VGA Colour Palette Shifter

Stefan Martynkiw 2016-03-14

---

1 Introduction

With the SRAM containing 8-bit-wide words, it is occaisionally desired to have graphical output with an increased colour depth. This block allows you to specify a transition between an 8-bit colour value and an RGB565 output value. This block contains a 2**8 = 256 byte deep palette ram, which performs a colour translation when required.

In QSYS, this block slots in between the Video FB streamer (which pulls out data from the SRAM) and the Video RGB resampler.

2 Clocking

The Colour Palette Shifter runs on the same video clock as the VGA pixel buffer discussed in our team's other application note.

3 Interfaces

The interfaces used for this block are shown in the block diagram below.

4 Avalon-ST

Avalon-ST is the interface used for streaming data between components. This is the name of the interface that connects the video output components in the provided example Qsys system.

This bus consists of a data signal, as well as several control signals. The interface is not bi-directional, and must flow from source to sink.

For video data as we are using it, the Avalon-ST bus must be clocked at the same rate as the VGA pixel clock: 25.2MHz. Each clock cycle, a pixel is transferred over the `data` signal. The first pixel in a frame is transmitted with the `startofpacket` control signal asserted. Likewise, the final pixel in a frame is transmitted with the `endofpacket` control signal asserted. The sink will assert `ready` when it is able to accept new pixel data. In the case of a VGA

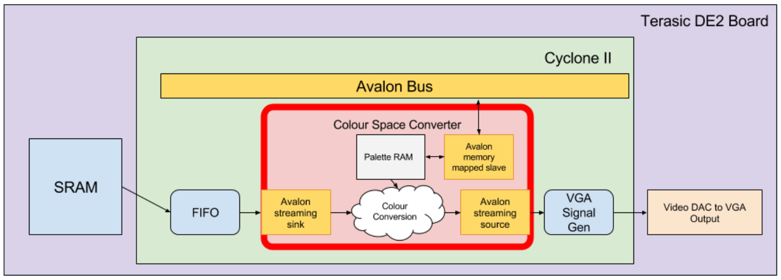


Figure 1: Block Diagram for Qsys Block

video signal, `ready` is asserted while pixels are being output to the display, and is de-asserted during the VGA blanking period. The `valid` control signal is asserted by the source when its data is ready to send. The Altera-provided video blocks expect this signal to always be asserted.

In figure 2, the end of transmitting a video frame is shown, with the last several pixels being transmitted, and then the first pixel of the following frame being sent, at which point the VGA controller begins a blanking period. Note that the next `data` value must be available before `ready` is asserted again.

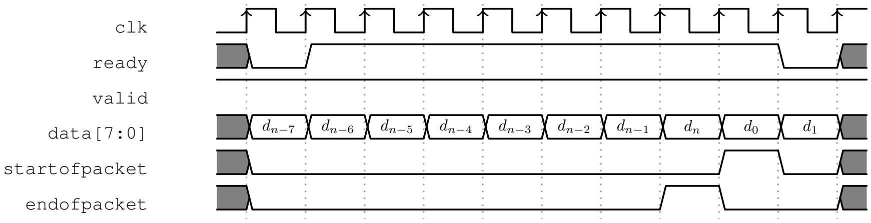


Figure 2: Avalon-ST Timing Diagram

5 Memory

The color palette shifter contains an instatiated dual port RAM. This dual-port RAM has one port attached as an Avalon slave, allowing the user to edit the palette. The other port of the RAM is attached to signals which are used in the state-machine that operates on the Avalon-ST bus in this block.

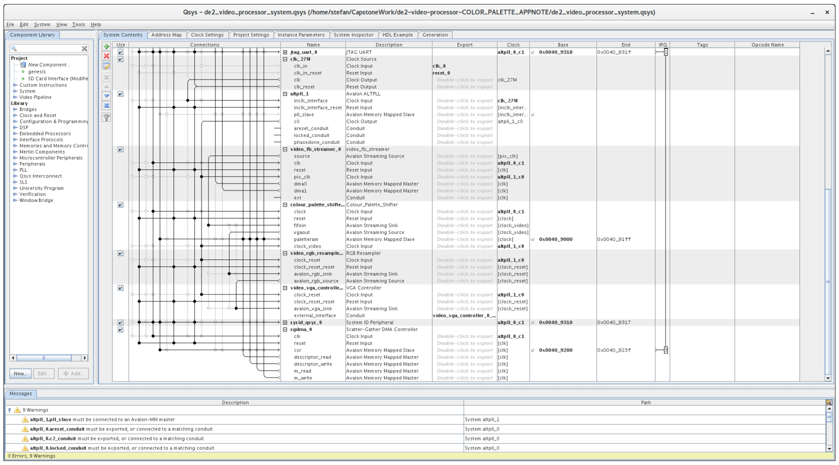
The addressing scheme for the palette is very simple. The 8-bit SRAM data "I", corresponds to an 8-bit address, which returns a 2-byte RGB565 value from dual_port[I].

