Lecture 4

Design of Embedded Hardware and Firmware

The LCD on MSE-Embedded Board
Design of a LCD Avalon Slave Core

TSM_EmbHardw
Mar. 16, 2016

Prof. A. Habegger
Bern University of Applied Sciences

Notes

Agenda

- ▶Intro
- ▶Avalon to Extern Bridge
- ▶Liquid Crystal Display
- ▶LCD Testing - ModelSim
- ▶An example Interface

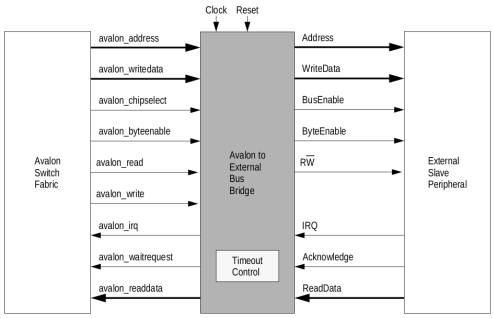
Notes

The Goals

- Exercise counter based on `Timer` and `simplePIO` finished
- More details to Avalon Slaves
- Let's get in touch with the LCD
  - Understand the interface
  - Implement an Avalon slave IP
  - Extend firmware to display a on the LCD
  - Display an image (optional)

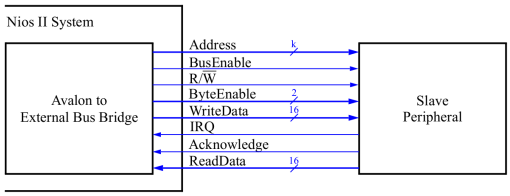
Notes

Overview



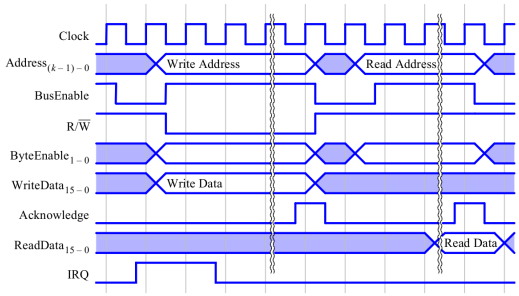
Notes

Signals



Notes

Communication



Notes

LCD Description

- Color display
- 2.4” Diagonal
- 240 x 320 pixels
- Up to 18 bit resolution (We will use 16)
- ILI9341 controller
- Controller allows several interfaces. We will use the 8080-series parallel interface with 16 bits.
- Available documentation:
  - LCD documentation (practical and very short)
  - ILI9341 controller documentation (very complete... too much)

Notes

8080 Series Parallel Interface

- The LCD interface has 7 signals:
  - DB (16 bits) for data bus (also used for commands).
  - Read control signal (RD_n)
  - Write control signal (WR_n)
  - Chip-Select (CS_n)
  - D/Cx control signal indicates whether in the bus we are sending data or commands (D_C_n)
  - LCD_Reset_n
  - **I**nterface **M**ode signal allowing to select either an interface on 8 or 16 bits (*IM*0)

Notes

Commands (or registers) [1]

- As set of commands allow to initialize power setting, adjust gamma settings, select resolution (18 bits, 16 bits,...), transfer mode, screen size, scrolling, etc.
- These commands are performed in two accesses: first a register address followed by the data to be written to the register.
- Addresses and data must be sent in 16 bits. We will thus use the 16 bits interface. It is easier and faster.

Notes

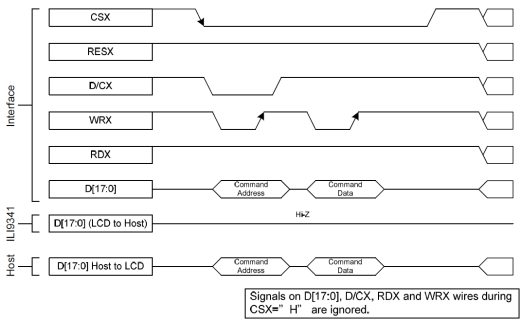
Commands (or registers) [2]

- You can either study every register to write your initialization or use the `init_LCD()` function available in Moodle.
- This initialization sets a pixel format `BGR` on 16 bits assigning 5 (B),6 (G), and 5 bits (R) respectively for each color
- For sending pixels you must initially write to the register 0x002C and then send 240x320 pixels to fill the screen. Each pixel sent in a 16-bit transfer.

