LAB 3.0 REPORT

Camera, LCD & VGA Conceptual Design

CS 473: Embedded Systems

Jason Mina

Ahmet Avcioglu

Contents Table

[I. Introduction: 3](#_Toc123755304)

[II. High Level Description: 3](#_Toc123755305)

[III.Low Level Descriptions: 4](#_Toc123755306)

[a-)Custom LCD Interaction Module IP Block Diagram: 4](#_Toc123755307)

[IV. Finite State Machine of DMA 10](#_Toc123755308)

[V. Finite State Machine of LCD Controller 14](#_Toc123755309)

[F. Simulation Results 18](#_Toc123755310)

[Registers Component: 18](#_Toc123755311)

[LCD Controller: 19](#_Toc123755312)

[Case 1: 19](#_Toc123755313)

[Case 2: 19](#_Toc123755314)

[DMA Controller: 20](#_Toc123755315)

[Case 1: 20](#_Toc123755316)

[Case 2: 21](#_Toc123755317)

[Case 3: 21](#_Toc123755318)

[Full Component: 22](#_Toc123755319)

[G. Qsys Design for the LCD part 22](#_Toc123755320)

[G. Testing on Hardware 23](#_Toc123755321)

[Writing an Image to the Memory 23](#_Toc123755322)

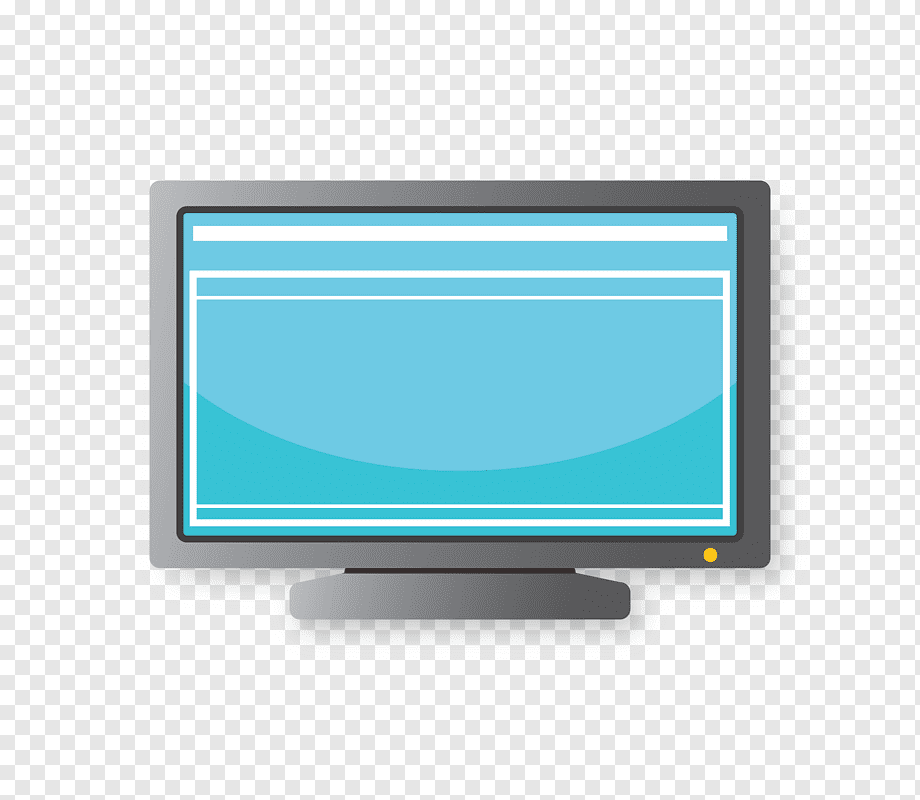
[Writing an Image to the Memory 24](#_Toc123755323)

# I. Introduction:

The purpose of Lab3 is to design custom master interfaces to be used for complex cases where large amounts of data need to be moved from a peripheral (i.e camera) to memory and from memory to a peripheral (i.e display). In this report, the design of an embedded system on the FPGA to implement the previous described system Is presented, where a high level diagram is presented to show the complete interaction between the different components, and where more detailed explanations of the design choices for both the memory fetch and memory deposit are presented, and the choice of compatibility between these two subsystems is explained.

# II. High Level Description:

The following High-Level Block Diagram Demonstrates the high level interaction of the DE0-Nano-Soc FPGA with the camera and the LCD. The LCD interaction Module and Camera Interaction Module Components are required to have access to the Avalon Bus to fetch/send information from/to memory, so they have a master interface with the Avalon Bus, and since both modules need components to be Initialized, they have a slave interface with the Avalon Bus through which the Nios Processor accesses and writes information to these component’s registers. The rest of the report discusses the components of the LCD Interaction Module and Camera Interaction Module in detail.



I2C

SDRAM Controller

Figure : High Level Diagram of the System

Master Slave

AVALON BUS

NIOS II Processor

JTAG UART

LCD Control Module

Camera Control Module

SRAM



SDRAM

# III.Low Level Descriptions:

## a-)Custom LCD Interaction Module IP Block Diagram:

FIFO

LCD Control

LCD Conduit

LCD\_ON

RESET\_N

CS\_N  
R\_S

WR\_N  
RD\_N

D[15:0]

RdData[15:0]

FIFO\_Almost\_Full

Master

Controller

ReadDataValid

clk

Reseet\_n

Address[2:0]

C\_S

read\_data[31:0]

read

Ads\_Src[31:0]

Burst\_Cnt[6:0]

read

Wait\_request

Read\_data[31:0]

Burst\_Cnt[6:0]

Data\_Length[15:0]

Ads\_Src[31:0]

Figure : IP Block Diagram of the Custom LCD

**Avalon Bus**

Write\_data[31:0]

write

Registers

Master\_Begin

LCD\_ON

clk

WrData[31:0]

WrFIFO

RdFIFO

clk

FIFO\_Almost\_Empty

FIFO\_Empty

frame\_end

Data[15:0]

C\_D

LCD\_INIT[3:0]

Change\_LCD

R\_S

FIFO\_Full

frame\_end\_M

The LCD control Module consists of 4 main components: the Registers Avalon Slave, the Avalon Master Controller, the First In First Out Buffer (FIFO), and the LCD control. The registers component is needed to initialize both the Master Controller and the LCD control at startup. It consists of several registers that are defined which need to be programmed by the NIOS II processor to program the Master controller and the LCD control at startup. As an Avalon Slave, the Registers component has an interface with the Avalon Bus through which is connects to the NIOS processor master, through a set of predefined signals specific to Avalon specifications. The Master controller is to transfer a huge amount of data from memory to the LCD control, and thus acts as a DMA, which also has an interface with the Avalon Bus through predefined signals according to Avalon specifications. The LCD control is used to act as an intermediary to sequence the RGB data, between the Registers and LCD controller ILI9341 on one side and the FIFO and ILI9341 on the other side. Since the two asynchronous systems of read from memory and write to memory are to interact, the FIFO components needs to act as a buffer to synchronize the read/write procedures between the data source through the Master controller and the data sink through the LCD control.

1. Registers:

The IP custom component presented above interacts with the NIOS II processor through the Avalon bus through its register component which acts as the Avalon slave of the component. The registers component is connected to both the master controller and LCD control to initialize these components and synchronize them.

Registers Map:

To initialize the LCD, the ILI9341 need to execute a certain sequence of commands. Therefore, the ILI9341 needs to be provided with certain addresses of registers to write at and the data to be written at these certain addresses.

* iReg\_Cmd\_Adrs:

this register is needed to store the address of the ILI9341 to which the data will be written at for it to execute commands.

* iReg\_Cmd\_Data:

this register is needed to store the data of the command to be written at the specific address given in iReg\_Cmd\_Adrs.

* iReg\_LCD\_Init:

this register stores the bits needed as the inputs to configure the ILI9341 controller:

1. LCD\_ON:

This bit is set to “1” when the Display is on, and to ‘0’ when it is off.