6 Project Setup

The following sections detail how to configure a project using the provided video blocks.

6.1 Qsys

The QSYS configuration of this block follows the vga$_{p}ix_{b}ufconfiguration.Betweenthevideo\_fb\_streamerandth$



Use	Name	Description	Export	Clock	Base	End	IRQ
☑	jtag_uart_0	JTAG UART		altpll_0_c1	0x0040_9318	0x0040_931f	1
☑	clk_27M	Clock Source					
	clk_in	Clock Input	clk_0				
	clk_in_reset	Reset Input	reset_0				
	clk	Clock Output	Double-click to export	clk_27M			
	clk_reset	Reset Output	Double-click to export				
☑	altpll_1	Avalon ALTPLL					
	inclk_interface	Clock Input	Double-click to export	clk_27M			
	inclk_interface_reset	Reset Input	Double-click to export	[inclk_inter...			
	pll_slave	Avalon Memory Mapped Slave	Double-click to export	[inclk_inter...			
	c0	Clock Output	Double-click to export	altpll_1_c0			
	areset_conduit	Conduit	Double-click to export				
	locked_conduit	Conduit	Double-click to export				
	phasedone_conduit	Conduit	Double-click to export				
☑	video_fb_streamer_0	video_fb_streamer					
	source	Avalon Streaming Source	Double-click to export	[pix_clk]			
	clk	Clock Input	Double-click to export	altpll_0_c1			
	reset	Reset Input	Double-click to export	[clk]			
	pix_clk	Clock Input	Double-click to export	altpll_1_c0			
	dma0	Avalon Memory Mapped Master	Double-click to export	[clk]			
	dma1	Avalon Memory Mapped Master	Double-click to export	[clk]			
	ext	Conduit	Double-click to export	[clk]			
☑	colour_palette_shifte...	Colour_Palette_Shifter					
	clock	Clock Input	Double-click to export	altpll_0_c1			
	reset	Reset Input	Double-click to export	[clock]			
	fifoin	Avalon Streaming Sink	Double-click to export	[clock_video]			
	vgaout	Avalon Streaming Source	Double-click to export	[clock_video]			
	paletteram	Avalon Memory Mapped Slave	Double-click to export	[clock]	0x0040_9000	0x0040_91ff	
	clock_video	Clock Input	Double-click to export	altpll_1_c0			
☑	video_rgb_resample...	RGB Resampler					
	clock_reset	Clock Input	Double-click to export	altpll_1_c0			
	clock_reset_reset	Reset Input	Double-click to export	[clock_reset]			
	avalon_rgb_sink	Avalon Streaming Sink	Double-click to export	[clock_reset]			
	avalon_rgb_source	Avalon Streaming Source	Double-click to export	[clock_reset]			
☑	video_vga_controlle...	VGA Controller					
	clock_reset	Clock Input	Double-click to export	altpll_1_c0			
	clock_reset_reset	Reset Input	Double-click to export	[clock_reset]			
	avalon_vga_sink	Avalon Streaming Sink	Double-click to export	[clock_reset]			
	external_interface	Conduit	video_vga_controller_0_...				
☑	sysid_qsys_0	System ID Peripheral		altpll_0_c1	0x0040_9310	0x0040_9317	
☑	sgdma_0	Scatter-Gather DMA Controller					
	clk	Clock Input	Double-click to export	altpll_0_c1			
	reset	Reset Input	Double-click to export	[clk]			
	csr	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0040_9200	0x0040_923f	3
	descriptor_read	Avalon Memory Mapped Master	Double-click to export	[clk]			
	descriptor_write	Avalon Memory Mapped Master	Double-click to export	[clk]			
	m_read	Avalon Memory Mapped Master	Double-click to export	[clk]			
	m_write	Avalon Memory Mapped Master	Double-click to export	[clk]			