Notes

About the Hardware Interface

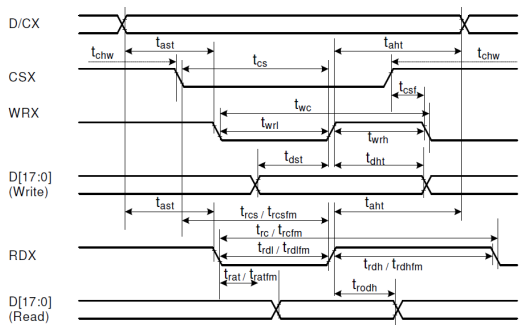
- Translate Avalon to LCD format accesses:



Notes

About the Controller Timing

- Respecting the following timing:



Notes

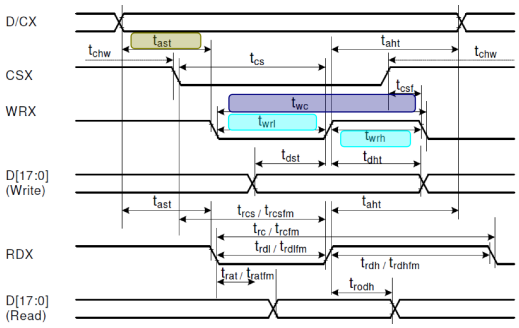
Details to the Timing

Signal	Symbol	Parameter	min	max	Unit	Description
DCX	tast	Address setup time	0	-	ns	
	taht	Address hold time (Write/Read)	0	-	ns	
CSX	tchw	CSX "H" pulse width	0	-	ns	
	tcs	Chip Select setup time (Write)	15	-	ns	
	trcs	Chip Select setup time (Read ID)	45	-	ns	
	trcsfm	Chip Select setup time (Read FM)	355	-	ns	
	tcsf	Chip Select Wait time (Write/Read)	10	-	ns	
WRX	twc	Write cycle	66	-	ns	
	twrh	Write Control pulse H duration	15	-	ns	
	twrl	Write Control pulse L duration	15	-	ns	
RDX (FM)	trcfm	Read Cycle (FM)	450	-	ns	
	trdhfm	Read Control H duration (FM)	90	-	ns	
	trdlfm	Read Control L duration (FM)	355	-	ns	
RDX (ID)	trc	Read cycle (ID)	160	-	ns	
	trdh	Read Control pulse H duration	90	-	ns	
	trdl	Read Control pulse L duration	45	-	ns	
D[17:0], D[17:10]&D[8:1], D[17:10], D[17:9]	tdst	Write data setup time	10	-	ns	For maximum CL=30pF For minimum CL=8pF
	tdht	Write data hold time	10	-	ns	
	trat	Read access time	-	40	ns	
	tratfm	Read access time	-	340	ns	
	trod	Read output disable time	20	80	ns	

Notes

About the Controller Timing

- Respecting the following timing:



Notes

The Design Steps

- Define a register model
  - How many "registers" must be available for a firmware-developer to properly communicate with the core
- Define an architecture with the following behavior
  - Block the avalon bus with waitrequest signal
  - Generate LCD control signals (LCD_WRn and LCD_D_Cn) respecting timing from the datasheet. They can be generated by a **state machine** or a **counter** followed by a **decoder**.
- Slow control signals like LCD_Reset_n, LCD_CS_n and *IM*0 can be generated by a GPIO (You got an idea by `simplePIO` how to design such ports).

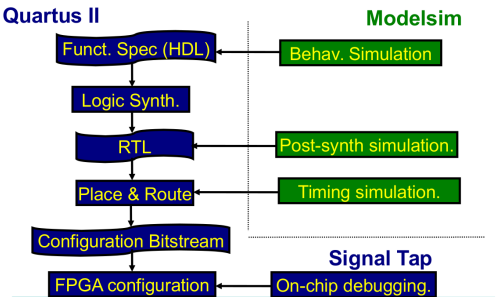
Notes

Simulation, Verification, and Debugging

- Complexity enhancement increases the probabilities of inserting bugs in your system.
- After describing a hardware architecture using `VHDL` we need to make sure it works.
- Several options are possible:
  - Trial and error... really an option?
  - Simulation or manual verification
  - Automatic verification
  - On-chip debugging

Notes

Test Results



Notes

Simulation, Verification, and Debugging

- ModelSim allows to perform a simple initial functional verification of a system (DUT) with a few simple steps:
  - Compile your VHDL code.
  - Write some input stimuli from the command line
  - Select the signals you want to observe
  - Run simulation
  - Analyze the resulting time-diagram

Notes

Manual Simulation

- Command lines can also be scripted as a `.do` file
- Script exemple:

File : `exampleDo.do`

```
restart
force clk 1 0, 0 10ns -repeat 20ns
force reset 1 0, 0 100ns
force avalon_address XX 0, 00 205ns, XX 225ns, 01 405ns,
XX 425ns
force avalon_write_data 16#XX 0, 16#2A 205ns, 16#XX 225ns,
16#CC 405ns, 16#XX 425ns
force avalon_wr 0 0, 1 205ns, 0 225ns, 1 405ns, 0 425ns
force avalon_cs 0 0, 1 208ns, 0 228ns, 1 405ns, 0 425ns
run 600ns
```