1. RESET\_N:

This bit is set to ‘0’ in order to reset the controller to defaults.

1. CS\_N:  
   this bit is set to ‘0’ always to enable the transmission of data.
2. Read\_Frame:

this bit is used to tell the LCD control that it is receiving the data from the FIFO. When this bit is set to ‘1’, LCD controller will start to read data from FIFO depending on the signals FIFO\_Empty and FIFO\_Almost\_Empty. This bit is also used to pulse Master\_Begin signal which is used to synchronize the DMA and LCD controller(will be explained later).

1. R\_S:

This bit is to differentiate between command data write or command address write when in the mode of receiving command info from the registers component. When data is written into iReg\_Cmd\_Adrs, it is set to zero, and when data is written to iReg\_Cmd\_Data, it is set to one.

* iReg\_Ads\_Src:

this register Is needed to store the address in memory from which the master controller is to fetch the pixel data. This address is provided by the controller to the Avalon bus when from\_reg = ‘1’ meaning that the LCD should give a display.

* iReg\_Data\_Length:

this register stores the length of the data to be fetched from the memory and which will be given to the DMA master controller. This length will be equal to one image captured by the camera.

* iReg\_Burst\_Cnt:

this register stores the number of bursts that the master requests data included in one read request. The value of Burst count will be chosen to be equal to 64.

* iReg\_LCD\_Done:

this register will be set to ‘1’ when a whole frame passes the LCD control. This is done by counting the number of pixels which have passed and is equal to pixels/frame.

* iReg\_DMA\_Done:  
  this register is similar to iReg\_LCD\_Done, but this checks whether a whole image has passed through the DMA by counting the number of bursts, which should reach a value of image\_size/size\_of\_burst = 38400/64= 600.

The following Register Map is obtained:

|  |  |
| --- | --- |
| Address | Register |
| 000 | iReg\_Cmd\_Adrs [15:0] |
| 001 | iReg\_Cmd\_Data [15:0] |
| 010 | iReg\_LCD\_Init [3:0]:  0: LCD\_ON  1: RESET\_N  2: from\_reg  3: R\_S |
| 011 | iReg\_Ads\_Src [31:0] |
| 100 | iReg\_Data\_Length [15:0] |
| 101 | iReg\_Burst\_Cnt [5:0] |
| 110 | iReg\_LCD\_Done |
| 111 | iReg\_DMA\_Done |

Figure : Table of Register Map

1. Master Controller:

As large bulks of data are to be transferred from memory to the display, a specialized unit is to be use to fetch the data instead of the processor, which is the master controller, acting as a DMA. The Master controller acts as the Avalon Master of the component through a set of specific signals specified by Avalon design. The DMA in general needs specific signals to operate, namely the source address, here the address of the memory where the picture is stored, the length of data (of the image), and the burst count (number of transfers in one read request to the Avalon Bus). In our case, three additional signals are provided through the interface between registers component and the master controller, which are the LCD\_ON (to synchronize the start of operation of the master and lcd controllers), and Master\_Begin, which is a signal to be synchronized with the from\_reg and is pulsed when from\_reg=’1’ (the lcd is in display mode). The master controller provides the signal frame\_end\_M to the registers when the DMA finishes fetching data from the memory.

The Ads\_Src and read\_data length is 32 bits which corresponds with the length of addresses in memory, while the burst\_cnt is 6 bits which can represent the burst count of 64.  
The master controller has a slave port where it is programmed at startup from the NIOS II processor through the registers component, and is presented by the arrows from the registers component into the master controller. As one master can access the memory at a time, arbitration is needed which is done through the “wait\_request” signal provided through the Avalon Bus.

The master controller has also an interface with the FIFO unit, where the data retrieved is provided to the FIFO through the WrData signal, after setting the write request WrFIFO and after checking if it is full through the signals coming from FIFO, the FIFO\_Almost\_Full and FIFO\_Full.

1. FIFO:

The main purpose of the project is to fetch data from camera to memory and from memory to display, however these two processes are asynchronous, and therefore a FIFO component is needed to synchronize the transfers.

The data is written from the Master controller to the FIFO through the WrData of width 32 bits, however the LCD\_Controller reads 1 pixel at a time, therefore 16 bits at a time through the RdData signal. Therefore the width of the FIFO is chosen to be equal to 32, and the height is chosen to be 4 times the burst count, in order to be able to handle delays of up to 4 burst counts.

The FIFO\_Almost\_Full will be set to one once 3 of the 4 rows are filled, while FIFO\_Almso\_Empty is set once 1 of the 4 rows are filled. This will dictate when the LCD\_control will read data and when the Master controller will provide the data.

32 bits

Figure : FIFO Diagram

**FIFO**

4xburst\_cnt

= 256 rows

1. LCD Control:

The LCD Control is use to sequence the LCD signals and send RGB data. The LCD LT24 is initialized through the ILI9341 set of commands whose addresses and data are sent from the Registers component through the LCD control. The LCD control also gets the Display data as 16 bits data from the FIFO when it is not empty.

LCD control sends the data read from the FIFO to a conduit to connect to to separate 16 GPIO pins each corresponding to one of the 16 bits.

The interface between the FPGA and the LT24 (ILI9341) consists of specific LCD configuration bits including the LCD\_ON, RESET\_N, CS\_N, from\_reg, and R\_S. The R\_S signal is sent on its own as a signal to be sent to the ILI9341 Cx/D when reading the data[15:0]. The LCD Control component sends the signal frame\_end to the registers component which will store it in its iReg\_LCD\_Done, once the whole frame has been processed by the LCD Control unit.

A picture containing diagram

Description automatically generated

Figure : FPGA and LT24 Interface

1. Camera and Display Communication Specifications:

Memory organization should be compliant between the camera and display modules. The LT24 LCD is a 240(H)x320(V), with each pixel being represented by 16 bits (5R, 6G, 5R). But since the memory addresses are 32 bits, each address will hold 2 pixels.

The camera will store the picture as rows of 320 pixels, summing up to 240 rows. Thus each row will sum up to 160 addresses, and the total number of addresses needed is 160\*240 = 38400 addresses. The below figures explain the chosen design.

Pixel 1. Pixel 0

pixel 319

Figure :Representation of the Image pixels

IMAGE

pixel 320

pixel 319 + 320\*239

Address 0 15 16 31

Row 2

Row 1

|  |  |  |
| --- | --- | --- |
| Start Address | Pixel 0 | Pixel 1 |
| Start Address +1 | Pixel 2 | Pixel 3 |
|  |  |  |
|  |  |  |
| Start Address + 159 | Pixel 318 | Pixel 319 |
|  |  |  |
|  |  |  |
| Start Address +256 | Pixel 320 | Pixel 321 |
| Start Address + 257 | Pixel 322 | Pixel 323 |
|  |  |  |
|  |  |  |

Figure : Representation of the Address Architecture

# IV. Finite State Machine of DMA

When designing the finite state machine, we should respect both the timing diagram of the DMA read cycle and the status of the FIFO. In this FSM, used reference values are as follows:

**Burst\_Counter**: This counter is used for counting the number of word transfers from memory to FIFO. This counter is incremented every time “readdatavalid” signal is generated because this indicates the number of bursts in a transfer. This counter is compared to the iReg\_Burst\_Cnt[5:0] which keeps the number of bursts that the Master Controller read from memory in one read request. For this design, we chose the burst number 64. Therefore, the state machine compares the counter with the value of 64.

**Frame\_Counter**: This counter is used for specifying whether one full frame is read or not. In other words, if Frame\_Counter is equal to 600, this means that one full frame is fetched from memory to FIFO. 600 is dependent on bursts that are defined in iReg\_Burst\_Cnt register. One frame covers 38400 addresses in memory, and DMA reads 64 of them in a burst transfer. Therefore, the burst transfers needed to transfer a full frame equals 38400/64 = 600. If the burst number is changed in the design, the value that we are comparing to the Frame\_Counter will change as well. It is important to note that Frame\_Counter is incremented in every burst transfer.

