CS473: Avalon master laboratory
Design of a camera controller

Olivér Facklam

January 7, 2021

Contents

1 Problem statement **1**

2 System overview **2**

3 Camera controller design **3**
3.1 Interfaces & register map 3
3.2 Internal architecture 4
3.3 Finite state machine (FSM) 4
3.4 Camera interface 5
3.5 DMA unit 8

4 Frame buffer organization **9**

5 System configuration & operation **9**

6 Implementation & testing **10**
6.1 System implementation 10
6.2 Testing phases 10

7 Results of merged system **11**

Note: sections 1 to 5 are directly taken from the lab 3 report, since the design hasn't changed in any significant manner.

1 Problem statement

The goal of this lab is to design a camera controller which is programmable and can capture and transfer frames directly from the camera to a memory module.

The controller must expose an Avalon slave interface, allowing it to be programmed. In particular, it should have registers to set the frame buffer addresses and lengths from the processor.

The controller must also have a conduit connecting it to the camera's clock and data output. This is the interface on which the frame pixels will be sampled.

Finally, the controller must contain an Avalon master interface, enabling fast data transfer directly to memory. Ideally, this interface should support burst transfers to speed up the

flushing process.

Since the LCD display has a resolution of 320×240 pixels, this is also the resolution we target for the camera frame capture. In terms of performance, we require at least 25 fps for the image to be fluid, but a frequency between 30 and 40 fps would be preferable.

I am working with the group composed of Samuel Thirion and Jonas Schweizer who are implementing the LCD controller.

2 System overview

The overall system architecture is presented in figure 1. It consists of a NIOS-II processor, connected to the I^2C, camera and SDRAM controllers through the Avalon bus. The camera controller's master interface can also access the Avalon bus to transfer data to the memory.

The camera is connected to the FPGA through the development board's GPIO pins. The pins for serial communication are mapped to the I^2C controller's `sclk` and `sdata` signals. The data signals (`data`, `fval`, `lval`) on the other hand are connected to the camera controller.

