LCD Interface Conceptual Design

1. About the project

The aim of the project is to develop an LCD controller, which displays images stored in the SDRAM. As LCD we use the LT24 module by Terasic, which includes the ILI9341 LCD driver and a 240x320 display. Incorporating a DMA, our IP read the data on the SDRAM through the Avalon Bus, then directly communicates with the LCD driver following an 8080 I protocol. The image data is provided by a camera module, following the format we present in this document. The design has been developed for the DE0-Nano-Soc Kit.

2. Feature Summary

- Two modes available:
  - video
  - photo
- Capability of sending instructions to the ILI9341 by the CPU
- Minimum involvement of the CPU in data routines
- Reconfigurable frame buffer address and burst count

3. System Overview

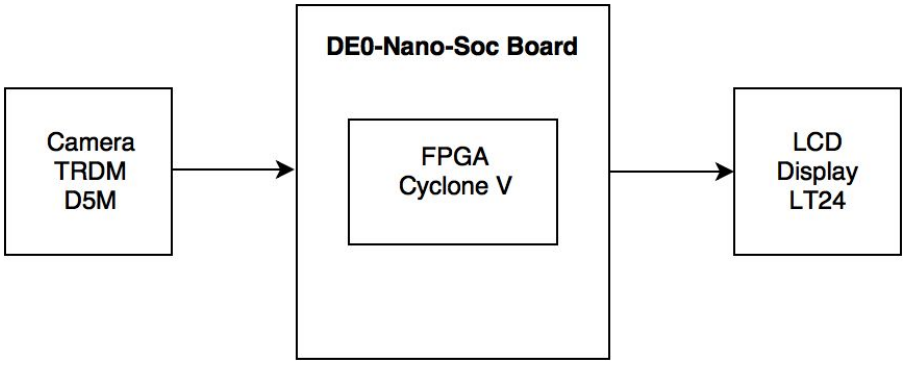


Fig 3.0

The full system block diagram is presented in Fig. 3.0. The camera stores pictures inside a RAM memory connected to the FPGA. The stored data is accessible by the LCD that will display the pictures.

This document focuses on the design of the LCD controller, which allows the transfer of images from memory to LCD.