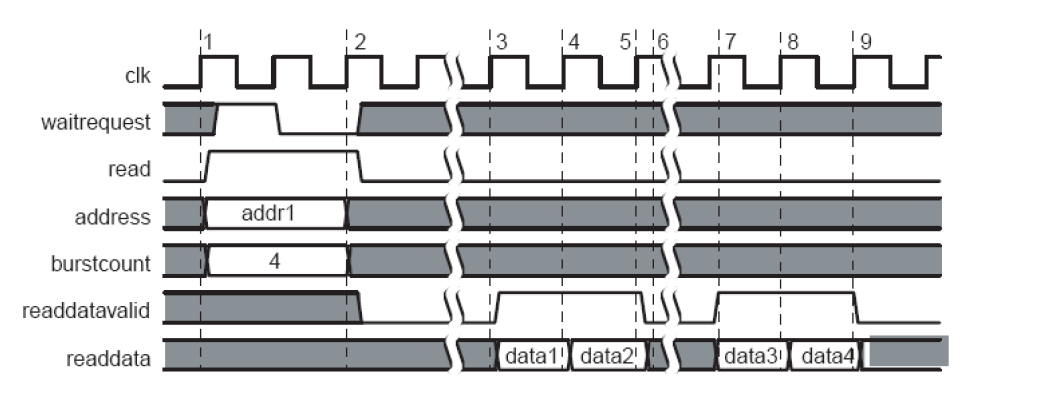


Figure : Read Cycle timing diagram of DMA

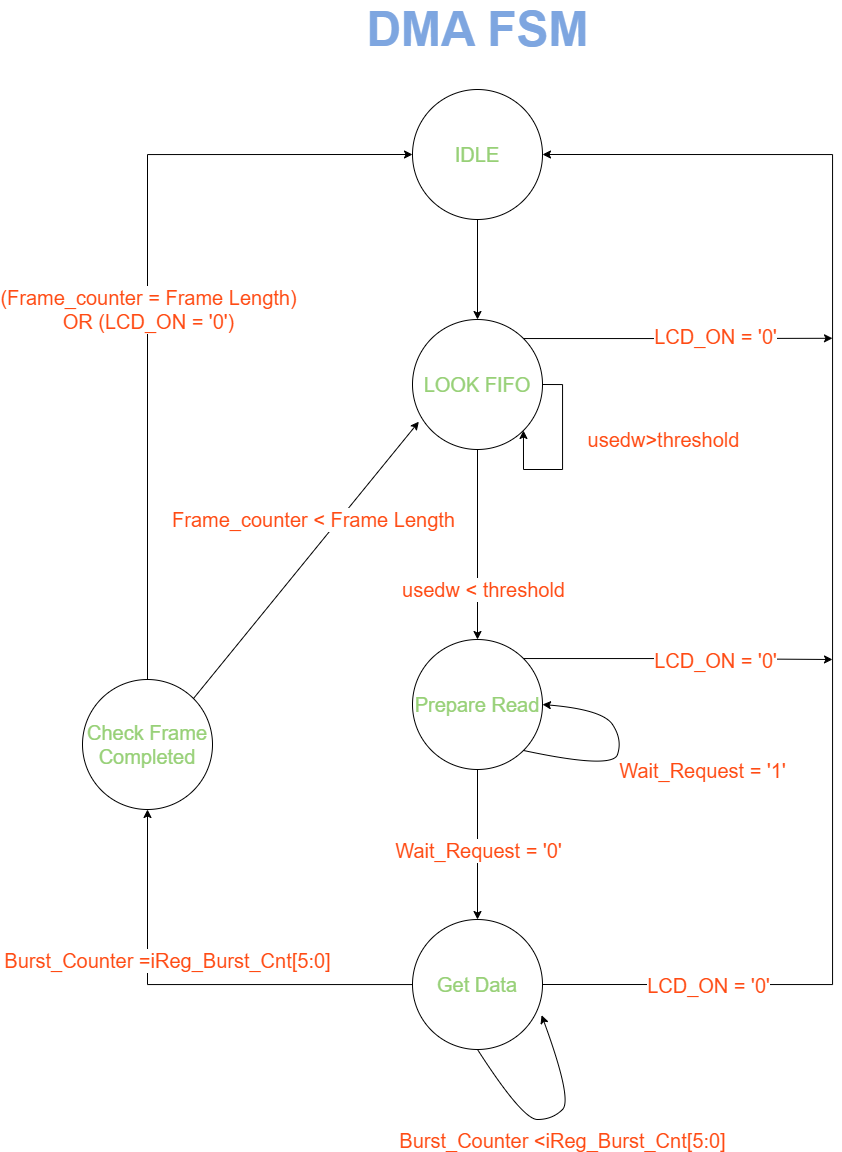


Figure : Finite State Machine of DMA

In our FSM, there are 5 states namely,” Idle”, “Look FIFO”, “Prepare Read”, “Get Data”, and “Check Frame Completed”. The operation and transition of the states are as follows:

**Idle**: In this state, everything is reset and nothing is happening which gives the state named “Idle”. It is important to note that all the counters are also reset here. “LCD\_ON” signal equal to ‘1’ indicates that the LCD is on and “Master\_Begin” signal equal to ‘1’ denotes that the mode of LCD controller is reading display data from the FIFO. Therefore, LCD\_ON = ‘1’ and Master\_Begin = ‘1’ starts the operation. Otherwise, the state is kept at Idle state. If the operation is started, the next state will be “Look FIFO” state in which we check whether FIFO is almost full or not.

**Look FIFO**: This state is used for coordinating the content of the FIFO. In other words, the aim of creating this state is to use FIFO efficiently and take measures against potential errors. Since FIFO has some capacity of storing data, we should control read and write operations to the FIFO. In our design, the usedw signal will denote that there is not enough space for storing a complete burst transfer. That is to say, when the usedw signal is greater than or equal to 192, FSM will stay in the state of Look FIFO. If the value of the signal usedw is lower than 192, then the state will be changed to “Prepare Read”. As mentioned earlier, we chose a FIFO depth of 256, which enables us to store 4 bursts transfer. If more than 192 rows of the FIFO are full, then the usedw signal specifies that and the state machine waits in this state for LCD controller to read some of the data and makes space for the new data. When the space for the new data is created, the transition to next state of “Prepare Read” is done.

**Prepare Read**: This state is created for setting required signals for Avalon bus read operation. In this state, “Read” signal is set to ‘1’. Also, the start address of the memory(Ads\_Src), and burst count (Burst\_Cnt) are set using the values that come from Avalon slave registers. At the same time, the system checks for the “Wait\_request” signal. If it is ‘1’, the system should wait because the bus is busy. Therefore, the state change is done when “Wait\_request” signal is equal to ‘0’. It is important to note that in “Prepare Read” state, if “Wait\_request” signal is ‘1’, we should keep the read, Ads\_Src and Burst\_Cnt signals valid until wait request signal goes to 0.

**Get Data**: In this state, the DMA fetches data from memory and puts it into FIFO. To do this, the DMA checks the validity of the read operation for every burst in this burst transfer. If the data is valid, DMA fetches 32-bit word from memory and sends it to the FIFO by sending a write request. This write request is done by WrFIFO signal. The data that will be sent is put in WrData[31:0] line. If the data is not valid, the WrFIFO signal and WrData line are kept at zero.

