Mini-Project: Digital Camera Design

Due: 11:59PM, Monday, March 30th 2015
Submission only by Canvas

Team Members **U#**	Christopher Frazier U58831412	Bassam Saed U52607130	John Gangemi U68714612
Work Distribution	33.3%	33.3%	33.3%

Total number of person hours spent:

62 hrs

Exercise difficulty (Easy, Average, Hard):

Hard

Issues you ran into:

- Timing issues between the image sensor and the memory/controller interface
  - (explained further in part B.1)
- The very open-ended nature of this project left us wasting time considering irrelevant parts.
- Project time management suffered due to heavy workloads in other classes.

Any suggestions to improve this project:

- More descriptive requirements for block parts
- Less redundancy in terms of questions being asked

Any other feedback:

- This project is very open-ended and inspires a lot of creativity and critical thinking. We all want to actually make the camera now and see if it really works. More descriptive requirements for the parts available for use would have saved us a lot of time considering our other options.

Part A Front End Interface Design

A.1(1 pt.) Specification Analysis: Analyze the design specification and identify all requirements. What additional features would you like to see in the camera? (Maximum 1 Page)

- The camera user should be able to focus on a subject or scene of interest and press a shutter button to take a snapshot.
- The resulting image should be stored in camera's local memory.
- The resolution of the image should be at-least 1,024H x 1,024V (~1 million pixels).
- The user should be able to take several images and store them in the memory.
- The total budget for the prototype cannot exceed $200.
- The user should be able to interface the camera with a PC to upload the images.
- Within the given budget, you need to:
    - (a) maximize the number of pictures that you can store in the camera; and
    - (b) maximize the rate at which snapshots can be taken successively.

Additional features we would like to see include:
- Optimized image capture speed, and memory size in price range

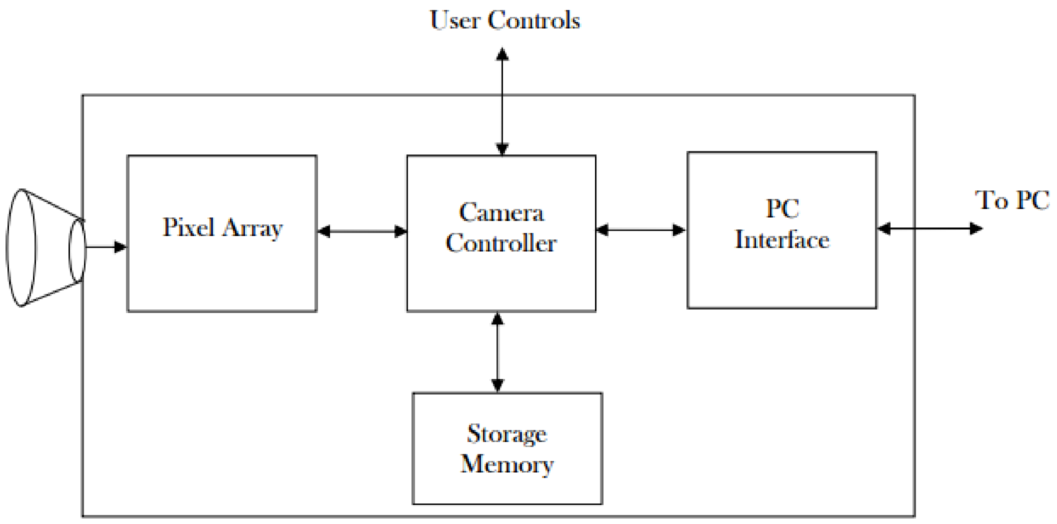


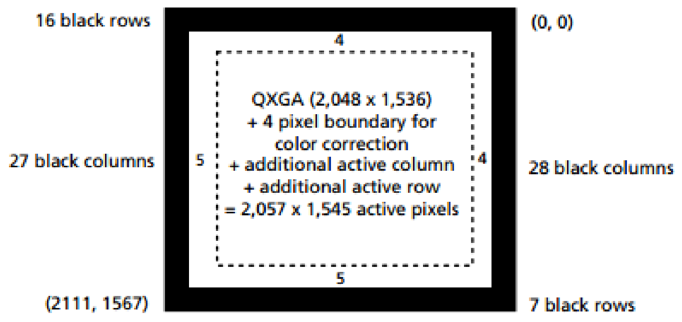
Figure 1: High-level block diagram of the simple digital camera

A.2(1 pt.) Read the datasheet and analyze the sensor array features. Summarize the features of the sensor array relevant to your design. (Maximum 1 page)

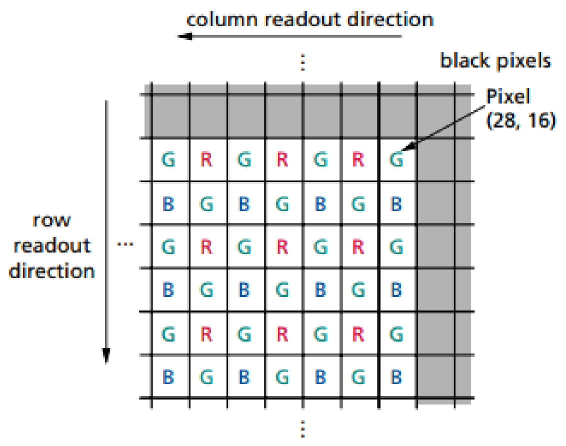
- Pixel array is configured as 2,112 columns by 1,568 rows
- Active pixels: 2,048H x 1,536V

- Columns from 0 through 27 and from 2,085 through 2,111 and rows from 0 through 15 and from 1,561 through 1,567 are optically black (These can be used to monitor the black level)
- Four pixel boundary around the QXGA (2,048 x 1,536) image to avoid boundary effects during color interpolation and correction.
- Shutter type: Global reset release (GRR), electronic rolling shutter (ERS)
- Maximum data rate/ master clock: 48 MPS/48 MHz
- Frame Rate: Programmable up to 12 fps
- ADC resolution: 10-bit, on-chip
- Programmable through a two-wire serial interface
- Programmed by the user for frame size, exposure, gain setting, and other parameters
- FRAME_VALID and LINE_VALID signals are output on dedicated pins, along with a pixel clock that is synchronous with valid data
- Uses a Bayer color pattern
    - The even-numbered rows contain green and red color pixels, and odd-numbered rows contain blue and green color pixels
    - The even-numbered columns contain green and blue color pixels; odd-numbered columns contain red and green color pixels

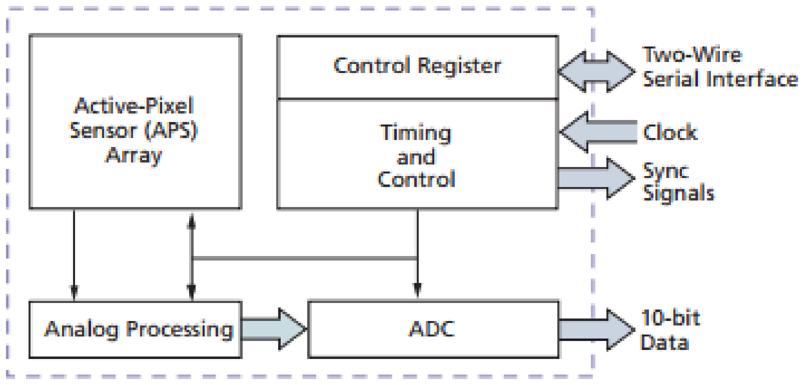
Pixel Array Description



Pixel Color Pattern Detail (Top Right Corner)



