ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

REPORT

LAB 4.0 EMBEDDED SYSTEM

EPFL COURSEWORK

Vincent Gherold, Pietro Mello Rella

16th February 2023

1 INTRODUCTION

The objective of the laboratory is to design a custom master interface for Avalon bus that allow to take pictures from a camera interface and save it to a specific address of memory. This sub-component is part of a bigger system, composed of a camera and a LCD display, with the objective of implementing a real-time camera system, which continuously takes and displays images.

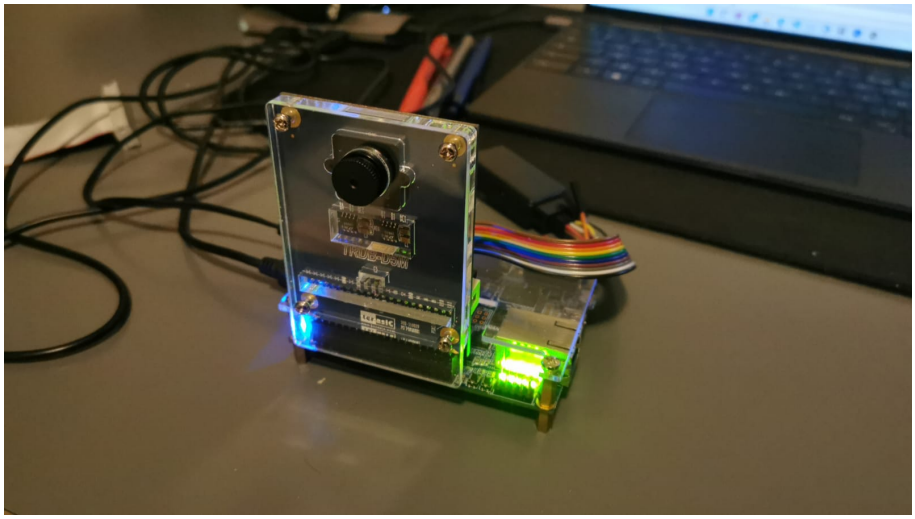

FIGURE 1
FPGA Camera System

2 OVERVIEW OF THE SYSTEM

For the system, a part from the our custom master and slave camera controller, we need other components for controlling the interface and for interfacing with the memory.

- Avalon Bus: this component interfaces all other components of the system together, by implementing synchronization and defining which of the components can transmit at a given time. This component is particularly important as we are implementing a multi-masters system.
- NIOS II Processor: this master component runs the compiled C code that allow us to control the whole system on a higher level, by setting memory addresses, camera and LCD options.
- JTAG Interface: this slave component interface the FPGA with the PC, it is useful for programming the behavior of the FPGA at compile time, and to have live feedback from the system via the console.
- SDRAM Controller: this slave component allow the communication with the external SDRAM memory, which we need to save the images from the camera system.
- SRAM: this slave component contains the memory of the processor, the compiled instructions and the variable of the system during execution.
- I2C Camera Interface: this slave component interfaces with the camera to provide the necessary working configuration.

In the diagram below, we can see all the components of the system, and their connections.

