Advanced Logic Synthesis for Electronics
www.ALSE-FR.com

A.L.S.E Application Note
Using the LT24 / ILI9341

Introduction

I got questioned a few days ago by an EE teacher who purchased the $39 LT24 240 x 320 LCD display extension to FPGA kits and didn't understand how to use it. At ALSE, we have designed tens of interfaces for various LCD displays. Altera also offers a parametrizable Qsys module (CVO) that can accommodate most video outputs (parallel Pixel + Hsync + Vsync + Pixel Clock), and I suspected it should be straightforward to drive the LT24 so I took a look at the datasheets involved and decided to dedicate a few hours of my Week-End to this matter.

The LT24 LCD array is controlled by an autonomous controller: the ILITEK ILI9341, which embarks a graphic memory array and can autonomously display and refresh the LCD array. However, the ILI9341 does not embed an internal graphic processor and it must receive all the pixels values (through various possible interfaces either serial or parallel). This controller is very popular for small micro-controllers use, because of the display autonomous behavior and the ability to drive it serially with very few signals (starting with 2 !).
While reading the controller's documentation, I noticed the "RGB mode" which is the usual "Hsync/Vsync" interface and thought it would be the easiest mode to use.
Very unfortunately, Terasic hasn't wired HSync-Vsync-DotClock signals, so the direct RGB mode cannot be used, and the parallel "16-bits CPU interface" must be used (this mode is hardwired).

Note that no serial interface has been wired either, but there is nothing to regret here : a complete HE10-40 connector is used by the display anyway, and the 16 bits parallel interface retained is the fastest (and is also very simple to implement in an FPGA). This link can allow the transfer of up to 10 Mbytes/s, fast enough to redraw the entire screen in just 10 ms (ie up to 100 times per second).

So it seemed that, even if some code had to be written (to drive the parallel interface), it would be straightforward... In reality, starting from scratch and displaying patterns on the screen is still quite a challenge, as can be witnessed in several forums :

1) The controller is quite complex, it has 48 Level 1 commands, 36 Level 2 commands, and the documentation (245 pages Manual) contains no practical examples.
2) Setting up the controller correctly is very challenging. The documentation does not provide a "typical initialization sequence". And the LT24 requires non-default settings to work !
3) I don't think there is *anything* useful in the Terasic "examples" nor "documentation". I recommend to completely ignore them and use this ApNote and the accompanying code to build your own solution. The only useful data is the pin assignment to the HE-10 connector and the ILI9341.pdf Controller's Manual (make sure you don't use a preliminary version from the Internet).
4) I didn't find straight answers using Google. However, you'll see that C graphics libraries have been developed for this controller, and should be easy to port to your FPGA provided you decide to use a Nios II embedded processor, master the initialization sequence and design the parallel interface.

About A.L.S.E

As of end 2016, ALSE is a 23 years old company consisting in 7 Expert Engineers, and being a valued Intel-FPGA Platinum partner, Certified Design Center and Certified Training Center.

ALSE is located in Paris France and offers a wide range of Digital Design Services, IPs and Training Courses, serving customers in France, in Europe and all around the world.

- Design Services : turnkey designs, System and Board Level Design, Audits, Methodology assistance…
- Intellectual Property blocks (IPs) including PCIexpress, 1G and 10G Ethernet communication (hardware stacks), numerous high-performance Memory Controllers, Video & Image Compression & Decompression (JPEG, Wavelet), Video IP blocks, Security, Hardware GZIP, Industrial Network...
- Training Services : all the EE languages and Methodologies including Digital Design, VHDL, Verilog, SystemVerilog, SystemC, TLM2.0, UVM, PSL, Tcl/Tk etc…
- All the Intel-FPGA training courses.
- Embedded Systems Design Services and Trainings.

Please visit our web sites (in English and in French).

Principles of the LT24 Reference Design

To debug the screen initialization and investigate how the display can be driven, I have designed a simple HDL module based on a State Machine that parses a list of (easily modifiable) commands & data, and sends them to the LT24 with optional delays between commands (a requirement after some commands).

This list can be edited quickly and tried after a 30 seconds compilation and download.

This design can serve as a base for any application that does not use an embedded processor.

For applications that use a processor, the solution is very straightforward :

- Add a 20 bits PIO (output only) to drive the LT24 signals
- The required functions `WriteLcdCommand(uint8)` and `WriteLcdData(uint16)` are trivial to write.
- Use an existing ILI9341 C library (Atmel, Arduino, Adafruit etc)
- Use the Initialization sequence that you have validated using this ApNote.

