HW-Modeling IP Core Documentation

T. Polzer, J. Lechner

March 11, 2010

Contents

1 Text Mode Video Controller Core

1.1 Description

The text mode video controller core establishes a simple character (ASCII) based interface to an external monitor. As video mode the standard VGA mode is used. On our target board 8-bit (=256) colors are supported.

The controller organizes the screen into 80 columns by 30 lines. All 256 ASCII characters are supported and decoded using the Western European Codepage (CP850)[1]. Figure 1 contains an example output generated by the video controller.

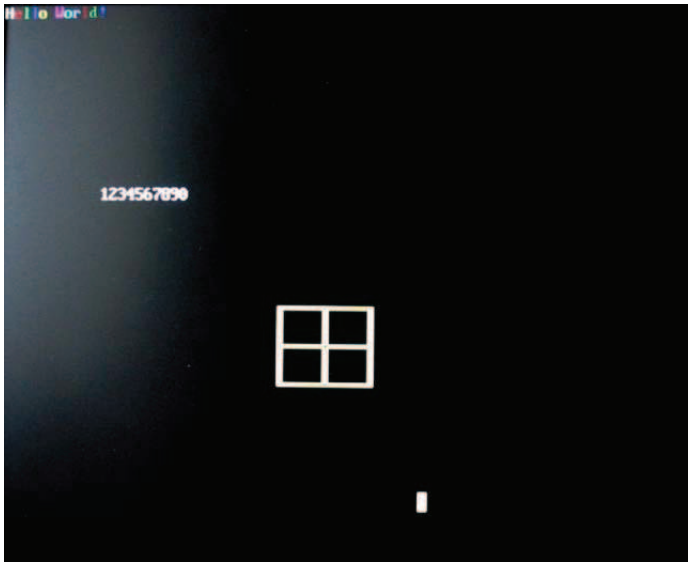

Figure 1: Example screen layout.

The controller implements a simple cursor interface. Characters are always written to the current cursor position. The newline and carrige return characters can be used for cursor control. Additionally the cursor can be freely positioned on the screen using special commands. The cursor supports three states (on, off or blinking) and can be displayed in any color.

Furthermore the background color is not fixed but can be freely chosen.

The integration of the controller into your design is simple, as the implemented controller interface is straight forward. Furthermore the user interface is independent of the target hardware (always 24-bit color and 8-bit ASCII data). All necessary adjustments (e.g., color adjustment from 24 to 8 bits) is done by the controller internally. In the following sections, the controller and its interfaces will be described in more detail.

1.2 Required VHDL files

The text mode video controller interface requires the following sourcefiles:

- console_sm.vhd
- console_sm_beh.vhd
- console_sm_sync.vhd
- console_sm_sync_beh.vhd
- font_pkg.vhd
- font_rom.vhd
- font_rom_beh.vhd
- interval.vhd
- interval_beh.vhd
- math_pkg.vhd
- textmode_vga.vhd
- textmode_vga_component_pkg.vhd
- textmode_vga_h_sm.vhd
- textmode_vga_h_sm_beh.vhd
- textmode_vga_pkg.vhd
- textmode_vga_platform_dependent_pkg.vhd
- textmode_vga_struct.vhd
- textmode_vga_v_sm.vhd
- textmode_vga_v_sm_beh.vhd
- video_memory.vhd
- video_memory_beh.vhd

