pale golden rod	#EEE8AA	dark blue	#00008B	misty rose	#FFE4E1
dark khaki	#BDB76B	medium blue	#0000CD	lavender blush	#FFF0F5
khaki	#F0E68C	blue	#0000FF	linen	#FAF0E6
olive	#808000	royal blue	#4169E1	old lace	#FDF5E6
yellow	#FFFF00	blue violet	#8A2BE2	papaya whip	#FFEFD5
yellow green	#9ACD32	indigo	#4B0082	sea shell	#FFF5EE
dark olive green	#556B2F	dark slate blue	#483D8B	mint cream	#F5FFFA
olive drab	#6B8E23	slate blue	#6A5ACD	slate gray	#708090
lawn green	#7CFC00	medium slate blue	#7B68EE	light slate gray	#778899
chartreuse	#7FFF00	medium purple	#9370DB	light steel blue	#B0C4DE
green yellow	#ADFF2F	dark magenta	#8B008B	lavender	#E6E6FA
dark green	#006400	dark violet	#9400D3	floral white	#FFF0F0
green	#008000	dark orchid	#9932CC	alice blue	#F0F8FF
forest green	#228B22	medium orchid	#BA55D3	ghost white	#F8F8FF
lime	#00FF00	purple	#800080	honeydew	#F0FFF0
lime green	#32CD32	thistle	#D8BFD8	ivory	#FFFFFF
light green	#90EE90	plum	#DDA0DD	azure	#F0FFFF
pale green	#98FB98	violet	#EE82EE	snow	#FFFAFA
dark sea green	#8FBBC8F	magenta / fuchsia	#FF00FF	black	#000000
medium spring green	#00FA9A	orchid	#DA70D6	dim gray / dim grey	#696969
spring green	#00FF7F	medium violet red	#C71585	gray / grey	#808080
sea green	#2E8B57	pale violet red	#DB7093	dark gray / dark grey	#A9A9A9
medium aqua marine	#66CDAA	deep pink	#FF1493	silver	#C0C0C0
medium sea green	#3CB371	hot pink	#FF69B4	light gray / light grey	#D3D3D3
light sea green	#20B2AA	light pink	#FFB6C1	gainsboro	#DCDCDC
dark slate gray	#2F4F4F	pink	#FFC0CB	white smoke	#F5F5F5
teal	#008080	antique white	#FAEBD7	white	#FFFFFF
dark cyan	#008B8B				

图9 - 取自[Reddit的命名颜色表](#)

## 直线模板

通过两个不同点作直线的方程可由正比例关系推导。但硬件实现直线时需使用整数系数。若最大公约数 gcd 超过 1，可将斜率分數缩减为更小数值，从而简化电路设计。

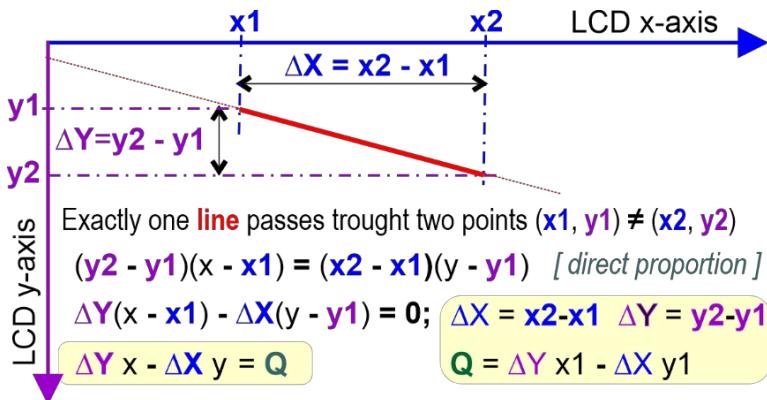


图10 - 直线方程推导过程

直线（以及椭圆）的方程可通过FPGA-LCD实用工具中的LCD几何尺计算获得，该工具与知名的Geogebra工具极为相似，但经过适配以支持整数结果及LCD坐标系——因历史原因，该坐标系中y轴方向为自上而下。

LCDlogic0的架构行为定义为：

常量 DARKBLUE: RGB\_t := ToRGB(0, 0, 139);

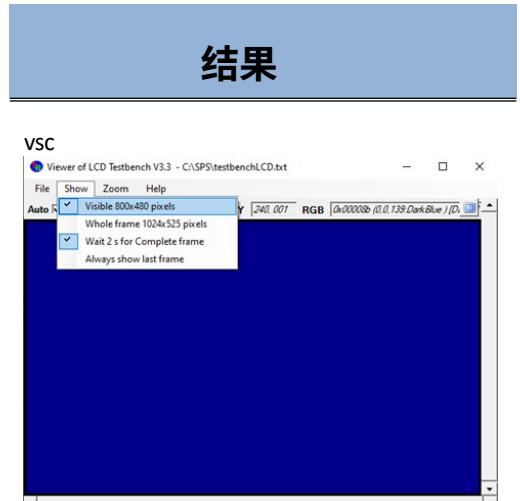
begin -- 架构

```
LSPImage : process (xcolumn, yrow, LCD_DE) variable RGB :RGB_t
:=BLACK; -- xy像素点颜色variable x : integer range 0 to
XCOLUMN_MAX:=0;variable y : integer range 0 to
YROW_MAX:=0;begin  x := to_integer(xcolumn);  y :=
to_integer(yrow);
----- 我们的图像 -----
RGB :=深蓝色;
```

RGBcolor <= RGB; -- 赋值输出信号

end process;

架构结束；



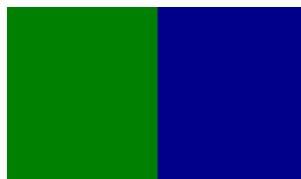
在以下代码中，仅“**我们的图像**”部分的行发生变化。

我们更倾向于使用多个独立的if-then语句。优先级较高的if语句应置于优先级较低的之后。相较于冗长的if-elsif-elsif...级联语句，这种风格更易于理解。经测试，Quartus对独立if-then语句的实现效果与if-elsif-elsif级联同样高效。此外，在嵌套的if-elsif-elsif语句中，代码拼写错误的发生率高于独立的if-then语句组合。这一结论同样得到了众多学生实践验证②

----- 我们的图片 -----

RGB:= DARKBLUE; -- 默认值 !!! 必须 !!!

如果 x < LCD\_WIDTH/2 则 RGB := GREEN; 结束如果;



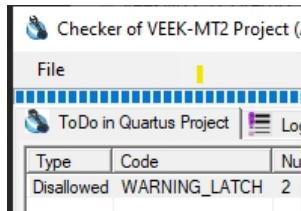
-- 未为RGB赋默认值的错误代码

----- 我们的图像 -----

if x< LCD\_WIDTH/2 then RGB:=GREEN; end if;

----- 在我们的代码中，RGB在处理过程中并非始终被赋值

--编译器提示 (10041): 为 LSPImage:RGB 推断出锁存器



----- 我们的图像 -----RGB := 深

蓝;

if x<LCD\_HEIGHT/2 then RGB:=YELLOW; end if;

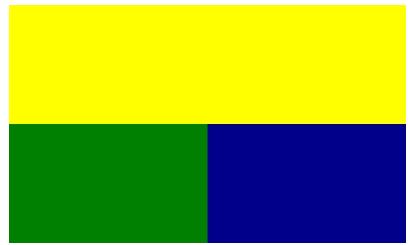


----- 我们的图像 -----RGB := 深

蓝色;

if x<LCD\_WIDTH/2 then RGB:=GREEN; end if;

如果 y < LCD\_HEIGHT/2 则 RGB := YELLOW; 结束如果;

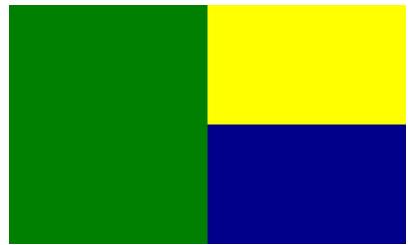


----- 我们的图像 -----RGB := 深

蓝色;

如果 y < LCD\_HEIGHT/2 则 RGB := YELLOW; 结束如果; 如果 x <

LCD\_WIDTH/2 则 RGB := GREEN; 结束如果;

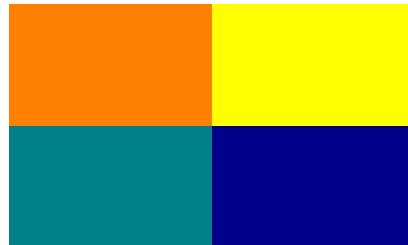


----- 我们的图像 -----RGB := 深

蓝色;

如果 y < LCD\_HEIGHT/2 则 RGB := YELLOW; 结束如果;

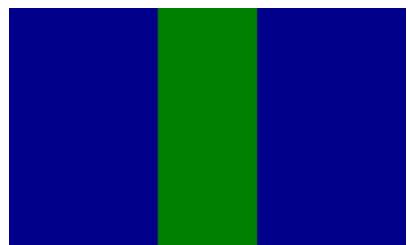
如果 x < LCD\_WIDTH/2 则 RGB := RGB xor GREEN; 结束如果;



----- 我们的图像 -----RGB := 深

蓝色;

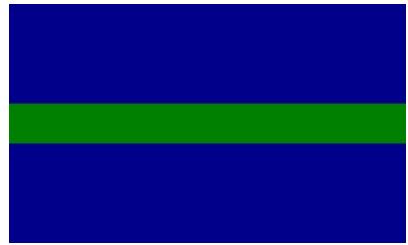
if (x≥300) and (x<500) then RGB := GREEN; end if;



----- 我们的图像 -----RGB := 深

蓝色;

if (y≥200) and (y<280) then RGB := GREEN; end if;

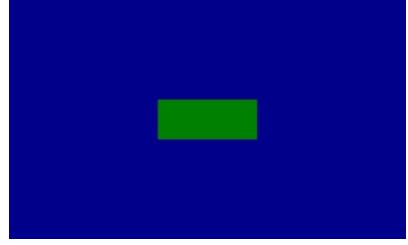


----- 我们的图像 -----RGB := 深

蓝色;

如果 (x≥300) 且 (x<500) 且 (y≥200) 且 (y<280)

则 RGB := GREEN; 结束条件语句;

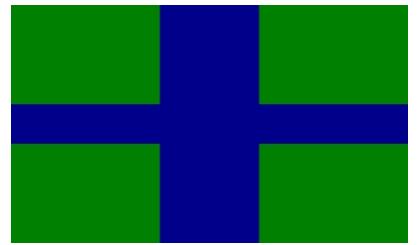


----- 我们的图像 -----

RGB := 深蓝;

如果 (( $x < 300$ ) 或 ( $x \geq 500$ )) 且 (( $y < 200$ ) 或 ( $y \geq 280$ ))

然后 RGB := GREEN; 结束 if;

