CS473: Embedded System Lab3: Camera Controller

Menghe Jin
menghe.jin@epfl.ch

Yuxuan Wang
yuxuan.wang@epfl.ch

1 System Overview

In lab 3.0, we focus on camera controller design that captures data from peripheral to memory. Figure 1 exhibits the minimum architecture of the system, which is made up of a CPU that manipulates operations on the peripherals, an on-chip DRAM that store the data, JTAG-UART to debug, a provided I2C module to define the camera behavior, and our designed camera controller.

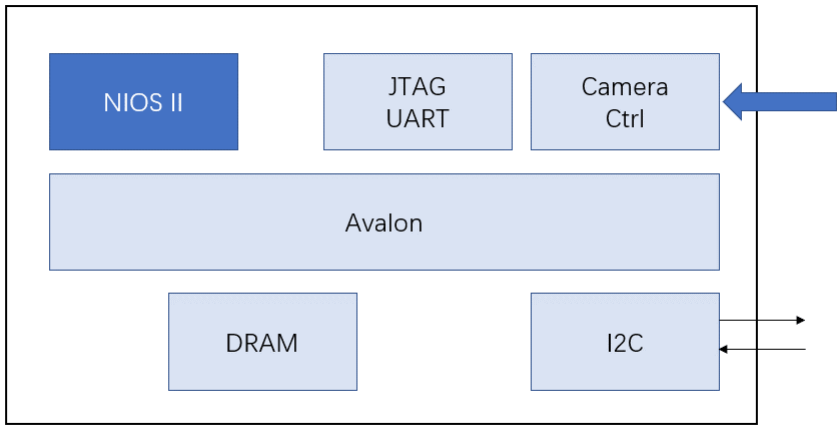

Figure 1: System diagram

2 Camera Configuration and Registers

Readout data format

We use default readout order to receive the data from the camera, that is, from pixel (16, 54) and read line by line. The row and column size can be configured by the processor, while these row/column size corresponds to the column/row of the LCD. Such strategy is used because LCD reads continuously the side with shorter field view and the camera reads the opposite. By restricting the active area of the camera, the output from the camera can be line by line instead of column by column. In this data format, the pixel data read out does not consider the blank area. All the active pixel data are located in the valid image area. Binning 2X is adopted by setting row/column bin and row/column skip to 1 so that the width and height is always four times of the specified value. Then four color pixels are to combined into one 3-channel pixel and sent to memory.

Output data timing

We use the default pixel output timing modes. The output images are divided into frames, which are further divided into lines. Only the output when both Fvalid and Lvalid are set would consider to be

valid by the camera controller interface. Meanwhile, we will adopt the default Read Mode for LValid and FValid, where LValid is set only when FValid is set.

Camera operating modes

There are five operating modes provided by TRDB-D5M camera hardware. We choose to use ERS Snapshot by writing the register snapshot = 1 using I2C module. So each frame is initiated by a trigger and output one at a time. The triggering occurs only when the TRIGGER pin is asserted. The camera controller is responsible for when to get new frame.

The registers for the camera are designed as in **Appendix A**.

3 Camera Controller

Figure 2 gives the detailed architecture of the modules we will implement. It includes a camera interface to trigger the camera and receive the data, an I2C module to configure the camera and an data acquisition module with avalon slave and master peripherals that transport data from the camera controller to memory.