3.1. General architecture

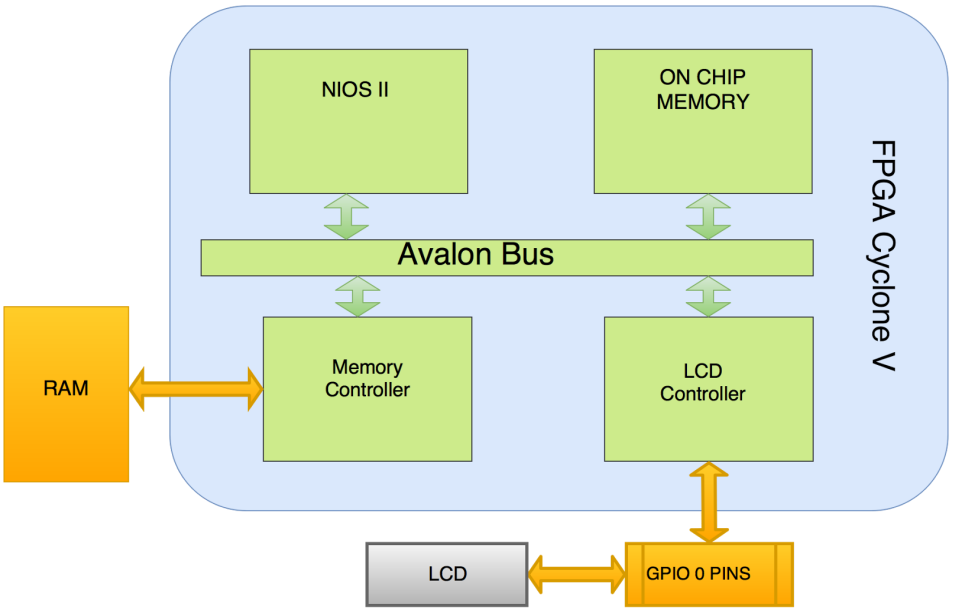


Fig. 3.1

3.2. Custom IP Diagram

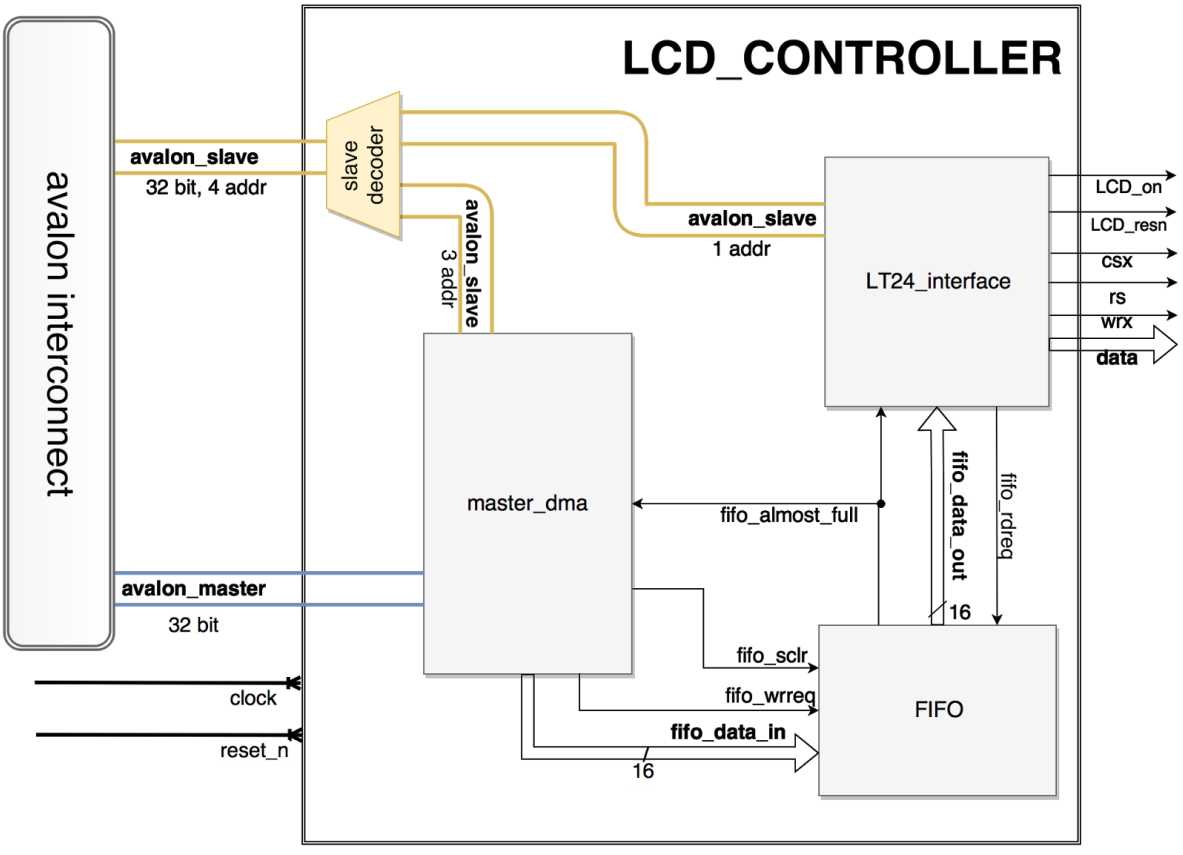


Fig. 3.2

The LCD_controller is the IP developed for the project. It includes a master and a slave avalon interface. The master acquires data from the SDRAM through the bus, while the slave is needed for the IP runtime configuration. In this proposed block diagram the slave interface is divided between the two submodules. This implementation allows to increase the modularity of the design, letting the development proceed in a parallel way. In addition the two slaves logic is very different, because in the dma it only writes and reads configuration registers, while in the LT24_interface it includes a pass-through command functionality. The two units are explained in more details in the following sections.

3.3. Custom IP Register Map

Base address: to be defined			**Len**: 32b
NAME	**BIT WIDTH**	**MODE**	**SYSTEM ADDRESS OFFSET**
Image start address	32	r/w	0
Image 2 start address	32	r/w	4
Transfer length	21	r/w	8
Burst number	8	r/w	8
Start transfer	1	r/w	8
Buffer selection	1	r/w	8
Restart	1	r/w	8
Command/data	16	w	12
DCX (RS)	1	w	12
continue	1	w	12
LCD_on	1	w	12
LCD_resn	1	w	12
regs_req	1	w	12

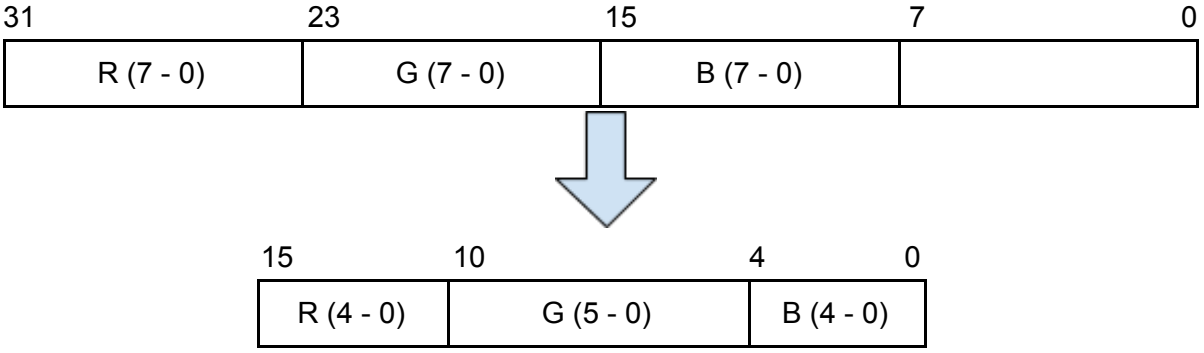
Table 3.1

In *Table 3.1*, we present the register map of our IP. Since the organization of bits inside the single registers only affects the C code, it is not defined here and it will be decided during the implementation.