This process will be repeated until all of the transfers in a burst transfer are done. In this design, we chose the burst count as 64 which is the value in the register iReg\_Burst\_Cnt. Therefore, this process will be repeated 64 times and this will be controlled by a counter named Burst\_Counter. The counter will be incremented in every transfer, and when Burst\_Counter is equal to the burst count (64 in this example), the state machine transits into the “Check Frame Completed” state. It is important to note that every time the FSM transits from “Get Data” to “Check Frame Completed” an internal signal will be updated to correctly access the corresponding section of the memory in the next burst transfers.

**Check Frame Completed**: As explained earlier, we need 600 burst transfers to get a complete image displayed on the LCD. Therefore, we put a state that checks whether a complete frame is read until now or not. This control is achieved utilizing a counter named Frame\_counter. If the value of Frame\_counter is less than 600, the system will return to “Look FIFO” state to restart a one cycle of burst transfer until it finally fetches a complete frame. If it is equal to 600, we set Frame\_End\_M signal which is sent to Avalon slave registers. This signal is used to tell the registers component that a full image has been read. The write FIFO signal and WrData line are reset and the transition to the “Idle” state is realized. Then we return to the beginning of the loop, and as explained earlier the system will wait for a new “Master\_Begin” signal to restart the loop and read a new frame.

As a general remark for the whole FSM, if the LCD is somehow turned off, regardless of which state the system is at, the next state is “Idle” state. This measure hinders the potential problem of stucking at any state especially “Look FIFO” state because if the LCD is turned off, the LCD controller does not read anything and FIFO does not drain away.

As a last remark for the Avalan Slave part and “Idle” state, the “Master\_Begin” signal is not always ‘1’. It is set to ‘1’ for just one clock cycle because the system will never stop at “Idle” state but keeps transiting to “Look FIFO” state. Therefore, the system will continuously load the FIFO which is not efficient. To use FIFO more efficiently, the LCD controller and the DMA controller is synchronized by setting the Master\_Begin signal whenever the Read\_Frame bit of iReg\_LCD\_init is set.

# V. Finite State Machine of LCD Controller

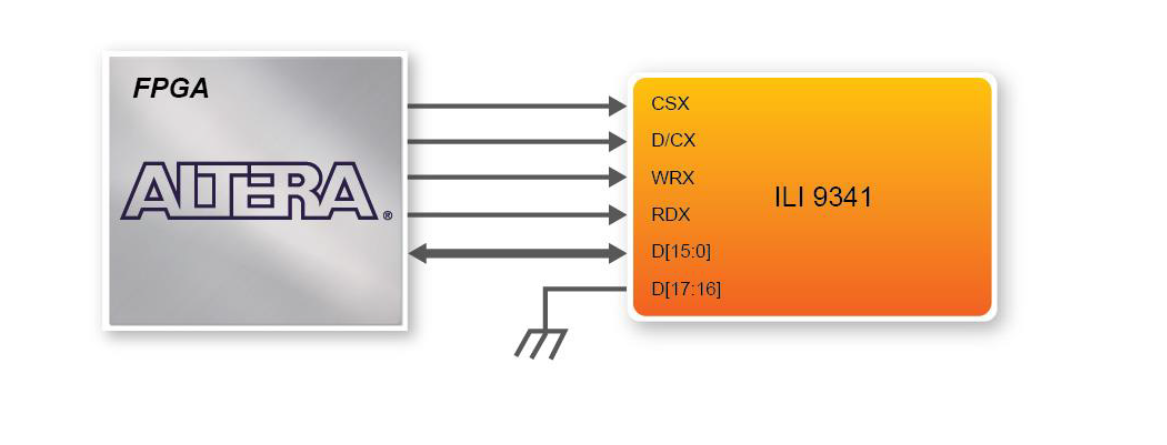


Figure : Connection of LCD controller and ILI 9341

As we can see in figure 7, there are 5 signals that should be managed to drive the LCD. The detailed timing diagram for these signals is presented in figure 8.

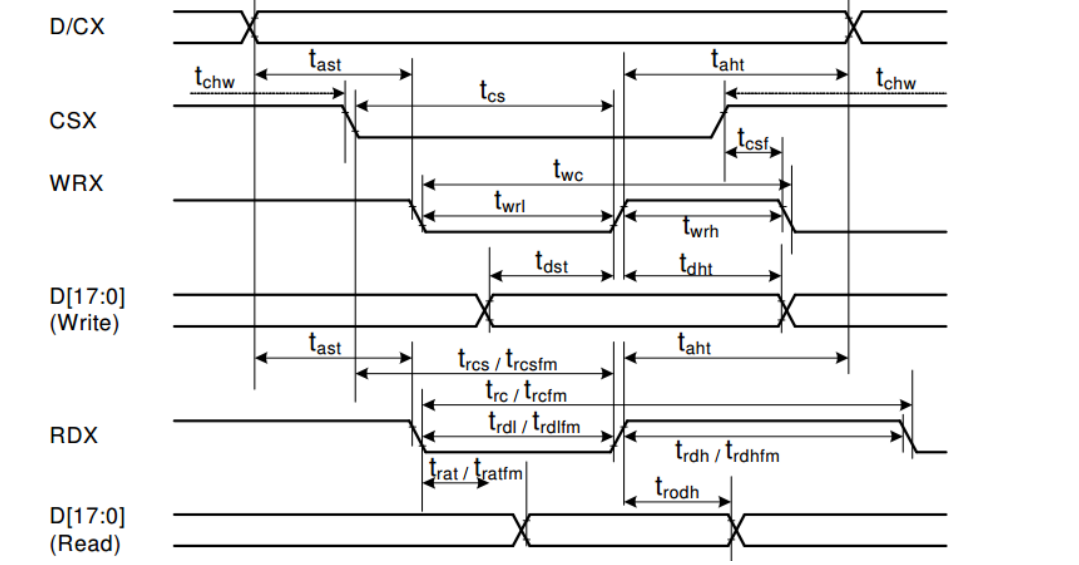


Figure : Timing diagram of LCD controller

As stated in figure 8, there are timing requirements needed to meet in order to drive LCD accurately. Since read cycle is used only for debugging purposes, our main focus in this report will be write cycle timing requirements. The most important ones are twc, twrl and twrh. “twc” indicates write cycle duration and must be minimum 66 ns. “twrh” specifies write control pulse H duration and must be minimum 15 ns. Similarly “twrl” corresponds to write control pulse L duration and must be minimum 15 ns. Since our chosen clock has period of 20 ns, we need at least 4 clock cycles to carry out write cycle. This can be implemented in two ways, one way is to create 2 states in which the state machine stays 2 cycle. The second way is to create 4 states each spending one clock cycle for transition. Since the second choice is efficient in terms of hardware, we chose the second one and added 2 wait states to the state machine of LCD controller.

The signal WRX is a write signal and writes the data available in the data line at the rising edge. CSX is an active-low chip select. D/CX is Data or Command selection pin. If it is ‘1’, the value in the data line is interpreted as data. If it is ‘0’ it is interpreted as a command. RDX is a read signal and MCU read data at the rising edge. This signal will be used only for debugging purposes. Lastly, D[15:0] is the data bus. The D0~D7 are used to translate the command to ILI9341 in the command mode(D/CX=0). The D0~D15 are used to translate the RGB data to ILI9341 in data mode(D/CX=1).

Our FSM consists of 5 states namely, Idle, reading data, wait 1, flip write, and wait 2. This finite state machine is constructed respecting the timing diagram as explained before. LCD\_counter is a counter which enables us to count the number of pixels. Since our LCD is 240x320 = 76800 pixels, the comparing value is 76800. If all the pixels are read, the state machine returns to the Idle state. Also, it is important to explain the signals “Change\_LCD” and “Read\_Frame”.

Change\_LCD: As explained earlier, this signal is used to indicate the first mode of reading from the Avalon slave registers instead of reading the frame.

Read\_Frame: As explained earlier, this signal inditaces that the LCD controller will read the data from FIFO which corresponds to data of a pixel.

