Lab 4.0 Camera, LCD & VGA implementation

Zehao Chen, Xinyu Zhu
Tz-Ching Yu, Hsu-li Huang, Hong-Bin Yang

January 5, 2023

1 Introduction

In the previous laboratory assignment, we were required to design a custom master interface to handle a large amount of data transfer. We utilized the LT24 LCD display and the TRDB-D5M camera as peripherals for this system. In this report, we will detail the steps taken to implement and integrate these peripherals into a functional frame acquisition and visualization system. We will also describe any modifications or improvements made during the development process and provide all necessary information about the different components of our system.

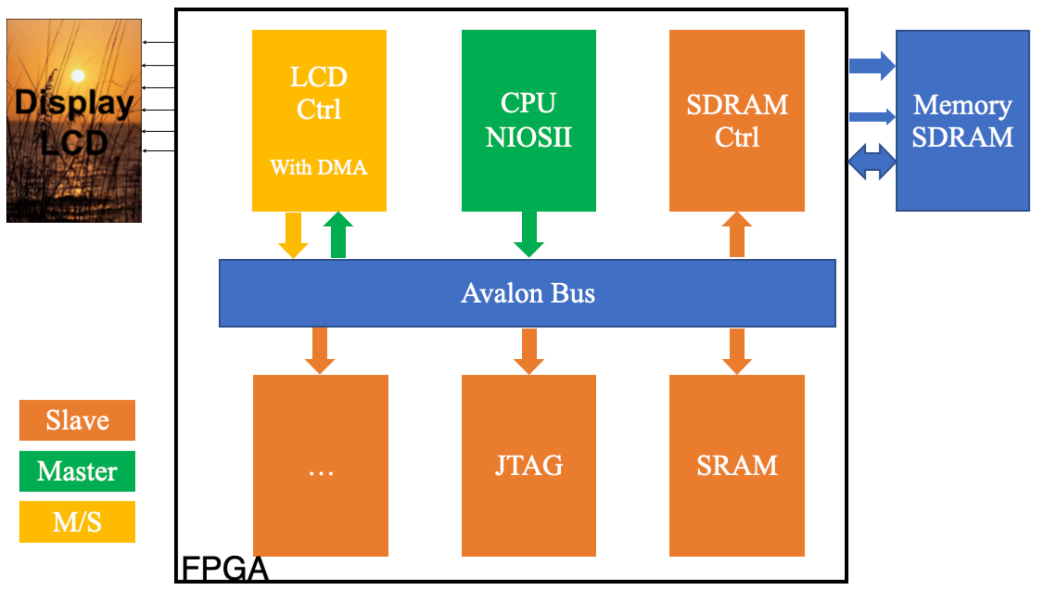


Figure 1: The diagram of System block design.

2 LCD controller

Inside the LCD Ctrl block, there are 4 subblocks: LCD Control, Registers, Master Control and FIFO. It includes LCD interface, Master Interface and Slave Interface(Registers). Registers work as Avalon Slave which is controlled by NIOS II.It can get data from NIOS II and provide data to NIOS II. Master control works as Avalon master which can give command to memory and get data stored in memory. FIFO is used for organizing and managing data which transfers from Master Control to LCD Control. LCD control is used to send command and pixel information to LCD device. Details are shown in Figure 2.

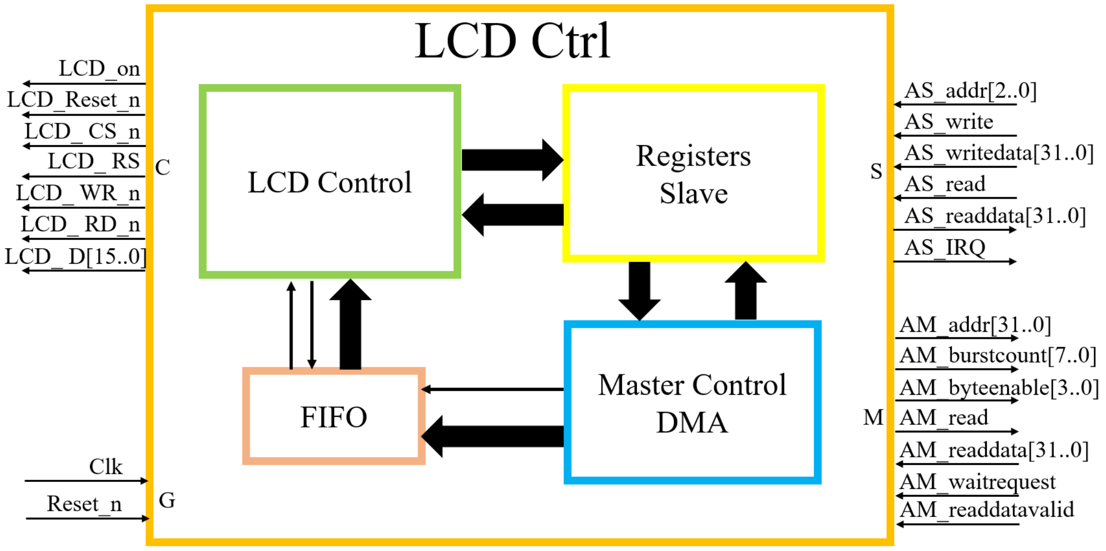
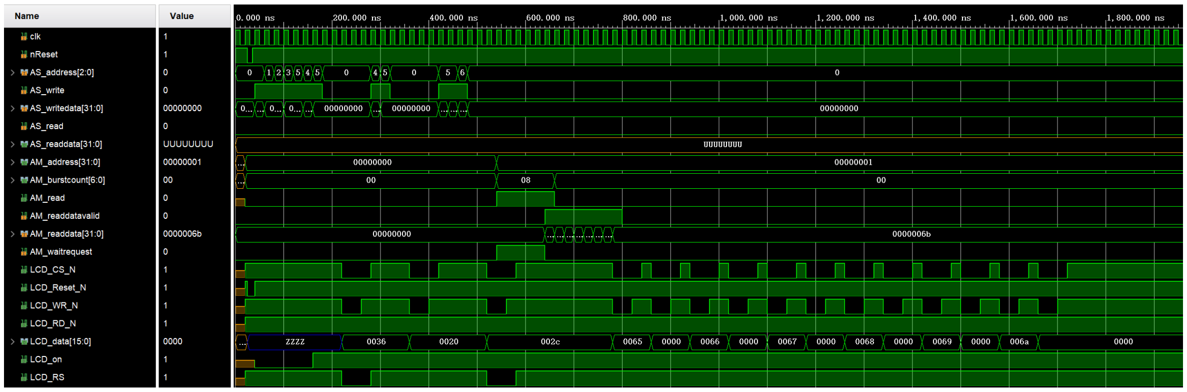


Figure 2: LCD control IP block diagram.