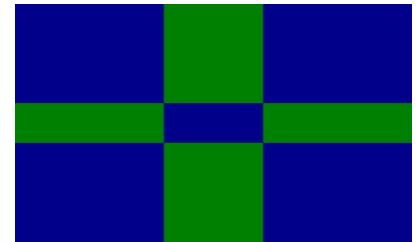


----- 我们的图像 -----

RGB := 深蓝;

如果 ( $x < 300$  或  $x \geq 500$ ) 且 ( $y < 200$  或  $y \geq 280$ )

则 RGB := GREEN; 结束 if;

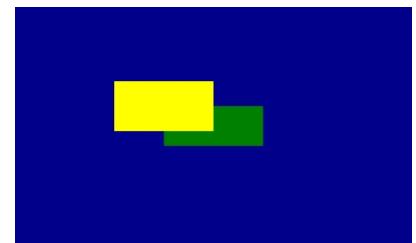


----- 我们的图像 -----

RGB := 深蓝;

如果 ( $x \geq 300$  且  $x < 500$  且  $y \geq 200$  且  $y < 280$ ) 则 RGB := GREEN; end if; if ( $x \geq 200$  and  $x < 400$

and ( $y \geq 150$  and  $y < 250$ ) then RGB := YELLOW; end if;

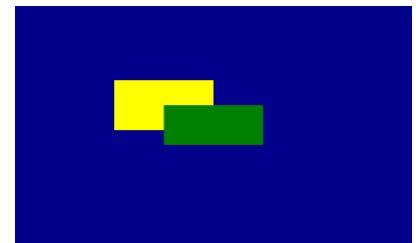


----- 我们的形象 -----

RGB := 深蓝色;

如果 ( $x \geq 200$  且  $x < 400$  且  $y \geq 150$  且  $y < 250$ ) 则 RGB := YELLOW; end if; if ( $x \geq 300$  and  $x < 500$

and ( $y \geq 200$  and  $y < 280$ ) then RGB := GREEN; end if;

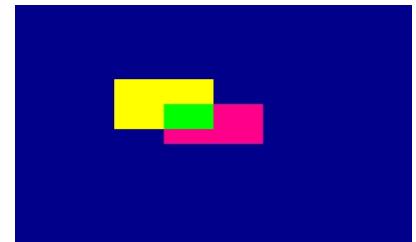


----- 我们的图像 -----

RGB := 深蓝;

if ( $x \geq 200$  and  $x < 400$  and  $y \geq 150$  and  $y < 250$ ) then RGB := YELLOW; end if;

if ( $x \geq 300$  and  $x < 500$  and  $y \geq 200$  and  $y < 280$ ) then RGB := RGB xor RED; end if;



FPGA-LCD实用工具中的LCD几何尺规还能计算倾斜线段的系数。打开一张LCD尺寸为800x480像素的图像（例如从测试台查看器保存的图像），插入我们的线段并优化其位置（方向盘图标）。优化器将调整线段端点X2、Y2坐标以寻找最大公约数。

例如，斜率为 $641/480$ 的直线具有最大公约数（gcd 1）。斜率为 $640/480$ 的直线更优，两者仅相差0.07度。我们将该斜率缩短160至 $4/3$ 。实现过程中将采用更小的乘数，从而减少所需逻辑单元数量。

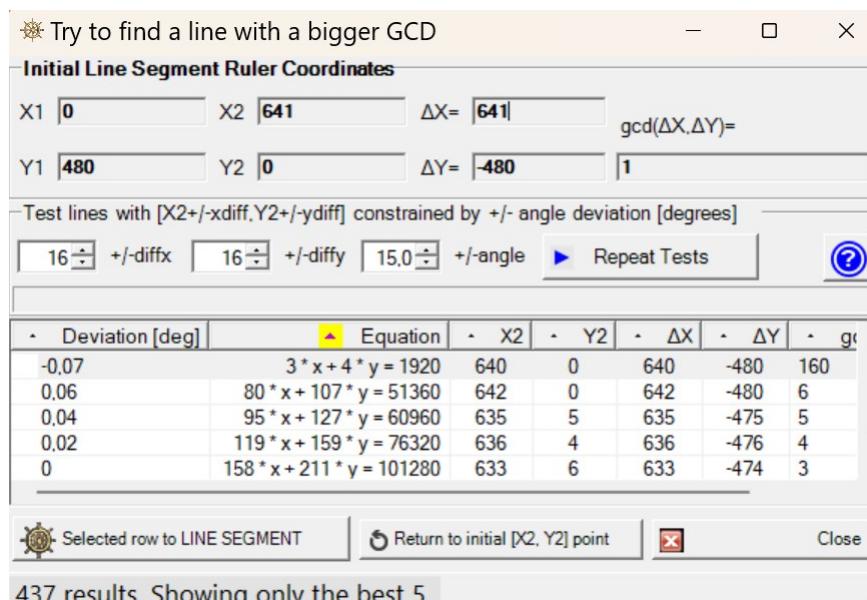


图11 - FPGA-LCD实用工具中LCD几何尺规的直线优化对话框

若在线方程中用合适的不等式替代等号，即可将其作为为整个LCD区域赋色的条件。通过组合条件，可创建由直线围成的形状，如下图所示（图源自测试台查看器输出结果）。

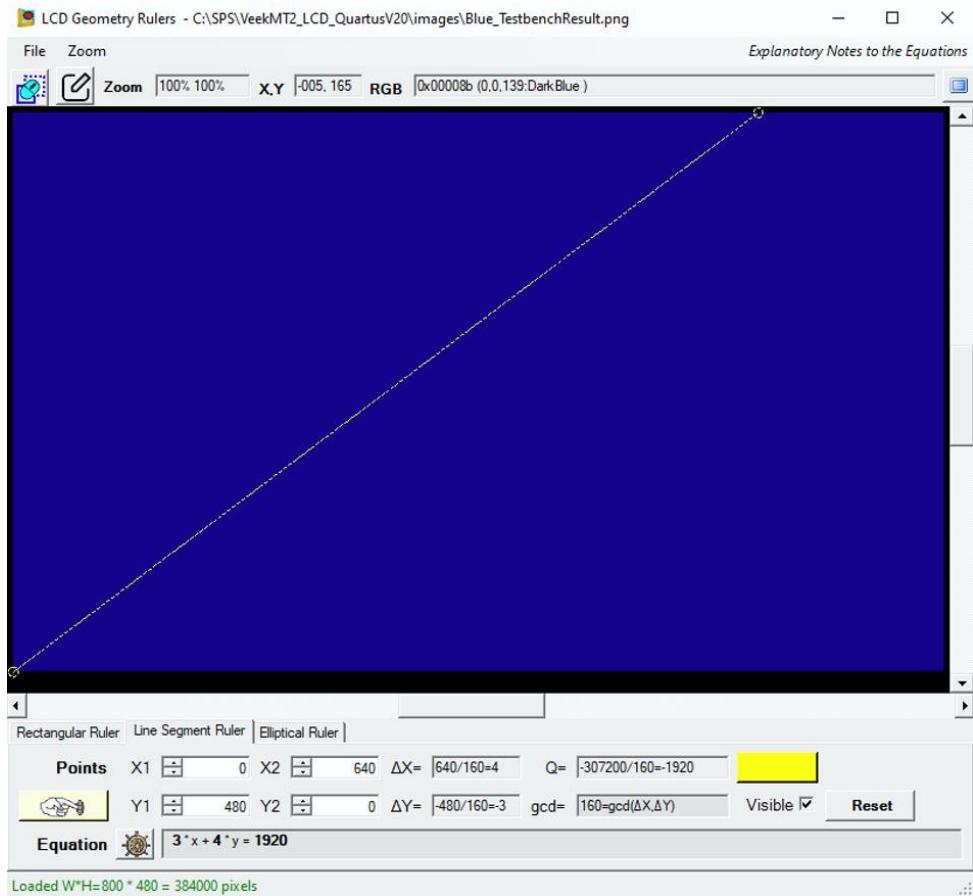
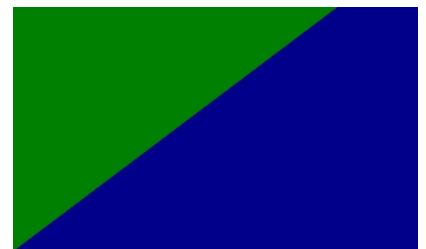


图12 - LCD几何尺规——最优直线方程

----- 我们的形象 -----

RGB:=深蓝色;

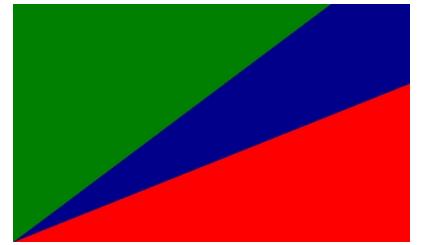
如果  $3 * x + 4 * y < 1920$  则 RGB:=GREEN; 结束如果;



----- 我们的图像 -----

RGB:=深蓝色;

如果  $3 * x + 4 * y < 1920$  则 RGB:=GREEN; 结束如果; 如果  $2 * x + 5 * y > 2400$  则 RGB:=RED; 结束如果;

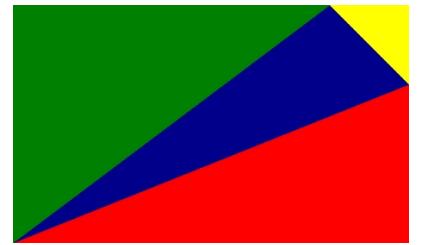


----- 我们的图像 -----

RGB:=深蓝;

如果  $x - y >= 640$  则 RGB:=YELLOW; 结束如果;

如果  $3 * x + 4 * y < 1920$  则 RGB:=GREEN; 结束如果; 如果  $2 * x + 5 * y > 2400$  则 RGB:=RED; 结束如果;



