Embedded Systems

Lab 3.0

Richard Ismer

Dimitri Jacquemont

Sina Schaeffler

Nicolas Barker

August 31, 2023

Contents

1 Problem Statement

This lab's objective is to establish a concept and a theoretical implementation in preparation for lab 4.0. The goal here is to create a working camera with LCD display of the video feed. The camera and LCD controller need to be separate modules, and will therefore exchange data using the development board's memory. Appart from the creation of the two components, one of the key challenge will be the memory organisation since the it can not be accessed by two different modules at the same time.

For this project are used the DEO-Nano-SoC board, the LT24 LCD and the TRDB-D5M camera module.

2 High-level Description

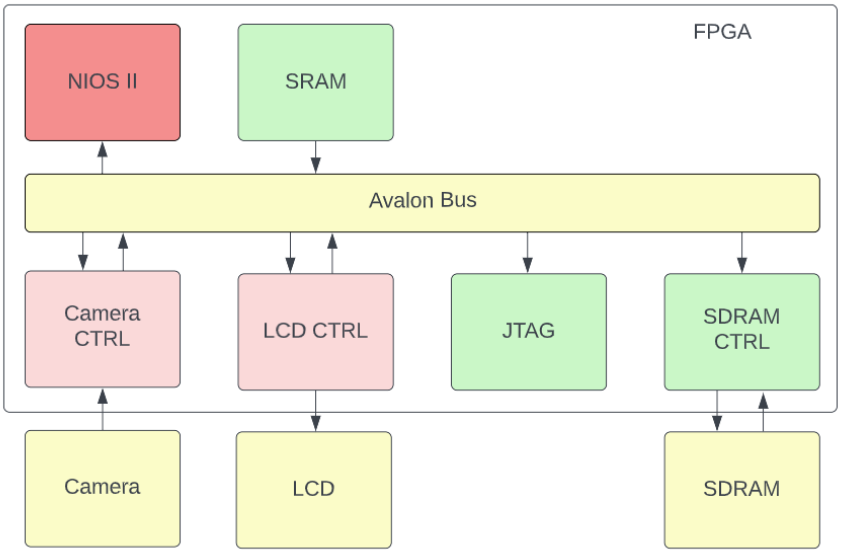

Figure 1: Full system block diagram

In this full system block diagram, the two IP component investigated in this report are the Camera CTRL and the LCD CTRL. Both are slave and master in order to be accessed by the processor and to access the SRAM or SDRAM. All the communications will take place on the Avalon bus.

3 Memory Organisation

The access to memory and its layout are the main point of contact between both components, camera and screen. Therefore, some agreements and specifications are necessary to manage the on-chip memory unit which can be accessed by the processor, camera- and screen controller.

3.1 Pixel format

Pixels will be represented in 5-6-5 RGB little-endian format, there are therefore 16 bit per pixel. This is exactly as the screen needs to receive them, the transformation of the color encoding is completely left to the camera controller.

3.2 Frame representation

A frame is represented in memory as a sequence of valid pixels, without any border. Pixels are a sequence to be displayed from left to right and the lines they from should be displayed from top downwards. The size of a frame is 320 $\times$ 240 pixel (320 pixels large, 240 high). With each pixel using 16 bits, a frame therefore needs 320 $\times$ 240 $\times$ 16 bit or 320 $\times$ 240 $\times$ 2 bytes.

3.3 Buffer management

The buffer management is done by the main NIOS processor via interrupt handlers associated to the interrupts from screen and camera. The processor should have at least 3 buffers available, of which each has a size which is a small multiple of the frame size, ideally 1. It is best when all buffers have equal size, to avoid unequal read and write times. Then the processor attributes to each of both devices a different buffer via their avalon slave interfaces using the corresponding Address register. Upon receiving an interrupt from the screen, the processor sets its Address to the buffer not attributed to the camera which was most recently written by the camera (maybe the same as it just read) and restarts it. Upon receiving an interrupt from the screen, it sets its Address to the buffer where the screen is not currently reading, and which was written the least recently (or never yet), and restarts it.

