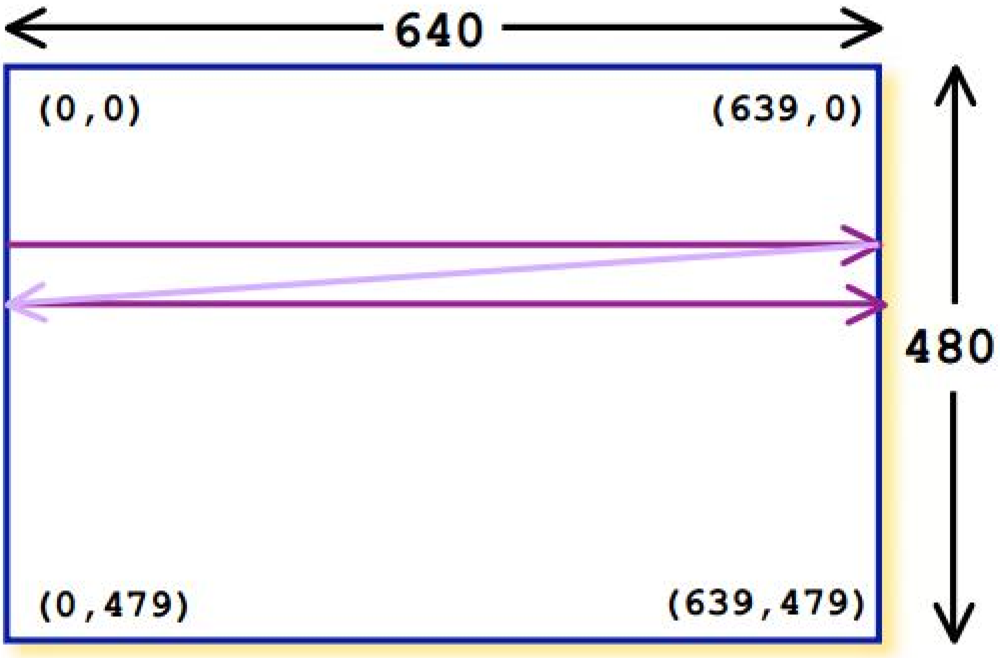
让 Arduino 输出 VGA 信号

首先需要学习 VGA 显示的基本知识。比如下面这个文档就介绍了 VGA 信号的基本常识
http://lslwww.epfl.ch/pages/teaching/cours_lsl/ca_es/VGA.pdf

a. VGA 信号主要有下面几个构成
   - Horizontal sync 作用是通知换行
   - Vertical sync 作用是通知换页
   - Red: 0-0.7v 红色信号的幅值
   - Green: 0-0.7v 绿色信号的幅值
   - Blue: 0-0.7v 蓝色信号的幅值