Block Diagram



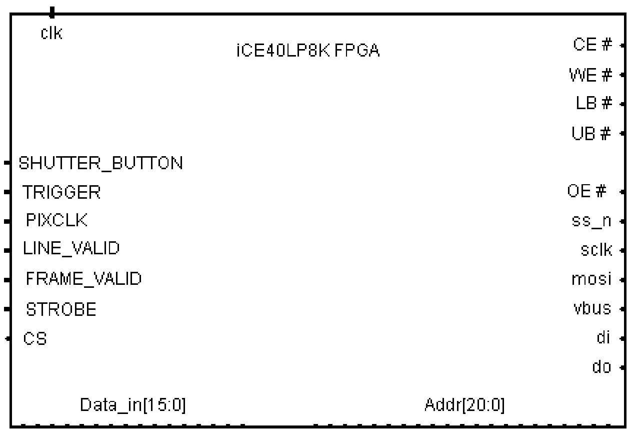
A.3(1 pt.) Define the port interface of the Camera controller block. Briefly describe the purpose of each port. (Maximum 1 page)

# indicates active LOW

SHUTTER_BUTTON: User activated button that triggers an image capture sequence
TRIGGER: Activates snapshot sequence.
PXLCLK: Pixel clock: pixel data outputs are valid during falling edge of this clock.

- Frequency = master clock

LINE_VALID: Output is pulsed during line of selectable valid pixel data
FRAME_VALID: Output is pulsed during frame of valid pixel data
STROBE: Output is pulsed to indicate sensor reset operation of pixel array has completed
CS: Chip Select: Activates data storage in memory
CE #: Chip Select: Activates SRAM cell
WE #: Write Enable: (SRAM)
LB #: Lower Byte Select (SRAM)
UB #: Upper Byte Select (SRAM)
OE#: Output Enable (SRAM)
ss_n: SPI slave select input (USB Controller)
sclk: SPI clock input (USB controller)
mosi: SPI slave input
vbus: VBUS Sensing Input (USB Controller)
di: Data In (SD Memory)
do: Data Out (SD Memory)
Data_in [15:0]: Receives 16-bit parallel data to be shifted into SD memory
Addr [20:0]: Sends 21-bit address to SRAM cell to map temporary memory



A.4(2.5 pts.) Analyze and define the timing interface required between the Pixel Array and Camera Controller blocks. (Use as many pages as needed)

Figure 7: **Timing Example of Pixel Data**

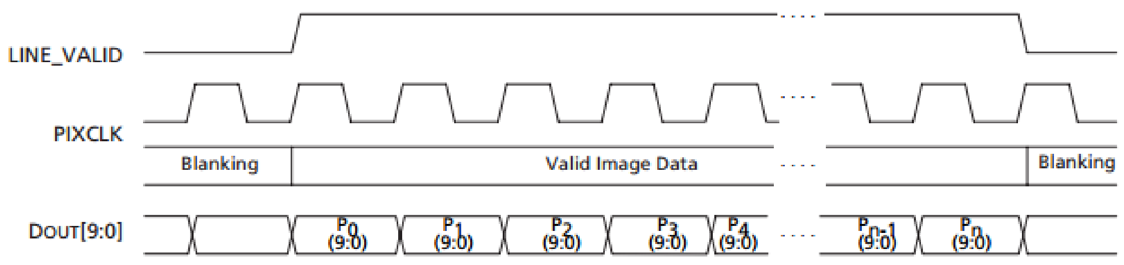


Figure 8: **Row Timing and FRAME_VALID/LINE_VALID Signals**

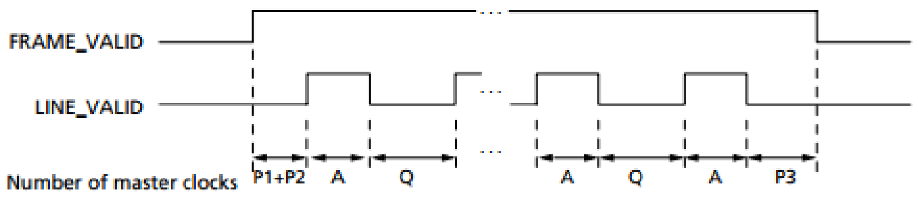
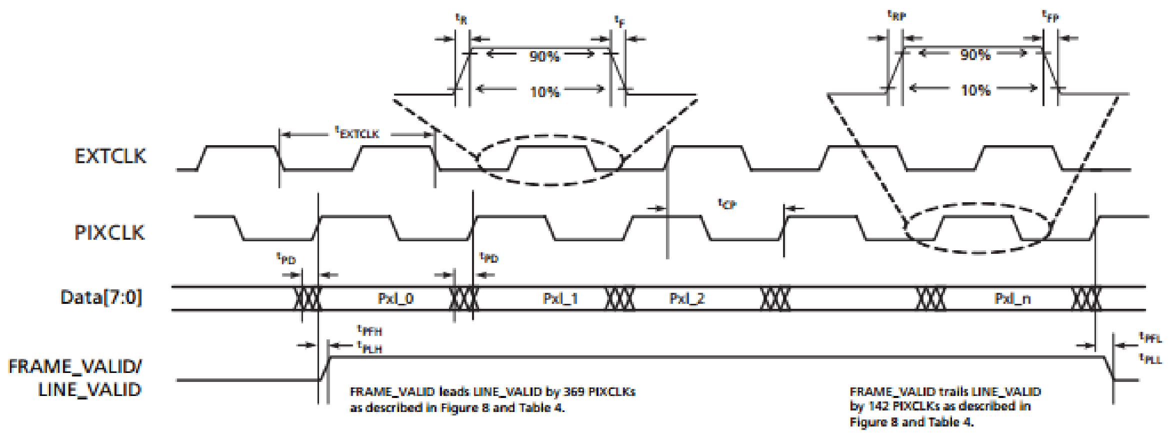


