Nicolas Cavedon & Augustin Mohr
14.12.2020

Embedded Systems – Lab 3: LCD Conceptual Design

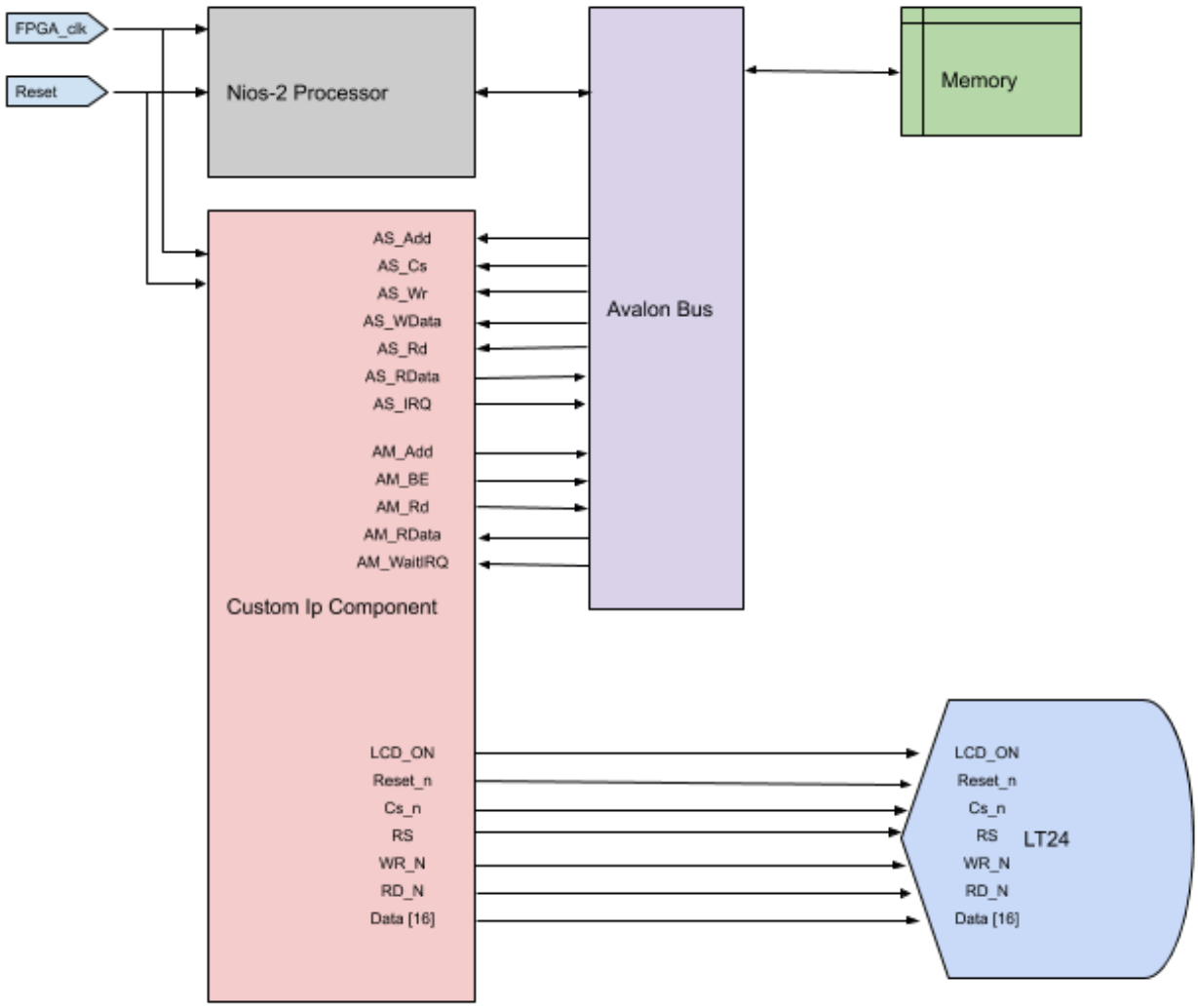
Introduction

For the course Embedded Systems, we will design a custom controller for the LT24 LCD using the Ili9341 driver. This report concerns the theoretical approach and researches done before starting the implementation of the controller.