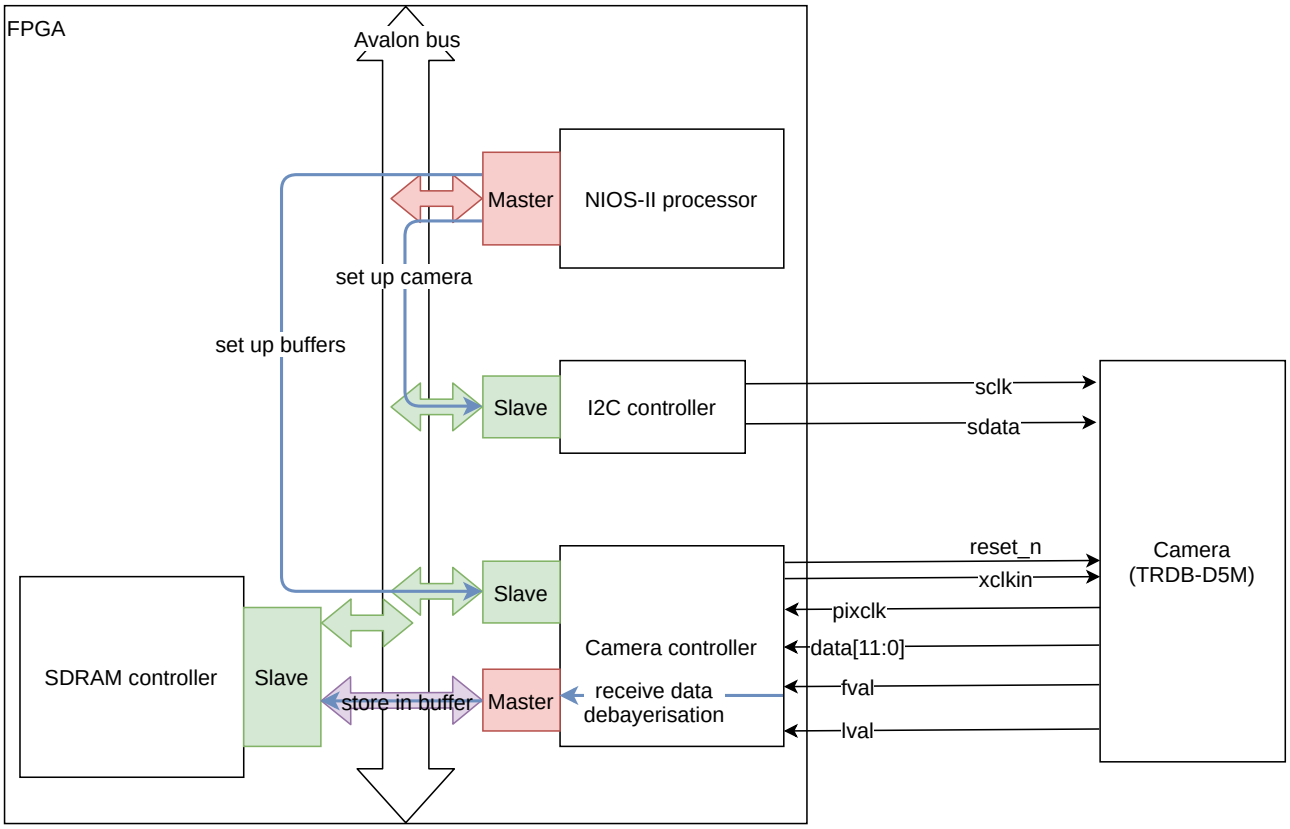

Figure 1: Overview of system architecture and operation

The system typically operates according to the following high-level description:

1. The processor sets up the camera parameters by sending the appropriate commands to the I^2C interface.
2. The processor initializes some buffer regions in memory and gives the buffer information to the camera controller.
3. The camera controller waits for the buffer to be ready.
4. The controller captures the next whole camera frame, performing the necessary debayerisation during this process.

5. The controller uses its master interface to copy the obtained pixel sequence to the provided buffer.
6. The capture process can be repeated as many times as necessary, alternating between the different buffers.

Table 1 shows how the camera pins are mapped to the system signals.

Signal	GPIO1 signal	Board pin
sclk	GPIO1[24]	JP7 pin 27
sdata	GPIO1[23]	JP7 pin 26
reset_n	GPIO1[17]	JP7 pin 20
xclkin	GPIO1[16]	JP7 pin 19
pixclk	GPIO1[0]	JP7 pin 1
fval	GPIO1[22]	JP7 pin 25
lval	GPIO1[21]	JP7 pin 24
data[11:0]	GPIO1[1, 3:13]	JP7 pins 2, 4..10, 13..16

Table 1: Development board pin assignments

3 Camera controller design

Figure 2 shows the general architecture and exposed interfaces of the camera controller design.

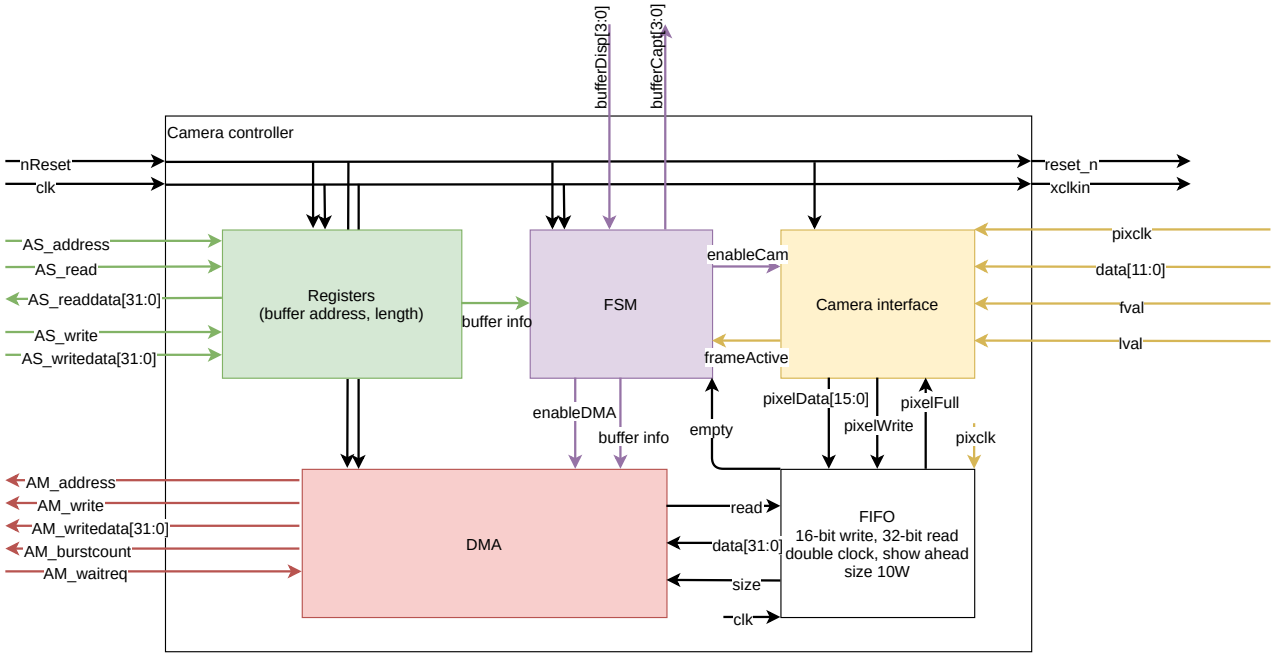

Figure 2: Architecture & signals of the camera controller

3.1 Interfaces & register map

The camera controller exposes 5 different interfaces.