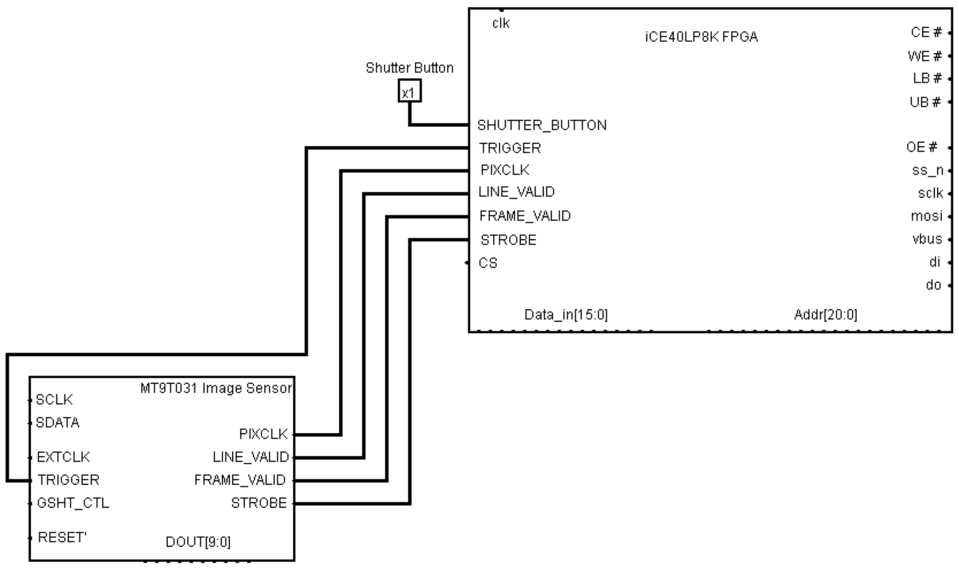
Figure 22: I/O Timing Diagram

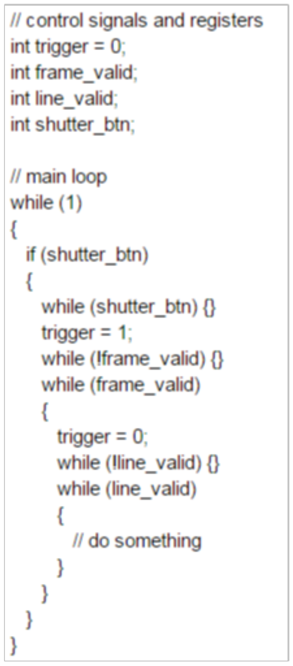

EXTCLK
PIXCLK
Data[7:0]
FRAME_VALID/
LINE_VALID
tR
tF
90%
10%
tRP
tFP
tEXTCLK
tCP
tPD
tPFH
tPLH
tPFL
tPLL
Pxl_0
Pxl_1
Pxl_2
Pxl_n
FRAME_VALID leads LINE_VALID by 369 PIXCLKs as described in Figure 8 and Table 4.
FRAME_VALID trails LINE_VALID by 142 PIXCLKs as described in Figure 8 and Table 4.

When asserting 'trigger' the image sensor with collect data from the array of pixels and assert 'frame_valid' immediately afterwards. Only when acquiring data from active columns of pixels will the 'line_valid' signal be asserted. If 'line_valid' is asserted then every negative edge of the 'pixclk' will produce 10 bits of pixel data. It is critical that the pixel data is captured on the following rising edge of the 'pixclk'.

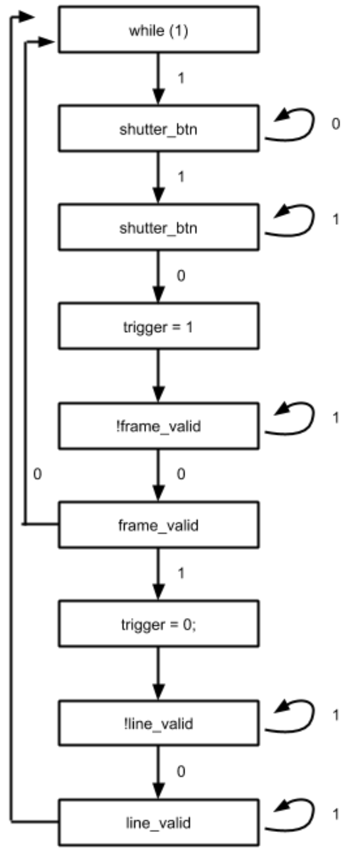
Since the camera controller operates at 100 Mhz and the image sensor operates at 48 MHz the possibility of capturing valid pixel data is quite high.

A.5(2.5 pts.) Implement an RTL design satisfying the port and timing interfaces determined in Questions (3) and (4). For the controller, you can stop at the state diagram. (Use as many pages as needed)

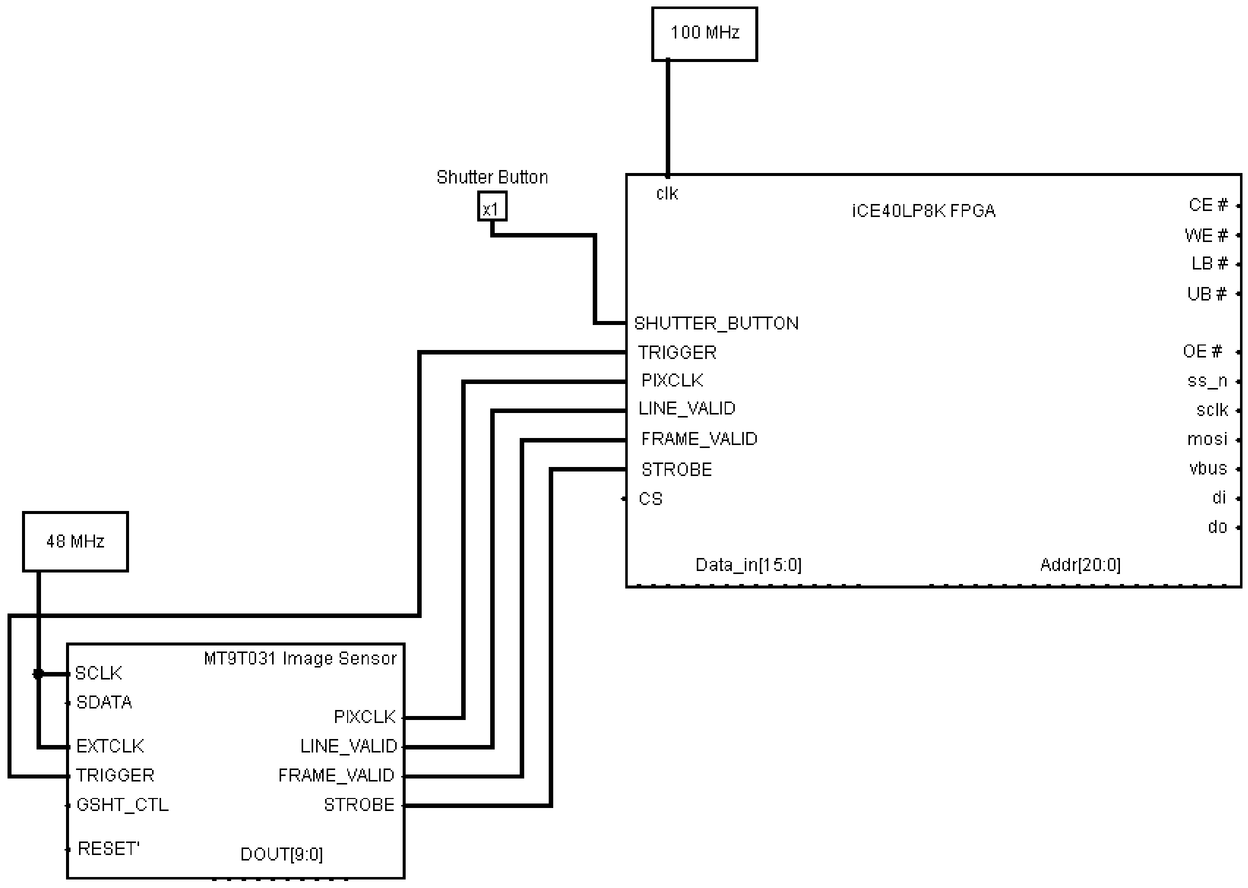




```
// control signals and registers
int trigger = 0;
int frame_valid;
int line_valid;
int shutter_btn;

// main loop
while (1)
{
  if (shutter_btn)
  {
    while (shutter_btn) {}
    trigger = 1;
    while (!frame_valid) {}
    while (frame_valid)
    {
      trigger = 0;
      while (!line_valid) {}
      while (line_valid)
      {
        // do something
      }
    }
  }
}
```



