Ecole Polytechnique Fédérale de Lausanne

CS-473

Embedded Systems

Lab 3: Camera conceptual design

Report

Authors:
Victor Casas - 237770
Hugo Miranda Queiros - 336706

Supervisors:
Prof. René Beuchat

Due December 11, 2021

Contents

1 Problem statement **3**

2 High level architecture **3**

3 Low level architecture **6**
3.1 Camera pixels reading and formatting 6
3.2 Memory organisation 8
3.3 Camera configuration 9
3.4 Description of the Camera Interface 10
3.5 Description of the Avalon Master 10
3.6 Description of the Avalon Slave 11
3.7 State Machine of our Component 11

List of Figures

1 ***Board and camera pins*** 3
2 ***I2c controller*** 3
3 ***High level diagram*** 4
4 ***Camera control*** 5
5 ***Camera control IP component*** 5
6 ***Camera pixels organisation*** 7
7 ***Bayer configuration*** 7
8 ***FIFO description*** 8
9 ***Organisation of the memory*** 9
10 ***Signals on Avalon master*** 10
11 ***State machine of camera controller*** 11

1 Problem statement

The goal of this lab was to propose a detailed design for an FPGA-based system which can interface with the TRDB-D5M camera on the DE0-Nano-SoC. Due to the large amount of data that needs to be moved to/from these peripherals, we implemented a custom IP component with a master interface to automatically handle the transfer of image frames from the camera to memory.

2 High level architecture

The TRDB-D5M is a 5 megapixel digital camera available as an extension card for Terasic FPGAs. It connects to a development board through a 2x20 pin GPIO connector. In Figure 1, one can see how we will connect the pins of the camera on the GPIO_1 connector on the development board (DE0-Nano-SoC).

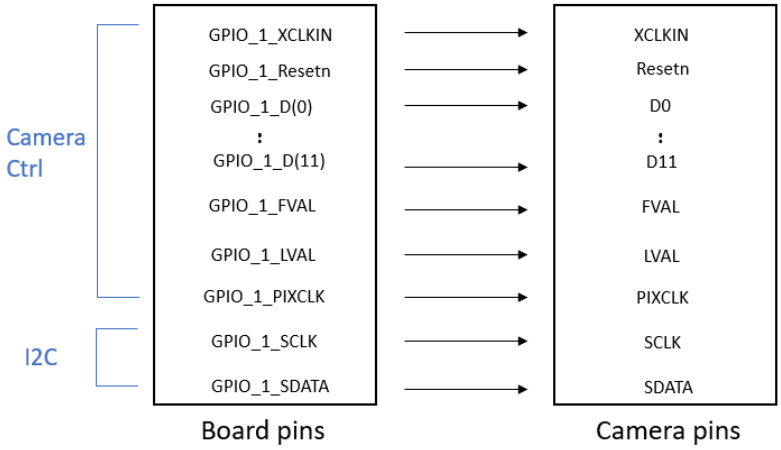

Figure 1: ***Board and camera pins***

A provided I2C controller will also be used to communicate with the camera for configuration purposes. One can also see in Figure 2 how it will communicate with the Camera.

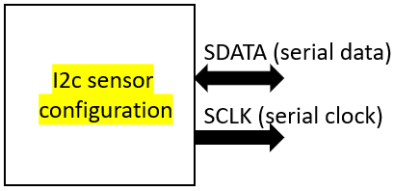

Figure 2: ***I2c controller***

In Figure 3 the high level diagram of our full system with the camera and the LCD is represented.