   从上面的 RGB 可以看出来，VGA 是模拟信号，并非完全的数字信号

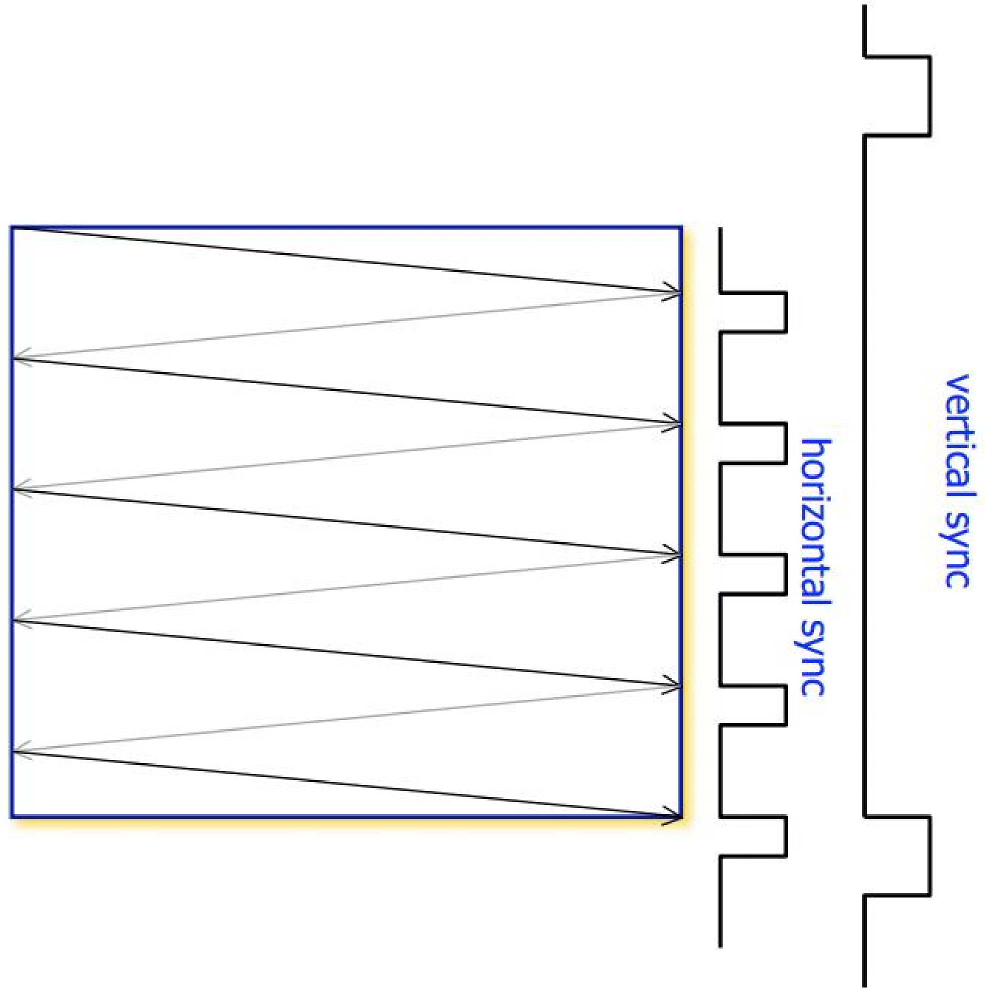
b. VGA 信号的扫描顺序是从左到右，从上到下。



c. 通常扫描速度是 60Hz。例如在 640x480 60 Hz 的条件下，一共有 640x480x60 = 18432000 个点，平均每个点 5.423 x 10 -8 次 秒 也就是 54ns。除了这个之外，每次换行返回行首，或者换页也都是需要时间的（front and back porches，资料上说，这个参数是最开始 CRT 需要的。电子枪射出电子，使用磁场来控制飞行的目标。发射换行或者换页之后，需要一段时间来调整，所以要求有一些延时）。综合起来对于带宽的要求在 25Mhz 左右

d. 当换行的时候（horizontal）,RGB 要给出黑色（全 0）

e. 下面是一个换行换页的 demo



对于上图的简单解释就是：通过 RGB 三个信号逐个给出每个点的颜色信息。然后扫描完一行之后要给一个 H Sync 信号通知这是一行的结束(有些资料说是负脉冲，但是实际做的时候好像正负脉冲都可以)。这样从上倒下从左到右扫描完一页之后，再发送一个 V Sync 通知页结束，开始一个新的页面，这样周而复始。

上述是定性的介绍，定量介绍可以参考下面的页面
http://wenku.baidu.com/link?url=q6VExjB4jAlxPaYxQkGBAhrpAxLXQSm2rYQBHkDAgz3tYdVP4bdD3x_IUFolEgOF2mY6cF4zm1mHb6xzEs_DQ0jwTixbBhYn4-o52G5jVy3

计算方法
http://wenku.baidu.com/link?url=q6VExjB4jAlxPaYxQkGBAhrpAxLXQSm2rYQBHkDAgz3tYdVP4bdD3x_IUFolEgOF2JZg-lVZPYHIWOO-Rf5uDmU1ASDDKdCDbBA97EP42I_