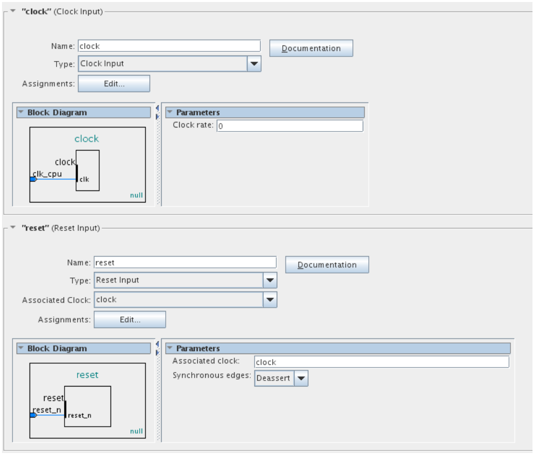
Description	Path
9 Warnings	
altpll_1.pll_slave must be connected to an Avalon-MM master	System.altpll_1
altpll_0.areset_conduit must be exported, or connected to a matching conduit.	System.altpll_0
altpll_0.c2_conduit must be exported, or connected to a matching conduit.	System.altpll_0
altpll_0.locked_conduit must be exported, or connected to a matching conduit.	System.altpll_0

0 Errors, 9 Warnings

Figure 3: Colour Palette Shifter in QSYS system.

The Colour Palette Shifter Block is configured as shown in the following screenshots.

Finally, the configuration for the `video_rgb_resampler` is shown in figure **??**. This block must be configured to go from 16-bit colour to 30-bit colour, to match the interfaces of the `colour_space_converter` and the `video_vga_controller`.



"clock" (Clock Input)

Name: clock
Type: Clock Input
Clock rate: 0

"reset" (Reset Input)

Name: reset
Type: Reset Input
Associated Clock: clock
Associated clock: clock
Synchronous edges: Deassert

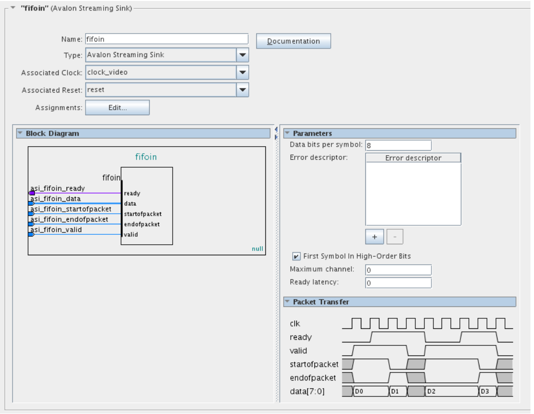
Figure 4: Configuration for `Colour_Space_Converter`

6.2 Quartus

You can use the Qsys-provided VHDL template to instantiate your system. This might look something like Listing 1 below.

Listing 1: Sample Top Level VHDL File

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY vga_pix_buffer IS
 PORT
 (
  -- Clocks
  CLOCK_50      : in        std_logic;
  CLOCK_27      : in        std_logic;

  -- SDRAM on board
  DRAM_ADDR     : out       std_logic_vector (11 downto 0);
  DRAM_BA_0     : out       std_logic;
  DRAM_BA_1     : out       std_logic;
  DRAM_CAS_N    : out       std_logic;
  DRAM_CKE      : out       std_logic;
  DRAM_CLK      : out       std_logic;
  DRAM_CS_N     : out       std_logic;
  DRAM_DQ       : inout     std_logic_vector (15 downto 0);
  DRAM_LDQM     : out       std_logic;
  DRAM_UDQM     : out       std_logic;
  DRAM_RAS_N    : out       std_logic;
  DRAM_WE_N     : out       std_logic;

  -- SRAM on board
  SRAM_ADDR     : out       std_logic_vector (17 downto 0);
  SRAM_DQ       : inout     std_logic_vector (15 downto 0);
  SRAM_WE_N     : out       std_logic;
```

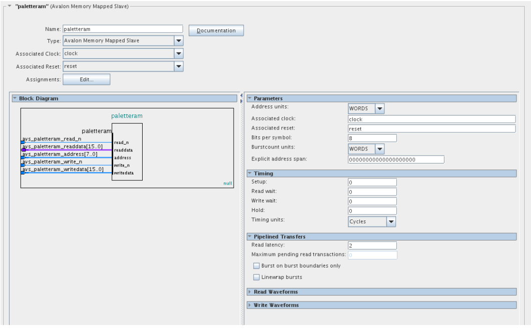


Name: fifoin
Type: Avalon Streaming Sink
Associated Clock: clock_video
Associated Reset: reset
Assignments:
Data bits per symbol: 8
Error descriptor:
☑ First Symbol In High-Order Bits
Maximum channel: 0
Ready latency: 0

Figure 5: Configuration for `Colour_Space_Converter`

Figure 6: Configuration for `Colour_Space_Converter`