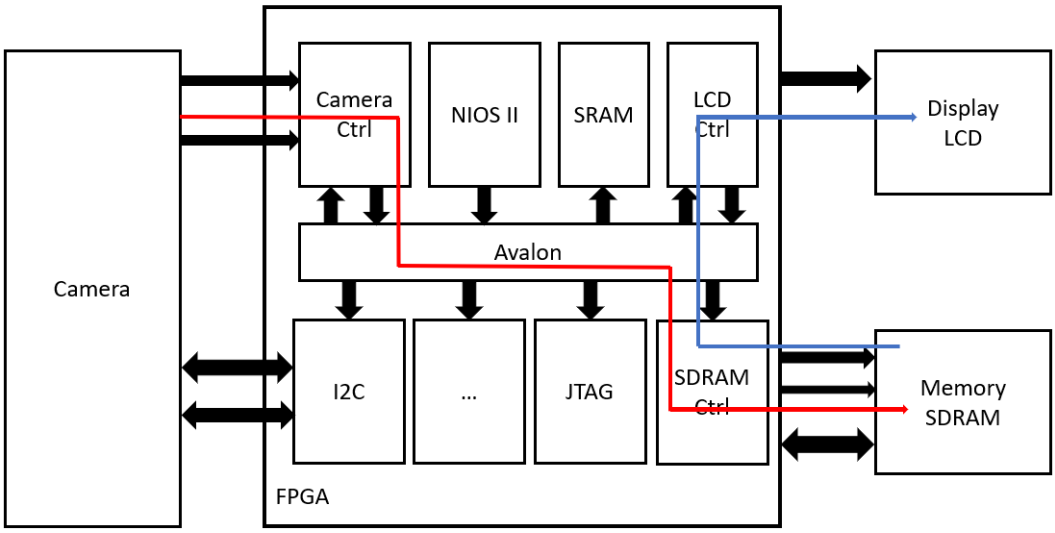

Figure 3: ***High level diagram***

To read the pixels from the Camera, convert them to the format and resolution of the LCD and to store them in the SDRAM memory, we created a custom IP component named Camera_Ctrl (Camera Controller). This component is directly connected to the camera through a conduit and to the Avalon bus through a Master and Slave interface.

Our Camera_Ctrl IP component is composed of the three main blocks in Figure 4 :

- one Avalon Master to send the pixels read from the Camera to the SDRAM memory through the Avalon Bus.
- one Avalon Slave that will be used to configure/read the registers of Camera_Ctrl and to receive the orders from the NIOS CPU to start/stop the storage of new frames or to know how to access memory.
- one Camera Interface with three FIFO :
  - 2 entry FIFO that both receive, one row of color pixel of each pair of rows, and then empty themselves in parallel in a component that will convert the color pixels sent by the Camera into the resolution and the format that is compatible with the LCD.
  - 1 exit FIFO that stores the final pixels that will be sent to the SDRAM memory.

The whole Camera_Ctrl component will be a functional DMA unit to allow the processor to offload tasks of packet data transfer (here the pixels of the camera).