The results of the top level LCD master control test are shown in Figure 3. To begin, data and address are written to the Avalon Slave. The registers are then enabled, allowing the Master control to receive signals, including command (2C), data, and address, from the Avalon Bus using a burst read transfer protocol. The data is then passed to the FIFO, which in turn passes it to the LCD control module. The LCD control module then sends the data to the LCD display for display.



Name	Value
clk	1
nReset	1
AS_address[2:0]	0
AS_write	0
AS_writedata[31:0]	00000000
AS_read	0
AS_readdata[31:0]	UUUUUUUU
AM_address[31:0]	00000001
AM_burstcount[6:0]	00
AM_read	0
AM_readdatavalid	0
AM_readdata[31:0]	0000006b
AM_waitrequest	0
LCD_CS_N	1
LCD_Reset_N	1
LCD_WR_N	1
LCD_RD_N	1
LCD_data[15:0]	0000
LCD_on	1
LCD_RS	1

Figure 3: Test result of LCD Ctrl(top level).

2.1 LCD Interface

The block shown in Figure 4 plays a crucial role in enabling communication with the LCD display. It is controlled directly by the Register, which is controlled indirectly by the NIOS II processor. The Register sends command, data and choosing signals to the LCD control block. Also, various LT24 LCD control signals such as LCD_on, LCD_Reset_n, and LCD_CS_n are sent to the LCD device by LCD control. These signals are used and processed by the LCD device (ILI9341) in order to configure the device. It is important to note that within this block, pixel data are transferred in 32-bit chunks through a FIFO (First-In, First-Out) buffer command and command data are transferred in 16-bit chunks through register.

The finite state machine (FSM) diagram of LCD Control is shown in Figure 5. At the start of its operation, the system is in the IDLE state, where it initializes its registers and remains inactive until it receives further instructions. When the Reg__LCD__en is set to 1 and Reg__LCD__cmd__or__data is set to 01, the system transitions to the Send__cmd state, where it begins transmitting commands to the LCD. Once it has finished transmitting the command and the command is 2C, the system automatically moves to the FIFO__wait state, where it is prepared to display images.If the command is not 2C, the system will go back to idle state. If the FIFO__Almost__Empty signal is 0, indicating that the FIFO is full and ready to send data to the LCD display, the system transitions to the Send__pix state. In the Send__pix state, the system continually sends data to the LCD display until the FIFO is empty, at which point it returns to the WAIT state. When the Reg__LCD__en is set to 1 and Reg__LCD__cmd__or__data is set to 10, the system transitions to the Send__cmd__data state, where it begins transmitting data to the LCD. Then go back to idle state.

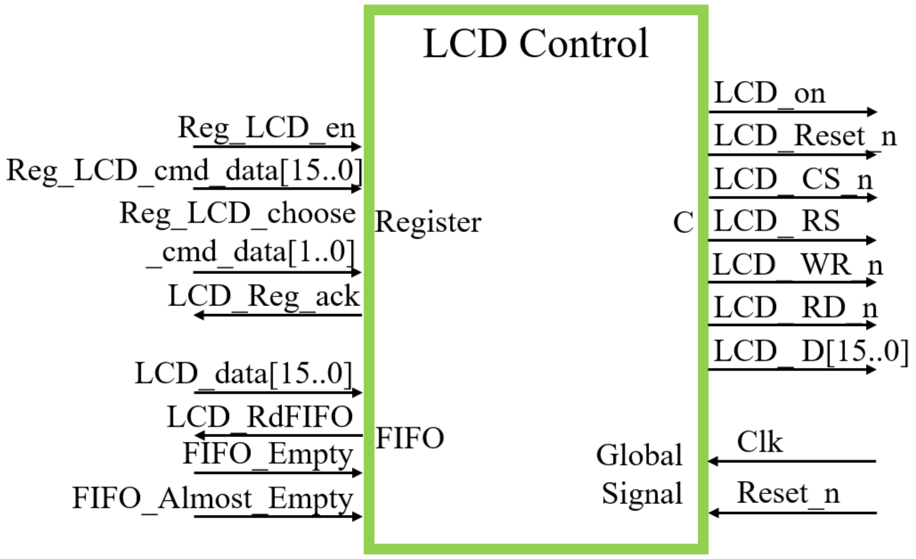


Figure 4: The subblock diagram of LCD Control.

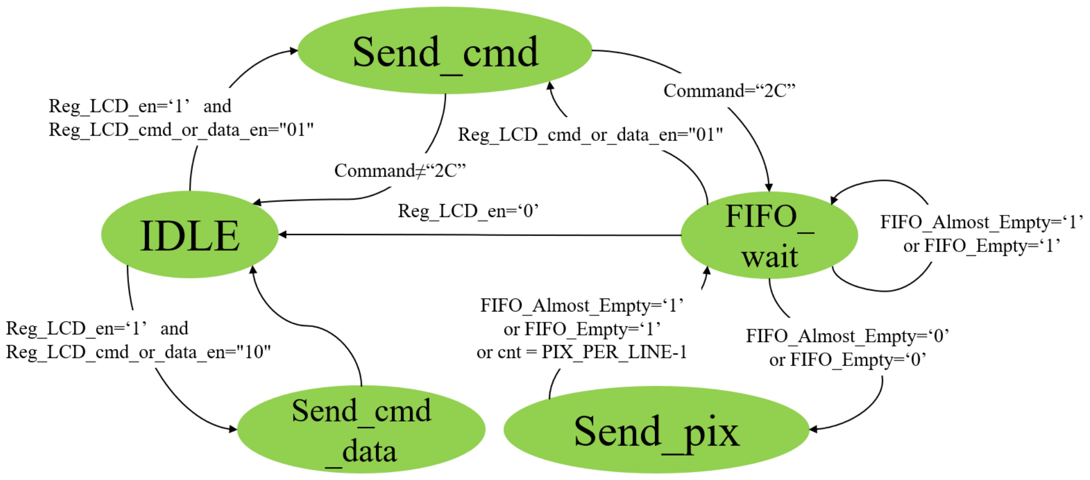


Figure 5: The finite state machine (FSM) diagram of LCD Control.

The test result is depicted in Figure 6. The initial state of LCD control is idle. First, we write a command (Memory Access Control: 0X36). The state of LCD data becomes send__cmd then goes back to idle. Then we write the corresponding data to exchange row and column. The state of LCD data becomes send__cmd__data then goes back to idle. Last we write the Display Command(0X2C). The state of LCD data becomes send__cmd then goes to FIFO__wait. Until FIFO__empty=0, the state becomes send__pix. Here, the command and data do not have special meaning. They are just used to check whether the LCD will work as we image or not.

