Tutorial of displaying pixels on VGA monitor

Jun Zhao (Group 3)

Feb 12, 2014

Introduction

This tutorial will demonstrate how to connect a VGA monitor with Altera DE2 board, and display pixels on VGA monitor.

Procedure

- Use Qsys to create an embedded system, and we need SRAM/SSRAM Controller, Pixel Buffer DMA Controller, RGB Resampler, Scaler, Dual-Clock FIFO, VGA Controller. In reference section, the *Video.h*explained the functionalities of the above components on page3 in details.
- Clock signals for DE-series Board Peripherals
 On DE2 board, different components have different required running clock signals. For example, VGA controller is running at 25 MHz, but the CPU is running at 50 MHz.



Clock Signals for DE-series Board Peripherals attera_up_clocks

* Block Diagram	
	Configurations
Show signals	DE-Series Board: DE2 -
up_clocks_0	Optional Clocks
	SDRAM
clock clock sys_clk	Video (VGA and 5MP Digitial Camera)
clk_in_primary_resetsys_clk_reset	
	Audio
clock sdram_clk	Audio Clock Frequency: 12.288
clockvga_clk	
attera_up_clocks	

3. SRAM/SSRAM Controller

We need a pixel buffer working with Pixel Buffer DMA Controller. Check "Use as a pixel buffer for video out".

SRAM/SSRAM Controller attera_up_avalon_sram	
Block Diagram Show signals	Configurations
Pixel_Buffer	✓ Use as a pixel buffer for video out
clock_reset	
external_interface conduit	
avalon_sram_slaveavalonattera_up_avalon_sramavalon_sramattera_up_avalon_sramavalon_sramavalonavalon	

4. Pixel Buffer DMA Controller

The "Default Buffer Start Address" is the base address of the pixel buffer which is SRAM in our design. And the "Default Back Buffer Start Address" is the base address of SDRAM. The Pixel Buffer DMA Controller will read image pixels from memory. In memory, the image is stored as 16 bit RGB, and 320*240 resolutions.

	MegaC	Pixel Buffe	e r DM	A Controlle	r				
1	▼ BI	lock Diagram					Addressing Parameters		
L		Show signals					Addressing Mode:	X-Y 🗸	
L	Г		Pival B	uffer DMA Cor	troller	1	Default Buffer Start Address:	0x01080000	
L						Default Back Buffer Start Address:	0x00800000		
L	5	clock_reset	clock	avalon	avalon_pixel_dma_master		Frame Resolution		_
L	5	clock_reset_reset	reset	avalon_streaming	avalon_pixel_source		Width (# of pixels):	320	
L	8	avalon_control_slave	avalon				Height (# of lines):	240	
L				altera	_up_avalon_video_pixel_buffer_dma		Pixel Format		
							Color Space:	16-bit RGB 👻	

5. RGB Resampler

It converts 16 bit/pixel to 30 bit/pixel, but remain 320*240 resolution formats.

RGB Resampler attera_up_avalon_video_rgb_resampler		
Block Diagram	Parameters	
Show signals	Incoming Format:	16-bit RGB 🗸
	Outgoing Format:	30-bit RGB 👻
Pixel_RGB_Resampler	Alpha Value for Output:	1023
clock_reset clock avalon_streaming avalon_rgb_source clock_reset_reset reset avalon_rgb_sink avalon_streaming		
altera_up_avalon_video_rgb_resampler		

6. Scaler

It converts 320*240 to 640*480 resolutions by changing "Scaling Parameters".

1.5	
MeanCore	

Scaler

MegaCore'	altera_up_avalon_video_scaler	
• Block	Diagram	
Show	, einnale	Scaling Parameters
5104	signals	Width Scaling Factor:
		Height Scaling Factor:

					-
				Width Scaling Factor:	2 🗸
	Pixel_Scaler			Height Scaling Factor	2 🗸
			L	Incoming Frame R	esolution
		avalon scaler source	L		
-	clock avalon_streaming		L	Width (# of pixels):	320
			L		
_	reset		L	Height (# of lines):	240
			L		
_	avalon_streaming		L	- Divel Correct	
			L	* Pixel Format	
		altera_up_avalon_video_scaler	l	Color Bits:	10
			l	Color Planes:	3 🗸

7. Dual-Clock FIFO

clock_reset clock_reset_reset avalon_scaler_sink

The clock_stream_out is running at 25 MHz, and the clock_stream_in is running 50 MHz.