- ***Image start address*** provides the initial address of the first buffer in memory where the picture is stored.
- ***Image 2 start address*** provides the initial address of the second buffer in memory where the picture is stored.
- ***Transfer length*** provides the number of burst transfer we need in order to load the full image.
- ***Burst number*** : number of burst for each transfer.
- ***Start transfer*** : start the transfer of data from memory to LCD.
- ***Buffer selection*** : select between image or image2 start address.
- ***Command/data*** : not a real register, but a port that can be accessed to send command/data to the LCD. It contains the 16 bit of data in the least significant bits and other two bits, one for the DCX [1](RS) value
- ***DCX*** (RS) : must be low when we send a command and high when we send data
- **continue**: high when the current command/data expect a following instruction
- ***LCD_on***: registered output
- ***LCD_resn***: registered output
- ***regs_req***: not a register, but a control bit to differentiate between commands for the LT24 and internal register writings from the CPU

3.4. Data Format

Data stored in the SDRAM is organized in frames. In the video mode two frame buffers are used, while in the photo mode only one. Each frame contains 320*240 pixels. Each pixel is 24 bits of data[2] (RGB: [8,8,8]) and is stored in one 32 bits location (8 bits are unused). Details of this data format will be further discussed with the camera designers.



[1] It is a bit required from the ILI9341 which says if the instruction sent is a command (0) or data (1).
[2] Reported by the camera group.

Count	0	1	2	3	…	238	239	240
D/CX	0	1	1	1	…	1	1	1
D15		0R4	1R4	2R4	…	237R4	238R4	239R4
D14		0R3	1R3	2R3	…	237R3	238R3	239R3
D13		0R2	1R2	2R2	…	237R2	238R2	239R2
D12		0R1	1R1	2R1	…	237R1	238R1	239R1
D11		0R0	1R0	2R0	…	237R0	238R0	239R0
D10		0G5	1G5	2G5	…	237G5	238G5	239G5
D9	**WRITE**	0G4	1G4	2G4	…	237G4	238G4	239G4
D8	**CMD**	0G3	1G3	2G3	…	237G3	238G3	239G3
D7	C7	0G2	1G2	2G2	…	237G2	238G2	239G2
D6	C6	0G1	1G1	2G1	…	237G1	238G1	239G1
D5	C5	0G0	1G0	2G0	…	237G0	238G0	239G0
D4	C4	0B4	1B4	2B4	…	237B4	238B4	239B4
D3	C3	0B3	1B3	2B3	…	237B3	238B3	239B3
D2	C2	0B2	1B2	2B2	…	237B2	238B2	239B2
D1	C1	0B1	1B1	2B1	…	237B1	238B1	239B1
D0	C0	0B0	1B0	2B0	…	237B0	238B0	239B0

Fig. 3.4 - Data format conversion from the SDRAM to the LT24

Organization of the data in row-wise or column-wise formats can be later decided, thanks to the ability of the ILI9341 LCD controller to easily flip and reflect frames.

Since the LCD can only accept a 16 bit wide RGB data [5,6,5], in the master_dma unit we select 16 bits (most significant bits for each color) among the 32 of the SDRAM location we loaded and we send them to the FIFO. Thus the FIFO is organized with a 16 bit data width (1 pixel per location), to fit the LCD data format.

3.5. Functional Flow

We plan to implement two different modes:

- **photo mode:**
  The Camera and the LCD are initialized. The CPU sends a command to the camera to take a picture. When the image is completely written in the SDRAM, the camera sends an interrupt to the CPU. In the interrupt routine the DMA is started, the image is loaded in the FIFO and automatically sent to the LCD when the FIFO is almost full.
- **video mode:**
  Same routine as before but this time we have two frame buffers in the SDRAM. The frame buffers should be switched such that when the DMA is reading from one of them, the camera is writing to the other one. The addresses can be specified each time by the CPU, which in this case takes care of the switching, or they can be set only once and switched by the CPU through a single control bit. In order to synchronize the camera and our dma, we thought about different alternatives:
  - The synchronization is fully controlled by interrupts, so the camera sends an interrupt when it finishes to write a frame and the dma sends an interrupt when it finishes to load a frame. In this case the CPU must make sure that the camera never writes on a location that is being loaded by the DMA.
  - The synchronization is controlled by a hardware signal that directly connects our DMA to the camera. In this case the camera sends an interrupt when it

finishes to write a frame, then checks the bridge signal to know if the next frame buffer is available.