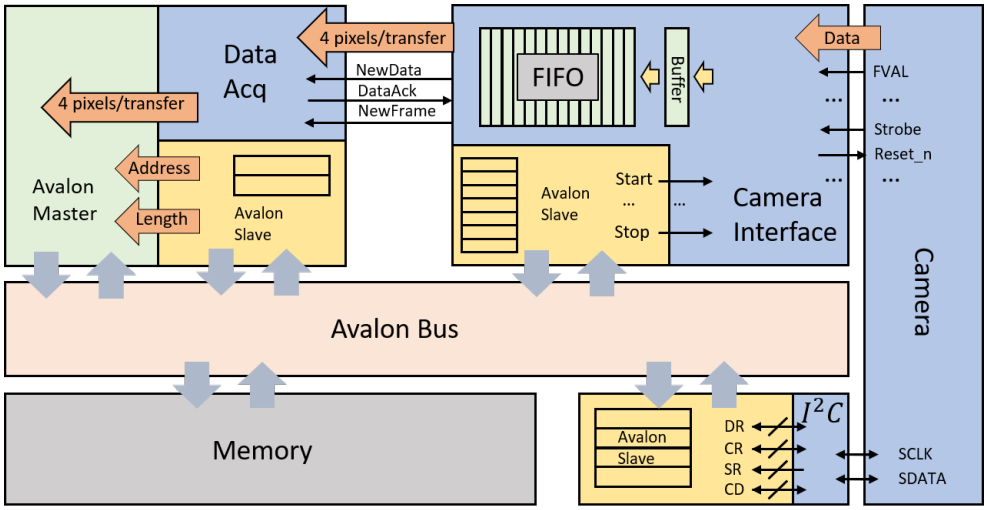

Figure 2: Detailed architecture of modules

3.1 Camera interface

The camera interface would trigger the camera for a new frame, process the raw input data and store them temporally. Every data pixel output by TRDB-D5M only represents one shade in Red, green, or blue. To obtain RGB image, we must transform a 2×2 pixel square(i.e. four 12bits data) to a 3-channel RGB pixel(32 bits).

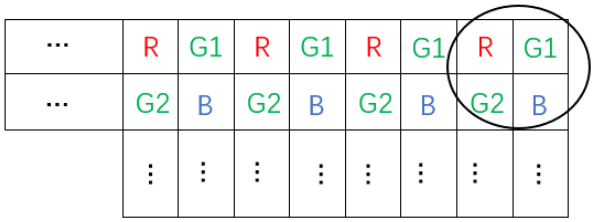

Figure 3: Input data format

Figure 3 illustrates the raw data format output by the camera and one proposed solution is shown in Figure 4. The resulting data from the four single colored pixels is 16-bit long, with 5 bits of red, 6 bits of green and 5 bits of blue. The five most significant bits of each color is used and the sum of two green colors is just 6 bits.

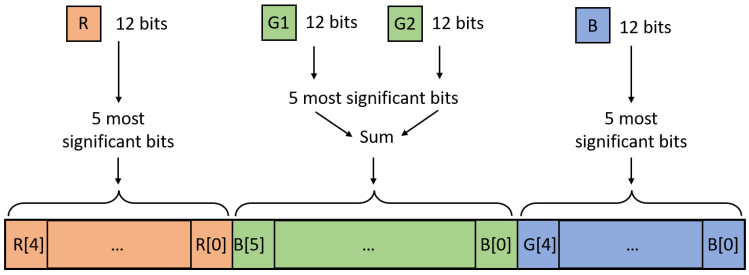

Figure 4: Output data format

To achieve the transformation, there are two buffers in FIFO interface, one is the line buffer, an array of 16 bits, to store the MSBs of the odd line of the image. Another is the process buffer. A FSM model of the two buffers are shown in Figure 5 and the steps to produce one 16-bit data is shown in Figure 6. When the line buffer is full, the upcoming data will be processed and then filled in the process buffer together with the data in the line buffer. The size of processing buffer equals to 16 bits.