```
  SRAM_OE_N     : out      std_logic;
  SRAM_UB_N     : out      std_logic;
  SRAM_LB_N     : out      std_logic;
  SRAM_CE_N     : out      std_logic;

  -- VGA output
  VGA_R         : out      std_logic_vector (9 downto 0);
  VGA_G         : out      std_logic_vector (9 downto 0);
  VGA_B         : out      std_logic_vector (9 downto 0);
  VGA_CLK       : out      std_logic;
  VGA_BLANK     : out      std_logic;
  VGA_HS        : out      std_logic;
  VGA_VS        : out      std_logic;
  VGA_SYNC      : out      std_logic;

  -- Input buttons
  KEY           : in       std_logic_vector (3 downto 0)
 );
END ENTITY vga_pix_buffer;

ARCHITECTURE arch OF vga_pix_buffer IS

 COMPONENT vga_pix_buffer_system IS
  PORT (
   clk_50_clk                  : in    std_logic            := ’X’;
   reset_50_reset_n            : in    std_logic            := ’X’;
   clk_27_clk                  : in    std_logic            := ’X’;
   reset_27_reset_n            : in    std_logic            := ’X’;
   sram_0_external_interface_DQ                    : inout
       std_logic_vector(15 downto 0) := (others => ’X’);
   sram_0_external_interface_ADDR                  : out
       std_logic_vector(17 downto 0);
   sram_0_external_interface_LB_N                  : out    std_logic;
```



"paletteram" (Avalon Memory Mapped Slave)

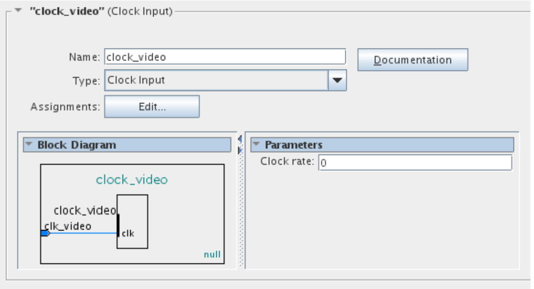
Name: paletteram
Type: Avalon Memory Mapped Slave
Associated Clock: clock
Associated Reset: reset
Assignments:

Parameters
Address units: WORDS
Associated clock: clock
Associated reset: reset
Bits per symbol: 8
Burstcount units: WORDS
Explicit address span: 00000000000000000000

Timing
Setup: 0
Read wait: 0
Write wait: 0
Hold: 0
Timing units: Cycles

Pipelined Transfers
Read latency: 2
Maximum pending read transactions: 0
☐ Burst on burst boundaries only
☐ Linewrap bursts

Figure 7: Configuration for `Colour_Space_Converter`



"clock_video" (Clock Input)

Name: clock_video
Type: Clock Input
Assignments:

Parameters
Clock rate: 0

Figure 8: Configuration for `Colour_Space_Converter`

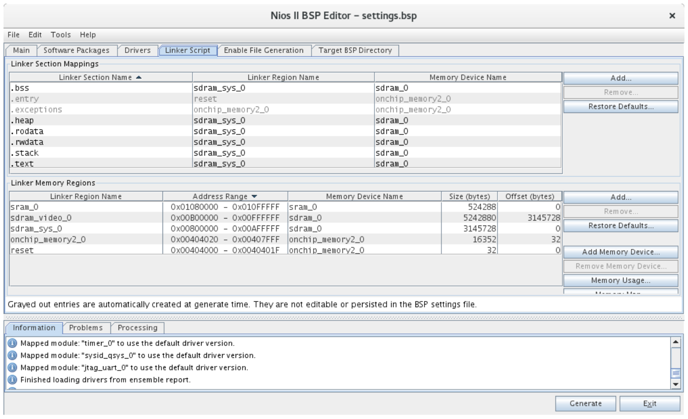
```
sram_0_external_interface_UB_N                : out    std_logic;
sram_0_external_interface_CE_N                : out    std_logic;
sram_0_external_interface_OE_N                : out    std_logic;
sram_0_external_interface_WE_N                : out    std_logic;
video_vga_controller_0_external_interface_CLK : out    std_logic;
video_vga_controller_0_external_interface_HS  : out    std_logic;
video_vga_controller_0_external_interface_VS  : out    std_logic;
video_vga_controller_0_external_interface_BLANK : out  std_logic;
video_vga_controller_0_external_interface_SYNC : out   std_logic;
video_vga_controller_0_external_interface_R   : out
    std_logic_vector(9 downto 0);
video_vga_controller_0_external_interface_G   : out
    std_logic_vector(9 downto 0);
video_vga_controller_0_external_interface_B   : out
    std_logic_vector(9 downto 0);
sdram_0_wire_addr                             : out
    std_logic_vector(11 downto 0);
sdram_0_wire_ba                               : out
    std_logic_vector(1 downto 0);
sdram_0_wire_cas_n                            : out    std_logic;
sdram_0_wire_cke                              : out    std_logic;
sdram_0_wire_cs_n                             : out    std_logic;
sdram_0_wire_dq                               : inout
    std_logic_vector(15 downto 0) := (others => 'X');
```