- We make sure that our DMA is always faster than the camera. We use two frame buffers. The camera writes in the first one and sends an interrupt. In the interrupt routine the CPU starts our DMA. The camera is free to write in the second buffer. When it finishes, it sends the second interrupt. At this point the DMA has surely finished the loading of the first buffer, so it can accept the request of the second loading. The camera is free to write again in the first buffer because it is not being read by the DMA anymore.

We didn't implement none of the proposed methods because we could not find an agreement with the camera group. This part will be further analyzed when implementing the system.

3.6. FIFO Buffer

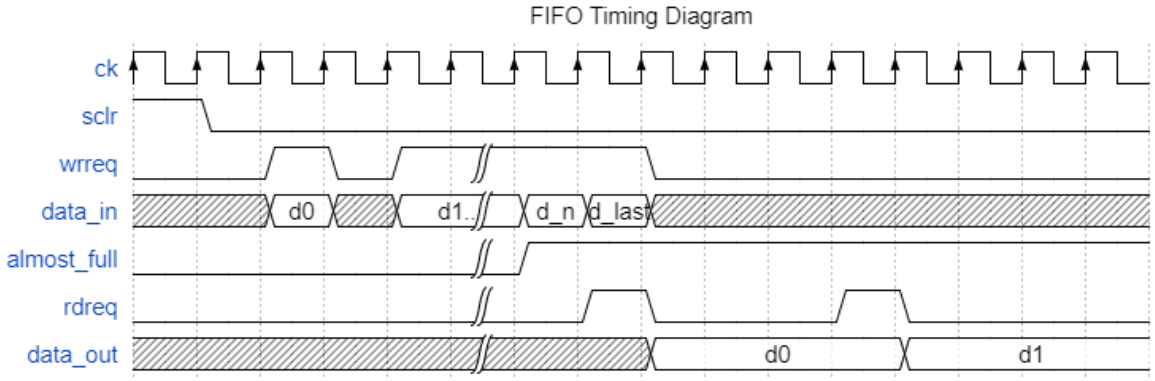


Fig. 3.5 - FIFO Timing Diagram (burstcount=2)

A FIFO memory can be efficiently instantiated using the template in the Quartus Megafunctions Wizard. We propose to implement a SCFIFO [3](Single Clock FIFO) with the following characteristics:

Data width: 16 bit	In accord with the LT24 interface data width (1 pixel).
Words size: 512	To overcome a very busy avalon bus or SDRAM (robust to 2048 wait cycles[4])
almost_full signal	To start sending the data to the LCD only when the FIFO is full enough. This signal is also used by the master_dma to know when to stop sending data. The almost_full signal will be

[3] Also a DCFIFO (Double Clock) can be used. If the rdclk is set with a 4 times greater clock period (as the 8080 protocol wants), the logic that requests the data from the FIFO can be simplified, and during the LCD loading, the rdreq can be fixed to 1 (pass-through of the data from the FIFO to the LT24). We decided to use a SCFIFO because the requesting logic already include a machine state to send the write command to the LCD, so it will not be inefficient to include another functionality.

[4] The LT24 consumes 16 bit each 4 cycle (timing constraints of 8080 interface) so it takes 2048 cycles to exhaust a full FIFO.

	designed to activate 8*32 bits from completeness. In this way, with a burst count of 8 (as we suggest[5]), the master can fill the FIFO sending a last transfer, after the signal almost_full is activated.
mode: normal mode	The show-ahead mode doesn't bring any particular advantage to our design, but it complicates the FIFO logic.

3.7. Avalon Master Unit

The function of the Avalon master unit is to move data from the memory to the FIFO. This operation is also called direct memory access (DMA). Since the unit has to take control of the bus it behaves as a master unit, meaning that the DMA accesses the bus concurrently with the NIOS II and the Avalon arbiter gives the grants.
An example of timing diagram of the master unit is presented in *Fig. 3.6.1*:

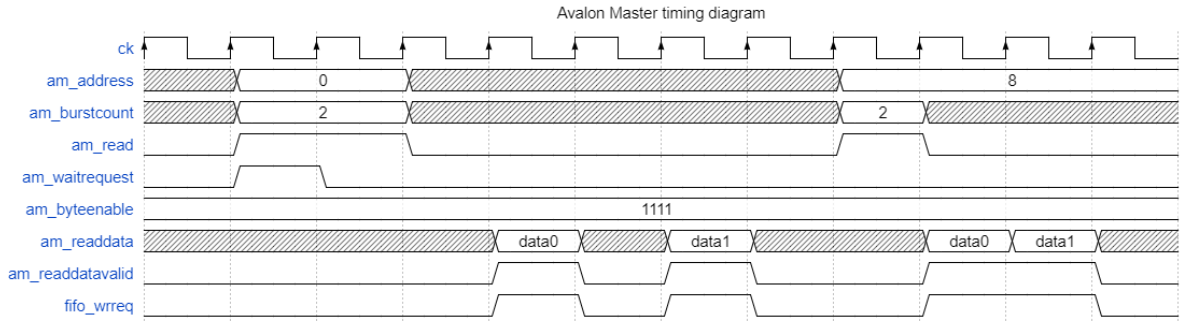


Fig. 3.6.1 - Avalon Master Reading Timing Diagram

ByteEnable is always equal to "1111", hence all the 4 bytes of *readData* are transferred in one-clock cycle. The DMA asserts *address*, *burstcount*, and *read* after the rising edge of clk. The slave asserts *waitrequest*, causing all inputs to be held constant through another clock cycle after the waitrequest is deasserted .
In the figure a *burstcount* of 2 is used for graphical reasons. On the contrary, we suggest to implement a burst count of 8, such that data is aligned with the FIFO dimension. With this burst size, the DMA takes control of the bus about 10%[6] of the time in segments of approximately 10 cycles, allowing the camera and the CPU to work properly. It must be noticed that in our design the burst capability is not fundamental, because the data is consumed by the LCD at a rate of 1 pixel each 4 cycles, slower than the dma (1 pixel each 2 cycles at full speed).

[5] The number of burst (that is software reconfigurable) must be equal or lower to the value we suggested in order to avoid data misalignment.
[6] A single access by the DMA brings 8 pixel data to the FIFO in approximately 9 cycles. To consider the latency of the DDR, let's say it takes 12 cycles. To fill a FIFO of 1KB (64 pixel), it takes 96 cycles. Then the dma asks for new data only once the LCD has consumed part of it (almost_full must go low). If we consider only the time while the bus is engaged by the DMA, we can consider the same transfer rate than before, so to load a total of 320*240 = 76800 pixels, it takes 115.200 cycles = 2.3 ms at 50 MHz. A frame is loaded in the LCD at a maximum rate of approximately 1 frame / 20 ms (50 Hz refresh frequency), so the bus is engaged about 2.3 ms each 20 ms, bringing us to a percentage of 10%.

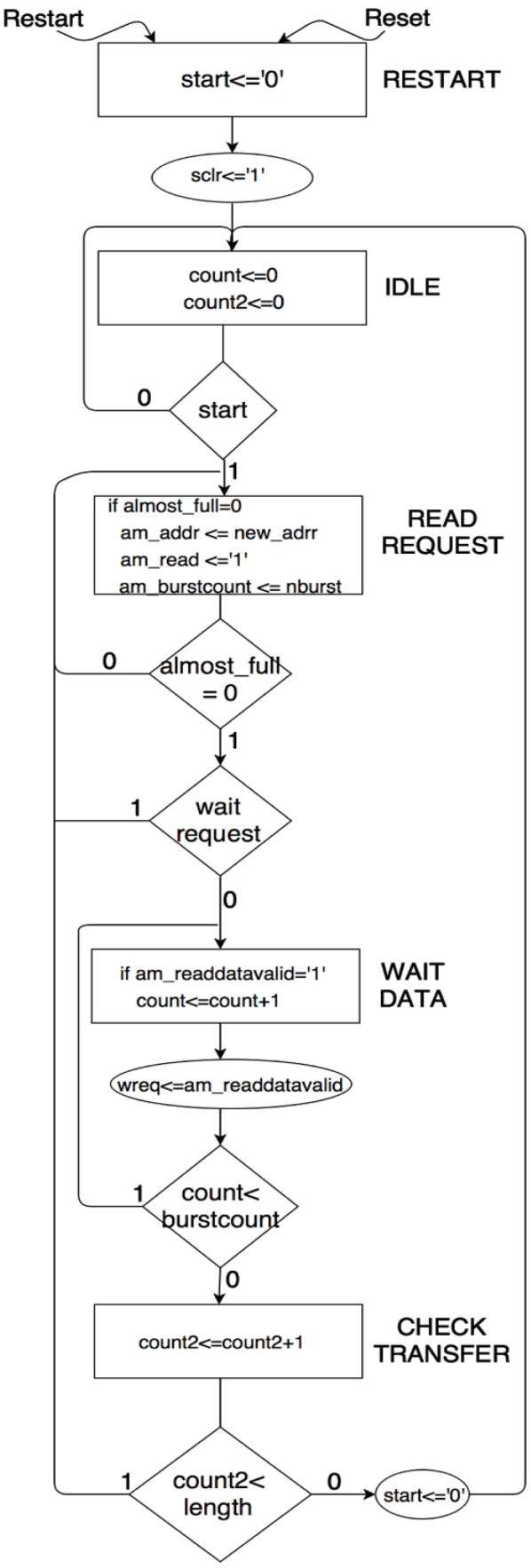


Fig. 3.6.2 - FSM diagram of Master_Dma

However a pipelined read access (no matter the burst) is strongly recommended when dealing with DDR controllers, which could have few cycles of latency, resulting in a DMA much slower than expected.