As in the DMA state machine, the LCD controller state machine will also loop over its state machine. This type of state machine is useful if you repeat some “unit” operations again and again. Therefore, we have designed the FSM as this.

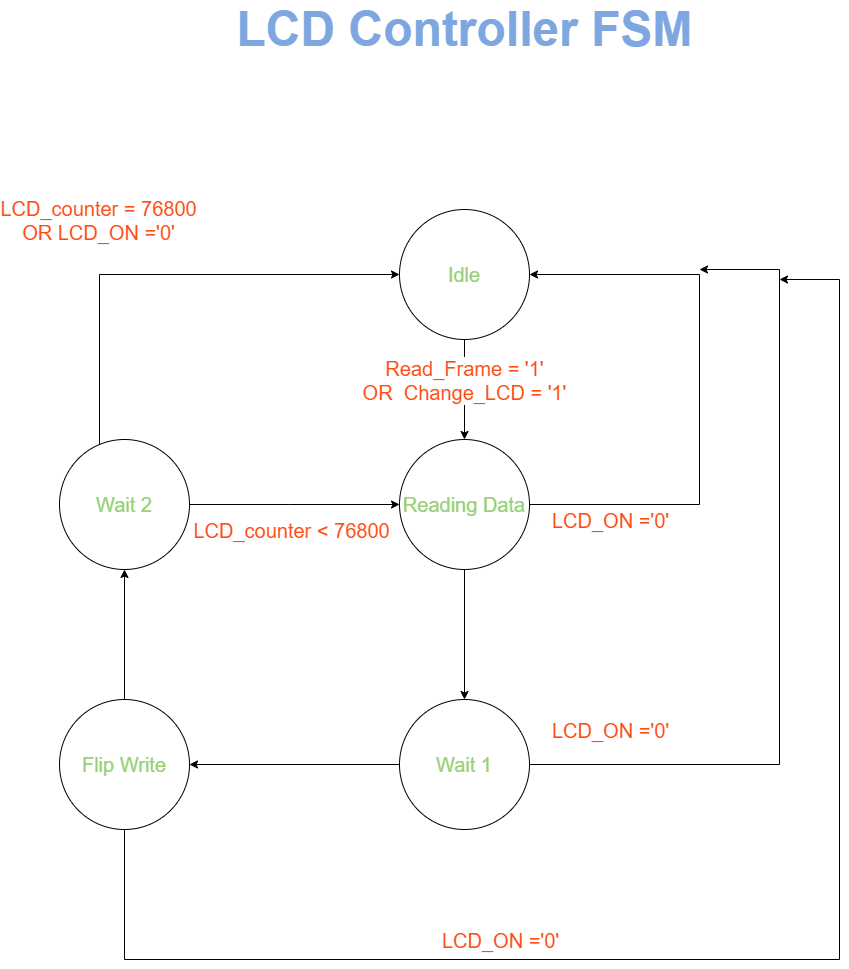


Figure Finite State Machine of LCD controller.

**Idle**: The state machine starts from this state. In this state, nothing is happening and every signal is Reset. When, Read\_Frame or Change\_LCD signal goes high the FSM transits from this state to the “Reading Data” state.

**Reading Data**: In order to understand the logic behind this state, one should look at the explanations of “ Read\_Frame” and “Change LCD”.

**Mode 1: Change\_LCD = ‘1’**

If this signal is ‘1’, the LCD controller read from registers. While reading from Avalon Slave Registers, we have two possibilities. Either command or data will be read. This differentiation is done by “R\_S” bits of “iReg\_LCD\_Init” register in the Avalon slave. If R\_S = ‘0’ then the LCD controller will write a command. In other words, the value of iReg\_Cmd\_Adrs will be put in the data line. If it is ‘1’, then the data will be written which means the value of iReg\_Cmd\_Data will be put in the data line.

**Mode 2: Read\_Frame = ‘1’**

If this signal is ‘1’, the LCD controller will read from the FIFO. Therefore, we check the FIFO\_empty signal to verify whether there is data to read or not.

If FIFO is not empty, the LCD controller will start reading 16 bits from the FIFO. This can be achieved by sending a read request through the RdFIFO signal. It is important to note that signal width is chosen compatible with the data line of ILI9341 which is 16 bits. If the FIFO is empty, the “WRX” signal will be held at ‘1’ and the state will not change until the FIFO gives the sign that it is not empty anymore.

To sum up, after this state, the data regardless of its type (command address, command data, or pixel data) will be put in the data line of ILI9341.

**Wait 1**: This state is a kind of dummy state. In this state “WRX” signal will be held at ‘0’ for one clock cycle. This state is used to meet the timing requirements as explained earlier.

**Flip Write**: In this state, the “WRX” signal will be flipped from 0 to 1.

**Wait 2**: During this state, the “WRX” signal will be held at ‘1’ for one clock cycle in order to meet the timing requirements.

Also, in this state, the controller checks whether the read\_frame signal is ‘1’ or not. If It is ‘1’, the LCD controller checks the LCD\_counter to verify whether one full frame is sent or not. If so, the controller sends the signal frame\_end to specify one full image is read and displayed. This signal will be used to load the base address for the new frame into Avalon slave registers. If the counter is less than the pixel number (76800), then the state transit back to the “Reading Data” state to complete the other pixels.

If the “Change\_LCD” signal is ‘1’, which means the LCD Controller was reading from the register map and executing commands, the state machine always transits back to the idle state where we wait for new commands or frame data to write.

# F. Simulation Results

We have built a testbench for each submodule for different cases namely registers component, LCD controller, DMA controller, and complete component. In this way, we made sure that each system works correctly which enable us to take firm steps forward.

## A computer screen capture Description automatically generated with medium confidenceRegisters Component:

The above figure shows the simulation of the registers component. AS can be seen, the Change\_LCD signal is pulsed whenever a value is written to the command address register or to the command data register. In the case of the command address register, the R\_S signal is set to 0. Writing ‘1’ to the third bit of the iReg\_LCD\_Init register will signal to the DMA the start of of reading from memory, which is done by pulsing Master\_Begin signal.

The following validations are observed.

First Case:

The Change\_LCD signal is pulsed whenever a value is written command address or command data register. This is done in order for the LCD controller to write to the LCD one time without repetition.

In the figure above, it is seen that the Change\_LCD pulses one clock cycle after ‘000’ or ‘001’ is written into the address register, which correspond to writing to command address and command data register respectively. When another value is written to address register, it is not pulsed.

Second Case:

The Registers component synchronizes the start of the master controller read-write process with the LCD Controller’s read-write process, which starts by setting the third bit of the ‘iReg\_LCD\_Init’ register to 1. Consequently, the Master\_Begin signal is pulsed as can be seen in the figure, telling it to begin reading the pixels from memory and writing them into the FIFO.

Pulsing is done so that the Master controller stops reading after a whole frame is read from memory in order to wait until another image is to be read.

## LCD Controller:

The LCD controller is the part that is responsible for generating correct signals for writing data or commands to the ILI9341. There are timing requirements that should be met which are explained in the laboratory 3 report and earlier sections of this report. Figure X shows a simulation of the LCD controller writing a command from the “registers” component, then reading pixel data from FIFO and writing it to the LCD. For writing, WRX\_GPIO signal is pulled down for a period of 2 cycles and pulled up by a period of 2 cycles. When there is no operation, the WRX signal is kept high. Reading of the FIFO is achieved by pulsing a read request “Rd\_FIFO” signal to FIFO.