```
   sdram_0_wire_dqm                                : out
       std_logic_vector(1 downto 0);
   sdram_0_wire_ras_n                              : out   std_logic;
   sdram_0_wire_we_n                               : out   std_logic;
   altpll_sys_dram_c0_clk                          : out   std_logic
  );
 END COMPONENT vga_pix_buffer_system;

 -- Signals to interface with DRAM
 SIGNAL BA   : std_logic_vector (1 downto 0);
 SIGNAL DQM  : std_logic_vector (1 downto 0);

BEGIN

 DRAM_BA_1 <= BA(1);
 DRAM_BA_0 <= BA(0);

 DRAM_UDQM <= DQM(1);
 DRAM_LDQM <= DQM(0);

 sys0 : COMPONENT vga_pix_buffer_system
  PORT MAP (
   clk_50_clk                                      => CLOCK_50,
   reset_50_reset_n                                => KEY(0),
   clk_27_clk                                      => CLOCK_27,
   reset_27_reset_n                                => KEY(0),
   sram_0_external_interface_DQ                    => SRAM_DQ,
   sram_0_external_interface_ADDR                  => SRAM_ADDR,
   sram_0_external_interface_LB_N                  => SRAM_LB_N,
   sram_0_external_interface_UB_N                  => SRAM_UB_N,
   sram_0_external_interface_CE_N                  => SRAM_CE_N,
   sram_0_external_interface_OE_N                  => SRAM_OE_N,
   sram_0_external_interface_WE_N                  => SRAM_WE_N,
   video_vga_controller_0_external_interface_CLK   => VGA_CLK,
   video_vga_controller_0_external_interface_HS    => VGA_HS,
   video_vga_controller_0_external_interface_VS    => VGA_VS,
   video_vga_controller_0_external_interface_BLANK => VGA_BLANK,
   video_vga_controller_0_external_interface_SYNC  => VGA_SYNC,
   video_vga_controller_0_external_interface_R     => VGA_R,
   video_vga_controller_0_external_interface_G     => VGA_G,
   video_vga_controller_0_external_interface_B     => VGA_B,
   sdram_0_wire_addr                               => DRAM_ADDR,
   sdram_0_wire_ba                                 => BA,
   sdram_0_wire_cas_n                              => DRAM_CAS_N,
   sdram_0_wire_cke                                => DRAM_CKE,
   sdram_0_wire_cs_n                               => DRAM_CS_N,
   sdram_0_wire_dq                                 => DRAM_DQ,
   sdram_0_wire_dqm                                => DQM,
   sdram_0_wire_ras_n                              => DRAM_RAS_N,
   sdram_0_wire_we_n                               => DRAM_WE_N,
   altpll_sys_dram_c0_clk                          => DRAM_CLK
  );