3.4 Memory access management

Both controllers can access memory via the avalon bus. The camera makes 3200 burst-transfers of 12 consecutive 32-bit (therefore 2-pixel) packets, while the screen does not make any burst transfers. If there is no congestion on the bus the camera will need a minimum of 41600 cycles

to write an entire frame to memory. For 70 frames per second the camera controller will use the bus for around 0.06 seconds (or 6% of the time to display all 70 frames) to perform all the transfers. For the LCD controller one frame will take at least 76800 cycles to be retrieved from memory. For 70 frames per second the bus usage will be about 0.11 seconds (or 11% of the time to display all 70 frames) to perform all the transfer. The total bus usage for memory access is therefore at a minimum of 17% which means that not only camera and screen won't cut each other from getting sufficient access to memory, but more the 50% are left for the NIOS II processor and other memory-consuming devices.

4 TRDB-D5M Camera Controller

The Camera Controller component is responsible for reading the raw camera data, reformatting it into RGB values, and saving it to shared memory frame buffers for the LCD screen to read.

4.1 Register Map

Name	Address	Used bits	Access	Description
CamAddr	0	[31..0]	RW	Address of the buffer in memory to which the camera is currently writing, will write next (if writewritten to 1 since ComAddr changed) or was writing last (if stopped)
CamLength	1	[7..0]	RW	Length (in bytes) of the buffer at CamAddr. This must be a multiple of frameszize = $320 \times 240 \times 2$.
CamComm	2	[0]	RW	Allows to select the stop mode for the future CamStop calls: 0 for SoftStop, 1 for HardStop
CamStatus	3	[2..0]	R	Camera status information as detailed below
CamStart	4	[0]	RW	Start capturing and storing current camera output at next frame. However storage is only possible if the 3 first registers are set appropriately.
CamStop	5	[0]	RW	Stop capture of camera output at end of frame (softStop) or immediately (hardStop). HardStop can cause a bad (that is, partial) frame to remain in memory.
CamSnapshot	6	[0]	RW	When transitioning from 0 to 1 while camera is stopped, a snapshot is taken and stored at the current CamAddr, whether CamComm was toggled or not. It is recommended to set CamLength to 1 frame = $(320 \times 240 \times 2)$ when using this mode.

Table 1: Register map of Camera Controller slave interface

CamStatus

This register reports the current status of the camera controller. Since this register provides different information via its different bits, these are explained below:

4.2 Register Map

4.3 Initialization and software requirements

Software needs to first initialize the camera via its I2C interface. The following registers should be set (mostly in this order) in order to have the camera activated and with a correct image size.

Still on the I2C interface of the camera, the following registers must be set in order to achieve a sufficient frame rate (assuming the camera receives an XCLKIN generated through a PLL on the FPGA which has 25MHz) and restart the output generation.

At the 2 places marked with "*wait*", the program should wait for at least 1 ms in order to let the camera take into account the changes.

Finally, software must set the registers of the memory-mapped avalon slave CameraController component as follows:

After this, the Camera will capture picture in continual mode and the controller will write them to the buffer in memory at the provided address.

However, for more long term functioning another software component needs to be present: An interrupt handler.

Therefore, the Camera Initialization process also must enable interrupts from the CameraController and provide a handler for it which must at least implement the following requirements:

- Remove the interrupt
- Set CamStart to 0
- Provide a new value to CamAddress (also a valid address in On-Chip Memory with free space for at least one frame after it)
- Provide an adequate value to CamLength, which contains the size of the available buffer at CamAddress in Bytes (1 Frame = $320 \times 240 \times 16$ bit). The value must be a multiple of frame size larger than 0.
- Set CamStart to 1