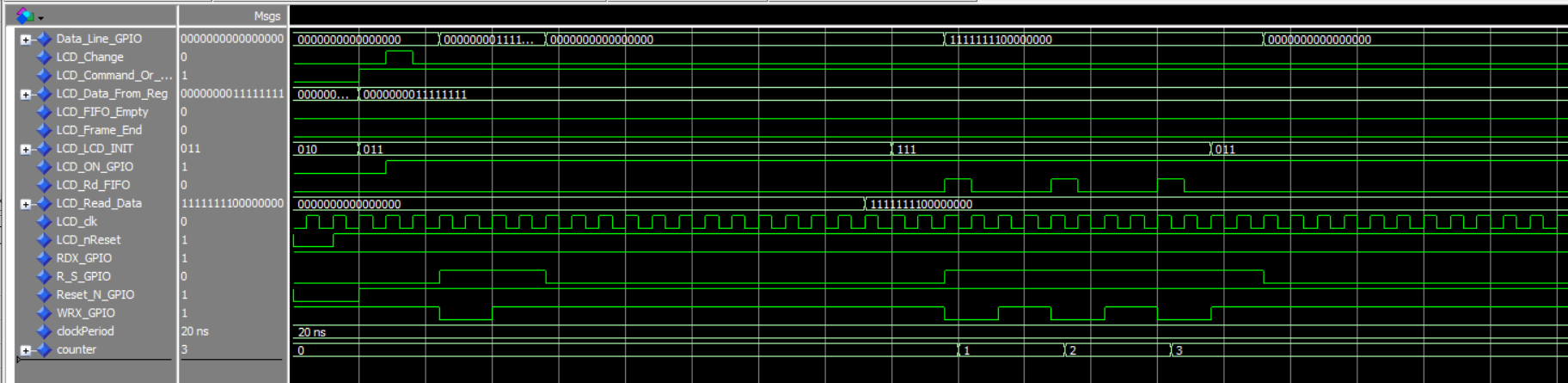


Figure : Full simulation results for the LCD controller.

### Case 1:

Our first case is executing a command for the LCD controller. While executing a command, the LCD controller should write the command data only once. As you can see below, the LCD\_change signal is pulsed for one clock cycle and then the controller pulls the WRX\_GPIO signal low for 2 cycles and alters it to high again for 2 cycles. In this way, the system met the timing requirements mentioned in the “finite state machine of LCD controller” part. Since for no operation case, the WRX signal should be high, after the writing process is completed the signal is staying at high.

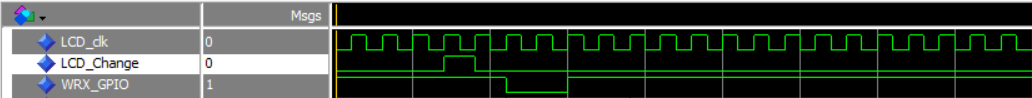


Figure : Simulation for the LCD controller in case 1.

### Case 2:

Our second case is writing the RGB data which is read from FIFO and belongs to a pixel. For writing pixel data, when the controller starts to read from FIFO it should continue as long as the FIFO is not empty. As you can see in the figure below, our system does all of the necessary steps and works properly.

Also, it is important to note that each write cycle is initiated by a read signal pulse which is the Rd\_FIFO signal sent to the FIFO. In each of these cycles, data of one pixel is sent to the LCD.

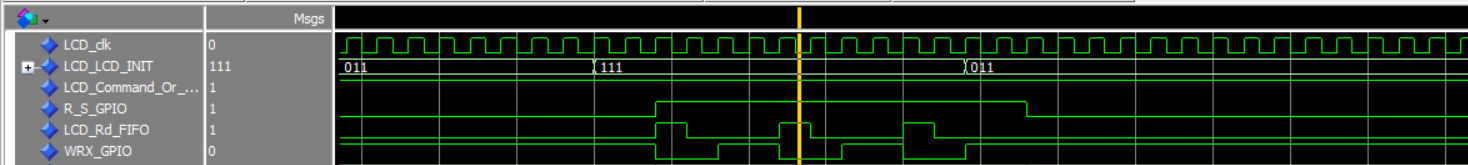


Figure : Simulation for the LCD controller in case 2.

The last thing to notice is when the LCD\_INIT[2:0] signal which comes from register components, is changed from 111 to 011, the operation stops. This is because the LCD\_change signal is 0 at the same time read\_frame bit in the registers component 0 which means there is no read operation. This is shown in the finite state machine of the LCD controller in the IDLE state.

## DMA Controller:

The DMA controller is the part that is responsible for dealing with a large amount of data transfers. For the LCD part, its duty is to get data from memory (SDCARD) where the data that came from the camera part is stored. Figure X shows the complete simulation of the DMA module. In the aforementioned figure, the burst count is taken to be 8 for visualization purposes. Figure X is proof that the system works properly. However, for a better understanding of the operation of the parts, we have divided the operation into cases.

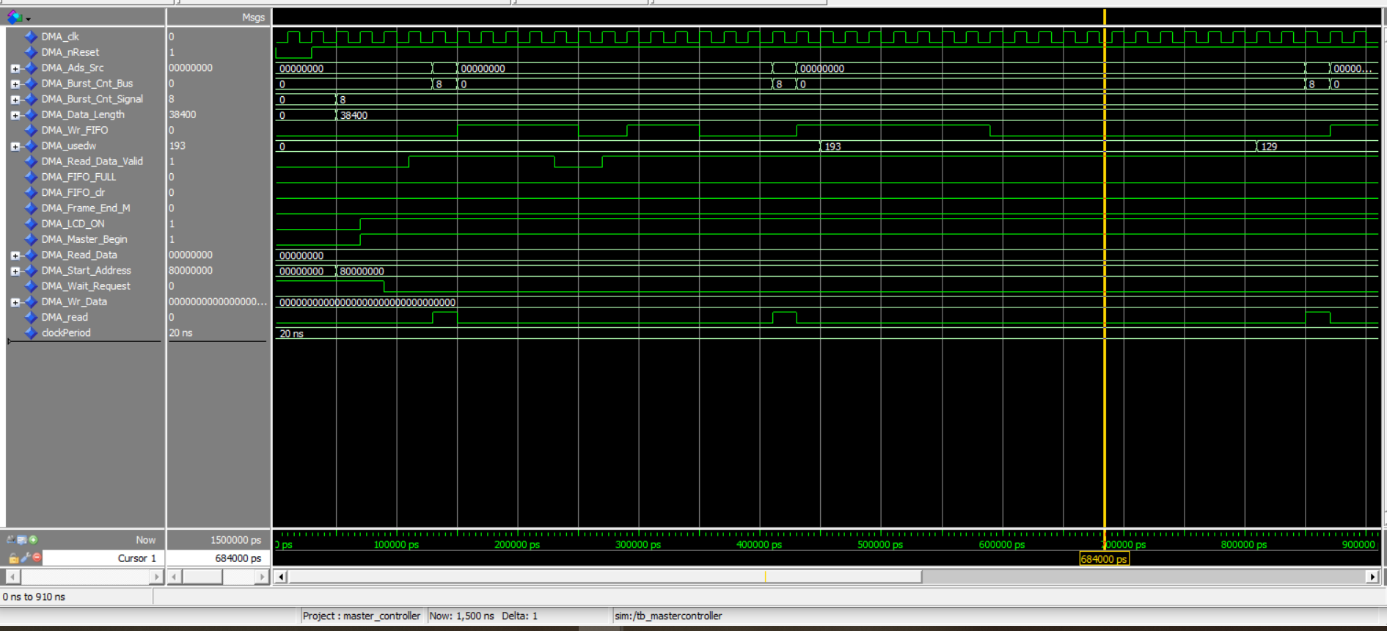


Figure : Simulation results of DMA covering all signals.

### Case 1:

Our first case is the main functionality of the DMA controller which is to write to the FIFO in burst transfers what it reads from the memory. In our design, the burst count is set as 64 but for showing the correct operation on the Modelsim, we have chosen it to be equal to 8.

The DMA component performs correctly. The “DMA\_Adr\_Src” and “DMA\_Burst\_Cnt\_Bus” signals are written for one clock cycle as required by the timing diagram. In order to read the content in memory, the system sends a read request to the system via a signal named “read”. For this operation to be completed, the wait\_request signal should be low. Otherwise, the system keeps the read signal high and keeps the values the same, and waits for the wait\_request signal to become low.