## 椭圆模板

若椭圆具有水平与垂直轴，则处于标准形。若其方程系数可被最大公因数 (gcd) 整除，硬件实现效果更佳。

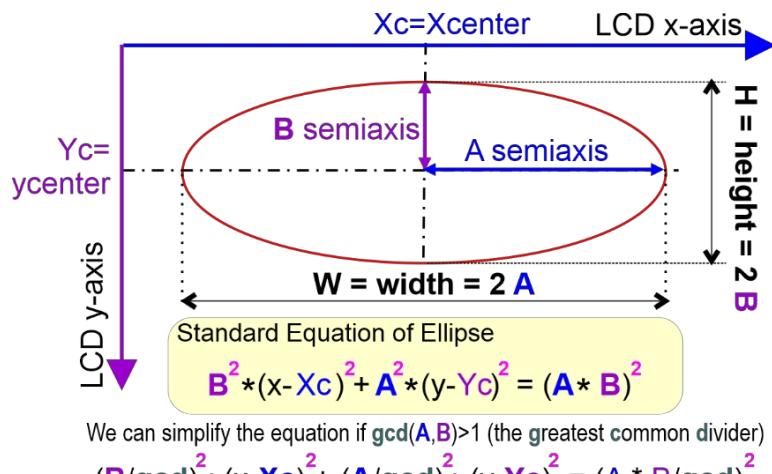


图13 - 标准形式的椭圆方程

我们仍可使用LCD几何尺规优化搜索：优先选择gcd值更高且硬件实现更优的邻近椭圆（方向盘图标）。

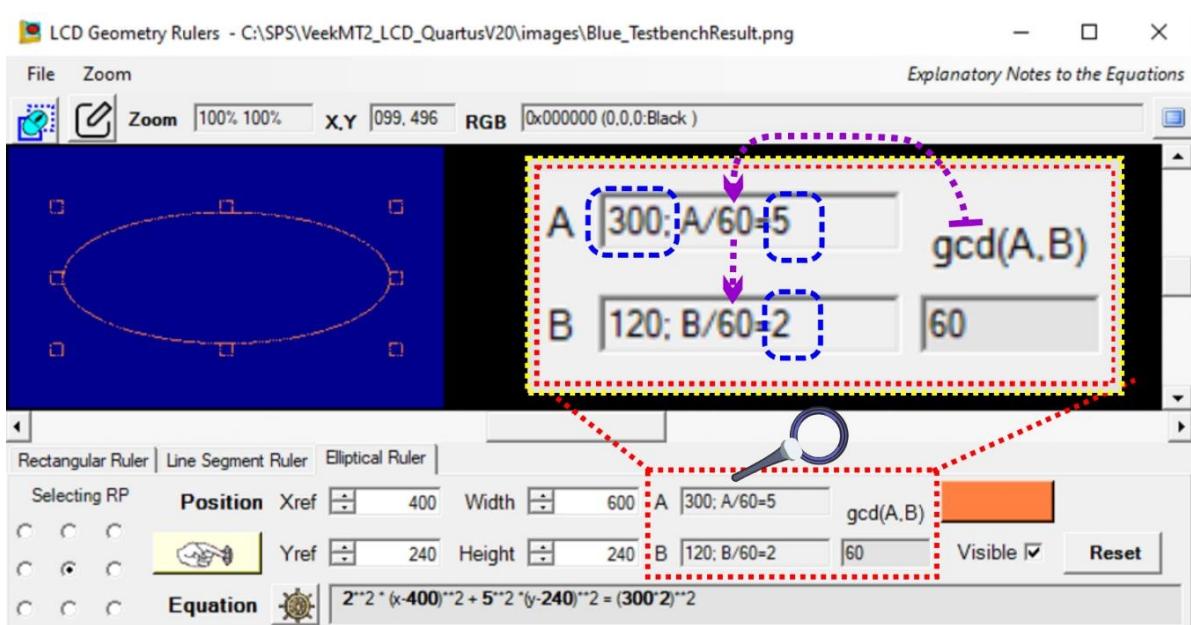


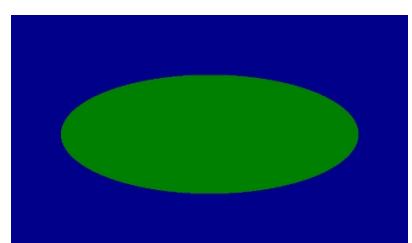
图14 - 求椭圆方程

----- 我们的图像 -----

RGB:=深蓝色;

--  $(2=B/\text{gcd})^{**2}$        $(5=A/\text{gcd})^{**2}$        $(A^*(B/\text{gcd}))^{**2}$

如果  $2^{**2} * (x-400)^{**2} + 5^{**2} * (y-240)^{**2} < (300*2)^{**2}$  则 RGB:=GREEN; 结束如果;

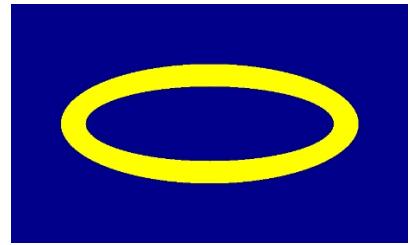


一般椭圆的轴线旋转角度为 $\theta$ ，当对其坐标系施加欧几里得旋转时，其二次方程可由标准形推导得出。相关公式（需代入标准形中的A、B系数及角度 $\theta$ ）可参见英文维基百科页面中的“一般椭圆”章节，或WolframCloud平台的“细节与选项”部分。

然而在设计LCD背景时，我们应采用规范形式的椭圆片段进行组合，这将更为便捷。

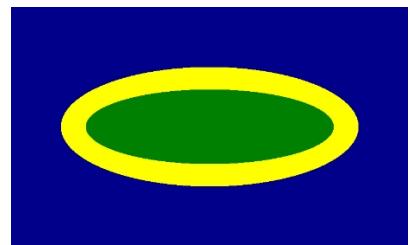
----- 我们的图像 -----

RGB:=深蓝色;  
--  $(B/gdc)^2$        $(A/gdc)^2$        $(A \cdot B/gdc)^{**2}$   
若  $2^{**2} \cdot (x-400)^{**2} + 5^{**2} \cdot (y-240)^{**2} < (300^2)^{**2}$   
且  $3^{**2} \cdot (x-400)^{**2} + 10^{**2} \cdot (y-240)^{**2} > (250 \cdot 3)^{**2}$   
则 RGB:=YELLOW; 结束 if;



----- 我们的图像 -----

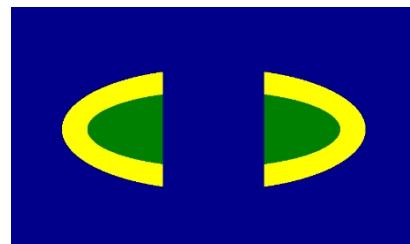
RGB:=深蓝;  
如果  $2^{**2} \cdot (x-400)^{**2} + 5^{**2} \cdot (y-240)^{**2} < (300^2)^{**2}$   
则 RGB:=YELLOW; 结束条件语句;  
如果  $3^{**2} \cdot (x-400)^{**2} + 10^{**2} \cdot (y-240)^{**2} < (250 \cdot 3)^{**2}$   
则 RGB:=GREEN; 结束 if;



RGB:=深蓝;

如果  $(x < 300)$  或  $(x \geq 500)$  则

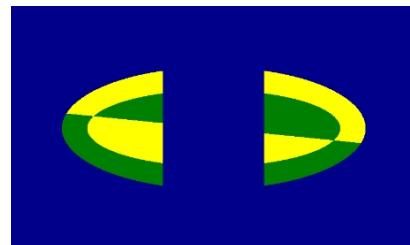
如果  $2^{**2} \cdot (x-400)^{**2} + 5^{**2} \cdot (y-240)^{**2} < (300^2)^{**2}$   
则 RGB:=YELLOW; 结束 if;  
如果  $3^{**2} \cdot (x-400)^{**2} + 10^{**2} \cdot (y-240)^{**2} < (250 \cdot 3)^{**2}$   
则 RGB:=GREEN; 结束条件;  
end if;



下图包含复杂代码，因此我们插入其完整架构，而非仅限于“我们的图像”部分

LCDlogic0的行为架构为：

常量 DARKBLUE: RGB\_t := ToRGB(0, 0, 139); -- 背景色  
begin  
LSPImage : process( xcolumn, yrow, LCD\_DE )  
变量 RGB :RGB\_t :=BLACK; -- 当前像素颜色变量 x : 整型 范围 0 至  
XCOLUMN\_MAX:=0; 变量 y : 整型 范围 0 至 YROW\_MAX:=0;  
变量 isAboveLine: 布尔值:=false; -- 位于我们的直线上方



begin -- 处理  
x := to\_integer(xcolumn); y := to\_integer(yrow); -- 将无符号输入转换为整数  
our image  
RGB:=深蓝色; isAboveLine:=( x - 10\*y >= -2000 );  
if (x<300) or (x>=500) then  
  if  $2^{**2} \cdot (x-400)^{**2} + 5^{**2} \cdot (y-240)^{**2} < (300^2)^{**2}$       则  
    如果 在直线上 则 RGB:=YELLOW; 否则 RGB:=GREEN; 结束如果; 结束如果;  
    如果  $3^{**2} \cdot (x-400)^{**2} + 10^{**2} \cdot (y-240)^{**2} < (250 \cdot 3)^{**2}$  则  
      如果 高于线则 RGB:=绿色; 否则 RGB:=黄色; 结束如果; 结束如果;  
  end if; -- if ((x<300) or (x>=500)) then

  RGBcolor <= RGB; -- 赋值输出信号  
  结束过程;

end architecture;

Veek-MT2开发板搭载Cyclone IV FPGA，包含115,000个逻辑单元(LE)。上图仅需177个LE，约占360字节。为此使用了十个硬件9位乘法器，仅占FPGA总乘法器数量的2%。

保存为PNG格式的图像约占用6.6KB空间，而质量设为80%的JPEG文件则可能高达15KB。

## 问题：为何不采用when-else条件赋值结构？

VHDL-2008支持when-else条件赋值结构，相当于C语言中的?:运算符。对应的VHDL代码如下：

```
-----our image-----
RGB:=DARKBLUE; isAboveLine:=(x - 10 * y >= -2000);
如果 (x<300) 或 (x≥500) 则
    如果 2**2 *(x-400)**2 + 5**2 *(y-240)**2 < (300*2)**2      则
        如果在直线上方则RGB:=黄色;否则RGB:=绿色;结束如果;
        RGB:=YELLOW when isAboveLine else GREEN;
    end if;
    如果 3**2 *(x-400)**2 + 10**2 *(y-240)**2 < (250*3)**2 则
        如果在直线上则RGB:=绿色;否则RGB:=黄色;结束如果;
        RGB:=GREEN 当 isAboveLine 否则 YELLOW;
    end if;
end if; -- if ((x<300) or (x≥500)) then
```

GHDL仿真器支持几乎全部VHDL-2008规范，因此我们可以使用更简洁的when-else语句。遗憾的是，Quartus Lite免费版仅允许使用VHDL 2008片段，不支持此便捷操作::( 该功能仅存在于付费版本。若需将结果上传至开发板，必须在Quartus中编译，因此我们省略了免费版会拒绝的构造。

但可通过便捷函数替代when-else：

```
function assignIf(cond:boolean; colorTrue, colorFalse:RGB_t) return RGB_t is begin
    if cond then return colorTrue; else return colorFalse; end if; end function;
```

该功能包含在LCDPackV2版本V2.1及更高版本中。

LCDlogic0 的架构行为定义为：

```
常量 DARKBLUE: RGB_t := ToRGB(0, 0, 139); -- 背景色
begin -- 架构
LSPImage : process(xcolumn, yrow, LCD_DE)
    变量 RGB:RGB_t := BLACK; -- 当前像素颜色
    变量 x:整型 范围 0 至 XCOLUMN_MAX:=0;
    变量 y:整型 范围 0 至 YROW_MAX:=0;
    变量 isAboveLine:布尔值:=false; -- 直线上方
begin -- 过程
    x := to_integer(xcolumn); y := to_integer(yrow); -- 将无符号输入转换为整数
-----our image-----
```

```
RGB:=深蓝色;是否高于线:=(x - 10*y >= -2000);
if (x<300) or (x≥500) then
    如果 2**2 *(x-400)**2 + 5**2 *(y-240)**2 < (300*2)**2 则
        RGB:= assignIf(isAboveLine, YELLOW, GREEN);
    end if;
    如果 3**2 *(x-400)**2 + 10**2 *(y-240)**2 < (250*3)**2 则
        RGB:= assignIf(isAboveLine, GREEN, YELLOW);
    end if;
end if;-- if ((x<300) or (x≥500)) then
```