Name	Address	Used bits	Access	Default	Description
CamAddr	0	[31..0]	RW	0	Address of the buffer in memory to which the camera is currently writing, will write next (if writewritten to 1 since ComAddr changed) or was writing last (if stopped)
CamLength	1	[7..0]	RW	0	Length (in bytes) of the buffer at CamAddr. Must be a multiple of frameszize = $320 \times 240 \times 2$.
CamComm	2	[0]	RW		Select the stop mode for future CamStop calls: 0 for SoftStop, 1 for HardStop
CamStatus	3	[2..0]	R	0	Camera status information
CamStart	4	[0]	RW	0	Start capturing and storing current camera output at next frame. Needs the first 2 registers set.
CamStop	5	[0]	RW	0	Stop capture of camera output at end of frame (softStop 0) or immediately (hardStop, 1)
CamSnapshot	6	[0]	RW	0	Enables snapshot mode. In that mode, when starting while camera is stopped, a snapshot is taken and stored at the current CamAddr. It is recommended to set CamLength to 1 frame = ($320 \times 240 \times 2$) when using this mode.
Debug	7	[0]	R	0	Remaining length of the current buffer

Table 2: Register map of Camera Controller slave interface

Ideally, the buffers are not too large in order to have small delays between writes by the camera and reads by the screen (1-3 frames maybe). Also, it is best to have 3 available buffers of equal size and to have the screen read the one the Camera most recently finished writing to, while the camera writes to a buffer which the screen is not currently reading.

Name	Address	Bits	Access	Description
Writing	12	[2]	RW	Currently writing to memory
Stopping	12	[1]	RW	SoftStop was initialized and is not yet finished
Running	12	[0]	RW	Started and si far not stopped (also asserted while stopping)

Table 3: Detailed description of CamStatus register

Name	Number	Bits	Value
Chip_Enable	0x007	[1]	1
Column_Size	0x003	[15..0]	2559
Row_Size	0x004	[15..0]	1919
Row_Bin	0x022	[5..4]	3
Row_Skip	0x022	[2..0]	3
Column_Bin	0x023	[5..4]	3
Column_Skip	0x023	[2..0]	3
Shutter_Width_Lower	0x009	[15..0]	478
Shutter_Width_Upper	0x008	[15..0]	0

Table 4: I2C registers of the camera to be initialized at start by software

Name	Address	Bits	Value
Power_PLL	0x010	[0]	1
PLL_m_Factor	0x011	[15..8]	96
PLL_n_Divider	0x011	[5..0]	24
wait			
Use_PLL	0x010	[1]	1
wait			
Restart	0x00B	[0]	1

Table 5: I2C Camera clock initialization sequence

For using snapshot mode, the camera's snapshot mode exposed via I2C must not be used (except if buffer length is 1, the single provided frames will never be displayed).

The right way to use snapshots with out controller is to stop the camera controller via a soft stop by asserting the 0^{th} bit of the CamStop register (address 4) via the avalon bus, waiting until the CamStatus register shows that the controller is stopped, provide a CamAddress, set CamLength to 1 frame ($320 \times 240 \times 2$ bytes) and enable CamComm. Now assert CamSnapshot to take a snapshot. In order to take another one, set CamComm to 0, CamSnapshot to 0, and then set both back to 1 in the same order. Changing the CamAddress is only necessary if you

Name	Address	Bits	Value
CamAddress	0	[31..0]	Valid address in On-Chip Memory with free space for at least CamLength frames after it
CamLength	1	[7..0]	positive non-zero multiple of $(320 \times 240 \times 2)$
CamComm	2	[0]	1
CamStart	3	[0]	1

Table 6: Camera Controller registers to be initialized by software via the avalon bus

do not want the last picture to be overwritten.