In black: the 50 MHz FPGA clock input, alongside the associated reset signal.
In yellow: a conduit interface for camera communication. This interface contains the pixel clock, data and frame/line valid signals.

In purple: a conduit interface for synchronization. In the final system, these signals will be connected to the LCD controller to determine the states of the different buffers. The `bufferDisp` input signal denotes when a buffer is being emptied by the LCD controller; the `bufferCapt` output signal shows a buffer is being filled by the camera controller.

In green: a 32-bit Avalon slave interface, supporting 0-wait writes and 1-wait reads. The registers accessible through this interface are described in table 2.

In red: a 32-bit Avalon master interface, supporting burst writes.

Address	Register	Bits	Description
0x00	BUF0ADD	0-31	Address in memory of buffer 0.
0x01	BUFLEN	0-28	Length of each buffer. Buffer is considered valid when len > 0.
	BUFNUMBER	29-31	Number of buffers.

Table 2: Register map of the camera controller

3.2 Internal architecture

The internal architecture of the camera controller is also presented on figure 2. The system consists of 5 blocks:

- the "Registers" block, implementing the 0-wait write / 1-wait read Avalon slave interface, storing the values of the 2 registers described previously, and providing them to the rest of the system;
- the FIFO block: a double-clock, show-ahead FIFO from Quartus, with a 16-bit write bus (driven by `pixclk`), and a 32-bit read bus (driven by `clk`), capable of storing 10 lines of pixels;
- the "FSM" block, implementing the finite state machine described in the next section, driving the `enableCam` and `enableDMA` signals;
- the "Camera interface" block, driven by the camera's pixel clock, sampling the incoming pixels and storing them in the FIFO;
- the "DMA" unit, using the buffer info to flush the data stored in the FIFO to the main memory, in bursts of half-lines.

3.3 Finite state machine (FSM)

Figure 3 shows the main finite state machine controlling the system operations.

Each buffer's state is monitored through an instance of the state machine at the bottom of the figure, cycling through the 4 states based on the `bufferCapt` and `bufferDisp` signals of the synchronization interface.

The main state machine waits in the IDLE state for the next buffer to become available. It then enters an ARMED phase, waiting for the current camera frame to finish (so we can capture a whole frame). Both the camera interface and the DMA are then enabled until the end of the next frame. Finally, the state machine waits in the FLUSHING state until the FIFO is completely empty, before moving on to the next buffer.