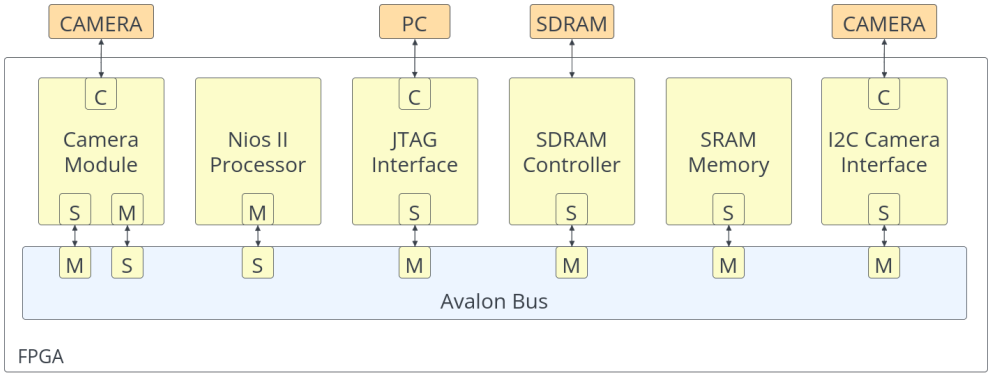

FIGURE 2
Full system diagram

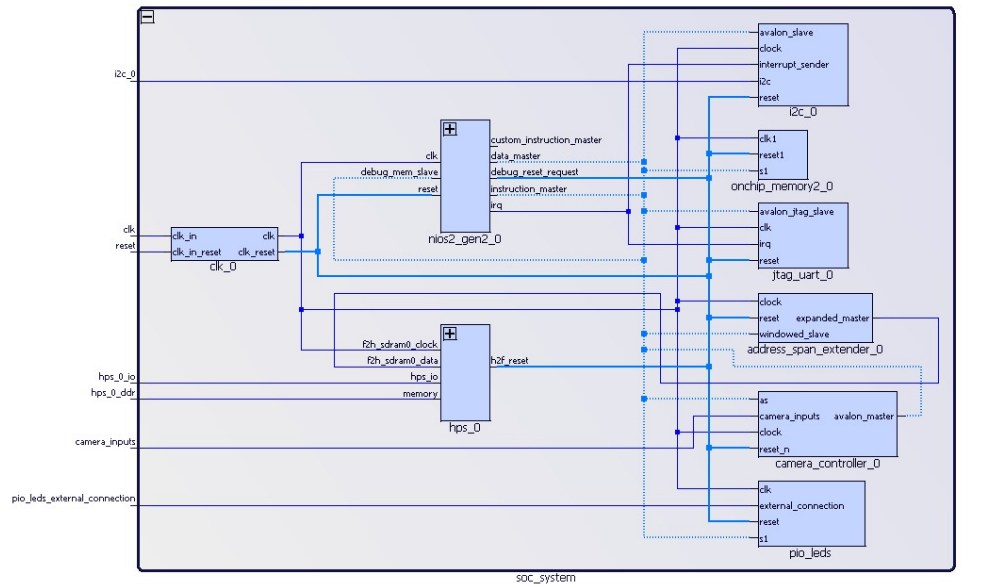

FIGURE 3
System Components connections

FIGURE 4
SOC System design

3 CUSTOM MASTER AND SLAVE CAMERA CONTROLLER

3.1 OVERVIEW

Our main architecture for the Camera Controller is the following:

We have two main sub-components the Avalon Slave and the Avalon Master:

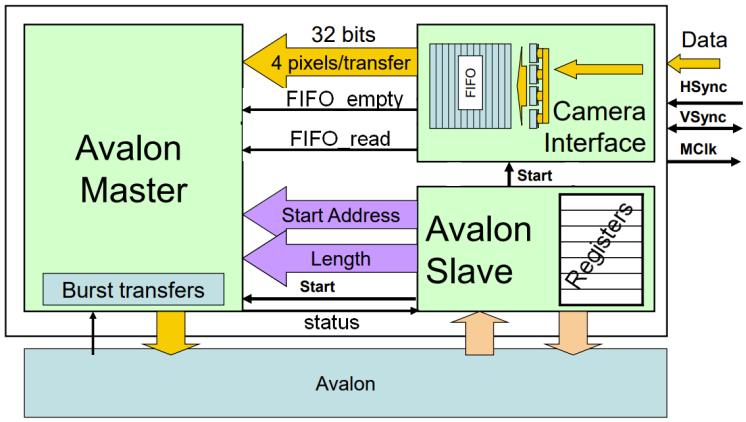

FIGURE 5
High Level Camera Architecture

The process flow is the following:

The software user configures the camera via the Ic2 interface which is connected to the camera. After he provides the memory address and the memory length to the Camera Controller. After he can start the

camera controller by setting the register start to 1.

The camera controller fetches the data and fills the memory. During that time the software user pools the status register to know when the transfer is finished.

3.2 AVALON SLAVE

The sub-component Avalon slave gets the signal from the camera and stores each pixel into a FIFO in order to be ready to send to the memory by the Avalon Master Camera Interface.

The Avalon Slave Camera Interface can be configured by the following registers:

Register	Description
cam_addr	the memory address to start writing the frame image
cam_length	the length of the data that be written by the camera controller in the memory, starting at cam_addr
cam_start	start fetching the data from the camera
cam_status	is 1 when the Avalon master has finished his work

TABLE 1
Register description for Camera Controller IP Component

Here is the register map Camera Controller:

Address	Write register	Writedata[7..0]	Read register	Readata[7..0]
0	cam_addr	$\rightarrow cam\_addr$	cam_addr	cam_addr $\rightarrow$
1	cam_length	$\rightarrow cam\_length$	cam_length	cam_length $\rightarrow$
2	cam_start	$\rightarrow cam\_start$	cam_start	cam_start $\rightarrow$
3	cam_status	$\rightarrow cam\_status$	cam_status	cam_status $\rightarrow$