4.4 Internal structure

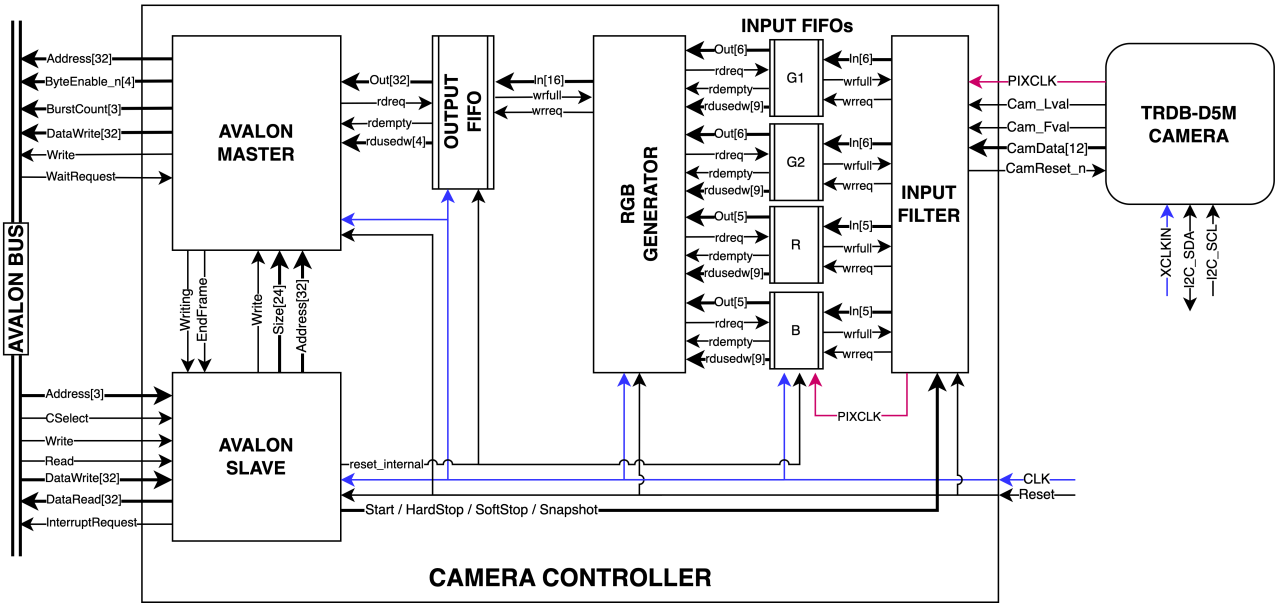

Figure 2: Block diagram of the Camera Controller

4.5 Subcomponent description and state machines

InputFilter

The input filter operates at the frequency of PIXLK, 96 MHz, and is responsible for correctly filling the Color FIFOs with the raw data received from the camera. In order to do this, it uses the lineValid and frameValid signals and a 2-bit colorCounter (line of type 0/1, bit of type 0/1) to keep track of which color pixel is being read currently, truncates the received colors (to 6

bits for green, and 5 bits for red and blue), and ouptuts them to the correct FIFOs (depending again on the colorCounter bits). Also implements snapshot mode by reading one frame and stopping after the next frameValid.

Starting after initialization or after a stop requires to wait for the begin of a frameValid signal. This can be done using an start bit initialized to 1 and set to 1 at every start signal received from the slave. This start bit which blocks output, and the InputFilter clears it when the FrameValid signal goes up.

Other possible input signals are snapshot and stop. On reception of a stop signal, a stopping bit is set until and of frame, which is cleared at frame end and replaced by a stopped bit then, which is cleared on reception of a start signal. When the snapshot signal is asserted, every end of frame triggers the stopped bit to be set.

Color FIFOs

G1, G2 use 6 bit words, while R and B use 5 bit words. Each FIFO can contain an entire line, which is twice the capacity which should be used in average during a cycle, since each color only takes 1 in two pixels in a line containing this color. In total, the green FIFOs G1 and G2 need therefore a capacity §6x320 bits, and the read and blue one of 5x320 bit.

The FIFOS not only allow the InputFilter to store the colored pixel it receives and the RGB generator to wait until a suitable quadruple of colored pixels is available to generate a RGB pixel from it, but also allow to manage the different timings between InputFilter using PIXCLK at 96MHz and RGB generator using clk at 50MHz. Therefore each of these FIFOs uses 2 different clocks: PIXCLK for inputs, clk for outputs.

On reset, the FIFOs need to be emptied. The same reset procedure must be triggered by the avalon slave when a hardStop is required.