RGBcolor <= RGB; -- 赋值输出信号

结束处理；结束架构

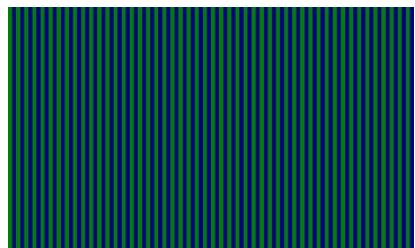
;

assignIf必须针对每种类型单独定义，这相较于更通用的when-else结构是其缺点，但它支持类似C语言的重载机制。本包包含针对整数的assignIf实现。

## 基于2的幂次除法的模式生成器

逻辑方程能有效生成自循环的图形。其原理在于每个LCD帧都是以像素流形式生成的。若将发送至选定元素的坐标改为周期性坐标，图形便会循环重复。例如，根据整数除以8的余数结果调整颜色： $x \bmod 8 = 2^{(3)}$  在硬件中，表达式 $((x / 8) \bmod 2) = 0$ 通过检测第3位实现，即 $x\text{column}(3) = '0'$ 。

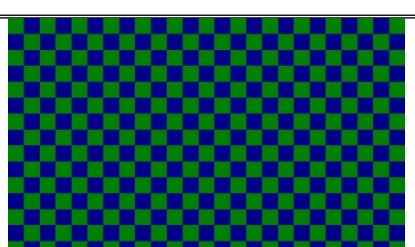
----- 我们的图像 -----  
----- RGB:=深蓝; -----  
**if**  $((x / 8) \bmod 2) = 0$  **then** RGB:=GREEN; **end if;** **if** LCD\_DE= '0' **then** RGB:=BLACK; **end if;**



----- 我们的图像 -----  
----- RGB:=深蓝; -----  
**if**  $(x\text{column}(3) \text{ xor } y\text{row}(3)) = '0'$  **then** RGB:=GREEN; **end if;** **if** LCD\_DE= '0' **then** RGB:=BLACK; **end if;**



----- 我们的图像 -----  
----- RGB:=深蓝; -----  
**if**  $(x\text{column}(5) \text{ xor } y\text{row}(5)) = '0'$  **then** RGB:=GREEN; **end if;** **if** LCD\_DE= '0' **then** RGB:=BLACK; **end if;**



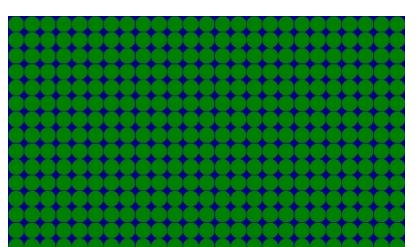
我们还可以通过在整个区域内复制图案来呈现更复杂的形状。先从单个图案开始：

----- 我们的图像 -----  
----- RGB:=深蓝; -----  
**if**  $(x-16)^{**2} + (y-16)^{**2} < 16^{**2}$  **then** RGB:=GREEN; **end if;** **if** LCD\_DE= '0' **then** RGB:=BLACK; **end if;**

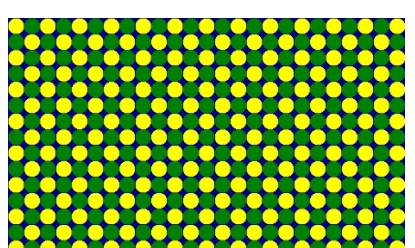


现在，我们将使用x和y除以32后的余数代替原始坐标。

----- 我们的图像 -----  
----- RGB:=深蓝; -----  
**if**  $(x \bmod 32 - 16)^{**2} + (y \bmod 32 - 16)^{**2} < 16^{**2}$  **then** RGB := GREEN; **end if;** **if** LCD\_DE = '0' **then** RGB := BLACK; **end if;**



----- 我们的图像 -----  
----- RGB:=深蓝; -----  
**如果**  $(x \bmod 32 - 16)^{**2} + (y \bmod 32 - 16)^{**2} < 16^{**2}$  **则如果**  $((x/32) \bmod 2 = 0) \text{ xor } ((y/32) \bmod 2 = 0)$  **则**  
RGB:=GREEN; **else** RGB:=YELLOW; **end if;**  
**结束条件判断;**  
**如果** LCD\_DE = '0' **则** RGB:=BLACK; **结束如果;**



实现LCDlogic0中最后一张图像的复杂度仅需九个逻辑元件和两个9位乘法器。整个绘图系统（含生成器和寄存器）仅使用77个逻辑元件构建。

上述两个9位乘法器。PNG格式存储带圆圈的图案需41KB空间，而JPEG格式则需多达141KB。

如此鲜明的圆形图案或许仅适用于展示逻辑运算能力:-) 为提升实用性，可通过减少色彩差异优化效果。例如从图9第10页按色相排序的色板中选取新颜色，最终背景将呈现更柔和的装饰图案：LSPimage : process( xcolumn, yrow, LCD\_DE )

```
变量 RGB :RGB_t :=BLACK; -- 当前像素颜色
变量 x, y : 整型 取值范围 0 至 XCOLUMN_MAX:=0; 变量 eqcircle : 整型
取值范围 0 至 2*(16**2):=0; begin-- 处理x := to_integer(xcolumn); y
:= to_integer(yrow);-- 将无符号输入转换为整数
x := to_integer(xcolumn); y := to_integer(yrow); -- 将无符号输入转换为整数
ourimage
RGB:=深蓝色;
eqcircle := (x mod 32 -16)**2+(y mod 32-16)**2;
if eqcircle<16**2 and eqcircle>=12*2 then
    如果 ((x/32) mod 2=0) 且 ((y/32) mod 2=0) 则 RGB:=X"0000FF"; 否则 RGB:=X"0000CD"; 结束如果; 结束如果;
if LCD_DE='0' then RGB:=BLACK; end if;
```

RGBcolor <= RGB; -- 赋值输出信号

结束过程;结束架构;

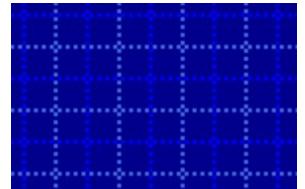
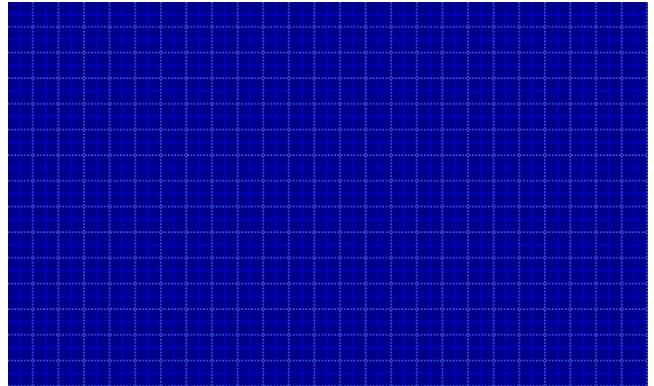
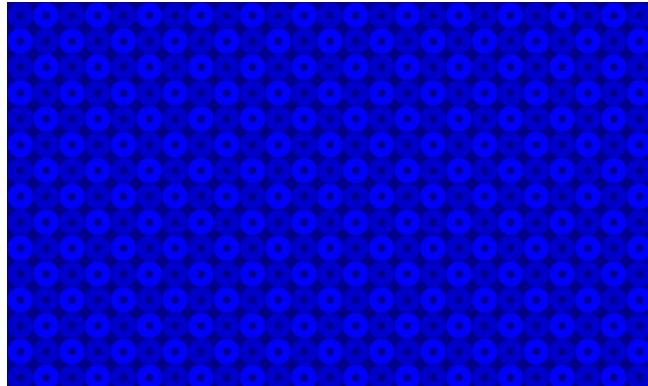


图15 - 技术主题：左上角代码生成，右下角输出

我们可以使用这种点虚线网格类比来装饰技术控制面板。垂直线和水平线的条件是分开的——毕竟它们的绘制是相互独立的！点虚线是通过在沿线轴运行的变量（例如 y 或 x）上插入条件来创建的。

```
RGB:=深蓝色;
如果 (y mod 16 ≥ 14) 且 (x mod 4) < 2 则
    如果 ((y/16) mod 2) = 0 则 RGB:=X"0000FF"; 否则 RGB:=X"4169E1"; 结束如果; 结束如果;
如果 (x mod 16≥14) 且 (y mod 4)<2 则
    如果 ((x/16) mod 2) = 0 则 RGB:=X"0000FF"; 否则 RGB:=X"4169E1"; 结束如果; 结束如果;
如果 LCD_DE='0' 则 RGB:=BLACK; 结束如果;
```

进度条是面板的常见功能，参见下图。

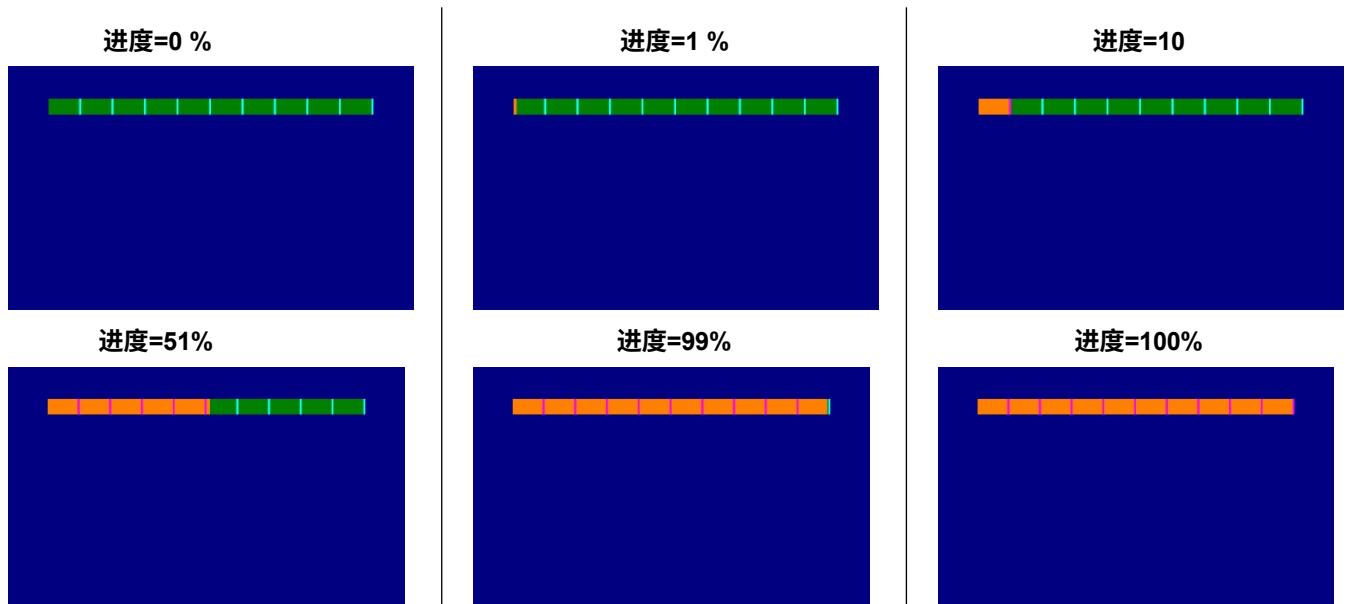


图16 - 线性指示器

实现此功能时，我们将使用除以  $2^{**}6=64$  后的余数。由于 64 不能整除宽度 LCD\_WIDTH=800（存在余数），因此结果呈现不对称性。右侧的 if 语句通过使用不同颜色进行区分，通过偏移  $80 = (800-10*64)/2$  来居中图案。