TABLE 2
Register map for Camera Controller IP Component

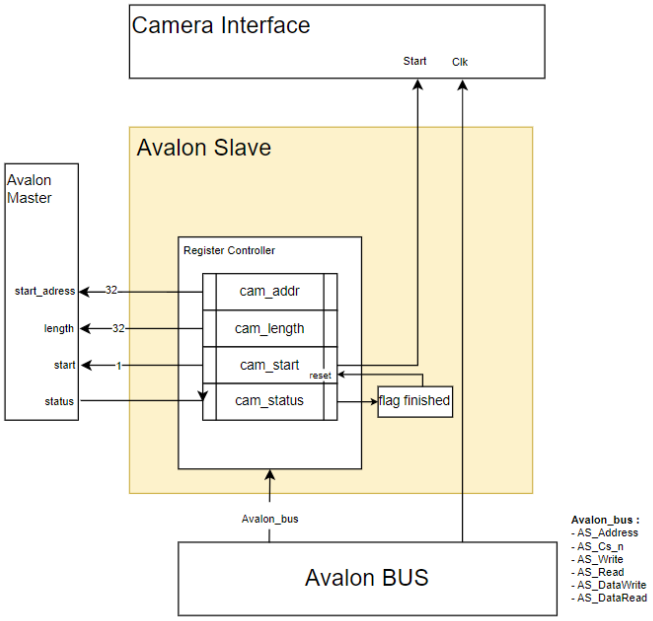

FIGURE 6
Low Level Avalon Slave Architecture

As you can see the Avalon slave distributes all the signal to the camera interface and the Avalon master. When the transfer is finished the register start is automatically cleared.

3.3 CAMERA INTERFACE

3.3.1 LOW LEVEL DESIGN

We have 2 color FIFO, that can store a line of pixels. And a third FIFO to store the blue pixel. The global FIFO store 32 bits elements and should be able 10 to 50 pixels (decided arbitrary because as soon there is a pixel inside, the pixel is fetched by the avalon master). The FIFO have the functionality show headed for the read in order to have the read as a acknowledgement instead of a request and a faster system. We choose to have two counters that can up to 1 (so 1 bit) in order to distinguish between odd and even rows and columns. For each pixel we keep the 8 MSB and all of them are merged in order to create a pixel of 32 bits composed of (R G2 G1 B) in this order (from MSB to LSB).

For the board connection, we need to connect:

- PIXCLK to PIXCLK of our camera interface
- D[11:4] to the data input of the camera interface
- XCLKIN to the avalon clock
- RESETn to the resetn of the camera controller
- FVAL and LVAL to camera interface submodule
- SDATA and SCLK to the IC2 module

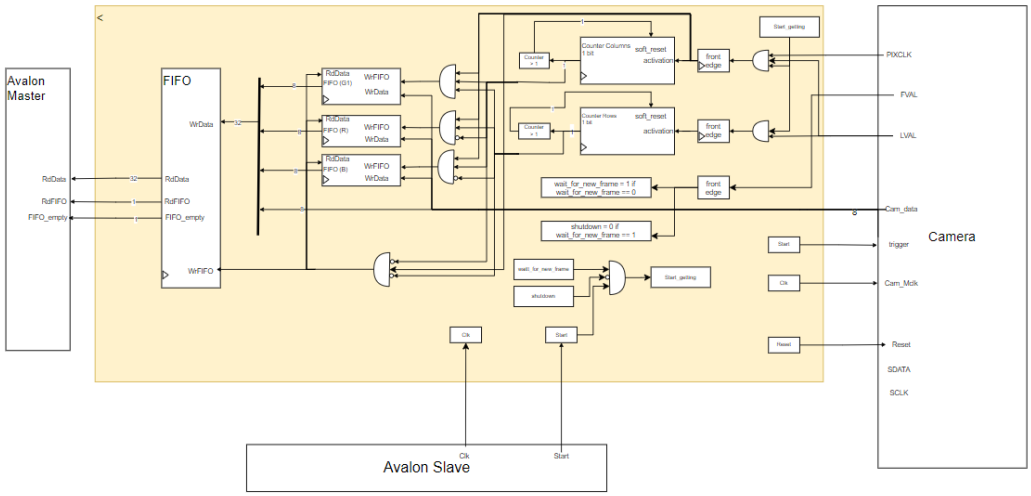

FIGURE 7
Low-Level Camera interface Architecture

3.3.2 CAMERA SETTINGS

We choose to have the default Line_valid format. (LINE_VALID to be negated when FRAME_VALID is negated)

We choose to have a resolution of 640x480 so the pixel dimension will be 320x240 (exactly the size of our LCD screen) and a binning mode so we have a nice merged image.