RGB generator

The RGB Generator reads from the four Color FIFOs (provided they all contain at least one pixel), concatenating the data into the form: R||G||B (16 bit width 5-6-5). The G1 and G2 values are averaged to determine the G value. The 16 bit RGB pixel data is then written to the Output FIFO.

Output FIFO

This FIFO contains RGB pixels and stores them until they can be sent to the on-chip memory by the avalon bus. Both of its sides work on clk with frequency 50MHz. The RGB transformer

inputs pixels (16 bit words) while the Avalon master retrieves data by 32 bit word for sending them. The master also checks the current content size in order to know whether a full burst transfer is possible. Its capacity should be of at least 32x10*12 bits, to avoid data loss if the master cannot immediately send the next pixels due to reaching the end of a buffer for example.

On reset, the FIFO needs to be emptied and reset. It also has to be emptied by the avalon slave if it receives a hardStop request.

Avalon Master

The Avalon Master provides a master interface to Avalon bus. When there are at least BurstLength (12) * two pixels (32 bits) = (48 bytes) present in the Output FIFO, these will be read and consequently written to the external shared frame buffers. This writing process occurs as follows: the Avalon Master provides the Avalon slave with a high *end_frame* signal when an entire frame has been transferred. If there is no space in the current buffer left, the master sets the *writing* signal to low. The slave in turn sends an interrupt request to the CPU. The CPU responds with the address and length of the next frame buffer to be written to. The Avalon Slave then transmits these instructions to the Master via the *Write*, *Address*, and *Size* signals. The master then initiates a series of burst transfers of length BurstLength (12) as soon as the required amount of data is available in the Output FIFO. A bytesTransferredCounter keeps track of how many bytes of the frame have been transferred. When bytesTransferred-Counter >= (bufferLength-BurstSize-1), the process starts again with the *writing* signal going low. To ensure stability, if the amount of remaining pixels to send is less than the BurstLength, a shorter burst will be sent of the remaining pixel length. EndFrame is also pulsed high to the slave for one cycle every time a frame length of data has been transferred.

Avalon Slave

The avalon slave's behaviour can be described in terms of reaction to its input signals, especially the writable registers:

When CamStart goes up, the current value of CamAddr and CamLength are transferred to the Address and Size signals it gives to the master. At same time, all FIFOs are reset and the start signal to InputFilter is set for 1 cycle. Also the Write signal to the master is set to 1, and the running and writing bit in CamStatus to 1. While stopping, start signals are ignored.

Otherwise CamLength and CamAddr changes are only stored in the corresponding register without any particular action.

On CamStop, if CamComm is 0, it triggers a softStop. This means it outputs a SoftStop signal is outputted to InputFilter for 1 cycle. At same time, the stopping bit is the CamStatus register is asserted. Upon reception of the next endFrame Signal from the Avalon master, the writing, running and the stopping bit in the CamStatus register are removed. If the buffer size

is reached with the end of that frame, no IRQ is generated.

On CamStop, if CamComm is 1, it triggers a hardStop. This means it outputs a HardStop signal is outputted to InputFilter for 1 cycle. At same time, the running bit is the CamStatus register is set to 0, and a reset signal is emitted to all FIFOs.

When the writing signal from the Avalon Master is going to 0, an IRQ is emitted, the writing bit is set to 0 in the CamStatus register and the write signal to the avalon master is set to 0.

When CamSnapshot is set, the snapshot signal to InputFilter is set to 1. It is set back to 0 when the CamSnapshot register is set to 0.

On reset signal, it resets to its initial state.

4.6 External interfaces

The Camera controller's interfaces with the external world can be grouped in four main categories:

- **General signals** clock and reset
- **Avalon master signals** for interfacing with memory
- **Avalon slave signals** for receiving instructions from the NIOS processor and setting interrupts
- **Camera signals** for interfacing with the camera, including clock settings and data reception

The signals in the Camera signal group must all be linked to the corresponding pins of the GPIO to which the camera will be branched, in our case GPIO_1. Then the signals are mapped to pins as follows:

GPIO_1	**signal**
0	Cam_Pixclk
[12..1]	Cam_Data[0..11]
16	Cam_Reset_n
20	Cam_Lval
21	Cam_Fval

Table 7: Camera Controller - Signals to camera through GPIO_1

On top of that, the Camera needs an input clock XCLKIN provided at bit 15 of the GPIO. This clock is expected to be at 25MHz in the initialization software, therefore a PLL must be used to provide it. However, this PLL is be external to our component.