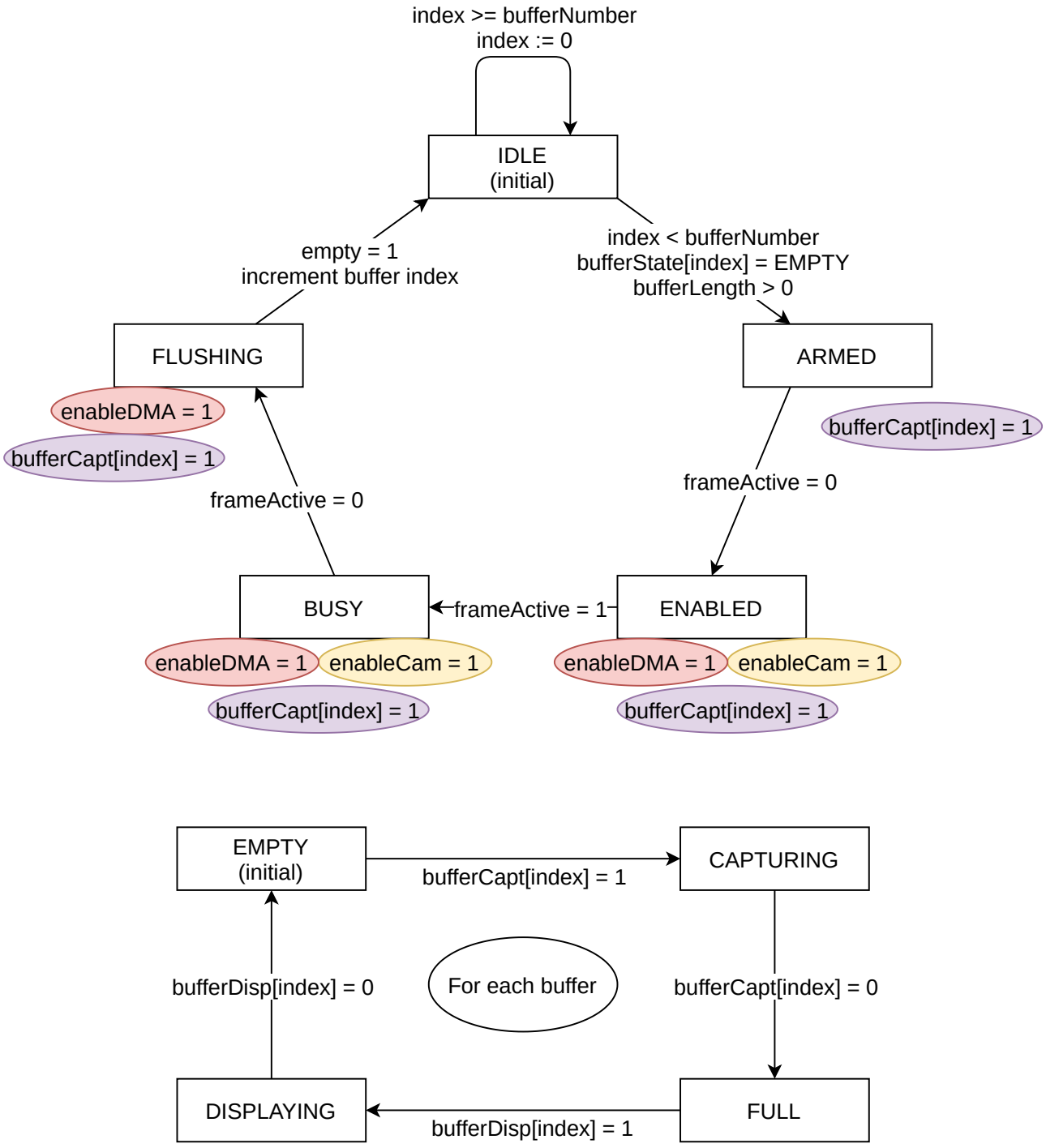

Figure 3: State machine of the camera controller

3.4 Camera interface

The whole camera interface module is driven by the camera's pixel clock. It contains two single-clock FIFOs, respectively storing the green1 and red values. The signals and internal state machines of the camera interface are shown in figure 4.

When capturing is enabled, and the frame & line are valid, the interface samples the incoming 12-bit data on the falling edge of the pixel clock. The pixels of even rows (starting at 0) are buffered alternatively in the 2 FIFOs (G1 & R). The pixels of odd rows are simply stored in temporary registers (B & G2).

In parallel to the sampling of the odd rows, the green1 and red pixels are read back from the