The data signal is 12 bits, we choose to keep only the first 6 bits MSB bits (discarding the 6 others). Each signal PIXCLK, FVAL, LVAL should be read at the down edge of the PIXCLK. Because we have 640x320 = 204 800 pixels, so by choosing a clock divider of 4 (means 8 because the formula is PIXCLK = CLK_input / (2 * clock_divider).

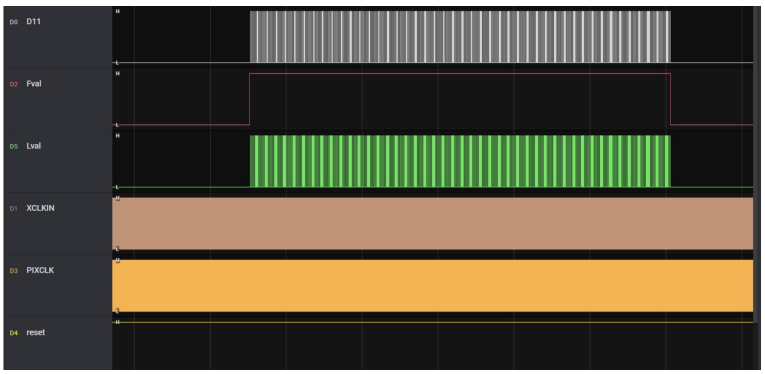

FIGURE 8
Camera Signals

3.3.3 DIAGRAM FLOW

The diagram flow is the following:

First, we reset the system, after we setup the program with the IC2 interface and cam_address and cam_length. To start the fetching, we put cam_start to 1. The memory is filled with the pixels from the camera. When it's finished, start is automatically set to 0. The camera controller for the next frame from camera and start fetching by monitoring the fval signal.

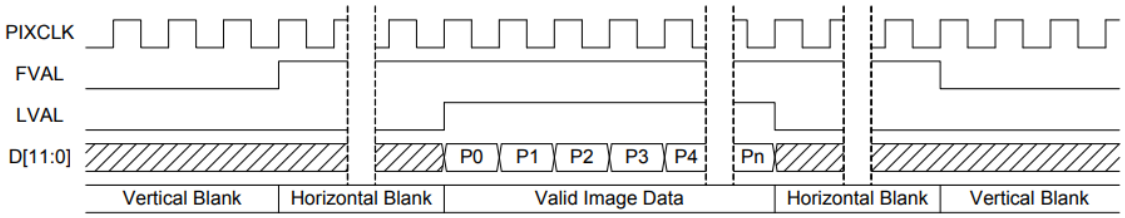

FIGURE 9
Input signals from the camera

We distinguish between odd and even rows and columns. The two counters for rows and columns are incremented and fill the FIFO R and G1 for each odd row. Alternating between R for odd and G1 for even columns. (the first row and first column are 1 so odd).

For the even rows, the big FIFO (32 bits) is filled up with the FIFO R G1 and FIFO B (see diagram), and G2. As soon the FIFO is filled a bit, the master-slave start picking data.