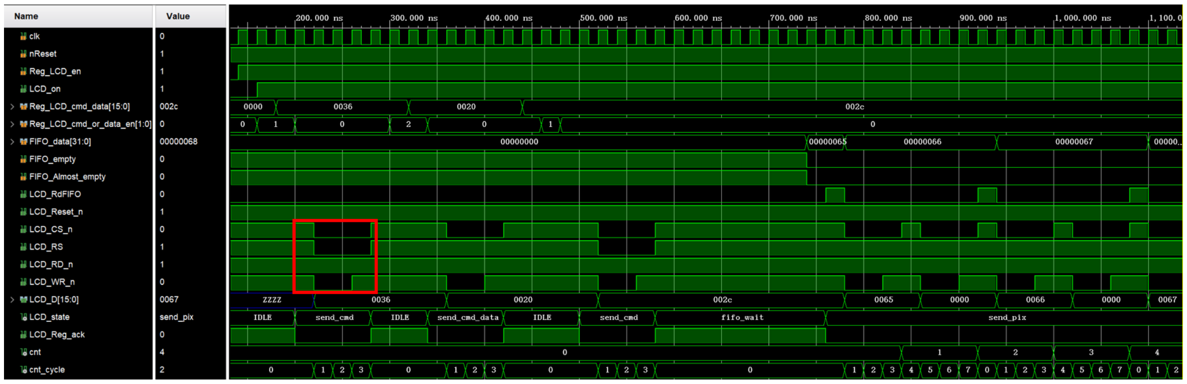
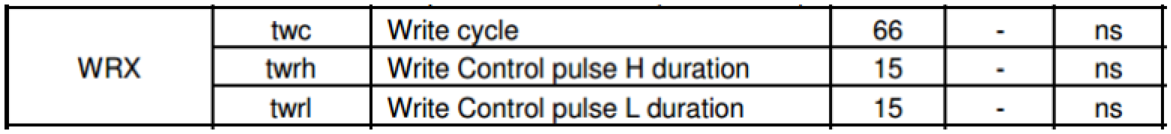


Figure 6: The test result of LCD Control.

Because the minimum write cycle of WRX signal is 66 ns (Figure 7). And the working frequency of the broad is 50 MHz, which means the clock cycle is 20 ns. So in the program we use 4 clock cycles (80ns) to write data into LT24 LCD device and set write pulse H duration and write control pulse L duration to 40 ns (shown in the red square in Figure 6).



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

Figure 7: Detailed timing diagram.

2.2 Master Interface

This block(shown in Figure 8) is controlled by Register. Register gives Master Control information about how to read data and where to read data.

Also, Register sends enable signal to start it. Then LCD Control can give command to DRAM Memory and receive the data from it through Avalon Bus. Then data is sent indirectly to the LCD interface through FIFO. The usage of a master makes it possible to bypass the processor.

The master interface works as Direct Memory Access. The initial State is IDLE, when it receives signal go equals 1 from slave interface, it goes to next state START. In this state, address is read and then goes to next state WAIT. In WAIT state, master control waits until FIFO has transferred all the WaitRequest signal to 0 and the state goes to READ. In Read state, FIFO will reads data from DRAM through Avalon Bus, when FIFO reads all the burst data, it will go to START state again.

The results of the Master control test are presented in Figure 10. The initial state is IDLE. When the Master control receives an enable signal from the registers, it transitions to the next state, BurstStart. In BurstStart, the Master control checks whether the FIFO is full or not. If the FIFO is not full, the Master control proceeds to the BurstWait state, where it waits for signals from the Avalon Bus, including the address and burst count. Once these signals are received, the Master control transitions to the BurstRead state, where it uses a burst read protocol to read data from the Avalon Bus and transfer it to the FIFO for storage.

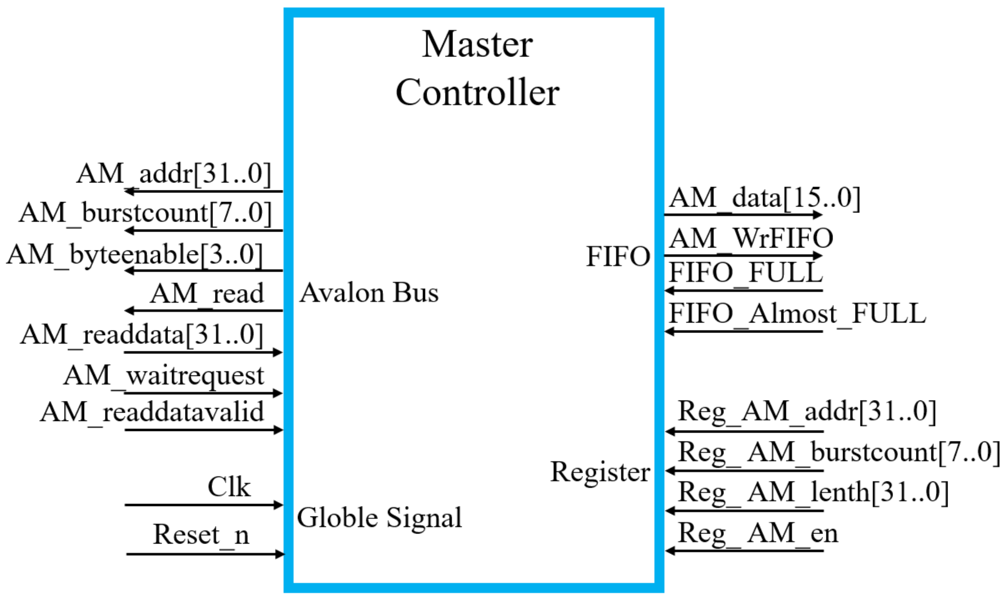


Figure 8: The subblock diagram of Master Control.

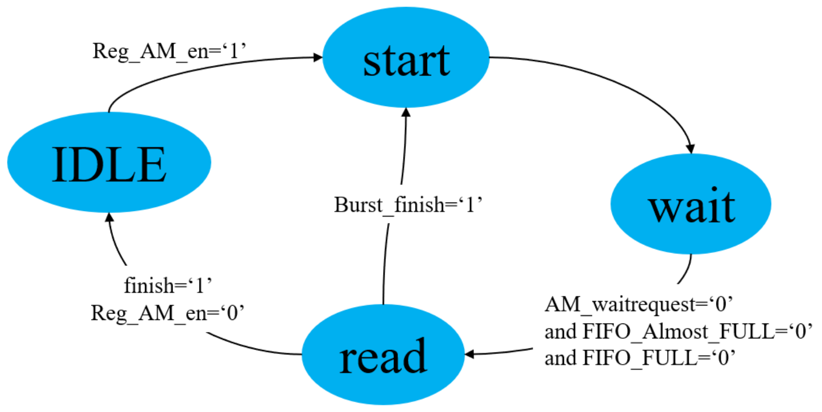


Figure 9: The Finite state machine of Master Control.

2.3 Avalon Slave/Register

This block shown in Figure 11 receives the command coming from the processor. The LCD control IP and master control IP cannot work entirely on their own and need to get instructions from the processor. The instructions can then be used to control the two other blocks (LCD control and Master). Register transfers command signal and enable signal to LCD Control and information about how and where to read data to Master Control. Also, there are feedback signals which tell Registers whether LCD Control and Master Control are busy or not.