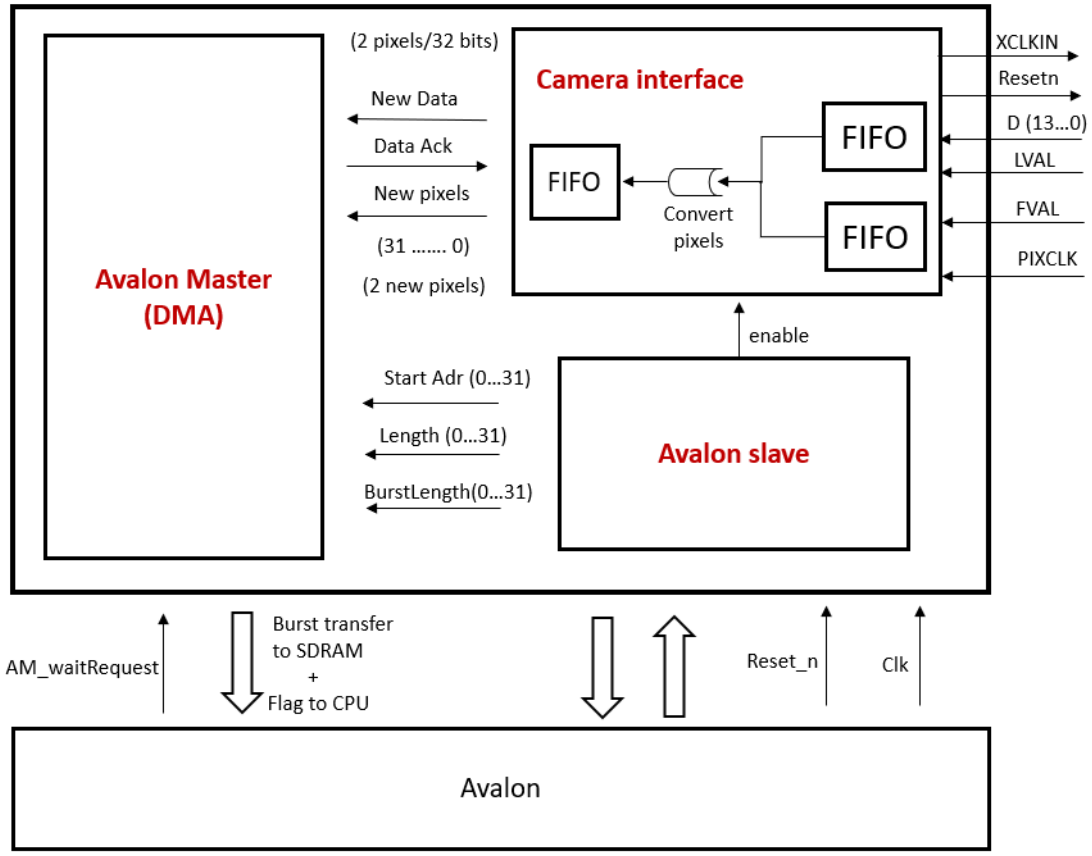

Figure 4: *Camera control*

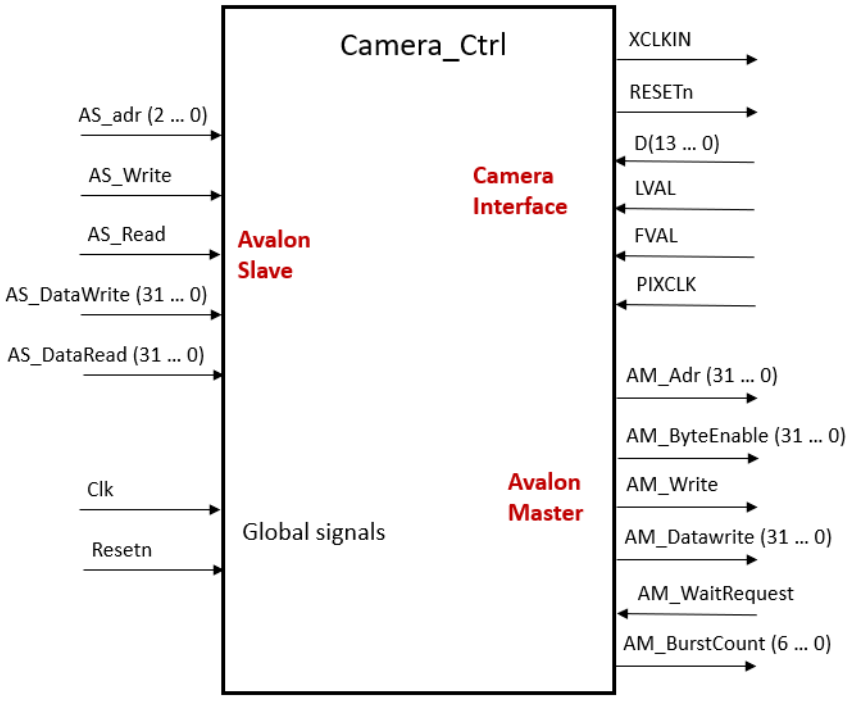

Figure 5: *Camera control IP component*

3 Low level architecture

Our Camera_Ctrl component has a register interface that can be configured by the NIOS CPU to start/stop the storage of new frames or to know how to access memory. We are going to use the following register interface :

- RegAdr - A location in the register interface where the start address of the new frame to write in memory is updated by the CPU.
- RegLength - A location in the register interface where the length of the frame to write in memory is given by the CPU.
- RegEnable - A location in the register interface where the status of the acquisition of data from the Camera by "Camera Interface" is enabled or stopped by the CPU.
- RegBurst - A location in the register interface where the length of the burst transfer in words to write in memory is given by the CPU.