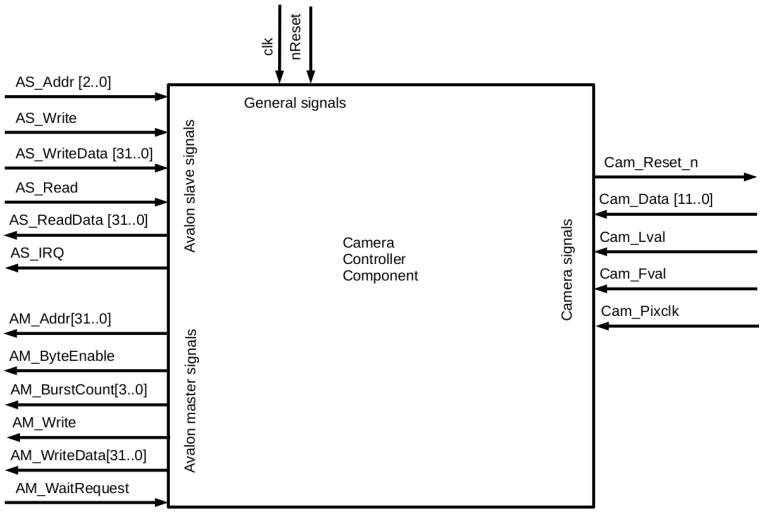

Figure 3: High-level Block diagram of the Camera controller containing all used signals

5 LT24 Driver

5.1 Block Diagram

The LT24 driver and its sub-components will be implemented as follows

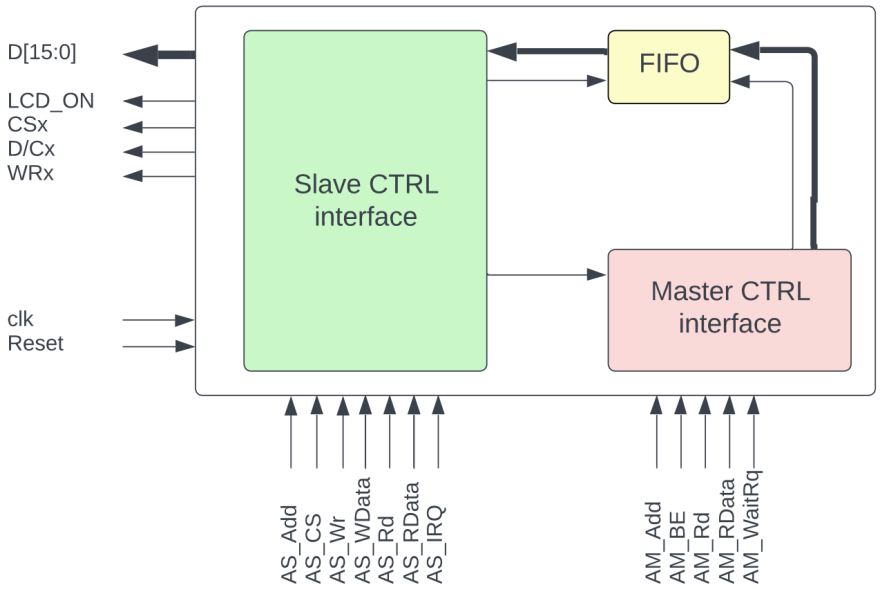

Figure 4: Block Diagram of the LT24 Driver

Sub-components details

Slave interface/ LCD CTRL

This driver's slave interface is in charge of receiving the commands coming from the NIOS II processor (software written commands that can be found in the register map), communicating the buffer length and address to the master interface (to fetch the data from memory), and communicating the display parameters to the LCD CTRL in the same module. In order to get new buffer information (address), it will be sending an interrupt each time it finishes sending a frame to the LCD. This module also acting as the LCD CTRL will be receiving data from the FIFO, and will be sending commands to the LCD to start of stop the communications.