As shown in the figure below, the DMA reads from memory and writes into the FIFO in bursts of 8, given that the bus provides a read\_data\_valid signal as 1. If the read\_data\_valid signal is pulled low in some part of the burst transfer, the burst transfer is interrupted. However, as soon as the read\_data\_valid signal is acknowledged, the transfer continues. This operation can be seen in the figure below. Lastly, it is important to note that after one transfer is completed, the incremented address and new burst count number(for our design it is constant but one can adjust it) are written to the bus. However, the scale in the figure does not allow us to show the incremented address. (Since we did not initialize the address and burst count signals, the simulator shows them as 0 in the graph.)

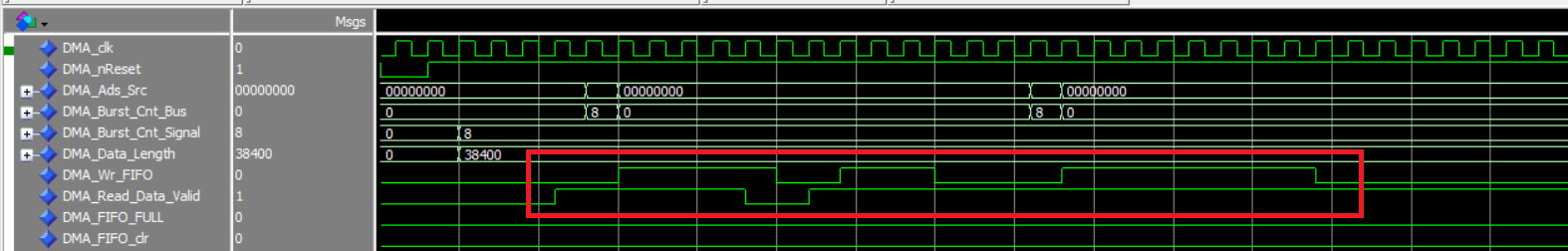


Figure : Simulation for the DMA controller in case 1.

### Case 2:

Actually, this case is discussed a little bit in case 1. We tested whether the systems continue to write data when there is no acknowledgment coming from the bus. As shown below, the system stops writing to FIFO if the bus does not send an acknowledgment.

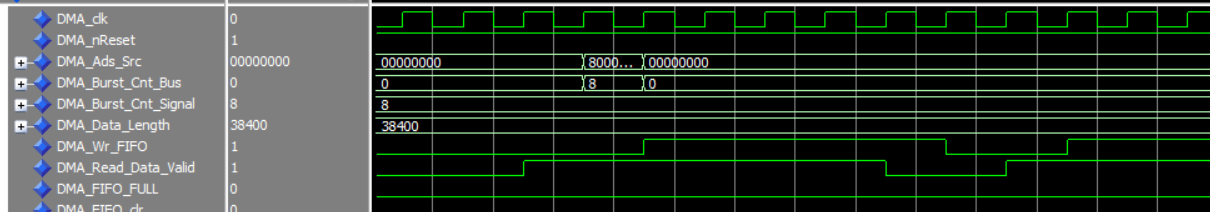


Figure : Simulation for the DMA controller in case 2.

### Case 3:

For the third case, we have considered the situation where the “usedw” signal of FIFO gives a number greater than 192 which means there is no space for one complete burst transfer. We have chosen the upper limit as 192 because we have designed our FIFO as 256 rows deep and burst count as 64; therefore, 192 is the critical value that determines whether FIFO can store one more burst transfer of data or not. For correct operation, in this case, the system should not start a new burst transfer as long as the “usedw” signal is greater than 192. As shown below, the “usedw” signal is changed after the burst transfer is started. Therefore, the system continued to complete the burst transfer. Then, it did not initiate a new burst transfer which shows that the system works properly.

In the next step, we decreased the “usedw” signal again and saw that the system starts a new burst transfer in this case. This shows that the system performs as expected in this case.

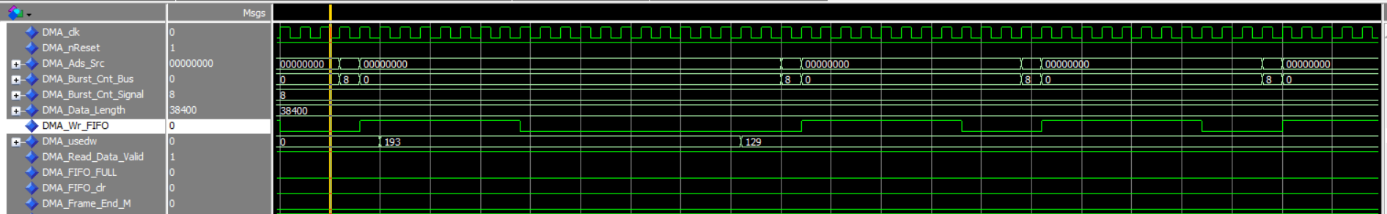


Figure : Simulation for the DMA controller in case 2.

## Full Component:

A picture containing diagram

Description automatically generatedAll the sub-component were combined to form the Full IP Custom component, with a corresponding testbench to make sure of the correct functioning of the component. The resulting simulation is shown in figure below validating the functionality of the component.

# G. Qsys Design for the LCD part

After verifying our design in simulation, we created a component in platform designer. We have added this “Custom IP” component to the Qsys that was provided in the lab template. For this stage of the laboratory, our design does not include the camera part. Since there is not enough space in on-chip memory to store a full frame, we have used SDRAM attached to the FPGA.

In order to use the DDR3 memory, instructors helped us to configure the processor’s memory controller on the development board. However, for not accessing the parts which should not be changed, we have taken some measures. We have used the address span extender provided by the instructors of the course. This module redirects accesses to a specific offset within the DDR3 memory where 256 MB of continuous memory is reserved.