更详细一点的介绍
http://www.cnblogs.com/spartan/archive/2011/08/16/2140546.html

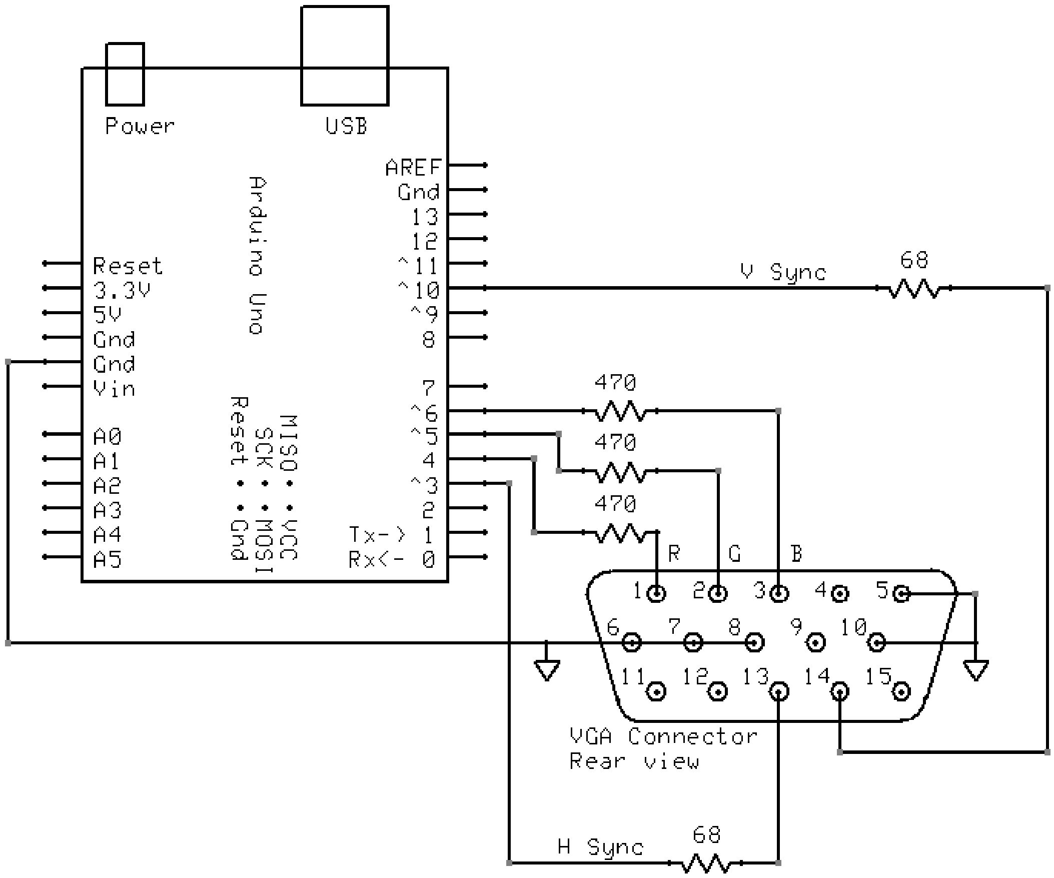
提供一张 Xilinx 的 table 有详细的参数
http://blog.163.com/qingyu_1984/blog/static/144414503201165106475 63/

在下面这个网站也提供了 Time Table
http://martin.hinner.info/vga/

下面这个页面给出了使用 Arduino 来产生 VGA 信号并且 Show 在显示器上的例子，
http://www.gammon.com.au/forum/?id=11608

