VGA Pixel Buffer

Stephen Just 2016-02-20

1 Introduction

Video output is often a useful addition to interactive projects but typically there have been many performance limitations with respect to video out on the DE2. This application note details how it is possible to provide a double buffered video output without creating large memory bandwidth pressures on your system's SDRAM that you need to run your application code.

The code package provided with this application note is designed to work on a DE2 board using any Nios II processor. As with most other applications, the best performance can be achieved using a faster variant of the processor.

Using the provided code package, you will be able to generate video output at a resolution of 640x480 with a video refresh rate of 60Hz and use 8-bit colour. The maximum achievable frame update rate will vary depending on how you choose to populate the video frame buffers.

2 Clocking

Outputting a video signal requires a variety of clocks. Your system clock, usually running at 50MHz, will be faster than the video pixel clock. For a 640x480@60Hz video signal, a typical pixel clock operates at 25.175MHz. [1] The DE2 is not capable of producing that exact frequency, but it can come close enough with a 25.2MHz clock that can be derived from the 27MHz oscillator on the board.

To generate a 25.2MHz clock, you can connect the 27MHz clock input to a PLL, and configure the PLL with a multiplication factor of 14 and a division factor of 15. In the provided sample code, this is produced by the `altpll_video` Qsys component.

3 VGA Output

Altera provides a video output module in the University Program IP collection to generate the necessary signals to output to the VGA port on the DE2. This IP is designed to output a 640x480 pixel video signal. The refresh rate of this VGA signal is determined by the clock you supply to this component. In the provided sample code, the VGA output block is called `video_vga_controller_0`. [2]

The Altera-provided output block has some notable quirks that you should keep in mind if you want to use it in your own project. Most importantly, it does not handle incorrectly formatted input in an acceptable way. For example, if you generate an Avalon-ST data stream with packets of incorrect length, you will completely lose your video sync and the picture on the screen will appear incorrectly. The provided `video_fb_streamer_0` component in the example code correctly outputs video data for the VGA output block.

In figure 1, the relationship between a row of pixel data and the VGA `hsync` signal is shown. The period surrounding the `hsync` pulse where no data is visible is called the horizontal blanking period. The `hsync` pulse is preceded by what is called the horizontal front-porch, and followed by what is called the back-porch. All three of these components make up the horizontal blanking period.

There is a similar mechanism for vertical timing, where the vertical blanking period is several multiples of the horizontal timing period long.

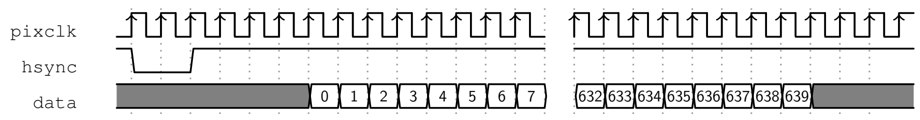


Figure 1: VGA Horizontal Timing

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 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.

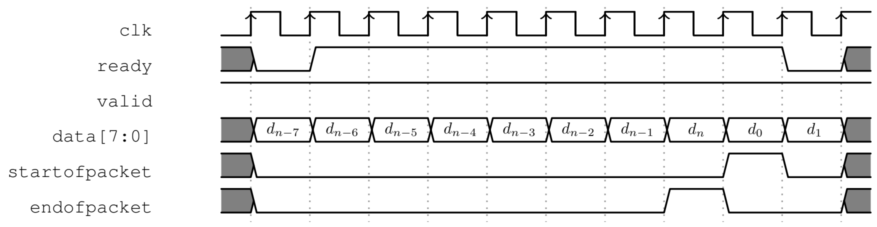


Figure 2: Avalon-ST Timing Diagram

5 Memory

Altera provides a pixel frame-buffer component similar to the one provided with this application note in their University Program IP collection, but it was determined to not be suitable for high performance graphics. The Altera solution provides two pixel frame-buffers in SDRAM. Either of the buffers can be output to the VGA controller, with a simple command to swap between which one is active. The problem that this solution has is that the VGA controller must always be reading from the SDRAM so that it has pixel data available to output when it is required. This means that the system processor must compete with the video output pipeline for access to the SDRAM, which slows down any drawing operations. If the application code is also running from SDRAM, this effectively requires the designer to use a variant of the Nios II processor with a cache.