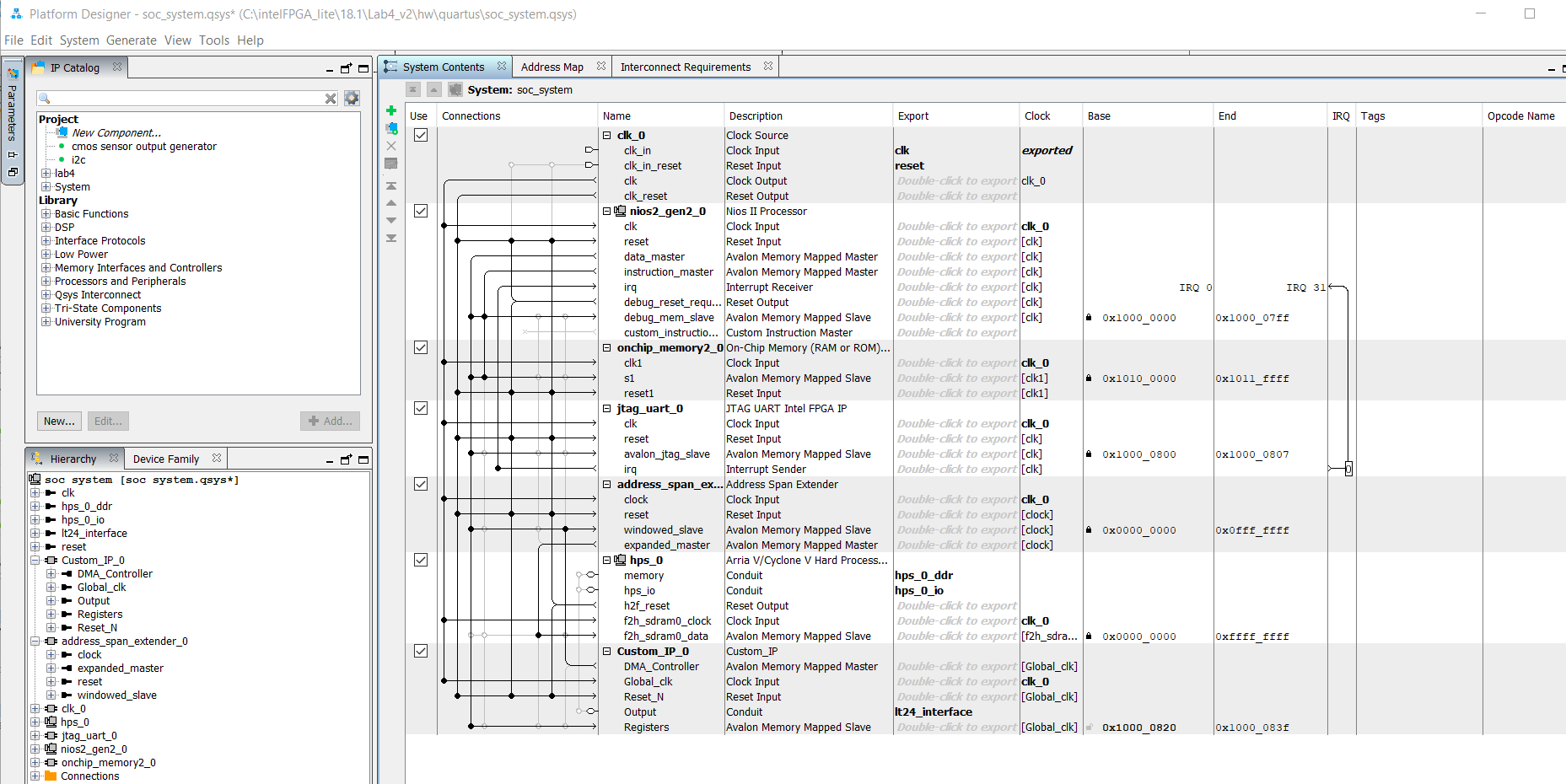


Figure : Qsys Component

# G. Testing on Hardware

In order to test our system before combining two designs(LCD and camera), we tested our system using the host filesystem.

## Writing an Image to the Memory

In order to verify the correct operation of our system, we should have data in the memory. For this purpose, we found an image and converted its pixel data into a text file. Firstly, we have resized the image to have a height of 320 px and a width of 240 px. Then, we extracted its pixel information using a tool named “LCD-Image-Converter”. The extracted information is in the form of 16(R5G6B5) bits.

These hexadecimal 16-bit pixels were transferred and saved into a .txt file, which was then used in Eclipse to write to the memory. This is done by “fscanf” function. At this point, it is important to notice that the information is 16 bits whereas the memory has 32 bits elements. Therefore, we read 2-pixel information at the same time and concatenate them in order to make them compatible with our design of the “memory representation”. After concatenation, the C code wrote the information to the memory. This is achieved using a function called “load\_image”.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Figure : LCD-Image Converter

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Figure 3: load\_image function

## Writing an Image to the Memory

The main function of the C code is provided below.

First, we need to initialize the LCD by sending the required data to the correct addresses of the ILI9341. This initialization step is provided by the instructors in the template. We just write it in a function so that reusing it will be effective. For this initialization step and other subsequent steps, we have used two different functions called “LCD\_WR\_REG” and “LCD\_WR\_DATA”. “LCD\_WR\_REG” is the function responsible for writing the commands whereas “LCD\_WR\_DATA” is responsible for writing the data. Addition to these, there are two other macros to reset and clear the LCD. These functions are “Set\_LCD\_RST” and “Clr\_LCD\_RST”.

The LCD\_Init function contains the commands and data to send as provided by the instructors.

It is important to note that the displayed image is saved in memory using the load image function and using the base address of the HPS which is the address span extender module. The same address and burst count are provided to the DMA controller in order to read the memory content.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Figure : Main function of the C code.

In below, we have provided an image of the displayed picture on the LCD. We have used a photo of Messi and Ronaldo.

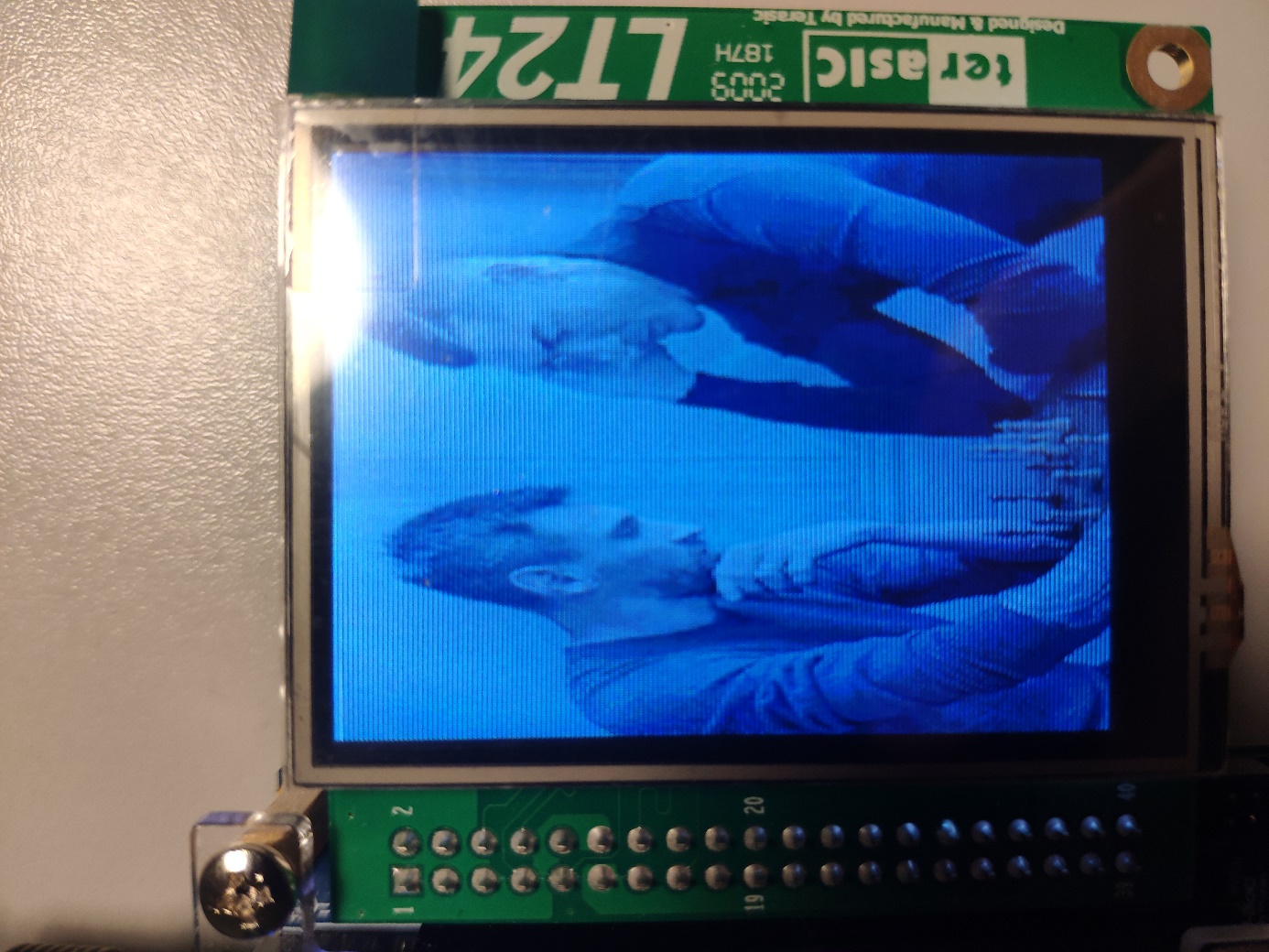


Figure : Picture of properly displayed image on the LCD.