A.6(1 pt.) Draw a detailed schematic of the partial design of the front-end as well as user interfaces. Identify any other components that are required (for example, crystal-controlled oscillator). Show these components as well in the schematic.(Maximum 1 page)



A.7(1 pt.) Estimate: (a) how long it will take for one image capture; and (b) the approximate dollar cost to implement the front-end interface.(Use as many pages as needed)

(A)
Image Capture Time (as noted in table 4 of the MT9T031 datasheet) :

- 48 MHz / 3,997,168 total pixel clocks = 83.27 ms total frame times

(B)
iCE40LP8K-CM121 FPGA Chip Cost: \$9.39
MT9T031: 1/2-Inch 3-Mp Digital Image Sensor: \$15.86
Total: \$25.25

Part B Memory & PC Interface Design

B.1(1 pt.) Memory Component: Choose an off-the-shelf memory component that can be used as internal memory for the camera. List the memory components that you have researched and provide arguments for your memory choice.

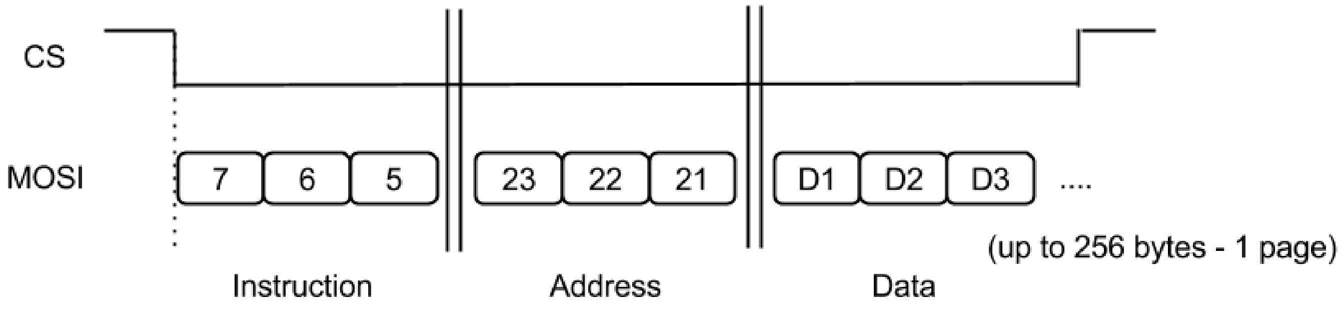
We chose to use the Apacer Industrial 4GB Micro SD Card 3.0 to store each picture, and a 32Mb SRAM cell to store each frame before the controller shifts it into memory. We chose the microSD because it uses an SPI interface to store non-volatile flash memory. Not only was it much easier to implement than other options because of the SPI interface, it was also much cheaper. We needed to temporarily store each frame in SRAM because the controller would take too many clock cycles to configure the SD memory for a write each time the sensor sent data for a pixel.

As you can see in the diagram below, the SD memory must serially receive an 8-bit write instruction followed by a 24-bit address specifying where in memory to send the data. This means that 32 clock cycles pass before the SD memory can store the data from the first frame. Moving the 8-bit RGB pixel data into temporary SRAM allows us to give the controller time to configure the SD memory for a write.

We chose SRAM because it allows us to use a 16-bit parallel interface to store two pixels quickly. The method for writing two pixels to a single word is implemented by manipulating the access to the least and most significant bytes of a word in memory.

Ex.

- Assuming the address pointer in memory is referencing the first word (16-bits = 2 bytes)
- The first bits of pixel data are written to the least significant byte of a word
- The second bits of pixel data are then written to the most signicant byte of the same word
- Increment the address pointer to the next word (+16-bits)



B.2 (1 pt.) Memory Component Features: Read the datasheet of the selected memory component and briefly summarize its features.

MicroSD Features include:

- Sustained Read: Up to 19 MB/sec
- Sustained Write: Up to 12 MB/sec
- Supports SD SPI mode
- Operating frequency: up to 100 MHz
- Power consumption*
    - - Operating: 47 mA
    - - Standby: 150 uA

B.3(1 pt.)Port Interface:Define the port interface of the memory with the camera controller. Briefly describe the purpose of each port.

<u>microSD</u>
CS: Active low chip select enables communication with a slave device from the master
DI: Data input used to recieve serial data
DO: Data output used to transmit serial data
SCLK: Clock provided to slave device

- For SPI protocol the maximum clock rate is defined as 25 MHz according the Apacer microSD datasheet

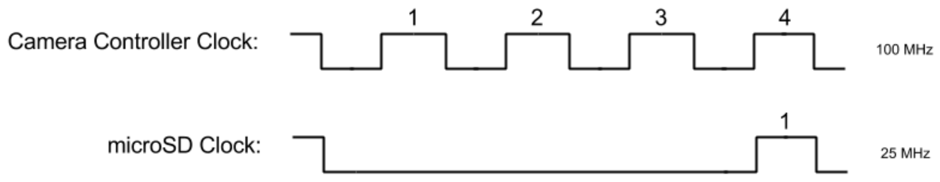
When writing to the microSD module the lsb of ‘Data_in’ is assigned to the output signal ‘di’ of the camera controller.

When reading from the microSD module the output signal ‘DO’ of the microSD is sent to the input ‘do’ of the camera controller.