`if y<(x mod 2**6) then RGB:=RED; end if;`



`if y<((x-80) mod 2**6) then RGB:=GREEN; end if;`



若引入常量 P0 作为 x 轴原点，步长 ST=64，则架构如下：

LCDlogic0 的架构指示符为

信号 progress: 整型 范围 0 至 100 := 51; -- 该值由其他进程生成

begin -- 架构

LSPImage : process( xcolumn, yrow, progress) variable RGB :RGB\_t  
:=BLACK; -- 像素颜色变量 x : integer range 0 to 1023:=0;

变量 y : 整型 取值范围 0 至 524 := 0; -- YROW\_MAX-1

常量 P0: 整型 := 80; 常量 ST: 整型 := 2\*\*6; --P-原点步长

开始 x := to\_integer(xcolumn); y := to\_integer(yrow); RGB := NAVY;

----- 进度条 -----

如果 y>=ST 且 y<ST+ST/2 且 x>=P0 且 x<P0+10\*ST 则- 高度, 单位为 <64,96> 宽度, 单位为 <80,720>

RGB:=assignIf( ((x-P0) mod ST)<ST - 4, GREEN, AQUA); --间隙

if x<((progress\*205+16)/32 + P0) then RGB:=RGB xor YELLOW; end if; end if;

RGBcolor <= RGB;

结束进程;

iProgress : process(YEND\_N) -- 进度信号 动态模拟常量 MD:integer:=2\*\*5; 变量 cntr : integer range

0 to MD\*100:=0; beginif falling\_edge(YEND\_N) then  
if cntr< MD\*100 then cntr:=cntr+1; else cntr:=0; end if; end if;  
progress<=cntr/MD;

结束进程; 结束架构;

## 由计数器生成的重复形状

在之前的代码中，通过复杂公式将 **progress** 存储值转换为 x 轴长度：(**progress\*205+16**)/32，其中加 16 模拟了四舍五入。将 0 到 100% 的进度值拉伸到 0 到 640 像素区间的关系可重写为：

$$\text{round}(\text{progress} * 205.0 / 2^{**5}) \approx \text{progress} * 205.0 / 32 = \text{progress} * 6.40625 \approx \text{progress} * 6.4.$$

更优的转换方案是将 ST (步进) 值设为 60，此时进度值将乘以 6，但计算( $x \bmod 60$ )的电路需要大量硬件逻辑元件。我们将用计数器替代模运算。像素坐标 **xcolumn** 和 **yrow** 在 LCD\_DCLK 上升沿发生变化，因此让计数器运行至 LCD\_DCLK 下降沿——此时坐标已稳定，可安全检测而无亚稳态风险。我们将结果赋值给 **xbarmod** 信号，时机恰与 LCD\_DCLK 上升沿同步，即与像素坐标变化保持一致。

LCDlogic0 的架构条 60

信号 **progress**: 整型 范围 0 至 100 := 1; -- 来自另一个进程

常量 **P0**: 整型 := 100; 常量 **ST**: 整型 := 60; 信号 **xbarmod** : 整型 范围 0 至 **ST-1** := 0;

开始 -- 架构

iModulo : 进程(LCD\_DCLK)

变量 **cntr** : 整型 范围 0 到 **ST-1** := 0;

开始 如果是LCD\_DCLK的下降沿 则 **cntr:=assignlf(cntr>=ST-1 or xcolumn<P0, 0, cntr+1)**; 结束如果; 如果是LCD\_DCLK的上升沿 则 **xbarmod<=cntr**; 结束如果;

结束进程;

LSPImage : 过程( **xcolumn**, **yrow**, **progress**, **xbarmod**) 变量 RGB :RGB\_t := BLACK;

-- 像素颜色

变量 **x** : 整型 范围 0 到 1023 := 0; -- XCOLUMN\_MAX-1

变量 **y** : 整型 范围 0 到 524 := 0; -- YROW\_MAX-1

开始 **x := to\_integer(xcolumn)**; **y := to\_integer(yrow)**; **RGB := NAVY**;

----- 我们的图像 -----

if **y>=ST and y<ST+ST/2 and x>=P0 and x<P0+10\*ST then** -- 高度 + 宽度

**RGB:=assignlf(xbarmod<ST-4, GREEN, AQUA);--间隙 if(x<(6\*progress + P0)) then**

**RGB:=RGB xor YELLOW; end if;**

**end if;**

**RGBcolor <= RGB;**

结束进程;

iProgress : process(YEND\_N) -- 进度信号的动态仿真

常量 **MD**: 整型 := 2\*\*5;

变量 **cntr** : 整型 范围 0 至 **MD\*100** := 0;

开始 if **YEND\_N** 的下降沿 则 **cntr:=assignlf(cntr< MD\*100, cntr+1, 0)**; 结束 if; **progress<=cntr/MD**;

结束进程; 结束架构

;

若需观察 iModulo 进程的输出结果，需编写测试平台，并在其中嵌入该进程代码及必要定义：

```
library ieee; use ieee.std_logic_1164.all; use ieee.numeric_std.all; library work;
```

实体 testbench\_Modulo 是结束实体；架构 rtl 的

testbench\_Modulo 是

信号 xcolumn: unsigned(9 downto 0):=(others => '0'); -- LCDgen输出的仿真

信号 LCD\_DCLK : std\_logic:='0';

常量 P0: 整型 := 100; 常量 ST:整型:=60; 信号 xbarmod : 整型 范围 0 到

ST-1:=0;

begin -- 架构

iModulo : process(LCD\_DCLK) -- 被测代码的副本

变量 cntr : 整型 取值范围 0 到 ST-1:=0;

开始倘若 LCD\_DCLK 降沿则 cntr := 赋值条件(cntr >= ST-1 或 xcolumn < P0, 0, cntr+1); 结束倘若; 倘若 LCD\_DCLK 上升沿则  
xbarmod <= cntr; 结束倘若;

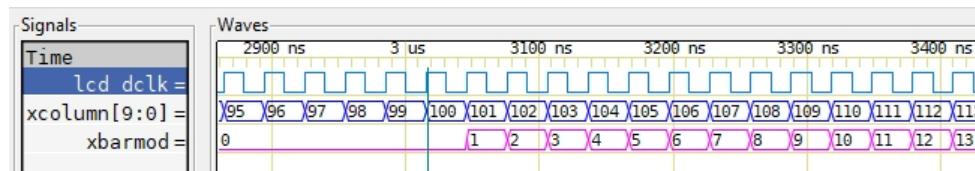
结束过程;

LCD\_DCLK <= not LCD\_DCLK after (1 sec)/(2\*33000000); -- 33 MHz信号的周期/2

xcolumn <= xcolumn + 1 when rising\_edge(LCD\_DCLK);

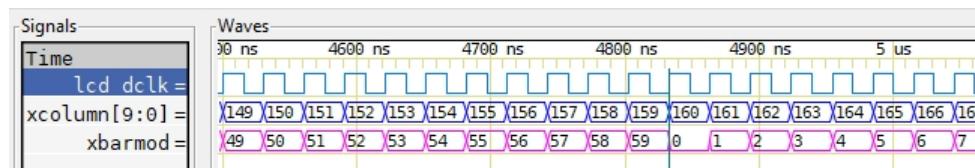
end architecture;

GHDL仿真在GTKView中显示以下曲线（为打印效果部分反转了颜色）。xbarmod的值在xcolumn=100之前不会计算，此前无需该

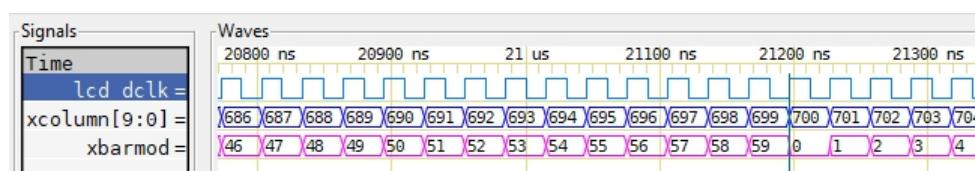


值：

该值增长至xcolumn=159后，将从0重新开始：



恰在xcolumn=699处（即LCD指针终点点），xbarmod=59



在GTKWave中，我们还可将xbarmod和xcolumn解释为模拟信号。选中一个信号后右键显示上下文菜单，依次选择：数据格式 → 模拟 → 步进。随后为每个信号添加"插入模拟高度扩展"。此时可在LCD行清晰观察波形，垂直标记与前图位置一致，位于xcolumn=699处。

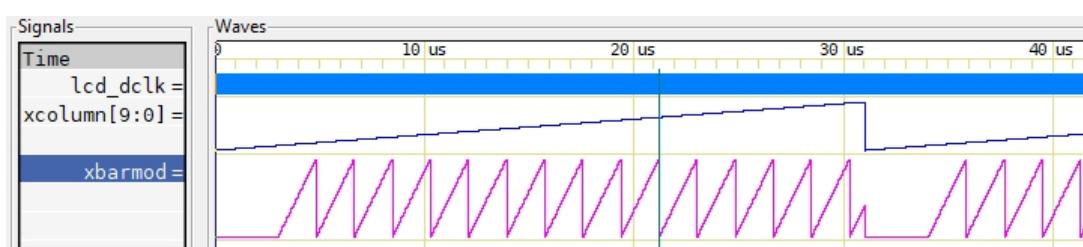


图17 - GTKView对模拟信号xcolumn和xbarmod值的解释

## 从FPGA ROM存储器插入图像

某些要求较高的部件值得转换为内存并在图像渲染时读取。在FPGA内部，我们有两种存储方案：

- **逻辑单元 (LE)** 能实现对数据的最快访问。然而，它们的使用远比单纯存储值更为灵活，从而承担了内存无法完成的更复杂运算。
- 存储器块主要用于FPGA中的海量数据处理。即使在Quartus开发环境中，逻辑单元有时也会被转换为存储器读取操作。这类单元能实现更高的数据信息密度，因为其制造所需的硅片更少。它们还支持多端口访问，允许在不同地址进行独立的数据操作。但需注意的是，每个存储器块必须被完全占用——即便仅使用其中一位。这完全取决于存储器内容的设计。