其中提到：因为显示器中 R G B 上已经有 75 欧的电阻，所以外端加入 470 欧电阻就能保证输入在 0-0.7v 了。

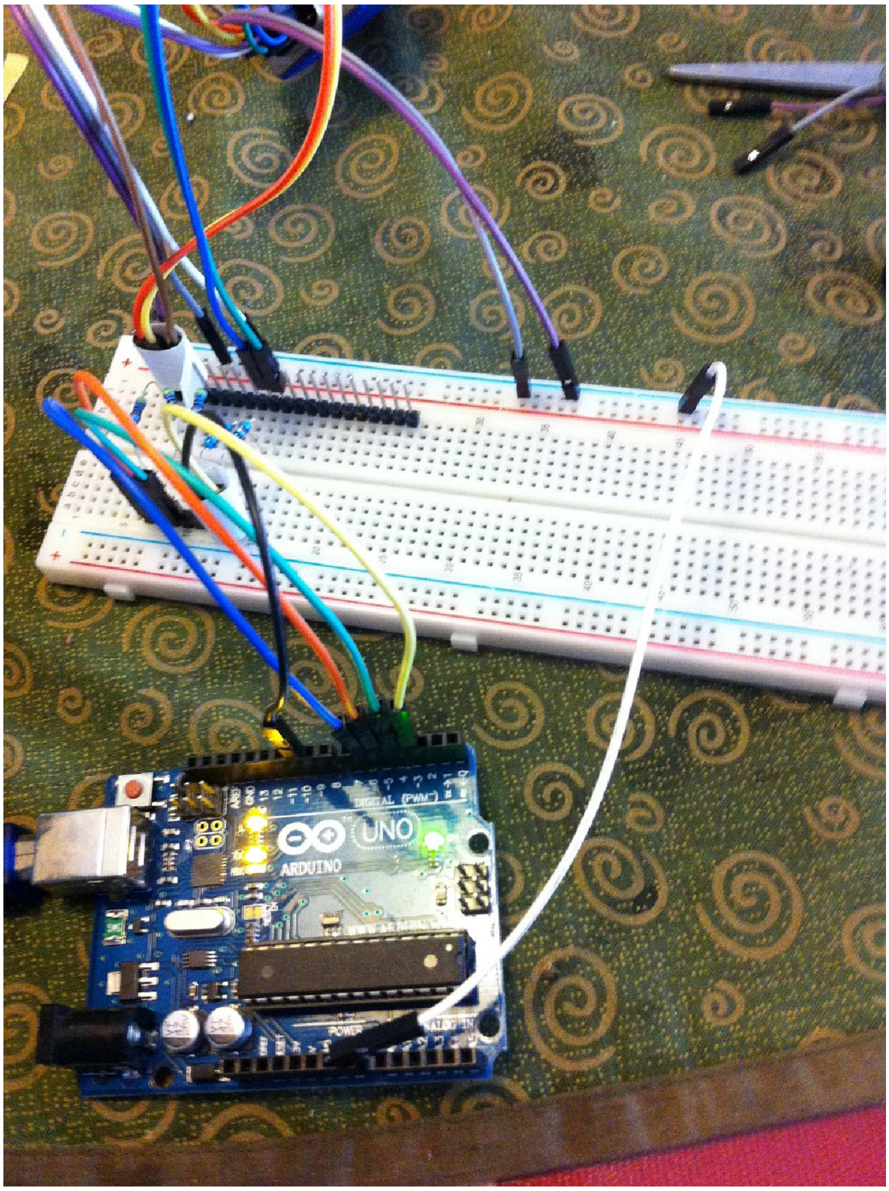
电路图：



基本的连接

Arduino 端	VGA 显示器端（上图是公头）
D3	Pin13
D4	Pin1
D5	Pin2
D6	Pin3
GND	Pin6/Pin7/Pin8/Pin5/Pin10
D10	Pin14

整体来说还是挺简单的



下面的代码来自 http://www.gammon.com.au/forum/?id=11608

```
/*
 VGA colour video generation

 Author:    Nick Gammon
 Date:      22nd April 2012
 Version:   1.0

 Version 1.0: initial release
 Version 1.1: Amended to output 64 colours
```

Connections:

D0 : Dull Red pixel output (1K resistor in series) --> Pin 1 on DB15 socket
D1 : Dull Green pixel output (1K resistor in series) --> Pin 2 on DB15 socket
D2 : Dull Blue pixel output (1K resistor in series) --> Pin 3 on DB15 socket
D3 : Horizontal Sync (68 ohms in series) --> Pin 13 on DB15 socket
D4 : Red pixel output (470 ohms in series) --> Pin 1 on DB15 socket (also)
D5 : Green pixel output (470 ohms in series) --> Pin 2 on DB15 socket (also)
D6 : Blue pixel output (470 ohms in series) --> Pin 3 on DB15 socket (also)
D10 : Vertical Sync (68 ohms in series) --> Pin 14 on DB15 socket