Full System Block Diagram

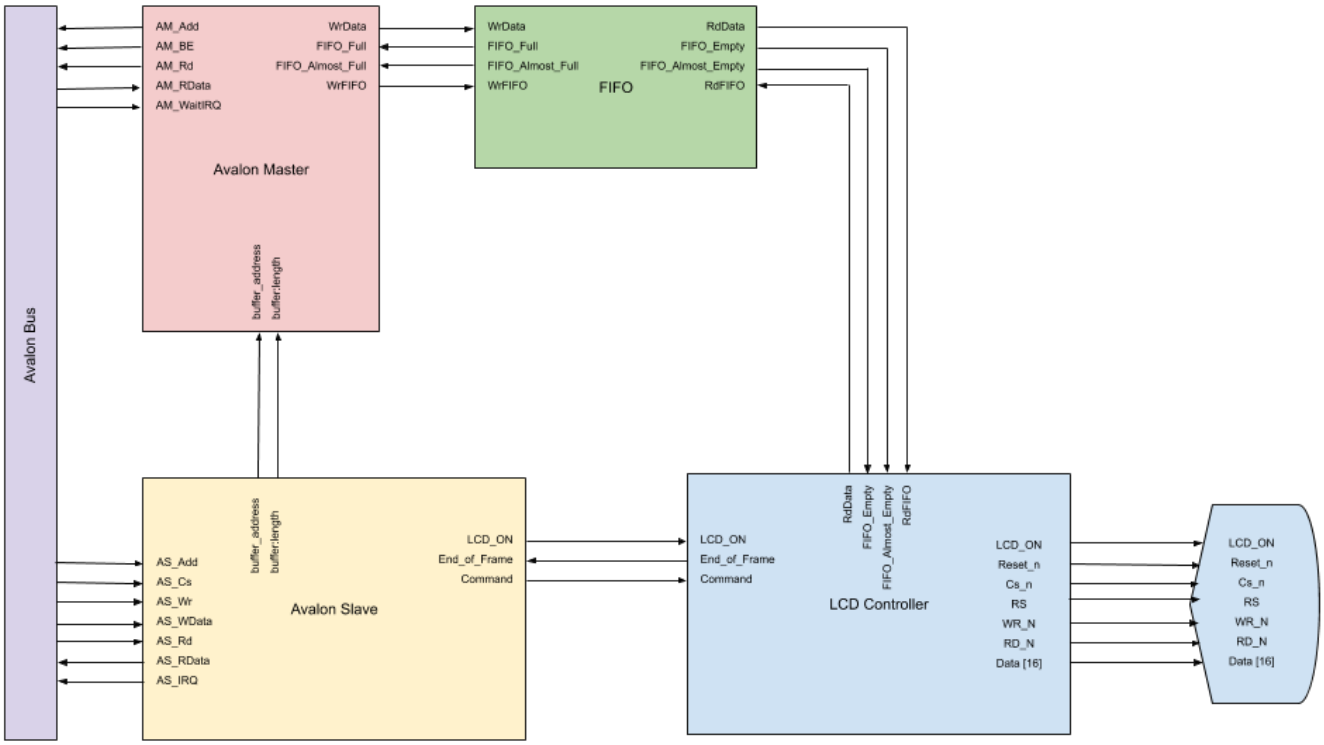
Here is the full system block diagram:

The frame to display will be stored in a memory buffer. Our controller will have an Avalon master module designed to fetch the image from the memory, an Avalon slave module to get information from the Processor, and an external conduit controlling the LT24 peripheral.



Custom Ip Block Diagram

Here is the diagram of our custom controller and its different internal parts.