1.3 Component Declaration

The declaration of the textmode video controller can be found in Listing 1, while the functionality of each generic is described in Table 1 and of each signal in Table 2. Multiple constants are used to specify the width of the signals. These constants are declared in the package *textmode_vga_pkg* and the package $textmode\_vga\_platform\_dependent\_pkg$, respectively.

```
component textmode_vga is
  generic
  (
    -- VGA clock frequency [Hz]
    VGA_CLK_FREQ : integer;
    -- Cursor blink interval [ms]
    BLINK_INTERVAL_MS : integer;
    -- Number of stages used for internal synchronizers
    SYNC_STAGES : integer
  );
  port
  (
    -- Interface to user logic
    sys_clk, sys_res_n : in std_logic;
    command : in std_logic_vector(COMMAND_SIZE - 1 downto 0);
    command_data : in std_logic_vector(3 * COLOR_SIZE +
        CHAR_SIZE - 1 downto 0);
    free : out std_logic;

    -- External VGA interface
    vga_clk, vga_res_n : in std_logic;
    vsync_n : out std_logic;
    hsync_n : out std_logic;
    r : out std_logic_vector(RED_BITS - 1 downto 0);
    g : out std_logic_vector(GREEN_BITS - 1 downto 0);
    b : out std_logic_vector(BLUE_BITS - 1 downto 0)
  );
end component textmode_vga;
```

Listing 1: Textmode video controller declaration

Generic name	Functionality
VGA_CLK_FREQ	Actual clock frequency of the *vga_clk* signal given in Hz. Normally the frequency should be 25.175 MHz. Unfortunately not all target boards support this frequency. Monitors are tolerant to variations in this frequency, therefore values in the range [25,25.175] MHz should work.
BLINK_INTERVAL_MS	Blink interval of the cursor given in milliseconds.
SYNC_STAGES	Number of synchronizer stages used for crossing the clock domain between *sys_clk* and *vga_clk*.

Table 1: Textmode video controller generics description

Signal name	Direction	Signal width	Functionality
sys_clk	in	1	System clock signal.
sys_res_n	in	1	System reset signal (low active, not synchronized).
command	in	COMMAND_SIZE (8)	Command which should be executed by the controller.
command_data	in	3 * COLOR_SIZE + CHAR_SIZE (32)	The data field of the command.
free	out	1	Status signal, if 0 the controller is busy, if 1 the controller is ready to receive a new command.
vga_clk	in	1	Clock signal used for establishing the VGA timing. The clock frequency should be in the range of [25,25.175] MHz.
vga_res_n	in	1	Reset signal of the VGA timing generator (low active, not synchronized).
vsync_n	out	1	Low active vertical synchronization signal (VGA interface).
hsync_n	out	1	Low active horizontal synchronization signal (VGA interface).
r	out	RED_BITS (platform dependent)	Red color output (VGA interface).
g	out	GREEN_BITS (platform dependent)	Green color output (VGA interface).
b	out	BLUE_BITS (platform dependent)	Blue color output (VGA interface).

Table 2: Textmode video controller signal description

```
textmode_vga_inst : textmode_vga
  generic map
  (
    VGA_CLK_FREQ => 25000000,
    BLINK_INTERVAL_MS => 1000,
    SYNC_STAGES => 2
  )
  port map
  (
    sys_clk => sys_clk,
    sys_res_n => sys_res_n,
    command => command,
    command_data => command_data,
    free => free,
    vga_clk => vga_clk,
    vga_res_n => vga_res_n,
    vsync_n => vsync_n,
    hsync_n => hsync_n,
    r => r,
    g => g,
    b => b
  );
```

Listing 2: Textmode video controller instantiation

1.4 Instantiation Template

To be able to instantiate the textmode video controller, the packages *textmode_vga_pkg*, *textmode_vga_component_declaration_pkg* and *textmode_vga_platform_dependent_pkg* must be included within your VHDL source. An instantiation template can be found in Listing 2.

For details on the type and size of the used signals, see Section 1.3.

1.5 Interface Protocol

The video controller is operated by a simple command interface. To execute a command, it must be assigned to the *command* port for exactly one cycle. At the same time, the corresponding data field must be applied to the *command_data* input (see Figure 2). The controller buffers the request internally and acknowledges its reception by setting the $free$ signal to low. As long as it remains low (it could stay low for multiple *sys_clk* cycles), no new command will be accepted by the controller and therefore the command $COMMAND\_NOP$ should be applied during this time frame. When the execution of the command is finished, the $free$ signal is set to high again. At this point a new command may be asserted.