Gnd : --> Pins 5, 6, 7, 8, 10 on DB15 socket

Note: As written, this sketch has 34 bytes of free SRAM memory.

```
*/

#include "TimerHelpers.h"
#include <avr/pgmspace.h>
#include <avr/sleep.h>

const byte hSyncPin = 3;          // <------- HSYNC

const byte redPin = 4;            // <------- Red pixel data
const byte greenPin = 5;          // <------- Green pixel data
const byte bluePin = 6;           // <------- Blue pixel data

const byte dullRedPin = 0;          // <------- Red pixel data 50% brightness
const byte dullGreenPin = 1;        // <------- Green pixel data 50% brightness
const byte dullBluePin = 2;         // <------- Blue pixel data 50% brightness

const byte vSyncPin = 10;         // <------- VSYNC

const int horizontalBytes = 60;    // 480 pixels wide
const int verticalPixels = 480;    // 480 pixels high

// Timer 1 - Vertical sync

// output    OC1B   pin 16  (D10) <------- VSYNC

//   Period: 16.64 mS (60 Hz)
//      1/60 * 1e6 = 16666.66 uS
//   Pulse for 64 uS  (2 x HSync width of 32 uS)
//    Sync pulse: 2 lines
//    Back porch: 33 lines
//    Active video: 480 lines
//    Front porch: 10 lines
//       Total: 525 lines

// Timer 2 - Horizontal sync

// output    OC2B   pin 5  (D3)    <------- HSYNC

//   Period: 32 uS (31.25 kHz)
//      (1/60) / 525 * 1e6 = 31.74 uS
//   Pulse for 4 uS (96 times 39.68 nS)
//    Sync pulse: 96 pixels
//    Back porch: 48 pixels
```