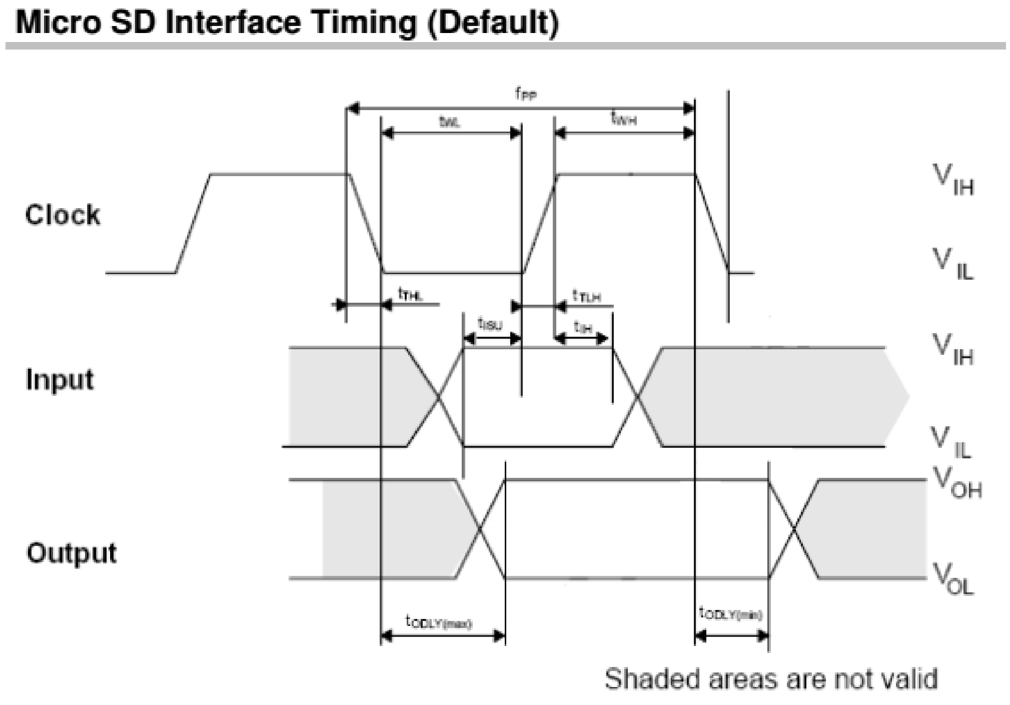
<u>32Mbit SRAM</u>
CE#: Active low signal enables communication with master device
OE#: Active low signal configures I/O pins 15-0 as outputs
WE#: Active low signal configures I/O pins 15-0 as inputs
LB#: Active low signal that allows access to the least sigificant byte
UB#: Active low signal that allows access to the most significant byte
I/O Pins [15:0]:Input and output pins to read/write a word of data
Address Inputs [20:0]: Define where to write/read data in memory

B.4(1 pt.)Timing Interface: Analyze and define the timing interface required between the memory and the rest of the system.

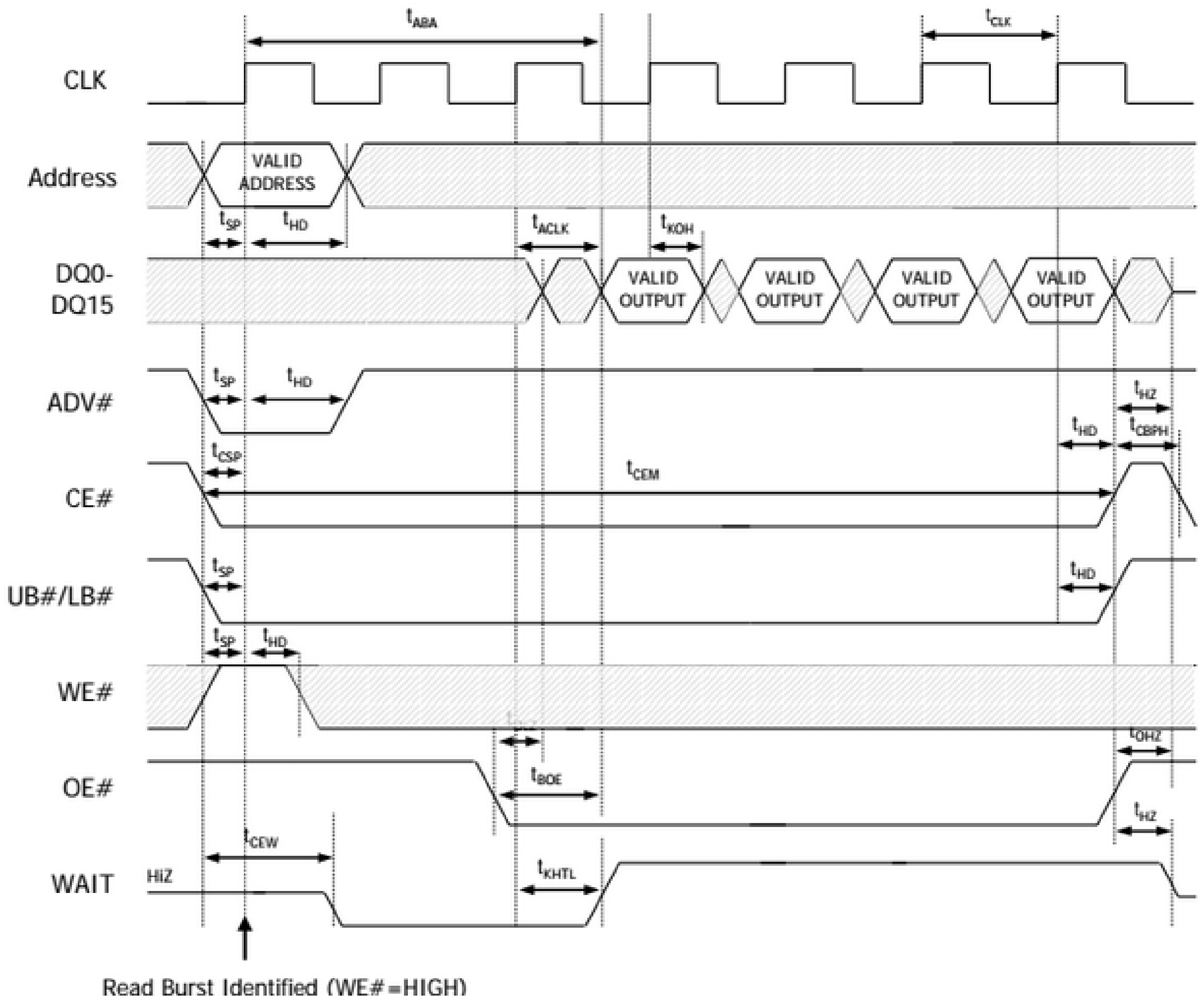
The microSD uses a 25 MHz clock per implementation of SPI protocol therefore it is necessary that the camera controller provide input and accept output every **four** clock cycles given the 100 MHz clock rate of the camera controller.