In the frame-buffer component provided with this application note, a "live" frame-buffer exists in SRAM which is continuously read to the video output pipeline, and a "working" frame-buffer exists in SDRAM. A Nios II custom instruction is provided, `ci_frame_swap_0`, to trigger a fast copy from the SDRAM buffer to SRAM.

Because of this arrangement of frame-buffers, it is possible to quickly perform draw operations to the "working" buffer without potentially interrupting the video output component or without the video output component blocking the processor.

The copy from SDRAM to SRAM is implemented in such a way that the output video signal will not experience any video tearing. Tearing occurs when the frame data changes while the frame is being drawn, so that the user sees parts of two different versions of the frame at once.

Because part of the SDRAM is dedicated to act as a frame-buffer, it is necessary that when you write a Nios II application to use this functionality, you modify the linker settings in your BSP so that the area of memory allocated to your program does not intersect with that of the frame-buffer. This can be seen in the provided example code.

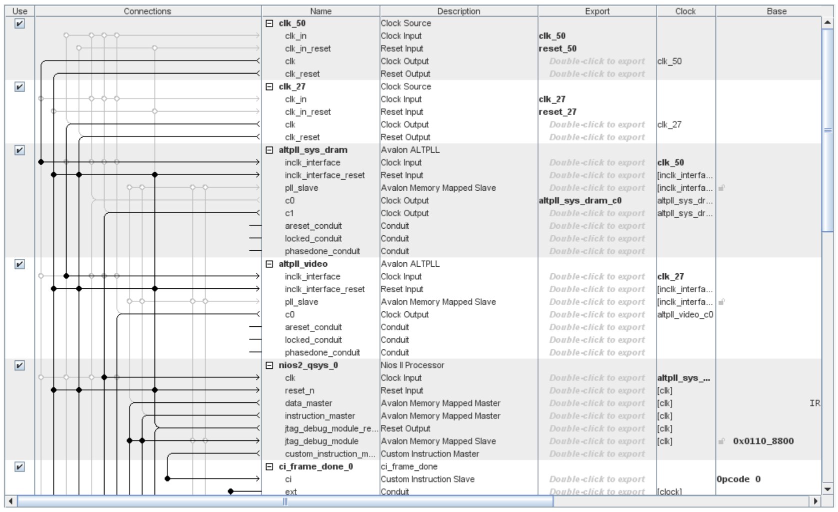
6 Project Setup

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

6.1 Qsys

In Qsys, in addition to the components required for a simple Nios II system, you must also instantiate a second clock input, a second Avalon ALTPLL, a `ci_frame_done` custom instruction, a `video_fb_streamer` component, a `video_rgb_resampler` component, and a `video_vga_controller` component.

In figure 3, note that the 50MHz clock is the source of `altpll_sys_dram`, whereas the 27MHz clock is the source of `altpll_video`. The `c0` output of `altpll_video` is only connected to the blocks that are part of the video output pipeline, shown in figure 4.