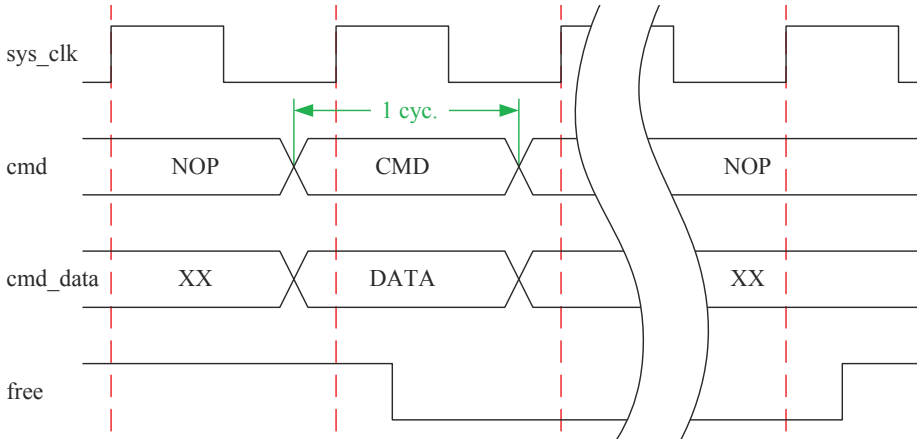

Figure 2: Textmode video controller timing

1.5.1 Command Description

In the previous section we have described how to execute a command on the textmode video controller. Table 3 summarizes the available commands.

Command	Data	Description
COMMAND_NOP	31-0: don't care	No operation. This command is assigned, if the video controller should not execute any operations.
COMMAND_SET_CHAR	7-0: ASCII code 15-8: Blue color component 23-16: Green color component 31-24: Red color component	Displays the given character in the given color at the current cursor position and afterwards advances the cursor position by one column. If the end of a line is reached, the cursor is positioned at the start of the next line. If the cursor was already at the last line, the whole screen is scrolled up by one line and the cursor is set to the beginning of the last line. If the newline character is set ($0A$h), additional to the normal operation, the cursor is advanced to the next line (incl. scrolling, if necessary). If the carrige return character ($0D$h) is set, the cursor is set to the beginning of the current line.
COMMAND_SET_BACKGROUND	7-0: Blue color component 15-8: Green color component 23-16: Red color component 31-24: don't care	Sets the background color.
COMMAND_SET_CURSOR_STATE	1-0: 00 - Off 1-0: 01 - On 1-0: 1x - Blink 31-2: don't care	Sets the state of the cursor (either on, off or blinking).
COMMAND_SET_CURSOR_COLOR	7-0: Blue color component 15-8: Green color component 23-16: Red color component 31-24: don't care	Sets the color of the cursor.
COMMAND_SET_CURSOR_COLUMN	6-0: X-Coordinate 31-7: don't care	Sets the cursor to column X. The line coordinate (Y) of the cursor is not changed.
COMMAND_SET_CURSOR_LINE	4-0: Y-Coordinate 31-5: don't care	Sets the cursor to line Y. The column coordinate (X) of the cursor is not changed.

Table 3: Textmode video controller commands

2 PS/2 Keyboard Controller Core

The PS/2 keyboard controller core can be used to easily interface with a standard PS/2 keyboard. The core is initialized automatically. On the reception of a new scancode (corresponding to set 2, see [3, 2]) the controller transmits it to the user logic without any further processing.

Therefore the handling of special keys (like $Ctrl$ or $Shift$) as well as the key pressed and key release handling are not performed by the controller. Instead the corresponding scancodes are directly sent to the user logic.

2.1 Required VHDL files

The PS/2 keyboard controller interface requires the following sourcefiles:

- ps2_keyboard_controller.vhd
- ps2_keyboard_controller_beh.vhd
- ps2_keyboard_controller_pkg.vhd
- ps2_transceiver.vhd
- ps2_transceiver_beh.vhd
- ps2_transceiver_pkg.vhd

2.2 Component Declaration

The declaration of the PS/2 keyboard controller can be found in Listing 3, while the functionality of each generic is described in Table 4 and of each signal in Table 5.