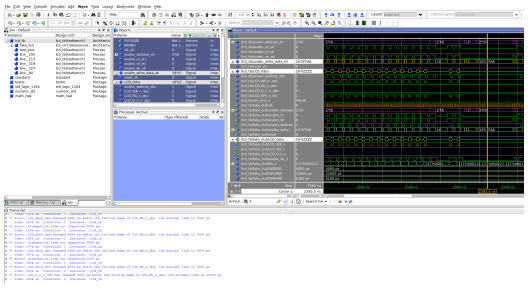

Notes

Manual vs Automated Simulation

- Manual
  - \+ quick and easy to setup and use
  - \- “human-in-the-loop” verification process
  - \- It can be very tedious for manually generating every test to be evaluated
  - \- It is up to you to verify behavior correctness
  - \- Complex designs may have many signals to analyze... and should be run for a long time
- A smarter approach is needed...
- A testbench can be used for generating input stimuli and verifing responses through output signals
- We provide a textbench for the LCD (how to develop testbenches is beyond the scope of this course

Notes

Test Results



Notes

En Example Entity

File : `ctrl_lcd-entity.vhdl`

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

ENTITY ctrl_lcd_avalonSlave IS
   PORT ( -- Here the internal interface is defined
          Clock                 : IN  std_logic;
          Reset                 : IN  std_logic;

          -- Here the avalon slave interface is defined
          slave_address         : IN  std_logic_vector( 1 DOWNTO 0 );
          slave_cs              : IN  std_logic;
          slave_we              : IN  std_logic;
          slave_rd              : IN  std_logic;
          slave_write_data      : IN  std_logic_vector(31 DOWNTO 0 );
          slave_read_data       : OUT std_logic_vector(31 DOWNTO 0 );
          slave_wait_request    : OUT std_logic;


          -- Here the external LCD-panel signals are defined
          ChipSelectBar         : OUT std_logic;
          DataCommandBar        : OUT std_logic;
          WriteBar              : OUT std_logic;
          ReadBar               : OUT std_logic;
          ResetBar              : OUT std_logic;
          IM0                   : OUT std_logic;
          DataBus               : INOUT std_logic_vector( 15 DOWNTO 0 ));
END ctrl_lcd_avalonSlave;
```


Notes

Adapt the Component

```
-- instantiation of a fake not synthesisable DUT
component LCD_ctrl_fake is
        generic ( FIFOSIZE : integer := 8;
                  ERRNO : integer := 0 );
        port (
                clk             : in    std_logic;
                avalon_address  : in    std_logic_vector(1 downto 0);
                avalon_cs       : in    std_logic;
                avalon_wr       : in    std_logic;
                avalon_waitreq  : out   std_logic;
                avalon_write_data : in  std_logic_vector(15 downto 0);
                reset           : in    std_logic;
                LCD_data        : inout std_logic_vector(15 downto 0);
                LCD_WR_n            : out   std_logic;
                LCD_RD_n            : out   std_logic;
                LCD_D_C_n           : out   std_logic
        );
end component;

-- Replace this component by your LCD_controller!!!
component LCD_ctrl is
        port (
                clk             : in    std_logic;
                avalon_address  : in    std_logic_vector(1 downto 0);
                avalon_cs       : in    std_logic;
                avalon_wr       : in    std_logic;
                avalon_waitreq  : out   std_logic;
                avalon_write_data : in  std_logic_vector(15 downto 0);
                reset           : in    std_logic;
                LCD_data        : inout std_logic_vector(15 downto 0);
                LCD_WR_n            : out   std_logic;
                LCD_RD_n            : out   std_logic;
                LCD_D_C_n           : out   std_logic
        );
end component;
```

Wave | LCD_tb.vhd

Notes

Adapt the Port-Map

```
        end process;

        process
        begin
                clk_sti <= '0';
                wait for CLK_PERIOD/2;
                clk_sti <= '1';
                wait for CLK_PERIOD/2;
                if (sim_end_s) then
                        wait;
                end if;
        end process;


----- THE DESIGN UNDER TEST- REPLACE THE FAKE DUT BY YOURS -----

--fake_dut: LCD_ctrl
fake_dut: LCD_ctrl_fake
        generic map( FIFOSIZE => FIFOSIZE,
                     ERRNO => ERRNO )

        port map(
                clk             => clk_sti,
                avalon_address  => avalon_address_sti,
                avalon_cs       => avalon_cs_sti,
                avalon_wr       => avalon_wr_sti,
                avalon_waitreq  => avalon_waitreq_obs,
                avalon_write_data => avalon_write_data_sti,
                reset           => reset_sti,
                LCD_data        => LCD_data,
                LCD_WR_n        => LCD_WR_n_obs,
                LCD_RD_n        => LCD_RD_n_obs,
                LCD_D_C_n       => LCD_D_C_n_obs
        );

end testbench;
```

Wave | LCD_tb.vhd

Notes