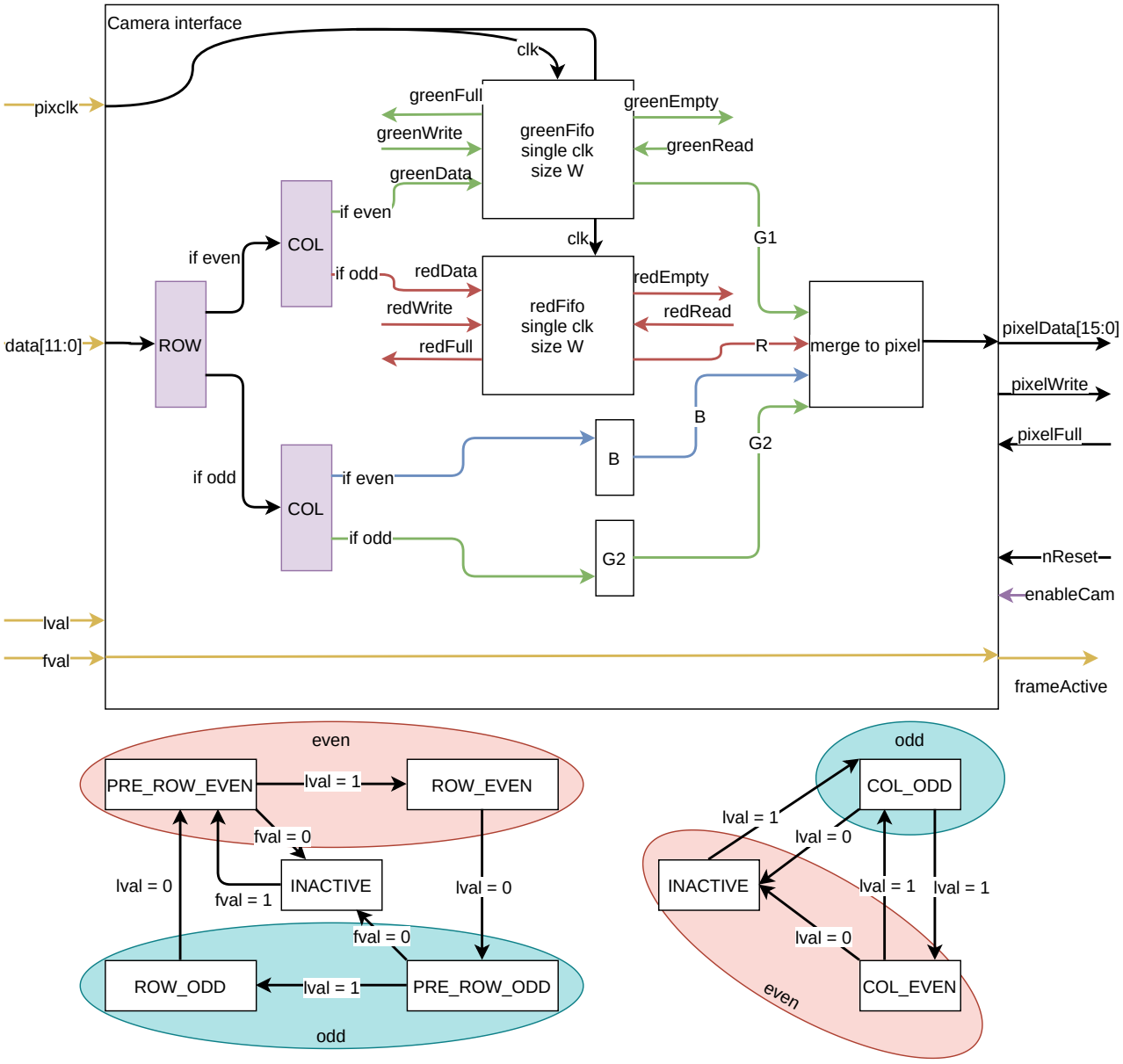

Figure 4: Design and operation of the camera interface block

FIFOs. This allows us to group together the four colors from a 2×2 square, and merge them into a single 16-bit RGB pixel. This pixel value is then output through the `pixelData` signal, and written in the main FIFO.

The state machines shown in figure 4 simply keep track of the row and column parity. All the read and write signals are directly driven by some combinational logic of the input and state signals mentioned previously. These functions can be seen at the very bottom of the figure.

Typical timings for the capture of a single camera frame are shown in the diagram of figure 5.

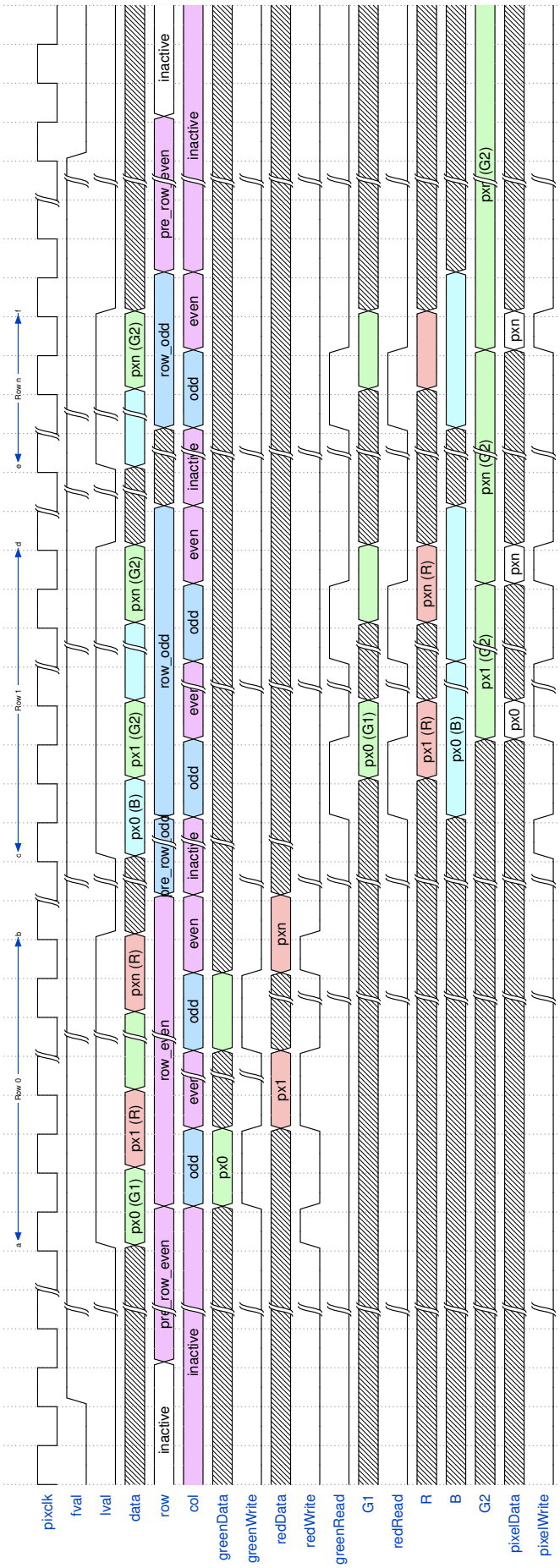

Figure 5: Timing of the camera interface signals during a frame capture

3.5 DMA unit

The DMA unit is responsible for driving the Avalon master interface, and writing the pixels into the appriopriate memory buffer. Once enabled, the DMA keeps track of the current address we are writing to, starting from the buffer address and incrementing after every burst. The writing is done in half-line bursts (160 pixels = 80 words). At the beginning of each burst, the DMA waits in the BURSTPREPARE state, for enough pixels to become available in the FIFO. It then outputs the 80 words successively (monitoring the `AM_waitreq` signal), keeping track with a counter. Once all pixels have been written, the burst is terminated.