The conceptual finite state machine of the DMA is presented in Fig. 3.6.2

When the unit is reset or the restart signal is asserted, the module goes in the RESTART state. The restart signal is used as a soft-reset for the DMA when the LCD controller has to be reconfigured during the transfer of a frame. In the IDLE state the DMA waits the Avalon slave unit to provide the start trigger. Once the start bit is asserted, if there is a sufficient number of empty words in the FIFO the master sends a read request to the memory.
The new_addr “variable” will start from buffer1 or buffer2 address depending on a 1 bit signal selection and it is updated according to the formula:
new_addr <= new_addr + 4*burst_number
We need the multiplier 4 because our data is 32 bits width and the memory is organized in bytes.
When the memory has acquired the read request, it deasserts the wait request signal. The DMA increments an internal counter (count) each time the memory sends a valid data until burst_number of data are received. Data is sent to the FIFO through the data line of the Avalon master bus.
In the last state, if an entire frame has been transferred, the DMA goes in the IDLE state waiting for another start.

3.8. Avalon Slave

The Avalon slave allows the software configuration of the Master unit and LT24 interface. It receives commands by the CPU through the Avalon Bus . In particular:

- It provides the starting addresses and the length of one buffer in memory to the DMA.
- It can start/stop/restart the DMA.
- It sends configuration commands of the LCD (4 cycles fixed latency, see *Fig 3.8.2* for details)
- it controls the LCD_on and LCD_resn outputs

3.9. LT24 Interface

This unit acts as the medium between the LT24 and the processor or the FIFO. In the "*command routine*", it accepts commands/data from the CPU through the avalon slave and let them reach the LT24. For each transfer it makes the Avalon bus wait for 4 cycles, in order to fulfill the 8080 I timing specifications. When a write command is sent, the module switches to the "*loading routine*". At the beginning of each frame loading, it waits for the almost_full signal, then it pulls data from the FIFO (sending a read request every 4 cycles[7]) and sends it to the LT24. This routine can be interrupted by any command sent by the CPU, in any state, in order to not loose any CPU instruction. It is important in this case to soft-reset the dma in order to assures that the FIFO content is synchronized with the frame.

We decided to use registered outputs (assignment is done in the sequential process and has effect on the next rising edge), in order to have clean signals to the ILI9341. In fact, the 8080 communication is asynchronous, so glitches and spikes produced by race conditions or other agents could have dramatic effect on the system. The insertion of registers, delays the output of one clock cycle, but the communication is unidirectional, so it does not cause any problem.

In the photo mode there is no need to send the data at each refresh of the display, because the ILI9341 contains an internal Graphic RAM, which can hold the data. In video mode, instead, we should load a new frame (if available) at the starting of each frame. The controller ILI9341 has a specific signal called *Tearing effect*, which is activated when a new frame is being displayed (vsync). It would have been useful to synchronize loadings, but we could not find this signal in the LT24 schematic. We suppose it is not cabled to the GPIO interface. However, the ILI9341 datasheet assures that no graphic artefact is produced when loading the data while it is being displayed, so we don't need any synchronization logic.

Waitrequest signal is not shown in the FSM, but in order to make the master keep the data for 4 cycles it must assume the same value of as_write in CMD_WAIT, CMD1, CMD2 and must be reset to 0 in CMD3 (to release the master). It can be a combinatorial signal associated to the state. We could store the as_wrdata in a register rather than holding the master, but in this way we could not handle consecutive accesses by the CPU.

[7] If the FIFO is Dual Clock, this logic has not to be included.

Continue is a register which remains high when a command, which requires following data, is sent. It makes sure that the csx is not deactivated during a multi-transfers command. It can be set through a bit in the avalon wrdata, so the software should take care of it.