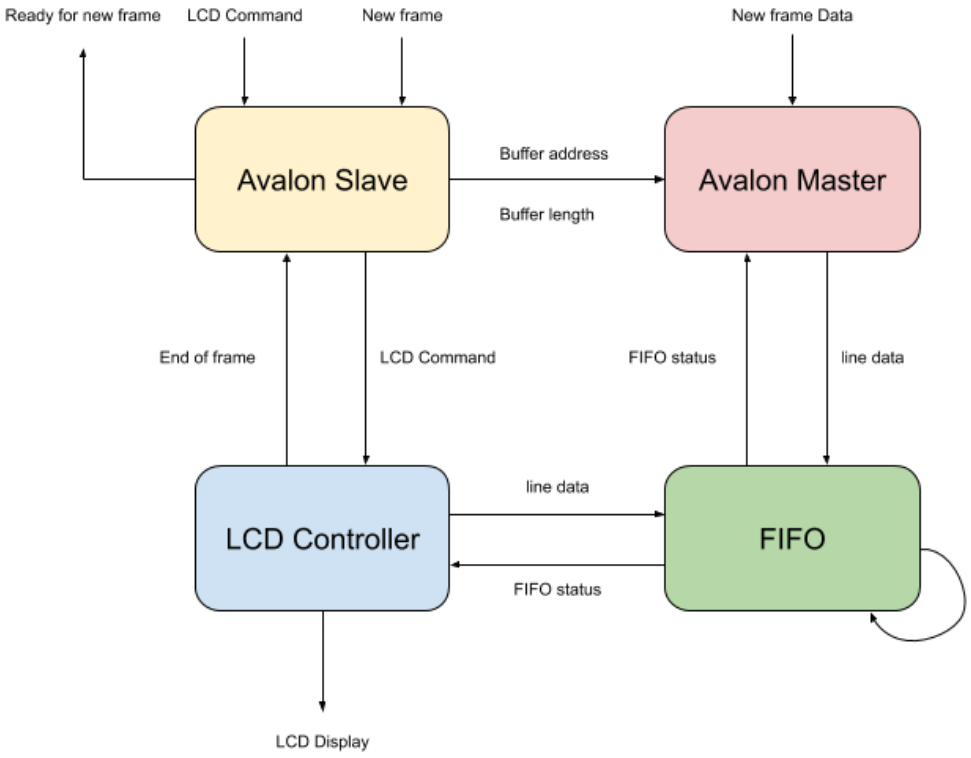
The Avalon slave will communicate with the processor via the Avalon bus. It will have access to all of our internal registers, in both read and write mode. It will also provide commands to the LCD Controller, as well as the buffer address and sizes for the memory to the Avalon Master. The Avalon master communicates with the memory via that same bus. It will configure the read access to read the data from the memory. It transfers the data to a FIFO when it is available, and the LCD Controller will read that data to transfer it to the LCD with the right timing.

The control signals to operate the FIFO are WrFIFO and RdFIFO, which are respectively sent from the Avalon Master and the LCD Controller. The FIFO also outputs lines of data describing its state to the revelant modules.

The Controller will communicate with the LT24 and will be able to give commands provided by the Avalon slave.

The AM_WaitRQ is there because we do not know when the data is available for the Avalon Master. With this line, we are able to know when the Avalon Master is able to fetch the data from the memory.

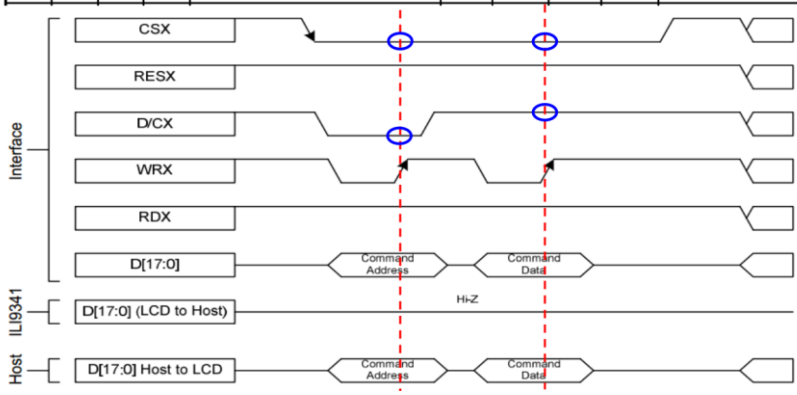
Custom Ip Finite State Machine



The protocol to access the controller that is on the LT is the following:

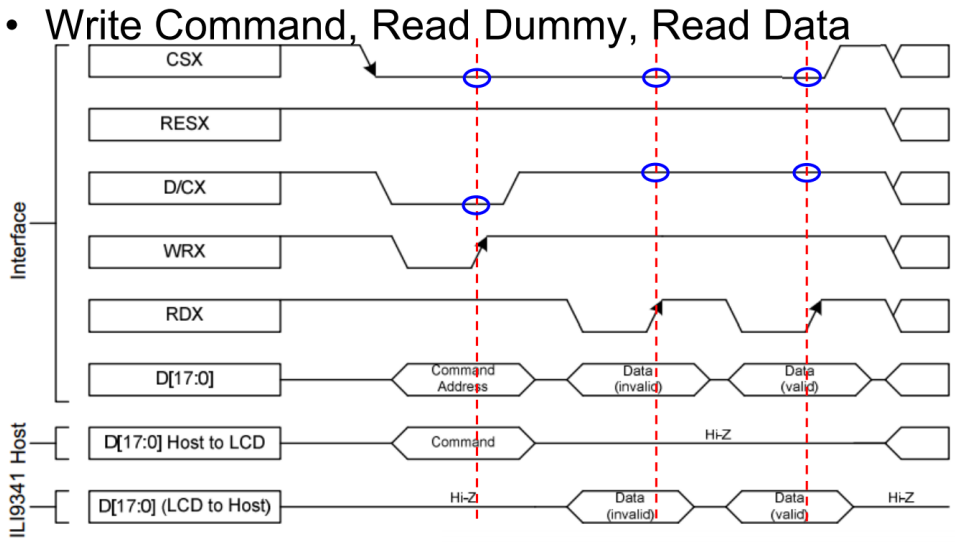
In the first phase, we firstly need to send a command to the chip. It will contain the command address, which will select one of the internal registers. The second part of that command will be the command data which we will send to that address. We first start by activating the CS line (Chip Select), which is defined as active low. Then, we will also lower the D/C line (Data Control) to '0'. While that line is activated, the rising edge of the WR line will allow for the command address on the data bus to be caught by the processor. For the next cycle, we will set the D/C line to '1' so that the next rising edge of the WR line allows the command data to be caught. The timing diagram can be seen bellow:

IM3	IM2	IM1	IM0	MCU-Interface Mode	CSX	WRX	RDX	D/CX	Function
0	0	0	1	8080 MCU 16-bit bus interface I	"L"	⎍	"H"	"L"	Write command code.
					"L"	"H"	⎍	"H"	Read internal status.
					"L"	⎍	"H"	"H"	Write parameter or display data.
					"L"	"H"	⎍	"H"	Reads parameter or display data.



When we want to make a write access, the following process is applied:

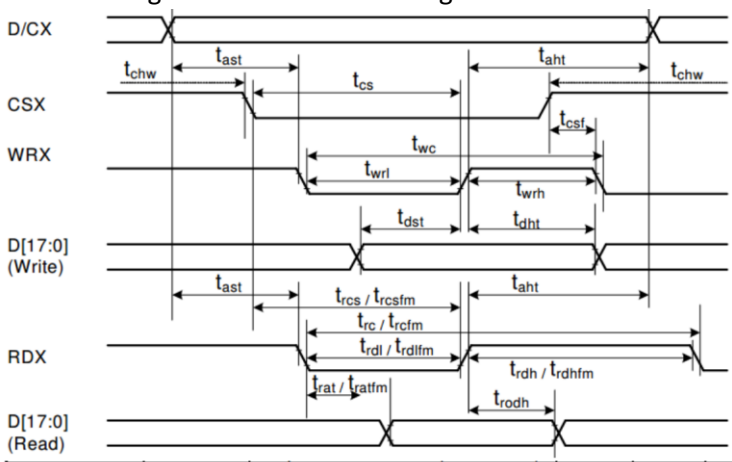
We first need to activate the CS line as before (by pulling it to '0'), and we need to specify that we want to make a 'write' by specifying it with the correct command. This is done as explained previously using the rising edge of the WR line and by specifying the command address and the corresponding command. For this, the D/R line needs to be pulled to '0'. Next, we will pull the RD line to make the data be caught by the controller. Since the time between the previous action and this one is short, the data will be invalid, and the RD line needs to be pulled to '1' a second time for the data to be valid. This process can be seen on the diagram bellow:



The very first command we want to send to our module is the command that rotates the display 90 degrees. This makes it so that the display is in landscape mode. When we send a new line to the module, we will need to send 320 pixels instead of 240.

Each time we want to send a line of 320 pixels to the module, we first need to send a line of command to the module. It will be 8 bits long, and since the data bus is 16 bits wide, will be sent in the lower half (D0 – D7). Then, for the next 320 cycles, we will send a pixel for each cycle, in RGB format. In this format, the 5 lower bits (0 – 4) will be dedicated to the colour blue, the 6 next bits (5 - 10) will be dedicated to green, and the last 5 bits (11 - 15) will be dedicated to red.