Our Reference Design is based on an existing Max10 FPGA kit, which is not in the list of the supported boards by Terasic. It was not difficult to connect the LT24 to this board, but the pin assignment had to be entered carefully. The entire design fits in just 145 Logic Elements !

The design was tested by simulation with ModelSim Intel Edition before testing on the board.

The complete design was compiled using the free Quartus Lite v16.1.

Pin Assignment

To test our design, we used the small and cheap Max10 kit from Arrow : the **BeMicroMax10**.

The LT24 must be plugged on J3 (top HE10-40) :

See the pin assignment on the right : →

IMPORTANT: the ILI9341 must be driven using 2.5V or 3.3V (max) signals ! This is perfect for FPGA boards like ours but some micro-controllers still use +5V I/Os and may require require level shifters.

LT 24 on BeMicroMax10

LT24 signal	HE10	FPGA pin
ADC_PENIRQ_n	J3-1	B2
ADC_DOUT	J3-2	B1
ADC_BUSY	J3-3	C3
ADC_DIN	J3-4	A2
ADC_DCLK	J3-5	B3
ADC_CS_n	J3-39	K14
LT_RD_n	J3-13	B7
LT_WR_n	J3-14	A6
LT_RS	J3-15	A7
LT_CS_n	J3-28	B14
LT_DB(0)	J3-9	B5
LT_DB(1)	J3-8	A4
LT_DB(2)	J3-7	B4
LT_DB(3)	J3-6	A3
LT_DB(4)	J3-16	A8
LT_DB(5)	J3-17	A9
LT_DB(6)	J3-18	B8
LT_DB(7)	J3-19	B10
LT_DB(8)	J3-20	C9
LT_DB(9)	J3-21	H12
LT_DB(10)	J3-22	J11
LT_DB(11)	J3-23	E12
LT_DB(12)	J3-24	D13
LT_DB(13)	J3-25	D14
LT_DB(14)	J3-26	E13
LT_DB(15)	J3-27	A14
LT_RESET_n	J3-38	E15
LT_LCD_ON	J3-40	K15

Top Level Interface to LT24

We have added the LT24 signals to the FPGA top level:

```
    -- LT24 graphic Controller
    LT_DB          : inout std_logic_vector(15 downto 0); -- we use it as OUT only
    LT_WR_n        : out   std_logic;  -- aka WRX
    LT_RD_n        : out   std_logic;  -- aka RDX
    LT_RS          : out   std_logic;  -- must be D/CX
    LT_CS_n        : out   std_logic;  -- aka CSX
    LT_RESET_n     : out   std_logic;  -- aka RESX
    LT_LCD_ON      : out   std_logic   -- Transistor to drive LCD lighting (1 = on)
```

Note that we drive one LED with the ADC_PEN_IRQ output from the TouchScreen logic.

Physical Interface

The ILI9341is very versatile and even parallel interfaces can be 8-9-16 or 18 bits wide ! The LT24 hardwired selection imposes a ***16 bits 8080 type I*** interface. Some configuration registers also need to be correctly programmed.

A State Machine (FSM) handles the communication with the ILI9341 Controller.

The ILI9341 Manual explains the required timing for this interface. We can note that raising (inactivating) the Chip Select signal is NOT necessary. We take it low at the beginning of the session and raise it when the FSM has finished.

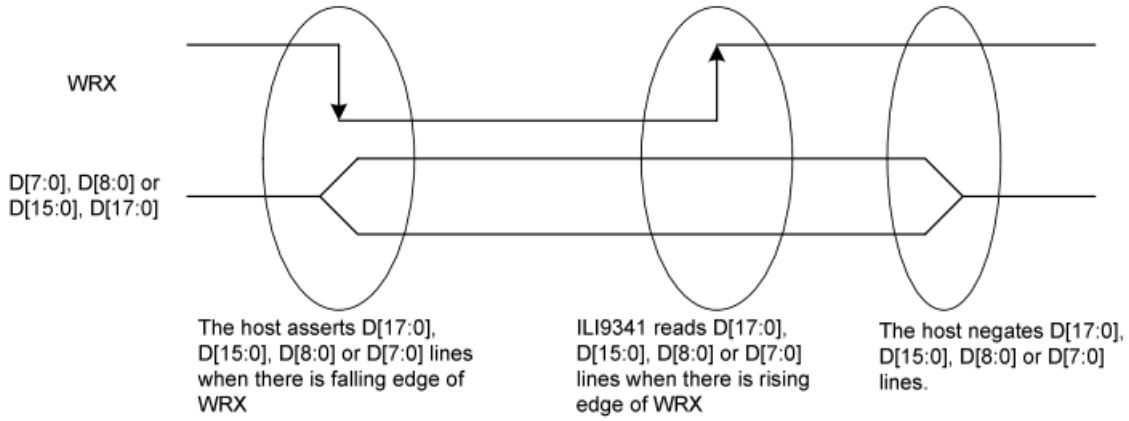
To Send a Command or Data :

- Set LT_RS (aka D/CX) to '0' (for command) or '1' (for Data)
  Set LT_DB to the Command (8 LSBs) or Data (16 bits) value
- Set LT_WR_n = '0' (active)
- Wait Nsetup clock cycles
- Set LT_WR_n = '1' (going active triggers the actual writing)
- Wait Nhold clock cycles

It's done through 3 states in the FSM.