Block Diagram						
Show signals					Pixel Forma	at
					Color Bits:	10
	VGA_Dual	_Clock_FIFO		7	Color Planes:	3 🗸
clock stream in			avalon de buffer sourc			
	clock	avalon_streaming				
clock_stream_in_reset	reset					
clock_stream_out	clock					
clock_stream_out_reset	reset					
avalon_dc_buffer_sink						

8. VGA Controller

The clock_reset is running at 25 MHz. external_interface is conduit and is exported.

9. Clock Signals for DE-series Board Peripherals In our project, we have SDRAM and Video running at different clock frequencies. And sdram_clk is exported for SDRAM because SDRAM is running some delay compared with CPU.

Clock Signals for DE-series Board Person attera_up_clocks	eripherals
Block Diagram Show signals up_clocks_0 clk_in_primary clock clock clk_in_primary_reset reset clock clk_in_primary_reset reset clock clo	✓ Configurations DE-Series Board: DE2 ✓ ✓ ✓ Optional Clocks ✓ SDRAM ✓ Video (VGA and 5MP Digitial Camera) Audio Audio Audio
altera_up_clocks	

10. The final looking should be like the following:

$\circ - \circ - \circ - \circ - \circ - \circ$	F clk 0	Clock Source			
· · · · · · · · · · · · · · · · · · ·	Inios2 gsvs 0	Nios II Processor		up clocks 0 svs clk	0x0110 a800
· • • • • • • • • • • • • • • • • • • •	onchip memory2 0	On-Chip Memory (RAM or ROM)		up clocks 0 svs clk	0x0110 4000
· · · · · · · · · · · · · · · · · · ·	F svsid gsvs 0	System ID Peripheral		up clocks 0 sys clk	0x0110 b040
→ • • • • • • • • • • • • • • • • • • •	timer 0	Interval Timer		up clocks 0 svs clk	0x0110 b000
· • • • • • • • • • • • • • • • • • • •		JTAG UART		up clocks 0 sys clk	0x0110 b038
· · · · · · · · · · · · · · · · · · ·	F sdram 0	SDRAM Controller		up clocks 0 sys clk	0x0080 0000
→ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓	FI VGA Char Buffer With DMA	Character Buffer for VGA Display		up clocks 0 svs clk	multiple
• • • • • • • • •	Pixel Buffer	SRAM/SSRAM Controller		up clocks 0 sys clk	■ 0x0108 0000
	Pixel Buffer DMA Controller	Pixel Buffer DMA Controller		/_	_
♦ ♦ ♦ ♦ >	clock_reset	Clock Input	Double-click to export	up_clocks_0_sys_clk	
$ \downarrow \rightarrow$	clock_reset_reset	Reset Input	Double-click to export	[clock_reset]	
≻ <u></u>	avalon_pixel_dma_master	Avalon Memory Mapped Master	Double-click to export	[clock_reset]	
→	avalon_control_slave	Avalon Memory Mapped Slave	Double-click to export	[clock_reset]	0x0110 b020
	avalon_pixel_source	Avalon Streaming Source	Double-click to export	[clock_reset]	-
	Pixel_RGB_Resampler	RGB Resampler			
	clock_reset	Clock Input	Double-click to export	up_clocks_0_sys_clk	
	clock reset reset	Reset Input	Double-click to export	[clock reset]	
$ \bullet \circ \circ \circ + \rightarrow$	avalon_rgb_sink	Avalon Streaming Sink	Double-click to export	[clock_reset]	
	avalon_rgb_source	Avalon Streaming Source	Double-click to export	[clock_reset]	
	Pixel_Scaler	Scaler			
	clock_reset	Clock Input	Double-click to export	up_clocks_0_sys_clk	
$ \downarrow \downarrow \downarrow$	clock_reset_reset	Reset Input	Double-click to export	[clock_reset]	
$\diamond \bullet \diamond \bullet + + \rightarrow$	avalon_scaler_sink	Avalon Streaming Sink	Double-click to export	[clock_reset]	
	avalon_scaler_source	Avalon Streaming Source	Double-click to export	[clock_reset]	
	VGA_Dual_Clock_FIFO	Dual-Clock FIFO			
	clock_stream_in	Clock Input	Double-click to export	up_clocks_0_sys_clk	
$ \downarrow \rightarrow$	clock_stream_in_reset	Reset Input	Double-click to export	[clock_stream_in]	
	clock_stream_out	Clock Input	Double-click to export	up_clocks_0_vga_clk	
$ \downarrow \downarrow$	clock_stream_out_reset	Reset Input	Double-click to export	[clock_stream_out]	
\diamond \diamond \diamond \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow	avalon_dc_buffer_sink	Avalon Streaming Sink	Double-click to export	[clock_stream_in]	
	avalon_dc_buffer_source	Avalon Streaming Source	Double-click to export	[clock_stream_out]	
	UGA_Controller	VGA Controller			
	clock_reset	Clock Input	Double-click to export	up_clocks_0_vga_clk	
$ \downarrow \downarrow \downarrow$	clock_reset_reset	Reset Input	Double-click to export	[clock_reset]	
\diamond \diamond \bullet	avalon_vga_sink	Avalon Streaming Sink	Double-click to export	[clock_reset]	
	external_interface	Conduit	vga_controller_external_interface		