In this system, Register works as slave interface inside the LCD Ctrl part. NIOS II gives Register setting signals, such as start address of Memory, length of data and burstcount number. Register map is shown in Figure 12.

The test results for the registers are depicted in Figure 13. The registers continuously receive data and address from the Avalon Bus. Based on the address map, the registers provide specific signals to the Master control or the LCD control module as needed.

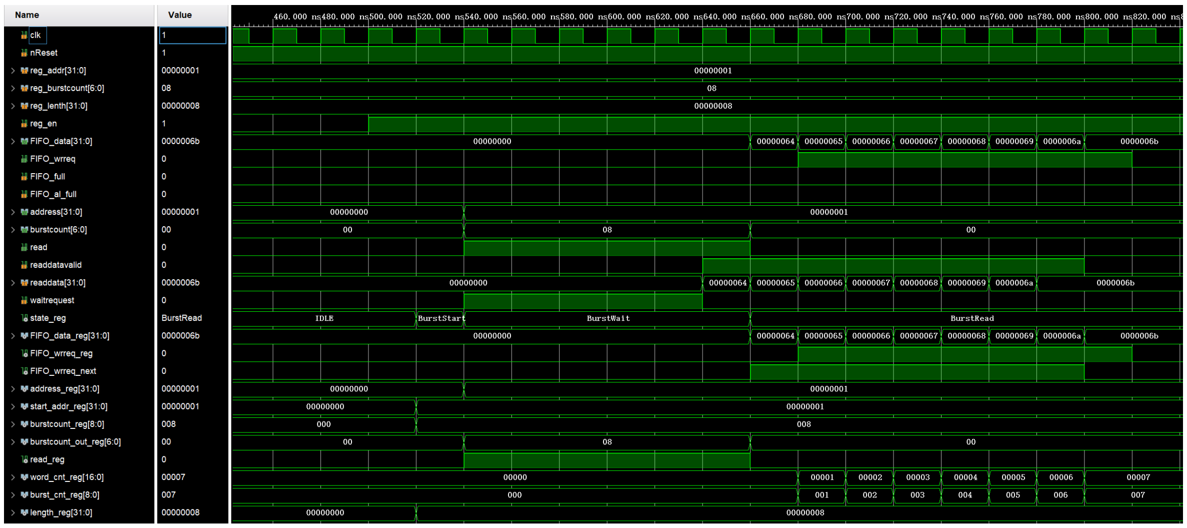


Figure 10: The test result of Master Control.

2.4 FIFO

FIFO is generated in Quartus by using IP catalog. Its block is shown in (Figure 14).we use a common clock for both reading and writing FIFO, set the width of FIFO to 32 bits and set the depth of FIFO to 1024 words. Also we add 2 optional signals: almost full and almost empty. The most important setting is to choose show-ahead synchronous FIFO mode, which means the data becomes available before 'rdreq' is asserted.

3 Camera Controller

3.1 Overall Architecture

The camera controller is composed by several components, including an Avalon master acting as a DMA module responsible for transferring pixel data to memory, an Avalon slave which should be programmed by the processor to configure the controller, and a camera interface handling the acquisition of camera pixels. The block diagram is shown in Figure 15.

The camera interface will directly receive pixels from the camera module, and the received pixels will be put in a temporary storage, as we will assemble the 4 pixels, i.e. R, G1, G2, and B in one composite pixel. The assembled pixel will then be transferred to the Avalon master where each pixel will then be sent to memory by the DMA. The function of the controller can be summarized as a finite state machine and the according diagram is shown in Figure 16.

The register map is listed as table 1, where the user should specify the base address where the pixel value will be stored and the "Start" flag to start capturing images.

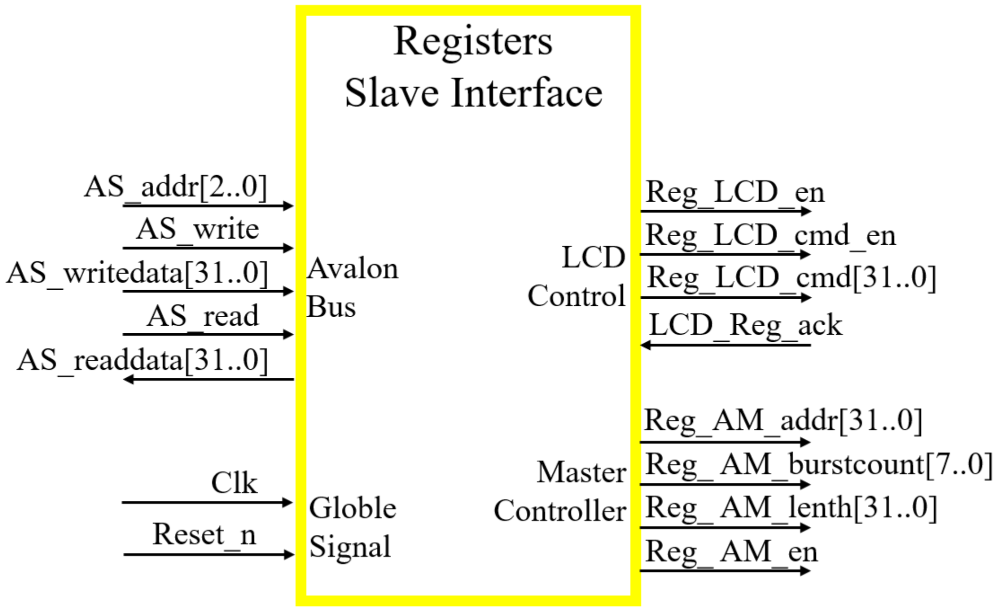


Figure 11: The subblock diagram of Register.

Address	Register	Reset Value	Description
0	Start	0	Set to start capturing
1	Base Address	0	Specify the base memory address to write to