The principle is documented in the Controller's Manual ;

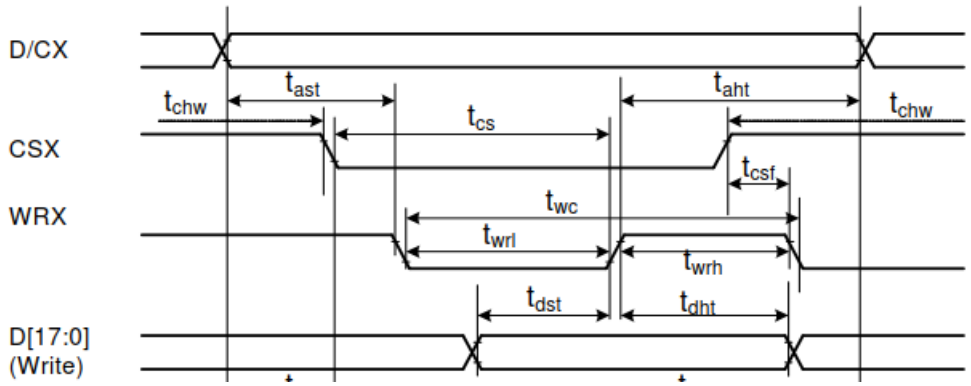
The following figure shows a write cycle for the 8080- I MCU interface.



And at the end of the manual we find the complete

18.3 AC Characteristics

18.3.1 Display Parallel 18/16/9/8-bit Interface Timing Characteristics (8080- I system)



The timing values can be found under the waveform on the same page.

Init Sequence

The pattern is correctly displayed after the following (*minimal*) initialization :

- Hardware Reset (active for 15 ms) followed by 300 ms wait.
- Exit Sleep command (0x11) followed by 800 ms delay (to see the effect of the next command)
- Display ON (0x29) followed by 1 ms wait
- Set COLMOD (0x3A) with parameter 0x0055 (16 bits per pixel, one data per pixel)
- SET Memory Access Control (0x36) with BGR filter !
- Enable 3G (0xF2) with 0x0000 parameter (disable 3 Gamma)

More settings can be experimented with, once the image is displayed as assured by the above sequence. You can review and test the (much longer) initialization sequences found in various libraries.

It is critical to set up the mode where one Pixel is stored by one unique 16-bits write since the screen and the graphic memory are actually 18 bits (RGB 6-6-6) and some modes require several data for one pixel.

Pattern

To exercise an important feature, we set up the Starting Column Address and Page address. And we verify that the pattern starts inside the screen and not at 0,0.

Then we issue the Memory Write Command (0x2C) followed by the pixels.

The pattern itself is 8 White pixels, 8 Black, 8 Red, 8 Green and 8 Blue, repeated 5 times (200 pixels total).

It is terminated by a NoOp command (no effect).

The (upside-down) picture above shows the displayed pattern (right to left) matching the vectors.

Files

The most interesting file of the project is the top-level **top.vhd** which contains all the logic except the Frequency divider that creates the 1 millisecond tick used for the various delays.
The important parts of this file are reproduced below.

The table of vectors is easy to edit :

```
  subtype Cmd_t is std_logic_vector (28 downto 0);
  -- Dly_ms(12) & C/D & Data(16) - 29 bits, could be a record
  type Commands_t is array (natural range <>) of Cmd_t;

  constant Commands_c : Commands_t := (

--  delay_ms D/Cd_n  DB
    x"001" & '0' & x"0011",  -- Exit Sleep
--  x"100" & '0' & x"0001",  -- Software reset
    x"800" & '0' & x"0029",  -- Display ON

    x"001" & '0' & x"003A",  --  Set COLMOD
    x"000" & '1' & x"0055",  --  non-default 16 bits

    x"000" & '0' & x"0036",  --  Memory Access Control
    x"000" & '1' & x"0008",  --  non-default BGR filter !

    x"000" & '0' & x"00F2",  --  Enable 3G
    x"000" & '1' & x"0000",  --  non-default = disable 3 gamma
```

```
    x"000" & '0' & x"002A",  -- Set Column address
    x"000" & '1' & x"0000",  --
    x"000" & '1' & x"0010",  -- 16

    x"000" & '0' & x"002B",  -- Set Page address
    x"000" & '1' & x"0000",  --
    x"000" & '1' & x"000A",  -- 10

    x"000" & '0' & x"002C",  -- Memory Write

    x"000" & '1' & x"FFFF", -- WHITE
    x"000" & '1' & x"FFFF", -- WHITE
    x"000" & '1' & x"FFFF", -- WHITE
    x"000" & '1' & x"FFFF", -- WHITE
   ETC ETC ETC
```