Since there are 4 registers in our register map, the smallest address bus width we could use to index all registers is 2 bits, but we will use 3 bits in case we need more registers for debugging purposes. Table 1 summarizes the write and read behavior of these registers.

Address	Write register	Writedata[31...0]	Read register	Readdata[31...0]
0	RegAdr	→ iRegAdr	RegAdr	iRegAdr →
1	RegLength	→ iRegLength	RegLength	RegLength →
2	RegEnable	→ iRegEnable	RegEnable	iRegEnable →
3	RegBusrt	→ iRegBurst	RegBurst	iRegBurst →
4	-	Don't care	-	0x00
5	-	Don't care	-	0x00
6	-	Don't care	-	0x00
7	-	Don't care	-	0x00

Table 1: Register map of the camera

3.1 Camera pixels reading and formatting

The TRDB-D5M camera has by default a 2592x1944 active color pixel image as one can see in the Figure 6.

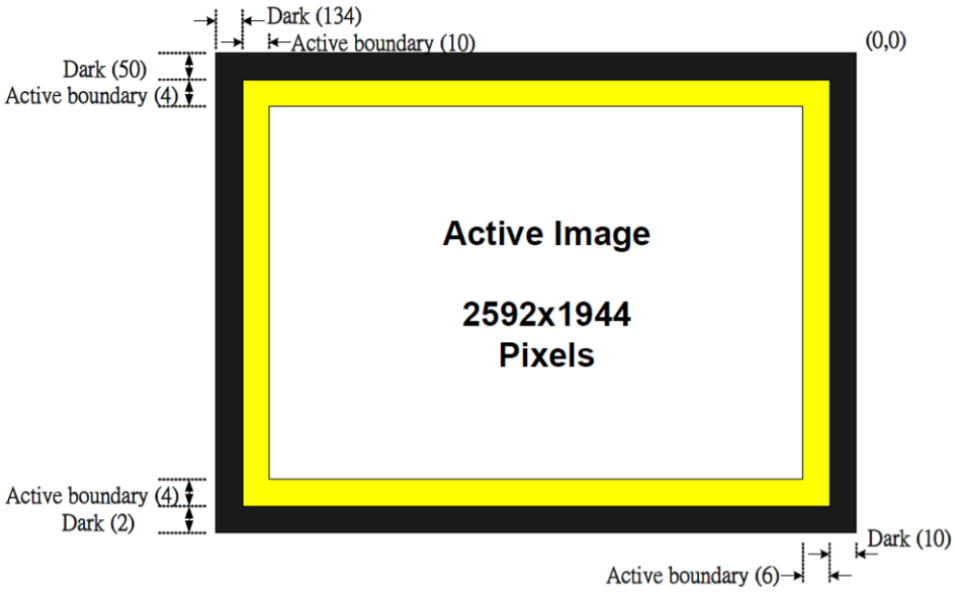

Figure 6: ***Camera pixels organisation***

These pixels are output in a Bayer pattern format consisting of four "colors"—Green1, Green2, Red, and Blue (G1, G2, R, B)—representing three filter colors each in a size of 12 bits as one can see the Figure 7.