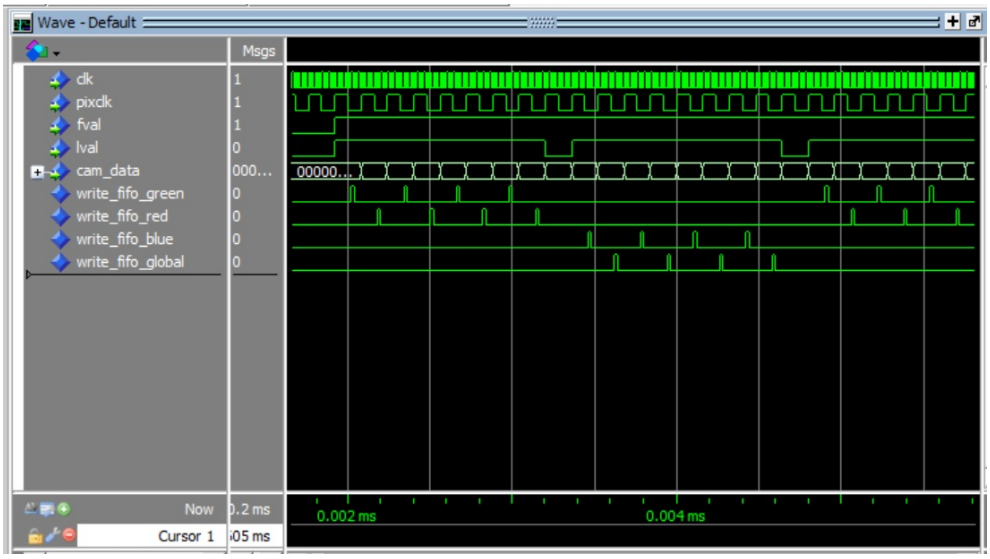

FIGURE 10
Camera interface Modelsim working simulation

As you can see we have the fifo green red blue global activating in the right order. Each pulse for the write in the fifo is one clk period, we have the pixclk much slower that the clk as intended.

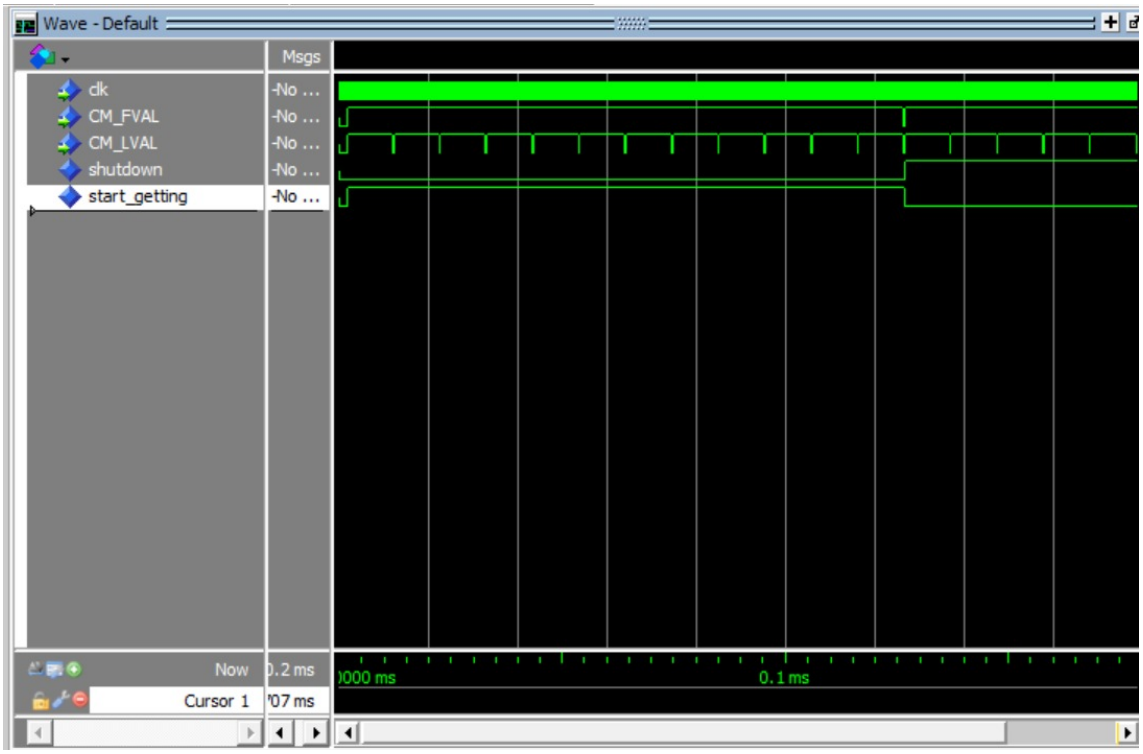

FIGURE 11
Shutdown signal for the camera

When cam_start is set to 1, our system wait for the next rising edge of fval in order to start the capture. After the second rising edge of fval starts, the capture is stopped by setting the shutdown signal to 1 (as you can see above).

3.4 AVALON MASTER

This sub-component gets various information from the Avalon Slave, the length of the transfer (the pixel size of the image), begin address in the memory and the signal to start the transfer, and from the Camera interface, the pixel feed and whether the FIFO buffer contains data. It outputs the pixels to the memory, with burst of 32 x 32 bit signals, each containing 2 pixels.

3.4.1 MEMORY FORMAT

For the memory pixel format we chose to map every pixel into a 16 bits (2 bytes) cell, where the 5 MSB bits contains red value, the 5 LSB bits contains blue value, and the remaining 6 bits represent green value. This format cause the image to have a lower color definition, but it requires a lower memory size.