END ARCHITECTURE arch;
```

6.3 Nios II SBT for Eclipse

This section is identical to the vga$_pixbufapplicationnoteasthisapplicationnoteextendsthepreviousone.$

When you create a new "Nios II Application and BSP from Template" project in Eclipse, you must manually configure the memory map used in the BSP project. This configuration is shown in figure 9. In particular note the definitions for `sdram_video_0` and `sdram_sys_0`. This must match your planned memory layout. If you compare the memory addresses in figure 9 and the `SDRAM_VIDEO_OFFSET` value in the sample code in Listing **??**, they should refer to the same location in memory.



Linker Section Mappings

Linker Section Name ▲	Linker Region Name	Memory Device Name
.bss	sdram_sys_0	sdram_0
.entry	reset	onchip_memory2_0
.exceptions	onchip_memory2_0	onchip_memory2_0
.heap	sdram_sys_0	sdram_0
.rodata	sdram_sys_0	sdram_0
.rwdata	sdram_sys_0	sdram_0
.stack	sdram_sys_0	sdram_0
.text	sdram_sys_0	sdram_0

Linker Memory Regions

Linker Region Name	Address Range ▼	Memory Device Name	Size (bytes)	Offset (bytes)
sram_0	0x01080000 – 0x010FFFFF	sram_0	524288	0
sdram_video_0	0x00B00000 – 0x00FFFFFF	sdram_0	5242880	3145728
sdram_sys_0	0x00800000 – 0x00AFFFFF	sdram_0	3145728	0
onchip_memory2_0	0x00404020 – 0x00407FFF	onchip_memory2_0	16352	32
reset	0x00404000 – 0x0040401F	onchip_memory2_0	32	0

Grayed out entries are automatically created at generate time. They are not editable or persisted in the BSP settings file.

```
Mapped module: "timer_0" to use the default driver version.
Mapped module: "sysid_qsys_0" to use the default driver version.
Mapped module: "jtag_uart_0" to use the default driver version.
Finished loading drivers from ensemble report.
```

Figure 9: BSP Linker Configuration

7 Sample Code

The following code draws a test palette, switches some

Listing 2: Sample Program that Generates a Test Pattern, then changes the color

```
#include <stdio.h>
#include <io.h>
#include <system.h>
#include <sys/alt_stdio.h>
#include <string.h>

unsigned int palette_ega[16] =
    /* EGA Colour Palette */
    /* Black        Blue          Green        Cyan          Red */
    {/*00*/ 0x0000, /*01*/0x0015, /*02*/0x2704, /*03*/0x1E79, /*04*/0
        xA800,
     /*     Magenta       Brown    Light Grey     Dark Grey   Bright
         Blue*/
     /*05*/ 0xA815, /*06*/0xE3C1, /*07*/0xAD55, /*08*/0x52AA, /*09*/0
         x52BF,
      /* BGreen            BCyan         Bred       B Magenta     B
          Yellow*/
      /*10*/ 0x57EA, /*11*/0x57FF, /*12*/0xFAAA, /*13*/0xFABF, /*14*/0
          xFFEA,
      0xFFFF /*B White */
     };