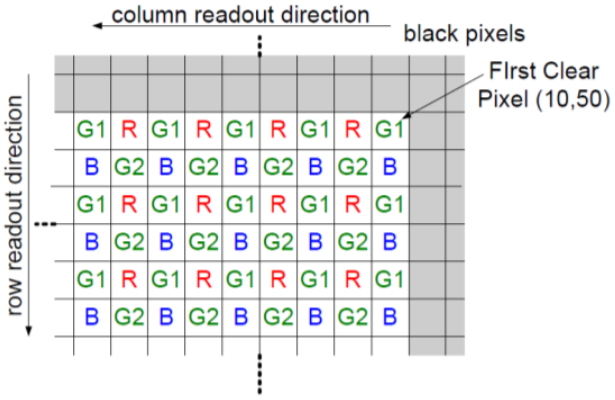

Figure 7: ***Bayer configuration***

However, the LCD driver that will be created by the other group is only compatible with RGB pixels of total size 16 bits and with a resolution of 320x240. To achieve this format, the first thing to do is to configure the camera to output a frame of resolution 640 x 480. To do so, we will order the camera to apply binning x4 to the columns and the rows, i.e that there are 4 columns to be read and averaged per column, and 4 rows to be read and averaged per row. This operation allows us to reduce the impact of aliasing introduced by the use of skip modes. Then, the camera will also apply skipping x4 to the rows and the columns, i.e that for 1 pair of rows the next 3 pairs are skipped and for 1 pair of columns the next 3 pairs are skipped.

After doing all these operations, we can deduce from the datasheet that with a clock of 50

MHz, the camera will output new frames at a rate of 40.3 fps (as opposed to the 77.4 fps with 96 MHz clock in the datasheet) .

After that, the camera will give us rows of 640 bayers pixels. Therefore, we will input the first row of each pair of rows in the first entry FIFO and the second row in the second entry FIFO. When the two rows are stored, the signal FIFO_Full (see Figure 8) of both FIFO will enable a component to read in parallel the two rows and will therefore average the two green pixel (G1, G2) to form only one green pixel, and will concatenate the R, G, B 12 bit bayer color pixels to form only one RGB pixel of 36 bit. This will be done with binary operation over both rows in one pair. Then the component will convert all the 36 bit RGB pixels into 16 bit RGB pixels with 5 bit for blue, 6 bits for green, and 5 bits for red. To do this conversion we decided to keep only the most significant bits as we suppose that we will be using our camera in good lighting conditions.

Every time that two 16 bits RGB pixels are ready, they will be sent to the exit FIFO. For this report, we chose a burst count of 64, but we will tune this value in the real system. So when this FIFO will have at least 128 pixels ready (i.e 64 words of 32 bits ready to be sent) the signal NewData will be set to '1' thanks to FIFO_Almost_Empty signal (see Figure 8) and will warn the Avalon Master that a burst transfer of 64 new 32 bit words of data is possible.

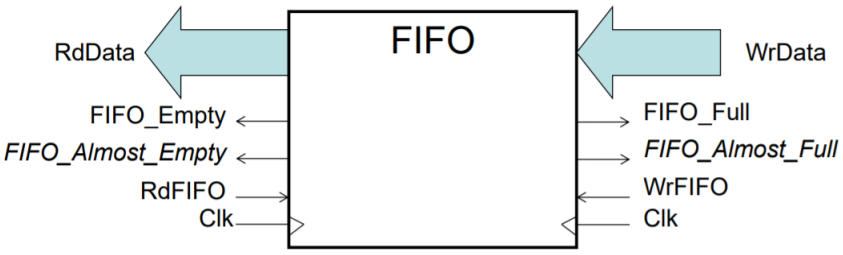

Figure 8: ***FIFO description***

3.2 Memory organisation

We chose with the group that is creating the LCD driver to store two frames in memory. Therefore we will use two buffers, one for each frame (Double buffering). As each frame needs to have 320 RGB pixels of 16 bits per row, that means that we have 160 words of 32 bits per row. As a frame has 240 rows the total size of a frame in the memory will be : 160 * 240 = 38400 words of 32 bits, i.e 38 400 * 32 = 1 228 800 bits.

As we will send the pixels in burst tranfers of 64 words of 32 bits (this value may change with testing in the real system) we will need to do : 38400/64 = 600 burst transfers to write a full frame in memory. The Figure 9 shows an illustration of our memory.