Cyclone IV电路包含可配置输出数据宽度的M9K存储器模块。单个M9K模块的可能变体（以字数×每字位数表示）如下：

$8192 \times 1, 4096 \times 2, 2048 \times 4, 1024 \times 8, 1024 \times 9, 512 \times 16, 512 \times 18, 256 \times 32, 256 \times 36$

例如， $1024 \times 8$  表示一种内存配置，其中使用 10 位地址 ( $2^{10} = 1024$ ) 选择 8 位字。因此，它拥有 1024 个 8 位宽的字，即 8192 位。M9K 存储器也可设置为 9 位输出（即可能带奇偶校验），使用全部 9216 位，详见 [Cyclone4\\_memoryM9Kblocks.pdf](#)。

从内存读取始终是同步的——选择矩阵要求如此。我们写入地址时对应一个时钟沿，数据则在延迟后出现在内存输出端。下图中的寄存器会延迟一个时钟周期，但数据在时钟周期内始终保持恒定值，这更适合实现。

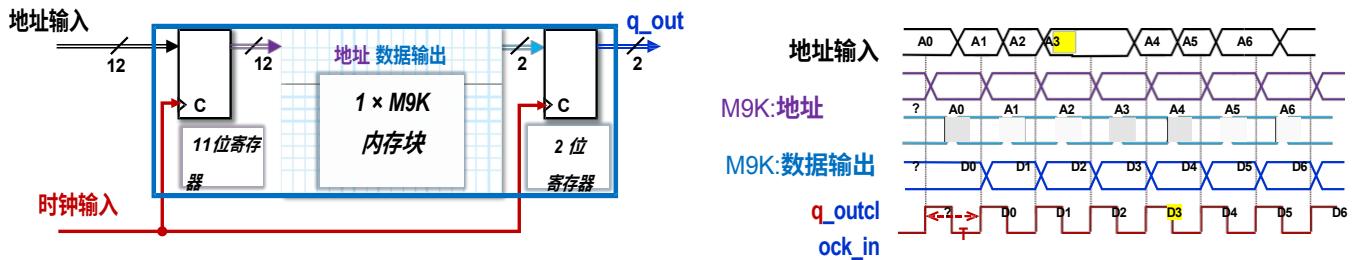


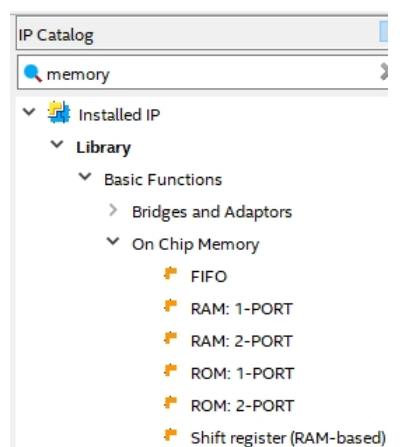
图18 - 旋风IV内存块M9k采用4096x2配置

更大的存储器由多个M9K块组装而成，值得监控其消耗规模，因为即使数据量微增也可能增加大量M9K块——这些块始终以完整形式使用。然而我们需要某种方式初始化存储器块。Quartus环境中提供两种方案：

- 1) 从制造商的IP目录中选择存储器类型（参见左侧图片）。该目录将启动Quartus中的MegaWizard插件管理器工具，我们在其中输入必要的存储器参数及初始化文件（类型为\*.MIF，即存储器初始化文件）。此流程较为繁琐，但可提供多种选项。然而这会增加GHDL仿真复杂度，因为必须插入Quartus内部库文件，操作并不简单。
- 2) 若单端口ROM存储器容量充足，可生成VHDL文件。Quartus将其转换为存储器。此方法亦便于GHDL仿真，我们将在本文中演示。前置步骤详见前述M9K存储器手册。

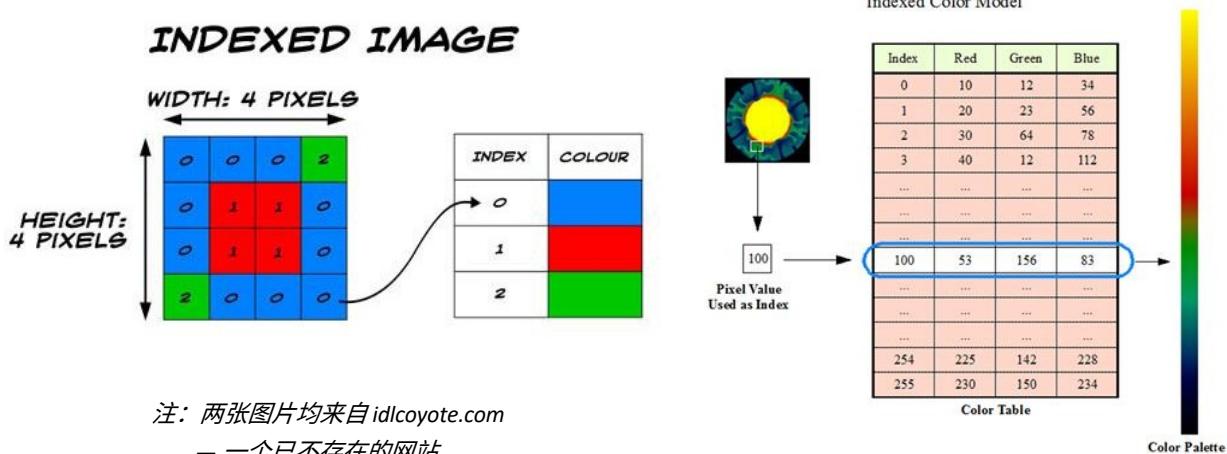
FPGA-LCD工具包中的Bitmap2VHDL工具可同时生成初始化\*.MIF文件

文件和Quartus可转换为ROM的\*.vhd文件：1-Port。



## 位图转换

若颜色数量较少，可通过为颜色分配索引值并仅存储索引来减少数据量。这些索引指向颜色表，便于后续修改。详情请参阅：[https://en.wikipedia.org/wiki/Indexed\\_color](https://en.wikipedia.org/wiki/Indexed_color)



注：两张图片均来自 [idlcoyote.com](#)  
—一个已不存在的网站。

图19 - 索引颜色

未进行光栅化处理的图像（即未启用“空间抗锯齿”功能，详见下文）因其色彩数量较多，最适合进行索引处理。若所选图像已进行光栅化，建议通过图形工具（如免费软件FastStone Image Viewer）减少色彩数量。

图形工具可通过光栅化添加过渡色调实现视觉边缘平滑。其对立技术“抖动”则通过色散生成半色调替代缺失色彩，从而平滑图像。硬件层面而言，实现 $3 \times 3$ 或 $5 \times 5$ 高斯模糊卷积矩阵相对简单，Veek-MT2液晶面板亦具备此功能。  
VeekMT2\_LCDregV2会关闭抖动功能，以便您能清晰看到其显示内容。通过将LCD\_DITH输出设置为'0'即可启用该功能。