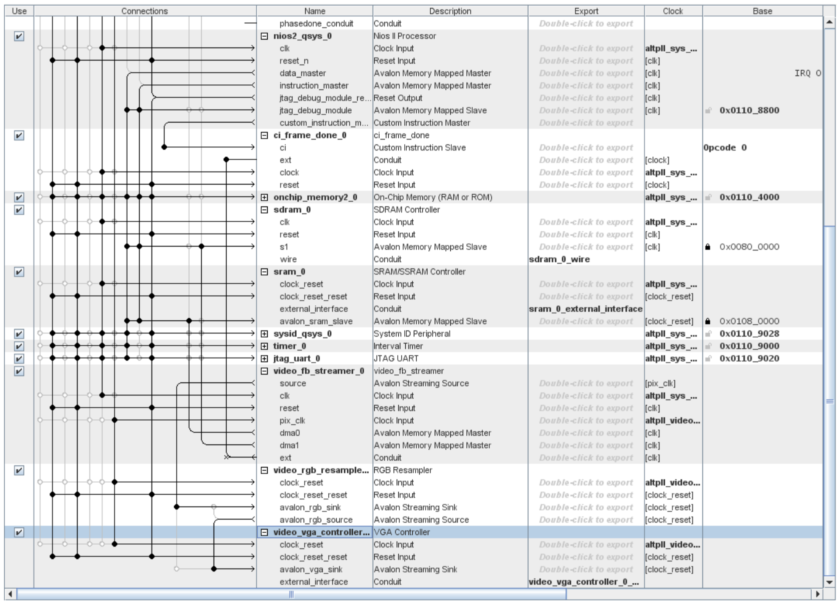
Use	Connections	Name	Description	Export	Clock	Base
✓		**clk_50**	Clock Source			
		clk_in	Clock Input	**clk_50**		
		clk_in_reset	Reset Input	**reset_50**		
		clk	Clock Output	*Double-click to export*	clk_50	
		clk_reset	Reset Output	*Double-click to export*		
✓		**clk_27**	Clock Source			
		clk_in	Clock Input	**clk_27**		
		clk_in_reset	Reset Input	**reset_27**		
		clk	Clock Output	*Double-click to export*	clk_27	
		clk_reset	Reset Output	*Double-click to export*		
✓		**altpll_sys_dram**	Avalon ALTPLL			
		inclk_interface	Clock Input	*Double-click to export*	**clk_50**	
		inclk_interface_reset	Reset Input	*Double-click to export*	[inclk_interfa...	
		pll_slave	Avalon Memory Mapped Slave	*Double-click to export*	[inclk_interfa...	
		c0	Clock Output	**altpll_sys_dram_c0**	altpll_sys_dr...	
		c1	Clock Output	*Double-click to export*	altpll_sys_dr...	
		areset_conduit	Conduit	*Double-click to export*		
		locked_conduit	Conduit	*Double-click to export*		
		phasedone_conduit	Conduit	*Double-click to export*		
✓		**altpll_video**	Avalon ALTPLL			
		inclk_interface	Clock Input	*Double-click to export*	**clk_27**	
		inclk_interface_reset	Reset Input	*Double-click to export*	[inclk_interfa...	
		pll_slave	Avalon Memory Mapped Slave	*Double-click to export*	[inclk_interfa...	
		c0	Clock Output	*Double-click to export*	altpll_video_c0	
		areset_conduit	Conduit	*Double-click to export*		
		locked_conduit	Conduit	*Double-click to export*		
		phasedone_conduit	Conduit	*Double-click to export*		
✓		**nios2_qsys_0**	Nios II Processor			
		clk	Clock Input	*Double-click to export*	**altpll_sys_...**	
		reset_n	Reset Input	*Double-click to export*	[clk]	
		data_master	Avalon Memory Mapped Master	*Double-click to export*	[clk]	IR
		instruction_master	Avalon Memory Mapped Master	*Double-click to export*	[clk]	
		jtag_debug_module_re...	Reset Output	*Double-click to export*	[clk]	
		jtag_debug_module	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	**0x0110_8800**
		custom_instruction_m...	Custom Instruction Master	*Double-click to export*		
✓		**ci_frame_done_0**	ci_frame_done			
		ci	Custom Instruction Slave	*Double-click to export*		**Opcode 0**
		ext	Conduit	*Double-click to export*	[clock]	

Figure 3: Clock Connections for Qsys System

It is important to note how the each of the video output blocks are connected to the rest of the system. The `video_fb_streamer` component has two Avalon-MM Master interfaces on it. Interface `dma0` must connect to the SRAM component, while `dma1` must connect to the SDRAM component. This allows the streamer component to talk to both memories simultaneously.

Another important thing to note is that the conduits of `ci_frame_done` and `video_fb_streamer` are connected together. This needs to be present because the custom instruction interfaces directly with the video streamer.

In addition to these things, it is important that the base memory address for the SRAM and SDRAM components are locked. When you configure the video_fb_streamer component, you must provide memory addresses for both the SDRAM and SRAM video buffers. If the memory addresses for your RAM components changed, then the streamer component would break. With the base addresses shown in figure 4, you can configure the video_fb_streamer block as shown in figure 5.