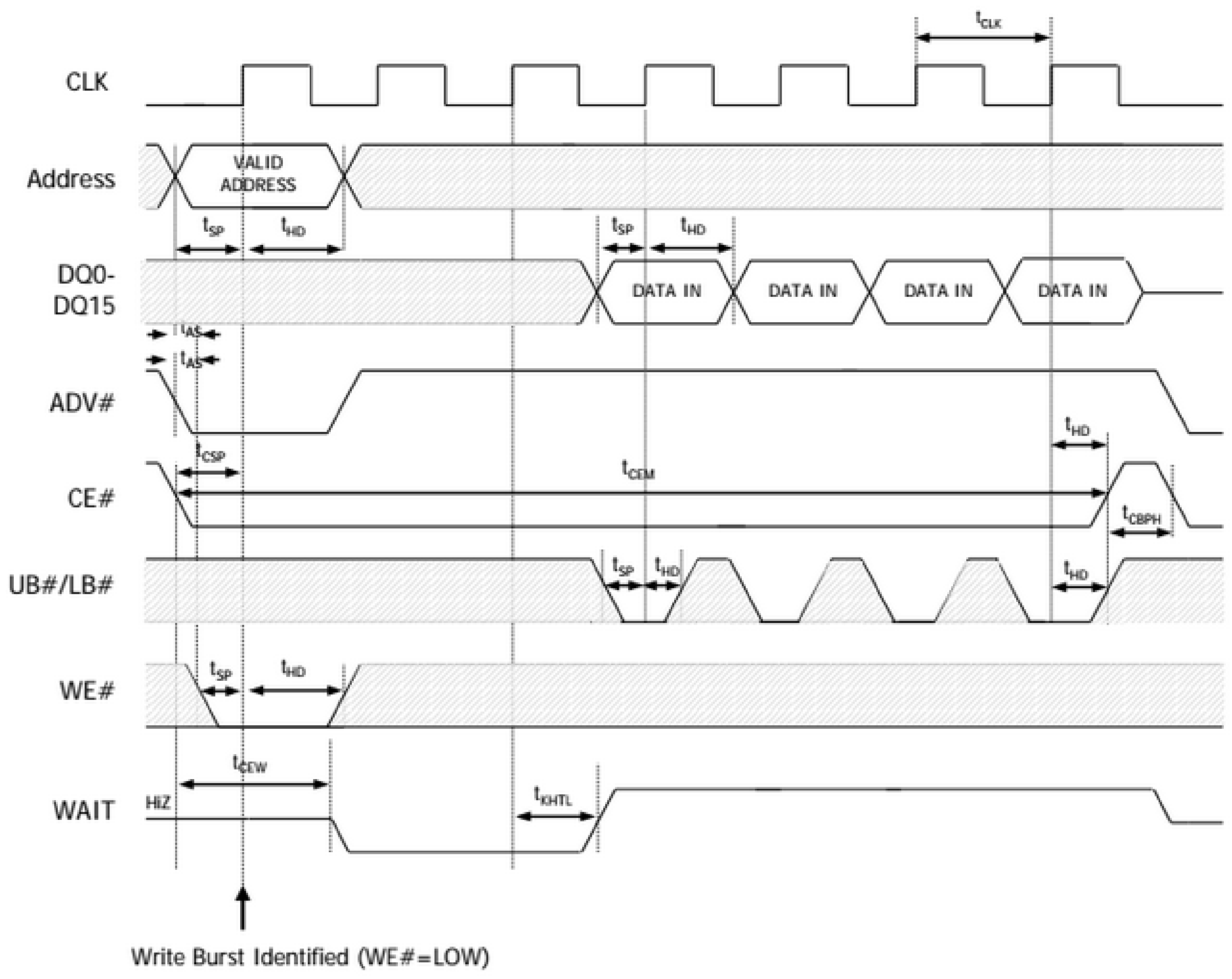
The 32Mbit SRAM operates at 48 MHz to interface nicely with the image sensor so that pixel data generated on the negative edge of ‘pixclk’ can be written to the SRAM module on the accompaning rising clock edge. The 21-bit address line provided to the SRAM module is driven by a 21-bit register in the camera controller, namely ‘Addr’. Every two ‘pixclk’ cycles (16-bits) ‘Addr’ is incremented by 16 to point to the next word (row) in the SRAM module.



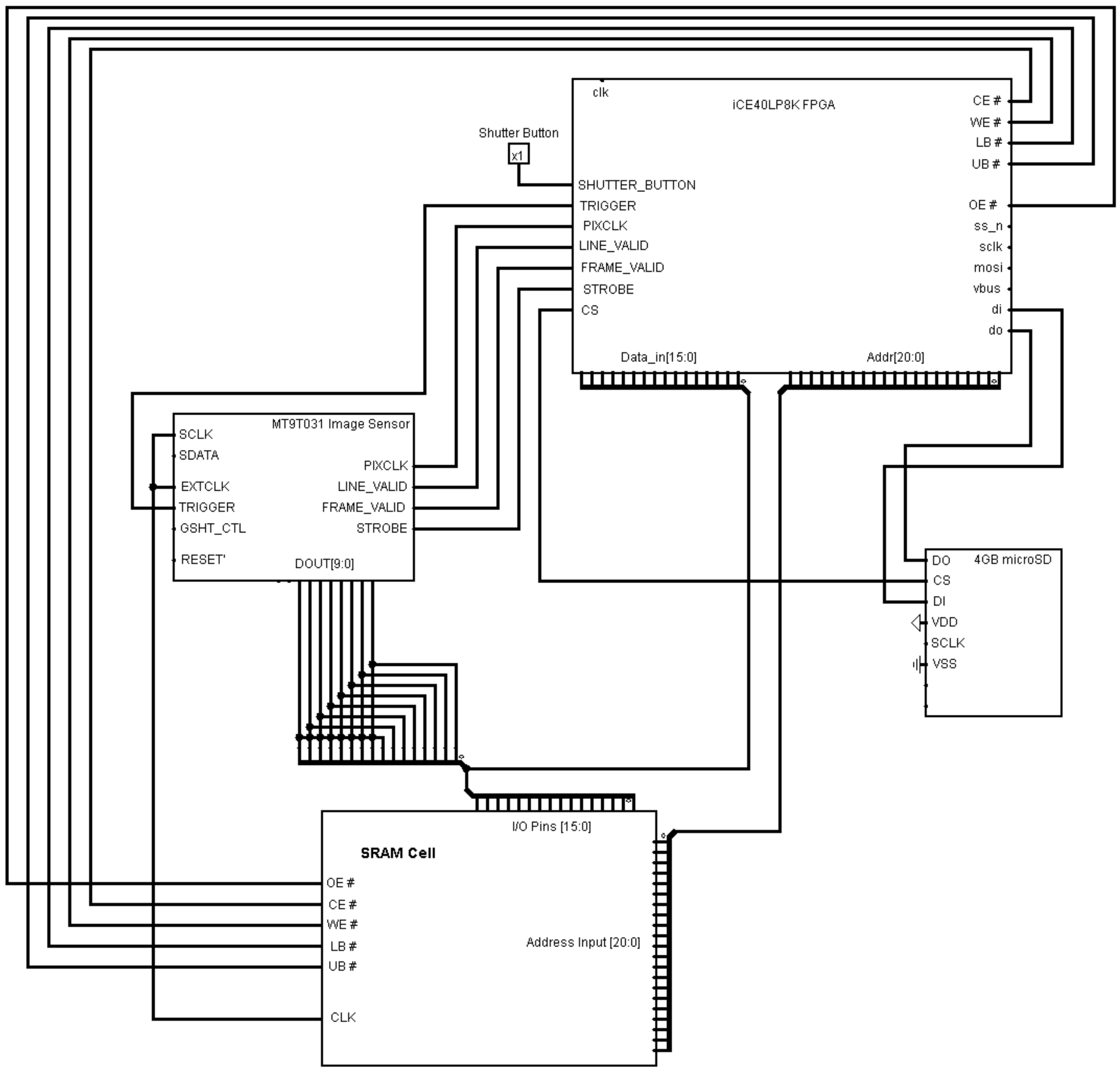
Synchronous Read Access Timing For SRAM Memory