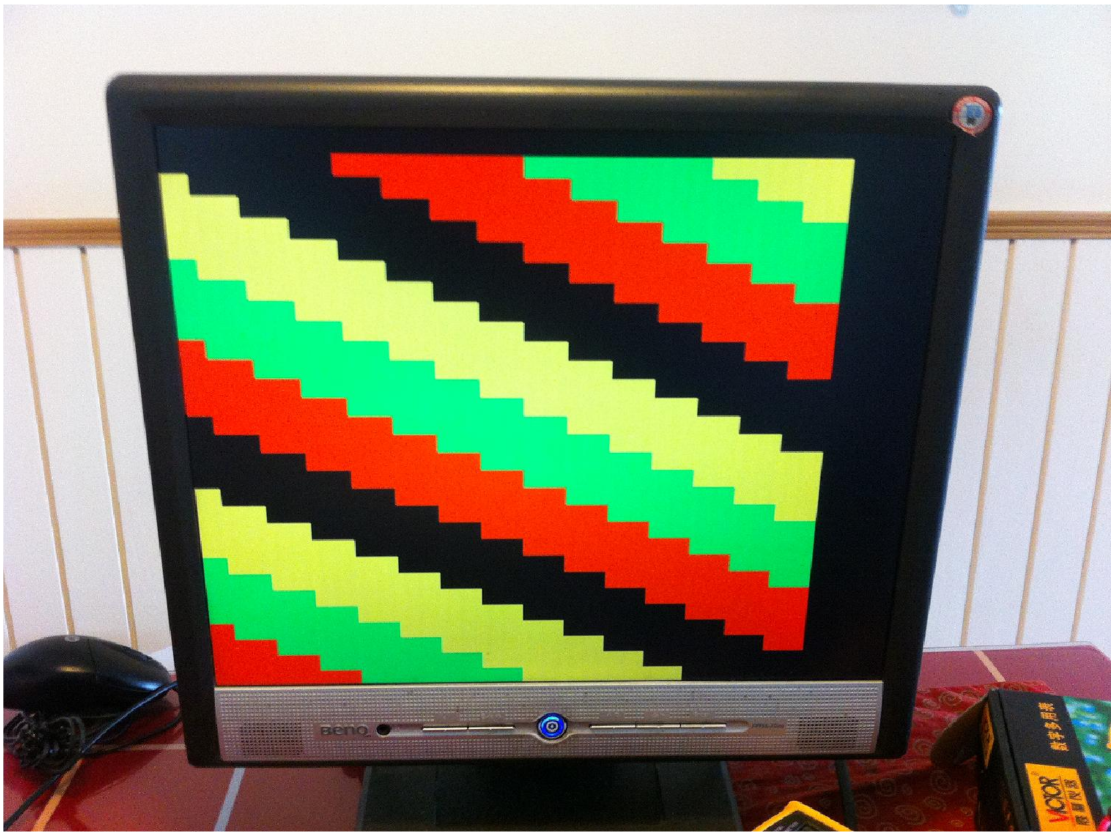
```
//        Active video: 640 pixels
//        Front porch: 16 pixels
//             Total: 800 pixels

// Pixel time =    ((1/60) / 525 * 1e9) / 800 = 39.68    nS
//    frequency =    1 / (((1/60) / 525 * 1e6) / 800) = 25.2 MHz

// However in practice, it we can only pump out pixels at 375 nS each because it
//    takes 6 clock cycles to read one in from RAM and send it out the port.

const int verticalLines = verticalPixels / 16;
const int horizontalPixels = horizontalBytes * 8;

const byte verticalBackPorchLines = 35;    // includes sync pulse?
const int verticalFrontPorchLines = 525 - verticalBackPorchLines;

volatile int vLine;
volatile int messageLine;
volatile byte backPorchLinesToGo;

#define nop asm volatile ("nop\n\t")

// bitmap - gets sent to PORTD
// For D4/D5/D6 bits need to be shifted left 4 bits
//    ie. 00BGR0000

char message [verticalLines]    [horizontalBytes];

// ISR: Vsync pulse
ISR (TIMER1_OVF_vect)
   {
   vLine = 0;
   messageLine = 0;
   backPorchLinesToGo = verticalBackPorchLines;
   } // end of TIMER1_OVF_vect

// ISR: Hsync pulse ... this interrupt merely wakes us up
ISR (TIMER2_OVF_vect)
   {
   } // end of TIMER2_OVF_vect


void setup()
   {
```

```
byte count = 0;
// initial bitmap ... change to suit
for (int y = 0; y < verticalLines; y++)
   for (int x = 0; x < horizontalBytes; x++)
         {
         message [y] [x] = ((count << 1) & 0x70) | (count & 0x07);
         if (++count >= 64)
            count = 0;
         }


// disable Timer 0
TIMSK0 = 0;    // no interrupts on Timer 0
OCR0A = 0;      // and turn it off
OCR0B = 0;

// Timer 1 - vertical sync pulses
pinMode (vSyncPin, OUTPUT);
Timer1::setMode (15, Timer1::PRESCALE_1024,
Timer1::CLEAR_B_ON_COMPARE);
OCR1A = 259;    // 16666 / 64 uS = 260 (less one)
OCR1B = 0;        // 64 / 64 uS = 1 (less one)
TIFR1 = bit (TOV1);      // clear overflow flag
TIMSK1 = bit (TOIE1);    // interrupt on overflow on timer 1

// Timer 2 - horizontal sync pulses
pinMode (hSyncPin, OUTPUT);
Timer2::setMode (7, Timer2::PRESCALE_8, Timer2::CLEAR_B_ON_COMPARE);
OCR2A = 63;      // 32 / 0.5 uS = 64 (less one)
OCR2B = 7;        // 4 / 0.5 uS = 8 (less one)
TIFR2 = bit (TOV2);      // clear overflow flag
TIMSK2 = bit (TOIE2);    // interrupt on overflow on timer 2

// prepare to sleep between horizontal sync pulses
set_sleep_mode (SLEEP_MODE_IDLE);

// pins for outputting the colour information
pinMode (redPin, OUTPUT);
pinMode (greenPin, OUTPUT);
pinMode (bluePin, OUTPUT);

pinMode (dullRedPin, OUTPUT);
pinMode (dullGreenPin, OUTPUT);
pinMode (dullBluePin, OUTPUT);
```

```
}    // end of setup

// draw a single scan line
void doOneScanLine ()
  {

  // after vsync we do the back porch
  if (backPorchLinesToGo)
    {
    backPorchLinesToGo--;
    return;
    }    // end still doing back porch

  // if all lines done, do the front porch
  if (vLine >= verticalPixels)
    return;

  // pre-load pointer for speed
  register char * messagePtr =    & (message [messageLine] [0] );

  delayMicroseconds (1);

  // how many pixels to send
  register byte i = horizontalBytes;

  // blit pixel data to screen
  while (i--)
    PORTD = * messagePtr++;

  // stretch final pixel
  nop; nop; nop;

  PORTD = 0;    // back to black
  // finished this line
  vLine++;

  // every 16 pixels it is time to move to a new line in our text
  if ((vLine & 0xF) == 0)
    messageLine++;

  }    // end of doOneScanLine

void loop()
```