The State Machine reproduced below holds the parsing of the table above and the communication logic.

```
-- ---------------------------
-- ILI9341 interface
-- ---------------------------
-- CSX can be kept low
-- Write cycle > 66ns   = 100 ns (5c = 2 + 3) can work
-- Tdst data setup > 10 ns
-- Tdht data hold > 10 ns
-- /!\ Data read (not used here) is SLOW (500 ns)

LT_RD_n  <= '1'; -- we don't read

process (Clk,Rst)
begin
  if Rst='1' then
    State <= Boot;
    LT_DBi <= (others=>'0');
    LT_WR_n <= '1';
    LT_CS_n <= '1';
    LT_RS   <= '0';
    LT_RESET_n <= '0';
    LT_LCD_ON  <= '0';
    Index <= 0;
    Cntr  <= 15;
    Cycle <= 0;
    Cmd   <= (others=>'0');

  elsif rising_edge(Clk) then

    Cmd <= Commands(Index);

    case State is

      when Boot =>
        LT_RESET_n <= '0';
        Index <= 0;
        if Tick1ms='1' then
          if Cntr=0 then
            LT_RESET_n <= '1';
            Cntr <= 300;
            State <= Idle;
          else
            Cntr <= Cntr - 1;
          end if;
        end if;

      when Idle =>
        if Cntr=0 then
          LT_LCD_ON <= '1'; -- Light ON
          LT_CS_n <= '0';
          State <= GetCmd;
        elsif Tick1ms='1' then
          Cntr <= Cntr - 1;
        end if;

      when GetCmd =>
        Cntr   <= to_integer(unsigned(Cmd(Cmd_t'high downto Cmd'high-11)));
        LT_DBi <= Cmd(LT_DBi'range);
        LT_RS  <= Cmd(LT_DBi'length);
        State  <= Dly;
```

```
      when Dly =>
        if Cntr = 0 then
          Cycle <= Nsetup-1;
          LT_WR_n <= '0';
          State <= Wr1;
        elsif Tick1ms='1' then
          Cntr <= Cntr-1;
        end if;

      when Wr1 =>  -- Setup
        if Cycle = 0 then
          Cycle <= Nhold-1;
          LT_WR_n <= '1'; -- rising edge (trig write)
          State <= Wr2;
        else
          Cycle <= Cycle-1;
        end if;

      when Wr2 =>  -- Hold
        if Cycle = 0 then
          State <= Done;
        else
          Cycle <= Cycle-1;
        end if;

      when Done =>
        if Index=Commands'length-1 then  -- last command has been sent
          LT_CS_n <= '1'; -- de-select the interface
          State <= Done;  -- deadlock (default)
        else
          Index <= Index + 1; -- fecth next vector
          State <= Done1;
        end if;

      when Done1 =>  -- compensate double pipeline
          State <= GetCmd;

    end case;

  end if;
end process;
```

As we can see, interfacing to the Controller requires less than one page of code !

Conclusion

As you should now realize, the Terasic LT24 may not be the simplest choice in terms of LCD display : other displays driven in RGB (Hsync/Vsync) mode are typically more widely used.

However, this kind of display can be found at low prices (< $10 per unit) and sending a frame buffer from an FPGA to the LT24 over the 16-bits MCU interface is not very difficult (as demonstrated here) and can achieve high enough frame rates (thus allowing video). If you want to go in this direction, you can easily modify the FSM to suit your application needs.

If you plan to display Texts and 2D graphics (GUI), ALSE has developed a Sprite-based Hardware Graphic Engine (“Gradien”) that could also be used for this purpose, but a processor and a graphics library is usually the recommended solution.

I hope you enjoyed this Reference Design and it will help you use cheap LCDs with your FPGA applications.

Bertrand Cuzeau – A.L.S.E
Tel +33 (0)1 84 16 32 32 – info at alse-fr dot com