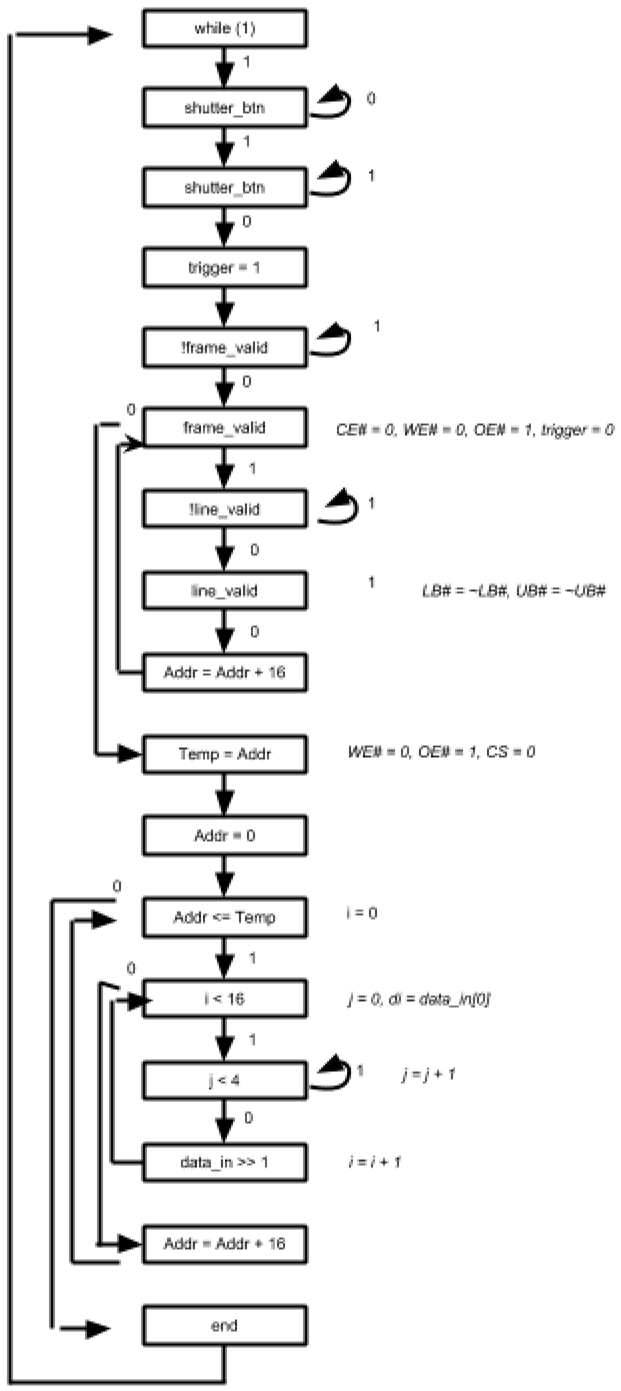
Synchronous Write Access Timing For SRAM Memory



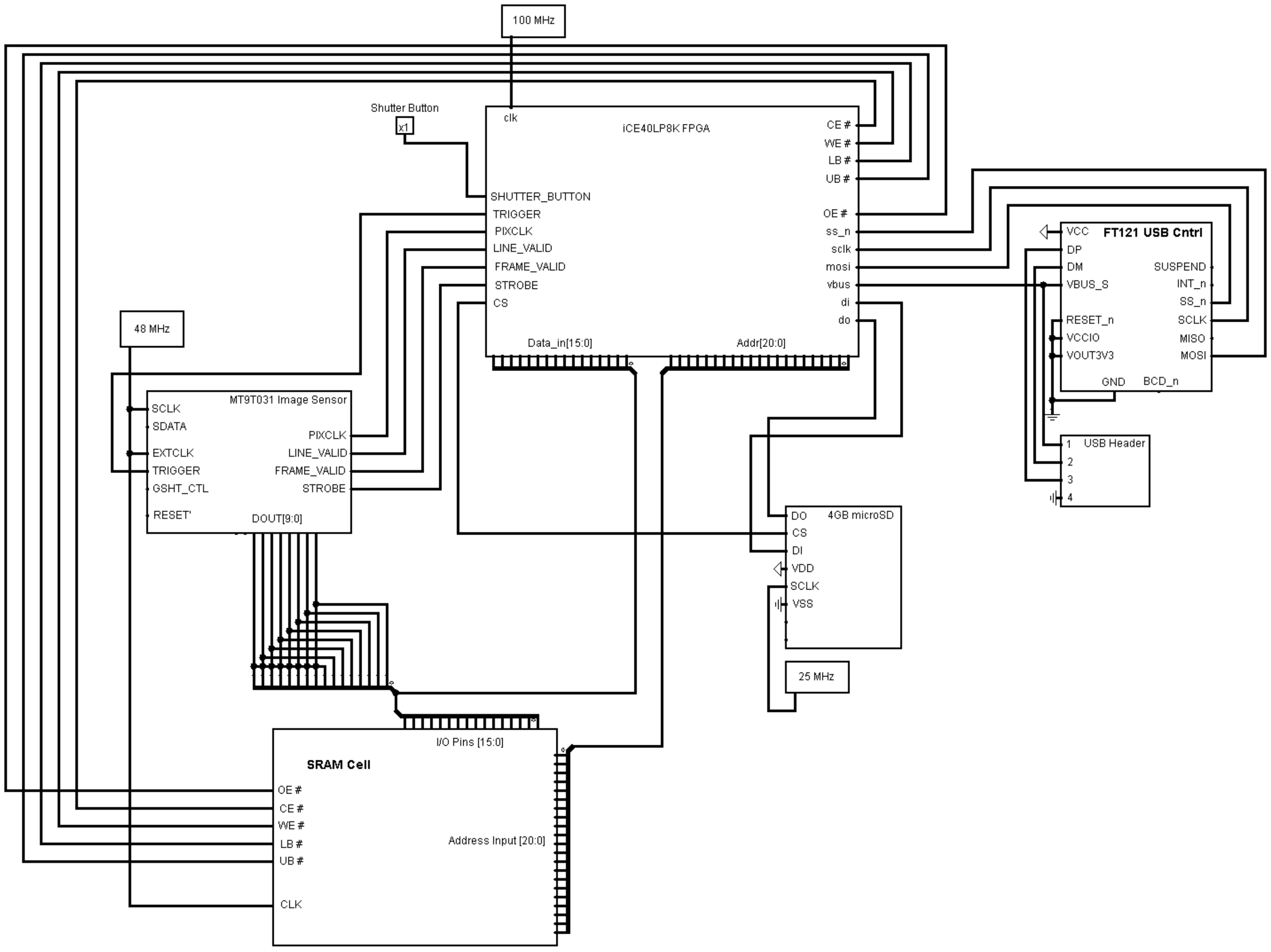
B.5(2 pts) Port and Timing Interfaces: Extend your design (developed in Part A) to implement the port and timing interfaces determined in Questions B.3 and B.4. For the controller, you can stop at the state diagram.