unsigned int palette_stephen[256] =
    /* Stephen's RGB323 -> RGB565 Palette */
    {0x0 , 0xa , 0x15 , 0x1f , 0x120 , 0x12a , 0x135 , 0x13f , 0x240 ,
    0x24a , 0x255 , 0x25f , 0x360 , 0x36a , 0x375 , 0x37f , 0x480 ,
    0x48a , 0x495 , 0x49f , 0x5a0 , 0x5aa , 0x5b5 , 0x5bf , 0x6c0 ,
    0x6ca , 0x6d5 , 0x6df , 0x7e0 , 0x7ea , 0x7f5 , 0x7ff , 0x2000 ,
    0x200a , 0x2015 , 0x201f , 0x2120 , 0x212a , 0x2135 , 0x213f , 0
        x2240 ,
    0x224a , 0x2255 , 0x225f , 0x2360 , 0x236a , 0x2375 , 0x237f , 0
        x2480 ,
    0x248a , 0x2495 , 0x249f , 0x25a0 , 0x25aa , 0x25b5 , 0x25bf , 0
        x26c0 ,
    0x26ca , 0x26d5 , 0x26df , 0x27e0 , 0x27ea , 0x27f5 , 0x27ff , 0
        x4800 ,
    0x480a , 0x4815 , 0x481f , 0x4920 , 0x492a , 0x4935 , 0x493f , 0
        x4a40 ,
    0x4a4a , 0x4a55 , 0x4a5f , 0x4b60 , 0x4b6a , 0x4b75 , 0x4b7f , 0
        x4c80 ,
    0x4c8a , 0x4c95 , 0x4c9f , 0x4da0 , 0x4daa , 0x4db5 , 0x4dbf , 0
        x4ec0 ,
    0x4eca , 0x4ed5 , 0x4edf , 0x4fe0 , 0x4fea , 0x4ff5 , 0x4fff , 0
        x6800 ,
    0x680a , 0x6815 , 0x681f , 0x6920 , 0x692a , 0x6935 , 0x693f , 0
        x6a40 ,
    0x6a4a , 0x6a55 , 0x6a5f , 0x6b60 , 0x6b6a , 0x6b75 , 0x6b7f , 0
        x6c80 ,
    0x6c8a , 0x6c95 , 0x6c9f , 0x6da0 , 0x6daa , 0x6db5 , 0x6dbf , 0
        x6ec0 ,
    0x6eca , 0x6ed5 , 0x6edf , 0x6fe0 , 0x6fea , 0x6ff5 , 0x6fff , 0
        x9000 ,
    0x900a , 0x9015 , 0x901f , 0x9120 , 0x912a , 0x9135 , 0x913f , 0
        x9240 ,
    0x924a , 0x9255 , 0x925f , 0x9360 , 0x936a , 0x9375 , 0x937f , 0
        x9480 ,
    0x948a , 0x9495 , 0x949f , 0x95a0 , 0x95aa , 0x95b5 , 0x95bf , 0
        x96c0 ,
    0x96ca , 0x96d5 , 0x96df , 0x97e0 , 0x97ea , 0x97f5 , 0x97ff , 0
        xb000 ,
    0xb00a , 0xb015 , 0xb01f , 0xb120 , 0xb12a , 0xb135 , 0xb13f , 0
        xb240 ,
    0xb24a , 0xb255 , 0xb25f , 0xb360 , 0xb36a , 0xb375 , 0xb37f , 0
        xb480 ,
    0xb48a , 0xb495 , 0xb49f , 0xb5a0 , 0xb5aa , 0xb5b5 , 0xb5bf , 0
        xb6c0 ,
    0xb6ca , 0xb6d5 , 0xb6df , 0xb7e0 , 0xb7ea , 0xb7f5 , 0xb7ff , 0
        xd800 ,
    0xd80a , 0xd815 , 0xd81f , 0xd920 , 0xd92a , 0xd935 , 0xd93f , 0
        xda40 ,
    0xda4a , 0xda55 , 0xda5f , 0xdb60 , 0xdb6a , 0xdb75 , 0xdb7f , 0
        xdc80 ,
    0xdc8a , 0xdc95 , 0xdc9f , 0xdda0 , 0xddaa , 0xddb5 , 0xddbf , 0
        xdec0 ,
    0xdeca , 0xded5 , 0xdedf , 0xdfe0 , 0xdfea , 0xdff5 , 0xdfff , 0
        xf800 ,
    0xf80a , 0xf815 , 0xf81f , 0xf920 , 0xf92a , 0xf935 , 0xf93f , 0
        xfa40 ,
    0xfa4a , 0xfa55 , 0xfa5f , 0xfb60 , 0xfb6a , 0xfb75 , 0xfb7f , 0
        xfc80 ,
    0xfc8a , 0xfc95 , 0xfc9f , 0xfda0 , 0xfdaa , 0xfdb5 , 0xfdbf , 0
        xfec0 ,
    0xfeca , 0xfed5 , 0xfedf , 0xffe0 , 0xffea , 0xfff5 , 0xffff };

unsigned int palette_magenta[256] = {0xA815};

unsigned int bunch_o_blues[256] = {0x000 , 0x001 , 0x002 , 0x003 , 0
    x004 , 0x005 , 0x006 , 0x007 , 0x008 ,
    0x009 , 0x00a , 0x00b , 0x00c , 0x00d , 0x00e , 0x00f , 0x0010 ,
    0x0011 , 0x0012 , 0x0013 , 0x0014 , 0x0015 , 0x0016 , 0x0017 , 0
        x0018 ,
    0x0019 , 0x001a , 0x001b , 0x001c , 0x001d , 0x001e , 0x001f };

unsigned int bunch_o_reds[256] = {0x000 , 0x010 , 0x020 , 0x030 , 0x040
     , 0x050 , 0x060 , 0x070 , 0x080 ,
    0x090 , 0x0a0 , 0x0b0 , 0x0c0 , 0x0d0 , 0x0e0 , 0x0f0 , 0x0100 ,
    0x0110 , 0x0120 , 0x0130 , 0x0140 , 0x0150 , 0x0160 , 0x0170 , 0
        x0180 ,
    0x0190 , 0x01a0 , 0x01b0 , 0x01c0 , 0x01d0 , 0x01e0 , 0x01f0};



unsigned int* palettes[6] = {&palette_ega[0], &palette_stephen[0], &
    palette_magenta[0], &bunch_o_blues[0], &bunch_o_reds[0]};
unsigned int palettes_legngth = 5;