Since the processes described above are asynchronous, we will need to define a strict timing to be followed. A few of those timings can be seen in the diagram bellow:



WRX	twc	Write cycle	66	-	ns
	twrh	Write Control pulse H duration	15	-	ns
	twrl	Write Control pulse L duration	15	-	ns

Custom Ip Register Map

From the software point of view, the programmer will be able to write in the registers of the custom component. This way, the programmer will send first the command he wants to use, and the data he wants for this command.

As the port addresses as concerned, we will be using the same structure as described by terasic in the LT24 user manual:

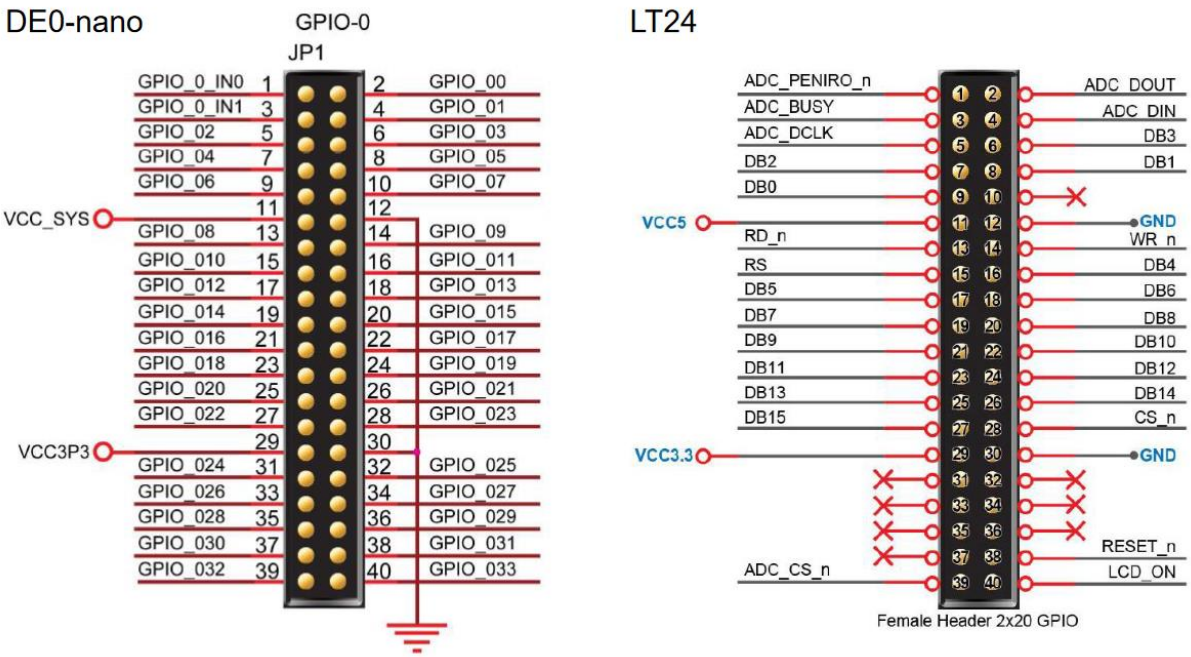
Byte offset	Register Name	Description
0x00	Control Port	Write control command to the LCD driver
0x01	Data Port	Write data to the LCD driver

This way, the programmer can use "*IOWR(LT24_BASE, 0x00, CommandValue)*" to write a command, and *"IOWR(LT24_BASE, 0x01, DataValue)"* to write data.

The DataValue and CommandValue will be directly coming from the datasheet of the Ili9341.

Top Level Block Diagram

For this project, we will use the pin out displayed on the figure below:



Memory Organization

The frame to display is stored in a memory buffer. Our custom Ip component receives the start address and the length of the frame. The pixels of the frame are stored from right to left, and then top to bottom of the frame. One pixel is coded in 16-bit RGB in the 8080 system.

Conclusion

In conclusion, this report should allow us to structure our work for the actual implementation of this controller. Although every detail is not set here, the overall structure of each part is described in this document and can serve as a guide for the rest of the work on this project.