clk
iCE40LP8K FPGA
CE #
WE #
LB #
UB #
Shutter Button
x1
SHUTTER_BUTTON
TRIGGER
PIXCLK
LINE_VALID
FRAME_VALID
STROBE
CS
OE #
ss_n
sclk
mosi
vbus
di
do
Data_in[15:0]
Addr[20:0]
MT9T031 Image Sensor
SCLK
SDATA
EXTCLK
TRIGGER
GSHT_CTL
RESET'
PIXCLK
LINE_VALID
FRAME_VALID
STROBE
DOUT[9:0]
4GB microSD
DO
CS
DI
VDD
SCLK
VSS
I/O Pins [15:0]
SRAM Cell
OE #
CE #
WE #
LB #
UB #
CLK
Address Input [20:0]

```
// control signals and registers
int frame_valid;
int line_valid;
int data_in;
int Addr;
int trigger = 0;
int cs = 1;
int di;
int CE# = 1;
int WE# = 1;
int OE# = 1;
int LB# = 1;
int UB# = 0;
int Temp;
int i;
int j;

// main loop
while (1)
{
   // process a snapshot
   if (shutter_btn)
   {
      while (shutter_btn) {}
      trigger = 1;
      while (!frame_valid) {}

      // store pixel data into SRAM
      while (frame_valid)
      {
         CE# = 0;
         WE# = 0;
         OE# = 1;
         trigger = 0;
         while (!line_valid) {}
         while (line_valid)
         {
            // Cycle writes to LSByte and MSByte of word in SRAM
            LB# = ~LB#;
            UB# = ~UB#;
         }
         Addr = Addr + 16;
      }
      Temp = Addr;
      Addr = 0;
      WE# = 1;
      OE# = 0;
      CS = 0;

      // store contents of SRAM into microSD
      while (Addr <= Temp)
      {
         i = 0;
         while (i < 16)
         {
            di = data_in[0];
            j = 0;
            while (j < 4) {j = j + 1} // stall 4 cycles for microSD
            data_in >> 1; // shift out lsb of data_in
            i = i + 1;
         }
         Addr = Addr + 16;
      }
      CE# = 1;
      OE# = 1;
      CS = 1;
      Addr = 0;
   }
}
```

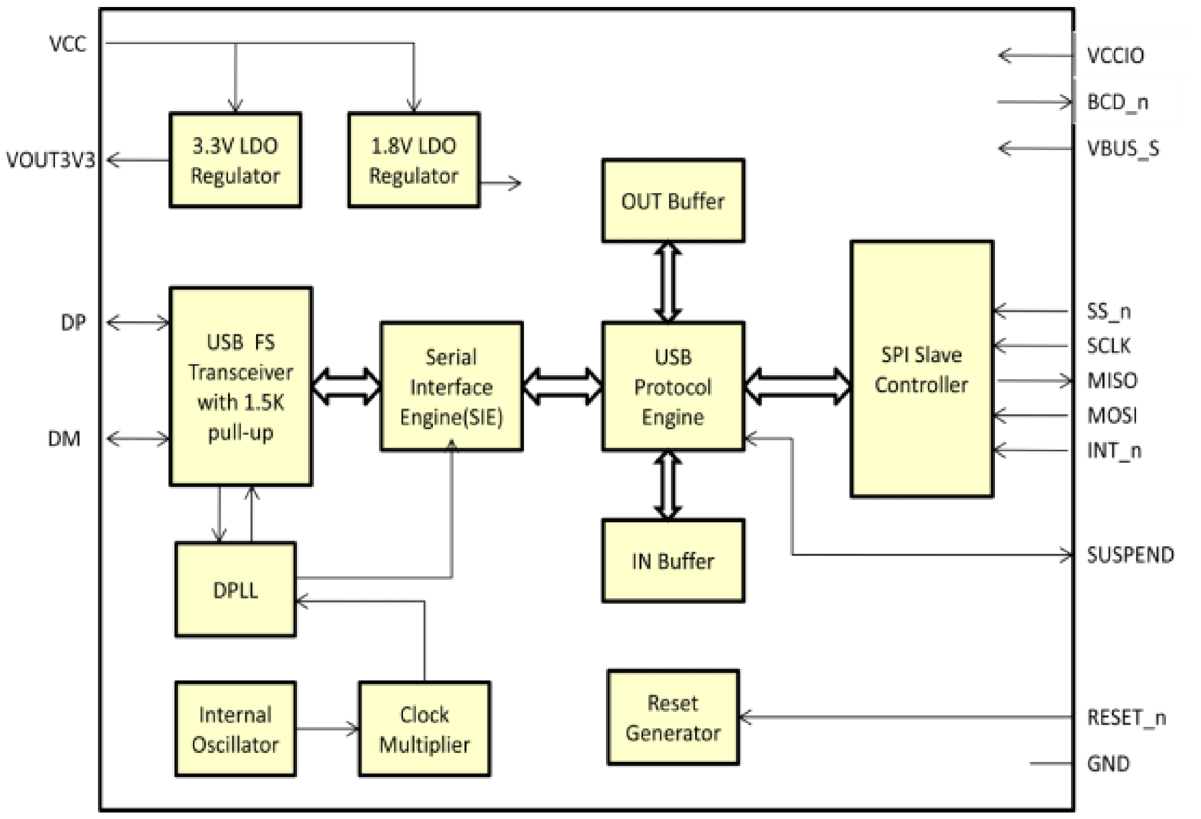


B.6(1 pt.)Detailed Schematic: Extend the detailed schematic of your partial design (developed in Part A) to include the memory. Identify any other components that are required. Show these components as well in the schematic.



B.7(1.5 pts) PC Interface Choose a suitable interface (serial/parallel/wireless) between the camera and PC such as USB, Firewire, Bluetooth, etc. Suggest an off-the-shelf solution to implement this interface. You can “drop in” an existing design provided by the interface vendor. You need not extend the camera controller for this interface. However, you should include the interface cost in your final cost estimation.

We chose to use the FT121 USB controller to serially transmit data to the PC. This is a simple (4-wire SPI interface), widely supported (USB standard), and cheap (Cost: $2.16) solution.



B.8(1.5 pt.) Estimations Estimate: (a) the maximum number of images we can store in the memory; (b) the time required to store/retrieve one image; and (c) the approximate dollar cost to prototype the camera (excluding costs for PCB design and manufacturing, component soldering, and testing).

(A) One Image = 2,048 x 1,536 pixels = 3,145,728 pixels
3,145,728 pixels * 8 bits = 25,165,824 bits
25,165,824 bits / 8 = 3,145,728 bytes
3,145,728 bytes / 10^6 = 3.15 MB

Our microSD card has 4 GB of storage…

3.15 MB = 0.00315 GB
4GB / 0.00315 GB = **1,269 maximum pictures**

(B) Image Capture Time = 48 MHz / 3,997,168 total pixel clocks = **83.27 ms**
Image Store Time = 25,165,824 bits / 16 bits (SRAM word) = ,1,572,864 words
1,572,864 words * 64 clock cycles (shifting 16 bits, 4 stall cycles per bit shift)
= 100,663,296 clock cycles @ 100MHz = **1.006 seconds**

Image Retrieve Time = 25 MHz SPI / 25,165,824 bits per image = **1.006 seconds**

(C)	iCE40LP8K-CM121 FPGA Chip	Cost: $9.39
	MT9T031: 1/2-Inch 3-Mp Digital Image Sensor	Cost: $15.86
	FT121 USB	Cost: $2.16
	25MHz Oscillator	Cost: $0.92
	48MHz Oscillator	Cost: $0.95
	100MHz Oscillator	Cost: $0.97
	32 Mbit SRAM cell	Cost: $5.18
	Total	**Cost: $35.43**

All part prices acquired from www.Mouser.com