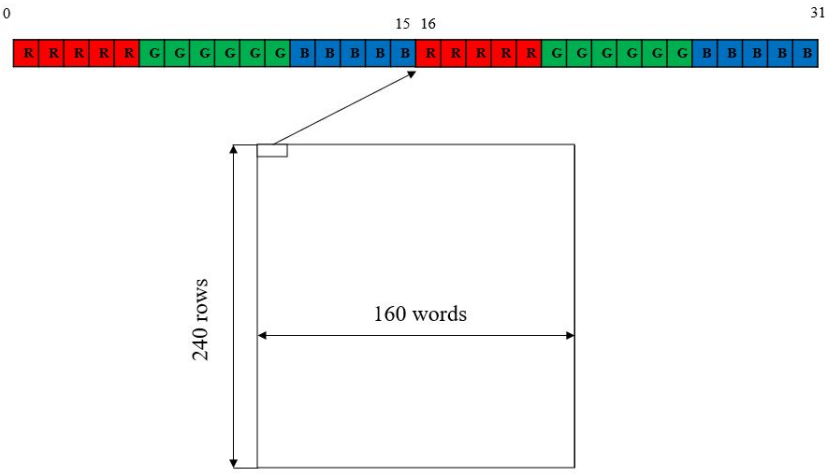

Figure 9: ***Organisation of the memory***

3.3 Camera configuration

In order to achieve the pixel reading and formatting explained before with the right resolution, we need to configure the registers of the camera with particular values.

First, the datasheet gives us in Table 2 below, the values to put into the registers in order to achieve binning x4 and skipping x4 both on rows and columns with a final resolution of 640 * 480.

Resolution	Frame Rate	Sub-sampling Mode	Column _Size (R0x04)	Row_ Size (R0x03)	Shutter_ Width_ Lower (R0x09)	Row_ Bin (R0x22 [5:4])	Row_ Skip (R0x22 [2:0])	Column_ Bin (R0x23 [5:4])	Column_ Skip (R0x23 [2:0])
640 x 480 VGA	77.4	bining	2559	1919		3	3	3	3

Table 2: Datasheet of the camera for bining 4x and skipping x4 for both columns and rows

We will also put the register R0x00A bit 15 (Invert Pixel Clock) at 1 in order to capture new data on the rising edge of the clock. We will let the default values of register R0x001 Row_Start = 54 and register R0x002 Column_start = 16 because they already respect all the requirements for the binning and skipping that we will need, i.e they are both even and Column_start is a multiple of 16. Finally, to respect the minimal blanking requirements we will write in the register R0x005 Horizontal Blank = 906 and in the register R0x006 $VerticalBlank = max(8, SW - 1919) + 1$, with SW that corresponds to the shutter width value and this one will be adjust at the end in the real system by writing in the registers R0x008 Shutter Width Upper and R0x009 Shutter Width Lower.

3.4 Description of the Camera Interface

When the register of the Avalon Slave RegEnable equals to '1', then the Camera Interface is reading the data of pixels. To read this data the signlas FVAL and LVAL of the camera allow to know when a row is beginning or ending and the same for the frame. Each pixel data D[11 ... 0] is catched on the rising edge of the clock of PIXCLK. The two entry FIFO will both receive, one row of each pair of rows, and then when they both receive a full row, they will empty themselves in parallel in a component that will convert the color pixels sent by the Camera into the resolution and the format that is compatible with the LCD.

3.5 Description of the Avalon Master

When the Camera Interface is enabled, the exit FIFO will store the pixels that are ready to be sent, and every time that 128 pixels are ready to be sent, i.e 64 words of 32 bits are ready, the DMA will send those words in a BurstCount of 64 (64 words of 32 bits) to the SDRAM memory through the Avalon Bus. We can see an example of transfer in Figure 10. Every time that a burst transfer is made, we will then need to increment the writing address by BurstLength * 4 units (i.e 64 * 4 = 256) before starting the next burst transfer.