Master interface

This driver's master interface receives the buffer's information from the slave interface (buffer address and length), and reads display data from memory through Avalon read accesses. It will be sending the previously fetched display data to the module's FIFO in order to guarantee constant flow of data to the LCD CTRL.

FIFO

The FIFO will be configured through MegaWizard from Quartus II. It allows for a constant flow of data between a data producer and data receiver, and it is needed because the master interface will be fetching the data asynchronously, while the LCD CTRL will be needing a continuous data flow. It will be sending a signal when the FIFO is almost full or empty in order to adjust the data flow from the master interface.

5.2 Finite State Machine

To be able to send different commands and data to the LCD we need a state machine.

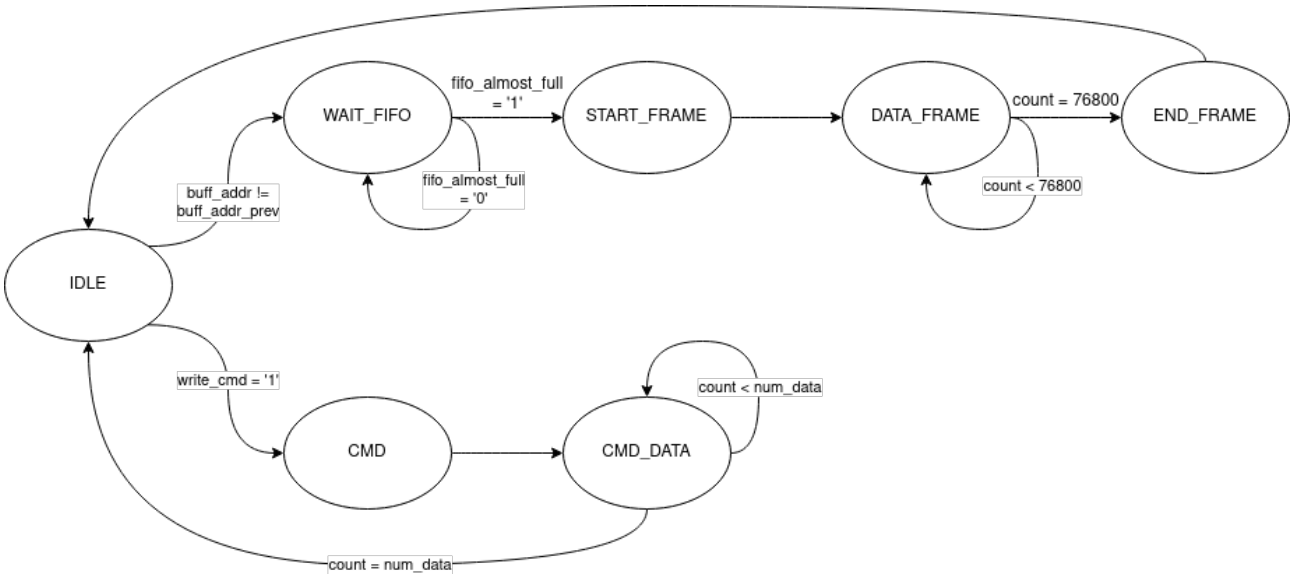

Figure 5: State Machine of the lcd controller

We can find below the explanation on the different states:

IDLE The controller is waiting either for a command to be send from the processor or for the buffer address to change, meaning a new frame is available.

WAIT_FIFO The controller waits for the FIFO to be almost full. This allows a continuous data flow from the FIFO to the LCD.

START_FRAME The controller sends a start of frame command. This state takes 4 cycles to execute.

DATA_FRAME The controller sends a data from the FIFO while there is some in it. This state takes 4 cycles to execute.

END_FRAME The controller sends a nop command to the LCD to finish the sending of the current frame. This state takes 4 cycles to execute.

CMD The controller sends a command located in the CMD_command register to the LCD. This state takes 4 cycles to execute.