11. Toplevel file in VHDL

I exported VGA Controller external_interface, so in the toplevel file I added 8 more outputs:

```
-- VGA Controller

VGA_R : OUT STD_LOGIC_VECTOR (9 downto 0);

VGA_B : OUT STD_LOGIC_VECTOR (9 downto 0);

VGA_G : 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
```

Then port map those 8 outputs signals:

```
-- VGA Controller
```

```
vga_controller_external_interface_CLK => VGA_CLK, -- vga_contro
vga_controller_external_interface_HS => VGA_HS, --
vga_controller_external_interface_VS => VGA_VS, --
vga_controller_external_interface_BLANK => VGA_BLANK, --
vga_controller_external_interface_SYNC => VGA_SYNC, --
vga_controller_external_interface_R => VGA_R, --
vga_controller_external_interface_G => VGA_G, --
vga_controller_external_interface_B => VGA_B, --
```

- 12. Compile, Program Device, Nios II Software Build Tools for Eclipse
- 13. Create a new "Nios II Application and BSP from Template".
- 14. You need to include the file "altera_up_avalon_video_pixel_buffer_dma.h" to use some functions inside.
 - 1). Open the pixel buffer device

2). There are two buffers, primary buffer which is the SRAM and back buffer which is the SDRAM in our project. You can also assign your own buffer addresses. By choosing primary buffer, the data in primary buffer will be sent to VGA controller. By choosing back buffer, the data in back buffer will be sent to VGA controller. VGA Controller is responsible to deliver image pixels to VGA monitor to display. So make sure correct buffer data will be sent.

Check if the pixel buffer device has the primary buffer address. If not, then change to the primary buffer address. "p_addrs" is the primary buffer base address.

```
// Make sure the pixel buffer device has the primary buffer address
if (pixel_buf_dev->buffer_start_address != p_addrs)
{
    alt_up_pixel_buffer_dma_change_back_buffer_address(pixel_buf_dev, p_addrs);
    alt_up_pixel_buffer_dma_swap_buffers(pixel_buf_dev);
}
```

3). Clean the screen. The parameter "0" means clear the primary buffer, and "1" means clear back buffer.

```
// Clear the screen
alt_up_pixel_buffer_dma_clear_screen(pixel_buf_dev,0);
```

4). The function "alt_up_pixel_buffer_dma_draw(pixel_buf_dev, color, x, y);" is working on back buffer, if you want to use this function to display pixels on VGA monitor, then choosing the back buffer. The function

"alt_up_pixel_buffer_dma_swap_buffers(pixel_buf_dev);" is used to swap the buffer that will be sent to VGA controller.

```
// "alt_up_pixel_buffer_dma_draw" is working on back buffer, so we need sent
// back buffer to VGA controller
alt_up_pixel_buffer_dma_swap_buffers(pixel_buf_dev);
for (y = 0; y <480; y++)
{
    for (x = 0; x<640; x++)
    {
        // Draw pixels to VGA monitor
        error = alt_up_pixel_buffer_dma_draw(pixel_buf_dev, color, x, y);
        if (error == 1)
        {
            printf ("Error while drawing. The position is (%d,%d) \n", x,y);
        }
    }
}</pre>
```

5). In some functions, there is a parameter to select the back buffer or the primary buffer. You don't have to change it by yourself again. For example, "void alt_up_pixel_buffer_dma_draw_box(alt_up_pixel_buffer_dma_dev *pixel_buffer, int x0, int y0, int x1, int y1, int color, int backbuffer);" "voidalt_up_pixel_buffer_dma_draw_hline(alt_up_pixel_buffer_dma_dev *pixel_buffer, int x0, int x1, int y, int color, int backbuffer);"

The parameter "backbuffer" set to 1 is to select back buffer, and 0 is to select primary buffer.

Reference

Video.h file:///C:/altera/12.1sp1/ip/University_Program/Audio_Video/Video/doc/Video.pdf