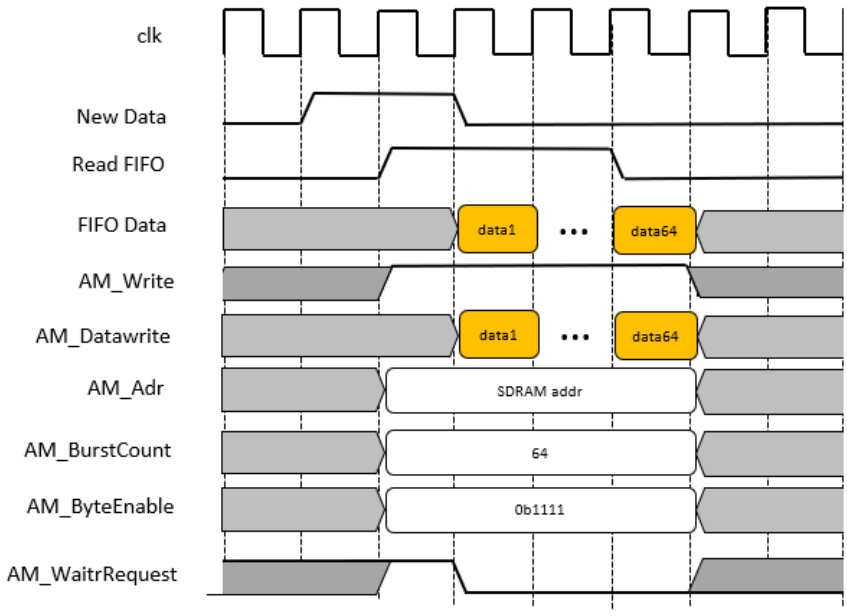

Figure 10: ***Signals on Avalon master***

When it finishes writing a frame in the memory, RegEnable is set to '0' and the Master sends an interrupt flag to the CPU to notify it that it finishes writing a frame and that it wants the permission to write a new frame in the second buffer. Similarly, the LCD controller will also send an interrupt flag to the CPU when it finished to read a frame in a buffer.

3.6 Description of the Avalon Slave

The CPU will update the registers of Camera_Ctrl to start/stop the storage of new frames or to know how to access memory. Every Time, an interrupt flag is sent by the Master of the camera controller or the LCD controller, these registers will be updated in consequence.

3.7 State Machine of our Component

In Figure 11 one can see a detailed state machine (FSM) diagram showing how our interface sequences its various sub components to achieve its function.

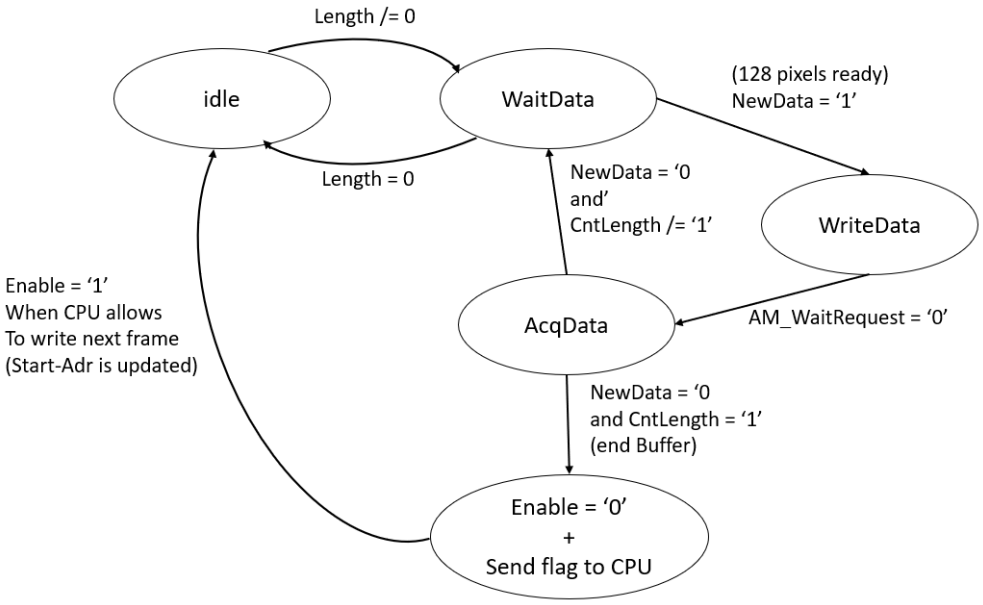

Figure 11: ***State machine of camera controller***