FIGURE 12
Pixel format in memory

For the Red and Blue pixel we keep the 5 MSB bits (a better alternative is to add the LSB bit to the count, but was omitted in the system design for simplicity), for the Green pixel we sum the values of G1 and G2, and keep the 6 MSB bits (it corresponds to take the medium of the two values).

To save a 320x240 pixel image, the space required is $320 \cdot 240 \cdot 2 = 153'600B = 150KB$, while to save at the maximum resolution 1296x972, the space required is $1296 \cdot 972 \cdot 2 = 2'519'424B = 2.40MB$.

3.4.2 PIXEL REDUCER DESIGN

This component takes the pixel data coming from the camera interface, it selects the used bits of RGB, then applies the transformation to the pixel 16 bits format, then it stores the value of the pixel in a FIFO buffer. The buffer gets 16 bit in input (1 pixel) and outputs 32 bit (2 pixels) at a time. This way we can use the entire capacity of the Avalon Bus.

The FIFO from the camera interface uses the RdFIFO signal as a ACK for the received data, while the FIFO to store the reduced pixel uses the WrFIFO signal to input the data at the next rising edge, so the two signals can be sent simultaneously.

The two diagrams below show the architecture and the time behavior of the component:

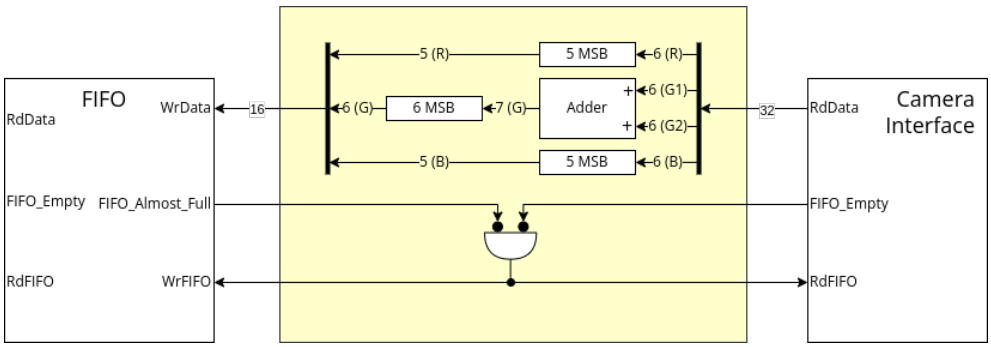

FIGURE 13
Pixel Reducer Architecture

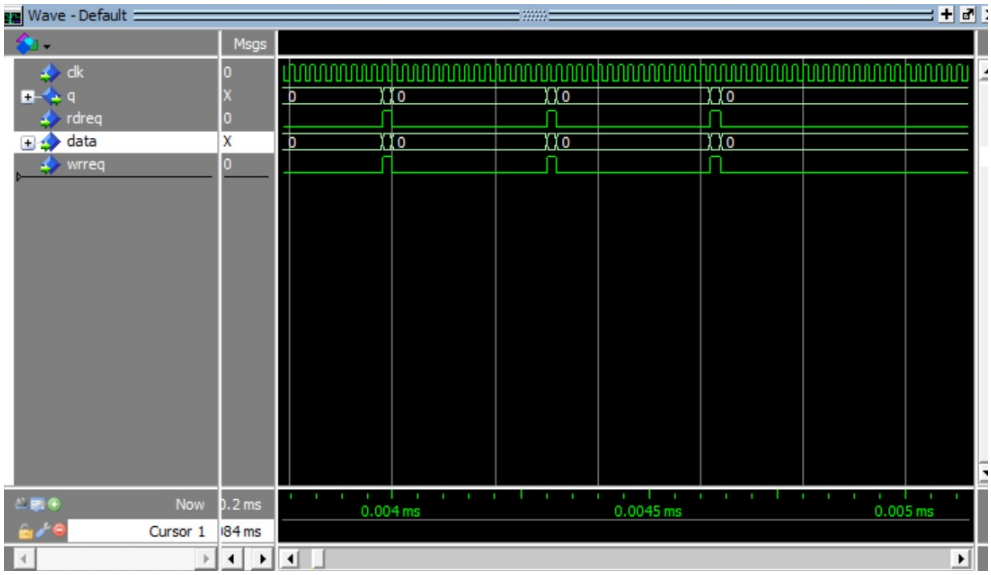

FIGURE 14
Time behavior Pixel Reducer

3.4.3 LOW LEVEL DESIGN

This component can be logically divided in different subdesign:

Counter Logic: State is set to 1 if the start signal is 1, the pixel FIFO has enough capacity for 64 pixels, and it is not in the situation Wait_Request and Write at 1 (it means another pixel is being written). The write signal gets activated at the following rising edge of the clock, when the value of the new pixel is ready in RdData. The counter gets updated every time RdFIFO is set to 1, since a new pixel gets read from the FIFO. The output address is obtained by masking the last 5 bits of the counter, shifting left of 2 positions (to transform from 2 pixels to 1 byte) and summing the result to cam_address.