//Make a function to switch the palettes given a pointer to a palette
    array.
void switchPalette(unsigned int* palette, int length){

  int i = 0;
  for (i = 0; i < length; i++){
    IOWR_16DIRECT(COLOUR_PALETTE_SHIFTER_0_BASE, 2*i, 0x0000);
    IOWR_16DIRECT(COLOUR_PALETTE_SHIFTER_0_BASE, 2*i, palette[i]);
  }


}


void printPalette(int n){
  // Print everything in the palette ram, upto int colours.
  int i;
  unsigned int c;

  unsigned int results[512] = {'\0'};
  for (i = 0; i < n; i++){
    c = IORD_16DIRECT(COLOUR_PALETTE_SHIFTER_0_BASE, 2*i); //offset
        multiplied by 2 to be on 16-bit boundaries.
    //alt_printf("palette[ %x ]: %x ", 2*i, c);
    results[i] = c;
  }

  for (i = 0; i < n; i++){
    alt_printf("palette[␣%x␣]:␣%x␣", 2*i, results[i]);
  }

}

int main()
{


  unsigned int row = 0;
  unsigned int col = 0;
  unsigned int delay = 0;

  unsigned int color;
  int i;



  // Clear the screen first.
  alt_putstr("Clear␣the␣screen\n");
  for (col = 0; col < 640; col = col + 4){
    for (row = 0; row < 480; row++){
      color = 0;
      IOWR_32DIRECT(SRAM_0_BASE, row * 640 + col, color << 24 | color
          << 16 | color << 8 | color << 0);
    }
  }


  //Switch palettes here (to EGA colour palette)
  printPalette(16);
  switchPalette(palette_ega, 16);
  printPalette(16);

  alt_putstr("Screen␣painting␣demo\n");

  for (i = 0; i < 16; i++){
    alt_printf("Colour");
    for (delay = 0; delay < 700/*2000*/; delay++){
      unsigned int tdelay = delay;
      for (tdelay; tdelay > 0; tdelay--){}
    }

    for (row=0; row<480; row++){
      for (col = 0; col < 640; col=col+4){
        color = i;
        IOWR_32DIRECT(SRAM_0_BASE, row * 640 + col, color << 24 | color
             << 16 | color << 8 | color << 0);
      }
    }
  }



  alt_putstr("\nStarting␣Stephen␣Test␣Pattern\n");
  int p = 0;
  for (p = 0; p < 8; p++) {

    switchPalette(palettes[p%palettes_legnth], 256);

    alt_putstr("Clear␣the␣screen\n");
    for (col = 0; col < 640; col = col + 4){
      for (row = 0; row < 480; row++){
        color = 0;
        IOWR_32DIRECT(SRAM_0_BASE, row * 640 + col, color << 24 | color
             << 16 | color << 8 | color << 0);
      }
    }


    //Now draw the test pattern
    for (row = 0; row < 480; row++)
    {
      for (col = 0; col < 640; col = col + 4)
      {
        color = ((row + col) % 256) << 0 | ((row + col) % 256) << 8 |
            ((row + col) % 256) << 16 | ((row + col) % 256) << 24;
        if (row == 0 || row == 479)
        {
          IOWR_32DIRECT(SRAM_0_BASE, row * 640 + col, 0xFFFFFFFF);
        }
        else if (col == 0)
        {
          IOWR_32DIRECT(SRAM_0_BASE, row * 640 + col, 0x000000FF |
              color);
        }
        else if (col == 636)
        {
          IOWR_32DIRECT(SRAM_0_BASE, row * 640 + col, 0xFF000000 |
              color);
        }
        else
        {
          IOWR_32DIRECT(SRAM_0_BASE, row * 640 + col, color);
        }
      }
    }


  }
```

```
  alt_putstr("Done.\n");


  // Do Stephen’s pattern display.

  // And switch the palettes.


  return 0;
}
```