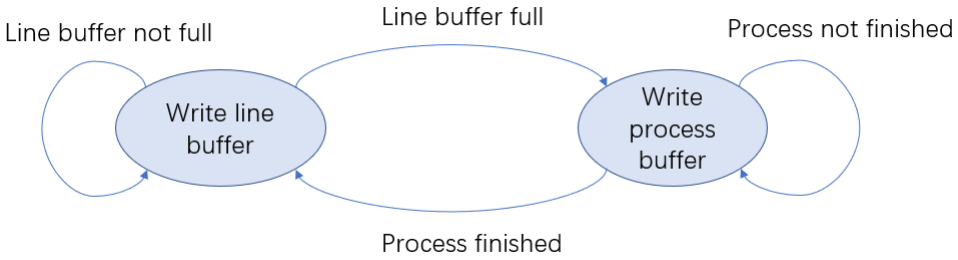

Figure 5: FSM for line and processing buffer

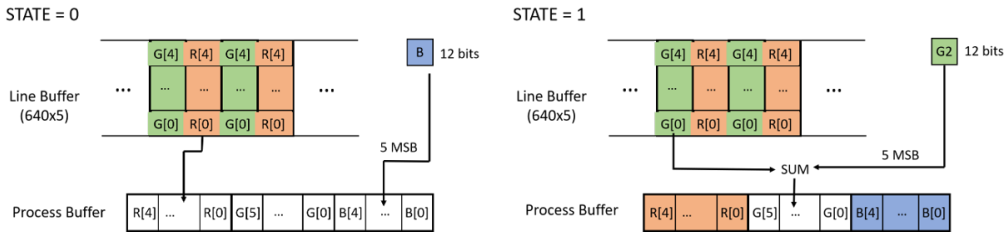

Figure 6: Steps for process buffer

We will use a matrix of 16-bit registers to store a frame (or less) for the function similar to a FIFO. While writing the line into the matrix, the camera interface can send the transformed data to the data acquisition module by stacking two 16-bit vectors in a line. The first-in-first-out property is kept by always increment the write and read index by 1. Valid data is ensured by keeping the read index no greater than the write index. Addition to writing while reading, we can also implement the case where reading only happens when all pixels in a frame are read. Since the transformation to the 16-bit data takes two clock cycles, the ideal data flow when processing the even lines at the write side of the matrix is like Figure 7. Once the data in the even line has all been processed, both line buffer and process buffer would be cleared and ready to receive the next line. When the reception of all lines in a frame success, camera interface would assert TRIGGER pin to get new frame from the camera.

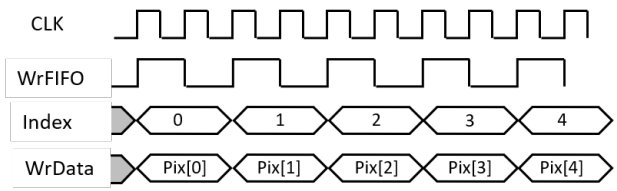

Figure 7: Writing into matrix

An Avalon slave interface is implemented for the processor to control the camera. The designed registers includes a RW **start** and **stop** to indicate whether to acquire data from the camera, a RO **status** to indicate if the camera is reading the line, waiting for a new line or waiting for a new frame and RW **row_size** and **column_size** to configure the active area of the camera.

4 Data acquisition module

A data acquisition module is implemented between the camera interface and the Avalon master interface. When the Avalon master is in transmission mode, the new data signal would be active if new data received, and the ACK would be asserted if receive successfully. An additional Avalon slave interface is added to this module and there are two registers in the Avalon slave peripherals as in Table 1. The processors can write to the Avalon master to specify the memory location and numbers of desired data.

Table 1: Register in avalon slave

Name	Offset	Data Width	Function
AcqAddress	0	32 bits	Start address of the memory to store data
BurstCount	1	32 bits	Numbers of data for the memory to store

For the FSM for avalon master transmission, we borrow the idea from the example design, shown in Figure 8.