State Logic: The pixels are output in burst of 64, after the 32nd transfer the state gets reset back to 0, waiting for the setting condition to start a new cycle.

End of Transfer Logic: The counter value gets subtracted to the bytes length divided by 4 (right shift 2), if the result is equal to 0 it means that the last pixel was transfer, status goes to 1 and the counter gets reset to FFFFFFFF (this way the first byte to be read puts counter to 0). Status also put start to 0.

The two diagrams below show the architecture and the time behavior of the component:

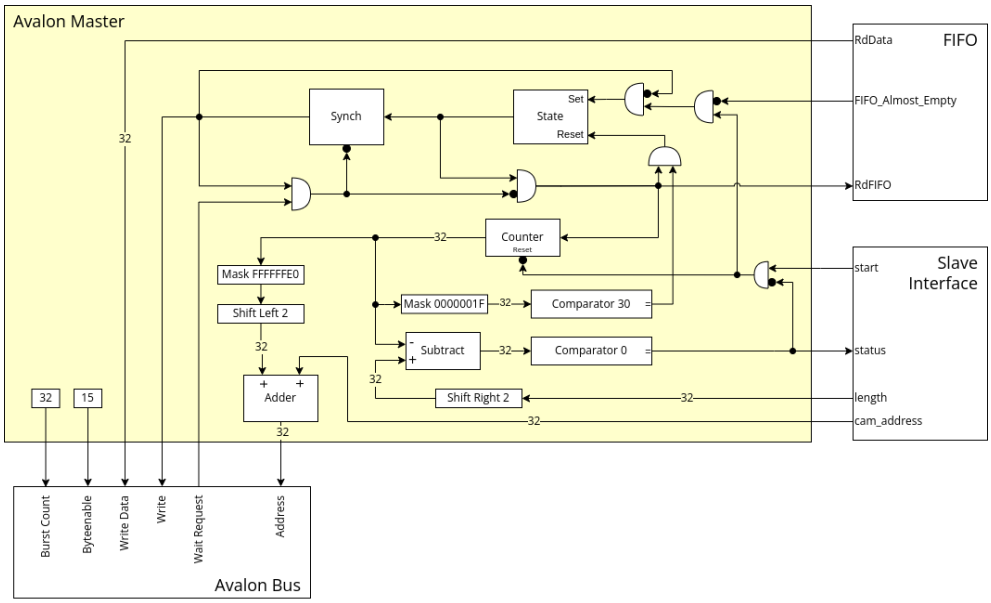

FIGURE 15
Low Level Master Architecture

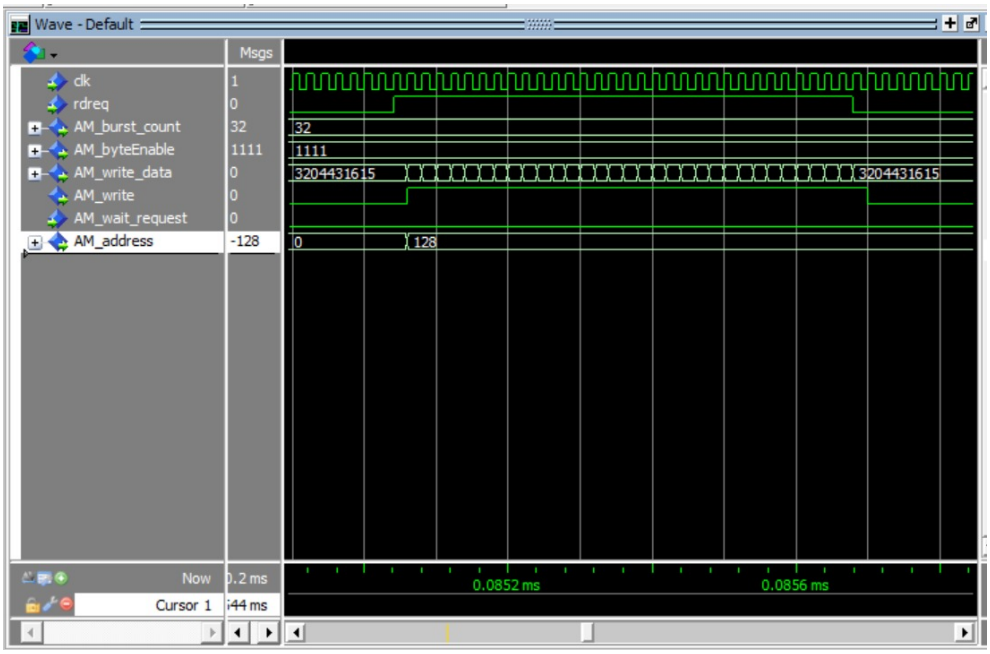

FIGURE 16
Time Behavior of a Avalon Master Operation

4 PROCESSOR SOFTWARE CONFIGURATION AND CONTROL

In order for the camera component to work correctly, some configuration is required to initialize the camera, to start the image capture and to export the image to a binary file.

4.1 CAMERA CONFIGURATION

In order to use our camera controller, you need to reset our components and the camera before the execution. The following options are required to properly set the camera.

- Row_size (R0x03) to 1919 to set the vertical resolution to 240 .
- Column_size (R0x04) to 2559 to set the horizontal resolution to 320.
- Row_bin (R0x22 [5:4]) and Row_skip (R0x22 [2:0]) to 3 to enable the binding between adjacent rows.
- Column_Bin (R0x23 [5:4]) and Column_Skip (R0x23 [2:0]) to 3 to enable the binding between adjacent columns.
- R0x00A [6:0] to 8 in order to divide the input clock by 16.
- R0x035 to 16 in order to increase the global gain and have brighter image.

For all the remaining registers the default settings can be used, to ensure them it is sufficient to reset the camera before the execution.

4.2 ACQUISITION START