Table 1: Camera Controller Register Map

3.2 Camera Interface

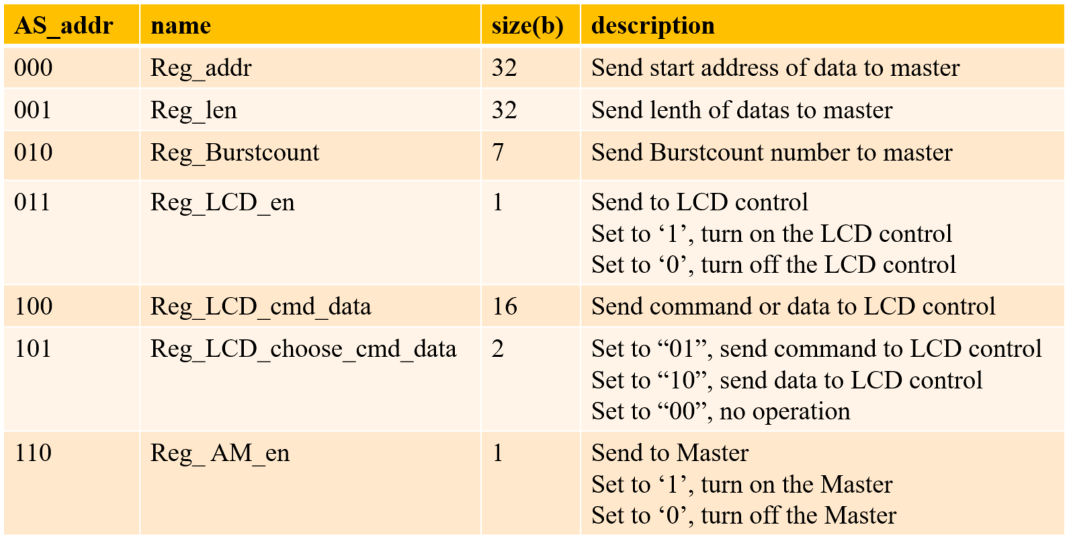
The overall behavior of the camera interface can be summarized as two FSMs. The corresponding state diagram is shown in 16. The FSM shown in the lower part of the figure keeps track of the current line and column being read. The component is set into motion when **start** is asserted. It will then waits for **frame__valid** and **line__valid** to become 1 which indicates the data from the camera is valid pixels. The FSM in the upper part is responsible for facilitating the transfer of pixels from the camera interface to the Avalon Master. It will block until the Avalon Master acknowledges the previously transferred pixel. Specifically, the component comprises mainly of two FIFOs and a circuit to combine four single-color pixels into one RGB pixel. The high-level circuit is shown in 17. The data from camera is routed to merging FIFO or registers directly. The two green pixels are averaged and combined with pixels of two other pixels. The combined pixels are fed into synchronization FIFO to synchronize two clock domains.

3.2.1 Demosaicking

At this stage, the image is a single channel image with size 640×480 in the Bayer pattern. To reduce the size to 320×240 and demosaicking the Bayer filter array, four pixels are combined in one by taking R and B values directly and averaging G_1 and G_2, as shown in Figure 18.

3.3 Pixel Output

Although the camera uses 12 bits to store the pixel value for each color, since the LCD takes only 16 bits as input (5 bits red, 6 bits green, and 5 bits blue), we decided to discard the unused bits by keeping only the most significant bits. This process is performed in the camera interface module.



AS_addr	name	size(b)	description
000	Reg_addr	32	Send start address of data to master
001	Reg_len	32	Send lenth of datas to master
010	Reg_Burstcount	7	Send Burstcount number to master
011	Reg_LCD_en	1	Send to LCD control Set to ‘1’, turn on the LCD control Set to ‘0’, turn off the LCD control
100	Reg_LCD_cmd_data	16	Send command or data to LCD control
101	Reg_LCD_choose_cmd_data	2	Set to “01”, send command to LCD control Set to “10”, send data to LCD control Set to “00”, no operation
110	Reg_ AM_en	1	Send to Master Set to ‘1’, turn on the Master Set to ‘0’, turn off the Master

Figure 12: The register map.

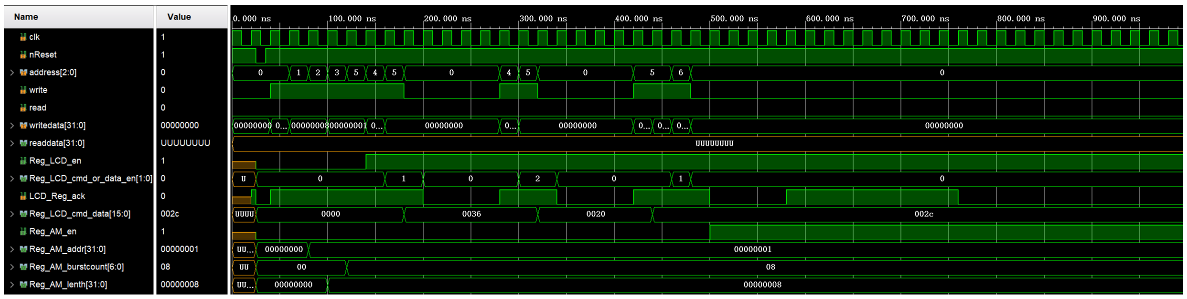


Figure 13: The test result of register.

3.3.1 Verification

We verified the component with a simple testbench. The simulation starts by resetting the interface and setup pixel patterns. We then checked the result manually. The following waveform Figure 20 is the result of the simulation. As we can observe the output RGB pixel follows the expected output of the four single-color pixels combined, we are confident that this component works as expected.

3.4 Avalon Master/Slave

The Avalon slave is used to read the variable from the NIOS controller and update the internal register. In our design, there are only two registers, Start and Base_Address, which can be assigned.

On the other hand, the Avalon master is responsible for putting the composited pixel value into the desired memory location. The camera interface will send the 16-bits pixel value one by one from a FIFO queue, and the Avalon master will write the value into the corresponding memory address based on the base address specified by the user, the frame buffer flag (using 1st buffer or 2nd), and the current offset counted by itself.

Since we always write 16-bits into the memory, the burst count is always set to 2, and the byte enable is set to ”1100”. After the pixel value is written into the memory, an ack flag is set to inform the camera interface that new data may be sent.

3.4.1 Testbench output

We made a testbench to test our Avalon master/slave for debugging, ensuring everything works properly. In the beginning, *Start* and *Base_Address* are set to the desired value. After that, new data start transfer. When a new frame is served, *am_new_frame* is set to 1 to inform the Avalon master

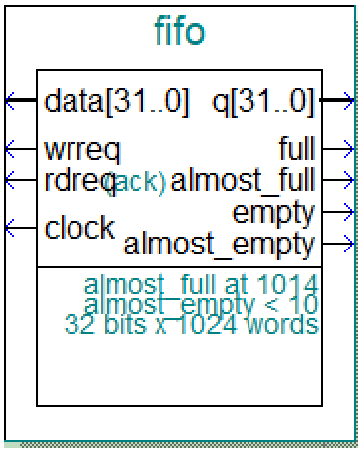


Figure 14: The subblock diagram of FIFO.

of resetting the frame buffer as well as the offset address value. When new pixel data is transfered, *am_new_data* is set to 1, and *am_pixel_transfer* is updated. We also simulated the situation of a busy Avalon bus, where the *am_wait_request* will be 1 for one cycle, and then become 0 after one cycle. At the same time, the data will be written to the memory, where the address is calculated by summing the *as_start_address* and *current_offset*. As soon as all the data is successfully written to the memory, the Avalon master sets *am_data_ack* to 1, so the camera interface can start transferring the next pixel value.