2.3 Instantiation Template

To be able to instantiate the keyboard controller, the package $ps2\_keyboard\_controller\_pkg$ must be included within your VHDL source. An instantiation template can be found in Listing 4.

For details on the type and size of the used signals, see Section 2.2.

Generic name	Functionality
CLK_FREQ	Frequency of the system clock given in Hz.
$SYNC\_STAGES$	Number of synchronizer stages used for synchronizing external signals.

Table 4: PS/2 keyboard controller generics description

```
component ps2_keyboard_controller is
  generic
  (
    -- System clock frequency [Hz]
    CLK_FREQ : integer;
    -- Number of stages used for synchronizers
    SYNC_STAGES : integer
  );
  port
  (
    -- Interface to user logic
    sys_clk, sys_res_n : in std_logic;
    new_data : out std_logic;
    data : out std_logic_vector(7 downto 0);

    -- External PS/2 interface
    ps2_clk, ps2_data : inout std_logic
  );
end component ps2_keyboard_controller;
```

Listing 3: PS/2 keyboard controller declaration

Signal name	Direction	Width	Functionality
sys_clk	in	1	System clock signal
sys_res_n	in	1	System reset signal (low active, not synchronized)
new_data	out	1	Signalizes the availability of a new scancode.
data	out	8	Scancode output
ps2_clk	bidirectional	1	Keyboard clock line
ps2_data	bidirectional	1	Keyboard data line

Table 5: PS/2 keyboard controller signal description

```
ps2_keyboard_controller_inst : ps2_keyboard_controller
  generic map
  (
    CLK_FREQ => 33330000,
    SYNC_STAGES => 2
  )
  port map
  (
    sys_clk => sys_clk,
    sys_res_n => sys_res_n,
    new_data => keyboard_new_data,
    data => keyboard_data,
    ps2_clk => ps2_keyboard_clk,
    ps2_data => ps2_keyboard_data
  );
```

Listing 4: PS/2 keyboard controller instantiation

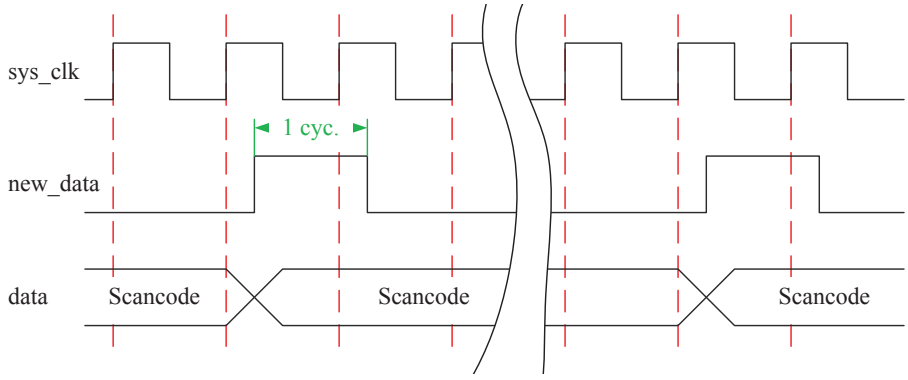

Figure 3: PS/2 keyboard controller timing

2.4 Interface Protocol

The keyboard controller utilizes a simple, straight forward interface. If a new scancode is available, it is assigned to the *data* port and the signal *new_data* is set high for exactly one clock cycle. The *data* port stays unchanged until the next scancode is received from the keyboard. For details on the protocol see Figure 3.

References

[1] Codepage 850. (e.g. Wikipedia: http://de.wikipedia.org/wiki/CP850).

[2] Description of scancodes including a list of scancodes for us keyboards. http://www.win.tue.nl/ aeb/linux/kbd/scancodes-10.html.

[3] Wikipedia scancode description. http://en.wikipedia.org/wiki/Scancode.