The design of the DMA unit is shown in figure 6, the timing diagram in figure 7.

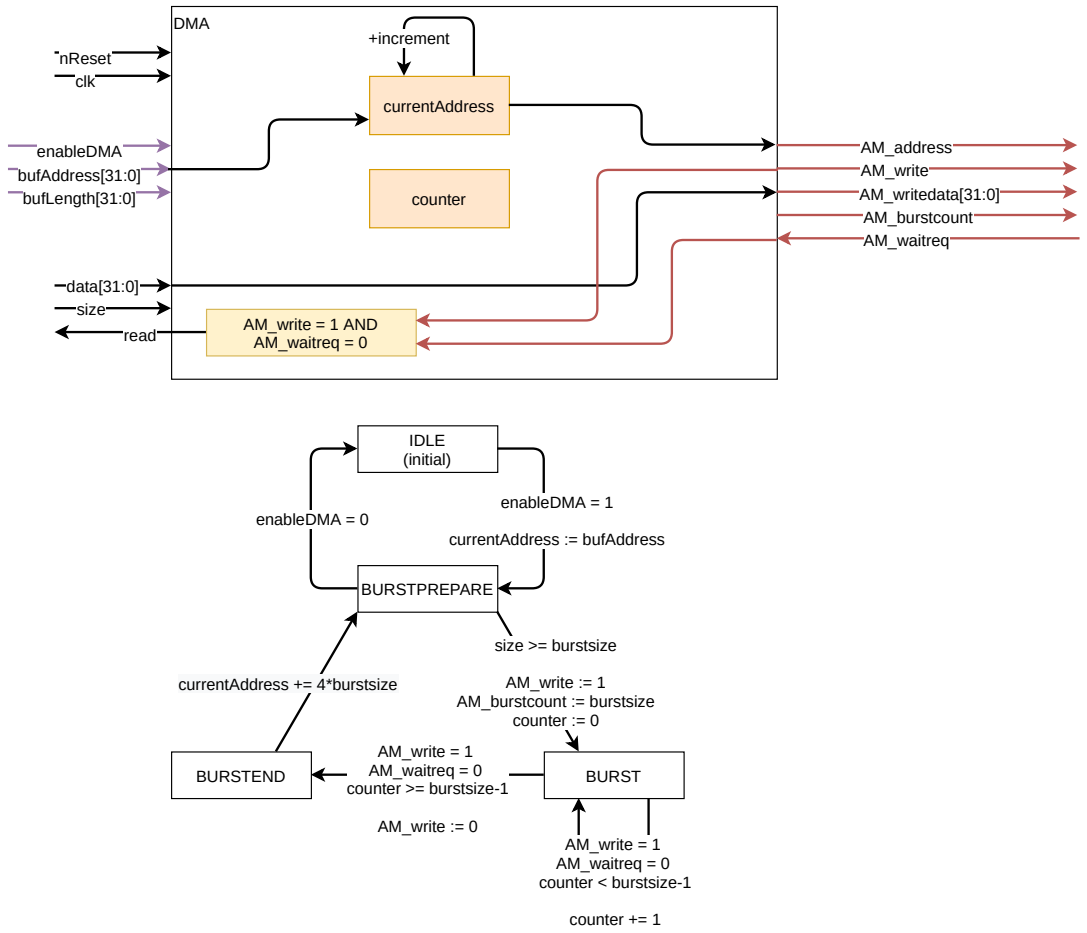

Figure 6: Design and operation of the DMA unit

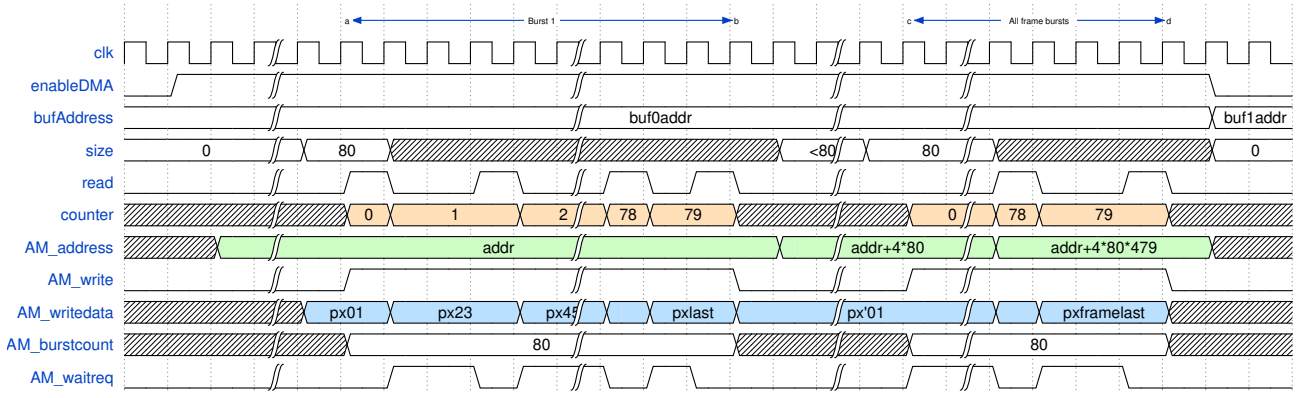

Figure 7: Timing of the DMA signals during a frame flush

4 Frame buffer organization

Pixels are stored on 16 bits in 5-6-5 RGB format, like shown in table 3. Thus 2 pixels fit into a 32-bit word in memory.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R4	R3	R2	R1	R0	G5	G4	G3	G2	G1	G0	B4	B3	B2	B1	B0

Table 3: Pixel organization

In the frame buffer, the 320 pixels from a camera row are stored contiguously, on 160 successive words. The 240 rows are also stored contiguously. The memory organization is shown figure 8.

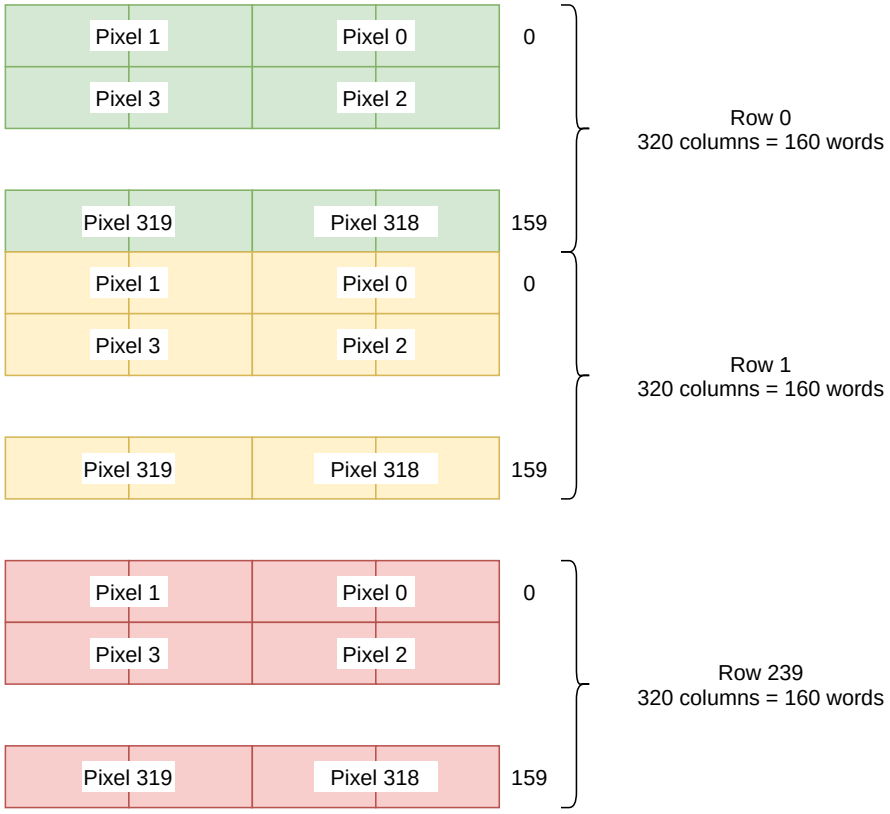

Figure 8: Buffer organization

5 System configuration & operation

During system setup, the processor must configure some important camera settings to make it work in the appropriate mode. First, the size of the captured image should be set by writing `Column_Size` = 2559 and `Row_Size` = 1919. Then, both row and column skip and bin should be set to 3, in order to achieve the desired resolution of 640×480. Since the camera pixels are grouped by the camera interface into 2×2 squares to form RGB pixels, this is ideal in order to create the 320×240 image we are targeting.

After the camera setup, the processor must allocate some buffer areas in the main memory, and set their information in the camera controller's registers.

From then on, the camera controller operates independently, synchronizing through the previously described conduit with the LCD controller to determine the state of the buffers and to rotate them.