3.5 Camera Configuration

The display resolution of the LCD is 320×240, so we set the resolution of the camera to be 640×480 by setting the column size, row size, binning, and skipping registers according to the following values.

Address	Register Name	Value
0x3	Row Size	1919
0x4	Column Size	2559
0x22[5:4]	Row_Bin	3
0x22[2:0]	Row_Skip	3
0x23[5:4]	Column_Bin	3
0x23[2:0]	Column_Skip	3

Table 2: Camera register setup

These registers will be set via I2C writes. By setting *Row_Skip* and *Column_Skip* to 3, the camera would skip three sets of pixels during readout, resulting in an output image with resolution equals to 640×480. However, this would cause an undesired aliasing effect. To obtain a smoother image, we can set *Row_Bin* and *Column_Bin* to 3, so the camera module would combine the neighbor pixel value by averaging, thus the aliasing effect would be eliminated. The pixel readout is shown in Figure 19. Concretely, we set these registers with the following listed code.

```c
bool init_camera(void) {
    i2c_dev i2c = i2c_inst((void *) TRDB_D5M_0_I2C_0_BASE);
    i2c_init(&i2c, I2C_FREQ);
    bool success = true;

    /* setup camera resolution */
    success &= trdb_d5m_write(&i2c, 0x3, 1919);
    success &= trdb_d5m_write(&i2c, 0x4, 2559);

    /* setup binning and skipping */
    success &= trdb_d5m_write(&i2c, 0x22, 0x33);
```

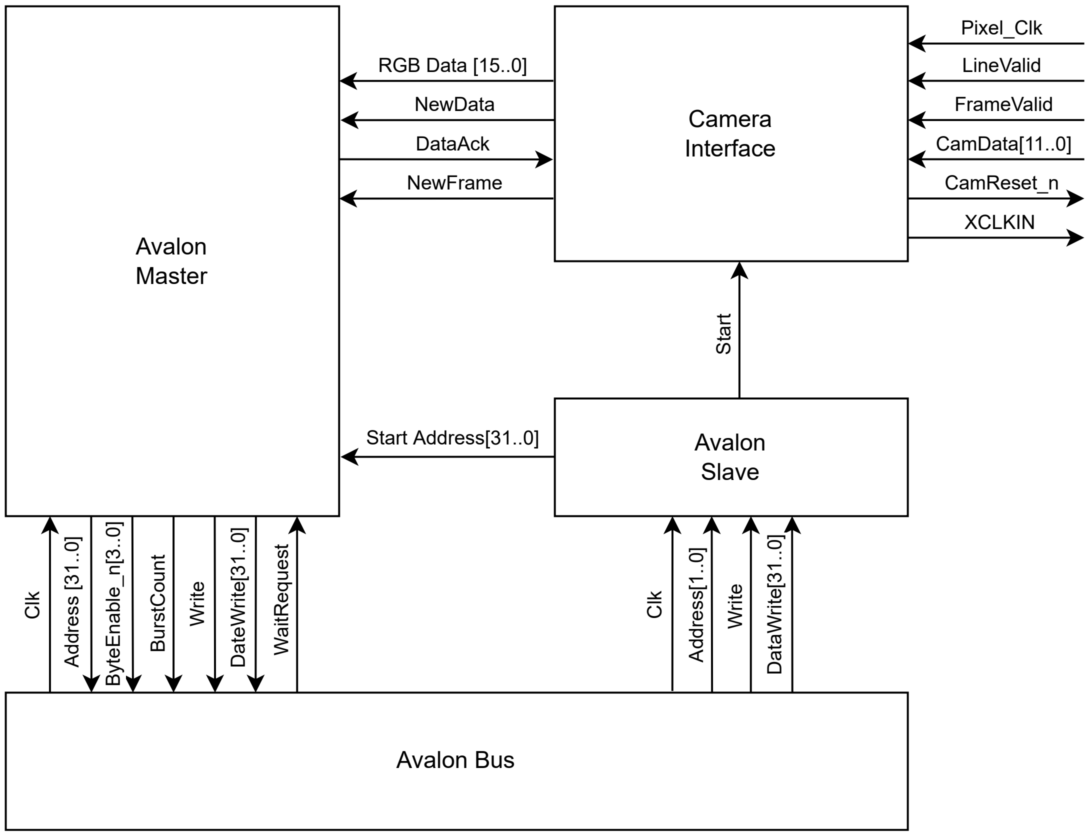


Figure 15: The block diagram of the Camera Module.

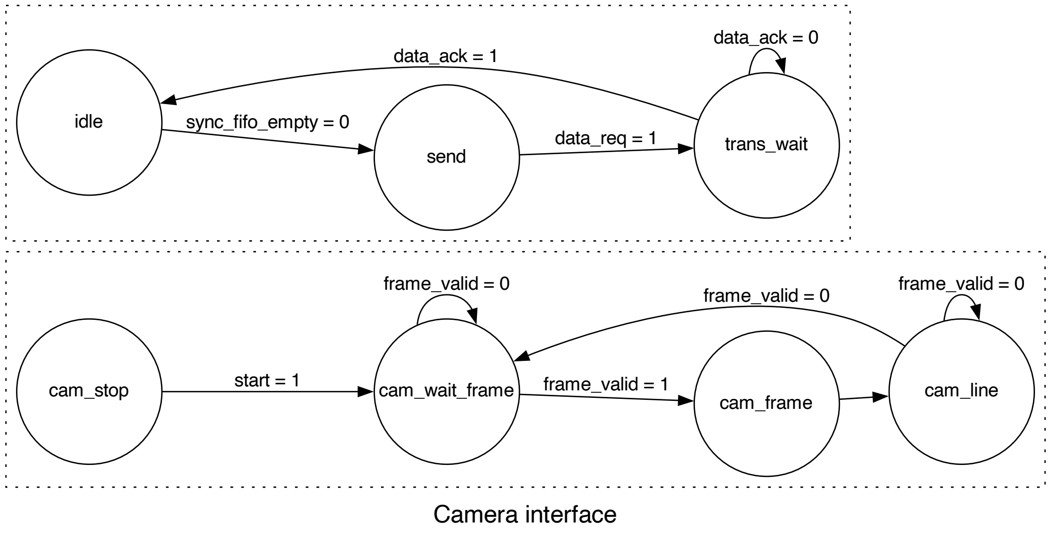


Figure 16: Finite-state machine of the Camera Interface.