Use	Connections	Name	Description	Export	Clock	Base
		phasedone_conduit	Conduit	Double-click to export		
✓		⊟ nios2_qsys_0	Nios II Processor			
		clk	Clock Input	Double-click to export	altpll_sys_...	
		reset_n	Reset Input	Double-click to export	[clk]	
		data_master	Avalon Memory Mapped Master	Double-click to export	[clk]	IRQ 0
		instruction_master	Avalon Memory Mapped Master	Double-click to export	[clk]	
		jtag_debug_module_re...	Reset Output	Double-click to export	[clk]	
		jtag_debug_module	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0110_8800
		custom_instruction_m...	Custom Instruction Master	Double-click to export		
✓		⊟ ci_frame_done_0	ci_frame_done			
		ci	Custom Instruction Slave	Double-click to export		Opcode 0
		ext	Conduit	Double-click to export	[clock]	
		clock	Clock Input	Double-click to export	altpll_sys_...	
		reset	Reset Input	Double-click to export	[clock]	
✓		⊞ onchip_memory2_0	On-Chip Memory (RAM or ROM)		altpll_sys_...	0x0110_4000
✓		⊟ sdram_0	SDRAM Controller			
		clk	Clock Input	Double-click to export	altpll_sys_...	
		reset	Reset Input	Double-click to export	[clk]	
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0080_0000
		wire	Conduit	sdram_0_wire		
✓		⊟ sram_0	SRAM/SSRAM Controller			
		clock_reset	Clock Input	Double-click to export	altpll_sys_...	
		clock_reset_reset	Reset Input	Double-click to export	[clock_reset]	
		external_interface	Conduit	sram_0_external_interface		
		avalon_sram_slave	Avalon Memory Mapped Slave	Double-click to export	[clock_reset]	0x0108_0000
✓		⊞ sysid_qsys_0	System ID Peripheral		altpll_sys_...	0x0110_9028
✓		⊞ timer_0	Interval Timer		altpll_sys_...	0x0110_9000
✓		⊞ jtag_uart_0	JTAG UART		altpll_sys_...	0x0110_9020
✓		⊟ 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_sys_...	
		reset	Reset Input	Double-click to export	[clk]	
		pix_clk	Clock Input	Double-click to export	altpll_video...	
		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]	
✓		⊟ video_rgb_resample...	RGB Resampler			
		clock_reset	Clock Input	Double-click to export	altpll_video...	
		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_controller...	VGA Controller			
		clock_reset	Clock Input	Double-click to export	altpll_video...	
		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_...		

Figure 4: Connections for Qsys Video Blocks

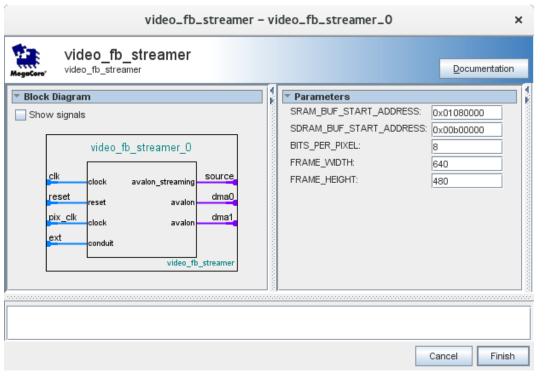
Finally, the configuration for the video_rgb_resampler is shown in figure 6. This block must be configured to go from 8-bit colour to 30-bit colour, to match the interfaces of the video_fb_streamer and the video_vga_controller.

6.2 Quartus

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

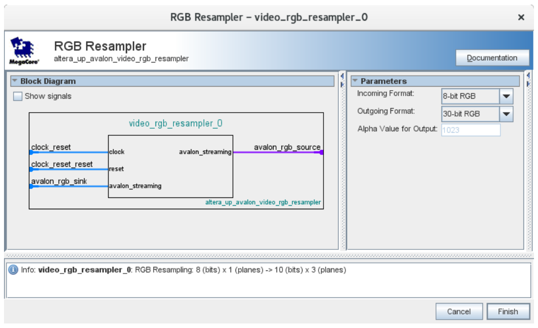
Listing 1: Sample Top Level VHDL File

```
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
```