6 Implementation & testing

6.1 System implementation

The camera controller is implemented precisely as outlined in previous sections. Each subsystem is coded as a component in a separate VHDL file in the `hw/hdl/camera_controller/comp` folder.

The camera interface is implemented in `cam_interface.vhd` and internally uses two single-clock FIFOs generated by Quartus, as well as a pixel merger component defined in `pixel_merger.vhd`. The registers implementation can be found in `registers.vhd`.

The DMA unit is declared in `dma.vhd` and contains a generic parameter to set the burst size. Similarly, the controller's finite state machine can be found in `fsm.vhd` and has a generic parameter to set the maximum number of buffers. It also uses an auxiliary state machine defined in `buffer_sm.vhd` and instantiated through a `for...generate` loop to track the state of each buffer.

Finally, the camera controller as a whole is defined in the top-level `camera_controller.vhd` file. It has two generic parameters (burst size and max buffer count), and exposes the 5 interfaces described previously.

The software implementation contains code inspired from the I^2C demo to initialize the camera hardware. Optionally, the camera can be set to output test patterns instead of the video feed, which is useful for debugging some issues. The startup code also sets up the camera controller's registers, thereby starting the acquisition of images. Some code is also included to print out the memory contents, and to save the buffers as files on the host computer through the `altera_hostfs` package.

6.2 Testing phases

Figure 9: Image captured with TRDB-D5M and the custom controller, saved to ppm

The camera controller has been tested in 3 phases. First, each VHDL component has a corresponding testbench in the `hw/hdl/camera_controller/tb` folder, through which they were tested in ModelSim.

As a second step, the camera controller was added to QSys alongside the CMOS generator. This allowed to test the acquisition in hardware and to print out the memory contents and verify the result manually.

The third step was to test the final acquisition pipeline, with the controller being connected to the camera over the GPIO1 interface. The memory contents was written to a ppm file. Figure 9 showcases such a capture. The image is upside-down due to the order in which the camera reads out the pixels.

7 Results of merged system

After merging the two groups' work and adding both the camera and the LCD controller to the FPGA, the final system works as expected. Figure 10 shows the hardware setup.

The images captured by the camera are correctly displayed on the LCD. However, the frame rate is relatively low, probably only around 10 FPS compared to the 25 FPS we were targeting. Also the displayed image quality could be improved upon by modifying color correction settings (saturation, brightness...).

Overall, the system works well. You can find a short demo video next to this report.

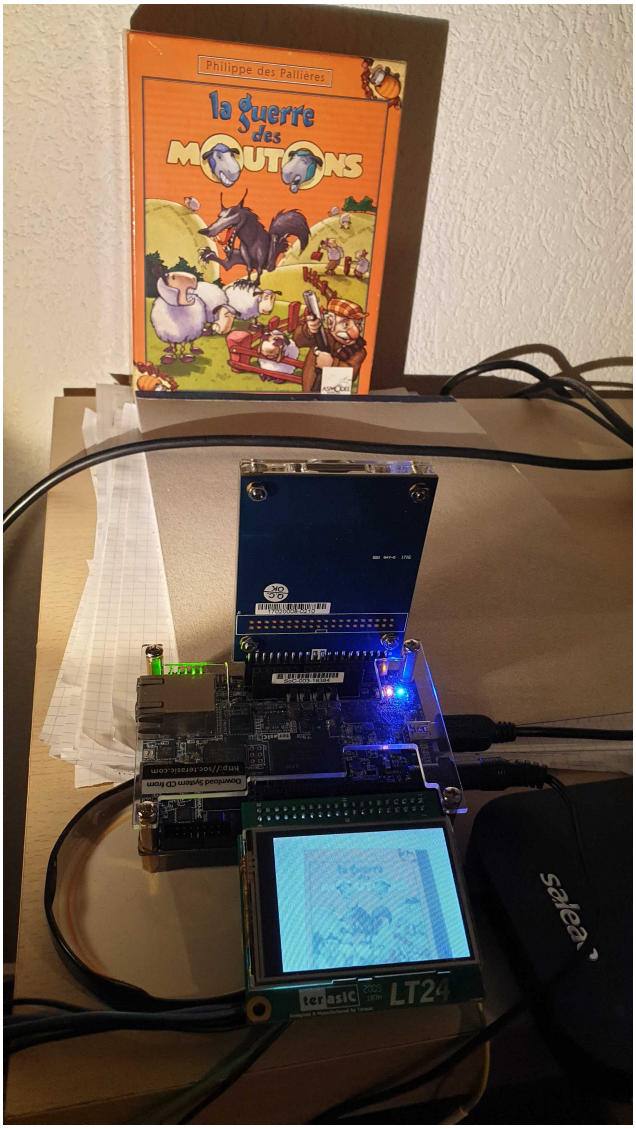

Figure 10: Hardware setup for merged system