CMD_DATA In this state the controller sends the data associated to the command to the LCD. This state is repeated CMD_numData times and takes 4 cycles to execute.

5.3 Register Map

Name	Address	Used bits	Access	Description
bufferAddr	0	[31..0]	RW	Address of the buffer in memory to which the LCD is currently reading. Writing the same value will not trigger a new frame to be read
bufferLength	4	[31..0]	RW	Length (in pixels) of the buffer at bufferAddr
CMD_command	8	[31..0]	RW	Commands to be sent to the lcd
CMD_numData	12	[5..0]	RW	Number of data to be sent with the command associated
status	16	[1..0]	RW	Writing to this address begins the command and data transfer, and reading informs of the module's current state (command state at address 0 and frame state at address 1)
CMD_data	20	[31..0]	W	N data to be sent with the command, writing to this register adds a data to the current command

Table 8: LCD driver - Register Map

5.4 LT24 sequence command

Multiple commands have to be written to the LCD to be able to display an image. After a reset the following commands have to be send to the LCD controller to send them do the LCD. The commands are:

Name	**Command**	**Data**
Exit sleep	0x0011	
Power control B	0x00CF	0x0000 0x0081 0x00C0
Power on sequence control	0x00ED	0x0064 0x0003 0x0012 0x0081
Driver timing control A	0x00E8	0x0085 0x0001 0x0798
Power control A	0x00CB	0x0039 0x002C 0x0000 0x0034 0x0002
Pump ratio control	0x00F7	0x0020
Driver timing control B	0x00EA	0x0000 0x0000
Frame control	0x00B1	0x0000 0x001B
Display function control	0x00B6	0x000A 0x00A2
Power control 1	0x00C0	0x0005
Power control 2	0x00C1	0x0011
VCM control 1	0x00C5	0x0045 0x0034
VCM control 2	0x00C7	0x00A2
Memory access control	0x0036	0x0008
Enable 3G	0x00F2	0x0000
Gamma set	0x0026	0x0001
Positive gamma correction	0x00E0	0x000F 0x0026 0x0024 0x000B 0x0000
		0x0008 0x004B 0x00A8 0x003B 0x000A
		0x0014 0x0006 0x0010 0x0009 0x0000
Negative gamma correction	0x00E1	0x0000 0x001C 0x0020 0x0004 0x0010
		0x0008 0x0034 0x0047 0x0044 0x0005
		0x000B 0x0009 0x002F 0x0036 0x000F
Column address set	0x002A	0x0000 0x0000 0x0000 0x00EF
Page address set	0x002B	0x0000 0x0000 0x0001 0x003F
Pixel format set	0x003A	0x0055
Interface control	0x00F5	0x0001 0x0030 0x0000
Display on	0x0029	
New frame	0x002C	from FIFO
End frame/NOP	0x0000	

Table 9: LT24 sequence command

The processor is sending these commands except for the two last ones that are sent by the state machine. To send the commands to the controller, the processor has to follow these instructions:

- Send the command to the register CMD_command
- Send the number of data to the register CMD_numData
- Send the data one after the other to the register CMD_data
- Set the CMD_write bit to '1' by writing in it
- Wait for the CMD_write bit to be '0' to send a new command

Sending the data, num_data and command in different order has unknown behavior.

5.5 Top-level Block Diagram

Since the LCD screen has to be connected to a full GPIO, in our case the GPIO_0, we will detail what each connection denotes and what are there purpose. The connector has 2x20 pins with many of them unused because we won't be using the touchscreen and therefore we don't need the ADC pins.

GPIO_0	**signal**
[9..6]	Data_[0..3]
11	VCC5
12	Ground
13	Read_n
14	Write_n
15	Register Select - Data/Command_n
[27..16]	Data_[15..4]
28	Chip Select
29	VCC3.3
30	Ground
38	Reset_n
40	LCD_ON

Table 10: LCD driver - Signal to output connections

6 Conclusion

In this lab, a theoretical implementation was suggested. This theoretical implementation will be useful in lab 4 for the practical implementation of this system.