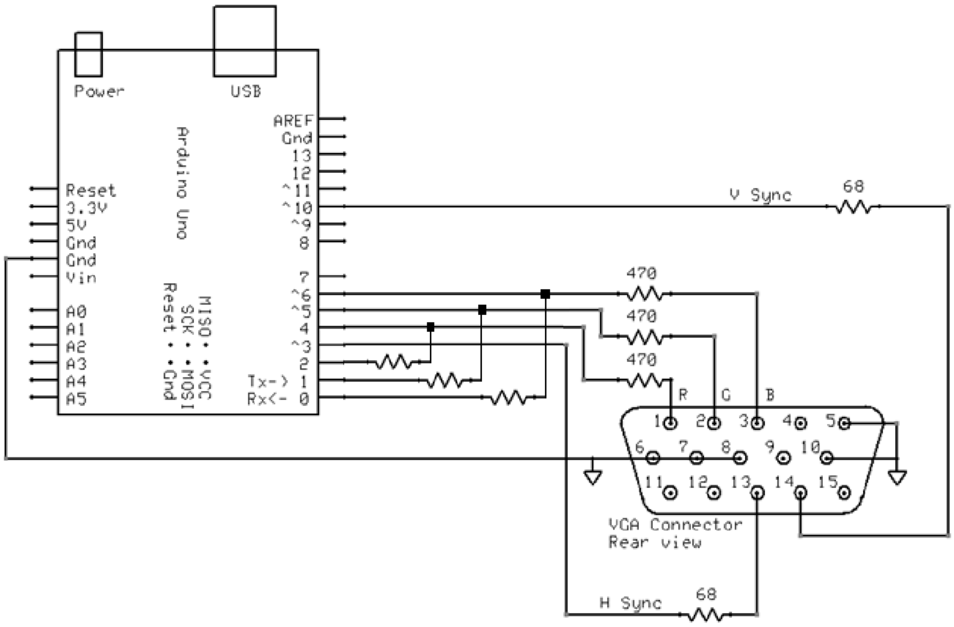
```
{
// loop to avoid overhead of function all
while (true)
  {
  // sleep to ensure we start up in a predictable way
  sleep_mode ();
  doOneScanLine ();
  }   // end of while
}   // end of loop
```