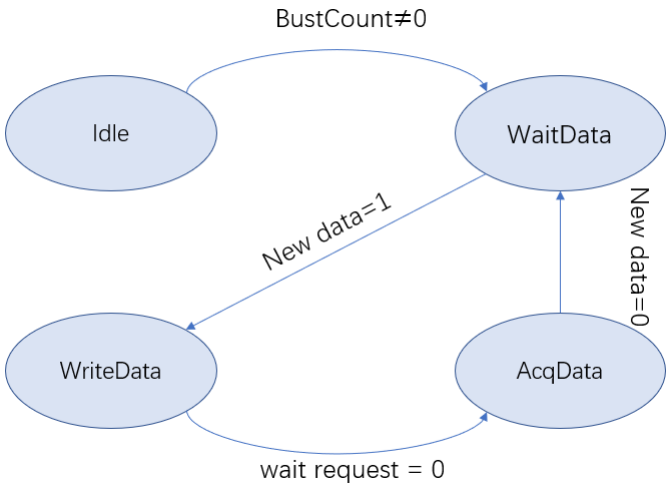

Figure 8: FSM for Avalon

When the desired data specified by BurstCount does not equal to zero, the Avalon master transform to waitData state from Idle. If new data is asserted, Avalon master would write data to memory in the next cycle. Since all bits of the 32-bit data is meaningful in our design, all ByteEnable signal will be set as 0xF. If wait request is asserted, the Avalon would maintain its states and data until the wait

request is negative.Then it transfers to AcqData status and record the success transmission. When New data is negative, it goes back to WaitData, and wait for the next transmission.

5 I2C module

The control registers of the camera are read and write with I2C protocol. We will adopt the provided I2C module together with some constant registers defined in C file to configure the camera.

Four Avalon slave registers are designed in the I2C module for the processor to control the module, which are listed in **Appendix B**. The functions for the processor to access these registers are also provided and we will use them directly. Since our FPGA clock frequency is 50MHz and the target clock frequency for I2C is 400kHz, we will initialize the module by setting the clock divider register to 125 or by using function *i2c_init* and set *i2c_frequency* to 50M. The interrupt will be disabled so the function *i2c_configure* will by executed with *irq* = False. For accessing the control and status registers, functions *I2C_WR_CONTROL*, *I2C_RD_CONTROL* and *I2C_RD_STATUS* are used.

Since the I2C control of camera registers involves the 8-bit device ID with read/write, the 8-bit register address and the 16-bit register content in order inside a single message, accessing multiple registers are not possible within a single message, so the registers are accessed one by one. To set up the camera as designed we will execute multiple *i2C_write_array* functions, which generate a complete message with both start and end signal. The generated signal with look like figure 9. The value for each register is listed in **Appendix A**.

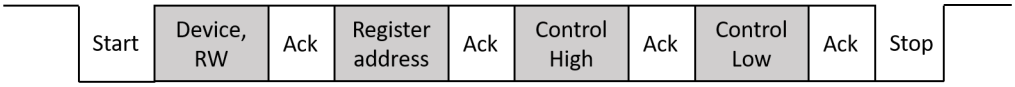

Figure 9: Control message from I2C

6 Pin assignment

The TCL scripts and top-level vhdl file used in this lab is the one forTRDB_D5M on GPIO_1.The following table shows the pin assignment between the board and the camera.

Table 2: Pin assignment

Pin on TRDB_D5M	Pin on board	Description
RESETn	GPIO_1_D5M_RESET_N	Reset signal
XCLKIN	FPGA_CLK1_50	Clock signal for camera
PIXCLK	GPIO_1_D5M_PIXCLK	Pixel clock
D[0:11]	GPIO_1_D5M_D	Pixel data
TRIGGER	GPIO_1_D5M_TRIGGER	Snapshot trigger
STROBE	GPIO_1_D5M_STROBE	Snapshot strobe
LVALID	GPIO_1_D5M_LVAL	Line valid
FVALID	GPIO_1_D5M_FVAL	Frame valid
SDATA	GPIO_1_D5M_SDATA	Set the camera register using I2C
SCLK	GPIO_1_D5M_SCLK	Set the camera register using I2C