To start the acquisition some registers of the camera controller slave interface need to be set:

- cam_address: we need to set the starting address where to save the captured image inside the SDRAM.
- cam_length: the number of bytes we wish to save - 1, for a 240x320 image the length is 240x320x2-1.
- cam_start: this register must be set last and starts the capture. When the register is back to 0 it means that the capture of the image is complete.

4.3 IMAGE SAVING

The captured image can saved on the PC, thanks to this it can be exported to an image format.

To save it to the PC the addresses on the SDRAM can be read from the processor using the command IORD_16DIRECT(address, offset). Some binary operations are necessary to change the format to 24 bits and save the 3 individual values of RGB as an array. The array can be saved in a binary file using the command fwrite().

Once the image is saved on the computer, it can be exported to png by using a python script that puts the binary file data into a numpy array and uses openCV to export it.

4.4 FINAL STATE MACHINE

The component can work in pair with the LCD component to get the image from the camera and display it on the LCD screen.

The cycle starts with a camera read, once the read is done (when the register cam_start is back to 0), we start a write on the LCD display, and start a new read on a different address in the SDRAM memory.

The cycle can be summarized in the following diagram:

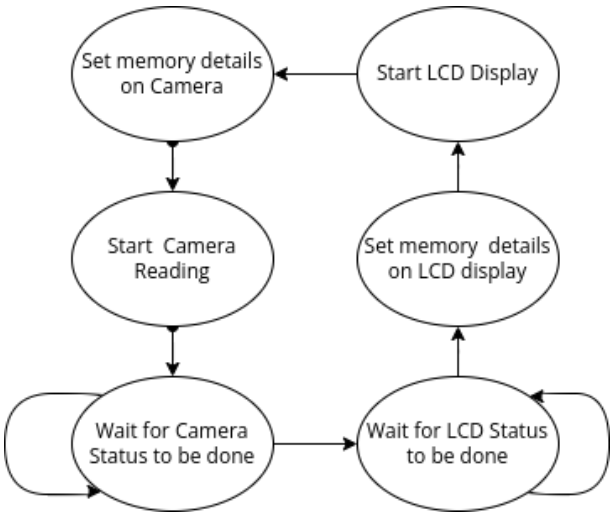

FIGURE 17
High Level Flowchart

5 RESULTS

The camera system works and correctly saves images in the memory. Below are some shots taken using the camera system, exported to a binary file and shown using openCV.

FIGURE 18
Images taken by the camera system

Can you guess the object in the picture taken by the camera? PS: it has strings =)