SRAM_BUF_START_ADDRESS: 0x01080000
SDRAM_BUF_START_ADDRESS: 0x00b00000
BITS_PER_PIXEL: 8
FRAME_WIDTH: 640
FRAME_HEIGHT: 480

Figure 5: Configuration for `video_fb_streamer`



Incoming Format: 8-bit RGB
Outgoing Format: 30-bit RGB
Alpha Value for Output: 1023

Info: **video_rgb_resampler_0**: RGB Resampling: 8 (bits) x 1 (planes) -> 10 (bits) x 3 (planes)

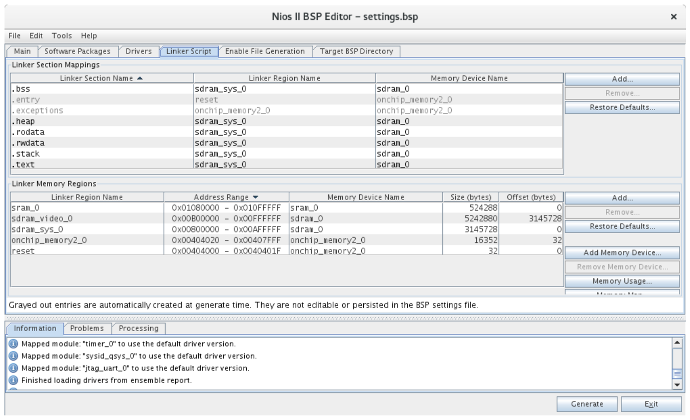
Figure 6: Configuration for `video_rgb_resampler`

```
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;
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;
   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

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 7. 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 7 and the `SDRAM_VIDEO_OFFSET` value in the sample code in Listing 2, 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 7: BSP Linker Configuration

7 Sample Code

The following code shows how you might draw a simple animation to the video output.

Listing 2: Sample Program that Generates a Moving Line

```c
/* This test program generates a simple pattern to test for tearing.
 *
 * The video pattern consists of a white vertical line that will move
 * from side to side along the frame. If there is tearing, the line
 * will appear broken at some points in time.
 */

#include <io.h>
#include <system.h>

#define SDRAM_VIDEO_OFFSET 0x300000
#define FRAME_WIDTH 640
#define FRAME_HEIGHT 480
#define COLOR_BLACK 0x00
#define COLOR_WHITE 0xFF

int main()
{
  int row = 0;
  int col = 0;

  // Clear the screen
  for (row = 0; row < FRAME_HEIGHT; row++)
  {
    for (col = 0; col < FRAME_WIDTH; col = col + 4)
    {
      IOWR_32DIRECT(SDRAM_0_BASE, SDRAM_VIDEO_OFFSET + row *
          FRAME_WIDTH + col, COLOR_BLACK);
    }
  }
  ALT_CI_CI_FRAME_DONE_0; // Custom command to trigger frame swap

  // Draw pattern
  unsigned int position = 0;
  while (1)
  {
    for (row = 0; row < FRAME_HEIGHT; row++)
    {
      // Clear previous position of line
      if (position == 0) {
        IOWR_8DIRECT(SDRAM_0_BASE, SDRAM_VIDEO_OFFSET + row *
            FRAME_WIDTH + FRAME_WIDTH - 8, COLOR_BLACK);
      } else {
        IOWR_8DIRECT(SDRAM_0_BASE, SDRAM_VIDEO_OFFSET + row *
            FRAME_WIDTH + position - 8, COLOR_BLACK);
      }
      // Draw new line
      IOWR_8DIRECT(SDRAM_0_BASE, SDRAM_VIDEO_OFFSET + row * FRAME_WIDTH
           + position, COLOR_WHITE);
    }

    position = (position + 8) % 640;
    ALT_CI_CI_FRAME_DONE_0; // Trigger frame swap
  }

  return 0;
}
```

References

[1] TinyVGA, "Vga signal 640 x 480 @ 60 hz industry standard timing." http://tinyvga.com/vga-timing/640x480@60Hz. Accessed: 2016-02-20.

[2] Altera, "Video ip cores for altera de-series boards." ftp://ftp.altera.com/up/pub/Altera_Material/12.1/University_Program_IP_Cores/Audio_Video/Video.pdf. Accessed: 2016-02-20.