Appendix A. Camera Control Registers

Register Address	Register Name	Value	Description
R0x000	Chip Version	Default	
R0x001	Row Start	Default	Valid pixels start from (16, 54).
R0x002	Column Start		
R0x003	Row Size	TBD	The row and column size will be 4 times the values specified by the CPU. Then row size will correspond to the side of LCD with 240 pixels.
R0x004	Column Size		
R0x005	Horizontal Blank	Default	We will not change the exposure or the frame rate of the camera.
R0x006	Vertial Blank		
R0x007	Output Control	Default	The output slew rate, PIXCLK slew rate are kept as default. The output FIFO on the camera module is not used. The chip is enabled so that analog-to-digital conversion can be made. The changes of control registers are not synchronized and they are executed once they are set by the CPU.
R0x008	Shutter Width Upper	Default	Shutter width are not changed in the lab.
R0x009	Shutter Width Lower		
R0x00A	Pixel Clock Control	Default	The LVAL, FVAL and D[11:0] is captured on the falling edge of PIXCLK so that they are stable at the rising edge of the PIXCLK. Since the default clock frequency(50MHz) of the FPGA falls into the working range of PIXCLK, we will use this one as the input external clock for the camera module and the clock divider is 0.
R0x00B	Restart	Default	Since we are setting the camera to Snapshot mode and the triggering totally depends on the TRIGGER pin, we will not using the Trigger bit. The Pause Restart is not activated, either.
R0x00C	Shutter Delay	Default	
R0x00D	Reset	0x0000	We will use the RESETn Pin for resetting, so this register is cleared.
R0x010	PLL Control	Default	Since PIXCLK is designed to be at the same frequency as the XCLK, PLL module is not activated and the three registers are set as default.
R0x011	PLL Config 1		
R0x012	PLL Config 2		
R0x01E	Read Mode 1	0x0116	We want the LVAL signal is produced only when FVAL is set, so the 11th and 10th bit are cleared. The default sense mode of the TRIGGER pin is used. The Snapshot mode is used so the 8th bit is set and the strobe signal is enabled. The shutter, exposure time and strobe start/end will be set as default.
R0x020	Read Mode 2	Default	The rows and columns are not mirrored. The dark rows and columns are not shown. Row BLC is used to digitally compensate for differing black levels. Column averaging is done in binning.
R0x022	Row Address Mode	0x0011	Binning 2X is used so row/comumn bin and row/column skip are set as 1.
R0x023	Column Address Mode	0x0011	
R0x02B	Color/Global Gains	Default	
...			
R0x035			
R0x049	BLC Related	Default	
...			
R0x064			
R0x0A0	Test-Pattern Related	Default	
...			
R0x0A4			

Appendix B. I2C Slave Interface Registers

```
-- Registers description:
--
-- Adr RW Name
-- 00  RW DR - transmit and receive data register
-- 01  RW CR - control register (changed to RW mode)
-- 10  RO SR - status register
-- 11  RW CD - clock divisor
--
-- SR - status register bits
--
--  +---------+-------+-------+-------+-------+
--  |bit 7..5 | bit 3 | bit 2 | bit 1 | bit 0 |
--  +---------+-------+-------+-------+-------+
--  | UNUSED  | TIP   | IPE   | BSY   | LAR   |
--  +---------+-------+-------+-------+-------+
--
-- TIP   - transfer in progress
-- IPE   - interrupt pending
-- BSY   - I2C bus busy
-- LAR   - last acknowledge received
--
-- CR - control register bits
--
--  +---------+-------+-------+-------+-------+-------+-------+
--  | bit 7..6| bit 5 | bit 4 | bit 3 | bit 2 | bit 1 | bit 0 |
--  +---------+-------+-------+-------+-------+-------+-------+
--  | UNUSED  | IEN   | WR    | RD    | STA   | STP   | ACK   |
--  +---------+-------+-------+-------+-------+-------+-------+
--
-- ACK    - Acknowledge bit for reading
-- STP    - Generate a I2C Stop Sequence
-- STA    - Generate a I2C Start Sequence
-- RD     - Read command bit
-- WR     - Write command bit
-- IEN    - Interrupt Enable
--
-- To start a transfer WR or RD *MUST* BE set.
-- When command transfer has started TIP goes high
-- and write to CR are ignored until TIP goes low.
-- At end of transfer IRQ goes high if interrupt is enabled (IEN=1).
```