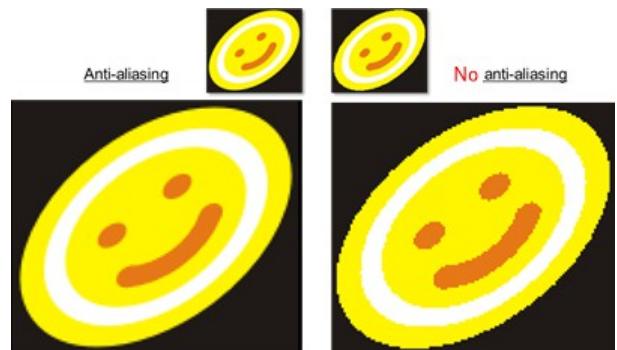


图20- 抗锯齿处理

使用图形工具裁剪图像的选定部分，并将其保存为位图。

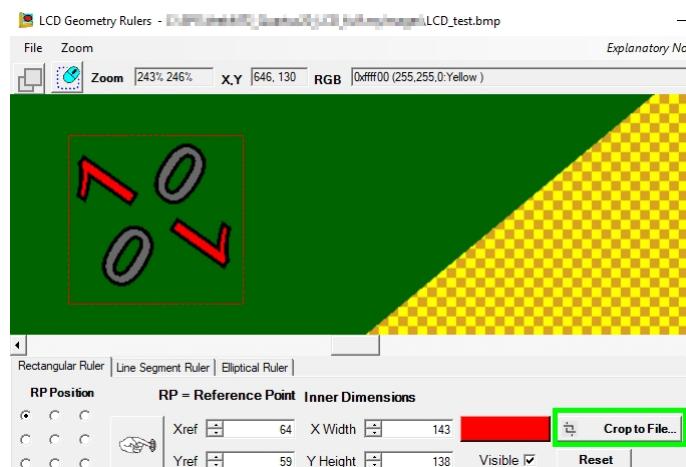


图21- 使用LSP几何尺工具裁剪文件

随后可运行 FPGA-LCD 实用工具中的 BMP 转换器：并加载该位图。

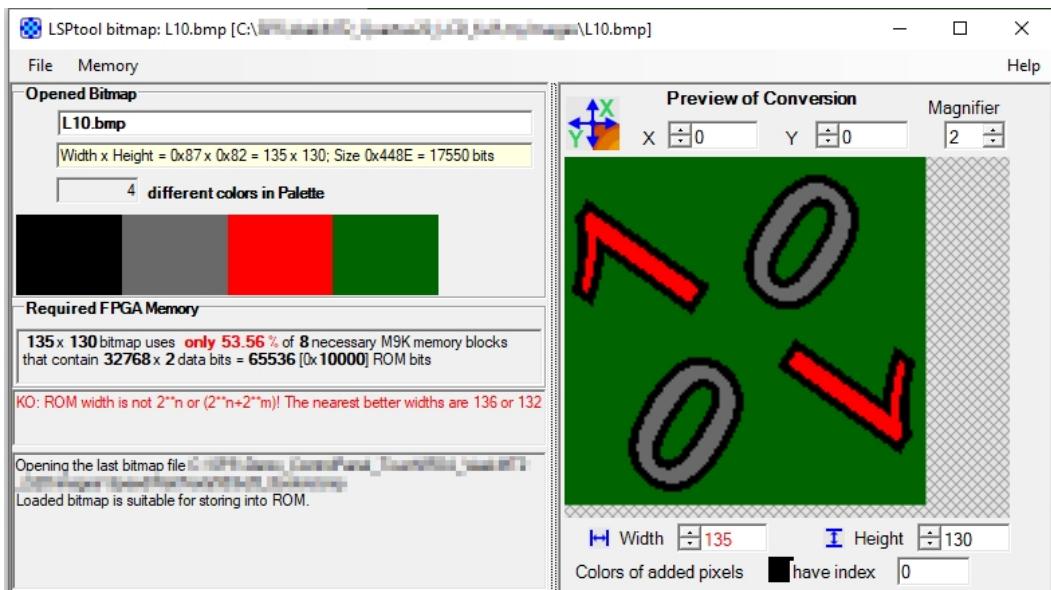
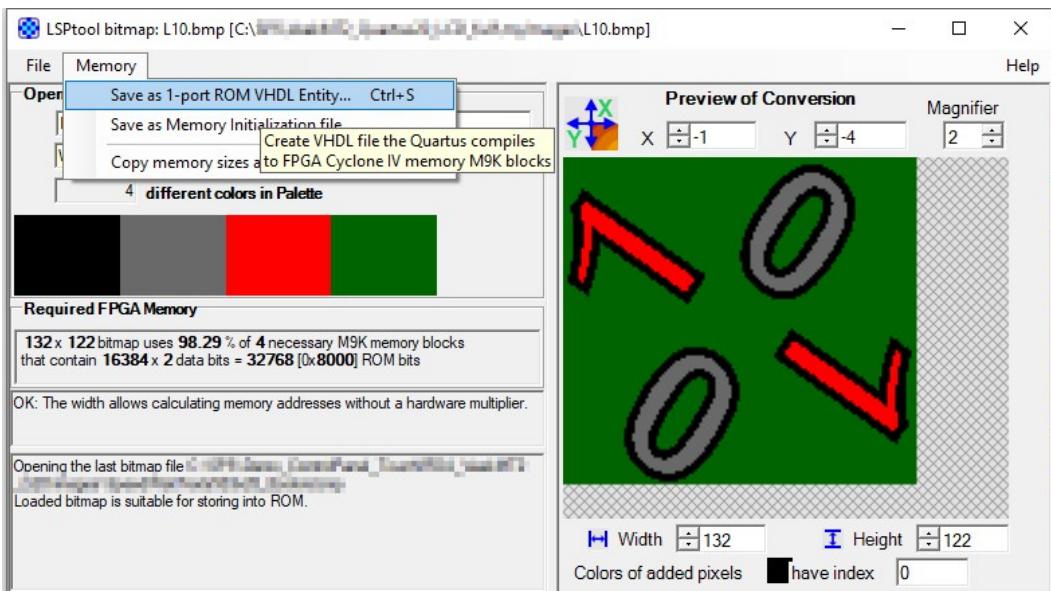


图22 - FPGA-LCD工具：位图转VHDL

该图像使用四种颜色，我们将其编码为两个数据位。然而，由此产生的内存将消耗8个M9K存储单元，且仅使用其中一半容量。我们裁剪图像时需注意：**裁剪后的宽度将乘以原始宽度**，因此选择2的幂次或两个2的幂次之和，这将使硬件实现更为简洁（参见第6.3.2节FPGA上的逻辑电路）。

使用上下控制键调整尺寸。最左侧单元保持空闲，因为我们将以延迟1个时钟周期的方式从存储器读取数据。



通过菜单"Memory->Save as 1-port ROM VHDL Entity"保存修改后的位图，例如在Quartus项目主目录下以L10rom.vhd命名！



请勿以任何方式修改生成的VHDL文件L10rom.vhd，以免破坏其在Quartus开发环境文档中规定的精确结构——该结构专为使用M9K模块实现而设计。不过我们可以读取其头部信息，其中包含内存大小和调色板的相关数据。

-- FPGA-LCD Utils 根据位图L10.bmp 生成的文件  
-- 调整为尺寸：宽度x高度 = 132x122=16104 [0x3EE8] 像素。  
-- 配置32768 [0x8000] 位存储器，通过14位地址总线读取2位数据输出。  
-- 按索引顺序排列的颜色调色板，作为std\_logic\_vector(23 downto 0) 项：  
-- X"000000", X"696969", X"FF0000", X"006400" -- 0 至3

```
library ieee, work; use ieee.std_logic_1164.all; use ieee.numeric_std.all;
```

实体 L10rom 定义为

```
port (地址: in std_logic_vector(13 downto 0):=(others=>'0');时钟: in std_logic:='1';  
      q: out std_logic_vector(1 downto 0):=(others=>'0'));  
end entity;
```

架构 rtl of L10rom 是

类型 arr\_t 为 array(0 至 2\*\*address'LENGTH-1) 的 std\_logic\_vector(q'RANGE) 数组；

常量 arr :arr\_t:=( 0 至 484 ≈ "11", 485 至 492 ≈ "00", 493 至 614 ≈ "11", 615 至 626 ≈ "00",

-- 后续行定义内存内容

```
15741至15753→"00", 15754至15875→"11", 15876至15882→"00",  
其余行→"11");
```

```
begin process(clock)
```

变量 ix:整型 范围 0 至 2\*\*地址'长度-1:=0;

开始

如果上升沿(时钟)则

```
  ix := to_integer(unsigned(address)); q≤ arr(ix);
```

end if;

```
end process;end
```

```
architecture;
```

我们明智地将转换后图像的首列保留为空，因此常量数组arr初始化时高亮显示的绿色值对应图像背景色的索引——该索引区域被设置为透明。

该过程代码引发疑问：为何引入新变量 ix 而未使用复合指令：

```
q≤ arr(to_integer(unsigned(address))); ?
```

推荐的编码风格（参见《Quartus 从 HDL 代码推断 ROM 功能》手册）要求地址输入采用 std\_logic\_vector 类型。但我们改用了更简洁的整型变量 ix，该变量在地址中定义了 to 范围。由于添加了包含精确值域信息的子类型，Quartus 也能正确编译 ix 变量。

注：Quartus 同样允许为 std\_logic\_vector 类型显式指定地址范围。但 VHDL 标准并未包含此特性。此类写法将产生依赖编译器的非移植性代码，即所谓的“编译器依赖代码”。

## 如何从内存读取图像？

下图展示了将 $4 \times 3$ 像素位图转换后的存储状态。该位图按行顺序以一维向量形式存储在内存中，这恰好与图像在液晶显示屏上的写入方式一致。这种存储多维数组的方法称为“行优先顺序”，C语言也采用此种存储方式。

内存仅存储两位颜色索引值（即0至3的数值）。我们向内存发送数据读取地址，该地址关系将存储图像映射至LCD的最终显示效果。右侧展示了多种可能变体中的两种：上图左上角位于LCD的 $yrow=1$ 、 $xcolumn=2$ 位置，内存数据按顺序读取。若设定 $x=xcolumn$ 且 $y=yrow$ ，则需先将 $x$ 和 $y$ 转换为相对于图像角点的相对坐标，进而计算出ROM存储器中的地址：

$$\text{内存地址} \equiv (\text{y-矩形1.Y}) * \text{img.Width} + (\text{x-矩形1.X})$$

我们将图像宽度相乘；此处我们巧妙地选择了两个2的幂的和，这在电路中更易实现。第二张图像被翻转，其相对y轴方向相反：

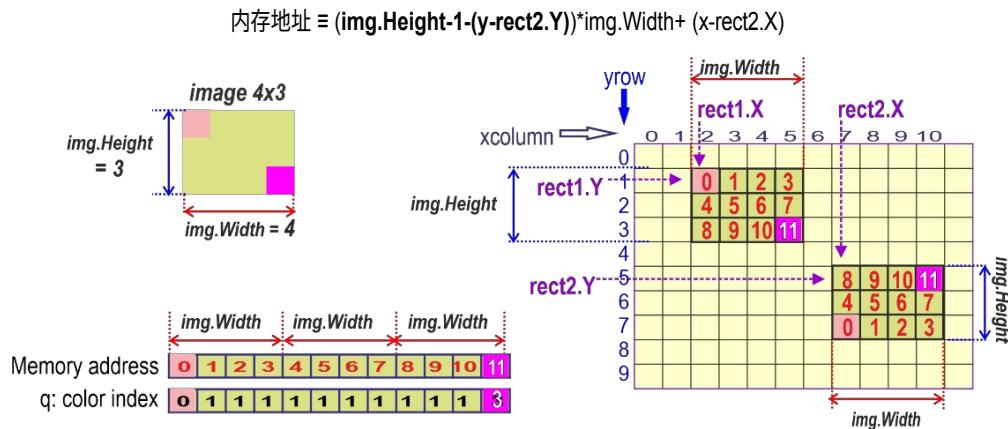
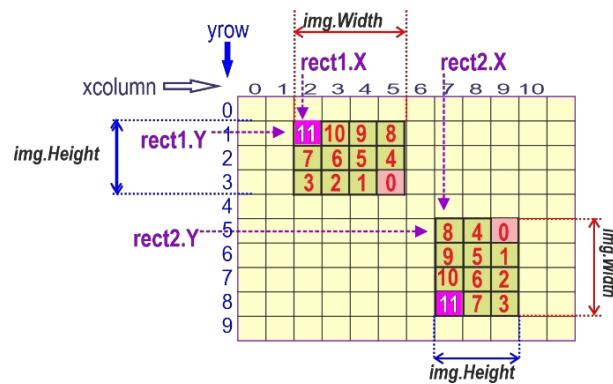


图23 - 图像1：正常 2：反转

定位方案多样，均仅需调整地址计算逻辑，此规则同样适用于90度旋转模块。180度旋转时双轴读取方向反转；90度旋转时仅单轴旋转，但轴向需互换。同时需修改矩形绘制区域的检测逻辑。

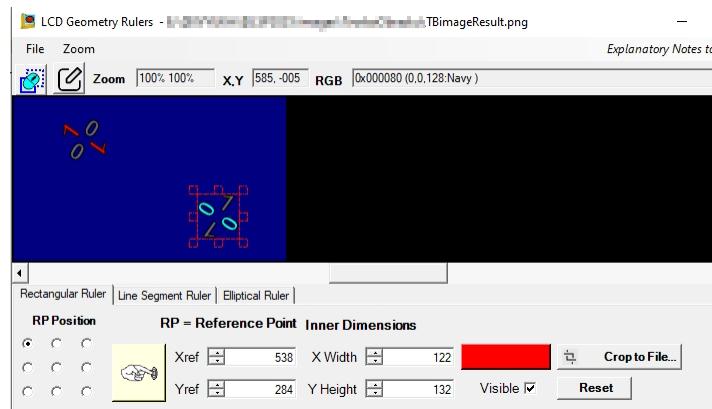
图24 - 旋转效果 1：180度 2：90度



90度旋转的实现将在下一章代码中呈现。

## 包含从内存插入图像的VHDL代码

若存在图形模板，则据此确定图像坐标。当然，我们只能估算其位置，并根据测试台结果进行修正——例如使用LCD几何尺规——直至对位置满意为止：



编写代码时需注意变量命名。我们采用VHDL记录类型，其与C语言结构体直接对应。实体结构未变，故从架构开始：

架构 img of LCDlogic0 定义如下：

类型 sizes\_t 是记录 宽度, 高度: 整数; 结束记录; 常量 L10img : sizes\_t :=(132,122);