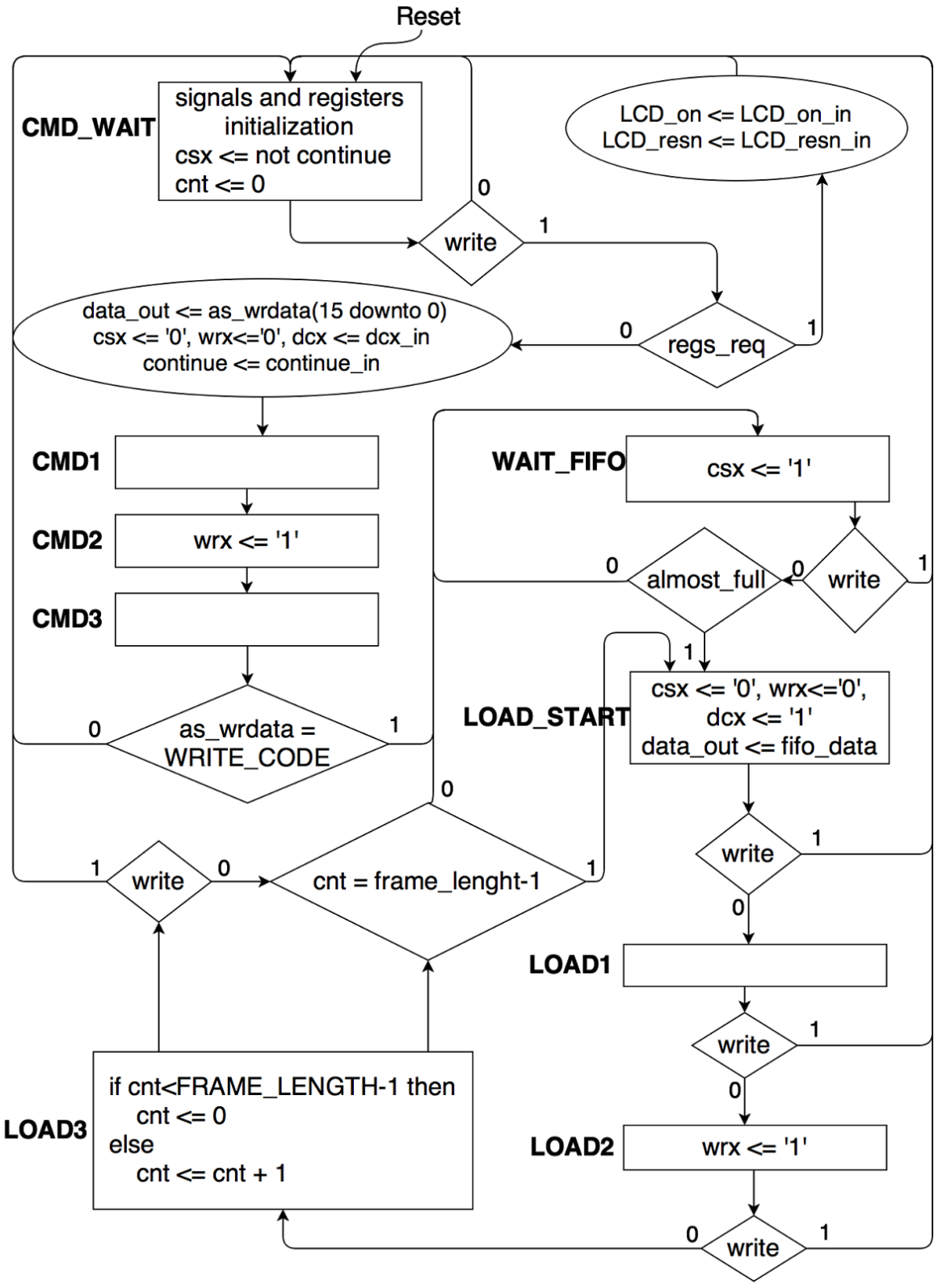


Fig. 3.8.1 - FSM diagram of LT24_interface

LCD_on and *LCD_resn* are two registered outputs which drive LCD_on and reset_n signals of the LT24. In order to set their values, *regs_req* [8]must be asserted during a write cycle. Normally LCD_on must be high in order to supply power to the display and reset_n can be activated before the initialization of the LT24 for a complete reset of the LCD.

The reset routine must follow some timing constraints which are shown in the LT24 initialization template, provided in the course slides. This template is actually also used to initialize the LCD. The system was designed to be compatible with it.

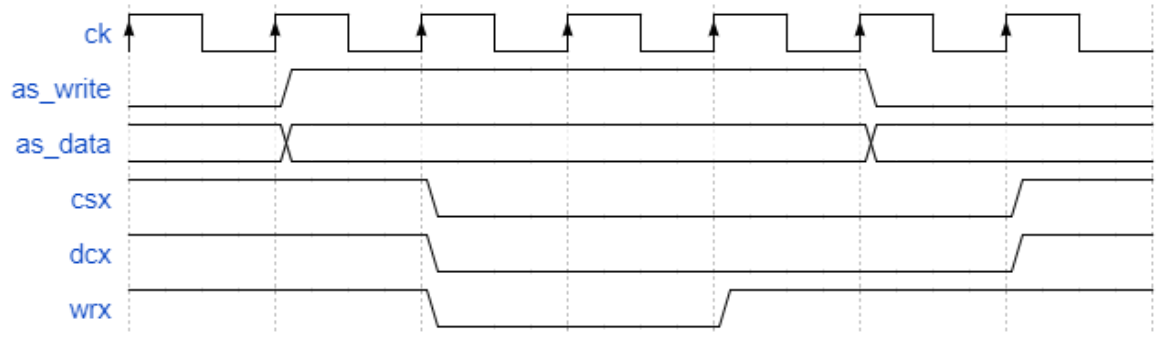


Fig 3.8.2 - LT24 command timing diagram (registered outputs)