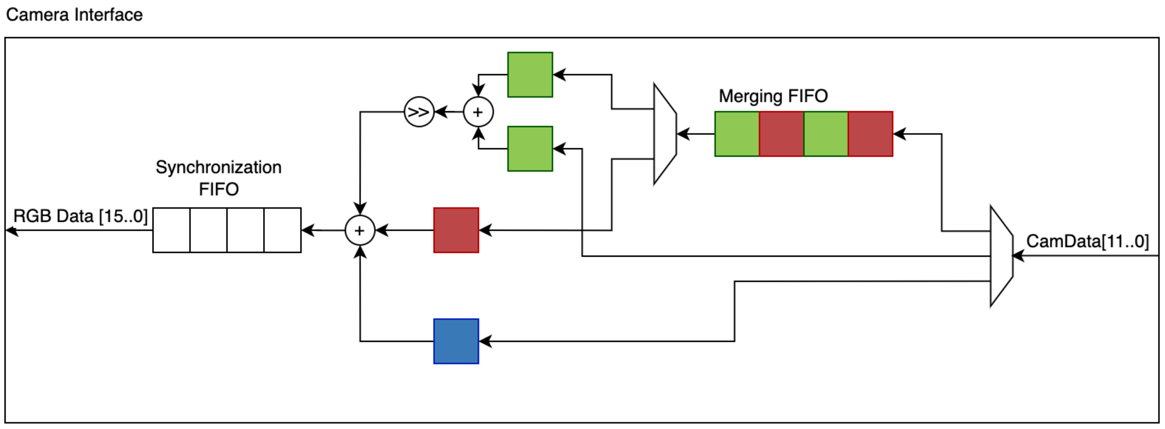


Figure 17: Finite-state machine of the Camera Interface.

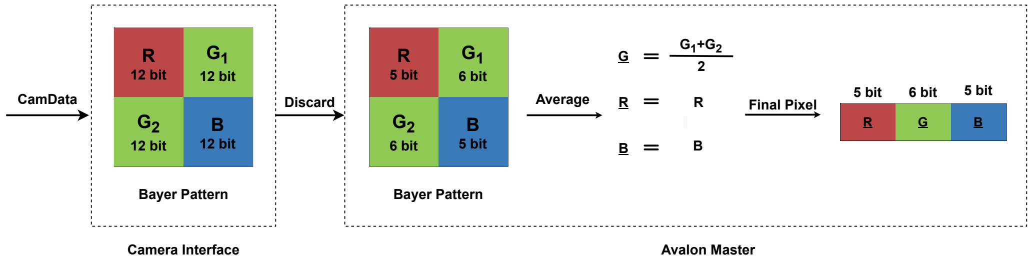


Figure 18: Image Subsampling and Pixel Data Demosaicking

```
    success &= trdb_d5m_write(&i2c, 0x23, 0x33);

    return success;
}
```

3.6 Pin Connection

The camera module will be connected to the **GPIO1** 2×20 GPIO connection on the FPGA board. Hence, the data will be transferred through pin 16, 15, 14, 13, 10, 9, 8, 7, 6, 5, 4, and 2, and the I2C data and clock will be located at pin 26 and 27 respectively.

4 Top-level block diagram

For the LCD, the following are the top-level connections: RESET n, LCD ON, CSX, DCX, WRX, RDX and the Data bus should be connected to their corresponding pins on the GPIO 0. Do not worry about the ADC pins on the LT24 LCD device. The main reset should be connected to the KEY N(0) pin of the board, and the clock should be set to clock 1 at 50MHz Figure 22 shows details about connections.

5 Memory organization

We plan to use double buffering to store an image in DRAM in a way that avoids artifacts that can occur when a buffer is being written to and read from at the same time. The address in DRAM where the image will be stored will be specified by the buffer manager. This approach is intended to ensure that the image is stored and accessed in a way that avoids any potential issues with concurrent read and writes operations.

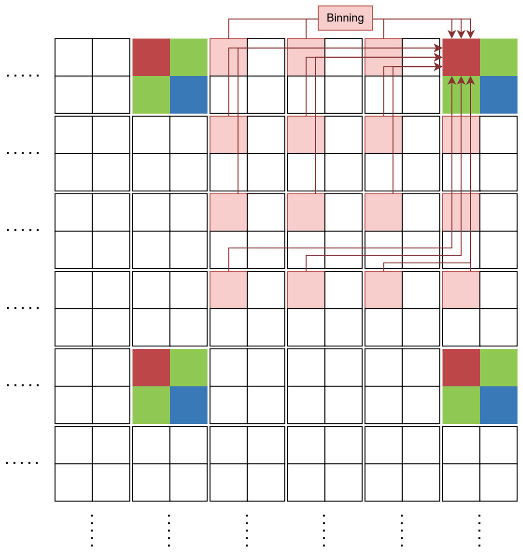
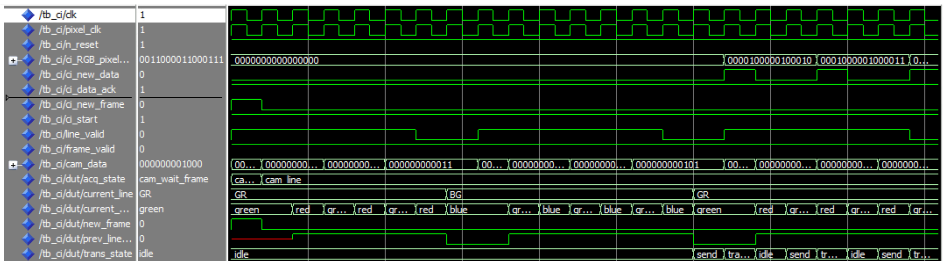


Figure 19: Pixel readout when $Row\_Bin = 3, Row\_Skip = 3, Column\_Bin = 3, Column\_Skip = 3$.



```
/tb_ci/clk                 1
/tb_ci/pixel_clk           1
/tb_ci/n_reset             1
/tb_ci/ci_RGB_pixel...     0011000011000111
/tb_ci/ci_new_data         0
/tb_ci/ci_data_ack         1
/tb_ci/ci_new_frame        0
/tb_ci/ci_start            1
/tb_ci/line_valid          0
/tb_ci/frame_valid         0
/tb_ci/cam_data            000000001000
/tb_ci/dut/acq_state       cam_wait_frame
/tb_ci/dut/current_line    GR
/tb_ci/dut/current_...     green
/tb_ci/dut/new_frame       0
/tb_ci/dut/prev_line...    0
/tb_ci/dut/trans_state     idle
```

Figure 20: Simulation waveform of camera interface

The system will allocate a contiguous memory region with a size of two frames for the camera controller's Avalon master to write pixels to, starting at the base address. When the master reaches the boundary of the region, it will wrap back to the base address and continue writing pixels. The LCD controller's Avalon master can then read from the base address until it has retrieved the size of a single frame, after which it will know to start reading data for the next frame.

To be specific, since each pixel value is 16 bits, we will need $320 \times 240 \times 16 bits = 76800 \times 16 bits = 153600 bytes$ for each frame. To increase the efficiency, 2 pixels are stored in one address. For example, Pixel$_1$ is stored in Base_Address [15:0] and Pixel$_2$ is stored in Base_Address [31:16]. Thus, each frame needs $\frac{153600}{2} = 79800 bytes = 0x12C00 bytes$ of memory.