---

常量类型 sizes\_t 包含宽度和高度，使这两个值组合在一起。

---

常量 L10r1 : rect\_t :=(140, 64, L10img.Width, L10img.Height);常量 L10r2 : rect\_t :=(538, 284, L10img.Height, L10img.Width);

我们为两个位置定义了rect\_t矩形的尺寸。在常量L10r2中，内存尺寸被交换，因为它指代的是将旋转90度的图像位置。

-- 类型 rect\_t 及其相关函数 inRect 存在于 LcdPackV2 版本 v2.1 及更高版本中

---

函数 inRect(r:rect\_t; x,y:integer) 返回 boolean 如下：

```
begin return x>=r.X and x<=r.X+r.W and y>=r.Y and y<=r.Y+r.H;
end function;
```

---

通过定义一个函数来检测当前坐标是否位于矩形内，从而简化主代码。

---

type palette4\_t is array (0 to 3) of RGB\_t;

常量 L10p1:palette4\_t:=(BLACK, X"696969", X"FF0000", X"006400");

常量 L10p2:palette4\_t:=(BLACK, AQUA, X"696969", X"006400");

---

我们基于转换位图的VHDL文件中的数据创建了第一个基本调色板。在第二个调色板中，我们重新为某些项目着色。

---

信号 L10addr: std\_logic\_vector(13 downto 0):=(others => '0');

信号 L10q, L10q0: std\_logic\_vector(1 downto 0):=(others=>'0');

---

我们将向存储器发送地址并从中读取数据。两个信号的大小必须根据存储器文件L10rom.vhd的输入输出进行创建。

---

function toSlv(n:integer; slvWidth:positive) return std\_logic\_vector is begin return std\_logic\_vector(to\_unsigned(n,slvWidth));
end function;

---

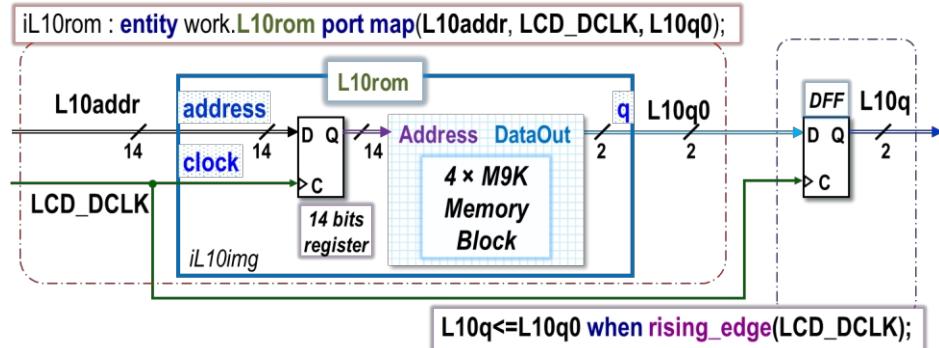
我们将使用整数计算地址，但内存将其作为类型为std\_logic\_vector的输入，因此我们定义了一个转换函数来简化主代码。

---

## begin -- 架构

在代码中，我们将创建内存实体的实例，并在其后放置输出值寄存器。同时插入一个DFF电路，其D输入为L10q0信号，输出L10q信号。

```
iL10rom : entity work.L10rom port map(L10addr, LCD_DCLK, L10q0); L10q<=L10q0 when  
rising_edge(LCD_DCLK);
```



由上述两条指令构成的电路如图所示：

```
LSPImage : process( xcolumn, yrow, L10q) variable RGB :RGB_t  
:=BLACK; -- 像素颜色  
变量 x : 整型 范围 0 至 1023 := 0; -- 至 XCOLUMN_MAX-1  
变量 y : 整型 取值范围 0 至 524 := 0; -- 取值上限为 YROW_MAX-1
```

我们将L10q（即从内存读取的值）添加到该过程的敏感性列表中，RGBcolor过程的输出（即像素颜色）也依赖于此列表。随后，我们插入了已知的x和y变量定义。

```
变量 L10idRect : 整型 取值范围 0 至 2 := 0; -- 标记像素点位于矩形内部的标志位, 0 表示不在  
变量 L10ixColor : 整型 取值范围 L10p1'RANGE:=0; -- 从内存读取的颜色索引
```

第一个变量L10idRect列表将作为标识符，表示像素的x,y坐标位于图像的矩形区域内，其中0表示超出图像范围。第二个变量是转换为整数的颜色索引。

```
begin -- 处理  
x := to_integer(xcolumn); y := to_integer(yrow);  
L10idRect:=0; -- 不在矩形内  
if InRect(L10r1, x, y) then L10idRect:=1; elsif InRect(L10r2,x,y) then L10idRect:=2; end if;
```

我们通过连续测试矩形内部的位置，为矩形分配标识符L10idRect。

```
L10ixColor := to_integer(unsigned(L10q)); -- 调色板索引
```

我们将从内存读取的值转换为整数，用于索引调色板。

----- 我们的图像 -----

```
RGB := NAVY;  
if L10idRect> 0 and L10ixColor/=3 then -- 当前像素是否位于矩形内且颜色索引为半透明色?  
if L10idRect= 1 then RGB:=L10p1(L10ixColor); else RGB:=L10p2(L10ixColor); end if; end if;
```

若像素点[x, y]位于任意图像矩形内（L10idRect>0），且同时从图像内存读取的颜色值不同于3（即不同于需透明化的背景颜色索引L10ixColor/=3），则覆盖对应调色板的RGB值。

```

case L10idRect is
    当 1 > L10addr <= toSlv( y-L10r1.Y)*L10img.Width+(x-L10r1.X), L10addr'LENGTH );
    当 2 > L10addr <= toSlv( (L10img.Height-1-(x-L10r2.X))*L10img.Width+(y-L10r2.Y), L10addr'LENGTH );
    当 others > L10addr <= toSlv( (L10img.Height-1-(x-L10r2.X))*L10img.Width+(y-L10r2.Y), L
end case;

```

我们通过直接将二维数组的索引转换为向量来计算第一个矩形中的内存地址。第二个矩形旋转了90度，因此其相对的x轴和y轴被互换：x轴沿着L10img.Height-1到0的行号逆向读取，而y轴则沿着图像中行号的正向延伸。

```

RGBcolor <= RGB;
end process;end
architecture;

```

**GHD伙伴关于-a参数的建议：**在运行LCDlogic\*之前，必须将内存文件添加到runLCD.bat批处理文件的文件列表中，否则无法编译。

```
set FILES=..//LCDpackV2.vhd ..//L10rom.vhd ..//LCDlogic0.vhd
```

但内存文件无需永久保留在列表中。我的参数伙伴-e和-r仅需编译生成的\*.o（目标文件），它们能成功据此生成exe文件。只需完整编译一次，随后修改runLCD.bat或新建\*.bat文件，将FILES行修改为仅保留..//LCDlogic0.vhd即可。

```
rem 设置 FILES=..//LCDpackV2.vhd ..//L10rom.vhd ..//LCDlogic0.vhd
set FILES=..//LCDlogic0.vhd
```

您将更快看到结果。前一个示例为我节省了4秒宝贵时间！

仅在修改L10rom.vhd和LCDpackV2.vhd后才需进行完整文件编译。

**Quartus Lite补充说明：**类似技巧对我无效——我可不是自由发挥的GHD伙伴！

我坚持精确性！内存文件（此处为..//L10rom.vhd）始终会出现在我的“文件”选项卡列表中。

如果该文件不存在，请立即添加。可右键点击文件打开其关联的上下文菜单，或使用我的主菜单：

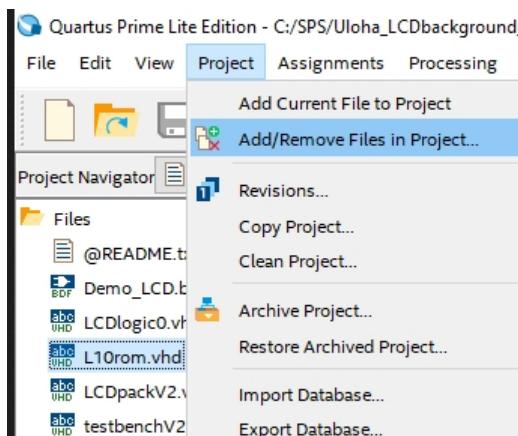


图25 - 向文件列表添加内存

在GHD中翻译和仿真VHDL代码约需7秒，我们仅执行一步操作——通过Visual Studio Code终端运行批处理文件。Quartus还提供安装Intel Questa仿真器的选项，但获取其免费许可证的复杂程度不亚于使用过程。Questa虽具备些许优势，但

有时能检测到更多时序错误，但其免费版仅能多捕获少量错误。不过使用GHDL进行调试的速度更快、操作更简便。◎

注：1/ 若将Quartus中调试过的VHDL代码编译后，即可加载至开发板。

2/ 需注意：即便VHDL代码在仿真中表现完美，FPGA板卡仍可能无法正常工作，因为仿真终究只是仿真。硬件才是最终判定是否满足所有时序要求的标准。

3/ 若Quartus编译器提示定时定义不完整警告：

严重警告(332168)：以下时钟传输未分配时钟不确定性。为获得更精确的结果，请应用时钟不确定性分配或使用 derive\_clock\_uncertainty 命令... 或

严重警告(332049)：忽略了 VeeekMT2\_LCD.sdc(50) 处的 create\_generated\_clock 命令：参数 <targets> 为空集合

此外，BDF架构中顶级实体存在错误实例，详见第4页图2:-)

➤ VeeekMT2\_LCDgenV2必须使用实例名iLCDgenerator。

仅此实例名在文件VeeekMT2\_LCD.sdc中为TimeQuest Analyzer提供了定义。若更改实例名，必须重新生成sdc文件... 不过修改实例

---

本身更为简便快捷。

## 教材结束

---

学生马克索·格劳乔的抗议：结局居然在最悬念迭起的时刻戛然而止，简直像在看惊悚恐怖剧集？！你们开玩笑吧。这下可好，我们得在极其艰难、令人沮丧的漫长时光里煎熬了！

总会有现成的解决方案，能快速“复制”个人背景吧？

答：才怪！GHDL模拟运行迅速，你可以自由实验，用模板创作趣味作品。

马克索·格劳乔的惊恐：我遍寻网站都找不到模板里那些独立背景的VHDL代码，根本没法快速复制！

答：这些代码根本不存在，也永远不会存在——这能帮你节省时间。若不计注释，整个背景（含图片插入）的代码量不足2000字符（含空格）。借助编辑器的自动补全功能，你几分钟就能完成编写，还能更深入理解其工作原理。

请务必创建专属代码！毕竟您必然拥有尺寸各异、地址可能不同且数据范围各异的图片。直接复制原始代码会连带复制原始数字，届时排查错误将耗费大量时间！

---

以上就是全部内容...

~ 完 ~

---