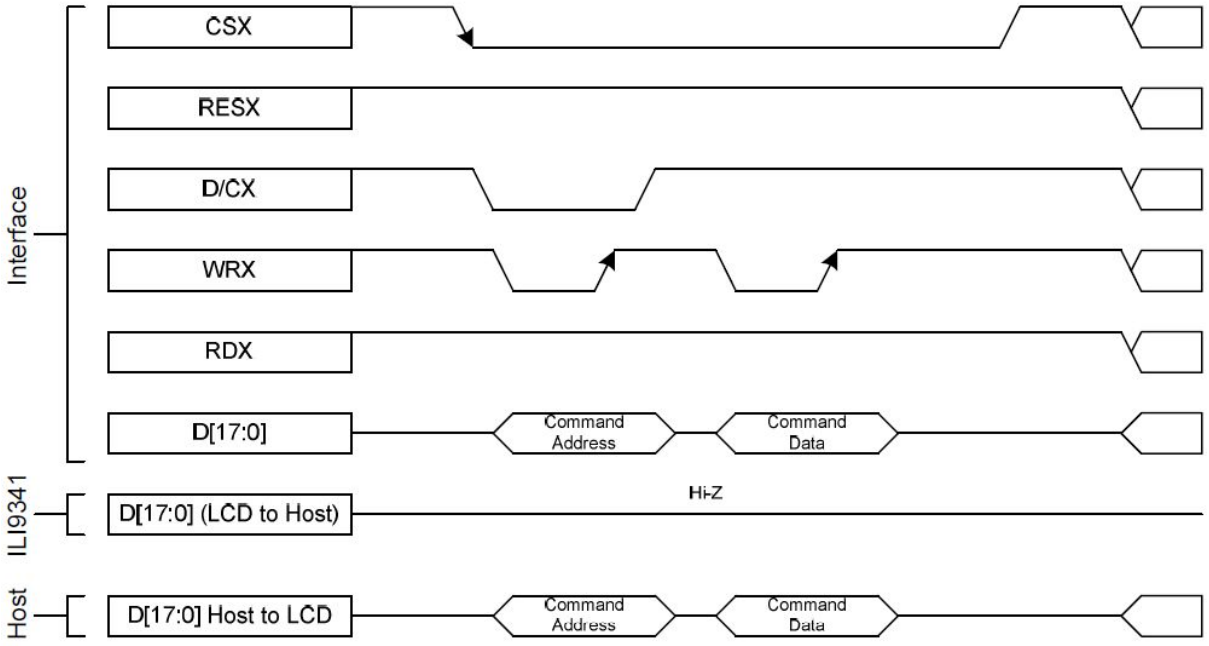
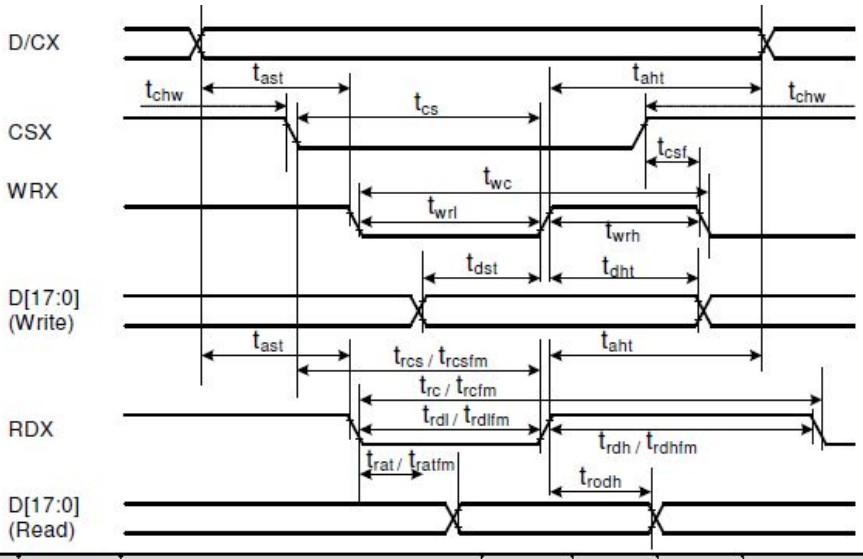


Fig. 3.8.3 - Example of “continuous” command

[8] It is a special purpose bit of the avalon slave data bus.



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	

Fig. 3.8.4 - Timing constraints of 8080 I interface of LT24

Finally, the *Fig 3.8.3* shows the timing constraints of the LT24 interface. They are fulfilled using 4 cycles of a 50 Mhz clock, as proposed above. The wrx signal must be activated for the first two cycles and deactivated for the others.

4. References

1. Avalon® Interface Specifications
2. SCFIFO and DCFIFO IP Cores User Guide
3. ILI9341
4. LT24 User Manual
5. LT24 Schematic (LT24_v.1.0.2_SystemCD)