Therefore, the first frame will be stored in Base_Address + 0x00000 to Base_Address + 0x0x12BFC; while the second frame will be stored in Base_Address + 0x0x12C00 to Base_Address + 0x0x257FC. The design is shown in Figure 23.

6 Sequence of Commands

The command we plan to use are Exit Sleep, Display ON, Memory Access Control, Column Address Set, Page Address Set, COLMOD: Pixel Format Set, COLMOD: Pixel Format Set and Memory Write. The address of the commands is 11h, 29h, 36h, 2Ah, 2Bh, 3Ah, F6h and 2Ch. After we turn on the LCD, configure it and send a memory write, then we need to continually send pixel data to the address.

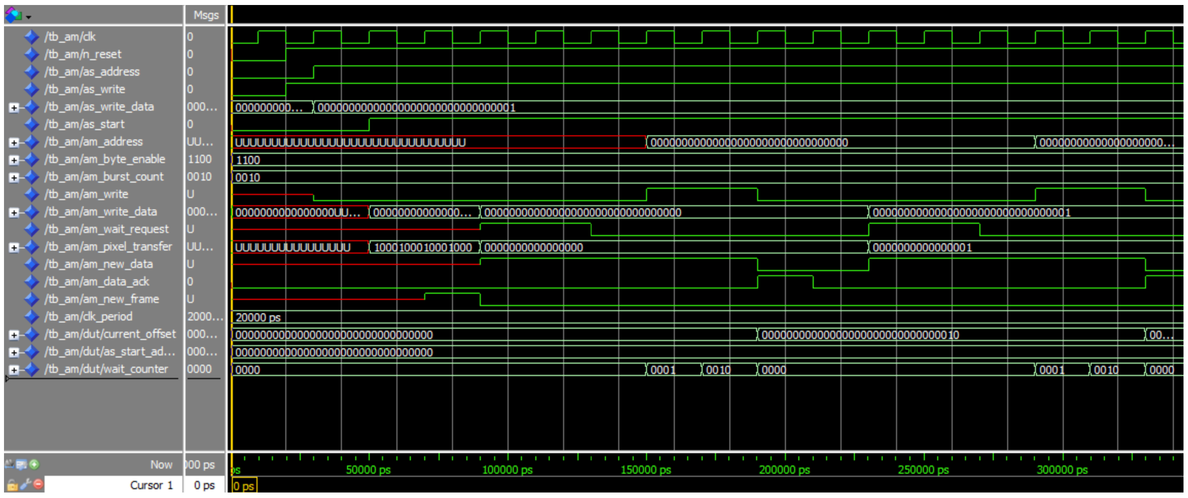


Figure 21: Output of Avalon master/slave testbench on ModelSim.

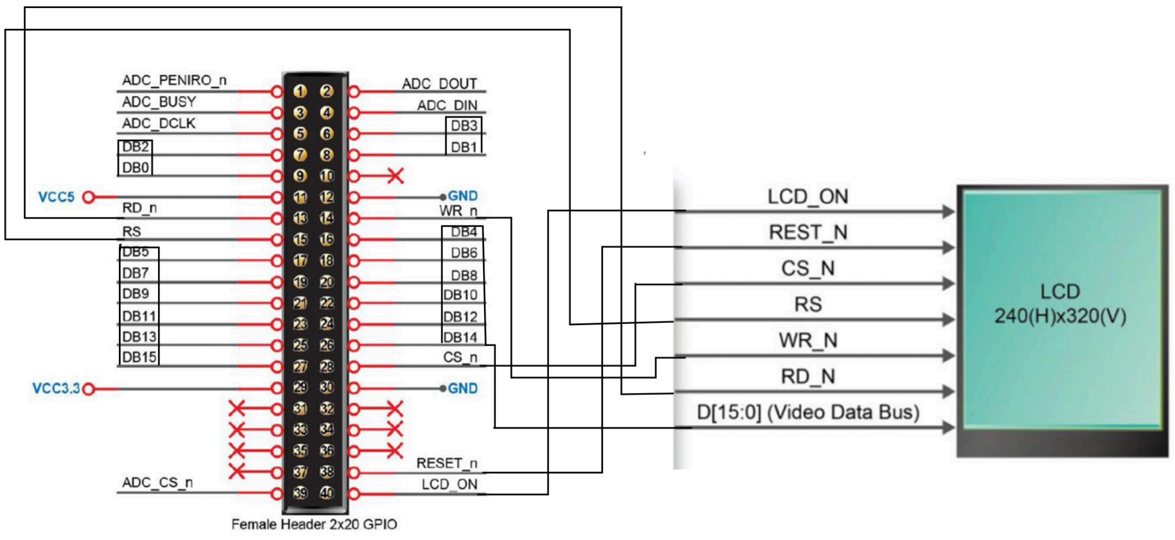


Figure 22: Pins connections of LCD with the board.

7 Conclusion

In this report, we showed how we designed the system that configure LT24 LCD display and TRDB-D5M camera with block diagram indicating the internal and external signal connection and the finite state machine explaining the detailed design of each component.

To ensure the image taken from the camera can be properly shown on the LCD display, we adopt double-frame buffer to prevent conflict of simultaneously read and write.

Following with a simulation testbench result, we showed that each component in the system runs exactly as we expected. However, although they seem to work fine by testing individually, when we try to merge all the components, there are some bugs that we couldn't figure out before the report submission deadline, where the image is not shown on the LCD display. We will keep working in the following days and we hope we can demo it without any problem during the presentation.

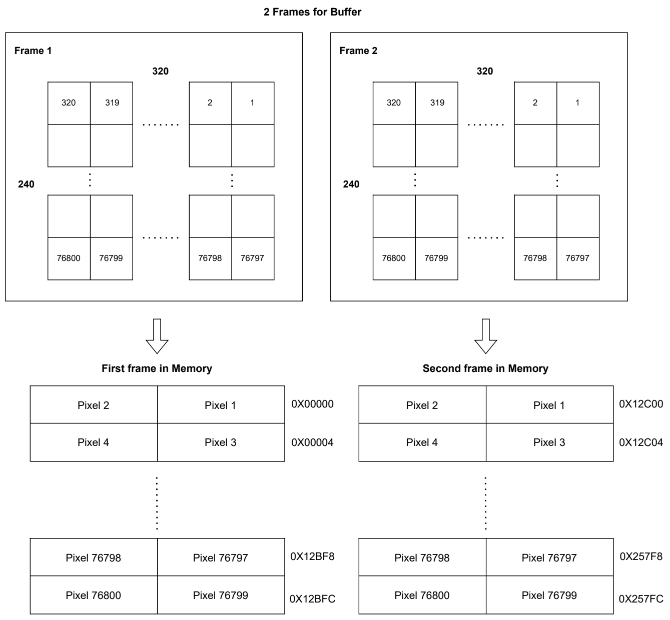


Figure 23: Frame Buffer and Memory Organization