按照上述连接硬件和软件之后，得到下面的结果：



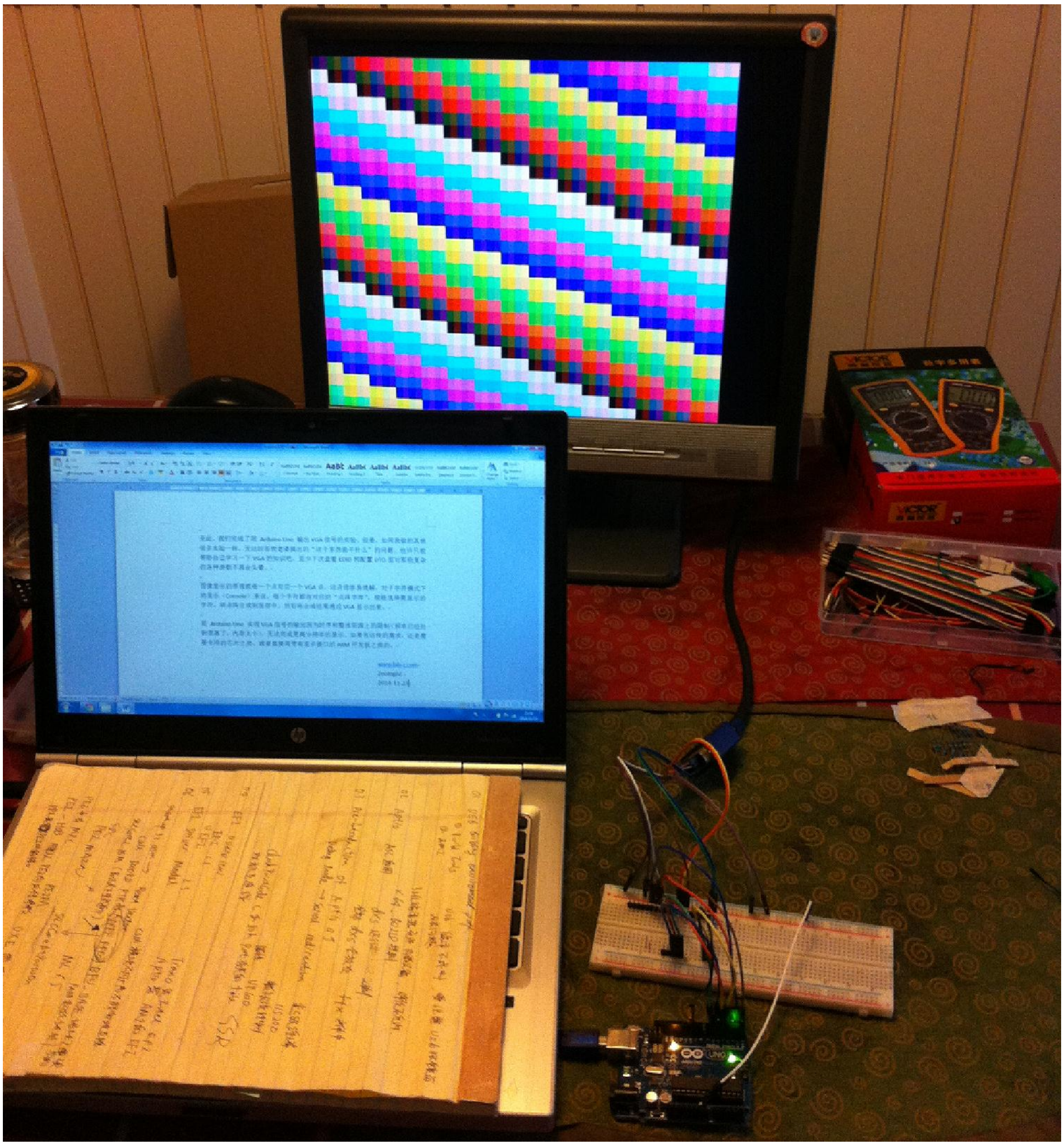
看起来和文章中提到的不同。尝试调试，可以怀疑的只有 RGB 三个信号（Hsync 和 Vsync 错误的时候会无显示）。一个是 RGB 连接的顺序，一个是 RGB 的通断。从上面的图片可以看到，缺少的应该是 Blue 信号，经过检查，确实 Blue 信号连接不良。最后得到的结果是：

不过这个结果和文章中的图片相比，看起来颜色数不够多，再仔细阅读文章，原来这个程序是需要额外的电阻配合才能显示更多颜色。需要加入 3 个 1K 电阻。修改后的电路图如下（多出来的是 1K 电阻）



修改之后的结果如下，可以看出来颜色更多一些（本质上是数字信号分压组合而来的，所以颜色数量有限）

最后，放一张完整的工作照（显示器屏幕没有完全显示，因为 Arduino 所能提供的最高频率有限）



至此，我们完成了用 Arduino Uno 输出 VGA 信号的实验。但是，如同我做的其他很多实验一样，无法回答我老婆提出的“这个东西能干什么”的问题。也许只能帮助自己学习一下 VGA 的知识吧，至少下次查看 EDID 和配置 DTD 面对那些复杂的各种参数不再会头晕。

图像显示的原理就是一个点对应一个 VGA 点，这点很容易理解；对于字符模式下的显示（Console）来说，每个字符都有对应的“点阵字库”，根据要显示的字符将对应的字模合成到显存中，最终将合成结果通过 VGA 显示出来。

用 Arduino Uno 实现 VGA 信号的输出因为时序和整体资源上的限制（频率已经拉到很高了，内存太小），无法完成更高分辨率的显示。如果有这样的需求，还是需

要专用的芯片之类。或者直接用带有显示接口的 ARM 开发板之类的。

www.lab-z.com
Zoologist
2014-11-23